NAME

ic -- Input message manager control

SYNOPSIS

```
#include <immcommon.h>

ic_chlchg(chlid, ofcid)
char chlid, ofcid;

ic_ofcinit(ofcid)
char ofcid;

ic_suslvl(chlid)
char chlid;

ic_getparms(chlid, ofcp, mlnp, cinhp, oinhp, sysinhp)
char chlid;
char *ofcp, *mlnp, *cinhp, *oinhp, *sysinhp;

ic_logchg(chlid, mln)
char chlid, mln;

ic_audchl(chlid, ofcid, mln)
char chlid, ofcid, mln;

ic_updaud()

get_imcntl()

ic_pchl(chlid, lvl)
char chlid, lvl;

ic_pofc(ofcid, lvl)
char ofcid, lvl;

ic_psys(lvl)

static ic_setpri(command, id, lvl)
char id, lvl;
```

DESCRIPTION

ic chlchg -- changes the office association of a channel. It should be called each time that a channel ID is retired or activated.

ic ofcinit -- should be called each time an office ID is activated in order to clear the IMM suspension level.

ic suslvl -- gets the effective suspension level of any given channel. It is the maxinun of the applicable suspension levels (channel, office, and system).

<u>ic getparms</u> -- gets the effective suspension level of a channel, as well as all other relevant parameters.

<u>ic logchg</u> -- should be called each time a change in the state af logging is made.

<u>ic audchl</u> -- is used to audit the IMCNTLFILE.

<u>ic updaud</u> -- updates the disc copy of IMCNTLFILE and notifies the Input Message Manager of the change.

<u>get imcntl</u> -- gets the IMCNTLFILE off disc and puts it into core.

<u>ic pchl</u> -- sets the priority of a channel.

<u>ic pofc</u> -- sets the priority of an office.

<u>ic psys</u> -- sets the priority of the system.

<u>ic setpri</u> -- the common priority setting routine.

FILES

/usr/include/immcommon.h

LIBRARY

/lib/lib1.a

NAME

icat -- concatenate arbitrary strings

SYNOPSIS

```
icat(s1,n1,s2[,...],0)
char *s1,*s2;
int n1;
```

DESCRIPTION

icat returns an integer indicating the length of the resulting string s1. The value returned is the same as that returned by the len function. The icat function concatenates the strings s2, s3, s4, etc into the target string s1 which has a maximum size indicated by n1. Icat accepts a variable number of arguments.

s1 buffer area for the target string.

n1 integer which specifies the maximum number of characters which can be stored into s1 including the terminating null character.

s2 source string which is copied into s1.

... secondary source strings are concatenated with s1.

0 a null pointer terminates the argument list.

If the address pointed to by s1 or s2 is zero or if the value of n1 is zero or negative, icat will immediately terminate and return the value -1. If the target string is filled to maximum, icat will return a number one less than n1. If the number of characters requested to be stored in the target string including the terminating null character is larger than n1, icat will return the value -1 but a properly terminated string will remain in s1. This string will return a value from len equal to one less than n1 which indicates that the null character is in the last position of the string. The sizeof function can be used for n1. It should be noted that icat becomes a copy string function when only one source string argument, s2, is supplied.

The strings s1, s2, and etc. are each defined as a null terminated array of characters. The returned integer can also be considered the number of characters preceding the terminating null character.

If s2 and all subsequent arguments point to empty strings, the target string s1 will be set empty and the returned value will be zero. If one of the source strings s2, s3, etc is empty, the remaining strings will be concatenated as if the empty string did not exist.

LIBRARY
 /lib/lib3.a

SEE ALSO
 strcat(3), sprintf(3), pcat(3)

NAME

idchl, idofc -- copy channel/office name and store chldata/ofcids record

SYNOPSIS

```
#include <chldata.h>
idchl(id, ptr)
char *ptr;

#include <ofcid.h>
idofc(id, ptr)
char *ptr;
```

DESCRIPTION

idchl (*idofc*) - copies the channel (office) name to the location pointed to by *ptr*. The chldata (ofcids) record is also placed in the global structure chl (oidbuf).

LIBRARY

/lib/libl.a

DIAGNOSTICS

returns -1 if id is >= MAXCHL (MAXOID) in <*chldata*.*h*> (<*ofcid*.*h*>).

returns -1 if /*sccetc*/*chldata* (/*sccetc*/*ofcids*) cannot be opened.

returns -1 if the channel (office) data file was not read correctly.

returns -1 if the channel (office) name is not filled in. (The first character of the channel/office name is a blank or a dot.)

returns 0 if successful.

FILES

/sccetc/chldata
(/sccetc/ofcids)

NAME

index -- find index of a string in a array of strings

SYNOPSIS

```
index(vect, str)
char *vect[], *str;
```

DESCRIPTION

Compares the string _str_ with each string in the array of strings _vect_ and, if a match is found, the value returned is the index into _vect_.

DIAGNOSTICS

Returns a -1 if no match is found

LIBRARY

/lib/lib1.a

NAME

ismln -- check for valid mln number

SYNOPSIS

```
ismln(mlnno)
int mlnno;
```

?

DESCRIPTION

Checks if _mlnno_ is vlaid and if /dev/ln<mlnno> exists.

DISGNOSTICS

Returns:

```
 1   if valid mln
-1   if mlnno is out of range (see MAXMLN in
     ) or if /dev/ln<mlnno> does not exist.
-2   if the user is not in the same location as the
     mln.
```

LIBRARY

/lib/lib1.a

NAME

iss_list - locate next issue record in issue data file

SYNOPSIS

```
#include <issfil.h>

char *iss_list();
```

DESCRIPTION

Iss list should be used by those routines that need to generate a list of supported issues for a specified generic. Prior to calling *iss list* for the first time, the calling routine must first call *gen list*(*3L*) or *get gen*(*3L*) to extract the desired generic record and associated issue records (generic-issue message) from the appropriate *issue* file. During each call to *iss list*, the starting address of the next issue record is returned to the calling routine. If, however, the next issue record does not exist, the value **ILR_NME** is returned to the calling routine and *iss list* is initialized such that a subsequent call to this subroutine will return the first issue record associated with the current generic-issue message.

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 *get iss*(*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.

FILES

/*usr*/*include*/*issfil*.*h* which specifies structures for an issue record and defines valid return codes for this subroutine.

LIBRARY

/lib/lib1.a

SEE ALSO

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

DIAGNOSTICS

BUGS

NAME

lbit - long bit extraction.

SYNOPSIS

```
lbit(hi,lo,lprt,resutls)

int hi,lo;
long *lptr,*results;
```

DESCRIPTION

Lbit extracts a bit field from a long. The bits in the long are numbered from 0 to **N**-1 going from right to left. **N** is the number of bits defined as a long (see /user/include/constants.h).

Hi is the number of the left-most bit to be extracted.

Lo is the number of the right-most bit to be extracted.

Lptr is the address of the long variable from which the bits are to be extracted.

Results is the address of the long variable where the binary number the bits extracted is to be placed.

LIBRARY

/lib/lib1.a

SEE ALSO

bit(3L) .

DIAGNOSTICS

A -1 is returned for error conditions hi less than lo, lo less than 0 and hi greater than **N**-1. Otherwise, a zero is returned.

NAME

len -- length of string

SYNOPSIS

len(s1)
char *s1;

DESCRIPTION

len returns an integer indicating the length of the string _s1_.

_s1_ string to be evaluated

The string _s1_ is defined as a null terminated array of characters. The value of the integer that is returned is the array index of the terminating null character.

This returned integer can also be considered the number of characters preceding the terminating null character.

An empty string is one whose first character is the null character. If _s1_ is empty the integer that is returned is zero.

LIBRARY

/lib/lib3.a

SEE ALSO

strlen(3), size in /usr/include/sccmacros.h, plen(3)

NAME

logmsg -- write a message into a specified logging file

SYNOPSIS

```
logmsg(fsp, mesg, fd)
struct fs *fsp;
char *mesg;
int fd;
```

DESCRIPTION

Logmsg writes a message into a specified logging file

ARGUMENTS:

struct _fs_ *_fsp_;	ptr to fs entry in core
char *_mesg_;	mesg to be written
int _fd_;	logdev file desc

RESTRICTIONS

This routine does not exist beyond SC5.

LIBRARY

/lib/lib1.a

NAME

lopen - open unix or linked logging file

SYNOPSIS

```
#include <lopen.h>

struct CHLCTL locc;
struct FS_CB lofs;
lopen(name, mode, &data)
char *name;
struct {
     char *l_block;
     char  l_name[LONAMESIZ];
} data;
```

DESCRIPTION

Lopen opens the given file for whichever mode specified (see *open*(2)). *Name* is the address of a string of ASCII characters, terminated by a null character, which defines a unix or logging file according to the conventions which follow.

ofc.chl
: Where "chl" is a valid channel name (see *chlnams*(*3L*)) and "ofc" is a valid office on the SCCS. The corresponding logging file is opened.

ofc
: Where "ofc" is the name of a valid office on the SCCS. The logging file "ofc.mtc" is opened.

chl
: Where "chl" is a valid channel name. The specified channel on the current default office is opened. I.e. the logging file defined by the combination ".office/chl" is opened.

unix
: The unix file in the current directory is opened.

usr.unix
: The unix file in the specified user directory is opened.

ofc.unix
: The unix file in the specified office is opened.

Lopen uses the following ordering when attempting to find the given file:

```
if(name contains a dot) {
     if((2nd half is a chl name) && (channel exists))
          return(logging);
     if(name is in current directory)
          return(unix);
     if(name is in user directory)
```

```
                    return(unix);
               if(name is in office directory)
                    return(unix);
               return(bad);
          }


          if(name is a chl name) {
               if(<default office>/name exists)
                    return(logging);
          } else
               if(name.mtc exists)
                    return(logging);
          if(name is in current directory)
               return(unix);
          return(bad);
```

When the file which is opened is a unix file, *lopen* clears the element l_block in the *data* structure, and copies *name* into the l_name element. When the file which is opened is a logging file, however, a lseek(2) is done to the disk block containing the latest messages, and that block number is written into the l_block element. The CHLCTL and FS_CB structures read in the process are available to the caller as globals *locc* and *lofs*, respectively. In all but the "ofc" and "chl" cases above, *name* is copied verbatim into the l_name element. In the exceptional cases, the l_name element contains a string of the form "ofc.chl" to reflect the full name of the logging file which was opened. The length of the l_name element is defined in the header file /usr/include/lopen.h.

In addition, in the "ofc" and "ofc.chl" cases, *lopen* updates the ".office" link to point to the office specified in *name*.

LIBRARY
/lib/lib1.a

FILES
/usr/include/lopen.h
.dfltparm

SEE ALSO
updofc(3), chl(5), fs(5), chldata(5), getdfprm(3L)

DIAGNOSTICS
A negative returned value indicates an error. The external elements E_SPCL, E_TYPE, E_CODE, E_NUM, and E_MSG contain the appropriate strings to be passed to *sccerr*(*3L*).

NAME

m_samptim, m_tcalc, m_time -- time measurement routines

SYNOPSIS

m_samptim(interval)

m_tcalc(timlowword)

m_time()

DESCRIPTION

m_samptim -- determine if it is time for a measurement sample. Returns true if, roughly, the interval time has passed since the last call.

m_tcalc -- calculate how much time has passed since a time with a given low word; this may seem strange, but it is used to calculate the delay time for alarms. The alarm distributer has the low word of the logging time of a mesage and it must be known how long it has been since the message has been logged.

Portability Note: By necessity, this routine must assume that time arrives as a long, the second integer of which is the low word. On 32 bit machines, the input from the alarm distributer will probably be different and the whole routine wil have to be reworked.

m_time -- compute time since last call as an unsigned integer used in the measurement macros.

Portability Note: The conversion from long to unsigned may vary from machine to machine.

LIBRARY

/lib/lib1.a

SEE ALSO

time(2)

NAME

lpropen - open pipe to the line printer

SYNOPSIS

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

FILE *lpropen(lprstr,loc,ofp)
char *lprstr;
short loc;
FILE *ofp;
```

DESCRIPTION

Lpropen returns a file pointer used to write to the line printer queue specified by *lprstr*. If *lprstr* is 0, a default queue is used. *Ofp* is the file pointer used to report any errors encountered as well as the spooling system JOB ID. If *ofp* is 0, all messages are discarded. *Loc* is the location to which the line printer queue has been assigned. A value of ANY_GID (aparam.h) or less specifies queues in location ANY. If the queue *lprstr* is not assigned to location *loc*, *lpropen* attempts to access the queue *lprstr* in location ANY, and failing that will return an error.

Since *lpropen* spins off the lpr program, to insure that the output from the lpr program is not intermixed with other output, it is recommended that the file pointer returned be explicitly closed, and that a wait be executed when all output for the printer has been generated.

SEE ALSO

lpr(1)

DIAGNOSTICS

0 is returned if the specified queue does not exist, a pipe cannot be created, or a fork cannot be executed.

BUGS

Though *lpropen* only returns one file pointer, two descriptors are required for a short time in order ro set up a pipe to the spooling system.

NAME

match - pattern matcher

SYNOPSIS

```
          char    *ppcursor;
          char    *ppdot;

     int match(patptr,progarg0,progarg1,...)  /* old interface */
          PPAT *patptr;
          PPROGARG progarg0;
          PPROGARG progarg1;
            .
            .
            .
```

DESCRIPTION

Ppmatch and **match** provide two ways to call the common pattern package **pattern matcher.** In general a pattern matcher takes a pattern and one (or more) strings and determines if the pattern matches the string(s). The common pattern package pattern matcher preforms this function and several other functions to include:

1) Pattern matching on one or more strings given in the progarg array as determined by the **switch** built-in pattern.

2) Return an integer value as specified by the **succ** built-in pattern.

3) Mark one or more positions in any of the strings provided by the **dot** and **mdot** built-in patterns.

4) Provide the addresses of one or more pieces of the string or pattern in a user supplied buffer (specified by ppmdot(3L)) as a first step in reformating one or more strings using the **startfld, endfld** and **deffld** built-in patterns.

The arguments to **match**() are as follows:

patptr is a pointer to the pattern to be used by the matcher.

progarg0,progarg1,... are application defined inputs. The first element (**patarg0**) must point to the start of the first text-area. All other elements may point to any valid program argument type as defined in the **<ppsubs.h>** header file.

Ppmatch and match never change anything pointed to by their arguments.

Ppmatch and match sets the value of several external variables as described below.

ppcursor - contains the value of the matcher cursor (pointer to first text-area) at the time the matcher returned. In the old version of the pattern matcher **cursor** was used instead of **ppcursor** For upward compatibility purposes **cursor** is equivalent to **ppcursor**

ppdot - is set to the current cursor position when a **dot** built-in pattern is encountered in the pattern. If no **dot** built-in pattern is encountered, Then the value of ppdot is not changed. In the old version of the pattern matcher **dot** was used instead of **ppdot** For upward compatibility purposes **dot** is equivalent to **ppdot**

The first element (zero subscript) of the **patarg** array (and parg0 in match()) should be a text-area. This element is used to initialize the matcher cursor (pointer to the text-area being pattern matched). A **switch** keyword in the pattern may change the text-area being pattern matched (as well as the pattern). Therefore, the use of a **switch** keyword in the pattern may require additional text-areas which must have pointers (to them) included in the array. The index of the pointer in the array corresponds to the number argument in the **switch** keyword. For example the keyword **switch(2,arb 'aaa')** requires progarg[2] to be a pointer to a text-area.

Ppmatch and match returns one of the integer values described below:

PPSUCCESS - indicates a successful match

PPABORT - indicates an unsuccessful match

PPUNDEFKEY - indicates a zero value primitive was found in the pattern. This indicates that the pattern has been scribbled (or is not a pattern).

n - where **n >= 0;** and **n** is the value of a **succ** built-in pattern argunment which is encountered by ppmatch and match

SEE ALSO

ppchkpat(3L), ppmatch(3L), ppsmdot(3L)

DIAGNOSTICS

Ppmatch and match produces no diagnostics except that a **PPUNDEF-KEY** value will be returned when a zero value primitive is discovered in the pattern (zero is an invalid primitive value).

BUGS

Ppmatch and match do not check the pattern or the elements of **progarg** If any of their values are improper, then unpredictable/terrible things may occur (e.g., trying to execute instructions in data or stack space). To avoid some of the posible problems **ppchkpat(3L)** should be used.

NAME

meas - measurement interface routines

SYNOPSIS

meas(offset, argcnt, arg1, ...)
m_closemeas()
m_clearmeas()
m_samptim(interval)
m_tcalc(timelow)
m_time()

DESCRIPTION

LIBRARY

/lib/lib1.a

NAME

mdfopen,e_mdfopen - open a file with specific mode

SYNOPSIS

```
#include <stdio.h>

FILE *mdfopen (filename, type, mode)
char *filename, *type;
int mode;

FILE *e_mdfopen (filename, type, mode, errcode)
char *filename, *type;
int mode, errcode;
```

DESCRIPTION

Mdfopen will open *filename* like *fopen*(3S) and if *file* is created, the mode of the created file will be *mode*, where *mode* is a three digit octal number.

For example:

```
    mdfopen ("myfile", "w+", Ø644);
```

will create the file, **myfile**, for update and the mode of the file will be **Ø644**.

If *filename* already exists, it´s current mode will be unchanged.

E_mdfopen is the same except it interfaces to the e_routines. See *e_stdio*(3L).

SEE ALSO

fopen(3S), e_stdio(3L)

LIBRARY

/lib/lib1.a

BUGS

You can do some dumb things with *mdfopen*. For example:

```
    mdfopen ("myfile", "w", Ø444);
```

will create the file **myfile** for writing but the mode **Ø444** will not permit writing if it is closed and reopened.

DIAGNOSTICS

Mdfopen and *e_mdfopen* return the pointer **NULL** if *filename* cannot be accessed.

NAME

mesg - print string on standard error output.

SYNOPSIS

```
mesg(s)
char *s;
```

DESCRIPTION

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Same as _write_(_2_) including errno.

NAME

mkdir - make directory

SYNOPSIS

```
mkdir(file,owner,mode,group)
char *file;
int owner, mode, group;
```

DESCRIPTION

Mkdir will make a directory, link the necessary `.' and `..' pointers and set the specified mode, owner, and group based on the following arguments:

file	A pointer to a string representing a full or partial pathname of a directory to be made.
owner	An integer representing the owner of the made directory.
mode	An integer representing the mode of the directory. The mode represents a value acceptable to a chmod system call.
group	An integer representing the group id of the made directory.

Mkdir returns:

0	directory successfully made.
-1	file already exists.
-2	Cannot do a mknod, link, chown, chmod or chgrp or the effective uid is not super user.

The subroutines *chgrp*(2), *chmod*(2), *chown*(2), *getuid*(2), *link*(2), *mknod*(2) and *stat*(2) are used by *mkdir*.

If mkdir returns with a **-2**, then any work it has done is still there, e.g. if it cannot do a chown, the directory that the mknod and linking has created prior to the chown still exists.

SEE ALSO

rmdir(3C)

DIAGNOSTICS

A return code of **-2** is serious because it means that *mknod* has done some but not all of its work.

BUGS

Mkdir should not require the effective user id to be super user.

If the requested action cannot be performed, *mkdir* should undo whatever has been done.

NAME

modlns -- modify lines file

SYNOPSIS

```
modlns(entry, addflg)
char *entry;
int addflg;
```

DESCRIPTION

Modlns adds or deletes entries to the /etc/lines file, depending on whether addflg is true or false, respectively. The entry (pointed to by entry) is added before the "80" entry in the lines file. The entry must include the "newline". For "delete", only the first two characters of the entry string matter. After modification, init(1) is signalled to rescan the lines file.

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Error returns:

-1 An entry with the same ID exists in the lines file, so the entry is not added.

-2 Problems opening, reading or writing lines file or temporary file. If the user is not root and the lines file is readable and writable by everyone, then the lines file is modified and a -2 is returned because a kill could not be sent to init(1). lines file or temporary file.

-4 The temporary file /etc/ltmp already exists, so try later.

NAME

modmsg -- modify message

SYNOPSIS

```
int LEN_MSG;
char modmbuf[82];

char *modmsg(fmt, arg1, arg2, ...)
char *fmt;
```

DESCRIPTION

This subroutine allows users to utilize all the features of _printf_ without printing out the results. The _modmsg_ subroutine returns the address of a string containing the modified message and is called identically to _printf_. Two global variables are utilized by _modmsg_. LEN_MSG contains the length of the string generated, while _modmbuf_ is a character array initially of size 82, used to assemble the resulting string. _Modmbuf_ may be redeclared as a global character array by the user; in this case it will have as its size the maximum of 82 and the size specified by the user.

RESTRICTIONS

This implementation of _modmsg_ is a modified version of _sprintf_(3) from the C library. It will remain in existance only until all code using _modmsg_ is converted to use _sprintf_.

LIBRARY

/lib/lib1.a

SEE ALSO

sprintf(3)

NAME

modusr - modify /*etc*/*passwd* file for specified user

SYNOPSIS

```
modusr(user,gid,wdir,shell,pwd)
char *user, *wdir, *shell, *pwd;
int gid;
```

DESCRIPTION
DIAGNOSTICS

Modusr modifies the entry in the /*etc*/*passwd* file for the specified user (*user*). The remaining arguments are:

gid	An integer specifying the new group id. If -1, the group id is not changed.
wdir	A pointer to a character string containing the pathname of the new working directory. If null, it is not changed.
shell	A pointer to a character string containing the pathname of the new shell. If null, the shell is not changed.
pwd	A pointer to a character string containing the new encrypted password. If null, it is not changed.

Modusr will return the following values:

```
 0 - if it was successful.
-1 - if it could not find the user in the /etc/passwd file
-2 - system problems: could not open or link /etc/passwd
     or /etc/ptmp files, or perform other
     system calls (stat,fstat,etc.).
     close(2) is the only sys call not checked.
-3 - given user has a colon in name
-4 - /etc/ptmp already exists (try again).
```

LIBRARY

/lib/lib1.a

FILES

/*etc*/*passwd*
/*etc*/*ptmp*

NAME

notany -- match character against character not in string

SYNOPSIS

```
notany(c1,s1)
char c1, *s1;
```

DESCRIPTION

notany returns an integer indicating the success or failure of the pattern match. If the value returned is zero the match was a success. If the value returned is -1 the match was a failure. This function indicates if the character c1 matches none of the characters in the string s1.

c1 a character to be searched.

s1 a string of characters used as a pattern.

The pattern, s1, can be any null terminated string of characters. Repeated characters in s1 are ignored. The pattern string "Mississippi" is equivalent to the pattern string "iMps".

If c1 is null, the function always returns a -1.

If s1 is empty, the function returns a zero unless c1 is null.

If the character c1 is found in s1, the function returns a -1, otherwise, the function returns a zero.

LIBRARY

/lib/lib3.a

SEE ALSO

strchr(3), pos(3)

NAME

nwk_tbl -- get requested information from "nwktbl" file

SYNOPSIS

```
#include <gtlhdr.h>
#include <nwktbl.h>

nwk_tbl(nwkfunc, nwkfd, nwkkey, maxnwk, nwkcr, nwksw)
int nwkfunc, nwkfd, maxnwk, nwkcr[], nwksw[];
char *nwkkey;
```

DESCRIPTION

Extract all requested information from the "nwktbl" file. A request may be submitted for the "switch type" or "network conc. ratio" associated with a specified network or all networks of a specified type. The concentration ratios or switch types for all networks of a specified type are stored in the external arrays, "nwkcr" and "nwksw", respectively, and are indexed by the network number. These arrays are initialized to -1 for all networks. If a concentration ratio or switch type has been requested for a specified network, then the appropriate value is returned by this subroutine.

Arguments:

nwkfunc	identifies the type of function that is to be performed by this subroutine. The permitted values are:
	NWF_SW Extract switch type for specified network.
	NWF_CR Extract conc. ratio for specified network.
	NWF_ALL Extract requested info. for all equipped networks and store in the arrays provided.
nwkfd	is the file descriptor of the "nwktbl" file that has been opened by the calling program.
nwkkey	identifies the network type or a specific network when only one conc. ratio or switch type is to be extracted.
maxnwk	specifies the maximum number of networks of type "nwkkey" that can be provided.
nwkcr	if nonzero, specifies the address of an integer array in which the network conc.

	ratios for all equipped networks of the specified type are to be stored.
nwksw	if nonzero, specifies the address of an integer array in which the switch types for all equipped networks of the specified type are to be stored.

The values returned by _nwk_tbl_() are:

NWR_FUNC	Invalid function requested.
NWR_NN	Specified network does not exist.
NWR_READ	Error detected by _gtline_(3L) while trying to read "nwktbl" file.
NWR_NORM	Normal termination when switch types and/or conc. ratios have been extracted for all equipped networks.
> NWR_NORM	Contains the switch type (values from 1 to 3) or the conc. ratio for the specified network.

FILES
/usr/include/gtlhdr.h /usr/include/nwktbl.h

LIBRARY
/lib/lib1.a

SEE ALSO
gtline(3L)

NAME

ofcclli - translate office name to CLLI code.

SYNOPSIS

```
#include "ofcclli.h"

ofcid = ofcclli(cllip, ofcp, flag)
     char *cllip;          /* CLLI name string */
     char *ofcp;           /* office name string */
     int flag;             /* keep file open flag */
     int ofcid;            /* office id */
```

DESCRIPTION

Ofcclli translates an office name into its corresponding standard Common Language Location Identification (CLLI). The office must be a valid office on the system. It assumes the office name is pointed to by *ofcp* and is null-terminated. *Ofcclli* looks up the office in the file /sccetc/ofcclli and copies the corresponding CLLI to the string pointed to by *cllip*, then null-terminates it. The CLLI string must be at least CLLINAMSIZ+1 characters long. In addition, the subroutine returns the office's *office id* (values 0 to MAXOID-1) as the returned value (but see DIAGNOSTICS below).

The *flag* argument is used to tell the subroutine whether to leave the file descriptor open for the /sccetc/ofcclli file before returning. *Flag* = 0 means close the file descriptor; *flag* = 1 means leave it open. Programs which expect to call *ofcclli* a lot may want to leave the file descriptor open, so that the subroutine does not have to re-open the file each time; other programs which use the subroutine only once or who are more concerned about file descriptor usage may want to close the file descriptor.

The /sccetc/ofcclli file is maintained by the commands *RC*:*CLLI* and *VRFY*:*CLLI*.

FILES

/sccetc/ofcclli

LIBRARY

/lib/lib1.a

SEE ALSO

dltclli(3L), ofcclli(3L)

DIAGNOSTICS

Ofcid will have the following values in error cases:

-1 if system error occurred (open or read failure);
standard SCCS errors are printed in this case.

-2 if subroutine could not find the office name
in the /sccetc/ofcclli file.

NAME

ofcdata -- get office data

SYNOPSIS

```
#include <chldata.h>

struct CHL_B *ofcdata(ofcnam)
char *ofcnam;
```

DESCRIPTION

Reads th office data from the oparm file for ofcnam into a static area and returns a pointer to it.

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Return NULL if the oparm file cannot be opened or read.

NAME

ofcid -- get office ID

SYNOPSIS

ofcid(ofcnam)
char *ofcnam;

DESCRIPTION

Returns the integer office ID for the office _ofcnam_. Uses ofcdata(3) to access the oparm file.

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Returns -1 if cannot open oparm file.

NAME

ofcotyp -- get office type

SYNOPSIS

```
ofcotyp(ofcnam)
char *ofcnam;
```

DESCRIPTION

Returns the integer office type for the office _ofcnam_. This can be used to index into the _ofctyps_(3) array to get the ASCII string describing the office type. Uses _ofcdata_(_3_) to access the oparm file.

LIBRARY

/lib/lib1.a

SEE ALSO

ofctyps(3)

DIAGNOSTICS

Returns -1 if cannot open the oparm file.

NAME

ofctyps -- office type array

SYNOPSIS

#include <office.h>

DESCRIPTION

List of standard office types referenced by common header file _office.h_

```
          char *ofctyps[] {
               "sccs",        /* type 0 */
               "ess1",        /* type 1 */
               "ess2",        /* type 2 */
               "ess3",        /* type 3 */
               "ess2b",       /* type 4 */
               "tsps",        /* type 5 */
               "ess101",      /* type 6 */
               "ess1a",       /* type 7 */
               "int1a",       /* type 8 */
               "ess4",        /* type 9 */
               "tn",          /* type 10 */
               "epscs",       /* type 11 */
               "vss",         /* type 12 */
               "ess5",        /* TYPE 13 */
               "ais",         /* type 14 */
               "e911",        /* type 15 */
               "tsps1b",      /* type 16 */
               "dacs",        /* type 17 */
               "ncp",         /* type 18 */
               "toc4e",       /* type 19 */
               0
          };
```

LIBRARY

/lib/lib1.a

SEE ALSO

ofcotyp(3)

NAME

pbreaks -- look for first char in pattern

SYNOPSIS

```
pbreaks(s1,s2)
char *s1, *s2;
```

DESCRIPTION

pbreaks returns a pointer indicating the success or failure of the pattern match. If the address returned is not zero the match was a success. If the address returned is zero the match was a failure. This function indicates success if a character in the pattern string is found in the searched string.

s1 the searched character string.

s2 a string of characters used as a pattern.

The pattern, s2, can be any null terminated string of characters. Repeated characters in s2 are ignored. The pattern string "Mississippi" is equivalent to the pattern string "iMps".

This function is implemented with a table driven pattern matcher. The empty string is defined as a string whose first character is the null character.

If either or both of the strings is empty the address returned will be zero.

If a character of the string s2 is found in the string s1, the address of that character found in s1 will be returned.

If the entire string s1 is searched and no character in s2 is found in s1, the address returned will be zero.

LIBRARY

/lib/lib3.a

NAME

pcat -- concatenate arbitrary strings

SYNOPSIS

```
pcat(s1,n1,s2[,...],0)
char *s1,*s2;
int n1;
```

DESCRIPTION

Pcat returns a pointer indicating the address of the terminating _null_ character for the target string _s1_. The address returned is the same as that returned by the _plen_ function. The _pcat_ function concatenates the strings _s2_, _s3_, _s4_, etc into the target string _s1_ which has a maximum size indicated by _n1_. _Pcat_ accepts a variable number of arguments.

_s1_ buffer area for the target string.

_n1_ integer which specifies the maximum number of characters which can be stored into _s1_ including the terminating null character.

_s2_ source string which is copied into _s1_

... secondary source strings are concatenated with _s1_.

_0_ a null pointer terminates the argument list.

If the address pointed to by _s1_ or _s2_ is zero or if the value of _n1_ is zero or negative, _pcat_ will immediately terminate and return the address zero. If the target string is filled to maximum, _pcat_ will return the address of the last position in _s1_. If the number of characters requested to be stored in the target string including the terminating null character is larger than _n1_, _pcat_ will return the address zero but a properly terminated string will remain in _s1_. In this case, the function _plen_ will return the address of the last character in _s1_. The _sizeof_ function can be used for _n1_. It should be noted that _pcat_ becomes a copy string function when only one source string argument, _s2_, is supplied.

The strings _s1_, _s2_, and etc. are each defined as a null terminated array of characters. The returned pointer is the address of the terminating null character.

If _s2_ and all subsequent arguments point to empty strings, the target string _s1_ will be set empty and the returned address will be the address of the first character in _s1_. If one of the source strings _s2_, s3, etc is empty, the remaining strings will be concatenated as if the empty string did not exist.

LIBRARY
    /lib/lib3.a

SEE ALSO
    icat(3)

NAME

pdiff -- locate first string difference

SYNOPSIS

pdiff(s1,s2)
char *s1, *s2;

DESCRIPTION

Pdiff returns an address indicating the position within the first string, _s1_, that the first character mismatch was discovered. If the two strings are identical, the address returned is zero.

_s1_ primary string to be used in comparison.

_s2_ secondary string to be used in comparison.

An empty string is one whose first character is the null character. If both strings are empty, the address returned has the value zero.

The two strings are compared character by character until one or both are terminated by the null character or until a mismatch is found. If the two strings are identical until one string is terminated by the null character while the other string is not terminated, the returned address is the value of the pointer in string _s1_ when the comparison terminated.

If the two strings differ on a character other than the null character the address of the character position in string _s1_ that differs from the respective character in string _s2_ is returned.

LIBRARY

/lib/lib3.a

NAME

pindex -- find position of substring within a string

SYNOPSIS

```
pindex(s1,s2)
char *s1, *s2;
```

DESCRIPTION

Pindex returns a pointer indicating the starting position within the string _s1_ of a substring identical to string _s2_.

_s1_ string to be searched.

_s2_ string to be searched for.

If _s2_ does not occur in _s1_, the pointer returned is zero.

If _s2_ occurs more than once in _s1_, the starting address of the first occurrence is returned.

The strings _s1_ and _s2_ are each defined as a null terminated array of characters. The value of the pointer that is returned is the address of the beginning of the substring in _s1_. The returned pointer can have values from one to 65535.

An empty string is one whose first character is the null character. If one and only one of the two argument strings is empty, the result returned is zero. If both argument strings are empty, the result returned is the address of the null character in string _s1_.

LIBRARY

/lib/lib3.a

SEE ALSO

sindex(3L)

NAME

plen -- length of string - last address

SYNOPSIS

plen(s1)
char *s1;

DESCRIPTION

Plen returns a pointer to the terminating null character in string s1.

s1 string to be evaluated

The string s1 is defined as a null terminated array of characters. The address that is returned is the address the the terminating null character.

The returned address minus the starting address of the string s1 is the length of the string as defined by the function len.

An empty string is one whose first character is the null character. If s1 points to an empty string, the address that is returned is the address of the null character which is the value of s1.

LIBRARY

/lib/lib3.a

SEE ALSO

len(3)

NAME

pos -- position of char in string

SYNOPSIS

```
pos(s1,c1,n1)
char *s1, c1;
int n1;
```

DESCRIPTION

Pos returns an integer indicating the position within the string s1 of the character c1 after n1 occurrences of the character c1.

s1 string to be searched.

c1 character to be searched for.

n1 integer number of occurrences of c1 before final search.

If c1 does not occur in s1 after the n1 preliminary occurrences of c1, the value returned is -1.

If c1 occurs in s1 many more times than n1, the value returned is that of the first occurrence of c1 after the n1 preliminary occurrences of c1.

The string s1 is defined as a null terminated array of characters. The value of the integer that is returned is the array index of the character c1 in s1. The returned integer can have values from zero to 65535.

An empty string is one whose first character is the null character. If s1 is empty or c1 is the null character, the value -1 is returned.

The integer n1 can be any positive value from zero to 32767. If n1 is zero, the value returned is that of the first occurrence of the character c1 in the string s1.

LIBRARY

/lib/lib3.a

SEE ALSO

strchr(3C), notany(3L)

NAME

ppatell - return start of pattern part tell value

SYNOPSIS

```
#include <ppsubs.h>

        int pperrno;    /* error type external */

long ppatell(headptr)
        struct PPHEAD *headptr;
```

DESCRIPTION

Ppatell() returns the tell value for the start of the pattern part of the pattern file with header pointed to by **headptr**. The tell value can be used by **lseek(2)** or **fseek(3)**.

SEE ALSO

lseek(2), fseek(3), pattern(5L)

DIAGNOSTICS

Ppatell() returns a (**0L**) value when an error occurs. The subroutine will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs.

PPBADPAT - The internal format of the pattern was not correct. This could occur if the pattern was not made by **ppmkpat(1L)** or if the pattern had been scribbled.

NAME

ppatsiz - return size of pattern

SYNOPSIS

```
#include <ppsubs.h>

        int pperrno;    /* error type external */

unsigned ppatsiz(headptr)
        struct PPHEAD *headptr;  /* pointer to pattern header */
```

DESCRIPTION

Ppatsiz() returns the size (in bytes) of the pattern part of the pattern file with header pointed to by **headptr**.

SEE ALSO

pattern(5L)

DIAGNOSTICS

Ppatsiz() returns a NULL when an error occurs and sets the value of the external **pperrno** to one of the following values:

PPBADPAT - The internal format of the pattern was not correct. This could occur if the pattern was not made by **ppmkpat**(**1L**) or if the pattern had been scribbled.

NAME

ppchkpat - pattern checker subroutine

SYNOPSIS

```
#include <ppsubs.h>

        int pperrno;    /* error type external */

ppchkpat(headptr,viptr,patptr,progarg)
        struct PPHEAD *headptr;
        int *viptr;
        PPAT *patptr;
        PPROGARG *progarg;
```

DISCRIPTION

This subroutine checks for internal errors (scribbles) in the different parts of a pattern file.

If **headptr** equals (**sturct PPHEAD ***) **NULL**, then the header is not checked. Otherwise, **headptr** is assumed to be a pattern file header address which will be checked.

If **viptr** equals (**int ***) **NULL**, then the variable argument information is not checked. Otherwise, **viptr** is assumed to be a pointer to the pattern variable argument information which will be checked.

If **patptr** equals (**PPAT ***) **NULL**, then the pattern is not checked. Otherwise, **patptr** is assumed to be a pointer to a pattern which will be checked.

If **progarg** equals (**PPROGARG ***) **NULL**, the the application program arguments are not checked. Otherwise, **progarg** is assumed to be a pointer to the application program arguments which will be checked.

If nothing is wrong with any of the things which were checked, then **ppchkpat**() returns a **PPOK** value which is defined in **<ppsubs.h>**.

TO BE SPECIFIED BETTER LATER.

SEE ALSO

ppmatch(3L), pattern(5L)

DIAGNOSTICS

Ppchkpat() returns a **NULL** when an error occurs. **Ppchkpat**() will set the value of pperrno to one of the following values (defined in **<ppsubs.h>**) when a problem occurs:

PPBADPAT - The pattern header has erroneous information in it (i.e., the pattern header is not a pattern header or the pattern file has been scribbled or altered).

NAME

ppdefdso - defining the pattern directory search order

SYNOPSIS

#include <ppsubs.h>

DESCRIPTION

The searching order for pattern directories may be defined by an application program. The search order is used by the **PPGETPAT(3L)** and **PPOPENPAT(3L)** subroutines of the common patterns package.

Patterns are read from disk directory files. The directories which are searched for the patterns and the order in which the directories are searched can be specified by an application program. To specify the directory search order simply create an array of pointers to strings. Each string represents a directory name (e.g., **/usr/clr/pat**). The first element (zero subscript) in the array points to the name of the first directory to be searched, and the second element points to the second, and so on. The last element in the array has a value equal to **PPLASTDIR** (which is defined in the **ppsubs.h** header file). Several directories are available as part of the common pattern package. These directories are defined in the **ppsubs.h** header file. The following example should clear up any questions.

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

static PPATDIR dirso[] = { /* directory search order array */
        PPKEYDIR,       /* Pattern keyword directory, "/keypat" */
        PPUSRDIR,       /* present working directory, "." */
        "/type01/pat",
        PPCOMPATDIR,    /* Common pattern directory, "/compat" */
        PPUSRPATDIR,    /* User Common pat directory, "/usr/pat" */
        PPLASTDIR       /* End of Directory Search Array */
};

main()
{
        extern PPATDIR dirso[];

        FILE *patstream;        /* pointer to pattern stream */
        int patfdes;            /* pattern file descriptor */

        patstream = ppfopenpat("name",PPSTDFRMT,dirso);

                /* or */

        patfdes = ppopenpat("name",PPSTDFRMT,dirso);
    .
    .
    .
```

In the above part of a program, **ppopenpat(3L)** and **ppfopenpat()** would attempt to open the following files in the following order:

```
/keypat/name.p
name.p
/type01/pat/name.p
/compat/name.p
/usr/pat/name.p
```

SEE ALSO

ppgetpat(3L), ppopenpat(3L), ppsccsgp(3L), ppdftdso(3L)

NAME

ppdftdso - external default pattern directory search order

SYNOPSIS

```
#include <ppsubs.h>      /* pattern definitions and struct */


PPATDIR ppdftdso[] = {         /* Default Directory Search order array */
               PPKEYDIR,
               PPUSRDIR,
               PPCOMPATDIR,
               PPUSRPATDIR,
               PPLASTDIR
                };
```

DESCRIPTION

This is the Pattern Package default directory search order. It is used by several of the pattern library subroutines when a user defined directory search order is not given.

SEE ALSO

ppdefdso(3L), ppopenpat(3L), ppgetpat(3L), ppsccsgp(3L)

NAME

ppemsg - external pattern error message stings

SYNOPSIS

```
#include <ppsubs.h>      /* pattern definitions and struct */


char *ppemsg[] = {  /* pattern package error message array */

     "NO ERROR",
     "SYNTAX ERROR IN PATTERN DEFINITION",
     "PATTERN FILE INTERNAL FORMAT IS WRONG (FILE IS SCRIBBLED)",
     "CANNOT FIND PATTERN FILE",
     "PATTERN FILE NAME IS TOO LONG",
     "PATTERN DIRECTORY SEACH PATH NAME IS TOO LONG",
     "PATTERN SOFTWARE CANNOT PERFORM SYSTEM FUNCTION",
     "THE BUFFER GIVEN TO THE PATTERN SOFTWARE IS TOO SMALL",
     "PATTERN HAS NO VARIABLES",
     "UNKNOWN ERROR",
     "PATTERN FILES OF THIS TYPE HAVE NO SOURCE PART",
     "PATTERN FILE HAS PARTITIONED FORMAT WHICH REQUIRES ... ",
     "PATTERN FILES OF THIS TYPE HAVE NO VARIABLE INFORMATION",
     .
     .
     .
     0
};
```

DESCRIPTION

This is an array of strings which contains the pattern package subroutines error messages. ppemsg[pperrno] points to a string which contains the error message text.

SEE ALSO

pperrno(3L), ppoutemsg(3L)

NAME

pperrno - pattern external error indicator

SYNOPSIS

```
#include <ppsubs.h>      /* pattern definitions and struct */


int pperrno;
```

DESCRIPTION

This is the pattern library software's error depository. Any time an error occurs in a pattern library (-lpp) subroutine, the value of the external pperrno in set to a value which corresponds to the type of error. A string describing the type of error can be addressed by the expression **ppemsg[pperrno]**. Also the error values are described below (for the most up to date list of pperrno values see **ppsubs.h**).

PPBADDIR - The directory name given for the search path is too long (too many charaters). The directory name and pattern file name (including the ".p" or ".o") can be no longer than the **PPMAXNAM** value defined in **ppsubs.h**.

PPBADNAME - The pattern name had invalid syntax. Pattern names (like C language variables) must start with an alphabetic or '_' character (A-Z, a-z or '_'). They are also limited to a maximum length definedas **PPMAXNAM** in **ppsubs.h**.

PPBADPAT - The internal format of the pattern was not correct. This could occur if the pattern was not made by **ppmkpat(1L)** or if the pattern had been scribbled.

PPNOPAT - The pattern could not be openned or found.

PPNOSRC - This error occurs when the pattern format type is not standard. Only standard format type patterns have source included in the pattern file.

PPNOVI - This error occurs when the pattern format type is not standard. Only standard format type patterns have variable argument information included in the pattern file.

PPOVRFLOW - The part of the pattern to be read is larger than the buffer size as given in the subroutine call (**maxsize**). This was determined by comparing **max-size** to the information in the pattern header. No attempt was made to read anything into the buffer.

PPSYNTAX - The pattern definition given has one or more syntax errors. This was determined by the pattern compiler (ppmkpat(1L)), and the pattern compiler will have already have sent error messages to standard output.

PPSYSERR - A system call error occurred (usually no memory). Check the value of the external variable **errno**. The pattern was not read in.

NAME

ppfgainvi - get pattern variable info from stdio stream

SYNOPSIS

```
#include <ppsubs.h>       /* pattern definitions and struct */

        int pperrno;    /* ppsubs.h: error depository */

PPATVI *ppfgainvi(pfd,h,vihpaddr,vispaddr,vospaddr)
        register FILE *pfd; /* pattern stream pointer */
        struct PPHEAD *h;   /* pattern file header pointer */
        struct PPVIHEADER **vihpaddr; /* var info head ptr addr */
        struct PPVINFO **vispaddr;    /* var info struct ptr addr */
        struct PPVOCCUR **vospaddr;   /* var occur struct ptr addr */
```

DESCRIPTION

This is the Pattern Package Utility program to retrieve the variable information from a pattern file. Optionally, pointers may be set to various points in the variable information (e.g., the variable header, variable info structures array and variable occurance array), by passing the address of one or more pointers.

SEE ALSO

pperrno(3L), pattern(5L)

DIAGNOSTICS

ppfgainvi() returns a NULL value when an error occurs, and sets the value of the external variable **pperrno** to one of the following values (defined in <ppsubs.h>):

PPSYSERR - A system call error occurred (usually no memory). Check the value of the external variable **errno**. The pattern was not read in.

NAME

ppfgetpat - stdio get a common pattern package pattern

SYNOPSIS

```
#include <stdio.h>      /* only needed for ppfgetpat */
#include <ppsubs.h>

        int pperrno;    /* error type external */

PPAT *ppfgetpat(patname,pattype,dirso)
        char *patname;
        int pattype;
        PPATDIR dirso[];
```

DESCRIPTION

Ppfgetpat and **ppgetpat** provide an easy method for obtaining a pattern from the disk. **Ppfgetpat** is a **<stdio.h>** version of **ppgetpat** otherwise they function identically.

Ppfgetpat and **ppgetpat** use **ppfopenpat(3L)** and **ppopenpat(3L)** to open the pattern disk file. Once filename is opened, the pattern header is read. Memory is allocated (using **malloc(3)**), and the pattern is read into the memory. The pattern file is closed. **Ppfgetpat** and **ppgetpat** return a pointer to the pattern. The pattern memory may be freed or reallocated as described under **malloc(3)**.

SEE ALSO

ppgetpat(3L), ppfopenpat(3L), ppdefdso(3L), malloc(3), fopen(3), fclose(3)

DIAGNOSTICS

Ppfgetpat and **ppgetpat** return a (**PPAT ***) **NULL** when an error occurs. **Ppfgetpat** and **ppgetpat** will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs:

PPBADNAME - The pattern name had invalid syntax.

PPNOPAT - The pattern could not be openned or found.

PPBADPAT - The internal format of the pattern was not correct. This could occur if the pattern was not made by **ppmkpat(1L)** or if the pattern had been scribbled.

PPSYSERR - A system call error occurred (usually no memory). Check the value of the external variable **errno**. The pattern was not read in.

NAME

ppfopenpat - stdio open pattern disk file

SYNOPSIS

```
#include <stdio.h>  /* only needed for ppfopenpat */
#include <ppsubs.h>

     int pperrno;    /* error type external */
     char *ppathname;    /* full path name of openned file */

FILE *ppfopenpat(patname,pattype,dirso)
     char *patname;
     int pattype;
     PPATDIR dirso[];

int ppopenpat(patname,pattype,dirso)
     char *patname;
     int pattype;
     PPATDIR dirso[];
```

DESCRIPTION

Ppopenpat and **ppfopenpat** provides an easy method for openning a pattern file on the disk for reading. They are equivalent except that **ppfopenpat** is a **<stdio.h>** version of **ppopenpat.**

Ppopenpat and **ppfopenpat** first look at the string pointed to by **patname. Ppopenpat** and **ppfopenpat** use the string to form the disk file name for the pattern. To be valid, the string must be null terminated and no longer than 256 characters. If **patname** points to a valid string, then the string is copied into a buffer. If **patname** points to a **\0** (null string) or if **patname** = **NULL**, then **PPDFLTNAM** is copied into the buffer.

If **pattype** is **PPOBJFRMT**, then a **.o** is appended to the name in the buffer. If **pattype** is **PPSTDFRMT**, then a **.p** is appended. If **pattype** is **PPMODFRMT**, then nothing is appended. The address of the buffer is put into the external **ppathname.** This buffer is used for the filename.

If filename starts with a /, then **ppopenpat** and **ppfopenpat** will try to open filename. If filename does not start with a /, then **ppopenpat** and **ppfopenpat** will search the pattern directories (in order). The pattern directory search order may be specified as detailed in **ppdefdso(3L).**

If the search order is not specified (i.e., **dirso = (PPATDIR NULL)**, then a default order is used. The default search order is as follows:

/keyword	**pattern keyword and primitives directory**
.	**present working directory**
/compat	**common pattern directory**
/usr/pat	**common user pattern directory**

Ppopenpat and **ppfopenpat** will try to open filename in the first pattern directory in which filename is found.

Once filename is opened, **ppfopenpat** returns a stream file pointer, and **ppopenpat** returns the file descriptor of the open file.

SEE ALSO

ppdefdso(3L), ppdftdso(3L), pattern(5L)

DIAGNOSTICS

Ppopenpat returns a **EOF** when an error occurs. **Ppfopenpat** returns a **NULL** when and error occurs. **Ppopenpat** and **ppfopenpat** will set **pperrno** to one of the following values (defined in **ppsubs.h**) when a problem occurs:

PPBADDIR	- The directory name given for the search path is too long (too many charaters). The directory name and pattern file name (including the ".p" or ".o") can be no longer than the **PPMAXNAM** value defined in **ppsubs.h**.
PPBADNAME	- The pattern name had invalid syntax.
PPNOPAT	- The pattern could not be openned or found.

NAME

ppforkvar - external pattern fork trys variables

SYNOPSIS

```
#include <ppsubs.h>       /* pattern definitions and struct */


int pptryagain = { 0 };  /* how many times to try and fork */
int ppsleep = { 5 };     /* how many seconds sleep before next
                            try */
```

DESCRIPTION

These are the pattern library fork() variables. They are used by any pattern library subroutine which must fork() a new process (such as the pattern compiler, ppmkpat(1L), or RC:PAT).

SEE ALSO

ppmakepat(3L), pprcpat(3L), ppsccsgp(3L)

NAME

ppfrdhdr - stdio read pattern header

SYNOPSIS

```
#include <stdio.h>
#include <ppsubs.h> /* pattern definitions and structs */

        int pperrno;     /* error type external */

int ppfrdhdr(patstream,hdrptr)
        FILE *patstream;
        struct PPHEAD *hdrptr;
```

DESCRIPTION

Pprdhdr and **ppfrdhdr** read the header information from a pattern file (**patfdesorpatstream**). This information is read into a pattern header structure (**struct PPHEAD** as defined in the **<ppsubs.h>** header file) which is pointed to by **hdrptr.**

SEE ALSO

ppfopenpat(3L), ppfgetpat(3L), pphdrtell(3L), pphdrsiz(3L), pperrno(3L), pattern(5L)

DIAGNOSTICS

Normally this subroutine returns the number of bytes read. The subroutine returns a **NULL** when an error occurs. This subroutine will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs.

PPBADPAT - The size of a particular part of the pattern is smaller than indicated in the pattern header (i.e., the pattern has been scribbled or altered), or the pattern header has erroneous information in it (i.e., the pattern header is not a pattern header or the pattern file has been scribbled or altered).

PPSYSERR - A system call error occurred (usually read or seek problem). Check the value of the external variable **errno.** The pattern was not read in.

BUGS

If these subroutines are used to read pipes, then the seeks performed internal to the subroutines will most likely fail resulting in a PPSYSERR value in pperrno.

NAME

ppfrdpat - stdio read pattern

SYNOPSIS

```
#include <stdio.h>
#include <ppsubs.h> /* pattern definitions and structs */

        int pperrno;      /* error type external */

int ppfrdpat(patstream,patptr,maxsize,headptr)
        FILE *patstream;
        PPAT *patptr;
        int maxsize;
        struct PPHEAD *headptr;
```

DESCRIPTION

Pprdpat and **ppfrdpat** read the pattern part (part used by **ppmatch(3L)** and **match(3L)**) from a pattern file (**patfdesor-patstream**). The pattern part is read into a buffer which must start on a 16 bit word boundary which is pointed to by **patptr**. This buffer area is **maxsize** bytes in size. **Maxsize** should have a value greater than or equal to the value returned by the **ppatsiz(3L)** subroutine.

SEE ALSO

ppfopenpat(3L), ppfgetpat(3L), ppattell(3L), ppatsiz(3L), pperrno(3L), pattern(5L)

DIAGNOSTICS

Normally this subroutine returns the number of bytes read. The subroutine returns a **NULL** when an error occurs. This subroutine will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs.

PPBADPAT - The size of a particular part of the pattern is smaller than indicated in the pattern header (i.e., the pattern has been scribbled or altered), or the pattern header has erroneous information in it (i.e., the pattern header is not a pattern header or the pattern file has been scribbled or altered).

PPOVRFLOW - The part of the pattern to be read is larger than the buffer size as given in the subroutine call (**maxsize**). This was determined by comparing **maxsize** to the information in the pattern header. No attempt was made to read anything into the buffer.

PPSYSERR - A system call error occurred (usually read or seek problem). Check the value of the external variable **errno**. The pattern was not read in.

BUGS

If these subroutines are used to read pipes, then the seeks performed internal to the subroutines will most likely fail resulting in a PPSYSERR value in pperrno.

NAME

ppfrdsrc - stdio read pattern source

SYNOPSIS

```
#include <stdio.h>
#include <ppsubs.h> /* pattern definitions and structs */

        int pperrno;     /* error type external */

int ppfrdsrc(patstream,srcptr,maxsize,headptr)
        FILE *patstream;
        char *srcptr;
        int maxsize;
        struct PPHEAD *headptr;
```

DESCRIPTION

Pprdsrc and **ppfrdsrc** read the source part from a pattern file (**patfdesorpatstream**). The source part is only found in standard format type pattern files. This part comprises the ASCII character definition used to make the pattern. The source part is read into a buffer which is pointed to by **srcptr**. This buffer area is **maxsize** bytes in size. **Maxsize** should have a value greater than or equal to the value returned by the **ppsrcsiz(3L)** subroutine.

SEE ALSO

ppfopenpat(3L), ppfgetpat(3L), ppsrctell(3L), ppsrcsiz(3L), pperrno(3L), pattern(5L)

DIAGNOSTICS

Normally this subroutine returns the number of bytes read. The subroutine returns a **NULL** when an error occurs. This subroutine will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs.

PPBADPAT - The size of a particular part of the pattern is smaller than indicated in the pattern header (i.e., the pattern has been scribbled or altered), or the pattern header has erroneous information in it (i.e., the pattern header is not a pattern header or the pattern file has been scribbled or altered).

PPNOSRC - This error occurs when the pattern format type is not standard. Only standard format type patterns have source included in the pattern file.

PPOVRFLOW - The part of the pattern to be read is larger than the buffer size as given in the subroutine call (**maxsize**). This was determined by comparing **maxsize** to the information in the pattern header. No attempt was made to read anything into the buffer.

PPSYSERR - A system call error occurred (usually read or seek problem). Check the value of the external variable

errno. The pattern was not read in.

BUGS

If these subroutines are used to read pipes, then the seeks performed internal to the subroutines will most likely fail resulting in a PPSYSERR value in pperrno.

NAME

ppfrdvi - stdio read pattern variable information

SYNOPSIS

```
#include <stdio.h>
#include <ppsubs.h> /* pattern definitions and structs */

        int pperrno;     /* error type external */

int ppfrdvi(patstream,viptr,maxsize,headptr)
        FILE *patstream;
        int *viptr;
        int maxsize;
        struct PPHEAD *headptr;
```

DESCRIPTION

Pprdvi and **ppfrdvi** read the variable argument information part from a pattern file (**patfdesorpatstream**). The variable argument information part is read into a buffer which must start on a 16 bit word boundary which is pointed to by **viptr**. This buffer area is **maxsize** bytes in size. **Maxsize** should have a value greater than or equal to the value returned by the **ppvisiz(3L)** subroutine. If **pprdvi** or **ppfrdvi** return a **NULL** and **pperrno** == **NULL**, then the pattern is not a variable pattern (i.e., no variable arguments required). This is the only case where a **NULL** return value indicates a normal (no-error) termination.

SEE ALSO

ppfopenpat(3L), ppfgetpat(3L), ppvitell(3L), ppvisiz(3L), pperrno(3L), pattern(5L)

DIAGNOSTICS

Normally this subroutine returns the number of bytes read. The subroutine returns a **NULL** when an error occurs. This subroutine will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs.

PPBADPAT - The size of a particular part of the pattern is smaller than indicated in the pattern header (i.e., the pattern has been scribbled or altered), or the pattern header has erroneous information in it (i.e., the pattern header is not a pattern header or the pattern file has been scribbled or altered).

PPNOVI - This error occurs when the pattern format type is not standard. Only standard format type patterns have variable argument information included in the pattern file.

PPOVRFLOW - The part of the pattern to be read is larger than the buffer size as given in the subroutine call (**maxsize**). This was determined by comparing **maxsize** to the information in the pattern header. No attempt was made to read anything into the buffer.

PPSYSERR - A system call error occurred (usually read or seek problem). Check the value of the external variable **errno**. The pattern was not read in.

BUGS

If these subroutines are used to read pipes, then the seeks performed internal to the subroutines will most likely fail resulting in a PPSYSERR value in pperrno.

NAME

ppfgainvi - get pattern variable info from stdio stream

SYNOPSIS

```
#include <ppsubs.h>      /* pattern definitions and struct */

        int pperrno;   /* ppsubs.h: error depository */

PPATVI *ppfgainvi(pfd,h,vihpaddr,vispaddr,vospaddr)
        register FILE *pfd; /* pattern stream pointer */
        struct PPHEAD *h;   /* pattern file header pointer */
        struct PPVIHEADER **vihpaddr; /* var info head ptr
                                         addr */
        struct PPVINFO **vispaddr;    /* var info struct ptr
                                         addr */
        struct PPVOCCUR **vospaddr;   /* var occur struct ptr
                                         addr */
```

DESCRIPTION

This is the Pattern Package Utility program to retrieve the variable information from a pattern file. Optionally, pointers may be set to various points in the variable information (e.g., the variable header, variable info structures array and variable occurance array), by passing the address of one or more pointers.

SEE ALSO

pperrno(3L), pattern(5L)

DIAGNOSTICS

ppfgainvi() returns a NULL value when an error occurs, and sets the value of the external variable **pperrno** to one of the following values (defined in <ppsubs.h>):

PPSYSERR - A system call error occurred (usually no memory). Check the value of the external variable **errno**. The pattern was not read in.

NAME

ppgetpat - get a common pattern package pattern

SYNOPSIS

```
#include <ppsubs.h>

        int pperrno;    /* error type external */

PPAT *ppgetpat(patname,pattype,dirso)
        char *patname;
        int pattype;
        PPATDIR dirso[];
```

DESCRIPTION

Ppfgetpat and **ppgetpat** provide an easy method for obtaining a pattern from the disk. **Ppfgetpat** is a **<stdio.h>** version of **ppgetpat** otherwise they function identically.

Ppfgetpat and **ppgetpat** use **ppfopenpat(3L)** and **ppopenpat(3L)** to open the pattern disk file. Once filename is opened, the pattern header is read. Memory is allocated (using **malloc(3)**), and the pattern is read into the memory. The pattern file is closed. **Ppfgetpat** and **ppgetpat** return a pointer to the pattern. The pattern memory may be freed or reallocated as described under **malloc(3)**.

SEE ALSO

ppfgetpat(3L), ppopenpat(3L), ppdefdso(3L), malloc(3), fopen(3), fclose(3)

DIAGNOSTICS

Ppfgetpat and **ppgetpat** return a (**PPAT ***) **NULL** when an error occurs. **Ppfgetpat** and **ppgetpat** will set the value of **pperrno** to one of the following values (defined in **<ppsubs.h>**) when a problem occurs:

PPBADNAME	- The pattern name had invalid syntax.
PPNOPAT	- The pattern could not be openned or found.
PPBADPAT	- The internal format of the pattern was not correct. This could occur if the pattern was not made by **ppmkpat(1L)** or if the pattern had been scribbled.
PPSYSERR	- A system call error occurred (usually no memory). Check the value of the external variable **errno**. The pattern was not read in.

NAME

ppgetvi - get pattern variable info; open/read file

SYNOPSIS

```
#include <ppsubs.h>      /* pattern definitions and struct */

        struct PPHEAD pphead;    /* pattern header */
        int pperrno;   /* pattern subs error depository */
        int errno;     /* system I/O error depository */

PPATVI *ppfgetvi(name,type,dirso)
        char *name;     /* name of the pattern to get */
        int type;       /* pattern format type */
        PPATDIR *dirso; /* pattern directory search order */
```

DESCRIPTION

This is the Pattern Package Utility program to retrieve a the variable information from a pattern.

SEE ALSO

pphead(3L), pperrno(3L), intro(2), pattern(5L)

DIAGNOSTICS

ppgetvi() returns a NULL value when an error occurs, and sets the value of the external variable **pperrno** to one of the following values (defined in <ppsubs.h>):

PPSYSERR - A system call error occurred (usually no memory). Check the value of the external variable **errno**. The pattern was not read in.