NAME

ppgmdot - get mdot and field pointer buffer address

SYNOPSIS

char **ppgmdot()

DESCRIPTION

Ppgmdot(**3L**) will return the address of the buffer (which was set by the last **ppsmdot**). If **ppsmdot** had not been called prior to **ppgmdot**, then **ppgmdot** will return a zero.

Ppsmdot(**3L**) is used to tell the pattern matcher (**ppmatch**(**3L**)) the location of the buffer to be used to store the pointer values which are set by the **mdot**, **deffld**, **startfld** and **endfld** built-in patterns. If **ppsmdot** is never called or if the value of the **ppsmdot** argument is '(**int ***) **0**', then **mdot**, **deffld**, **startfld** and **endfld** primitives are ignored by the matcher.

Ppmdotsiz(**3L**) can be used to optain the size of the buffer whose address is obtained by ppgmdot().

SEE ALSO

ppmatch(3L), ppsmdot(3L), ppmdotsiz(3L), pattern(5L)

DIAGNOSTICS

Ppsmdot and **ppgmdot** produce no diagnostics, and they never change the value of **pperrno**.

BUGS

Ppsmdot and **ppgmdot** are very simple assembly language routines which are a part of the **ppmatch**(**3L**) subroutine in the pattern library. They do not use **csv**(**2**) and **cret**(**2**) so **adb**(**1**) will not show any auto variables for them.

NAME

ppgvdflt - get address of pattern variable default value

SYNOPSIS

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


char *ppgvdefault(viptr,index)
        PPATVI *viptr; /* pointer to the variable information */
        int index;     /* var info struct index */
```

DESCRIPTION

This subroutine returns a pointer to the <index> variable default found in the variable information pointed to by viptr.

SEE ALSO

pattern(5L)

DIAGNOSTICS

ppgvdflt() produces no diagnostics.

NAME

ppgvihdr - get address of pattern variable info header

SYNOPSIS

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


struct PPVIHEADER *ppgviheader(viptr)
        PPATVI *viptr; /* pointer to the variable information */
```

DESCRIPTION

This subroutine returns a pointer to the variable information header found in the variable information pointed to by viptr.

SEE ALSO

pattern(5L)

DIAGNOSTICS

ppgvihdr() produces no diagnostics.

NAME

ppgvinfo - get address of info on pattern variable

SYNOPSIS

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


struct PPVINFO *ppgvinfo(viptr,index)
        PPATVI *viptr; /* pointer to the variable information */
        int index;     /* var info struct index */
```

DESCRIPTION

This subroutine returns a pointer to the <index> variable information structure found in the variable information pointed to by viptr.

SEE ALSO

pattern(5L)

DIAGNOSTICS

ppgvinfo() produces no diagnostics.

NAME

ppgvname - get address of pattern variable name

SYNOPSIS

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


int *ppgvname(viptr,index)
        PPATVI *viptr; /* pointer to the variable information */
        int index;     /* var info struct index */
```

DESCRIPTION

This subroutine returns a pointer to the <index> variable name found in the variable information pointed to by viptr.

SEE ALSO

pattern(5L)

DIAGNOSTICS

ppgvname() produces no diagnostics.

NAME

ppgvoccur - get address of pattern variable occurrance

SYNOPSIS

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


struct PPVOCCUR *ppgvoccur(viptr,index)
        PPATVI *viptr; /* pointer to the variable information */
        int index;     /* var info struct index */
```

DESCRIPTION

This subroutine returns a pointer to the <index> variable occurance structure found in the variable information pointed to by viptr.

SEE ALSO

pattern(5L)

DIAGNOSTICS

ppgvoccur() produces no diagnostics.

NAME

pphdrsiz - return size of pattern header

SYNOPSIS

#include <ppsubs.h>

unsigned pphdrsiz()

DESCRIPTION

Pphdrsiz() is a macro defined in **<ppsubs.h>** and always returns the size (in bytes) of the pattern file header (**sizeof**(**struct PPHEAD**)) which is the same for any pattern file.

SEE ALSO

pattern(5L)

NAME

pphdrtell - return tell value for pattern header

SYNOPSIS

#include <ppsubs.h>

long pphdrtell()

DESCRIPTION

Pphdrtell() is a macro defined in **<ppsubs.h>** and always returns the tell value for the start of the pattern file header (**0L**) which is the same for any pattern file. The tell value can be used by **lseek**(**2**) or **fseek**(**3**).

SEE ALSO

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

NAME

pphead - external pattern file header buffer

SYNOPSIS

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


struct PPHEAD pphead;
```

DESCRIPTION

This is the pattern library pattern header structure. This structure is used by ppgetpat(3L) and ppsccsgp(3L) so that header information about the pattern which is returned is not lost.

SEE ALSO

ppfgetpat(3L), ppgetpat(3L), ppsccsgp(3L), pattern(5L)

NAME

ppmakepat - make pattern from definition

SYNOPSIS

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

        int ppsleep;    /* sleep time between fork trys */
        int pptryagain; /* how many fork() tries */
        int pperrno;    /* pattern subs error depository */
        int errno;      /* system I/O error depository */

int ppmakepat(patname,type,dirso,flags,def0,...,(char *) NULL)
        char *patname;      /* name of the pattern */
        int type;           /* pattern format type */
        PPATDIR *dirso;     /* pattern directory search order */
        unsigned flags;     /* ppmkpat program flags; +t */
        char *def0;         /* first definition string pointer */
```

DESCRIPTION

This is the pattern library subroutine which will take a pattern definition and make a pattern with arguments as follows:

patname This points to the name of the pattern to be created. This may be a full pathname (e.g., "/type01/pat/ex01"), but should not include the ".p" or ".o" ending.

type This describes the type (standard, object, etc.) of pattern to be created. One of the defined symbols in /usr/include/ppsubs.h should be used. For example PPSTDFRMT for a standard format type pattern.

dirso This describes the directory search order to be used when looking for predefined patterns in the definition. How to specify dirso is described in **ppdefdso(3L)**. If the default search order (as described in **ppdftdso(3L)**) is desired, then use **(PPATDIR *) NULL** for the value of dirso.

flags This variable allows the use of one or more compiler options for **ppmkpat(1L)**. The following options are available:

PPTRANFLAG	= +t; translate lowercase to uppercase
PPRESTRICT	= +r; restrict some built-in patterns
PPNOCPPFLAG	= -p; no C compiler prepass
PPONLYPPFLAG	= +p; only C compiler prepass
PPIPOKFLAG	= +ipok; output IP and OK acknowledgments

If more than one option is desired, then bit-or them together (e.g., PPTRANFLAG | PPRESTRICT will implement both the +t and +r ppmkpat(1L) options). If no options are desired, then use a 0 or NULL value for flags. The **ppsubs.h** header file should be consulted for additional options which may exist but are not described above.

def0 The definition may be one or more NULL terminated strings which are given as arguments after the **flags** argument. The last argument must always be a (**char ***) **NULL**. The pattern compiler (ppmkpat(1D)) is fork() and execve() with its input being each of the definitions strings given in the order they occur in the argument list.

SEE ALSO

ppsleep(3L), pptryagain(3L), pperrno(3L), intro(2), pattern(5L)

DIAGNOSTICS

ppmakepat() 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>):

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 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

ppmatch - pattern matcher

SYNOPSIS

```
        char    *ppcursor;
        char    *ppdot;

    int ppmatch(patptr,progarg)
        PPAT *patptr;
        PPROGARG *progarg;
```

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 **ppmatch**() are as follows:

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

progarg is a pointer to an array of application program defined inputs. The first element (**patarg[0]**) in the array must point to the start of the first text-area. All other elements of the array 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 patarg0 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

match(3L), ppchkpat(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

ppmdotsiz - return the size of mdot and field buffer

SYNOPSIS

```
#include <ppsubs.h>     /* pattern definitions and struct */
        int pperrno;    /* error type external */

unsigned ppmdotsiz(headptr)
        struct PPHEAD *headptr;
```

DESCRIPTION

Ppmdotsiz(3L) returns the size of the buffer (in bytes) required to store the pointers which are saved by the **mdot, deffld, startfld** and **endfld** built-in patterns which are included in the pattern whose header is pointed to by **headptr.** If **ppmdotsiz** return a **NULL** and **pperrno == NULL,** then the pattern does not need a buffer because no **mdot, deffld, startfld** or **endfld** built-in patterns were used. This is the only case where a **NULL** return value indicates a normal (no-error) termination.

Ppsmdot(3L) is used to tell the pattern matcher (**ppmatch(3L)**) the location of the buffer to be used to store the pointer values which are set by the **mdot, deffld, startfld** and **endfld** built-in patterns. If **ppsmdot** is never called or if the value of the **ppsmdot** argument is '**(int *) 0**', then **mdot, deffld, startfld** and **endfld** primitives are ignored by the matcher.

Ppgmdot(3L) will return the address of the buffer (which was set by the last **ppsmdot**). If **ppsmdot** had not been called prior to **ppgmdot,** then **ppgmdot** will return a zero.

SEE ALSO

ppmatch(3L), ppsmdot(3L), ppgmdot(3L), pattern(5L)

DIAGNOSTICS

When an error occurs in **ppmdotsiz,** it will return a **NULL** value and will set **pperrno** to one of the following values:

NULL	- As mentioned above, **NO ERROR EXISTS** a buffer is not needed because the pattern contains no **startfld, endfld** or **mdot** built-in patterns.
PPBADPAT	- The pattern header has erroneous information in it (i.e., the pattern header is not a pattern header or has been scribbled or altered).
PPNOMDOT	- This error occurs when the pattern format is not standard. Only standard format type patterns have the maximum **mdot** information.

BUGS

Ppmdotsiz() may return an erroneous (too small) value if one or more number variables are used in startfld, endfld, deffld or mdot built-in patterns. Ppmdotsiz() uses only built-in patterns without number variables when it determines the size of the buffer. This is normally not a problem because ppmatch(3L) and match(3L) will have many other problems if a variable pattern is used. They use only non-variable patterns (which includes variable patterns which have been compiled using specified arguments and default values into a non-variable pattern).

NAME

ppopenpat - 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

ppos -- address of char in string

SYNOPSIS

ppos(s1,c1,n1)
char *s1, c1;
int n1;

DESCRIPTION

Ppos returns the address of the character c1 within the string s1 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 address returned is zero.

If c1 occurs in s1 many more times than n1, the address 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 address that is returned is the address of the character c1 in s1. The returned address can be any legal address within the string s1. The address value of zero is reserved for the error return.

An empty string is one whose first character is the null character. If s1 is empty or c1 is the null character, the zero address is returned.

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

LIBRARY

/lib/lib3.a

SEE ALSO

pos(3L)

NAME

ppoutemsg - SCCS pattern software OM generater

SYNOPSIS

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

         char *ppemsg[]; /* reflexive error messages */
         int pperrno;    /* error number depository */
         int errno;      /* system error number depository */

ppoutemsg()
```

DESCRIPTION

This subroutine writes out an error message in standard SCCS format. **ppemsg[pperrno]** points to a string which contains the text of the error message. In some cases the value of errno is also used.

SEE ALSO

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

DIAGNOSTICS

ppoutemsg() does not return a value, but does return.

NAME

pprcpat - fork/exec RC:PAT command

SYNOPSIS

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

        int ppsleep;     /* sleep time between fork trys */
        int pptryagain;  /* how many fork() tries */
        int pperrno;     /* pattern subs error depository */
        int errno;       /* system I/O error depository */

int pprcpat(patname,ofctype,usrmsg)
        char *patname;  /* initial default pattern name */
        char *ofctype;  /* pointer to the office type character
                         * pair for example "01" = ESS1
                         */
        char *usrmsg;   /* pattern name for user message */
```

DESCRIPTION

This is the pattern library subroutine which will fork() and execve() the RC:PAT program so that the user can define a pattern. The default name is initially set to 'patname' and the SPCS type ('ofctype') is passed by the environment to /dist/rcpat.

SEE ALSO

ppsleep(3L), pptryagain(3L), pperrno(3L), intro(2), pattern(5L)

DIAGNOSTICS

pprcpat() 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 fork(2) or execve(2) failed). Check the value of the external variable **errno**. The pattern was not read in.

NAME

pprdhdr - read pattern header

SYNOPSIS

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

        int pperrno;     /* error type external */

int pprdhdr(patfdes,hdrptr)
        int patfdes;
        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

ppopenpat(3L), ppgetpat(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 for pperrno.

NAME

pprdpat - read pattern

SYNOPSIS

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

        int pperrno;      /* error type external */

int pprdpat(patfdes,patptr,maxsize,headptr)
        int patfdes;
        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 (**patfdes**or-**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

ppopenpat(3L), ppgetpat(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 (**max-size**). 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

pprdsrc - read pattern source

SYNOPSIS

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

        int pperrno;     /* error type external */

int pprdsrc(patfdes,srcptr,maxsize,headptr)
        int patfdes;
        char *srcptr;
        int maxsize;
        struct PPHEAD *headptr;
```

DESCRIPTION

Pprdsrc and **ppfrdsrc** read the source part from a pattern file (**patfdes** or **patstream**). 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

ppopenpat(3L), ppgetpat(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

pprdvi - read pattern variable information

SYNOPSIS

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

        int pperrno;     /* error type external */

int pprdvi(patfdes,viptr,maxsize,headptr)
        int patfdes;
        int *viptr;
        int maxsize;
        struct PPHEAD *headptr;
```

DESCRIPTION

Pprdvi and **ppfrdvi** read the variable argument information part from a pattern file (**patfdes** or **patstream**). 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

ppopenpat(3L), ppgetpat(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

ppsccsgp - SCCS get/create pattern from definition

SYNOPSIS

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

        int ppsleep;   /* sleep time between fork trys */
        int pptryagain;     /* how many fork() tries */
        struct PPHEAD pphead;    /* pattern header */
        int pperrno;   /* pattern subs error depository */
        int errno;     /* system I/O error depository */

PPAT *ppsccsgp(patdef,ofctype,usrmsg)
        char *patdef;  /* definition or name of the pattern */
        char *ofctype; /* two character string for the office
                          type */
        char *usrmsg;  /* user message pattern name */
```

DESCRIPTION

This is the pattern library subroutine which will take a a pattern definition and return a pattern. The definition may be a single pattern name, then ppgetpat(3L) is used to read it off of the disk. If the definition is more complicated, then the pattern compiler ppmkpat(1L) is fork(2) execve(2) with its input being the string pointed to by patdef.

If the definition comprises only a "+", then the RC:PAT program is called (using pprcpat(3L)) and the user is allowed to input the definition from his terminal.

SEE ALSO

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

DIAGNOSTICS

ppsccsgp() 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>):

PPBADNAME - The pattern name had invalid syntax.

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.

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 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

ppsleep - external pattern sleep time between fork trys varaible

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**(2) variables. They are used by any pattern library subroutine which must **fork**(2) a new process (such as the pattern compiler, **ppmkpat**(1L), or **RC:PAT**).

SEE ALSO

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

NAME

ppsmdot - set mdot and field pointer buffer for matcher

SYNOPSIS

```
ppsmdot(mdotbufptr)
        char **mdotbufptr;
```

DESCRIPTION

Ppsmdot(3L) is used to tell the pattern matcher (**ppmatch(3L)**) the location of the buffer to be used to store the pointer values which are set by the **mdot, deffld, startfld** and **endfld** built-in patterns. If **ppsmdot** is never called or if the value of the **ppsmdot** argument is '**(int *) 0**', then **mdot, deffld, startfld** and **endfld** primitives are ignored by the matcher.

Ppgmdot(3L) will return the address of the buffer (which was set by the last **ppsmdot**). If **ppsmdot** had not been called prior to **ppgmdot**, then **ppgmdot** will return a zero.

No check is made to ensure that the **ppsmdot** argument is valid or that it points to a large enough area to hold everything that is going to be put there. For example, if a '**mdot(<index>)**' pattern occurs, then the matcher writes the cursor value into memory location ***(mdotbufptr + index*2)**. To avoid some problems **ppmdotsiz** should be used to obtain the maximum offset from **mdotbufptr** which may occur.

SEE ALSO

ppmatch(3L), ppgmdot(3L), ppmdotsiz(3L), pattern(5L)

DIAGNOSTICS

Ppsmdot and **ppgmdot** produce no diagnostics, and they never change the value of **pperrno**.

BUGS

Ppsmdot and **ppgmdot** are very simple assembly language routines which are a part of the **ppmatch(3L)** subroutine in the pattern library. They do not use **csv(2)** and **cret(2)** so **adb(1)** will not show any auto variables for them.

NAME

ppsrcsiz - return size of pattern source definition

SYNOPSIS

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

        int pperrno;    /* error type external */

unsigned ppsrcsiz(headptr,patstream)
        struct PPHEAD *headptr;  /* pointer to pattern header  */
        FILE  *patstream;        /*  pattern file stream (stdio)
        */
```

DESCRIPTION

Ppsrcsiz() returns the size (in bytes) of the pattern source definition part of the pattern file with header pointed to by **headptr** and in the stdio file stream of **patstream.**

SEE ALSO

pattern(5L)

DIAGNOSTICS

Ppsrcsiz() 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.

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

NAME

ppsrctell - return tell value for pattern source definition

SYNOPSIS

```
#include <ppsubs.h>

        int pperrno;     /* error type external */

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

DESCRIPTION

Ppsrctell() returns the tell value for the start of the source 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

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

NAME

pptryagain - external pattern fork trys varaible

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), ppsleep(3L)

NAME
ppvisiz - return size of pattern variable information

SYNOPSIS
```
#include <ppsubs.h>

        int pperrno;     /* error type external */

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

DESCRIPTION
Ppvisiz() returns the size (in bytes) of the variable argument information part of the pattern file with header pointed to by **headptr**. If **ppvisiz**() returns 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
pattern(5L)

DIAGNOSTICS
Ppvisiz() 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.

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.

NAME

ppvitell - return tell value for pattern variable information

SYNOPSIS

```
#include <ppsubs.h>

        int pperrno;     /* error type external */

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

DESCRIPTION

Ppvitell() returns the tell value for the start of the variable argument information part of the pattern file with header pointed to by **headptr**. If **ppvitell**() returns a (**0L**) and **pperrno** == (**0L**), then the pattern is not a variable pattern (i.e., no variable arguments required). This is the only case where a (**0L**) return value indicates a normal (no-error) termination. The tell value can be used by **lseek**(**2**) or **fseek**(**3**).

SEE ALSO

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

DIAGNOSTICS

Ppvitell() 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.

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.

NAME

prepeat -- concatenate identical strings n times

SYNOPSIS

```
prepeat(s1,s2,n1)
char *s1, *s2;
int n1;
```

DESCRIPTION

Prepeat returns a pointer to the address of the position after the last character in the string _s1_. The value returned is the same as that returned by the _plen_ function.

_s1_ buffer area for the target string.

_s2_ source string which is copied into _s1_.

_n1_ integer which specifies the number of times _s2_ is copied into _s1_.

If the address pointed to by _s1_ is zero, the address returned is zero.

If the value of _n1_ is negative or zero, the target string _s1_ will be empty and the returned address will point to the null character at the beginning of string _s1_.

If the value of _n1_ is positive, the characters of the string _s2_ are copied into the string _s1_ the number of times indicated by _n1_. The target string _s1_ is then terminated with the null character. It should be noted that _prepeat_ becomes a copy string function when _n1_ is one.

The strings _s1_ and _s2_ are each defined as a null terminated array of characters. The returned address minus the starting address of the string _s1_ is the length of the string.

An empty string is one whose first character is the null character. If _s2_ is empty, the target string _s1_ will be set empty and the returned address will point to the null character at the beginning of the string.

LIBRARY

/lib/lib3.a

SEE ALSO

repeat(3L)

NAME

prmnull -- remove nulls from a series of strings

SYNOPSIS

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

DESCRIPTION

Prmnull returns a pointer indicating the address of the terminating null character for string _s1_ after _n1_ nulls have been removed and replaced with the character _c1_.

_s1_ string which is to be elongated by removal of nulls.

_c1_ character to which the null character is translated.

_n1_ number of null to _c1_ translations to be performed.

The string _s1_ is defined as a null terminated array of characters.

An empty string is one whose first character is the null character. If string _s1_ is empty and _n1_ is zero the address returned is the value of _s1_.

If the address pointed to by _s1_ is zero, the address returned will be zero.

If _c1_ is null, _prmnull_ returns the same address as it does for other values of _c1_ except that intervening nulls are not modified.

If the value of _n1_ is zero or negative, the address returned is the address of the terminating null character of the unmodified string _s1_.

LIBRARY

/lib/lib3.a

NAME

prompt -- prompt user

SYNOPSIS

```
char punc;
char instr[128];

prompt(str)
char *str;
```

DESCRIPTION

Prompt prompts the user (on the standard error output) with what is in *str* and collects the user's response in *instr*. The returned value is the length of the string that the user typed in. The user response must be terminated by a '!' or a '/', which is not part of the response but is returned in *punc*. New lines and leading and trailing spaces are ignored.

LIBRARY

/lib/lib1.a

DIAGNOSTICS

If the standard input cannot be read, the result is as if a zero lingth string was read with the terminating punctuation '/'.

NAME

pspan -- look for first char not in pattern

SYNOPSIS

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

DESCRIPTION

Pspan returns an address 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 returns the address of the first character found in the searched string that was not in the pattern 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.

The error code, zero, is returned only if the searched string, *s1*, is empty.

If a character not in the string *s2* is found in the string *s1*, the address of that character in *s1* will be returned.

If the entire string *s1* is searched and every character matches the pattern, the length pointer of the string *s1* is returned. The length pointer is the address of the terminating null byte.

LIBRARY

/lib/lib3.a

SEE ALSO

span(3L)

NAME

psubstr - copy substring of a string

SYNOPSIS

```
psubstr(s1,s2,p1,p2)
char *s1, *s2, *p1, *p2;
```

DESCRIPTION

Psubstr returns a pointer whose value is the address of the terminating null character at the end of the target string _s1_. The substring of _s2_ as specified by _p1_ and _p2_ is copied into _s1_. The address returned is the same as that returned by the function _plen_.

_s1_ the target string into which the extracted substring is copied. The target string is null terminated.

_s2_ the string from which the substring is to be extracted.

_p1_ a pointer that indicates the starting address of the substring in _s2_.

_p2_ a pointer that indicates the address of the last character in _s2_ to be transferred into _s1_.

An empty string is one whose first character is the null character. If the source string, _s2_, is empty, the target string, _s1_, is set to empty and the address return is zero. The exception to the above is when _p1_ points to the null character of the empty string and _p2_ is of an equal or higher address. In this case, the address returned is the address of the null character in string _s1_.

If the address pointed to by _s1_ is zero, the address returned is zero.

If _p1_ is higher than _p2_ or addresses a character past the end of the string, the target string is set empty and the address returned is zero.

The address of _p2_, however, may be any value equal or higher than _p1_. If _p2_ points to a character past the end of the source string, the substring will terminate with the last character of the the source string.

The only time that the address of the first character of the target string is returned is when _p1_ points to the null character of the source string. If _p1_ points to the null character of the source string the target string is set empty. For these cases _p2_ may be equal or higher than _p1_.

LIBRARY
/lib/lib3.a

SEE ALSO
substr(3L)

NAME

ral1msg -- notify the alerting system of a change in the logging status for a channel.

SYNOPSIS

```
#include <fs.h>

ral1msg(fspa, mp)
struct FS *fspa;
char *mp;
```

DESCRIPTION

FS_SEMA semaphore should be locked and unlocked around the call to this subroutine in order to prevent the logger from writing into the alerter 1 pipe at the same time as this routine.

LIBRARY

/lib/lib1.a

SEE ALSO

rcaltupd(3L)

RESTRICTIONS

This routine will not exist beyond SC5.

NAME

rateest -- estimate the rate of occurance of events.

SYNOPSIS

rateest()

DESCRIPTION

Based on the time of the current and previous call to _rateest_, the rate of occurance of calls is estimated using 7/8 first order linear filtering. The return value is the estimated rate in events per hour.

LIBRARY

/lib/lib1.a

NAME

ratesamp -- sampling function.

SYNOPSIS

ratesamp(crate, drate, minrate, inh)

DESCRIPTION

ARGUMENTS:

crate is current estimated rate of events

drate is desired rate of events

minrate is the minimum desired rate

inh is 1 if sampling cannot be done for the current event

Ratesamp returns 1 if event should be processed, 0 if not, attempting to get current rate to remain less than *drate*; inhibited events always cause a 1 return, but are approximately accounted for in the sampling. A 1 is also returned if the desired rate is less than *minrate*.

LIBRARY

/lib/lib1.a

SEE ALSO

rateest(3L)

NAME

rcaltupd -- notify the alerting system of a change in the data base for a logging channel.

SYNOPSIS

```
#include <chldata.h>
rcaltupd(cid, cdba)
struct CHL_B *cdba;
```

DESCRIPTION

LIBRARY

/lib/lib1.a

NAME

rctyps, rcbtyps -- standard recent change types

SYNOPSIS

#include <rc.h>

#include <rcbtyps.h>

DESCRIPTION

```
rctyps -- List of standard recent change types
referenced by rc.h
char *rctyps[] {
     "new",
     "out",
     "chg",
     0
};
```

rcbtyps -- List of standard recent change types referenced by the header file rcbtyps.h, and used by RC:BUILD programs.

```
char *rcbtyps[]     {
                "new",
                "out",
                "chg",
                "add",
                0
           };
```

LIBRARY

/lib/lib1.a

NAME

readint - buffered input for files containing integer data

SYNOPSIS

#include <rwint.h>
readint(inbuf)
struct IOBUF *inbuf;

DESCRIPTION

This subroutine provides buffered input capability for files containing integer data in records whose size is a power of two. It returns the starting address of the record in r0. However, if an error is detected or an end of file is encountered, a 0 is returned in r0 and a return code is returned in the structure variable, errval. The possible return codes are discussed below under DIAGNOSTICS. *Inbuf* is the address of a 522(10) byte buffer area whose format is:

```
    struct IOBUF
      {  int fildes;
         int errval;
         int idata;
         int recsize;
         int nread;
         int intbuf[IBUFSIZE];
      };
```

where

fildes	is the file descriptor of an open input file.
errval	is the return code which indicates an I/O error or an EOF.
idata	is the current number of records in the buffer that have been retrieved by the calling program. The calling program initializes this variable by setting it equal to the maximum number of records that can be contained in intbuf[].
recsize	is the record size in words. The record size must be a power of two; ie. 2, 4, 8, 16, etc., words.
nread	contains the number of bytes that have been read into the buffer. This variable should not be used or changed by the calling program.
intbuf	is the data buffer and should not be written into by the calling program.
IBUFSIZE	contains the value, 256.

The calling program must initialize the following structure variables for each input file that is to be read. These variables must be initialized prior to the first call to this subroutine to read the appropriate input file.

```
<structure>.fildes= <file descriptor of input file>;
<structure>.idata= <max. number of records that will
           fit in buffer (IBUFSIZE/recsize)>;
<structure name>.recsize= <record size in words>;
```

FILES

/usr/include/rwint.h which contains the definitions for IOBUF and IBUFSIZE.

LIBRARY

/lib/lib1.a

SEE ALSO

writint(3L)

DIAGNOSTICS

When this subroutine returns a 0 in r0, the following codes are returned in the structure variable, errval:

-1	I/O error.	0	End of file.

BUGS

NAME

refcdec, refcinc -- decrement/incremint mln's reference count in the lindata file

SYNOPSIS

refcdec(mln, fd)

refcinc(mln, fd)

DESCRIPTION

If _fd_ is negative, then the linedata file is opened and closed; otherwise _fd_ is the file descriptor of the already open linedata file.

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Returns:

```
 0   OK
-1   Could not open, read, or write the linedata file
-2   Invalid mln number
```

NAME

releaseds -- release data communications equipment

SYNOPSIS

#include <dial.h>

releaseds(dn_ptr) struct dntable *dn_ptr;

DESCRIPTION

The purpose of this subroutine is to process all releases of data communications equipment that was allocated by way of the *getds*(*3L*) subroutine. #*dn ptr* is a pointer to the structure returned by *getds*(*3L*).

Return values:

1 - the release was sucessful.
0 - the release was not sucessful.

FILES

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

LIBRARY

/lib/lib1.a

SEE ALSO

dial(3L), getds(3L)

NAME
ret_env -- return an environment parameter

SYNOPSIS
```
char **ret_env(parm_name)

char *parm_name;
```

DESCRIPTION
Ret env searches the current environment for the named parameter. The current environment is defined as the environment pointed to by the global cell, char **environ, set up by the C run-time start-off routine. The argument supplied, parm name, must be a pointer to a string specifying the target parameter. By convention, this string consists only of upper case alpha characters. The value returned by this routine allows the parameter to be redefined by changing the contents of the pointer returned. The new contents should be the address of a string having the form: <name>=<value> and stored in a protected, global (i.e.,non-volatile) data area.

LIBRARY
/lib/lib1.a

SEE ALSO
add_to_env(3L), exec(2), environ(7)

DIAGNOSTICS
If the named parameter is found, a pointer to the environment parameter pointer is returned, otherwise NULL is returned.

NAME

repeat -- concatenate identical strings n times

SYNOPSIS

```
repeat(s1,s2,n1)
char *s1, *s2;
int n1;
```

DESCRIPTION

Repeat returns an integer indicating the length of the resulting string *s1*. The value returned is the same as that returned by the *len* function.

s1 buffer area for the target string.

s2 source string which is copied into *s1*.

n1 integer which specifies the number of times *s2* is copied into *s1*.

If the address pointed to by *s1* is zero, the value returned is -1.

If the value of *n1* is negative or zero, the target string *s1* will be empty and the returned value will be zero.

If the value of *n1* is positive, the characters of the string *s2* are copied into the string *s1* the number of times indicated by *n1*. The target string *s1* is then terminated with the null character. It should be noted that *repeat* becomes a copy string function when *n1* is one.

The strings *s1* and *s2* 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.

An empty string is one whose first character is the null character. If *s2* is empty, the target string *s1* will be set empty and the returned value will be zero.

LIBRARY

/lib/lib3.a

SEE ALSO

prepeat(3L)

NAME

rmdir - remove directory

SYNOPSIS

```
rmdir (dirname)
char *dirname;
```

DESCRIPTION

Rmdir removes the directory specified by the partial or full pathname, *dirname*. *Dirname* is a string pointer. *Rmdir* checks the effective user id before doing anything. If the effective uid is not super user, control is returned to the caller with a -2 return value. Thereafter no checking is done on any unlinks or closings. Hence if the process executing *rmdir* is aborted or killed in the process of doing an unlink the file system could result in a bad link count.

Return codes:

0	successful *rmdir*.
-1	*dirname* not a directory or nonexistent.
-2	not allowed (could not open *dirname* or not super user).
-3	*dirname* not empty.

Rmdir calls *close*(2), *getuid*(2), *open*(2), *read*(2), *stat*(2) and *unlink*(2) while removing *dirname*.

SEE ALSO

mkdir:o(3C)

BUGS

Rmdir should not require the effective user id to be super-user.

NAME

rmgun -- remove group or user name

SYNOPSIS

rmgun(name,bit)
char name;
int bit;

DESCRIPTION

rmgun removes the given user (name) from the /etc/passwd file (if bit is 0) or removes the given group (name) from the /etc/group if the bit is non zero. rmgun will retun:

```
 0 - if it was successful.
-1 - if it could not find the user/group in
     the passwd/group file.
-2 - system problems: could not open or link
     passwd/group or /etc/ptmp files, or perform any
     of the other routine system calls.
     Close 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/group /etc/passwd /etc/ptmp

BUGS

NAME

rmnull -- remove nulls from a series of strings

SYNOPSIS

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

DESCRIPTION

Rmnull returns an integer indicating the length of the string _s1_ after _n1_ nulls have been removed and replaced with the character _c1_.

_s1_ string which is to be elongated by removal of nulls.

_c1_ character to which the null character is translated.

_n1_ number of null to _c1_ translations to be performed.

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 string _s1_ is empty and _n1_ is zero, the value return is zero.

If the address pointed to by _s1_ is zero, the value returned will be -1.

If _c1_ is null, _rmnull_ returns the same value as it does for other values of _c1_ except that intervening nulls are not modified.

If the value of _n1_ is zero or negative, the value returned is the length of the string _s1_ without modification.

LIBRARY

/lib/lib3.a

NAME

runlvl_ -- returns the run level read from /*etc*/*utmp*

SYNOPSIS

runlvl(flag)

DESCRIPTION

If *flag* is 0, then ,upon error, an error message is printed on the system console by calling *glberr*(*3L*).

FILES

/etc/utmp

LIBRARY

/lib/lib1.a

DIAGNOSTICS

Upon error -1 is returned (cannot open or read utmp or cannot find the RL entry)

NAME

scanf1 -- formatted input scanner

SYNOPSIS

int scanf1([-j[,input-string]],control-string,arg1,arg2,...)
char *input-string;
char *control-string;

DESCRIPTION

Scanf1 is patterned after the interface existing for the portable library routine scanf. It was developed to perform most of the features offered by scanf without incurring the penalty of scanf's size (approximately 7000 bytes). The size of scanf1 is about 1650 bytes.

Scanf1 is designed to read either from terminals or strings. On reads from terminals, scanf provides its own buffer. Terminal reads in excess of 100 characters may cause errors.

Scanf1 reads characters, interprets them according to a format and stores the results in its arguments. It expects as arguments:

1. An optional input-string, indicating the source of the input characters; if omitted the standard input is read.
2. A control-string described below.
3. A set of arguments, each of which must be a pointer, indicating where the converted input should be stored.

The integer j must be in the range of 4>j>0. If (j&1) is not equal to zero, the optional input string is to be specified. If (j&2) is not equal to zero, indirection is specified. See the description for format specification "i" below.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines which are ignored.
2. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, and optional numerical field width, and a conversion character.

A conversion specification is used to direct the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by the * character. An input field is defined as a string of non-space characters; it extends either to the next space character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. Pointers, rather than variable names, are required by the "call-by-value" semantics of the C language. The following conversion characters are legal:

d	indicates that a decimal integer is expected in the input stream; the corresponding argument should be an integer pointer.
o	indicates that an octal integer is expected in the input stream; the corresponding argument should be an integer pointer.
s	indicates that a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating "\0", which will be added. The input field is terminated by a space character or a newline.
a	indicates that a character string of non-space, non-slash, non-exclamation point characters is expected at this point. Otherwise it is handled as for "s" above.
r	indicates that all internal pointers are to be reset. From the terminal this will force a read.
i	indicates that the next argument in the call to scanf is to be taken as the address of a new argument list. All converted inputs are stored as directed by this argument list. There is no return to the original argument list.
c	indicates that a single character is expected; the corresponding argument should be a character pointer; the next input character is placed at the indicated spot. The normal skip over space characters is suppressed in this case; to read the next non-space character use %1s.
[	indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array. Right bracket may be escaped within brackets by preceding it with back slash.

For example, the call:

```
    int i;int j;char name[50];
    scanf1("%d%o%a",&i,&j,name);
```

with the input line

```
  77 77 test/
```

will assign to i the value of 77, to j the value of octal 77, and name will contain "test\0". The subsequent call

scanf1("%1s",name)

will move the string "/\0" into the array name.

Care should be exercised when reading from the terminal. If a format is specified such that it successfully matches to the end of the last string read, another read will be made from the terminal. This might cause the program to go to sleep on the terminal. The conversion character "a" is designed to make this problem easier to avoid from the SCCS shell.

Scanf1 returns as its value the number of successfully matched and assigned input items. This can be used to decide how many input items were found. On end of file, -1 is returned; note that this is different from 0, which means that the next input character does not match what you called for in the control string. Scanf1, if given a first argument of -1, will scan a string in memory given as the second argument. It differs from scanf in that the switching of input streams from a terminal to a string causes the pointers to the terminal stream to be lost. If a subsequent read is made to the terminal it should be reinitialized with the conversion character r. All scans from a string are automatically reinitialized.

LIBRARY

/lib/lib1.a

SEE ALSO

scanf(1S)

RESTRICTIONS

Used only prior to SC6.

NAME

sccerr -- SCC error routine

SYNOPSIS

sccerr(spcl,etype,ecode,enumber,emsg)
char *spcl,*etype,*ecode,*enumber,*emsg;

DESCRIPTION

This subroutine prints an error message on the user's standard output device (file descriptor = 2). If the severity or type of the error is not minor or major, _sccerr_ returns to the calling program. If, on the other hand, the severity or type of the error is minor or major, the subroutine _glberr_ is called to print a system error message on the system teletype, cause the error message to be logged onto the system error file, and cause an appropriate audible alarm to be generated.

Sccerr has five arguments, _spcl_, _etype_, _ecode_, _enumber_ and _emsg_. A description of these arguments follows.

spcl is the address of a string containing the special characters associated with an error message.

etype is the address of a string containing the severity or type of error.

ecode is the address of a string containing a three-character error code.

enumber is the address of a string containing a three-character error number.

emsg is the address of a string containing the message associated with the error.

LIBRARY

/lib/lib1.a

SEE ALSO

fmterr(3), glberr(3), shlerr(1L)

DIAGNOSTICS

BUGS

NAME

sccissue -- returns an identification if the current generic, generic issue and processor type being run

SYNOPSIS

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

sccissue(flag)
int flag;
```

DESCRIPTION

ARGUMENTS:

flag - the error printing control flag as defined for the e_subroutines and defined in *errfct.h*

RETURNS:

An integer value which is -1 if any errors are detected in opening, reading or the format of the sccissue file as defined in *sccissue.h*. When no errors are detected from a call of the form:

x = *sccissue*(*flag*);

the information is:

(&x)->*r_mach* = the processor type as defined by the define symbols **MACH40** and **MACH70** in *sccissue.h*

(&x)->*r_genumb* = the generic number of the system

(&x)->*r_isnumb* = the generic issue number of the system

The header file *sccissue.h* should be included by the calling program so that it may interpret the returned value.

FILES

The sccissue file as defined by the define symbol **I_NAME**(in *sccissue.h*) is read to determine the required data.

LIBRARY

/lib/lib1.a

NAME

sd_tbl -- extract requested information from signal distributor table file

SYNOPSIS

```
#include <sdtbl.h>

sd_tbl(funct,fd,key,imax,iarray);
int funct, fd, imax;
int iarray[];
char *key;

sdtb_fr(fd,key,imax);
sdtb_usd(fd,imax,iarray);
sdtb_ssd(fd,imax,iarray);
```

DESCRIPTION

Sd tbl searches an appropriate signal distributor table file and extracts the requested information specified by funct and key. If an error is detected a negative value is returned as discussed below.

The argument funct identifies the type of function that is to be performed. Examples are **SDF_FR**, to extract the frame type of a specific circuit and **SDF_ALL**, to extract all frame types for a specified circuit type.

The argument fd is the file descriptor of an opened signal distributor table file.

The argument key is the address of a null terminated character string identifying the specific circuit or circuit type for which the information is to be extracted.

The argument imax is an integer specifying the maximum number of circuits of type key that can be provided.

The argument iarray, if non-zero, specifies the address of a user supplied integer array of size imax which stores the frame types extracted by the function **SDF_ALL**.

To simplify the user's interface macros have been defined to access sd tbl. A brief description of the macros and their functions are given below.

sdtb_fr(fd,key,imax);
Extracts the frame type for a specific circuit. If the circuit does not exist a value of **SDR_CR** is returned. If the specified circuit does exist the value **>SDR_NORM** is returned to the calling program.

sdtb_usd(fd,imax,iarray);
Extracts the frame types for all supported USD circuits.

The value **SDR_NORM** is returned upon completion.

sdtb_ssd(fd,imax,iarray);
Extracts the frame type for all supported SSD circuits. The value **SDR_NORM** is returned upon completion.

The arguments to the macros are the same as those described above.

FILES

/usr/include/sdtbl.h which specifies the structure of a signal distributor table file entry and defines the return values and macro calls.

/usr/include/sdtint.h which initializes an array of character strings to valid frame types.

LIBRARY

/lib/libl.a

DIAGNOSTICS

The error codes returned by this subroutine are:

SDR_FUNC	The specified function is invalid.
SDR_READ	Error detected while trying to read the signal distributor table file.

BUGS

NAME

sindex -- find position of substring within a string

SYNOPSIS

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

DESCRIPTION

Sindex returns an integer 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 value returned is -1.

If _s2_ occurs more than once in _s1_, the starting position 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 integer that is returned is the array index of the substring in _s1_. The returned integer can have values from zero to 32767.

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 -1. If both argument strings are empty, the result returned is zero.

LIBRARY

/lib/lib3.a

SEE ALSO

pindex(3L)

NAME

span -- look for first char not in pattern

SYNOPSIS

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

DESCRIPTION

Span returns an integer indicating the success or failure of the pattern match. If the value returned is a positive array index the match was a success. If the value returned is -1 the match was a failure. This function returns the index of the first character found in the searched string that was not in the pattern 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.

The error code, -1, is returned only if the searched string, s1, is empty.

If a character not in the string s2 is found in the string s1, the array index of the character position in s1 will be returned.

If the entire string s1 is searched and every character matches the pattern, the length of the string s1 is returned. The length is the array index of the terminating null byte.

LIBRARY

/lib/lib3.a

SEE ALSO

pspan(3L)

NAME

spinoff -- ask user of process is to be spun off, then spin off if requested.

SYNOPSIS

spinoff(sig, sigval)

DESCRIPTION

Sig is the signal which is to be sent to the spunoff process in case the user wants to abort it using the ABORT input message. _Sigval_ is the interrupt address in the calling program. 0 and 1 are also acceptable values for _sigval_. If _sig_ is not a legal signal, INTR is used, but note that INTR's and QUIT's are ignored automatically.

Spinoff prompts the user to determine if process is to be spun off. If not, it returns a value of 0. If yes, it forks twice; the resulting processes do the following:

```
grandparent (original process):
      generates control file, passes it to
      child, waits for child to initialize it,
      then exits so that parent shell will return.

parent:
      waits for child to die, then removes control
      file and exits. removing control file is
      its only function.

child:
      sets up control file passed from grandparent,
      sets up abort signal by doing signal(sig, sigval)
      makes himself low priority (20),
      and returns a 1 to calling program.
```

If any I/O error occurs, or if either fork fails, an appropriate error message is printed and a 0 is returned, as if the user had indicated no spinoff.

LIBRARY

/lib/lib1.a

NAME

stdtime -- get date, time

SYNOPSIS

```
char *stdtime(tvecptr)
long *tvecptr;
```

DESCRIPTION

Converts the time pointed to by _tvecptr_ (such as returned by _time_(_2_)) and returns a pointer of the character string

mm/dd/yy hh:mm
SCCS

with date and time filled in

LIBRARY

/lib/lib1.a

SEE ALSO

timoa(3L)

NAME

stolc - ASCII string to lower case conversion

SYNOPSIS

```
stolc(strptr)
char *strptr;

nstolc(strptr, n)
char *strptr;
int n;

argstolc(argc, argptr)
int argc;
char *argptr[];
```

DESCRIPTION

stolc -- scans the null-terminated, ASCII string pointed to by 'strptr' and converts all upper case ASCII characters into lower case ASCII characters.

nstolc -- scans at most the first 'n' characters in the null-terminated, ASCII string pointed to by 'strptr' and converts all upper case ASCII characters into lower case ASCII characters.

argstolc -- converts arguments to main programs (by using argc and argv as arguments to the routine) or any null-terminated array of null-terminated ASCII strings to lower case. 'argc' represents the number of null-terminated ASCII strings in the array 'argptr'.

FILES

LIBRARY

/lib/lib3.a

NAME

styp_nams -- Initialize lists of names of supported network switch types

SYNOPSIS

#include <nwktbl.h>

DESCRIPTION

Lists of lower and upper case names of supported network switch types referenced by common header file nwktbl.h:

```
char *styp_lnams[] {
        "",
        "ferreed",
        "remreed",
};

char *styp_unams[] {
        "",
        "FERREED",
        "REMREED",
};
```

LIBRARY

/lib/libl.a

NAME

substr -- copy substring of a string

SYNOPSIS

```
substr(s1,s2,n1,n2)
char *s1, *s2;
int n1, n2;
```

DESCRIPTION

Substr returns an integer whose value is the length of the target string _s1_. The substring of _s2_ as specified by _n1_ and _n2_ is copied into s1. The value returned is the same as that returned by the function _len_.

_s1_ the target string into which the extracted substring is copied. The target string is null terminated.

_s2_ the string from which the substring is extracted.

_n1_ an integer that is the array index indicating the starting position of the substring in _s2_.

_n2_ an integer that is the array index indicating the position of the last character to be transferred to _s1_.

An empty string is one whose first character is the null character. If the source string, _s2_, is empty, the target string, _s1_, is set to empty and the value return is -1. The exception to the above is when _n1_ is zero and _n2_ is zero or larger. In this case the value returned is zero.

If the address pointed to by _s1_ is zero, the value returned is -1.

If _n1_ is larger than _n2_ or is negative or indexes a character past the end of the string, the target string is set empty and the value returned is -1.

The value of _n2_, however, may be any positive number. If _n2_ indexes a character past the end of the source string, the substring will terminate with the last character of the the source string.

The only time that zero is returned is when _n1_ indexes the null character of the source string. If _n1_ indexes the null character of the source string the target string is set empty but a zero is returned. For these cases _n2_ may be equal or greater than _n1_.

LIBRARY

/lib/lib3.a

SEE ALSO
psubstr(3L)

NAME

tellnice -- find nice level

SYNOPSIS

tellnice()

DESCRIPTION

tellnice returns the nice level of the current process.

LIBRARY

/lib/lib1.a

SEE ALSO

nice (_2_)

DIAGNOSTICS

BUGS

If the current nice level is negative and the effective user id is not the super user, the nice level is changed to zero. This results from the fact that _tellnice_ determines the nice level by setting the nice level to zero, noting the old value returned, and resetting the nice level.

NAME

termsg -- array of termination messages

SYNOPSIS

extern char ~~*~~*termsg[];

DESCRIPTION

```
source:
char *termsg[] {
          "Hangup",
          "Interrupted",
          "Quit",
          "Illegal Instruction",
          "Trace/BPT Trap",
          "IOT Trap",
          "EMT Trap",
          "Floating Point Exception",
          "Killed",
          "Bus Error",
          "Memory Fault",
          "Bad System Call",
          "Broken Pipe",
          "Alarm Timeout",
          "Software Kill",
          "16",
          "17",
          "Child Death",
          "Power Fail",
          "20",
          "21",
          "22",
          "23",
          "24",
          "25",
          "26",
          "27",
          "28",
          "29",
          "30",
          "31"
};
```

If a parent waits for the death of a child, a status is returned. The low byte of the status gives the reason why it terminated. This data structure contains common error messages to be used by the parent when reporting an error in termination.

LIBRARY

/lib/lib1.a

SEE ALSO

signal(2), wait(2)

NAME

timtoa - Convert time in seconds to an ASCII string.

SYNOPSIS

```
timtoa (tptr,tsec)
     char *tptr
     long *tsec
```

DESCRIPTION

This subroutine takes from _tsec_ a time in seconds such as returned by _time_(_2_) or _atotim_(_3L_) and converts it into a null terminated string of the format "mmddhhnnyy" which it stores in _tptr_.

```
mm - 2 digit month   01 to 12
dd - 2 digit day     01 to 31
hh - 2 digit hour    00 to 23
nn - 2 digit minutes 00 to 59
yy - 2 digit year.
```

Note that the seconds are truncated and not rounded to the nearest minute.

If successful a zero is returned, otherwise, -1 is returned.

LIBRARY

/lib/lib1.a

SEE ALSO

time(2),ctime(3),atotim(3),stdtime(3L)

DIAGNOSTICS

A -1 is returned if it encounters an error (when calling localtime).

NAME

??toa - machine format to ASCII conversion

SYNOPSIS

```
??toa(s1,n1)
char *s1;
int n1;
```

DESCRIPTION

??toa describes a family of 10 functions which convert binary numeric representations of a word or a double word to ASCII string format. The first five functions convert a word or integer to a string. The second five functions convert a double word or long to a string. The following is a list of the subroutine names:

btoa	- binary
dtoa	- signed decimal
otoa	- octal
utoa	- unsigned decimal
xtoa	- hexadecimal
lbtoa	- long binary
ldtoa	- long signed decimal
lotoa	- long octal
lutoa	- long unsigned decimal
lxtoa	- long hexadecimal

These functions return an integer indicating the length of the generated string s1 if no error occurred. If an error occurred, the value returned is zero. The value returned is the same as would be returned by the len function. The only cause for an error is the address zero for the string pointer s1.

s1 points to a buffer where the generated string will be stored. The buffer length is always assumed to be sufficient. The generated string is a null terminated string.

n1 an integer or long to be evaluated. Depending upon the function, the integer or long will be converted to an ASCII string.

The string generation conventions are minimum length strings except for the binary case in which leading zeros are preserved. In all conversions except binary leading zeros are deleted. For signed conversions, only the minus sign is generated. The terminating null character is placed immediately after the last numeric character. A zero numeric value will generate a string containing a single zero character.

The ranges of each of the conversion types are

btoa	- 16 zero's to 16 one's
dtoa	- -32768 to 32767
otoa	- 0 to 177777
utoa	- 0 to 65535

xtoa	-	0 to FFFF
lbtoa	-	32 zero's to 32 one's
ldtoa	-	-2147483648 to 2147483647
lotoa	-	0 to 37777777777
lutoa	-	0 to 4294967295
lxtoa	-	0 to FFFFFFFF

LIBRARY

/lib/lib3.a

SEE ALSO

ato??(3L)

NAME

trans -- translate characters

SYNOPSIS

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

DESCRIPTION

Trans returns an integer indicating the number of characters translated. If the value returned is -1 an illegal parameter was passed to the subroutine. *Trans* is a function which translates characters in string *s1* based on the contents of *s2*. String s2 consists of character pairs. If the first character of a character pair is found in string *s1*, that character is replaced with the second character of the character pair.

s1 the processed character string.

s2 a string of characters used as a pattern.

The pattern string, *s2*, is a null terminated string of characters whose content is character pairs. The length of *s2* as determined by *len* must be even. This function can be used to count the occurrence of a given character. For example, the pattern "AA" will count the number of capital A's in the string *s1*. If two character pairs have the same first character, the last character pair dominates. The pattern string "?Mississippi" is equivalent to "?Mssippi". Note that the pattern "?Mssippi" will change all i's to p's and all p's to i's in the source string. To capitalize the letters in a string one can use the 52 character string "aAbBcC...zZ" as a pattern string.

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 *s1* or *s2* is empty the value returned is zero.

The error code, -1, is returned if the address pointed to by *s1* is zero or if the length of *s2* is odd.

As the string *s1* is processed every character that is translated increments the translation count which is the value returned by the function.

LIBRARY

/lib/lib3.a

NAME

trnull -- replace a pattern char with a null

SYNOPSIS

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

DESCRIPTION

Trnull returns an integer indicating the number of matched characters found in the string _s1_ and translated to the null character.

_s1_ string which is to be modified by translation of matched characters, _c1_, to the null character.

_c1_ character if found in string _s1_ is translated to the null character.

_n1_ integer, maximum number of _c1_ to null translations to be performed.

The string _s1_ is defined as a null terminated array of characters. The value of the integer that is returned is the number of _c1_ characters found in _s1_ and replaced with a null. The maximum number of translations is determined by _n1_. The actual number of translations can vary from zero to _n1_ depending upon the number of _c1_ characters found before encountering the terminating null of the original string _s1_.

An empty string is one whose first character is the null character. If string _s1_ is empty or if _n1_ is zero or negative the value return is zero.

If the address pointed to by _s1_ is zero, the value returned will be -1.

If _c1_ is null, _trnull_ returns a zero.

LIBRARY

/lib/lib3.a

NAME

updacme -- modify ACME word and channel channel control file

SYNOPSIS

```
#include <acmestat.h>

updacme(flag, value, ofcname, chlname)
int flag, value;
char *ofcname, *chlname;
```

DESCRIPTION

This routine will be called by TRUMP (and NSCS) to modify the ACME word in the ACMESTAT maus area and the channel control file for the channel passed. The flag passed defines what bits in the ACME word are to be modified (use "DEFINE"s in /usr/include/acmestat.h). The value passed is the value retained by the modified bits. If a change is the ALERT bit in the FS file is required it will be made.

Arguments:

flag	- bits defined in acmestat.h
value	- new value defined in acmestat.h
ofcname	- pointer to office name string
chlname	- pointer to channel name string

Return Values:

0 if sucessful
-1 if system error
-2 if illegal flag

NOTE: The calling routine must lock the LN_RC_SEM semaphore. Also this routine uses the "e_" routines so messages are stored up (in case of system errors) and must be accessed via "e_output" or "e_wrapup".

FILES

/dev/maus/acmestat
/sccetc/fs
/office/<ofcname>/<chlname>

LIBRARY

/lib/lib1.a

NAME

updfs -- update FS file

SYNOPSIS

```
#include <chl.h>

updfs(value, chlhdr)
int value;
struct CHLHDR *chlhdr;
```

DESCRIPTION

This routine updates the FS file by setting the alerting bit to the value passed in _value_. This is done only if logging is active on the channel. _chlhdr_ is a pointer to the CHLHDR structure defined in /usr/include/chl.h.

Return values:

```
-1  if system error
 0  otherwise
```

FILES

/sccetc/fs

LIBRARY

/lib/lib1.a

NAME

updofc -- update default office name

SYNOPSIS

```
updofc(name)
char *name;
```

DESCRIPTION

Updofc changes the default office to the name given. This is done by calling *setdfprm*(*3*) to change or create a line in the .dftparm file in the current directory (usually a user directory) of the form

OFFICE=/office/*name*

Updofc allows *name* to take either of two forms:

officename

officename.chlname

In the latter case the .chlname is ignored.

DIAGNOSTICS

A -1 is returned if the *name* is too long, or there are troubles creating the .dfltparm file, in which case *errno* is set and an "e_" error message is stored that can be output by *e output*(*3*).

LIBRARY

/lib/lib1.a

BUGS

Does not check to insure that *name* corresponds to a valid office name.

SEE ALSO

lopen(3), getdfprm(3), e_output(3), e_syscall(3).

NAME

utoatnn -- convert unsigned integer to ASCII TNN

SYNOPSIS

```
utoatnn(str, value)
char str[];
unsigned value;
```

DESCRIPTION

This subroutine converts an unsigned integer into an ASCII representation of a trunk network number. A companion subroutine, atnntou, performs the conversion in the reverse direction.

Preconditions:
1. It is assumed that the string, str, is large enough to hold the resulting string.

Postconditions:
1. There are no failure modes. There is no return value.
2. The resulting null terminated string is exactly six chars long, excluding the null.
3. Two (2) implies that leading zeroes are not suppressed.

LIBRARY

/lib/lib1.a

SEE ALSO

atnntou(3L)

NAME

vqsort -- sorting algorithm

SYNOPSIS

```
vqsort(ptr, cnt, rec_size, p_comparison)
char *ptr;
int (**p_comparison)();
```

DESCRIPTION

Vqsort is an inplementation of the "quicker" sort algorithm. It allows a vector of comparison routines. When "ties" in the sort code are encountered, then the next comparison routine is called in order to resolve it unless a NULL is the next pointer.

Arguments

ptr -- is a pointer to an array of "records" to be sorted.

cnt -- is the number of records.

rec size -- is the size of each record in bytes.

p comparison -- is the address of an array of comparison routines. It sets the following externals:

v size -- the size of the records being sorted.

v comp -- a pointer to the first comparison routine.

v cvec -- a pointer to the vector of comparison routines.

v cc -- the return value of the comparison routine.

This subroutine returns no useful value.

LIBRARY

/lib/lib1.a

SEE ALSO

fixedsort(3L), qsort(3C)

NAME

writint -- buffered output for files containing integer data

SYNOPSIS

```
#include <rwint.h>

writint(func,recptr,outbuf)
int func;
int *recptr;
struct IOBUF *outbuf;
```

DESCRIPTION

This subroutine provides buffered output capability for files containing integer data in records whose size is a power of two. It returns a 1 if the task is completed successfully or a negative value if an error is detected.

The argument, func , must contain one of the following values:

-1 when the calling program has finished writing data to an output file. It causes a partially filled output buffer, if one exists, to be written to the output file.

0 when the calling program is writing data to an output file.

The argument, recptr , is the address of the record that is to be written to the output file.

Outbuf is the address of a 522(10) byte buffer area whose format is:

```
struct IOBUF
  {  int fildes;
     int errval;
     int idata;
     int recsize;
     int nread;
     int intbuf[IBUFSIZE];
  };
```

where fildes	is the file descriptor of an open output file.
errval	is not used by this subroutine.
idata	is the current number of records that has been written into the buffer by the calling program. The calling program initializes this variable by setting it equal to 0.

recsize	is the record size in words. The record size must be a power of two; ie. 2, 4, 8, 16, etc., words.
nread	is not used by this routine.
intbuf	is the output buffer.
IBUFSIZE	contains the value, 256.

The calling program must initialize the following structure variables for each output file that is to be written. These variables must be initialized prior to the first call to this subroutine to write to the appropriate output file.

```
  <structure>.fildes= <file descriptor of output file>;
  <structure>.idata= 0;  indicates that buffer is empty
  <structure>.recsize= <record size in words>;
```

Once the calling program has finished writing data to an output file, it must call this subroutine, as shown below, so that a partially filled output buffer, if one exists, will be written to the output file. This call should be made as follows:

```
  writint(-1,&<previously written record>,&<output buffer>);
```

Note that _func_ has the value, -1, which forces a partially filled output buffer, if one exists, to be written to the output file.

FILES

/usr/include/rwint.h which contains the definitions for IOBUF and IBUFSIZE.

LIBRARY

/lib/lib1.a

SEE ALSO

readint(3)

DIAGNOSTICS

The error codes returned by this subroutine, in r0, are:

-1 I/O error.

BUGS