NAME

abrtim - Get abbreviated time

SYNOPSIS

char *abrtim();

DESCRIPTION

Get abbreviated time (minutes past the hour).

SEE ALSO

ctime(3)

LIBRARY

/lib/lib1.a

NAME

addchl -- add a logging channel

SYNOPSIS

```
#include <fs.h>

addchl(mln, fsp, muxfd)
struct FS_CB *fsp;
```

LIBRARY

/lib/lib1.a

NAME

addgrp -- add group name to the group file

SYNOPSIS

addgrp(group,gid)
char *group; int gid;

DESCRIPTION

addgrp adds a group to the /etc/group file based on the following arguments:

group	a string pointer to the group entry name to be added to the /etc/group file.
gid	An integer representing the group id of the entry to be added. If 0, the next value greater than 10 is added.

Addgrp will return the following values in retcode:

```
retcode > 0    the group id added.
retcode = -1   group already exists (not added)
retcode = -2   System trouble (cannot open group file).
retcode = -3   Illegal group name (colon in name).
retcode = -4   Group id added is too large (max of 127).
```

Addgrp performs only one search of the group file looking for the given group. Also, if gid == 0, then each gid that is encountered is recorded in bitstr. Bitstr is a 128 bit string; each bit, if one, represents an assigned gid in the group file and if zero, an unassigned gid in group file.

LIBRARY

/lib/lib1.a

FILES

/etc/group

NAME

addusr -- add user to passwd file

SYNOPSIS

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

DESCRIPTION

addusr adds a user to the /etc/passwd file based on the following arguments:

usr	astring pointer to the usr entry name to be added to the /etc/passwd file.
uid	an integer containing the usr id to be added. If 0, the next uid > 10 is added.
gid	an integer containing the group id to be added. If -1, then LOCUSR_GID gid is added.
wdir	a pointer to a string containing the working directory to be added. If null, none is added.
shell	a string pointer to the shell to be added If null, no shell is added.
pwd	a pointer to a string containing an encrypted password to be added. If null, no password is added.

addusr will return the following values in retcode:

retcode > 0	the user id added.
retcode = -1	User already exists (not added)
retcode = -2	System trouble (cannot open passwd file).
retcode = -3	Illegal user name(colon in name).
retcode = -4	Uid number is too large (127 is max).

addusr performs only one search of the passwd file looking for the given user. Also, if uid == 0, then each uid that is encountered is recorded in bitstr. Bitstr is a 128 bit string; each bit, if one, represents an assigned uid in the passwd file and if zero, an unassigned uid in passwd file.

LIBRARY

/lib/lib1.a

FILES

/etc/passwd

NAME

add_to_env -- add a parameter to the environment

SYNOPSIS

```
add_to_env(new_parameter)

char *new_parameter;
```

DESCRIPTION

Add_to_env adds the specified environment parameter to the current environment by allocating new space and rewriting the environment pointer table, adding the new parameter pointer supplied and ensuring that the last entry in the table is NULL. 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 to this routine is a pointer to a string which, by convention, is of the form: <name>=<value> where <name> is all upper case and identifies the environment parameter and <value> is the value of that parameter.

USER BEWARE: the form of the environment parameter string is not enforced by this routine, but must be of the specified form and be stored in a protected, global (i.e., non-volatile) data area to have the desired effect.

LIBRARY

/lib/lib1.a

SEE ALSO

ret_env(3L), exec(2), environ(7)

DIAGNOSTICS

Add_to_env returns a 0 if the environment is successfully redefined with the additional parameter and a -1 otherwise with no changes to the current environment.

NAME

any -- match character against string

SYNOPSIS

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

DESCRIPTION

any 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 any character 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 returns a -1.

If *s1* is empty, the function returns a -1.

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

LIBRARY

/lib/lib3.a

NAME

atnntou -- convert ASCII TNN to unsigned integer

SYNOPSIS

```
atnntou(tnn, value)
char *tnn;
unsigned *value;
```

DESCRIPTION

The subroutine converts an ASCII string representation of a trunk network number, tnn, to an unsigned integer. Note that the string is neither a decimal or an octal number. The largest tnn string is '317377' which results in the largest 16 bit unsigned integer. A companion subroutine, utoatnn(), will convert an unsigned integer to a tnn ASCII string.

Preconditions:
1. No assumptions are made regarding the length or the contents of the string. However, a length greater than 6 will fail. Also, a string containing other than digits will fail.
2. No assumption is made regarding the contents of value.

Post conditions:
1. Return of 0 implies no errors detected. The contents of value are correct.
2. Return -1,1,2,3,4,5,6, or 7 depending on the error detected.
3. A return of nonzero implies the contents of value are undefined.
4. The contents of the original string are unchanged.

LIBRARY

/lib/lib1.a

SEE ALSO

utoatnn(3L)

NAME

ato?? -- ASCII to machine format conversion

SYNOPSIS

```
ato??(s1,v1)
char *s1;
int *v1;

/*
 'v' may be int or long depending on whether the function
 converts to int or long
*/
```

DESCRIPTION

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

*ato***b**	- binary
*ato***d**	- signed decimal
*ato***o**	- octal
*ato***u**	- unsigned decimal
*ato***x**	- hexadecimal
*ato***lb**	- long binary
*ato***ld**	- long signed decimal
*ato***lo**	- long octal
*ato***lu**	- long unsigned decimal
*ato***lx**	- long hexadecimal

These functions return an integer indicating the length of the 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.

s1 is the string to be evaluated. This string contains the ASCII representation of the number for the conversion.

v1 is a pointer to an integer variable or a pointer to a long integer variable depending upon whether the function converts to integer or long. If an error occurs, the data pointed to by *v1* is always set to zero. If the pointer *v1* is zero, the data pointed to by *v1* is not modified and the function returns a zero.

The input convention for the string *s1* for the signed decimal functions is as follows:

[<blanks>][<sign>][<leading zeros>]<numeric characters><null>
where <sign> ::= + | -

The input convention for the string *s1* for unsigned functions is as follows:

[<blanks>][<leading zeros>]<numeric characters><null>

There are six reasons for the error condition.

(A) s1 points to an empty string.

(B) s1 points to a string containing only blanks.

(C) s1 points to a string which contains characters out of the range required by the conversion. For example, atob will accept only zeros and ones. The legal characters for hexadecimal conversions are "012345789ABCDEF".

(D) s1 points to a string whose format is illegal. For example, the string contains trailing blanks.

(E) v1, the pointer to the word or double word, has the address zero.

(F) s1 points to a string whose numeric value exceeds that of the conversion type.

The ranges of each of the conversion types is listed

ato**b**	-	0 to 1111111111111111
ato**d**	-	-32768 to 32767
ato**o**	-	0 to 177777
ato**u**	-	0 to 65535
ato**x**	-	0 to FFFF
ato**lb**	-	0 to a string of 32 one's
ato**ld**	-	-2147483648 to 2147483647
ato**lo**	-	0 to 37777777777
ato**lu**	-	0 to 4294967295
ato**lx**	-	0 to FFFFFFFF

LIBRARY
/lib/lib3.a

SEE ALSO
??toa(3L) (listed alphabetically as toa(3L))