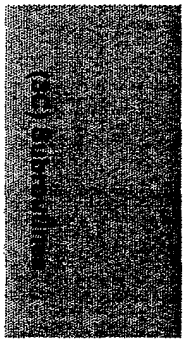
DYNIX/ptx®
Reference Manual
Section 3

1003-49233-05



NAME

dial – establish an out-going terminal line connection

SYNOPSIS

#include <dial.h>

int dial (call)
CALL call;

void undial (fd)
int fd;

DESCRIPTION

dial returns a file-descriptor for a terminal line open for read/write. The argument to **dial** is a **CALL** structure (defined in the **<dial.h>** header file).

When finished with the terminal line, the calling program must invoke **undial** to release the semaphore that has been set during the allocation of the terminal device.

The definition of **CALL** in the **<dial.h>** header file is:

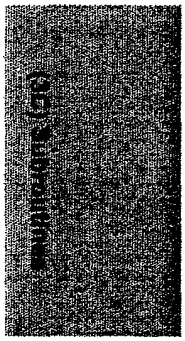
```
typedef struct {
  struct termio *attr;/* pointer to termio attribute struct */
  int   baud;     /* transmission data rate */
  int   speed;    /* 212A modem: low=300, high=1200 */
  char  *line;    /* device name for out-going line */
  char  *telno;   /* pointer to tel-no digits string */
  int   modem;    /* specify modem control for direct lines */
  char  *device;  /* unused */
  int   dev_len;  /* unused */
} CALL;
```

The **CALL** element **speed** is intended only for use with an outgoing dialed call. The value should match the modem speed setting. The typical values are: 110, 300, 600, 1200, 2400, 4800, 9600, and 19200. The **CALL** element **baud** is for the desired transmission baud rate. For example, one might set **baud** to 110 and **speed** to 300 (or 1200). However, if **speed** is set to 1200, **baud** must be set to high (1200).

If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the **line** element in the **CALL** structure. Legal values for such terminal device names are kept in the *Devices* file. In this case, the value of the **baud** element should be set to -1. This will cause **dial** to determine the correct value from the *Devices* file.

The **telno** element is for a pointer to a character string representing the telephone number to be dialed. Such numbers may consist only of these characters:

0-9	dial 0-9
*	dial *
#	dial #
=	wait for secondary dial tone
–	delay for approximately 4 seconds



NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent – get group file entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent ( )

struct group *getgrgid (gid)
gid_t gid;

struct group *getgrnam (name)
char *name;

void setgrent ( )

void endgrent ( )

struct group *fgetgrent (f)
FILE *f;
```

DESCRIPTION

getgrent, getgrgid and **getgrnam** each return pointers to an object with the following structure containing the broken-out fields of a line in the */etc/group* file. Each line contains a "group" structure, defined in the **<grp.h>** header file.

```
struct group {
        char   *gr_name;  /* the name of the group */
        char   *gr_passwd;/* the encrypted group password */
        gid_t  gr_gid;    /* the numerical group ID */
        char   **gr_mem;  /* vector of pointers to member names */
};
```

getgrent when first called returns a pointer to the first group structure in the file; thereafter, it returns a pointer to the next group structure in the file; so, successive calls may be used to search the entire file. **getgrgid** searches from the beginning of the file until a numerical group id matching **gid** is found and returns a pointer to the particular structure in which it was found. **getgrnam** searches from the beginning of the file until a group name matching **name** is found and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a NULL pointer.

A call to **setgrent** has the effect of rewinding the group file to allow repeated searches. **endgrent** may be called to close the group file when processing is complete.

fgetgrent returns a pointer to the next group structure in the stream **f**, which matches the format of */etc/group*.

FILES

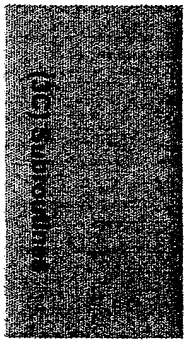
/etc/group

SEE ALSO

getlogin(3C), getpwent(3C), group(4)

DIAGNOSTICS

A NULL pointer is returned on EOF or error.



NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent – get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwent ( )

struct passwd *getpwuid (uid)
uid_t uid;

struct passwd *getpwnam (name)
char *name;

void setpwent ( )

void endpwent ( )

struct passwd *fgetpwent (f)
FILE *f;

DESCRIPTION

getpwent, getpwuid and **getpwnam** each returns a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/passwd* file. Each line in the file contains a "passwd" structure, declared in the **<pwd.h>** header file:

```
struct passwd {
        char  *pw_name;
        char  *pw_passwd;
        uid_t pw_uid;
        gid_t pw_gid;
        char  *pw_age;
        char  *pw_comment;
        char  *pw_gecos;
        char  *pw_dir;
        char  *pw_shell;
};
```

This structure is declared in **<pwd.h>** so it is not necessary to redeclare it.

The fields have meanings described in **passwd**(4).

getpwent when first called returns a pointer to the first passwd structure in the file; thereafter, it returns a pointer to the next passwd structure in the file; so successive calls can be used to search the entire file. **getpwuid** searches from the beginning of the file until a numerical user id matching **uid** is found and returns a pointer to the particular structure in which it was found. **getpwnam** searches from the beginning of the file until a login name matching **name** is found, and returns a pointer to the particular structure in which it was found. If an end-of-file or an error is encountered on reading, these functions return a **NULL** pointer.

A call to **setpwent** has the effect of rewinding the password file to allow repeated searches. **endpwent** may be called to close the password file when processing is complete.

NAME

getwidth – get information of supplementary code sets

SYNOPSIS

```
#include <euc.h>
#include <getwidth.h>

void getwidth(ptr)
eucwidth_t *ptr;
```

DESCRIPTION

The **getwidth** function reads the *character class table*, which is generated by **chrtbl**(1M) or **wchrtbl**(1M), to get information about supplementary code sets, and sets it into the structure **eucwidth_t**.

The structure **eucwidth_t** is defined in *euc.h* as follows:

```
typedef struct {
        short int _eucw1,_eucw2,_eucw3;
        short int _scrw1,_scrw2,_scrw3;
        short int _pcw;
        char   _multibyte;
} eucwidth_t;
```

The number of bytes used to represent a character from code set 1, code set 2, or code set 3 is stored in `_eucw1`, `_eucw2`, and `_eucw3`, respectively. The values returned in `_eucw2` and `_eucw3` do not include the single shift character used to distinguish code sets 2 or 3. So the total number of bytes needed to store a multibyte character from code sets 2 or 3 would actually be `_eucw2 + 1` or `_eucw3 + 1`. A value of 0 indicates that the supplementary code set is not defined.

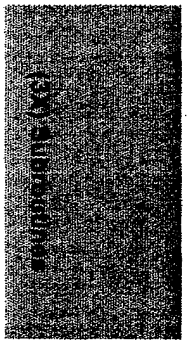
The number of columns that a character from a secondary code set occupies on the screen is stored in `_scrw1`, `_scrw2`, and `_scrw3`. A value of 0 indicates that the supplementary code set is not defined.

The number of bytes used to represent a process code (wide character) for the current locale is stored in `_pcw`. A value of 2 indicates that the sixteen bit process code format is being used for the current codeset. A value of 4 indicates that the thirty two bit process code format is being used for the the current codeset.

The value of `_multibyte` indicates whether multibyte characters are defined for the current locale. A value of 0 indicates that multibyte code sets are not defined. A value of 1 indicates that multibyte code sets are defined.

SEE ALSO

chrtbl(1M), wchrtbl(1M)



NAME

getaaent, getaanam, setaaent, endaaent, fgetaaent – get audit mask alias file entry

SYNOPSIS

```
#include <aualias.h>

struct aualias *getaaent ( )

struct aualias *getaanam (name)
char *name;

void setaaent ( )

void endaaent ( )

struct aualias *fgetaaent (fp)
FILE *fp;
```

DESCRIPTION

The **getaaent** and **getaanam** routines each return a pointer to an object with the following structure containing the broken-out fields of an entry in the */etc/audit_aliases* file. Each entry in the file contains an audit alias structure, declared in the **< aualias.h>** header file (see **audit_aliases(4)**):

```
struct aualias{
        char  *aa_name;
        char  *aa_list;
};
```

When first called, **getaaent** returns a pointer to the first *aualias* structure in the file; thereafter, it returns a pointer to the next *aualias* structure in the file so successive calls can be used to search the entire file. The **getaanam** routine searches from the beginning of the file until an alias name matching *name* is found, and returns a pointer to the particular structure in which it was found.

getaaent, **fgetaaent**, **setaaent**, and **endaaent** all use the same stream which is separate from that used by **getaanam**. As a result, **getaanam** may be called without interfering with the stream pointer used by calls to **getaaent**.

If an end-of-file or an error is encountered on reading, or there is a format error in the file, **getaaent** and **getaanam** return a **NULL** pointer. If there is a format error, **errno** is set to **EINVAL**.

A call to **setaaent** has the effect of rewinding the audit mask alias file to allow repeated searches. The **endaaent** routine may be called to close the audit alias file when processing is complete. **setaaent** and **endaaent** operate on the stream used by **getaaent**. They will have no effect on the stream used by **getaanam**.

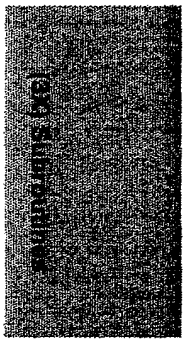
The **fgetaaent** routine returns a pointer to the next *aualias* structure in the stream *fp*, which matches the format of */etc/audit_aliases*.

FILES

/etc/audit_aliases

SEE ALSO

aliasmgmt(1M), putaaent(3X), audit_aliases(4)



NAME

getacent, getaccid, getacnam, setacent, endacent, fgetacent, setacfile, endacfile – get audit class entry

SYNOPSIS

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

FILE *AuClass_Fp

struct  auclass  *getacent()

struct  auclass  *getaccid(cid)
int  cid;

struct  auclass  *getacnam(name)
char  *name;

void  setacent()

void  endacent()

struct  auclass  *fgetacent(f)
FILE  *f;

void setacfile(file)

void endacfile()
```

DESCRIPTION

getacent, **getaccid**, and **getacnam** each return a pointer to an object with the **auclass** structure containing the fields of a line in the */etc/auclass* file. Each line in the file corresponds to an **auclass** structure, declared in the *<auclass.h>* header file:

```
struct  auclass
{
     int   ac_cid;
     char  *ac_name;
     char  *ac_function;
     char  *ac_desc;
};
```

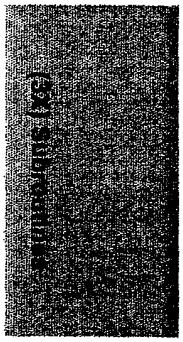
The fields have the meanings defined in **auclass**(4).

getacent, when first called, returns a pointer to the first **auclass** structure in the file. Thereafter, it returns a pointer to the next **auclass** structure in the file, so successive calls can be used to search the entire file.

getaccid searches from the beginning of the file until a numerical audit class ID matching **cid** is found and returns a pointer to the particular structure in which it was found.

getacnam searches from the beginning of the file until an audit class name matching **name** is found and returns a pointer to the particular structure in which it was found.

If an end-of-file or an error is encountered on reading, the three preceding functions return a NULL pointer.



NAME

getaeent, getaecid, getaenam, setaeent, endaeent, fgetaeent, setaefile, endaefile – get audit event type entry

SYNOPSIS

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

FILE *AuEvent_Fp;

struct auevent *getaeent()

struct auevent *getaecid(cid)
int cid;

struct auevent *getaenam(name)
char *name;

void setaeent()

void endaeent()

struct auevent *fgetaeent(f)
FILE *f;

void setaefile(file)

void endaefile()
```

DESCRIPTION

getaeent, **getaecid**, and **getaenam** each return a pointer to an object with the **auevent** structure containing the fields of a line in the */etc/auevent* file. Each line in the file corresponds to an **euevent** structure, declared in the *<auevent.h>* header file:

```
struct  auevent
{
        int     ae_cid;
        char    *ae_name;
        char    *ae_function;
        char    *ae_desc;
};
```

The fields have the meanings defined in **auevent(4)**.

getaeent, when first called, returns a pointer to the first **auevent** structure in the file. Thereafter, it returns a pointer to the next **auevent** structure in the file, so successive calls can be used to search the entire file.

getaecid searches from the beginning of the file until a numerical audit event type ID matching *cid* is found and returns a pointer to the particular structure in which it was found.

getaenam searches from the beginning of the file until an audit event name matching *name* is found and returns a pointer to the particular structure in which it was found.

If an end-of-file or an error is encountered on reading, the three preceding functions return a NULL pointer.

If the audit event file is already open, a call to **setaeent** has the effect of rewinding the audit event file to allow repeated searches. Otherwise,

NAME

getspent, getspnam, setspent, endspent, fgetspent, lckpwdf, ulckpwdf – get shadow password file entry

SYNOPSIS

#include <shadow.h>

struct spwd *getspent ( )

struct spwd *getspnam (name)
char *name;

void setspent ( )

void endspent ( )

struct spwd *fgetspent (fp)
FILE *fp;

int lckpwdf ( )

int ulckpwdf ( )

DESCRIPTION

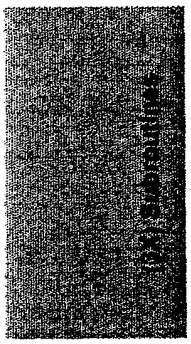
The **getspent** and **getspnam** routines each return a pointer to an object with the following structure containing the broken-out fields of a line in the */etc/shadow* file. Each line in the file contains a shadow password structure, declared in the **< shadow.h>** header file (see **shadow(4)**):

```
struct spwd{
        char  *sp_namp;
        char  *sp_pwdp;
        long  sp_lstchg;
        long  sp_min;
        long  sp_max;
        char  *sp_amask_alias;
        ushort  sp_anon;
};
```

When first called, **getspent** returns a pointer to the first *spwd* structure in the file; thereafter, it returns a pointer to the next *spwd* structure in the file so successive calls can be used to search the entire file. The **getspnam** routine searches from the beginning of the file until a login name matching *name* is found, and returns a pointer to the particular structure in which it was found.

The **getspent** and **getspnam** routines populate the **sp_min**, **sp_max**, **sp_lstchg**, **sp_amask_alias**, or **sp_anon** field with a default value if the corresponding field in */etc/shadow* is empty. For **sp_min**, **sp_max**, and **sp_lstchg** the default value is –1. For **sp_amask_alias**, **getspent** and **getspnam** will retrieve the value of **user_shadow_amask** from */etc/site_secp*. If **user_shadow_amask** is not defined in */etc/site_secp*, **sp_amask_alias** is set to **NULL**. For **sp_anon** the default value is **AU_ANON_OFF** (defined in **<shadow.h>**).

If an end-of-file or an error is encountered on reading, or there is a format error in the file, **getspent** and **getspnam** return a **NULL** pointer. If there is a format error, **errno** is set to **EINVAL**.



M_GRAIN Set **grain** to **value**. The sizes of all blocks smaller than **maxfast** are considered to be rounded up to the nearest multiple of **grain**. **grain** must be greater than 0. The default value of **grain** is the smallest number of bytes which will allow alignment of any data type. Value will be rounded up to a multiple of the default when **grain** is set.

M_KEEP Preserve data in a freed block until the next **malloc**, **realloc**, or **calloc**. This option is provided only for compatibility with the old version of **malloc** and is not recommended.

These values are defined in the *<malloc.h>* header file.

mallopt may be called repeatedly, but may not be called after the first small block is allocated.

mallinfo provides instrumentation describing space usage. It returns the structure:

```
struct mallinfo  {
        int arena;          /* total space in arena */
        int ordblks;        /* number of ordinary blocks */
        int smblks;         /* number of small blocks */
        int hblkhd;         /* space in holding block headers */
        int hblks;          /* number of holding blocks */
        int usmblks;        /* space in small blocks in use */
        int fsmblks;        /* space in free small blocks */
        int uordblks;       /* space in ordinary blocks in use */
        int fordblks;       /* space in free ordinary blocks */
        int keepcost;       /* space penalty if keep option */
                            /* is used */
}
```

This structure is defined in the *<malloc.h>* header file.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

SEE ALSO

brk(2), malloc(3C)

DIAGNOSTICS

If the process has used all of the memory that it is allowed to obtain, then **malloc**, **realloc**, and **calloc** return a NULL pointer and set errno to **[ENOMEM]**. If no memory is available, but the condition is temporary due to a shortage of system resources (for example, lack of swap space, etc.) and another call to **malloc**, **realloc**, or **calloc** might succeed, then **[EAGAIN]** is returned.

When **realloc** returns NULL, the block pointed to by **ptr** is left intact. If **mallopt** is called after any allocation or if **cmd** or **value** are invalid, non-zero is returned. Otherwise, it returns zero. **malloc(NULL)** returns NULL.