

**NAME**

gettent, gettuid, gettline, puttline, settent, endtnt, utmpname – access utmp file entries

**SYNOPSIS**

```
#include <utmp.h>

struct utmp *gettent(void);
struct utmp *gettuid(const struct utmp *ut);
struct utmp *gettline(const struct utmp *ut);

struct utmp *puttline(const struct utmp *ut);

void settent(void);
void endtnt(void);

int utmpname(const char *file);
```

**DESCRIPTION**

New applications should use the POSIX.1-specified "utmpx" versions of these functions; see CONFORMING TO.

**utmpname()** sets the name of the utmp-format file for the other utmp functions to access. If **utmpname()** is not used to set the filename before the other functions are used, they assume **\_PATH\_UTMP**, as defined in *<paths.h>*.

**settent()** rewinds the file pointer to the beginning of the utmp file. It is generally a good idea to call it before any of the other functions.

**endtnt()** closes the utmp file. It should be called when the user code is done accessing the file with the other functions.

**gettent()** reads a line from the current file position in the utmp file. It returns a pointer to a structure containing the fields of the line. The definition of this structure is shown in **utmp(5)**.

**gettuid()** searches forward from the current file position in the utmp file based upon *ut*. If *ut->ut\_type* is one of **RUN\_LVL**, **BOOT\_TIME**, **NEW\_TIME**, or **OLD\_TIME**, **gettuid()** will find the first entry whose *ut\_type* field matches *ut->ut\_type*. If *ut->ut\_type* is one of **INIT\_PROCESS**, **LOGIN\_PROCESS**, **USER\_PROCESS**, or **DEAD\_PROCESS**, **gettuid()** will find the first entry whose *ut\_id* field matches *ut->ut\_id*.

**gettline()** searches forward from the current file position in the utmp file. It scans entries whose *ut\_type* is **USER\_PROCESS** or **LOGIN\_PROCESS** and returns the first one whose *ut\_line* field matches *ut->ut\_line*.

**puttline()** writes the *utmp* structure *ut* into the utmp file. It uses **gettuid()** to search for the proper place in the file to insert the new entry. If it cannot find an appropriate slot for *ut*, **puttline()** will append the new entry to the end of the file.

**RETURN VALUE**

**gettent()**, **gettuid()**, and **gettline()** return a pointer to a *struct utmp* on success, and NULL on failure (which includes the "record not found" case). This *struct utmp* is allocated in static storage, and may be overwritten by subsequent calls.

On success **puttline()** returns *ut*; on failure, it returns NULL.

**utmpname()** returns 0 if the new name was successfully stored, or -1 on failure.

In the event of an error, these functions *errno* set to indicate the cause.

**ERRORS****ENOMEM**

Out of memory.

**ESRCH**

Record not found.

**setutent()**, **pututline()**, and the **getut\*()** functions can also fail for the reasons described in **open(2)**.

## FILES

*/var/run/utmp*

database of currently logged-in users

*/var/log/wtmp*

database of past user logins

## ATTRIBUTES

For an explanation of the terms used in this section, see **attributes(7)**.

Interface	Attribute	Value
<b>getutent()</b>	Thread safety	MT-Unsafe init race:utent race:utentbuf sig:ALRM timer
<b>getutid()</b> , <b>getutline()</b>	Thread safety	MT-Unsafe init race:utent sig:ALRM timer
<b>pututline()</b>	Thread safety	MT-Unsafe race:utent sig:ALRM timer
<b>setutent()</b> , <b>endutent()</b> , <b>utmpname()</b>	Thread safety	MT-Unsafe race:utent

In the above table, *utent* in *race:utent* signifies that if any of the functions **setutent()**, **getutent()**, **getutid()**, **getutline()**, **pututline()**, **utmpname()**, or **endutent()** are used in parallel in different threads of a program, then data races could occur.

## CONFORMING TO

XPG2, SVr4.

In XPG2 and SVID 2 the function **pututline()** is documented to return void, and that is what it does on many systems (AIX, HP-UX). HP-UX introduces a new function **\_pututline()** with the prototype given above for **pututline()**.

All these functions are obsolete now on non-Linux systems. POSIX.1-2001 and POSIX.1-2008, following SUSv1, does not have any of these functions, but instead uses

```
#include <utmpx.h>

struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *);
struct utmpx *getutxline(const struct utmpx *);
struct utmpx *pututxline(const struct utmpx *);
void setutxent(void);
void endutxent(void);
```

These functions are provided by glibc, and perform the same task as their equivalents without the "x", but use *struct utmpx*, defined on Linux to be the same as *struct utmp*. For completeness, glibc also provides **utmpxname()**, although this function is not specified by POSIX.1.

On some other systems, the *utmpx* structure is a superset of the *utmp* structure, with additional fields, and larger versions of the existing fields, and parallel files are maintained, often */var/\*/utmpx* and */var/\*/wtmpx*.

Linux glibc on the other hand does not use a parallel *utmpx* file since its *utmp* structure is already large enough. The "x" functions listed above are just aliases for their counterparts without the "x" (e.g., **getutxent()** is an alias for **getutent()**).

## NOTES

**Glibc notes**

The above functions are not thread-safe. Glibc adds reentrant versions

```
#include <utmp.h>

int getutent_r(struct utmp *ubuf, struct utmp **ubufp);
int getutid_r(struct utmp *ut,
              struct utmp *ubuf, struct utmp **ubufp);
int getutline_r(struct utmp *ut,
               struct utmp *ubuf, struct utmp **ubufp);
```

Feature Test Macro Requirements for glibc (see **feature\_test\_macros(7)**):

**getutent\_r(), getutid\_r(), getutline\_r():**

```
_GNU_SOURCE
/* since glibc 2.19: */ _DEFAULT_SOURCE
/* glibc <= 2.19: */ _SVID_SOURCE || _BSD_SOURCE
```

These functions are GNU extensions, analogs of the functions of the same name without the `_r` suffix. The `ubuf` argument gives these functions a place to store their result. On success, they return 0, and a pointer to the result is written in `*ubufp`. On error, these functions return -1. There are no `utmpx` equivalents of the above functions. (POSIX.1 does not specify such functions.)

**EXAMPLE**

The following example adds and removes a `utmp` record, assuming it is run from within a pseudo terminal. For usage in a real application, you should check the return values of **getpwuid(3)** and **ttyname(3)**.

```
#include <string.h>
#include <stdlib.h>
#include <pwd.h>
#include <unistd.h>
#include <utmp.h>
#include <time.h>

int
main(int argc, char *argv[])
{
    struct utmp entry;

    system("echo before adding entry:;who");

    entry.ut_type = USER_PROCESS;
    entry.ut_pid = getpid();
    strcpy(entry.ut_line, ttyname(STDIN_FILENO) + strlen("/dev/"));
    /* only correct for ptys named /dev/tty[pqr][0-9a-z] */
    strcpy(entry.ut_id, ttyname(STDIN_FILENO) + strlen("/dev/tty"));
    time(&entry.ut_time);
    strcpy(entry.ut_user, getpwuid(getuid())->pw_name);
    memset(entry.ut_host, 0, UT_HOSTSIZE);
    entry.ut_addr = 0;
    setutent();
    pututline(&entry);

    system("echo after adding entry:;who");

    entry.ut_type = DEAD_PROCESS;
    memset(entry.ut_line, 0, UT_LINESIZE);
    entry.ut_time = 0;
```

```
memset(entry.ut_user, 0, UT_NAMESIZE);
setutent();
pututline(&entry);

system("echo after removing entry:;who");

endutent();
exit(EXIT_SUCCESS);
}
```

**SEE ALSO**

**getutmp(3), utmp(5)**

**COLOPHON**

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.