



## ***Linux Ubuntu 26.04 Manual Pages on command 'futimens.3'***

### ***\$ man futimens.3***

utimensat(2)                      System Calls Manual                      utimensat(2)

#### NAME

utimensat, futimens - change file timestamps with nanosecond precision

#### LIBRARY

Standard C library (libc, -lc)

#### SYNOPSIS

```
#include <fcntl.h>                /* Definition of AT_* constants */
#include <sys/stat.h>

int utimensat(int dirfd, const char *path,
               const struct timespec times[_Nullable 2], int flags);

int futimens(int fd, const struct timespec times[_Nullable 2]);
```

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

utimensat():

Since glibc 2.10:

    \_POSIX\_C\_SOURCE >= 200809L

Before glibc 2.10:

    \_ATFILE\_SOURCE

futimens():

Since glibc 2.10:

    \_POSIX\_C\_SOURCE >= 200809L

Before glibc 2.10:

    \_GNU\_SOURCE

#### DESCRIPTION

utimensat() and futimens() update the timestamps of a file with nanosecond precision. This contrasts with the historical utime(2) and utimes(2), which permit only second and microsecond precision, respectively, when setting file timestamps.

With utimensat() the file is specified via the pathname given in path. With futimens() the file whose timestamps are to be updated is specified via an open file descriptor, fd.

For both calls, the new file timestamps are specified in the array times: times[0] specifies the new "last access time" (atime); times[1] specifies the new "last modification time" (mtime). Each of the elements of times specifies a time as the number of seconds and nanoseconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC). This information is conveyed in a timespec(3) structure.

Updated file timestamps are set to the greatest value supported by the filesystem that is not greater than the specified time.

If the tv\_nsec field of one of the timespec structures has the special value UTIME\_NOW, then the corresponding file timestamp is set to the current time. If the tv\_nsec field of one of the timespec structures has the special value UTIME\_OMIT, then the corresponding file timestamp is left unchanged. In both of these cases, the value of the corresponding tv\_sec field is ignored.

If times is NULL, then both timestamps are set to the current time.

The status change time (ctime) will be set to the current time, even if the other timestamps don't actually change.

#### Permissions requirements

To set both file timestamps to the current time (i.e., times is NULL, or both tv\_nsec fields specify UTIME\_NOW), either:

- ? the caller must have write access to the file;
- ? the caller's effective user ID must match the owner of the file; or
- ? the caller must have appropriate privileges.

To make any change other than setting both timestamps to the current time (i.e., times is not NULL, and neither tv\_nsec field is UTIME\_NOW and neither tv\_nsec field is UTIME\_OMIT), either condition 2 or 3 above must apply.

If both tv\_nsec fields are specified as UTIME\_OMIT, then no file ownership or permission checks are performed, and the file timestamps are not modified, but other error conditions may still be detected.

If path is relative, then by default it is interpreted relative to the directory referred to by the open file descriptor, dirfd (rather than relative to the current working directory of the calling process, as is done by utimes(2) for a relative pathname). See openat(2) for an explanation of why this can be useful.

If path is relative and dirfd is the special value AT\_FDCWD, then path is interpreted relative to the current working directory of the calling process (like utimes(2)).

If path is absolute, then dirfd is ignored.

The flags argument is a bit mask created by ORing together zero or more of the following values defined in <fcntl.h>:

AT\_EMPTY\_PATH (since Linux 5.8)

If path is an empty string, operate on the file referred to by dirfd (which may have been obtained using the open(2) O\_PATH flag). In this case, dirfd can refer to any type of file, not just a directory. If dirfd is AT\_FDCWD, the call operates on the current working directory. This flag is Linux-specific; define \_GNU\_SOURCE to obtain its definition.

AT\_SYMLINK\_NOFOLLOW

If path specifies a symbolic link, then update the timestamps of the link, rather than the file to which it refers.

## RETURN VALUE

On success, utimensat() and futimens() return 0. On error, -1 is returned and errno is set to indicate the error.

## ERRORS

EACCES times is NULL, or both tv\_nsec values are UTIME\_NOW, and the effective user ID of the caller does not match the owner of the file, the caller does not have write access to the file, and the caller is not privileged (Linux: does not have either the CAP\_FOWNER or the CAP\_DAC\_OVERRIDE capability).

EBADF (futimens()) fd is not a valid file descriptor.

EBADF (utimensat()) path is relative but dirfd is neither AT\_FDCWD nor a valid file descriptor.

EFAULT times pointed to an invalid address; or, dirfd was AT\_FDCWD, and path is NULL or an invalid address.

EINVAL Invalid value in flags.

EINVAL Invalid value in one of the tv\_nsec fields (value outside range [0, 999,999,999], and



utimens(fd, times) is implemented as:

```
utimensat(fd, NULL, times, 0);
```

Note, however, that the glibc wrapper for utimensat() disallows passing NULL as the value for path: the wrapper function returns the error EINVAL in this case.

## STANDARDS

POSIX.1-2024.

## HISTORY

utimensat()

Linux 2.6.22, glibc 2.6. POSIX.1-2008.

futimens()

glibc 2.6. POSIX.1-2008.

## NOTES

utimensat() obsoletes futimesat(2).

On Linux, timestamps cannot be changed for a file marked immutable, and the only change permitted for files marked append-only is to set the timestamps to the current time. (This is consistent with the historical behavior of utime(2) and utimes(2) on Linux.)

If both tv\_nsec fields are specified as UTIME\_OMIT, then the Linux implementation of utimensat() succeeds even if the file referred to by dirfd and path does not exist.

## BUGS

Several bugs afflict utimensat() and futimens() before Linux 2.6.26. These bugs are either nonconformances with the POSIX.1 draft specification or inconsistencies with historical Linux behavior.

? POSIX.1 specifies that if one of the tv\_nsec fields has the value UTIME\_NOW or UTIME\_OMIT, then the value of the corresponding tv\_sec field should be ignored. Instead, the value of the tv\_sec field is required to be 0 (or the error EINVAL results).

? Various bugs mean that for the purposes of permission checking, the case where both tv\_nsec fields are set to UTIME\_NOW isn't always treated the same as specifying times as NULL, and the case where one tv\_nsec value is UTIME\_NOW and the other is UTIME\_OMIT isn't treated the same as specifying times as a pointer to an array of structures containing arbitrary time values. As a result, in some cases: a) file timestamps can be updated by a process that shouldn't have permission to perform updates; b) file timestamps can't be updated by a process that should have permission to perform updates; and c) the wrong error value is returned in case of an error.

? POSIX.1 says that a process that has write access to the file can make a call with times as NULL, or with times pointing to an array of structures in which both tv\_nsec fields are UTIME\_NOW, in order to update both timestamps to the current time. However, futimens() instead checks whether the access mode of the file descriptor allows writing.

#### SEE ALSO

chattr(1), touch(1), futimesat(2), openat(2), stat(2), utimes(2), futimes(3), timespec(3), inode(7), path\_resolution(7), symlink(7)

Linux man-pages 6.17

2025-10-29

utimensat(2)