NAME

fstat, **fstat64**, **lstat**, **lstat64**, **stat**, **stat64**, **fstatat** — get file status

SYNOPSIS

#include <sys/stat.h>

int
fstat(*int fildes*, *struct stat *buf*);

int
lstat(*const char *restrict path*, *struct stat *restrict buf*);

int
stat(*const char *restrict path*, *struct stat *restrict buf*);

int
fstatat(*int fd*, *const char *path*, *struct stat *buf*, *int flag*);

TRANSITIIONAL SYNOPSIS (NOW DEPRECATED)

int
fstat64(*int fildes*, *struct stat64 *buf*);

int
lstat64(*const char *restrict path*, *struct stat64 *restrict buf*);

int
stat64(*const char *restrict path*, *struct stat64 *restrict buf*);

DESCRIPTION

The **stat**() function obtains information about the file pointed to by *path*. Read, write or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.

The **lstat**() function is like **stat**() except in the case where the named file is a symbolic link; **lstat**() returns information about the link, while **stat**() returns information about the file the link references. For symbolic links, the st_mode member contains meaningful information when used with the file type macros, and the st_size member contains the length of the pathname contained in the symbolic link. File mode bits and the contents of the remaining members of the stat structure are unspecified. The value returned in the st_size member is the length of the contents of the symbolic link, and does not count any trailing null.

The **fstat**() obtains the same information about an open file known by the file descriptor *fildes*.

The **fstatat**() system call is equivalent to **stat**() and **lstat**() except in the case where the *path* specifies a relative path. In this case the status is retrieved from a file relative to the directory associated with the file descriptor *fd* instead of the current working directory.

The values for the *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <`fcntl.h`>:

`AT_SYMLINK_NOFOLLOW`
    If *path* names a symbolic link, the status of the symbolic link is returned.

If **fstatat**() is passed the special value `AT_FDCWD` in the *fd* parameter, the current working directory is used and the behavior is identical to a call to **stat**() or **lstat**() respectively, depending on whether or not the `AT_SYMLINK_NOFOLLOW` bit is set in *flag*.

The *buf* argument is a pointer to a *stat* structure as defined by ⟨`sys/stat.h`⟩ and into which information is placed concerning the file. When the macro `_DARWIN_FEATURE_64_BIT_INODE` is not defined (see below for more information about this macro), the *stat* structure is defined as:

```
struct stat { /* when _DARWIN_FEATURE_64_BIT_INODE is NOT defined */
    dev_t    st_dev;    /* device inode resides on */
    ino_t    st_ino;    /* inode's number */
    mode_t   st_mode;   /* inode protection mode */
    nlink_t  st_nlink;  /* number of hard links to the file */
    uid_t    st_uid;    /* user-id of owner */
    gid_t    st_gid;    /* group-id of owner */
    dev_t    st_rdev;   /* device type, for special file inode */
    struct timespec st_atimespec;  /* time of last access */
    struct timespec st_mtimespec;  /* time of last data modification */
    struct timespec st_ctimespec;  /* time of last file status change */
    off_t    st_size;   /* file size, in bytes */
    quad_t   st_blocks; /* blocks allocated for file */
    u_long   st_blksize;/* optimal file sys I/O ops blocksize */
    u_long   st_flags;  /* user defined flags for file */
    u_long   st_gen;    /* file generation number */
};
```

However, when the macro `_DARWIN_FEATURE_64_BIT_INODE` is defined, the *stat* structure will now be defined as:

```
struct stat { /* when _DARWIN_FEATURE_64_BIT_INODE is defined */
    dev_t           st_dev;           /* ID of device containing file */
    mode_t          st_mode;          /* Mode of file (see below) */
    nlink_t         st_nlink;         /* Number of hard links */
    ino_t          st_ino;           /* File serial number */
    uid_t           st_uid;           /* User ID of the file */
    gid_t           st_gid;           /* Group ID of the file */
    dev_t           st_rdev;          /* Device ID */
    struct timespec st_atimespec;     /* time of last access */
    struct timespec st_mtimespec;     /* time of last data modification */
    struct timespec st_ctimespec;     /* time of last status change */
    struct timespec st_birthtimespec; /* time of file creation(birth) */
    off_t           st_size;          /* file size, in bytes */
    blkcnt_t        st_blocks;        /* blocks allocated for file */
    blksize_t       st_blksize;       /* optimal blocksize for I/O */
    uint32_t        st_flags;         /* user defined flags for file */
    uint32_t        st_gen;           /* file generation number */
    int32_t         st_lspare;        /* RESERVED: DO NOT USE! */
    int64_t         st_qspare[2];     /* RESERVED: DO NOT USE! */
};
```

The time-related fields of *struct stat* are as follows:

st_atime	Time when file data last accessed. Changed by the `mknod`(2), `utimes`(2) and `read`(2) system calls.
st_mtime	Time when file data last modified. Changed by the `mknod`(2), `utimes`(2) and `write`(2) system calls.
st_ctime	Time when file status was last changed (inode data modification). Changed by the `chmod`(2), `chown`(2), `link`(2), `mknod`(2), `rename`(2), `unlink`(2), `utimes`(2) and `write`(2) system calls.
st_birthtime	Time of file creation. Only set once when the file is created. This field is only available in the 64 bit inode variants. On filesystems where birthtime is not available, this field is set to 0 (i.e. epoch).

The size-related fields of the structures are as follows:

st_blksize	The optimal I/O block size for the file.
st_blocks	The actual number of blocks allocated for the file in 512-byte units. As short symbolic links are stored in the inode, this number may be zero.

The status information word *st_mode* has the following bits:

```
#define S_IFMT 0170000           /* type of file */
#define        S_IFIFO  0010000  /* named pipe (fifo) */
#define        S_IFCHR  0020000  /* character special */
#define        S_IFDIR  0040000  /* directory */
#define        S_IFBLK  0060000  /* block special */
#define        S_IFREG  0100000  /* regular */
#define        S_IFLNK  0120000  /* symbolic link */
#define        S_IFSOCK 0140000  /* socket */
#define        S_IFWHT  0160000  /* whiteout */
#define S_ISUID 0004000  /* set user id on execution */
#define S_ISGID 0002000  /* set group id on execution */
#define S_ISVTX 0001000  /* save swapped text even after use */
#define S_IRUSR 0000400  /* read permission, owner */
#define S_IWUSR 0000200  /* write permission, owner */
#define S_IXUSR 0000100  /* execute/search permission, owner */
```

For a list of access modes, see ⟨sys/stat.h⟩, access(2) and chmod(2).

For a list of the file flags in the *st_flags* field, see ⟨sys/stat.h⟩ and chflags(2).

_DARWIN_FEATURE_64_BIT_INODE

In order to accommodate advanced capabilities of newer file systems, the *struct stat*, *struct statfs*, and *struct dirent* data structures were updated in Mac OSX 10.5.

The most obvious change is the increased size of *ino_t* from 32 bits to 64 bits. As a consequence, storing an ino_t in an int is no longer safe, and file formats storing ino_t as 32-bit values may need to be updated. There are other changes as well, such as the widening of *f_fstypename*, *f_mntonname*, and *f_mntfromname* in *struct statfs*. Please refer to dir(5) for more detail on the specific changes to the other affected data structures.

On platforms that existed before these updates were available, ABI compatibility is achieved by providing two implementations for related functions: one using the legacy data structures and one using the updated data structures. Variants which make use of the newer structures have their symbols suffixed with $INODE64. These $INODE64 suffixes are automatically appended by the compiler tool-chain and should not be used directly.

Platforms that were released after these updates only have the newer variants available to them. These platforms have the macro _DARWIN_FEATURE_ONLY_64_BIT_INODE defined.

The _DARWIN_FEATURE_64_BIT_INODE macro should not be set directly. Instead, developers should make use of the _DARWIN_NO_64_BIT_INODE or _DARWIN_USE_64_BIT_INODE macros when the default variant is not desired. The following table details the effects of defining these macros for different deployment targets.

_DARWIN_FEATURE_ONLY_64_BIT_INODE **not defined**

	Deployment Target		
user defines:	< 10.5	10.5	> 10.5

(none)	32-bit	32-bit	64-bit
`_DARWIN_NO_64_BIT_INODE`	32-bit	32-bit	32-bit
`_DARWIN_USE_64_BIT_INODE`	32-bit	64-bit	64-bit

`_DARWIN_FEATURE_ONLY_64_BIT_INODE` **defined**

user defines:	Any Deployment Target
(none)	64-bit-only
`_DARWIN_NO_64_BIT_INODE`	*(error)*
`_DARWIN_USE_64_BIT_INODE`	64-bit-only

32-bit	32-bit inode values are enabled, and the legacy structures involving the *`ino_t`* type are in use. The macro `_DARWIN_FEATURE_64_BIT_INODE` is not defined.
64-bit	64-bit inode values are enabled, and the expanded structures involving the *`ino_t`* type are in use. The macro `_DARWIN_FEATURE_64_BIT_INODE` is defined, and loader symbols will contain the `$INODE64` suffix.
64-bit-only	Like 64-bit, except loader symbols do not have the `$INODE64` suffix.
(error)	A compile time error is generated.

Due to the increased benefits of the larger structure, it is highly recommended that developers not define `_DARWIN_NO_64_BIT_INODE` and make use of `_DARWIN_USE_64_BIT_INODE` when targeting Mac OSX 10.5.

In addition to the $INODE64 suffixed symbols, variants suffixed with 64 are also available for related functions. These functions were provided as a way for developers to use the updated structures in code that also made use of the legacy structures. The enlarged stat structures were also prefixed with 64 to distinguish them from their legacy variants. These functions have been deprecated and should be avoided.

RETURN VALUES

Upon successful completion a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

COMPATIBILITY

Previous versions of the system used different types for the `st_dev`, `st_uid`, `st_gid`, `st_rdev`, `st_size`, `st_blksize` and `st_blocks` fields.

ERRORS

The **`fstat`**() system call will fail if:

`[EBADF]`	*`fildes`* is not a valid open file descriptor.
`[EFAULT]`	*`Sb`* points to an invalid address.
`[EIO]`	An I/O error occurs while reading from or writing to the file system.

The **`lstat`**() and **`stat`**() system calls will fail if:

`[EACCES]`	Search permission is denied for a component of the path prefix.
`[EFAULT]`	*`Sb`* or *name* points to an invalid address.
`[EIO]`	An I/O error occurs while reading from or writing to the file system.
`[ELOOP]`	Too many symbolic links are encountered in translating the pathname. This is taken to be indicative of a looping symbolic link.

[ENAMETOOLONG]	A component of a pathname exceeds {NAME_MAX} characters, or an entire path name exceeds {PATH_MAX} characters.
[ENOENT]	The named file does not exist.
[ENOTDIR]	A component of the path prefix is not a directory.

The **fstat**(), **lstat**(), and **stat**() system calls will fail if:

[EOVERFLOW]	The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

In addition to the errors returned by the **stat**() and **lstat**(), **fstatat**() may fail if:

[EBADF]	The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for searching.
[EINVAL]	The value of the *flag* argument is not valid.
[ENOTDIR]	The *path* argument is not an absolute path and *fd* is neither AT_FDCWD nor a file descriptor associated with a directory.

CAVEATS

The file generation number, *st_gen*, is only available to the super-user.
The fields in the stat structure currently marked *st_spare1*, *st_spare2*, and *st_spare3* are present in preparation for inode time stamps expanding to 64 bits. This, however, can break certain programs that depend on the time stamps being contiguous (in calls to utimes(2)).

TRANSITIONAL DESCRIPTION (NOW DEPRECATED)

The *fstat64*, *lstat64* and *stat64* routines are equivalent to their corresponding non-64-suffixed routine, when 64-bit inodes are in effect. They were added before there was support for the symbol variants, and so are now deprecated. Instead of using these, set the _DARWIN_USE_64_BIT_INODE macro before including header files to force 64-bit inode support.

The *stat64* structure used by these deprecated routines is the same as the *stat* structure when 64-bit inodes are in effect (see above).

SEE ALSO

chflags(2), chmod(2), chown(2), utimes(2), compat(5), statfs(2), symlink(7)

BUGS

Applying fstat to a socket (and thus to a pipe) returns a zero'd buffer, except for the blocksize field, and a unique device and inode number.

STANDARDS

The **stat**() and **fstat**() function calls are expected to conform to IEEE Std 1003.1-1988 ("POSIX.1"). The **fstatat**() system call is expected to conform to POSIX.1-2008 .

HISTORY

An **lstat**() function call appeared in 4.2BSD. The **stat64**(), **fstat64**(), and **lstat64**() system calls first appeared in Mac OS X 10.5 (Leopard) and are now deprecated in favor of the corresponding symbol variants. The **fstatat**() system call appeared in OS X 10.10