Table 27.6. Lock-Mode Constants

LOCK_EX	Exclusive lock. Only one process may hold an exclusive lock for a given file at a time.
LOCK_NB	Don't block when locking. May be combined with other lock options using logical *or*.
LOCK_SH	Shared lock. Multiple processes may each hold a shared lock for a given file at the same time.
LOCK_UN	Unlock.

Class

File::Stat < Object

Objects of class File::Stat encapsulate common status information for File objects. The information is recorded at the moment the File::Stat object is created; changes made to the file after that point will not be reflected. File::Stat objects are returned by IO#stat, File.stat, File#lstat, and File.lstat. Many of these methods may return platform-specific values, and not all values are meaningful on all systems. See also Kernel#test on page 579.

Mixes in

Comparable:

`<, <=, ==, >=, >, between?`

Instance methods

<=> *statfile* <=> *other_stat* → −1, 0, 1

Compares File::Stat objects by comparing their respective modification times.

```
f1 = File.new("f1", "w")
sleep 1
f2 = File.new("f2", "w")
f1.stat <=> f2.stat   # =>   -1
# Methods in Comparable are also available
f1.stat > f2.stat     # =>   false
f1.stat < f2.stat     # =>   true
```

atime *statfile*.atime → *time*

Returns a Time object containing the last access time for *statfile*, or returns epoch if the file has not been accessed.

```
File.stat("testfile").atime        # =>   1969-12-31 18:00:00 -0600
File.stat("testfile").atime.to_i   # =>   0
```

blksize *statfile*.blksize → *int*

Returns the native file system's block size. Will return nil on platforms that don't support this information.

```
File.stat("testfile").blksize   # =>   4096
```

blockdev?

statfile.blockdev? → true or false

Returns true if the file is a block device and returns false if it isn't or if the operating system doesn't support this feature.

```
File.stat("testfile").blockdev?     # =>   false
File.stat("/dev/disk0").blockdev?   # =>   true
```

blocks

statfile.blocks → *int*

Returns the number of native file system blocks allocated for this file or returns nil if the operating system doesn't support this feature.

```
File.stat("testfile").blocks   # =>   8
```

chardev?

statfile.chardev? → true or false

Returns true if the file is a character device and returns false if it isn't or if the operating system doesn't support this feature.

```
File.stat("/dev/tty").chardev?   # =>   true
File.stat("testfile").chardev?   # =>   false
```

ctime

statfile.ctime → *time*

Returns a Time object containing the time that the file status associated with *statfile* was changed.

```
File.stat("testfile").ctime   # =>   2009-04-13 13:26:23 -0500
```

dev

statfile.dev → *int*

Returns an integer representing the device on which *statfile* resides. The bits in the device integer will often encode major and minor device information.

```
File.stat("testfile").dev          # =>   234881029
"%x" % File.stat("testfile").dev   # =>   "e000005"
```

dev_major

statfile.dev_major → *int*

Returns the major part of File::Stat#dev or nil if the operating system doesn't support this feature.

```
File.stat("testfile").dev_major   # =>   14
```

dev_minor

statfile.dev_minor → *int*

Returns the minor part of File::Stat#dev or nil if the operating system doesn't support this feature.

```
File.stat("testfile").dev_minor   # =>   5
```

directory?

statfile.directory? → true or false

Returns true if *statfile* is a directory and returns false otherwise.

```
File.stat("testfile").directory?   # =>   false
File.stat(".").directory?          # =>   true
```

executable?

statfile.executable? → true or false

Returns true if *statfile* is executable or if the operating system doesn't distinguish executable files from nonexecutable files. The tests are made using the effective owner of the process.

```
File.stat("testfile").executable?   # =>   false
```

executable_real?

statfile.executable_real? → true or false

Same as executable? but tests using the real owner of the process.

file?

statfile.file? → true or false

Returns true if *statfile* is a regular file (not a device file, pipe, socket, and so on).

```
File.stat("testfile").file?   # =>   true
```

ftype

statfile.ftype → *type_string*

Identifies the type of *statfile*. The return string is one of the following: file, directory, characterSpecial, blockSpecial, fifo, link, socket, or unknown.

```
File.stat("/dev/tty").ftype   # =>   "characterSpecial"
```

gid

statfile.gid → *int*

Returns the numeric group ID of the owner of *statfile*.

```
File.stat("testfile").gid   # =>   501
```

grpowned?

statfile.grpowned? → true or false

Returns true if the effective group ID of the process is the same as the group ID of *statfile*. On Windows, returns false.

```
File.stat("testfile").grpowned?      # =>   true
File.stat("/etc/passwd").grpowned?   # =>   false
```

ino

statfile.ino → *int*

Returns the inode number for *statfile*.

```
File.stat("testfile").ino   # =>   1707345
```

mode

statfile.mode → *int*

Returns an integer representing the permission bits of *statfile*. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

```
File.chmod(0644, "testfile")         # =>   1
File.stat("testfile").mode.to_s(8)   # =>   "100644"
```

mtime

statfile.mtime → *time*

Returns a Time object containing the modification time for *statfile*.

```
File.stat("testfile").mtime   # =>  2009-04-13 13:26:23 -0500
```

nlink

statfile.nlink → *int*

Returns the number of hard links to *statfile*.

```
File.stat("testfile").nlink             # =>  1
File.link("testfile", "testfile.bak")   # =>  0
File.stat("testfile").nlink             # =>  2
```

owned?

statfile.owned? → true or false

Returns true if the effective user ID of the process is the same as the owner of *statfile*.

```
File.stat("testfile").owned?      # =>  true
File.stat("/etc/passwd").owned?   # =>  false
```

pipe?

statfile.pipe? → true or false

Returns true if the operating system supports pipes and *statfile* is a pipe.

rdev

statfile.rdev → *int*

Returns an integer representing the device type on which *statfile* (which should be a special file) resides. Returns nil if the operating system doesn't support this feature.

```
File.stat("/dev/disk0s1").rdev   # =>  234881025
File.stat("/dev/tty").rdev       # =>  33554432
```

rdev_major

statfile.rdev_major → *int*

Returns the major part of File::Stat#rdev or nil if the operating system doesn't support this feature.

```
File.stat("/dev/disk0s1").rdev_major   # =>  14
File.stat("/dev/tty").rdev_major       # =>  2
```

rdev_minor

statfile.rdev_minor → *int*

Returns the minor part of File::Stat#rdev or nil if the operating system doesn't support this feature.

```
File.stat("/dev/disk0s1").rdev_minor   # =>  1
File.stat("/dev/tty").rdev_minor       # =>  0
```

readable?

statfile.readable? → true or false

Returns true if *statfile* is readable by the effective user ID of this process.

```
File.stat("testfile").readable?   # =>  true
```

readable_real?

statfile.readable_real? → true or false

Returns true if *statfile* is readable by the real user ID of this process.

```
File.stat("testfile").readable_real?     # =>   true
File.stat("/etc/passwd").readable_real?  # =>   true
```

setgid?

statfile.setgid? → true or false

Returns true if *statfile* has the set-group-id permission bit set and returns false if it doesn't or if the operating system doesn't support this feature.

```
File.stat("testfile").setgid?            # =>   false
File.stat("/usr/sbin/postdrop").setgid?  # =>   true
```

setuid?

statfile.setuid? → true or false

Returns true if *statfile* has the set-user-id permission bit set and returns false if it doesn't or if the operating system doesn't support this feature.

```
File.stat("testfile").setuid?     # =>   false
File.stat("/usr/bin/su").setuid?  # =>   true
```

size

statfile.size → *int*

Returns the size of *statfile* in bytes.

```
File.stat("/dev/zero").size   # =>   0
File.stat("testfile").size    # =>   66
```

size?

statfile.size? → *int* or nil

Returns nil if *statfile* is a zero-length file; otherwise, returns the file size. Usable as a condition in tests.

```
File.stat("/dev/zero").size?   # =>   nil
File.stat("testfile").size?    # =>   66
```

socket?

statfile.socket? → true or false

Returns true if *statfile* is a socket and returns false if it isn't or if the operating system doesn't support this feature.

```
File.stat("testfile").socket?   # =>   false
```

sticky?

statfile.sticky? → true or false

Returns true if *statfile* has its sticky bit set and returns false if it doesn't or if the operating system doesn't support this feature.

```
File.stat("testfile").sticky?   # =>   false
```

symlink?

statfile.symlink? → true or false

Returns true if *statfile* is a symbolic link; returns false if it isn't or if the operating system

doesn't support this feature. Because File.stat automatically follows symbolic links, symlink? will always be false for an object returned by File.stat.

```
File.symlink("testfile", "alink")   # =>   0
File.stat("alink").symlink?         # =>   false
File.lstat("alink").symlink?        # =>   true
```

uid

statfile.uid → *int*

Returns the numeric user ID of the owner of *statfile*.

```
File.stat("testfile").uid   # =>   501
```

world_readable?

File.world_readable?(*filename*) → *perm_int* or nil

1.9

If *filename* is readable by others, returns an integer representing the file permission bits of *filename*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

```
File.world_readable?("/etc/passwd")           # =>   420
File.world_readable?("/etc/passwd").to_s(8)   # =>   "644"
```

world_writable?

File.world_writable?(*filename*) → *perm_int* or nil

1.9

If *filename* is writable by others, returns an integer representing the file permission bits of *filename*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

```
File.world_writable?("/etc/passwd")    # =>   nil
File.world_writable?("/tmp")           # =>   511
File.world_writable?("/tmp").to_s(8)   # =>   "777"
```

writable?

statfile.writable? → true or false

Returns true if *statfile* is writable by the effective user ID of this process.

```
File.stat("testfile").writable?   # =>   true
```

writable_real?

statfile.writable_real? → true or false

Returns true if *statfile* is writable by the real user ID of this process.

```
File.stat("testfile").writable_real?   # =>   true
```

zero?

statfile.zero? → true or false

Returns true if *statfile* is a zero-length file; returns false otherwise.

```
File.stat("testfile").zero?   # =>   false
```