`lfs.attributes (filepath [, request_name | result_table])`

Returns a table with the file attributes corresponding to `filepath` (or `nil` followed by an error message and a system-dependent error code in case of error). If the second optional argument is given and is a string, then only the value of the named attribute is returned (this use is equivalent to `lfs.attributes(filepath)[request_name]`, but the table is not created and only one attribute is retrieved from the O.S.). if a table is passed as the second argument, it (`result_table`) is filled with attributes and returned instead of a new table. The attributes are described as follows; attribute `mode` is a string, all the others are numbers, and the time related attributes use the same time reference of `os.time`:

`dev`
: on Unix systems, this represents the device that the inode resides on. On Windows systems, represents the drive number of the disk containing the file

`ino`
: on Unix systems, this represents the inode number. On Windows systems this has no meaning

`mode`
: string representing the associated protection mode (the values could be `file`, `directory`, `link`, `socket`, `named pipe`, `char device`, `block device` or `other`)

`nlink`
: number of hard links to the file

`uid`
: user-id of owner (Unix only, always 0 on Windows)

`gid`
: group-id of owner (Unix only, always 0 on Windows)

`rdev`
: on Unix systems, represents the device type, for special file inodes. On Windows systems represents the same as `dev`

`access`
: time of last access

`modification`
: time of last data modification

`change`
: time of last file status change

`size`
: file size, in bytes

`permissions`
: file permissions string

`blocks`
: block allocated for file; (Unix only)

`blksize`
: optimal file system I/O blocksize; (Unix only)

This function uses `stat` internally thus if the given `filepath` is a symbolic link, it is followed (if it points to another link the chain is followed recursively) and the information is about the file it refers to. To obtain information about the link itself, see function lfs.symlinkattributes.

lfs.chdir (path)
Changes the current working directory to the given `path`.
Returns `true` in case of success or `nil` plus an error string.

lfs.lock_dir(path, [seconds_stale])
Creates a lockfile (called lockfile.lfs) in `path` if it does not exist and returns the lock. If the lock already exists checks if it's stale, using the second parameter (default for the second parameter is `INT_MAX`, which in practice means the lock will never be stale. To free the the lock call `lock:free()`.
In case of any errors it returns nil and the error message. In particular, if the lock exists and is not stale it returns the "File exists" message.

lfs.currentdir ()
Returns a string with the current working directory or `nil` plus an error string.

iter, dir_obj = lfs.dir (path)
Lua iterator over the entries of a given directory. Each time the iterator is called with `dir_obj` it returns a directory entry's name as a string, or `nil` if there are no more entries. You can also iterate by calling `dir_obj:next()`, and explicitly close the directory before the iteration finished with `dir_obj:close()`. Raises an error if `path` is not a directory.

lfs.lock (filehandle, mode[, start[, length]])
Locks a file or a part of it. This function works on open files; the file handle should be specified as the first argument. The string `mode` could be either `r` (for a read/shared lock) or `w` (for a write/exclusive lock). The optional arguments `start` and `length` can be used to specify a starting point and its length; both should be numbers.
Returns `true` if the operation was successful; in case of error, it returns `nil` plus an error string.

lfs.link (old, new[, symlink])
Creates a link. The first argument is the object to link to and the second is the name of the link. If the optional third argument is true, the link will by a symbolic link (by default, a hard link is created).

lfs.mkdir (dirname)
Creates a new directory. The argument is the name of the new directory.
Returns `true` in case of success or `nil`, an error message and a system-dependent error code in case of error.

lfs.rmdir (dirname)
Removes an existing directory. The argument is the name of the directory.
Returns `true` in case of success or `nil`, an error message and a system-dependent error code in case of error.

`lfs.setmode (file, mode)`

Sets the writing mode for a file. The mode string can be either `"binary"` or `"text"`. Returns `true` followed the previous mode string for the file, or `nil` followed by an error string in case of errors. On non-Windows platforms, where the two modes are identical, setting the mode has no effect, and the mode is always returned as `binary`.

`lfs.symlinkattributes (filepath [, request_name])`

Identical to lfs.attributes except that it obtains information about the link itself (not the file it refers to). It also adds a `target` field, containing the file name that the symlink points to. On Windows this function does not yet support links, and is identical to `lfs.attributes`.

`lfs.touch (filepath [, atime [, mtime]])`

Set access and modification times of a file. This function is a bind to `utime` function. The first argument is the filename, the second argument (`atime`) is the access time, and the third argument (`mtime`) is the modification time. Both times are provided in seconds (which should be generated with Lua standard function `os.time`). If the modification time is omitted, the access time provided is used; if both times are omitted, the current time is used.
Returns `true` in case of success or `nil`, an error message and a system-dependent error code in case of error.

`lfs.unlock (filehandle[, start[, length]])`

Unlocks a file or a part of it. This function works on open files; the file handle should be specified as the first argument. The optional arguments `start` and `length` can be used to specify a starting point and its length; both should be numbers.
Returns `true` if the operation was successful; in case of error, it returns `nil` plus an error string.