



LuaFileSystem

File System Library for the Lua Programming Language

Introduction

LuaFileSystem is a [Lua](#) library developed to complement the set of functions related to file systems offered by the standard Lua distribution.

LuaFileSystem offers a portable way to access the underlying directory structure and file attributes.

LuaFileSystem is free software and uses the same [license](#) as Lua 5.1.

Reference

LuaFileSystem offers the following functions:

lfs.attributes (filepath [, aname])

Returns a table with the file attributes corresponding to `filepath` (or `nil` followed by an error message in case of error). If the second optional argument is given, then only the value of the named attribute is returned (this use is equivalent to

`lfs.attributes(filepath).aname,`

but the table is not created and only one attribute is retrieved from the O.S.). 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

This represents the drive number of the disk containing the file

ino

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

Always 0 on Windows

gid

Always 0 on Windows

rdev

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

blocks

Not used on Windows

blksize

Not used on Windows

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 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.mkdir (dirname)

Creates a new directory. The argument is the name of the new directory.

Returns `true` if the operation was successful; in case of error, it returns `nil` plus an error string.

lfs.rmdir (dirname)

Removes an existing directory. The argument is the name of the directory.

Returns `true` if the operation was successful; in case of error, it returns `nil` plus an error string.

lfs.setmode (file, mode)

Sets the writing mode for a file. The mode string can be either `binary` or `text`. Returns the previous mode string for the file.

lfs.symlinkattributes (filepath [, aname])

This function is not available in Windows.

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` if the operation was successful; in case of error, it returns `nil` plus an error string.

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.

License

LuaFileSystem is free software: it can be used for both academic and commercial purposes at absolutely no cost. There are no royalties or GNU-like "copyleft" restrictions. LuaFileSystem qualifies as [Open Source](#) software. Its licenses are compatible with [GPL](#). LuaFileSystem is not in the public domain and the [Kepler Project](#) keep its copyright. The legal details are below.

The spirit of the license is that you are free to use LuaFileSystem for any purpose at no cost without having to ask us. The only requirement is that if you do use LuaFileSystem, then you should give us credit by including the appropriate copyright notice somewhere in your product or its documentation.

The LuaFileSystem library is designed and implemented by Roberto Ierusalimschy, André Carregal and Tomás Guisasola. The implementation is not derived from licensed software.

Copyright © 2003 Kepler Project.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.