# Generated Documentation

## Class mysqlfs

*[line 2]*

- **Package** default

**mysqlfs::$connection**

*mixed* = *[line 6]*

- **Access** protected

**mysqlfs::$group**

*mixed* = 0 *[line 3]*

# Contents

- **Access** public

**mysqlfs::$security_settings**

*mixed* = array() *[line 5]*

- **Access** protected

**mysqlfs::$table**

*mixed* =  *[line 7]*

- **Access** protected

**mysqlfs::$user**

*mixed* = NULL *[line 4]*

- **Access** public

Constructor *void* function mysqlfs::__construct($conn, [$tbl = 'index']) *[line 15]*
***Function Parameters:***

- **$conn**
- **$tbl**

**void __construct(  resource $connction [, string $table = 'index' ] )**
Startup function for the class. Registers MySQL connection and table settings

*void* function mysqlfs::chgrp($path, $group) *[line 473]*

***Function Parameters:***

- **$path**
- **$group**

**bool chgrp ( string $path, int $group )**

UNIX chgrp command  Only current group and owner can chgrp  Returns true on success, false on failure

*void* function mysqlfs::chmod($path, $mode) *[line 444]*

***Function Parameters:***

- **$path**
- **$mode**

**bool chmod ( string $path, int $mode )**

UNIX chmod command  Returns true on success, false on failure

*void* function mysqlfs::chown($path, $user) *[line 458]*

***Function Parameters:***

- **$path**
- **$user**

**bool chown (string $path, string $user )**

UNIX chown command  Only the current owner can chown  Returns true on success, false on failure

*void* function mysqlfs::closedir([$handle = NULL]) *[line 894]*

***Function Parameters:***

- **$handle**

**void closedir ( [sra $handle ]) Closes the directory stream indicated by $handle. The stream must have previously been opened by opendir().**

*void* function mysqlfs::copy($source, $dest) *[line 517]*
***Function Parameters:***

- **$source**
- **$dest**

**bool copy ( string $source, string $dest )**
Copy $source to $dest You must have reading permissions on $source and writing permissions on $dest Returns true on success, false on failure

*void* function mysqlfs::exec($path) *[line 430]*
***Function Parameters:***

- **$path**

**mixed exec ( string $path )**
Executes a PHP file Returns the value returned by the script

*void* function mysqlfs::fgetc($handle) *[line 625]*
***Function Parameters:***

- **$handle**

**string fgetc( sra $handle ) Gets a character from the given file pointer.**

*void* function mysqlfs::fgets($handle, [$length = -1]) *[line 603]*
***Function Parameters:***

- **$handle**

- **$length**

**string fgets( sra $handle, [int $length=-1] )  Gets a line from file pointer.**

*void* function mysqlfs::file($path, [$flags = 0]) *[line 214]*
***Function Parameters:***

- **$path**
- **$flags**

*void* function mysqlfs::filegroup($path) *[line 114]*
***Function Parameters:***

- **$path**

**int filegroup ( string $path )**
Gets the file group. The group ID is returned in numerical format

*void* function mysqlfs::filemtime($path) *[line 122]*
***Function Parameters:***

- **$path**

**int filemtime( string $path )  This function returns the time when the data blocks of a file were being written to, that is, the time when the content of the file was changed.**

*void* function mysqlfs::fileowner($path) *[line 130]*
***Function Parameters:***

- **$path**

**string fileowner ( string $path )  Gets the file owner.**

*void* function mysqlfs::fileperms($path) *[line 138]*
**_Function Parameters:_**

- **$path**

**int fileperms( string $path )  Gets permissions for the given file.**

*void* function mysqlfs::filesize($path) *[line 146]*
**_Function Parameters:_**

- **$path**

**int filesize (string $path )**
Gets the size for the given file in bytes

*void* function mysqlfs::filetype($path) *[line 154]*
**_Function Parameters:_**

- **$path**

**string filetype( string $path )  Returns the type of the given file.**

*void* function mysqlfs::file_exists($path) *[line 283]*
**_Function Parameters:_**

- **$path**

**bool file_exists ( string $path )**
Checks if a file exists  Returns true if $path exists, false if $path doesn't exist

*void* function mysqlfs::file_get_contents($path) *[line 200]*
**_Function Parameters:_**

- **$path**

**string|false file_get_contents( string $path )**
Returns contents of the file at $path or false if $path isn't a file

*void* function mysqlfs::file_put_contents($path, $data) *[line 347]*
***Function Parameters:***

- **$path**
- **$data**

**bool file_put_contents( string $path, string $data )**
Puts $data into $pathinfo  Returns true on success or false on failure

*void* function mysqlfs::fopen($path, $mode) *[line 551]*
***Function Parameters:***

- **$path**
- **$mode**

**sra fopen ( string $path, string $mode )  fopen() binds a named resource, specified by $path, to a stream.**
Returns sra on success or false on failure

*void* function mysqlfs::fpassthru($handle) *[line 644]*
***Function Parameters:***

- **$handle**

**int fpassthru( sra $handle )**
Reads the file from the current pointer to EOF

*void* function mysqlfs::fputs($handle, $c) *[line 661]*
***Function Parameters:***

- **$handle**
- **$c**

**bool fputs( sra $handle, string $string )**
Alias of fwrite

*void* function mysqlfs::fread($handle, $length) *[line 693]*
***Function Parameters:***

- **$handle**
- **$length**

**string fread ( sra $handle, int $length )**
fread() reads up to length bytes from the file pointer referenced by handle. Reading stops as soon as one of the following conditions is met: * length bytes have been read * EOF (end of file) is reached * 8192 bytes have been read (after opening userspace stream)

*void* function mysqlfs::fseek($handle, $offset, $whence) *[line 714]*
***Function Parameters:***

- **$handle**
- **$offset**
- **$whence**

**int fseek( sra $handle, int $offset, [ int $whence=SEEK_SET ] ) Sets the file position indicator for the file referenced by handle. The new position, measured in bytes from the beginning of the file, is obtained by adding offset to the position specified by whence.**

*void* function mysqlfs::ftell($handle) *[line 727]*

***Function Parameters:***

- **$handle**

**int ftell( sra $handle ) Returns the position of the file pointer referenced by handle.**

*void* function mysqlfs::ftruncate($handle, $size) *[line 735]*
***Function Parameters:***

- **$handle**
- **$size**

**bool ftruncate( sra $handle, int $size ) Takes the filepointer, handle, and truncates the file to length, size.**

*void* function mysqlfs::fwrite($handle, $c) *[line 670]*
***Function Parameters:***

- **$handle**
- **$c**

**bool fwrite ( sra $handle, string $string )**
fwrite() writes the contents of $string to the file stream pointed to by $handle

*void* function mysqlfs::getinfo($path, [$what = NULL]) *[line 98]*
***Function Parameters:***

- **$path**
- **$what**

**mixed getinfo( string $path [, string $what = NULL] )**
Gets system info for a path Returns the requested info $what or array ( [type]=> string(1) "f"|"d", [filename]=> string , [time]=> int timestamp, [rights]=> int(3) UNIX mod, [mime]=> string mimetype, [size]=>

int filesize in bytes, [mfi]=> int MasterFileIndex, [owner]=> string|NULL UNIX owner, [group]=> int(1) UNIX owner group )

*void* function mysqlfs::is_dir($path) *[line 191]*
***Function Parameters:***

- **$path**

**bool is_dir( string $path )**
Returns true if the path is a directory, false otherwise

*void* function mysqlfs::is_empty($path) *[line 238]*
***Function Parameters:***

- **$path**

**bool is_empty( string $path )**
Checks if a directory is empty  Returns true if $path is a file or if $path is an empty directory,  false if the directory $path is not empty

*void* function mysqlfs::is_executable($path) *[line 394]*
***Function Parameters:***

- **$path**

**bool is_executable ( string $path )**
Checks if the file is executable by php  Returns true if the file is executable, false otherwise

*void* function mysqlfs::is_file($path) *[line 182]*
***Function Parameters:***

- **$path**

**bool is_file ( string $path )**

Returns true if the path is a file, false otherwise

*void* function mysqlfs::is_readable($path) *[line 362]*

***Function Parameters:***

- **$path**

**bool is_readable( string $path )**

Checks if the file is readable  Returns true is the file is readable, false otherwise

*void* function mysqlfs::is_writable($path) *[line 375]*

***Function Parameters:***

- **$path**

**bool is_writable( string $path )**

Checks if the file is writeable  Returns true if the file is writeable, false otherwise

*void* function mysqlfs::is_writeable($path) *[line 386]*

***Function Parameters:***

- **$path**

*void* function mysqlfs::mime_content_type($path) *[line 333]*

***Function Parameters:***

- **$path**

**string mime_content_type( string $path );**

Gets the content type based on the extention  Returns the mimetype *void* function mysqlfs::mkdir($path, [$mode = NULL]) *[line 299]*

***Function Parameters:***

- **$path**
- **$mode**

**bool mkdir( string $path, [int $mode=NULL] )**

Creates a directory $path if $mode is NULL, it takes the mode of the directory above Returns true on success or false on failure

*void* function mysqlfs::mkfile($path, [$mode = NULL], [$mime = NULL]) *[line 316]*

***Function Parameters:***

- **$path**
- **$mode**
- **$mime**

**bool mkfile (string $path, [int $mode=NULL, [string $mime=NULL]] )**

Creates a file $path if $mode is NULL, it takes the mode of the directory. If $mime is NULL, it searches the type based on the extention Returns true on success or false on failure

*void* function mysqlfs::move_uploaded_file($source, $dest) *[line 763]*

***Function Parameters:***

- **$source**
- **$dest**

**bool move_uploaded_file ( string $filename, string $destination ) This function checks to ensure that the file designated by filename is a valid upload file (meaning that it was uploaded via PHP's HTTP POST upload mechanism). If the file is valid, it will be moved to the filename given by destination.**

WARNING: !!! This function works on the actual filesystem and moves it to MySQLfs. !!! Make sure $filename is a valid name

*void* function mysqlfs::opendir($path) *[line 855]*
***Function Parameters:***

- **$path**

**sra opendir ( string $path ) Opens up a directory handle to be used in subsequent closedir(), readdir_full(), and rewinddir() calls.**

*void* function mysqlfs::readdir([$handle = NULL]) *[line 872]*
***Function Parameters:***

- **$handle**

**string readdir ( [sra $handle] ) Returns the filename of the next file from the directory.**
The filenames are returned in the order in which they are stored by the filesystem.

*void* function mysqlfs::readdir_full($path) *[line 58]*
***Function Parameters:***

- **$path**

**array|NULL readdir_full( string $path )**
Reads the directory into an array, like scandir() Returns NULL on failure or array ( [type]=> string(1) "f"|"d", [filename]=> string , [time]=> int timestamp, [rights]=> int(3) UNIX mod, [mime]=> string mimetype, [size]=> int filesize in bytes, [owner]=> string|NULL UNIX owner, [group]=> int(1) UNIX owner group ) on success

*void* function mysqlfs::readfile($path) *[line 773]*
***Function Parameters:***

- **$path**

**int readfile ( string $path ) Reads a file and writes it to the output buffer.**

*void* function mysqlfs::rename($oldpath, $newpath) *[line 487]*

***Function Parameters:***

- **$oldpath**
- **$newpath**

**bool rename ( string $oldpath, string $newpath )**

Rename $oldpath to $newpath You must have reading permissions on $oldpath and writing permissions on $newpath Returns true on success, false on failure

*void* function mysqlfs::rewind($handle) *[line 783]*

***Function Parameters:***

- **$handle**

**bool rewind ( sra $handle ) Sets the file position indicator for handle to the beginning of the file stream.**

*void* function mysqlfs::rewinddir([$handle = NULL]) *[line 886]*

***Function Parameters:***

- **$handle**

**void rewinddir ( [sra $handle] ) Resets the directory stream indicated by $handle to the beginning of the directory.**

*void* function mysqlfs::rmdir($path) *[line 253]*

***Function Parameters:***

- **$path**

**bool rmdir( string $path )**

Removes the empty directory $path Returns true on success and false on failure (probably dir not empty) *void* function mysqlfs::scandir($path, [$sorting_order = 0]) *[line 902]*

***Function Parameters:***

- **$path**
- **$sorting_order**

**array scandir ( string $path [, int $sorting_order] ) Returns an array of files and directories from the directory.**

*void* function mysqlfs::security($key) *[line 409]*

***Function Parameters:***

- **$key**

**bool security ( mixed $key )**

You can add some extra security here To write (once) supply array with key-value pairs To read, supply the array key

*void* function mysqlfs::setuser([$user = NULL], [$group = 0]) *[line 29]*

***Function Parameters:***

- **$user**
- **$group**

**void setuser ( string $user, string $group )**

Sets $user and $group

*void* function mysqlfs::simplify_path($path) *[line 162]*

***Function Parameters:***

- **$path**

**string simplify_path( string $path )**
Removes /./ and resolves /../

*void* function mysqlfs::sizetohuman($size) *[line 77]*
***Function Parameters:***

- **$size**

**string Sizetohuman( int $size )**
Converts bytes to Normal file sizes

*void* function mysqlfs::unlink($path) *[line 268]*
***Function Parameters:***

- **$path**

**bool unlink( string $path )**
Removes the file at $path  Returns true on success and false on failure

*void* function mysqlfs::valid_user($function) *[line 37]*
***Function Parameters:***

- **$function**

**void valid_user ( string $function )**
Uses $function to check if a user name is valid for chown

# Appendix A - Class Trees

## Package default

## mysqlfs

- mysqlfs

# Index