

# Fuse Extension

*version 0.90*

**Erick Galesio**

Université de Nice - Sophia Antipolis  
930 route des Colles, BP 145  
F-06903 Sophia Antipolis, Cedex  
France

This document was produced using the Scribe Programming Language and its ConTEXt engine.

For further information on Scribe, see <http://www-sop.inria.fr/mimosa/fp/Scribe/>.

Document created on June 12, 2007.

## 1 Introduction

This extension permits the implementation of virtual file systems in Scheme thanks to the **FUSE library**. Here is an excerpt of the FUSE site:

*With FUSE it is possible to implement a fully functional filesystem in a userspace program. Features include:*

- *Simple library API*
- *Simple installation (no need to patch or recompile the kernel)*
- *Secure implementation*
- *Userspace - kernel interface is very efficient*
- *Usable by non privileged users*
- *Runs on Linux kernels 2.4.X and 2.6.X*
- *Has proven very stable over time*

## 2 Using the fuse extension

To use this extension you need to include the following form in your program:

```
(begin (require "stklos-fuse") (import "stklos-fuse"))
```

This library provides only one entry point called “**fuse-mount**”. This function takes a first parameter which is the list of the program arguments and key-list of functions used to implement the file system. The arguments recognized by the version 2.6.5 “**fuse-mount**” are given below:

```

usage: <filesystem> mountpoint '(options)

general options:
  -o opt, '(opt...)      mount options
  -h  --help             print help
  -V  --version          print version

FUSE options:
  -d  -o debug           enable debug output (implies -f)
  -f                                     foreground operation
  -s                                     disable multi-threaded operation

  -o allow_other         allow access to other users
  -o allow_root          allow access to root
  -o nonempty            allow mounts over non-empty file/dir
  -o default_permissions enable permission checking by kernel
  -o fsname=NAME         set filesystem name
  -o large_read          issue large read requests (2.4 only)
  -o max_read=N          set maximum size of read requests

  -o hard_remove         immediate removal (don't hide files)
  -o use_ino             let filesystem set inode numbers
  -o readdir_ino        try to fill in d_ino in readdir
  -o direct_io          use direct I/O
  -o kernel_cache       cache files in kernel
  -o '(no)auto_cache    enable caching based on modification times
  -o umask=M            set file permissions (octal)
  -o uid=N              set file owner
  -o gid=N              set file group
  -o entry_timeout=T    cache timeout for names (1.0s)
  -o negative_timeout=T cache timeout for deleted names (0.0s)
  -o attr_timeout=T     cache timeout for attributes (1.0s)
  -o ac_attr_timeout=T  auto cache timeout for attributes (attr_timeout)
  -o intr               allow requests to be interrupted
  -o intr_signal=NUM    signal to send on interrupt (10)

  -o max_write=N        set maximum size of write requests
  -o max_readahead=N    set maximum readahead
  -o async_read         perform reads asynchronously (default)
  -o sync_read          perform reads synchronously

```

The list of functions used to implement the file system is given in the next section

### 3 File system primitives

The following functions are available to implement a Scheme file system:

<code>:getattr</code>	<i>path</i>	returns a vector of 8 elements containing: mode bits, number of links, size, uid, gid, atime, mtime, ctime
-----------------------	-------------	--

<b>:opendir</b>	<i>path</i>	this is a hook for controlling directory access, returns 0 if no error
<b>:readdir</b>	<i>path</i>	returns a list of the files in the firectorry "path"
<b>:releasedir</b>	<i>path</i>	This is a hook called after readdir, returns 0 if no error
<b>:mknod</b>	<i>path mode</i>	creates the file named "path" with given "mode"
<b>:open</b>	<i>path mode fd</i>	opens file named "path" with given mode (O = RDONLY, 1 = WRONLY, 2 = RDWR). The value "fd" is an unique integer associated by the system to this file
<b>:read</b>	<i>fd size offset</i>	returns a string of "size" bytes starting at "offset" on "fd"
<b>:write</b>	<i>fd buffer size offset</i>	writes the first "size" characters of "buffer" at "offset" on "fd". The offset can be after the actual end of file
<b>:release</b>	<i>fd</i>	This function is called when there are no more references to the open file "fd". The return value of this function is ignored
<b>:rename</b>	<i>from to</i>	renames file "from" with name "to"
<b>:unlink</b>	<i>path</i>	removes the file with given "path"
<b>:link</b>	<i>old new</i>	creates a link from file "old" to file "to"
<b>:symlink</b>	<i>old new</i>	creates a symbolic link from file "old" to file "to"
<b>:readlink</b>	<i>path</i>	returns the file that the symbolic link "path" point to
<b>:mkdir</b>	<i>path</i>	creates directory "path"
<b>:rmdir</b>	<i>path</i>	removes directory "path"
<b>:chmod</b>	<i>path mode</i>	change the mode of file "path" to "mode"
<b>:chown</b>	<i>path uid gid</i>	changes the owner of "path" to "uid" and "gid"
<b>:utime</b>	<i>path atime mtime</i>	changes the access and modification time of file "path" to "atime" and "mtime"
<b>:truncate</b>	<i>path size</i>	changes the size of "path" to "size"
<b>:flush</b>	<i>path fd</i>	flushes cached data on file "fd"
<b>:fsync</b>	<i>path datasync fd</i>	if the "datasync" parameter is non-zero, then only the user data should be flushed, not the meta data

<b>:fsyncdir</b>	<i>path datasync</i>	if the "datasync" parameter is non-zero, then only the user data should be flushed, not the meta data
<b>:init</b>		This is a hook called when the file system is mounted. It can return a value which will be used when the file system is unmounted
<b>:destroy</b>	<i>data</i>	This is a hook called when the file system is unmounted. Its parameter is the return value of the "init" call.

#### 4 The hellofs filesystem

The following example is a very simple (even simplistic) file system written in STKLOS. This is a file system which contains only a file named "hello". You cannot do a lot with this file system and most actions produce errors. To mount the file system you can for instance type:

```
$ hellofs -f ~/fuse
```

This will mount the hellofs on the (already existing and empty) "~/fuse" directory. To unmount this file system, you can do:

```
$ fusermount -u ~/fuse
```

A more complete and realistic example is provided in the "examples" directory.

```

(require "stklos-fuse")

(define-module HELLOFS
  (import stklos-posix stklos-fuse)

(define content "Hello, world!\n")

(define (hellofs-main args)
  (fuse-mount args
    :getattr (lambda (path)
      (let ((tm (current-seconds)))
        (cond
          ((equal? path "/")
           (vector (+ posix/ifdir #o755) ;; mode
                   2 ;; links
                   123 ;; size (why not this one?)
                   (posix-user-id) ;; uid
                   (posix-group-id) ;; gid
                   tm tm tm) ;; atime, mtime, ctime)
          ((equal? path "/hello")
           (vector (+ posix/ifreg #o440) ;; mode
                   1 ;; links
                   (string-length content) ;; size
                   (posix-user-id) ;; uid
                   (posix-group-id) ;; gid
                   tm tm tm) ;; atime, mtime, ctime)
          (else (- posix/enoent))))))

    :readdir (lambda (path)
      (if (equal? path "/")
          '("." ".." "hello")
          (- posix/enoent)))

    :open (lambda (path mode fd)
      (cond
        ((not (equal? path "/hello"))
         (- posix/enoent))
        ((not (equal? mode 0))
         (- posix/eaccess))
        (else
         0)))

    :read (lambda (fd size offset)
      (let ((len (string-length content)))
        (if (< offset len)
            (begin
              (if (> (+ offset size) len)
                  (set! size (- len offset)))
              (substring content offset size)
              0))))))

) ;; End of module HELLOFS

;; =====
;; HELLOFS ...
;; =====

(define main (in-module HELLOFS hellofs-main))

```

