

Posix Extension

version 0.50

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 January 6, 2006.

1 Introduction

This extension exposes Posix functionalities in STKLOS. Note that this version is **very incomplete** and only a few Posix functionalities are available for now.

2 Basic Usage

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

```
(require "posix")
```

The functions of this extension may set a special parameter value when an error occurs. The mechanism used is described here (see also ?? for more information). In general a function returns a useful result (or **#t** if no useful value is possible) if no error is detected. When a function detects an error, it returns **#f**. The error number is then available through the **(posix-errno)** parameter and a string describing the error can be built by the **(posix-error)** primitive.

3 Posix API

3.1 Directories Functions

```
(posix-make-directory path)
```

STKLOS
procedure

Create a directory with name "path". Return **#t** in case of success.

```
(posix-delete-directory path)
```

STKLOS
procedure

Delete the directory "path". Return **#t** in case of success.

3.2 Links and Symbolic Links functions

```
(posix-make-link old-path new-path)
```

STKLOS
procedure

Create a hard link with the filename "new-path" that points to the file named "old-path". Return **#t** in case of success.

```
(posix-delete-link old-path new-path)
```

STKLOS
procedure

Create a symbolic link with the filename "new-path" that points to the file named "old-path". Return **#t** in case of success.

```
(posix-file-is-symbolic-link? path)
```

STKLOS
procedure

Return **#t** if "path" designates a symbolic link and **#f** otherwise.

```
(posix-read-symbolic-link path)
```

STKLOS
procedure

Return a string containing the filename to which the symbolic link "path" points to.

3.3 Error management

When an error occurs POSIX sets the `errno` global variable to a return code describing the error. This error code is available from the `posix-errno` Scheme parameter.

```
(posix-errno)
(posix-errno value)
```

STKLOS
procedure

Return the value of the POSIX `errno` variable. When a value is passed to the `posix-errno`, this value is set to the POSIX `errno` variable.

```
(posix-error)
```

STKLOS
procedure

Return a string describing the last POSIX error detected.

Furthermore, the POSIX extension defines also the following constants for representing POSIX error numbers.

posix/ E2BIG	posix/ EACCES	posix/ EADDRINUSE
posix/ EADDRNOTAVAIL	posix/ EAFNOSUPPORT	posix/ EAGAIN
posix/ EALREADY	posix/ EBADF	posix/ EBADMSG
posix/ EBUSY	posix/ ECANCELED	posix/ ECHILD
posix/ ECONNABORTED	posix/ ECONNREFUSED	posix/ ECONNRESET
posix/ EDEADLK	posix/ EDESTADDRREQ	posix/ EDOM
posix/ EDQUOT	posix/ EEXIST	posix/ EFAULT
posix/ EFBIG	posix/ EHOSTUNREACH	posix/ EIDRM
posix/ EILSEQ	posix/ EINPROGRESS	posix/ EINTR
posix/ EINVAL	posix/ EIO	posix/ EISCONN
posix/ EISDIR	posix/ ELOOP	posix/ EMFILE
posix/ EMLINK	posix/ EMSGSIZE	posix/ EMULTIHOP
posix/ ENAMETOOLONG	posix/ ENETDLOWN	posix/ ENETRESET
posix/ ENETUNREACH	posix/ ENFILE	posix/ ENOBUFS
posix/ ENODATA	posix/ ENODEV	posix/ ENOENT
posix/ ENOEXEC	posix/ ENOLCK	posix/ ENOLINK
posix/ ENOMEM	posix/ ENOMSG	posix/ ENOPROTOOPT
posix/ ENOSPC	posix/ ENOSR	posix/ ENOSTR
posix/ ENOSYS	posix/ ENOTCONN	posix/ ENOTDLIR
posix/ ENOTEMPTY	posix/ ENOTSOCK	posix/ ENOTSUP
posix/ ENOTTY	posix/ ENXIO	posix/ EOPNOTSUPP
posix/ EOVERFLOW	posix/ EPERM	posix/ EPIPE
posix/ EPROTO	posix/ EPROTONOSUPPORT	posix/ EPROTOTYPE
posix/ ERANGE	posix/ EROFS	posix/ ESPIPE
posix/ ESRCH	posix/ ESTALE	posix/ ETIME
posix/ ETIMEDOUT	posix/ ETXTBSY	posix/ EWOULDBLOCK
posix/ EXDEV		

3.4 System Informations

`(posix-file-informations path)`

STKLOS
procedure

Return a keyword list describing the cureent status of file "path".

```
(posix-file-informations "/")
⇒ (:dev 770 :ino 2 :mode 16877 :nlink 17 :uid 0 :gid
0
      :size 4096 :atime 1115017161 :mtime 1134028671
      :ctime 1134028671)
```

The following constants are defined to analyse the bits of the **mode** component returned by `posix-file-informations`:

posix/ IFSOCK	posix/ IFLNK	posix/ IFREG	posix/ IFBLK
posix/ IFDIR	posix/ IFCHR	posix/ IFIFO	posix/ ISUID
posix/ ISGID	posix/ ISVTX	posix/ IRWXU	posix/ IRUSR
posix/ IWUSR	posix/ IXUSR	posix/ IRWXG	posix/ IRGRP
posix/ IWGRP	posix/ IXGRP	posix/ IRWXO	posix/ IROTH
posix/ IWOTH	posix/ IXOTH		

`(posix-file-permissions path)`

STKLOS
procedure

Return the permissions flag associated to "path" if the file is accesible. Return **#f** othterwise.

`(posix-uid->string uid)`

STKLOS
procedure

Return the login name associated to uid.

`(posix-uid->string logname)`

STKLOS
procedure

Return the uid associated to the logname.

`(posix-gid->string uid)`

STKLOS
procedure

Return the group name associated to gid.

`(posix-uid->string grpname)`

STKLOS
procedure

Return the gid associated to the grpname.

`(posix-user-id)`

STKLOS
procedure

Return the user id of the running process.

`(posix-group-id)`STKLOS
procedure

Return the group id of the running process.

`(posix-effective-user-id)`STKLOS
procedure

Return the effective user id of the running process.

`(posix-effective-group-id)`STKLOS
procedure

Return the effective group id of the running process.

4 Example

Here is a simple program using the STKLOS Posix extension. It displays some information of the path given as parameter

```
(require "posix")

(define (usage)
  (format (current-error-port) "Usage: ~A file\n" (program-name))
  (exit 1))

(define (bit-set? bit in)
  (= (bit-and bit in) bit))

(define (main args)
  (if (not (= (length args) 2))
      (usage)
      (let ((stat (posix-file-informations (cadr args))))
        (if (not stat)
            (format (current-error-port) "Return code ~S. Message ~S\n"
                    (posix-errno) (posix-error))
            (let ((mode (key-get stat :mode)))
              (format #t "Type of file: ~A\n"
                      (cond
                       ((bit-set? posix/IFSOCK mode) "socket")
                       ((bit-set? posix/IFLNK mode) "symbolic link")
                       ((bit-set? posix/IFREG mode) "regular file")
                       ((bit-set? posix/IFBLK mode) "block device")
                       ((bit-set? posix/IFCHR mode) "character device")
                       ((bit-set? posix/IFDIR mode) "directory")))
              (format #t "Owner: ~A\n"
                      (posix-uid->string (key-get stat :uid)))
              (format #t "Group: ~A\n"
                      (posix-gid->string (key-get stat :gid))))))))))
```