Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'fts_close.3' command

$ man fts_close.3

FTS(3) Linux Programmer's Manual FTS(3)

NAME

fts, fts_open, fts_read, fts_children, fts_set, fts_close - traverse a file hierarchy

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fts.h>
FTS *fts_open(char * const *path_argv, int options,
        int (*compar)(const FTSENT **, const FTSENT **));
FTSENT *fts_read(FTS *ftsp);
FTSENT *fts_children(FTS *ftsp, int instr);
int fts_set(FTS *ftsp, FTSENT *f, int instr);
int fts_close(FTS *ftsp);
```

DESCRIPTION

The fts functions are provided for traversing file hierarchies. A sim? ple overview is that the fts_open() function returns a "handle" (of type FTS *) that refers to a file hierarchy "stream". This handle is then supplied to the other fts functions. The function fts_read() re? turns a pointer to a structure describing one of the files in the file hierarchy. The function fts_children() returns a pointer to a linked list of structures, each of which describes one of the files contained in a directory in the hierarchy.

In general, directories are visited two distinguishable times; in pre? order (before any of their descendants are visited) and in postorder (after all of their descendants have been visited). Files are visited once. It is possible to walk the hierarchy "logically" (visiting the files that symbolic links point to) or physically (visiting the sym? bolic links themselves), order the walk of the hierarchy or prune and/or revisit portions of the hierarchy.

Two structures (and associated types) are defined in the include file <fts.h>. The first type is FTS, the structure that represents the file hierarchy itself. The second type is FTSENT, the structure that repre? sents a file in the file hierarchy. Normally, an FTSENT structure is returned for every file in the file hierarchy. In this manual page, "file" and "FTSENT structure" are generally interchangeable.

The FTSENT structure contains fields describing a file. The structure contains at least the following fields (there are additional fields that should be considered private to the implementation):

```
typedef struct _ftsent {
    unsigned short  fts_info;     /* flags for FTSENT structure */
    char           *fts_accpath;  /* access path */
    char           *fts_path;     /* root path */
    short           fts_pathlen;  /* strlen(fts_path) +
                                     strlen(fts_name) */
    char           *fts_name;     /* filename */
    short           fts_namelen;  /* strlen(fts_name) */
    short           fts_level;    /* depth (-1 to N) */
    int             fts_errno;    /* file errno */
    long            fts_number;   /* local numeric value */
    void           *fts_pointer;  /* local address value */
    struct _ftsent *fts_parent;   /* parent directory */
    struct _ftsent *fts_link;     /* next file structure */
    struct _ftsent *fts_cycle;    /* cycle structure */
    struct stat    *fts_statp;    /* stat(2) information */
} FTSENT;
```

These fields are defined as follows:

fts_info

One of the following values describing the returned FTSENT structure and the file it represents. With the exception of di? rectories without errors (FTS_D), all of these entries are ter? minal, that is, they will not be revisited, nor will any of their descendants be visited.

FTS_D A directory being visited in preorder.

FTS_DC A directory that causes a cycle in the tree. (The fts_cycle field of the FTSENT structure will be filled in as well.)

FTS_DEFAULT

Any FTSENT structure that represents a file type not ex? plicitly described by one of the other fts_info values.

FTS_DNR

A directory which cannot be read. This is an error re? turn, and the fts_errno field will be set to indicate what caused the error.

FTS_DOT

A file named "." or ".." which was not specified as a filename to fts_open() (see FTS_SEEDOT).

FTS_DP A directory being visited in postorder. The contents of the FTSENT structure will be unchanged from when it was returned in preorder, that is, with the fts_info field set to FTS_D.

FTS_ERR

This is an error return, and the fts_errno field will be set to indicate what caused the error.

FTS_F A regular file.

FTS_NS A file for which no stat(2) information was available.

The contents of the fts_statp field are undefined. This is an error return, and the fts_errno field will be set to indicate what caused the error.

FTS_NSOK

A file for which no stat(2) information was requested.

The contents of the fts_statp field are undefined.

FTS_SL A symbolic link.

FTS_SLNONE

A symbolic link with a nonexistent target. The contents

of the fts_statp field reference the file characteristic

information for the symbolic link itself.

fts_accpath

A path for accessing the file from the current directory.

fts_path

The path for the file relative to the root of the traversal.

This path contains the path specified to fts_open() as a prefix.

fts_pathlen

The sum of the lengths of the strings referenced by fts_path and

fts_name.

fts_name

The name of the file.

fts_namelen

The length of the string referenced by fts_name.

fts_level

The depth of the traversal, numbered from -1 to N, where this

file was found. The FTSENT structure representing the parent of

the starting point (or root) of the traversal is numbered -1,

and the FTSENT structure for the root itself is numbered 0.

fts_errno

If fts_children() or fts_read() returns an FTSENT structure

whose fts_info field is set to FTS_DNR, FTS_ERR, or FTS_NS, the

fts_errno field contains the error number (i.e., the errno

value) specifying the cause of the error. Otherwise, the con?

tents of the fts_errno field are undefined.

fts_number

This field is provided for the use of the application program

and is not modified by the fts functions. It is initialized to 0.

fts_pointer

This field is provided for the use of the application program and is not modified by the fts functions. It is initialized to NULL.

fts_parent

A pointer to the FTSENT structure referencing the file in the hierarchy immediately above the current file, that is, the directory of which this file is a member. A parent structure for the initial entry point is provided as well, however, only the fts_level, fts_number, and fts_pointer fields are guaranteed to be initialized.

fts_link

Upon return from the fts_children() function, the fts_link field points to the next structure in the NULL-terminated linked list of directory members. Otherwise, the contents of the fts_link field are undefined.

fts_cycle

If a directory causes a cycle in the hierarchy (see FTS_DC), either because of a hard link between two directories, or a symbolic link pointing to a directory, the fts_cycle field of the structure will point to the FTSENT structure in the hierarchy that references the same file as the current FTSENT structure. Otherwise, the contents of the fts_cycle field are undefined.

fts_statp

A pointer to stat(2) information for the file.

A single buffer is used for all of the paths of all of the files in the file hierarchy. Therefore, the fts_path and fts_accpath fields are guaranteed to be null-terminated only for the file most recently returned by fts_read(). To use these fields to reference any files represented by other FTSENT structures will require that the path buffer be modified using the information contained in that FTSENT structure's

fts_pathlen field. Any such modifications should be undone before fur?
ther calls to fts_read() are attempted. The fts_name field is always
null-terminated.

fts_open()

The fts_open() function takes a pointer to an array of character point?
ers naming one or more paths which make up a logical file hierarchy to
be traversed. The array must be terminated by a null pointer.

There are a number of options, at least one of which (either FTS_LOGI?
CAL or FTS_PHYSICAL) must be specified. The options are selected by
ORing the following values:

FTS_COMFOLLOW

This option causes any symbolic link specified as a root path to
be followed immediately whether or not FTS_LOGICAL is also spec?
ified.

FTS_LOGICAL

This option causes the fts routines to return FTSENT structures
for the targets of symbolic links instead of the symbolic links
themselves. If this option is set, the only symbolic links for
which FTSENT structures are returned to the application are
those referencing nonexistent files. Either FTS_LOGICAL or
FTS_PHYSICAL must be provided to the fts_open() function.

FTS_NOCHDIR

As a performance optimization, the fts functions change directo?
ries as they walk the file hierarchy. This has the side-effect
that an application cannot rely on being in any particular di?
rectory during the traversal. The FTS_NOCHDIR option turns off
this optimization, and the fts functions will not change the
current directory. Note that applications should not themselves
change their current directory and try to access files unless
FTS_NOCHDIR is specified and absolute pathnames were provided as
arguments to fts_open().

FTS_NOSTAT

By default, returned FTSENT structures reference file character?

istic information (the statp field) for each file visited. This option relaxes that requirement as a performance optimization, allowing the fts functions to set the fts_info field to FTS_NSOK and leave the contents of the statp field undefined.

FTS_PHYSICAL

This option causes the fts routines to return FTSENT structures for symbolic links themselves instead of the target files they point to. If this option is set, FTSENT structures for all sym? bolic links in the hierarchy are returned to the application. Either FTS_LOGICAL or FTS_PHYSICAL must be provided to the fts_open() function.

FTS_SEEDOT

By default, unless they are specified as path arguments to fts_open(), any files named "." or ".." encountered in the file hierarchy are ignored. This option causes the fts routines to return FTSENT structures for them.

FTS_XDEV

This option prevents fts from descending into directories that have a different device number than the file from which the de? scent began.

The argument compar() specifies a user-defined function which may be used to order the traversal of the hierarchy. It takes two pointers to pointers to FTSENT structures as arguments and should return a negative value, zero, or a positive value to indicate if the file referenced by its first argument comes before, in any order with respect to, or af? ter, the file referenced by its second argument. The fts_accpath, fts_path, and fts_pathlen fields of the FTSENT structures may never be used in this comparison. If the fts_info field is set to FTS_NS or FTS_NSOK, the fts_statp field may not either. If the compar() argument is NULL, the directory traversal order is in the order listed in path_argv for the root paths, and in the order listed in the directory for everything else.

The fts_read() function returns a pointer to an FTSENT structure de?
scribing a file in the hierarchy. Directories (that are readable and
do not cause cycles) are visited at least twice, once in preorder and
once in postorder. All other files are visited at least once. (Hard
links between directories that do not cause cycles or symbolic links to
symbolic links may cause files to be visited more than once, or direc?
tories more than twice.)

If all the members of the hierarchy have been returned, fts_read() re?
turns NULL and sets the external variable errno to 0. If an error un?
related to a file in the hierarchy occurs, fts_read() returns NULL and
sets errno appropriately. If an error related to a returned file oc?
curs, a pointer to an FTSENT structure is returned, and errno may or
may not have been set (see fts_info).

The FTSENT structures returned by fts_read() may be overwritten after a
call to fts_close() on the same file hierarchy stream, or, after a call
to fts_read() on the same file hierarchy stream unless they represent a
file of type directory, in which case they will not be overwritten un?
til after a call to fts_read() after the FTSENT structure has been re?
turned by the function fts_read() in postorder.

fts_children()

The fts_children() function returns a pointer to an FTSENT structure
describing the first entry in a NULL-terminated linked list of the
files in the directory represented by the FTSENT structure most re?
cently returned by fts_read(). The list is linked through the fts_link
field of the FTSENT structure, and is ordered by the user-specified
comparison function, if any. Repeated calls to fts_children() will re-
create this linked list.

As a special case, if fts_read() has not yet been called for a hierar?
chy, fts_children() will return a pointer to the files in the logical
directory specified to fts_open(), that is, the arguments specified to
fts_open(). Otherwise, if the FTSENT structure most recently returned
by fts_read() is not a directory being visited in preorder, or the di?
rectory does not contain any files, fts_children() returns NULL and

sets errno to zero. If an error occurs, fts_children() returns NULL and sets errno appropriately.

The FTSENT structures returned by fts_children() may be overwritten af? ter a call to fts_children(), fts_close(), or fts_read() on the same file hierarchy stream.

The instr argument is either zero or the following value:

FTS_NAMEONLY

Only the names of the files are needed. The contents of all the fields in the returned linked list of structures are undefined with the exception of the fts_name and fts_namelen fields.

fts_set()

The function fts_set() allows the user application to determine further processing for the file f of the stream ftsp. The fts_set() function returns 0 on success, and -1 if an error occurs.

The instr argument is either 0 (meaning "do nothing") or one of the following values:

FTS_AGAIN

Revisit the file; any file type may be revisited. The next call to fts_read() will return the referenced file. The fts_stat and fts_info fields of the structure will be reinitialized at that time, but no other fields will have been changed. This option is meaningful only for the most recently returned file from fts_read(). Normal use is for postorder directory visits, where it causes the directory to be revisited (in both preorder and postorder) as well as all of its descendants.

FTS_FOLLOW

The referenced file must be a symbolic link. If the referenced file is the one most recently returned by fts_read(), the next call to fts_read() returns the file with the fts_info and fts_statp fields reinitialized to reflect the target of the sym? bolic link instead of the symbolic link itself. If the file is one of those most recently returned by fts_children(), the fts_info and fts_statp fields of the structure, when returned by

fts_read(), will reflect the target of the symbolic link instead of the symbolic link itself. In either case, if the target of the symbolic link does not exist, the fields of the returned structure will be unchanged and the fts_info field will be set to FTS_SLNONE.

If the target of the link is a directory, the preorder return, followed by the return of all of its descendants, followed by a postorder return, is done.

FTS_SKIP

No descendants of this file are visited. The file may be one of those most recently returned by either fts_children() or fts_read().

fts_close()

The fts_close() function closes the file hierarchy stream referred to by ftsp and restores the current directory to the directory from which fts_open() was called to open ftsp. The fts_close() function returns 0 on success, and -1 if an error occurs.

ERRORS

The function fts_open() may fail and set errno for any of the errors specified for open(2) and malloc(3).

The function fts_close() may fail and set errno for any of the errors specified for chdir(2) and close(2).

The functions fts_read() and fts_children() may fail and set errno for any of the errors specified for chdir(2), malloc(3), opendir(3), read? dir(3), and stat(2).

In addition, fts_children(), fts_open(), and fts_set() may fail and set errno as follows:

EINVAL options or instr was invalid.

VERSIONS

These functions are available in Linux since glibc2.

ATTRIBUTES

For an explanation of the terms used in this section, see at? tributes(7).

??

?Interface ? Attribute ? Value ?

??

?fts_open(), fts_set(), fts_close() ? Thread safety ? MT-Safe ?

??

?fts_read(), fts_children() ? Thread safety ? MT-Unsafe ?

??

CONFORMING TO

4.4BSD.

BUGS

In versions of glibc before 2.23, all of the APIs described in this man page are not safe when compiling a program using the LFS APIs (e.g., when compiling with -D_FILE_OFFSET_BITS=64).

SEE ALSO

find(1), chdir(2), stat(2), ftw(3), qsort(3)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.