NAME

fts - traverse a file hierarchy

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <fts.h>

*FTS **
fts_open(*char * const *path_argv, int options,*
*int (*compar)(const FTSENT * const *, const FTSENT * const *)*);

*FTSENT **
fts_read(*FTS *ftsp*);

*FTSENT **
fts_children(*FTS *ftsp, int options*);

int
fts_set(*FTS *ftsp, FTSENT *f, int options*);

void
fts_set_clientptr(*FTS *ftsp, void *clientdata*);

*void **
fts_get_clientptr(*FTS *ftsp*);

*FTS **
fts_get_stream(*FTSENT *f*);

int
fts_close(*FTS *ftsp*);

DESCRIPTION

The **fts** functions are provided for traversing UNIX file hierarchies. A simple overview is that the **fts_open**() function returns a "handle" on a file hierarchy, which is then supplied to the other **fts** functions. The function **fts_read**() returns 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 post-order (after all of their descendants have been visited). Files are visited once. It is possible to walk the hierarchy "logically" (ignoring symbolic links) or physically (visiting symbolic links), order the walk of the hierarchy or prune and/or re-visit portions of the hierarchy.

Two structures are defined (and typedef'd) in the include file <*fts.h*>. The first is *FTS*, the structure that represents the file hierarchy itself. The second is *FTSENT*, the structure that represents 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 *FTS* structure contains space for a single pointer, which may be used to store application data or per-hierarchy state. The **fts_set_clientptr**() and **fts_get_clientptr**() functions may be used to set and retrieve this pointer. This is likely to be useful only when accessed from the sort comparison function, which can determine the original *FTS* stream of its arguments using the **fts_get_stream**() function. The two get functions are also available as macros of the same name.

The *FTSENT* structure contains at least the following fields, which are described in greater detail below:

```
typedef struct _ftsent {
        int fts_info;                   /* status for FTSENT structure */
        char *fts_accpath;      /* access path */
        char *fts_path;                 /* root path */
        size_t fts_pathlen;     /* strlen(fts_path) */
        char *fts_name;                 /* file name */
        size_t fts_namelen;     /* strlen(fts_name) */
        long fts_level;                 /* depth (-1 to N) */
        int fts_errno;                  /* file errno */
        long 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 directories without errors (FTS_D), all of these entries are terminal, that is, they will not be revisited, nor will any of their descendants be visited.

FTS_D A directory being visited in pre-order.

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 explicitly described by one of the other *fts_info* values.

FTS_DNR A directory which cannot be read. This is an error return, 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 file name to **fts_open**() (see FTS_SEEDOT).

FTS_DP A directory being visited in post-order. The contents of the *FTSENT* structure will be unchanged from when the directory was visited in pre-order, except for the *fts_info* field.

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 non-existent 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 length of the string referenced by *fts_path*.

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 FTS_ROOTPARENTLEVEL (-1), and the *FTSENT* structure for the root itself is numbered FTS_ROOTLEVEL (0).

fts_errno Upon return of a *FTSENT* structure from the **fts_children**() or **fts_read**() functions, with its *fts_info* field set to FTS_DNR, FTS_ERR or FTS_NS, the *fts_errno* field contains the value of the external variable *errno* specifying the cause of the error. Otherwise, the contents 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, i.e., 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 NUL-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 further calls to **fts_read**() are attempted. The *fts_name* field is always NUL-terminated.

FTS_OPEN

The **fts_open**() function takes a pointer to an array of character pointers 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_LOGICAL or FTS_PHYSICAL) must be specified. The options are selected by *or*'ing 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 specified.

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 non-existent files. Either FTS_LOGICAL or FTS_PHYSICAL *must* be provided to the **fts_open**() function.

FTS_NOCHDIR
: To allow descending to arbitrary depths (independent of {PATH_MAX}) and improve performance, the **fts** functions change directories as they walk the file hierarchy. This has the side-effect that an application cannot rely on being in any particular directory during the traversal. The FTS_NOCHDIR option turns off this feature, 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 characteristic 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 symbolic 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 descent 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 after, 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.

FTS_READ

The **fts_read**() function returns a pointer to an *FTSENT* structure describing a file in the hierarchy. Directories (that are readable and do not cause cycles) are visited at least twice, once in pre-order and once in post-order. 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 directories more than twice.)

If all the members of the hierarchy have been returned, **fts_read**() returns NULL and sets the external variable *errno* to 0. If an error unrelated to a file in the hierarchy occurs, **fts_read**() returns NULL and sets *errno* appropriately. If an error related to a returned file occurs, 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 until after a call to **fts_read**() after the *FTSENT* structure has been returned by the function **fts_read**() in post-order.

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 recently 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 recreate this linked list.

As a special case, if **fts_read**() has not yet been called for a hierarchy, **fts_children**() will return a pointer to the files in the logical directory specified to **fts_open**(), i.e., the arguments specified to **fts_open**(). Otherwise, if the *FTSENT* structure most recently returned by **fts_read**() is not a directory being visited in pre-order, or the directory 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 after a call to **fts_children**(), **fts_close**() or **fts_read**() on the same file hierarchy stream.

Option may be set to 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. *Option* must be set to one of the following values:

FTS_AGAIN Re-visit the file; any file type may be re-visited. 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 post-order directory visits, where it causes the directory to be re-visited (in both pre and post-order) 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 symbolic 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 pre-order return, followed by the return of all of its descendants, followed by a post-order 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 a file hierarchy stream *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 the library functions open(2) and malloc(3).

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

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

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

[EINVAL] The options were invalid, or the list were empty.

SEE ALSO

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

HISTORY

The **fts** interface was first introduced in 4.4BSD. The **fts_get_clientptr**(), **fts_get_stream**(), and **fts_set_clientptr**() functions were introduced in FreeBSD 5.0, principally to provide for alternative interfaces to the **fts** functionality using different data structures.

BUGS

The **fts_open**() function will automatically set the FTS_NOCHDIR option if the FTS_LOGICAL option is provided, or if it cannot open(2) the current directory.