NAME

archive_read_disk_new, **archive_read_disk_open**, **archive_read_disk_open_w**, **archive_read_disk_set_behavior**, **archive_read_disk_set_symlink_logical**, **archive_read_disk_set_symlink_physical**, **archive_read_disk_set_symlink_hybrid**, **archive_read_disk_entry_from_file**, **archive_read_disk_gname**, **archive_read_disk_uname**, **archive_read_disk_set_uname_lookup**, **archive_read_disk_set_gname_lookup**, **archive_read_disk_set_standard_lookup**, **archive_read_disk_descend**, **archive_read_disk_can_descend**, **archive_read_disk_current_filesystem**, **archive_read_disk_current_filesystem_is_synthetic**, **archive_read_disk_current_filesystem_is_remote**, **archive_read_disk_set_matching**, **archive_read_disk_set_metadata_filter_callback**, - functions for reading objects from disk

LIBRARY

Streaming Archive Library (libarchive, -larchive)

SYNOPSIS

#include <archive.h>

*struct archive **
archive_read_disk_new(*void*);

int
archive_read_disk_open(*struct archive **, *const char **);

int
archive_read_disk_open_w(*struct archive **, *const wchar_t **);

int
archive_read_disk_set_behavior(*struct archive **, *int*);

int
archive_read_disk_set_symlink_logical(*struct archive **);

int
archive_read_disk_set_symlink_physical(*struct archive **);

int
archive_read_disk_set_symlink_hybrid(*struct archive **);

*const char **
archive_read_disk_gname(*struct archive **, *gid_t*);

*const char **
archive_read_disk_uname(*struct archive *, uid_t*);

int
archive_read_disk_set_gname_lookup(*struct archive *, void *, const char *(*lookup)(void *, gid_t), void (*cleanup)(void *)*);

int
archive_read_disk_set_uname_lookup(*struct archive *, void *, const char *(*lookup)(void *, uid_t), void (*cleanup)(void *)*);

int
archive_read_disk_set_standard_lookup(*struct archive **);

int
archive_read_disk_entry_from_file(*struct archive *, struct archive_entry *, int fd, const struct stat **);

int
archive_read_disk_descend(*struct archive **);

int
archive_read_disk_can_descend(*struct archive **);

int
archive_read_disk_current_filesystem(*struct archive **);

int
archive_read_disk_current_filesystem_is_synthetic(*struct archive **);

int
archive_read_disk_current_filesystem_is_remote(*struct archive **);

int
archive_read_disk_set_matching(*struct archive *, struct archive **,
*void (*excluded_func)(struct archive *, void *, struct archive entry *), void **);

int
archive_read_disk_set_metadata_filter_callback(*struct archive **,
*int (*metadata_filter_func)(struct archive *, void*, struct archive_entry *), void **);

DESCRIPTION

These functions provide an API for reading information about objects on disk. In particular, they provide an interface for populating struct archive_entry objects.

archive_read_disk_new()

Allocates and initializes a struct archive object suitable for reading object information from disk.

archive_read_disk_open()

Opens the file or directory from the given path and prepares the struct archive to read it from disk.

archive_read_disk_open_w()

Opens the file or directory from the given path as a wide character string and prepares the struct archive to read it from disk.

archive_read_disk_set_behavior()

Configures various behavior options when reading entries from disk. The flags field consists of a bitwise OR of one or more of the following values:

ARCHIVE_READDISK_HONOR_NODUMP

Skip files and directories with the nodump file attribute (file flag) set. By default, the nodump file attribute is ignored.

ARCHIVE_READDISK_MAC_COPYFILE

Mac OS X specific. Read metadata (ACLs and extended attributes) with copyfile(3). By default, metadata is read using copyfile(3).

ARCHIVE_READDISK_NO_ACL

Do not read Access Control Lists. By default, ACLs are read from disk.

ARCHIVE_READDISK_NO_FFLAGS

Do not read file attributes (file flags). By default, file attributes are read from disk. See chattr(1) (Linux) or chflags(1) (FreeBSD, Mac OS X) for more information on file attributes.

ARCHIVE_READDISK_NO_TRAVERSE_MOUNTS

Do not traverse mount points. By default, mount points are traversed.

ARCHIVE_READDISK_NO_XATTR

Do not read extended file attributes (xattrs). By default, extended file attributes are read from disk. See xattr(7) (Linux), xattr(2) (Mac OS X), or getextattr(8) (FreeBSD) for more information on extended file attributes.

ARCHIVE_READDISK_RESTORE_ATIME

Restore access time of traversed files. By default, access time of traversed files is not restored.

ARCHIVE_READDISK_NO_SPARSE

Do not read sparse file information. By default, sparse file information is read from disk.

archive_read_disk_set_symlink_logical(), **archive_read_disk_set_symlink_physical**(), **archive_read_disk_set_symlink_hybrid**()
This sets the mode used for handling symbolic links. The "logical" mode follows all symbolic links. The "physical" mode does not follow any symbolic links. The "hybrid" mode currently behaves identically to the "logical" mode.

archive_read_disk_gname(), **archive_read_disk_uname**()
Returns a user or group name given a gid or uid value. By default, these always return a NULL string.

archive_read_disk_set_gname_lookup(), **archive_read_disk_set_uname_lookup**()
These allow you to override the functions used for user and group name lookups. You may also provide a void * pointer to a private data structure and a cleanup function for that data. The cleanup function will be invoked when the struct archive object is destroyed or when new lookup functions are registered.

archive_read_disk_set_standard_lookup()
This convenience function installs a standard set of user and group name lookup functions. These functions use getpwuid(3) and getgrgid(3) to convert ids to names, defaulting to NULL if the names cannot be looked up. These functions also implement a simple memory cache to reduce the number of calls to getpwuid(3) and getgrgid(3).

archive_read_disk_entry_from_file()
Populates a struct archive_entry object with information about a particular file. The archive_entry object must have already been created with archive_entry_new(3) and at least one of the source path or path fields must already be set. (If both are set, the source path will be used.)

Information is read from disk using the path name from the struct archive_entry object. If a file descriptor is provided, some information will be obtained using that file descriptor, on platforms that support the appropriate system calls.

If a pointer to a struct stat is provided, information from that structure will be used instead of reading from the disk where appropriate. This can provide performance benefits in scenarios where struct stat information has already been read from the disk as a side effect of some other operation. (For example, directory traversal libraries often provide this information.)

Where necessary, user and group ids are converted to user and group names using the currently-

registered lookup functions above. This affects the file ownership fields and ACL values in the struct archive_entry object.

archive_read_disk_descend()
If the current entry can be descended, this function will mark the directory as the next entry for archive_read_header(3) to visit.

archive_read_disk_can_descend()
Returns 1 if the current entry is an unvisited directory and 0 otherwise.

archive_read_disk_current_filesystem()
Returns the index of the most recent filesystem entry that has been visited through archive_read_disk

archive_read_disk_current_filesystem_is_synthetic()
Returns 1 if the current filesystem is a virtual filesystem. Returns 0 if the current filesystem is not a virtual filesystem. Returns -1 if it is unknown.

archive_read_disk_current_filesystem_is_remote()
Returns 1 if the current filesystem is a remote filesystem. Returns 0 if the current filesystem is not a remote filesystem. Returns -1 if it is unknown.

archive_read_disk_set_matching()
Allows the caller to set struct archive *_ma to compare each entry during archive_read_header(3) calls. If matched based on calls to archive_match_path_excluded, archive_match_time_excluded, or archive_match_owner_excluded, then the callback function specified by the _excluded_func parameter will execute. This function will receive data provided to the fourth parameter, void *_client_data.

archive_read_disk_set_metadata_filter_callback()
Allows the caller to set a callback function during calls to archive_read_header(3) to filter out metadata for each entry. The callback function receives the struct archive object, void* custom filter data, and the struct archive_entry. If the callback function returns an error, ARCHIVE_RETRY will be returned and the entry will not be further processed.

More information about the *struct archive* object and the overall design of the library can be found in the libarchive(3) overview.

EXAMPLES

The following illustrates basic usage of the library by showing how to use it to copy an item on disk into an archive.

```
void
file_to_archive(struct archive *a, const char *name)
{
  char buff[8192];
  size_t bytes_read;
  struct archive *ard;
  struct archive_entry *entry;
  int fd;

  ard = archive_read_disk_new();
  archive_read_disk_set_standard_lookup(ard);
  entry = archive_entry_new();
  fd = open(name, O_RDONLY);
  if (fd < 0)
    return;
  archive_entry_copy_pathname(entry, name);
  archive_read_disk_entry_from_file(ard, entry, fd, NULL);
  archive_write_header(a, entry);
  while ((bytes_read = read(fd, buff, sizeof(buff))) > 0)
    archive_write_data(a, buff, bytes_read);
  archive_write_finish_entry(a);
  archive_read_free(ard);
  archive_entry_free(entry);
}
```

RETURN VALUES

Most functions return **ARCHIVE_OK** (zero) on success, or one of several negative error codes for errors. Specific error codes include: **ARCHIVE_RETRY** for operations that might succeed if retried, **ARCHIVE_WARN** for unusual conditions that do not prevent further operations, and **ARCHIVE_FATAL** for serious errors that make remaining operations impossible.

archive_read_disk_new() returns a pointer to a newly-allocated struct archive object or NULL if the allocation failed for any reason.

archive_read_disk_gname() and **archive_read_disk_uname**() return const char * pointers to the textual name or NULL if the lookup failed for any reason. The returned pointer points to internal storage that may be reused on the next call to either of these functions; callers should copy the string if they need to continue accessing it.

ERRORS

Detailed error codes and textual descriptions are available from the **archive_errno**() and **archive_error_string**() functions.

SEE ALSO

tar(1), archive_read(3), archive_util(3), archive_write(3), archive_write_disk(3), libarchive(3)

HISTORY

The **libarchive** library first appeared in FreeBSD 5.3. The **archive_read_disk** interface was added to **libarchive 2.6** and first appeared in FreeBSD 8.0.

AUTHORS

The **libarchive** library was written by Tim Kientzle <kientzle@FreeBSD.org>.

BUGS

The "standard" user name and group name lookup functions are not the defaults because getgrgid(3) and getpwuid(3) are sometimes too large for particular applications. The current design allows the application author to use a more compact implementation when appropriate.

The full list of metadata read from disk by **archive_read_disk_entry_from_file**() is necessarily system-dependent.

The **archive_read_disk_entry_from_file**() function reads as much information as it can from disk. Some method should be provided to limit this so that clients who do not need ACLs, for instance, can avoid the extra work needed to look up such information.

This API should provide a set of methods for walking a directory tree. That would make it a direct parallel of the archive_read(3) API. When such methods are implemented, the "hybrid" symbolic link mode will make sense.