NAME

archive_entry_gid, **archive_entry_set_gid**, **archive_entry_uid**, **archive_entry_set_uid**, **archive_entry_perm**, **archive_entry_set_perm**, **archive_entry_strmode**, **archive_entry_uname**, **archive_entry_uname_w**, **archive_entry_set_uname**, **archive_entry_copy_uname**, **archive_entry_copy_uname_w**, **archive_entry_update_uname_utf8**, **archive_entry_gname**, **archive_entry_gname_w**, **archive_entry_set_gname**, **archive_entry_copy_gname**, **archive_entry_copy_gname_w**, **archive_entry_update_gname_utf8**, **archive_entry_fflags**, **archive_entry_fflags_text**, **archive_entry_set_fflags**, **archive_entry_copy_fflags_text**, **archive_entry_copy_fflags_text_w** - functions for manipulating ownership and permissions in archive entry descriptions

LIBRARY

Streaming Archive Library (libarchive, -larchive)

SYNOPSIS

#include <archive_entry.h>

gid_t
archive_entry_gid(*struct archive_entry *a*);

void
archive_entry_set_gid(*struct archive_entry *a*, *gid_t gid*);

uid_t
archive_entry_uid(*struct archive_entry *a*);

void
archive_entry_set_uid(*struct archive_entry *a*, *uid_t uid*);

mode_t
archive_entry_perm(*struct archive_entry *a*);

void
archive_entry_set_perm(*struct archive_entry *a*, *mode_t mode*);

*const char **
archive_entry_strmode(*struct archive_entry *a*);

*const char **
archive_entry_gname(*struct archive_entry *a*);

*const wchar_t **
archive_entry_gname_w(*struct archive_entry *a*);

void
archive_entry_set_gname(*struct archive_entry *a*, *const char *a*);

void
archive_entry_copy_gname(*struct archive_entry *a*, *const char *name*);

void
archive_entry_copy_gname_w(*struct archive_entry *a*, *const wchar_t *name*);

int
archive_entry_update_gname_utf8(*struct archive_entry *a*, *const char *name*);

*const char **
archive_entry_uname(*struct archive_entry *a*);

*const wchar_t **
archive_entry_uname_w(*struct archive_entry *a*);

void
archive_entry_set_uname(*struct archive_entry *a*, *const char *name*);

void
archive_entry_copy_uname(*struct archive_entry *a*, *const char *name*);

void
archive_entry_copy_uname_w(*struct archive_entry *a*, *const wchar_t *name*);

int
archive_entry_update_uname_utf8(*struct archive_entry *a*, *const char *name*);

void
archive_entry_fflags(*struct archive_entry *a*, *unsigned long *set_bits*, *unsigned long *clear_bits*);

*const char **
archive_entry_fflags_text(*struct archive_entry *a*);

void

archive_entry_set_fflags(*struct archive_entry *a, unsigned long set_bits, unsigned long clear_bits*);

*const char **
archive_entry_copy_fflags_text(*struct archive_entry *a, const char *text*);

*const wchar_t **
archive_entry_copy_fflags_text_w(*struct archive_entry *a, const wchar_t *text*);

DESCRIPTION

User id, group id and mode

The functions **archive_entry_uid**(), **archive_entry_gid**(), and **archive_entry_perm**() can be used to extract the user id, group id and permission from the given entry. The corresponding functions **archive_entry_set_uid**(), **archive_entry_set_gid**(), and **archive_entry_set_perm**() store the given user id, group id and permission in the entry. The permission is also set as a side effect of calling **archive_entry_set_mode**().

archive_entry_strmode() returns a string representation of the permission as used by the long mode of ls(1).

User and group name

User and group names can be provided in one of three different ways:

char * Multibyte strings in the current locale.

wchar_t * Wide character strings in the current locale. The accessor functions are named **XXX_w**().

UTF-8 Unicode strings encoded as UTF-8. These are convenience functions to update both the multibyte and wide character strings at the same time.

archive_entry_set_XXX() is an alias for **archive_entry_copy_XXX**(). The strings are copied, and don't need to outlive the call.

File Flags

File flags are transparently converted between a bitmap representation and a textual format. For example, if you set the bitmap and ask for text, the library will build a canonical text format. However, if you set a text format and request a text format, you will get back the same text, even if it is ill-formed. If you need to canonicalize a textual flags string, you should first set the text form, then request the bitmap form, then use that to set the bitmap form. Setting the bitmap format will clear the internal text representation and force it to be reconstructed when you next request the text form.

The bitmap format consists of two integers, one containing bits that should be set, the other specifying bits that should be cleared. Bits not mentioned in either bitmap will be ignored. Usually, the bitmap of bits to be cleared will be set to zero. In unusual circumstances, you can force a fully-specified set of file flags by setting the bitmap of flags to clear to the complement of the bitmap of flags to set. (This differs from fflagstostr(3), which only includes names for set bits.) Converting a bitmap to a textual string is a platform-specific operation; bits that are not meaningful on the current platform will be ignored.

The canonical text format is a comma-separated list of flag names. The **archive_entry_copy_fflags_text**() and **archive_entry_copy_fflags_text_w**() functions parse the provided text and set the internal bitmap values. This is a platform-specific operation; names that are not meaningful on the current platform will be ignored. The function returns a pointer to the start of the first name that was not recognized, or NULL if every name was recognized. Note that every name -- including names that follow an unrecognized name -- will be evaluated, and the bitmaps will be set to reflect every name that is recognized. (In particular, this differs from strtofflags(3), which stops parsing at the first unrecognized name.)

SEE ALSO

archive_entry(3), archive_entry_acl(3), archive_read_disk(3), archive_write_disk(3), libarchive(3)

BUGS

The platform types *uid_t* and *gid_t* are often 16 or 32 bit wide. In this case it is possible that the ids can not be correctly restored from archives and get truncated.