NAME

libarchive — functions for reading and writing streaming archives

LIBRARY

library “libarchive”

OVERVIEW

The **libarchive** library provides a flexible interface for reading and writing archives in various formats such as tar and cpio. **libarchive** also supports reading and writing archives compressed using various compression filters such as gzip and bzip2. The library is inherently stream-oriented; readers serially iterate through the archive, writers serially add things to the archive. In particular, note that there is currently no built-in support for random access nor for in-place modification.

When reading an archive, the library automatically detects the format and the compression. The library currently has read support for:

- old-style tar archives,
- most variants of the POSIX “ustar” format,
- the POSIX “pax interchange” format,
- GNU-format tar archives,
- most common cpio archive formats,
- ISO9660 CD images (including RockRidge and Joliet extensions),
- Zip archives.

The library automatically detects archives compressed with gzip(1), bzip2(1), xz(1), or compress(1) and decompresses them transparently.

When writing an archive, you can specify the compression to be used and the format to use. The library can write

- POSIX-standard “ustar” archives,
- POSIX “pax interchange format” archives,
- POSIX octet-oriented cpio archives,
- Zip archive,
- two different variants of shar archives.

Pax interchange format is an extension of the tar archive format that eliminates essentially all of the limitations of historic tar formats in a standard fashion that is supported by POSIX-compliant pax(1) implementations on many systems as well as several newer implementations of tar(1). Note that the default write format will suppress the pax extended attributes for most entries; explicitly requesting pax format will enable those attributes for all entries.

The read and write APIs are accessed through the **archive_read_XXX**() functions and the **archive_write_XXX**() functions, respectively, and either can be used independently of the other.

The rest of this manual page provides an overview of the library operation. More detailed information can be found in the individual manual pages for each API or utility function.

READING AN ARCHIVE

See libarchive_read(3).

WRITING AN ARCHIVE

See libarchive_write(3).

WRITING ENTRIES TO DISK

The archive_write_disk(3) API allows you to write archive_entry(3) objects to disk using the same API used by archive_write(3). The archive_write_disk(3) API is used internally by **archive_read_extract**(); using it directly can provide greater control over how entries get written to disk. This API also makes it possible to share code between archive-to-archive copy and archive-to-disk extraction operations.

READING ENTRIES FROM DISK

The archive_read_disk(3) provides some support for populating archive_entry(3) objects from information in the filesystem.

DESCRIPTION

Detailed descriptions of each function are provided by the corresponding manual pages.

All of the functions utilize an opaque struct archive datatype that provides access to the archive contents.

The struct archive_entry structure contains a complete description of a single archive entry. It uses an opaque interface that is fully documented in archive_entry(3).

Users familiar with historic formats should be aware that the newer variants have eliminated most restrictions on the length of textual fields. Clients should not assume that filenames, link names, user names, or group names are limited in length. In particular, pax interchange format can easily accommodate pathnames in arbitrary character sets that exceed *PATH_MAX*.

RETURN VALUES

Most functions return **ARCHIVE_OK** (zero) on success, non-zero on error. The return value indicates the general severity of the error, ranging from **ARCHIVE_WARN**, which indicates a minor problem that should probably be reported to the user, to **ARCHIVE_FATAL**, which indicates a serious problem that will prevent any further operations on this archive. On error, the **archive_errno**() function can be used to retrieve a numeric error code (see errno(2)). The **archive_error_string**() returns a textual error message suitable for display.

archive_read_new() and **archive_write_new**() return pointers to an allocated and initialized struct archive object.

archive_read_data() and **archive_write_data**() return a count of the number of bytes actually read or written. A value of zero indicates the end of the data for this entry. A negative value indicates an error, in which case the **archive_errno**() and **archive_error_string**() functions can be used to obtain more information.

ENVIRONMENT

There are character set conversions within the archive_entry(3) functions that are impacted by the currently-selected locale.

SEE ALSO

tar(1), archive_entry(3), archive_read(3), archive_util(3), archive_write(3), tar(5)

HISTORY

The **libarchive** library first appeared in FreeBSD 5.3.

AUTHORS

The **libarchive** library was written by Tim Kientzle ⟨kientzle@acm.org⟩.

BUGS

Some archive formats support information that is not supported by struct archive_entry. Such information cannot be fully archived or restored using this library. This includes, for example, comments, character sets, or the arbitrary key/value pairs that can appear in pax interchange format archives.

Conversely, of course, not all of the information that can be stored in an struct archive_entry is supported by all formats. For example, cpio formats do not support nanosecond timestamps; old tar formats do not support large device numbers.

The `archive_read_disk`(3) API should support iterating over filesystems; that would make it possible to share code among disk-to-archive, archive-to-archive, archive-to-disk, and disk-to-disk operations. Currently, it only supports reading the information for a single file. (Which is still quite useful, as it hides a lot of system-specific details.)