NAME

archive_read_open, **archive_read_open2**, **archive_read_open_fd**, **archive_read_open_FILE**, **archive_read_open_filename**, **archive_read_open_memory** — functions for reading streaming archives

LIBRARY

Streaming Archive Library (libarchive, -larchive)

SYNOPSIS

#include <archive.h>

int
archive_read_open(*struct archive **, *void *client_data*, *archive_open_callback **, *archive_read_callback **, *archive_close_callback **);

int
archive_read_open2(*struct archive **, *void *client_data*, *archive_open_callback **, *archive_read_callback **, *archive_skip_callback **, *archive_close_callback **);

int
archive_read_open_FILE(*struct archive **, *FILE *file*);

int
archive_read_open_fd(*struct archive **, *int fd*, *size_t block_size*);

int
archive_read_open_filename(*struct archive **, *const char *filename*, *size_t block_size*);

int
archive_read_open_memory(*struct archive **, *const void *buff*, *size_t size*);

DESCRIPTION

archive_read_open()
: The same as **archive_read_open2**(), except that the skip callback is assumed to be NULL.

archive_read_open2()
: Freeze the settings, open the archive, and prepare for reading entries. This is the most generic version of this call, which accepts four callback functions. Most clients will want to use **archive_read_open_filename**(), **archive_read_open_FILE**(), **archive_read_open_fd**(), or **archive_read_open_memory**() instead. The library invokes the client-provided functions to obtain raw bytes from the archive.

archive_read_open_FILE()
: Like **archive_read_open**(), except that it accepts a *FILE ** pointer. This function should not be used with tape drives or other devices that require strict I/O blocking.

archive_read_open_fd()
: Like **archive_read_open**(), except that it accepts a file descriptor and block size rather than a set of function pointers. Note that the file descriptor will not be automatically closed at end-of-archive. This function is safe for use with tape drives or other blocked devices.

archive_read_open_file()
: This is a deprecated synonym for **archive_read_open_filename**().

archive_read_open_filename()
: Like **archive_read_open**(), except that it accepts a simple filename and a block size. A NULL filename represents standard input. This function is safe for use with tape drives or other blocked devices.

archive_read_open_memory()
: Like **archive_read_open**(), except that it accepts a pointer and size of a block of memory containing the archive data.

A complete description of the struct archive and struct archive_entry objects can be found in the overview manual page for libarchive(3).

CLIENT CALLBACKS

The callback functions must match the following prototypes:

typedef la_ssize_t **archive_read_callback**(*struct archive *, void *client_data, const void **buffer*)

typedef la_int64_t **archive_skip_callback**(*struct archive *, void *client_data, off_t request*)

typedef int **archive_open_callback**(*struct archive *, void *client_data*)

typedef int **archive_close_callback**(*struct archive *, void *client_data*)

The open callback is invoked by **archive_open**(). It should return **ARCHIVE_OK** if the underlying file or data source is successfully opened. If the open fails, it should call **archive_set_error**() to register an error code and message and return **ARCHIVE_FATAL**.

The read callback is invoked whenever the library requires raw bytes from the archive. The read callback should read data into a buffer, set the const void **buffer argument to point to the available data, and return a count of the number of bytes available. The library will invoke the read callback again only after it has consumed this data. The library imposes no constraints on the size of the data blocks returned. On end-of-file, the read callback should return zero. On error, the read callback should invoke **archive_set_error**() to register an error code and message and return -1.

The skip callback is invoked when the library wants to ignore a block of data. The return value is the number of bytes actually skipped, which may differ from the request. If the callback cannot skip data, it should return zero. If the skip callback is not provided (the function pointer is NULL), the library will invoke the read function instead and simply discard the result. A skip callback can provide significant performance gains when reading uncompressed archives from slow disk drives or other media that can skip quickly.

The close callback is invoked by archive_close when the archive processing is complete. The callback should return **ARCHIVE_OK** on success. On failure, the callback should invoke **archive_set_error**() to register an error code and message and return **ARCHIVE_FATAL**.

RETURN VALUES

These functions return **ARCHIVE_OK** on success, or **ARCHIVE_FATAL**.

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_read_data(3), archive_read_filter(3), archive_read_format(3), archive_read_set_options(3), archive_util(3), `libarchive(3)`, `tar(5)`