NAME

archive_read — functions for reading streaming archives

LIBRARY

Streaming Archive Library (libarchive, -larchive)

SYNOPSIS

```
#include <archive.h>
```

DESCRIPTION

These functions provide a complete API for reading streaming archives. The general process is to first create the struct archive object, set options, initialize the reader, iterate over the archive headers and associated data, then close the archive and release all resources.

Create archive object

See archive_read_new(3).

To read an archive, you must first obtain an initialized struct archive object from **archive_read_new**().

Enable filters and formats

See archive_read_filter(3) and archive_read_format(3).

You can then modify this object for the desired operations with the various **archive_read_set_XXX**() and **archive_read_support_XXX**() functions. In particular, you will need to invoke appropriate **archive_read_support_XXX**() functions to enable the corresponding compression and format support. Note that these latter functions perform two distinct operations: they cause the corresponding support code to be linked into your program, and they e

file descriptor, or **archive_read_extract**(), which recreates the specified entry on disk and copies data from the archive. In particular, note that **archive_read_extract**() uses the struct archive_entry structure that you provide it, which may differ from the entry just read from the archive. In particular, many applications will want to override the pathname, file permissions, or ownership.

Release resources

See archive_read_free(3).

Once you have finished reading data from the archive, you should call **archive_read_close**() to close the archive, then call **archive_read_free**() to release all resources, including all memory allocated by the library.

EXAMPLES

The following illustrates basic usage of the library. In this example, the callback functions are simply wrappers around the standard open(2), read(2), and close(2) system calls.

```
void
list_archive(const char *name)
{
  struct mydata *mydata;
  struct archive *a;
  struct archive_entry *entry;

  mydata = malloc(sizeof(struct mydata));
  a = archive_read_new();
  mydata->name = name;
  archive_read_support_filter_all(a);
  archive_read_support_format_all(a);
  archive_read_open(a, mydata, myopen, myread, myclose);
  while (archive_read_next_header(a, &entry) == ARCHIVE_OK) {
    printf("%s\n",archive_entry_pathname(entry));
    archive_read_data_skip(a);
  }
  archive_read_free(a);
  free(mydata);
}

la_ssize_t
myread(struct archive *a, void *client_data, const void **buff)
{
  struct mydata *mydata = client_data;

  *buff = mydata->buff;
  return (read(mydata->fd, mydata->buff, 10240));
}

int
myopen(struct archive *a, void *client_data)
{
  struct mydata *mydata = client_data;

  mydata->fd = open(mydata->name, O_RDONLY);
  return (mydata->fd >= 0 ? ARCHIVE_OK : ARCHIVE_FATAL);
```

```
}

int
myclose(struct archive *a, void *client_data)
{
  struct mydata *mydata = client_data;

  if (mydata->fd > 0)
    close(mydata->fd);
  return (ARCHIVE_OK);
}
```

SEE ALSO

tar(1), archive_read_data(3), archive_read_extract(3), archive_read_filter(3), archive_read_format(3), archive_read_header(3), archive_read_new(3), archive_read_open(3), archive_read_set_options(3), archive_util(3), libarchive(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

Many traditional archiver programs treat empty files as valid empty archives. For example, many implementations of tar(1) allow you to append entries to an empty file. Of course, it is impossible to determine the format of an empty file by inspecting the contents, so this library treats empty files as having a special "empty" format.