SWI-Prolog binding to libarchive

Jan Wielemaker
VU University Amsterdam
The Netherlands
E-mail: J.Wielemaker@vu.nl

January 12, 2018

Abstract

The library libarchive provides a portable way to access archive files as well as encoded (typically compressed) data. This package is a Prolog wrapper around this library. The motivation to introduce this library is twofold. In the first place, it provides a minimal platform independent API to access archives. In the second place, it allows accessing archives through Prolog streams, which often eliminates the need for temporary files and all related consequences for performance, security and platform dependency.

Contents

1 Motivation

Archives play two roles: they combine multiple documents into a single one and they typically provide compression and sometimes encryption or other services. Bundling multiple resources into a single archive may greatly simplify distribution and guarantee that the individual resources are consistent. SWI-Prolog provides archiving using its (rather arcane) saved-state format. See `resource/3` and `open_resource/3`. It also provides compression by means of library(zlib).

External archives may be accessed through the process interface provided by `process_create/3`, but this has disadvantages. The one that motivated this library was that using external processes provide no decent platform independent access to archives. Most likely zip files come closest to platform independent access, but there are many different programs for accessing zip files that provide slightly different sets of options and the existence of any of these programs cannot be guaranteed without distributing our own bundled version. Similar arguments hold for Unix tar archives, where just about any Unix-derives system has a tar program but except for very basic commands, the command line options are not compatible and tar is not part of Windows. The only format granted on Windows is .cab, but a program to create them is not part of Windows and the .cab format is rare outside the Windows context.

Discarding availability of archive programs, each archive program comes with its own set of command line options and its own features and limitations. Fortunately, libarchive provides a consistent interface to a wealth of compression and archiving formats. The library `archive` wraps this library, providing access to archives using Prolog streams both for the archive as a whole and the archive entries. E.g., archives may be read from Prolog streams and each member in turn may be processed using Prolog streams without materialising data using temporary files.

2 library(archive): Access several archive formats

See also http://code.google.com/p/libarchive/

This library uses *libarchive* to access a variety of archive formats. The following example lists the entries in an archive:

```
list_archive(File) :-
      archive_open(File, Archive, []),
      repeat,
         (   archive_next_header(Archive, Path)
         ->  format('~w~n', [Path]),
             fail
         ;   !,
             archive_close(Archive)
         ).
```

archive_open(+*Data, -Archive, +Options*) *[det]*

Wrapper around `archive_open/4` that opens the archive in read mode.

archive_open(+*Data, +Mode, -Archive, +Options*) *[det]*

Open the archive in *Data* and unify *Archive* with a handle to the opened archive. *Data* is either a file or a stream that contains a valid archive. Details are controlled by *Options*. Typically, the option `close_parent(true)` is used to close stream if the archive is closed using `archive_close/1`. For other options, the defaults are typically fine. The option `format(raw)` must be used to process compressed streams that do not contain explicit entries (e.g., gzip'ed data) unambibuously. The `raw` format creates a *pseudo archive* holding a single member named `data`.

close_parent(+*Boolean*)

If this option is `true` (default `false`), Stream is closed if `archive_close/1` is called on *Archive*.

compression(+*Compression*)

Synomym for `filter(Compression)`. Deprecated.

filter(+*Filter*)

Support the indicated filter. This option may be used multiple times to support multiple filters. In read mode, If no filter options are provided, `all` is assumed. In write mode, none is assumed. Supported values are `all`, `bzip2`, `compress`, `gzip`, `grzip`, `lrzip`, `lzip`, `lzma`, `lzop`, `none`, `rpm`, `uu` and `xz`. The value `all` is default for read, `none` for write.

format(+*Format*)

Support the indicated format. This option may be used multiple times to support multiple formats in read mode. In write mode, you must supply a single format. If no format options are provided, `all` is assumed for read mode. Note that `all` does **not** include `raw`. To open both archive and non-archive files, *both* `format(all)` and `format(raw)` must be specified. Supported values are: `all`, `7zip`, `ar`, `cab`, `cpio`, `empty`, `gnutar`, `iso9660`, `lha`, `mtree`, `rar`, `raw`, `tar`, `xar` and `zip`. The value `all` is default for read.

Note that the actually supported compression types and formats may vary depending on the version and installation options of the underlying libarchive library. This predicate raises a domain error if the (explicitly) requested format is not supported.

Errors

- `domain_error(filter, Filter)` if the requested filter is not supported.
- `domain_error(format, Format)` if the requested format type is not supported.

archive_close(+*Archive*) *[det]*

Close the archive. If `close_parent(true)` is specified, the underlying stream is closed too. If there is an entry opened with `archive_open_entry/2`, actually closing the archive is delayed until the stream associated with the entry is closed. This can be used to open a stream to an archive entry without having to worry about closing the archive:

```
archive_open_named(ArchiveFile, EntryName, Stream) :-
    archive_open(ArchiveFile, Handle, []),
    archive_next_header(Handle, Name),
```

```
    archive_open_entry(Handle, Stream),
    archive_close(Archive).
```

archive_property(+*Handle, ?Property*) *[nondet]*

True when *Property* is a property of the archive *Handle*. Defined properties are:

filters(*List*)

True when the indicated filters are applied before reaching the archive format.

archive_next_header(+*Handle, -Name*) *[semidet]*

Forward to the next entry of the archive for which *Name* unifies with the pathname of the entry. Fails silently if the name of the archive is reached before success. *Name* is typically specified if a single entry must be accessed and unbound otherwise. The following example opens a Prolog stream to a given archive entry. Note that *Stream* must be closed using `close/1` and the archive must be closed using `archive_close/1` after the data has been used. See also `setup_call_cleanup/3`.

```
open_archive_entry(ArchiveFile, Entry, Stream) :-
    open(ArchiveFile, read, In, [type(binary)]),
    archive_open(In, Archive, [close_parent(true)]),
    archive_next_header(Archive, Entry),
    archive_open_entry(Archive, Stream).
```

Errors `permission_error(next_header, archive, Handle)` if a previously opened entry is not closed.

archive_open_entry(+*Archive, -Stream*) *[det]*

Open the current entry as a stream. *Stream* must be closed. If the stream is not closed before the next call to `archive_next_header/2`, a permission error is raised.

archive_set_header_property(+*Archive, +Property*)

Set *Property* of the current header. Write-mode only. Defined properties are:

filetype(-*Type*)

Type is one of `file`, `link`, `socket`, `character_device`, `block_device`, `directory` or `fifo`. It appears that this library can also return other values. These are returned as an integer.

mtime(-*Time*)

True when entry was last modified at time.

size(-*Bytes*)

True when entry is *Bytes* long.

link_target(-*Target*)

Target for a link. Currently only supported for symbolic links.

archive_header_property(+*Archive, ?Property*)

True when *Property* is a property of the current header. Defined properties are:

filetype(-*Type*)
Type is one of `file`, `link`, `socket`, `character_device`, `block_device`, `directory` or `fifo`. It appears that this library can also return other values. These are returned as an integer.

mtime(-*Time*)
True when entry was last modified at time.

size(-*Bytes*)
True when entry is *Bytes* long.

link_target(-*Target*)
Target for a link. Currently only supported for symbolic links.

format(-*Format*)
Provides the name of the archive format applicable to the current entry. The returned value is the lowercase version of the output of `archive_format_name()`.

permissions(-*Integer*)
True when entry has the indicated permission mask.

archive_extract(+*ArchiveFile, +Dir, +Options*)
Extract files from the given archive into *Dir*. Supported options:

remove_prefix(+*Prefix*)
Strip *Prefix* from all entries before extracting

exclude(+*ListOfPatterns*)
Ignore members that match one of the given patterns. Patterns are handed to `wildcard_match/2`.

Errors
- `existence_error(directory, Dir)` if *Dir* does not exist or is not a directory.
- `domain_error(path_prefix(Prefix), Path)` if a path in the archive does not start with Prefix

To be done Add options

archive_entries(+*Archive, -Paths*) *[det]*
True when *Paths* is a list of pathnames appearing in *Archive*.

archive_data_stream(+*Archive, -DataStream, +Options*) *[nondet]*
True when *DataStream* is a stream to a data object inside *Archive*. This predicate transparently unpacks data inside *possibly nested* archives, e.g., a *tar* file inside a *zip* file. It applies the appropriate decompression filters and thus ensures that Prolog reads the plain data from *DataStream*. *DataStream* must be closed after the content has been processed. Backtracking opens the next member of the (nested) archive. This predicate processes the following options:

meta_data(-*Data:list(dict)*)
If provided, *Data* is unified with a list of filters applied to the (nested) archive to open the current *DataStream*. The first element describes the outermost archive. Each *Data* dict contains the header properties (`archive_header_property/2`) as well as the keys:

filters(*Filters:list(atom)*)
Filter list as obtained from `archive_property/2`

name(*Atom*)
Name of the entry.

Non-archive files are handled as pseudo-archives that hold a single stream. This is implemented by using `archive_open/3` with the options `[format(all),format(raw)]`.

archive_create(+*OutputFile, +InputFiles, +Options*) *[det]*
Convenience predicate to create an archive in *OutputFile* with data from a list of *InputFiles* and the given *Options*.

Besides options supported by `archive_open/4`, the following options are supported:

directory(+*Directory*)
Changes the directory before adding input files. If this is specified, paths of input files must be relative to *Directory* and archived files will not have *Directory* as leading path. This is to simulate `-C` option of the `tar` program.

format(+*Format*)
Write mode supports the following formats: '7zip, cpio, gnutar, iso9660, xar and zip'. Note that a particular installation may support only a subset of these, depending on the configuration of `libarchive`.

3 Status

The current version is merely a proof-of-concept. It lacks writing archives and does not support many of the options of the underlying library. The main motivation for starting this library was to achieve portability of the upcomming SWI-Prolog package distribution system. Other functionality will be added on 'as needed' basis.

Index