SWI-Prolog binding to zlib

Jan Wielemaker
HCS,
University of Amsterdam
The Netherlands
E-mail: wielemak@science.uva.nl

February 2, 2009

Abstract

The library `zlib` provides a binding to the zlib general purpose compression library. The prolog library aims as seamlessly reading and writing files compatible to the `gzip` program as well as compressed (network) communication.

Contents

1 Zlib and compression

Zlib is a widespread library implementing the RFC1950 (zlib wrapper), RFC1951 (deflate stream) and RFC1952 (gzip wrapper) compression standards. The SWI-Prolog binding is a foreign library that creates a compressed stream as a wrapper around a normal stream. Implemented this way, it can perform a wide variety of tasks:

- Read/write gzip compatible files
- Setup standard compressed stream communication
- Realise in-memory compression or decompression
- Deal with streams holding embedded compressed objects

The core predicate of the library is `zopen/3`. The remainder of the functionality of `zlib` is defined in Prolog and can be used as a starting point for other high-level primitives. See also `ztest.pl` providing test and demo code. This file is part of the source distribution.

Part of the functionality of this library can also be realised using the pipe interface and the `gzip` program. For example, a gziped file can also be opened in Prolog using the code below.

```
...
open(pipe('gunzip < file.gz'), read, In),
...
```

The advantage of this library for such tasks is enhanced platform independence and reduced time to open a file. Platform independence is improved as we do not have to worry about availability of the `gunzip` utility and we do not have to worry about shell and filename quoting issues. While the above works well on most modern Unix systems, it only works with special precautions on Windows.[1]

The library becomes really valuable if we consider compressed network communication. Here we get the stream from `tcp_open_socket/3`. The library provides efficient creation of a compressed stream, as well as support for flushing output through the standard Prolog `flush_output/1` call.

2 Predicate reference

zopen(+*Stream, -ZStream, +Options*)

Creates *ZStream*, providing compressed access to *Stream*. If an input stream is wrapped, it recognises a gzip or deflate header.

If an output stream is enabled, *Options* define the desired wrapper and compression level. Defined options on output streams are:

format(+*Format*)

Either `deflate` (default) or `gzip`. The `deflate` envelope is simple and short and is typically used for compressed (network) communication. The `gzip` envelope is compatible to the `gzip` program and intended to read/write compressed files.

[1] Install gunzip, deal with Windows path-names, the windows shell and quoting.

level(*+Level*)
Number between 0 and 9, specifying the compression level, Higher levels use more resources. Default is 6, generally believed to be a good compromise between speed, memory requirement and compression.

Generic options are:

close_parent(*Bool*)
If `true` (default), closing the compressed stream also closes (and thus invalidates) the wrapped stream. If `false`, the wrapped stream is *not* closed. This can be used to read/write a compressed ndata block as partial input/output on a stream.

gzopen(*+File, +Mode, -Stream*)
Same as `gzopen`File, Mode, Stream, [].

gzopen(*+File, +Mode, -Stream, +Options*)
Open `gzip` compatible *File* for reading or writing.

3 Interaction with Prolog stream predicates

Using `flush_output/1` on a compressed stream causes a `Z_SYNC_FLUSH` on the stream. Using `close/1` on a compressed stream causes a `Z_FINISH` on the stream. If the stream uses the `gzip` format, a `gzip` compatible footer is written to the stream. If `close_parent` is set (default) the underlying stream is closed too. Otherwise it remains open and the user can continue communication in non-compressed format or reopen the stream for compression using `zopen/3`.

4 Installation

4.1 Unix systems

Installation on Unix system uses the commonly found *configure*, *make* and *make install* sequence. SWI-Prolog should be installed before building this package. If SWI-Prolog is not installed as `pl`, the environment variable `PL` must be set to the name of the SWI-Prolog executable. Installation is now accomplished using:

```
% ./configure
% make
% make install
```

This installs the foreign libraries in `$PLBASE/lib/$PLARCH` and the Prolog library files in `$PLBASE/library`, where `$PLBASE` refers to the SWI-Prolog 'home-directory'.