Miniz

Miniz is a lossless, high performance data compression library in a single source file that implements the zlib (RFC 1950) and Deflate (RFC 1951) compressed data format specification standards. It supports the most commonly used functions exported by the zlib library, but is a completely independent implementation so zlib's licensing requirements do not apply. Miniz also contains simple to use functions for writing .PNG format image files and reading/writing/appending .ZIP format archives. Miniz's compression speed has been tuned to be comparable to zlib's, and it also has a specialized real-time compressor function designed to compare well against fastlz/minilzo.

Usage

Please use the files from the releases page in your projects. Do not use the git checkout directly! The different source and header files are amalgamated into one `miniz.c` / `miniz.h` pair in a build step (`amalgamate.sh`). Include `miniz.c` and `miniz.h` in your project to use Miniz.

Features

- MIT licensed
- A portable, single source and header file library written in plain C. Tested with GCC, clang and Visual Studio.
- Easily tuned and trimmed down by defines
- A drop-in replacement for zlib's most used API's (tested in several open source projects that use zlib, such as libpng and libzip).
- Fills a single threaded performance vs. compression ratio gap between several popular real-time compressors and zlib. For example, at level 1, miniz.c compresses around 5-9% better than minilzo, but is approx. 35% slower. At levels 2-9, miniz.c is designed to compare favorably against zlib's ratio and speed. See the miniz performance comparison page for example timings.
- Not a block based compressor: miniz.c fully supports stream based processing using a coroutine-style implementation. The zlib-style API functions can be called a single byte at a time if that's all you've got.
- Easy to use. The low-level compressor (tdefl) and decompressor (tinfl) have simple state structs which can be saved/restored as needed with simple memcpy's. The low-level codec API's don't use the heap in any way.
- Entire inflater (including optional zlib header parsing and Adler-32 checking) is implemented in a single function as a coroutine, which is separately available in a small (~550 line) source file: miniz_tinfl.c
- A fairly complete (but totally optional) set of .ZIP archive manipulation and extraction API's. The archive functionality is intended to solve common problems encountered in embedded, mobile, or game development situations. (The archive API's are purposely just powerful enough to write an entire archiver given a bit of additional higher-level logic.)

Known Problems

- No support for encrypted archives. Not sure how useful this stuff is in practice.
- Minimal documentation. The assumption is that the user is already familiar with the basic zlib API. I need to write an API wiki - for now I've tried to place key comments before each enum/API, and I've included 6 examples that demonstrate how to use the module's major features.

Special Thanks

Thanks to Alex Evans for the PNG writer function. Also, thanks to Paul Holden and Thorsten Scheuermann for feedback and testing, Matt Pritchard for all his encouragement, and Sean Barrett's various public domain libraries for inspiration (and encouraging me to write miniz.c in C, which was much more enjoyable and less painful than I thought it would be considering I've been programming in C++ for so long).

Thanks to Bruce Dawson for reporting a problem with the level_and_flags archive API parameter (which is fixed in v1.12) and general feedback, and Janez Zemva for indirectly encouraging me into writing more examples.

Patents

I was recently asked if miniz avoids patent issues. miniz purposely uses the same core algorithms as the ones used by zlib. The compressor uses vanilla hash chaining as described here. Also see the gzip FAQ. In my opinion, if miniz falls prey to a patent attack then zlib/gzip are likely to be at serious risk too.