Changelog

2.0.8

- Remove unimplemented functions (mz_zip_locate_file and mz_zip_locate_file_v2)
- Add license, changelog, readme and example files to release zip
- Fix heap overflow to user buffer in tinfl_status tinfl_decompress
- Fix corrupt archive if uncompressed file smaller than 4 byte and file is added by mz_zip_writer_add_mem*

2.0.7

- Removed need in C++ compiler in cmake build
- Fixed loads of uninitialized value errors found with Valgrind by memsetting m_dict to 0 in tdefl_init.
- Fix resource leak in mz_zip_reader_init_file_v2
- Fix assert with mz_zip_writer_add_mem* w/MZ_DEFAULT_COMPRESSION
- cmake build: install library and headers
- Remove _LARGEFILE64_SOURCE requirement from apple defines for large files

2.0.6

- Improve MZ_ZIP_FLAG_WRITE_ZIP64 documentation
- Remove check for cur_archive_file_ofs > UINT_MAX, because cur_archive_file_ofs is not used after this point
- Add cmake debug configuration
- Fix PNG height when creating png files
- Add "iterative" file extraction method based on mz_zip_reader_extract_to_callback.
- Option to use memcpy for unaligned data access
- Define processor/arch macros as zero if not set to one

2.0.4/2.0.5

- Fix compilation with the various omission compile definitions

2.0.3

- Fix GCC/clang compile warnings
- Added callback for periodic flushes (for ZIP file streaming)
- Use UTF-8 for file names in ZIP files per default

2.0.2

- Fix source backwards compatibility with 1.x
- Fix a ZIP bit not being set correctly

2.0.1

- Added some tests
- Added CI
- Make source code ANSI C compatible

2.0.0 beta

- Matthew Sitton merged to vogl ZIP64 changes. Miniz is now licensed as MIT since the vogl code base is MIT licensed
- Miniz is now split into several files
- Miniz does now not seek backwards when creating ZIP files. That is the ZIP files can be streamed
- Miniz automatically switches to the ZIP64 format the created ZIP files goes over ZIP file limits
- Similar to SQLite the Miniz source code is amalgamated into one miniz.c/miniz.h pair in a build step (amalgamate.sh). Please use miniz.c/miniz.h in your projects

v1.16 BETA Oct 19, 2013

Still testing, this release is downloadable from here. Two key inflator-only robustness and streaming related changes. Also merged in tdefl_compressor_alloc(), tdefl_compressor_free() helpers to make script bindings easier for rustyzip. I would greatly appreciate any help with testing or any feedback.

The inflator in raw (non-zlib) mode is now usable on gzip or similar streams that have a bunch of bytes following the raw deflate data (problem discovered by rustyzip author williamw520). This version should never read beyond the last byte of the raw deflate data independent of how many bytes you pass into the input buffer.

The inflator now has a new failure status TINFL_STATUS_FAILED_CANNOT_MAKE_PROGRESS (-4). Previously, if the inflator was starved of bytes and could not make progress (because the input buffer was empty and the caller did not set the TINFL_FLAG_HAS_MORE_INPUT flag - say on truncated or corrupted compressed data stream) it would append all 0's to the input and try to soldier on. This is scary behavior if the caller didn't know when to stop accepting output (because it didn't know how much uncompressed data was expected, or didn't enforce a sane maximum). v1.16 will instead return TINFL_STATUS_FAILED_CANNOT_MAKE_PROGRESS immediately if it needs 1 or more bytes to make progress, the input buf is empty, and the caller has indicated that no more input is available. This is a "soft" failure, so you can call the inflator again with more input and it will try to continue, or you can give up and fail. This could be very useful in network streaming scenarios.

- The inflator coroutine func. is subtle and complex so I'm being cautious about this release. I would greatly appreciate any help with testing or

any feedback. I feel good about these changes, and they've been through several hours of automated testing, but they will probably not fix anything for the majority of prev. users so I'm going to mark this release as beta for a few weeks and continue testing it at work/home on various things.

- The inflator in raw (non-zlib) mode is now usable on gzip or similar data streams that have a bunch of bytes following the raw deflate data (problem discovered by rustyzip author williamw520). This version should *never* read beyond the last byte of the raw deflate data independent of how many bytes you pass into the input buffer. This issue was caused by the various Huffman bitbuffer lookahead optimizations, and would not be an issue if the caller knew and enforced the precise size of the raw compressed data *or* if the compressed data was in zlib format (i.e. always followed by the byte aligned zlib adler32). So in other words, you can now call the inflator on deflate streams that are followed by arbitrary amounts of data and it's guaranteed that decompression will stop exactly on the last byte.
- The inflator now has a new failure status: TINFL_STATUS_FAILED_CANNOT_MAKE_PROGRESS (-4). Previously, if the inflator was starved of bytes and could not make progress (because the input buffer was empty and the caller did not set the TINFL_FLAG_HAS_MORE_INPUT flag - say on truncated or corrupted compressed data stream) it would append all 0's to the input and try to soldier on. This is scary, because in the worst case, I believe it was possible for the prev. inflator to start outputting large amounts of literal data. If the caller didn't know when to stop accepting output (because it didn't know how much uncompressed data was expected, or didn't enforce a sane maximum) it could continue forever. v1.16 cannot fall into this failure mode, instead it'll return TINFL_STATUS_FAILED_CANNOT_MAKE_PROGRESS immediately if it needs 1 or more bytes to make progress, the input buf is empty, and the caller has indicated that no more input is available. This is a "soft" failure, so you can call the inflator again with more input and it will try to continue, or you can give up and fail. This could be very useful in network streaming scenarios.
- Added documentation to all the tinfl return status codes, fixed miniz_tester so it accepts double minus params for Linux, tweaked example1.c, added a simple "follower bytes" test to miniz_tester.cpp. ### v1.15 r4 STABLE - Oct 13, 2013

Merged over a few very minor bug fixes that I fixed in the zip64 branch. This is downloadable from here and also in SVN head (as of 10/19/13).

v1.15 - Oct. 13, 2013

Interim bugfix release while I work on the next major release with zip64 and streaming compression/decompression support. Fixed the MZ_ZIP_FLAG_DO_NOT_SORT_CENTRAL_DIRECTORY bug (thanks kahmyong.moon@hp.com), which could cause the locate files

func to not find files when this flag was specified. Also fixed a bug in mz_zip_reader_extract_to_mem_no_alloc() with user provided read buffers (thanks kymoon). I also merged lots of compiler fixes from various github repo branches and Google Code issue reports. I finally added cmake support (only tested under for Linux so far), compiled and tested with clang v3.3 and gcc 4.6 (under Linux), added defl_write_image_to_png_file_in_memory_ex() (supports Y flipping for OpenGL use, real-time compression), added a new PNG example (example6.c - Mandelbrot), and I added 64-bit file I/O support (stat64(), etc.) for glibc.

- Critical fix for the MZ_ZIP_FLAG_DO_NOT_SORT_CENTRAL_DIRECTORY bug (thanks kahmyong.moon@hp.com) which could cause locate files to not find files. This bug would only have occurred in earlier versions if you explicitly used this flag, OR if you used mz_zip_extract_archive_file_to_heap() or mz_zip_add_mem_to_archive_file_in_place() (which used this flag). If you can't switch to v1.15 but want to fix this bug, just remove the uses of this flag from both helper funcs (and of course don't use the flag).
- Bugfix in mz_zip_reader_extract_to_mem_no_alloc() from kymoon when pUser_read_buf is not NULL and compressed size is > uncompressed size
- Fixing mz_zip_reader_extract_*() funcs so they don't try to extract compressed data from directory entries, to account for weird zipfiles which contain zero-size compressed data on dir entries. Hopefully this fix won't cause any issues on weird zip archives, because it assumes the low 16-bits of zip external attributes are DOS attributes (which I believe they always are in practice).
- Fixing mz_zip_reader_is_file_a_directory() so it doesn't check the internal attributes, just the filename and external attributes
- mz_zip_reader_init_file() - missing MZ_FCLOSE() call if the seek failed
- Added cmake support for Linux builds which builds all the examples, tested with clang v3.3 and gcc v4.6.
- Clang fix for tdefl_write_image_to_png_file_in_memory() from toffaletti
- Merged MZ_FORCEINLINE fix from hdeanclark
- Fix <time.h> include before config #ifdef, thanks emil.brink
- Added tdefl_write_image_to_png_file_in_memory_ex(): supports Y flipping (super useful for OpenGL apps), and explicit control over the compression level (so you can set it to 1 for real-time compression).
- Merged in some compiler fixes from paulharris's github repro.
- Retested this build under Windows (VS 2010, including static analysis), tcc 0.9.26, gcc v4.6 and clang v3.3.
- Added example6.c, which dumps an image of the mandelbrot set to a PNG file.
- Modified example2 to help test the MZ_ZIP_FLAG_DO_NOT_SORT_CENTRAL_DIRECTORY flag more.
- In r3: Bugfix to mz_zip_writer_add_file() found during merge: Fix possible src file fclose() leak if alignment bytes+local header file write

failed
- In r4: Minor bugfix to mz_zip_writer_add_from_zip_reader(): Was pushing the wrong central dir header offset, appears harmless in this release, but it became a problem in the zip64 branch

v1.14 - May 20, 2012

(SVN Only) Minor tweaks to get miniz.c compiling with the Tiny C Compiler, added #ifndef MINIZ_NO_TIME guards around utime.h includes. Adding mz_free() function, so the caller can free heap blocks returned by miniz using whatever heap functions it has been configured to use, MSVC specific fixes to use "safe" variants of several functions (localtime_s, fopen_s, freopen_s).

MinGW32/64 GCC 4.6.1 compiler fixes: added MZ_FORCEINLINE, #include <time.h> (thanks fermtect).

Compiler specific fixes, some from fermtect. I upgraded to TDM GCC 4.6.1 and now static ___forceinline is giving it fits, so I'm changing all usage of ___forceinline to MZ_FORCEINLINE and forcing gcc to use **attribute**((**always_inline**)) (and MSVC to use ___forceinline). Also various fixes from fermtect for MinGW32: added #include , 64-bit ftell/fseek fixes.

v1.13 - May 19, 2012

From jason@cornsyrup.org and kelwert@mtu.edu - Most importantly, fixed mz_crc32() so it doesn't compute the wrong CRC-32's when mz_ulong is 64-bits. Temporarily/locally slammed in "typedef unsigned long mz_ulong" and re-ran a randomized regression test on ~500k files. Other stuff:

Eliminated a bunch of warnings when compiling with GCC 32-bit/64. Ran all examples, miniz.c, and tinfl.c through MSVC 2008's /analyze (static analysis) option and fixed all warnings (except for the silly "Use of the comma-operator in a tested expression.." analysis warning, which I purposely use to work around a MSVC compiler warning).

Created 32-bit and 64-bit Codeblocks projects/workspace. Built and tested Linux executables. The codeblocks workspace is compatible with Linux+Win32/x64. Added miniz_tester solution/project, which is a useful little app derived from LZHAM's tester app that I use as part of the regression test. Ran miniz.c and tinfl.c through another series of regression testing on ~500,000 files and archives. Modified example5.c so it purposely disables a bunch of high-level functionality (MINIZ_NO_STDIO, etc.). (Thanks to corysama for the MINIZ_NO_STDIO bug report.)

Fix ftell() usage in a few of the examples so they exit with an error on files which are too large (a limitation of the examples, not miniz itself). Fix fail logic handling in mz_zip_add_mem_to_archive_file_in_place() so it always calls mz_zip_writer_finalize_archive() and mz_zip_writer_end(), even if the file add fails.

- From jason@cornsyrup.org and kelwert@mtu.edu - Fix mz_crc32() so it doesn't compute the wrong CRC-32's when mz_ulong is 64-bit.
- Temporarily/locally slammed in "typedef unsigned long mz_ulong" and re-ran a randomized regression test on ~500k files.
- Eliminated a bunch of warnings when compiling with GCC 32-bit/64.
- Ran all examples, miniz.c, and tinfl.c through MSVC 2008's /analyze (static analysis) option and fixed all warnings (except for the silly "Use of the comma-operator in a tested expression.." analysis warning, which I purposely use to work around a MSVC compiler warning).
- Created 32-bit and 64-bit Codeblocks projects/workspace. Built and tested Linux executables. The codeblocks workspace is compatible with Linux+Win32/x64.
- Added miniz_tester solution/project, which is a useful little app derived from LZHAM's tester app that I use as part of the regression test.
- Ran miniz.c and tinfl.c through another series of regression testing on ~500,000 files and archives.
- Modified example5.c so it purposely disables a bunch of high-level functionality (MINIZ_NO_STDIO, etc.). (Thanks to corysama for the MINIZ_NO_STDIO bug report.)
- Fix ftell() usage in examples so they exit with an error on files which are too large (a limitation of the examples, not miniz itself).

v1.12 - 4/12/12

More comments, added low-level example5.c, fixed a couple minor level_and_flags issues in the archive API's. level_and_flags can now be set to MZ_DEFAULT_COMPRESSION. Thanks to Bruce Dawson bruced@valvesoftware.com for the feedback/bug report.

v1.11 - 5/28/11

Added statement from unlicense.org

v1.10 - 5/27/11

- Substantial compressor optimizations:
- Level 1 is now ~4x faster than before. The L1 compressor's throughput now varies between 70-110MB/sec. on a Core i7 (actual throughput varies depending on the type of data, and x64 vs. x86).
- Improved baseline L2-L9 compression perf. Also, greatly improved compression perf. issues on some file types.
- Refactored the compression code for better readability and maintainability.
- Added level 10 compression level (L10 has slightly better ratio than level 9, but could have a potentially large drop in throughput on some files).

v1.09 - 5/15/11

Initial stable release.