DEFLATE

"Deflate" redirects here. For other uses, see Deflation (disambiguation).

In computing, **deflate** is a data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding. It was originally defined by Phil Katz for version 2 of his PKZIP archiving tool and was later specified in RFC 1951.[1]

The original algorithm as designed by Katz was patented as U.S. Patent 5,051,745 and assigned to PKWARE, Inc.[2][3] As stated in the RFC document, Deflate is widely thought to be implementable in a manner not covered by patents.[1] This has led to its widespread use, for example in gzip compressed files, PNG image files and the .ZIP file format for which Katz originally designed it.

1 Stream format

A Deflate stream consists of a series of blocks. Each block is preceded by a 3-bit header:

- First bit: Last-block-in-stream marker:
  - 1: this is the last block in the stream.
  - 0: there are more blocks to process after this one.
- Second and third bits: Encoding method used for this block type:
  - 00: a stored/raw/literal section, between 0 and 65,535 bytes in length.
  - 01: a *static Huffman* compressed block, using a pre-agreed Huffman tree.
  - 10: a compressed block complete with the Huffman table supplied.
  - 11: reserved, don't use.

Most blocks will end up being encoded using method 10, the *dynamic Huffman* encoding, which produces an optimised Huffman tree customised for each block of data individually. Instructions to generate the necessary Huffman tree immediately follow the block header.

Compression is achieved through two steps

- The matching and replacement of duplicate strings with pointers.
- Replacing symbols with new, weighted symbols based on frequency of use.

1.1 Duplicate string elimination

Main article: LZ77 and LZ78

Within compressed blocks, if a duplicate series of bytes is spotted (a repeated string), then a back-reference is inserted, linking to the previous location of that identical string instead. An encoded match to an earlier string consists of a length (3–258 bytes) and a distance (1–32,768 bytes). Relative back-references can be made across any number of blocks, as long as the distance appears within the last 32 kB of uncompressed data decoded (termed the *sliding window*).

1.2 Bit reduction

Main article: Huffman coding

The second compression stage consists of replacing commonly used symbols with shorter representations and less commonly used symbols with longer representations. The method used is Huffman coding which creates an unprefixed tree of non-overlapping intervals, where the length of each sequence is inversely proportional to the probability of that symbol needing to be encoded. The more likely a symbol has to be encoded, the shorter its bit-sequence will be.

A tree is created, containing space for 288 symbols:

- 0–255: represent the literal bytes/symbols 0–255.
- 256: end of block – stop processing if last block, otherwise start processing next block.
- 257–285: combined with extra-bits, a match length of 3–258 bytes.
- 286, 287: not used, reserved and illegal but still part of the tree.

A match length code will always be followed by a distance code. Based on the distance code read, further "extra" bits may be read in order to produce the final distance. The distance tree contains space for 32 symbols:

- 0–3: distances 1–4
- 4–5: distances 5–8, 1 extra bit
- 6–7: distances 9–16, 2 extra bits
- 8–9: distances 17–32, 3 extra bits
- ...
- 26–27: distances 8,193–16,384, 12 extra bits
- 28–29: distances 16,385–32,768, 13 extra bits
- 30–31: not used, reserved and illegal but still part of the tree.

Note that for the match distance symbols 2–29, the number of extra bits can be calculated as $\frac{n}{2} - 1$.

2 Encoder/compressor

During the compression stage, it is the *encoder* that chooses the amount of time spent looking for matching strings. The zlib/gzip reference implementation allows the user to select from a sliding scale of likely resulting compression-level vs. speed of encoding. Options range from -0 (do not attempt compression, just store uncompressed) to -9 representing the maximum capability of the reference implementation in zlib/gzip.

Other Deflate encoders have been produced, all of which will also produce a compatible bitstream capable of being decompressed by any existing Deflate decoder. Differing implementations will likely produce variations on the final encoded bit-stream produced. The focus with non-zlib versions of an encoder has normally been to produce a more efficiently compressed and smaller encoded stream.

2.1 Deflate64/Enhanced Deflate

Deflate64, specified by PKWare, is a proprietary variant of the Deflate procedure. The fundamental mechanisms remain the same. What has changed is the increase in dictionary size from 32kB to 64kB, an addition of 14 bits to the distance codes so that they may address a range of 64kB, and the length code has been extended by 16 bits so that it may define lengths of 3 to 65538 bytes.[4] This leads to Deflate64 having a slightly higher compression ratio and a slightly lower compression time than Deflate.[5] Several free and/or open source projects support Deflate64, such as 7-Zip,[6] while others, such as zlib, do not, as a result of the proprietary nature of the procedure[7] and the very modest performance increase over Deflate.[8]

3 Using Deflate in new software

Implementations of Deflate are freely available in many languages. C programs typically use the zlib library (licensed under the zlib License, which allows use with both free and proprietary software). Programs written using the Borland dialects of Pascal can use paszlib; a C++ library is included as part of 7-Zip/AdvanceCOMP. Java includes support as part of the standard library (in java.util.zip). Microsoft .NET Framework 2.0 base class library supports it in the System.IO.Compression namespace.

3.1 Encoder implementations

- PKZIP: the first implementation, originally done by Phil Katz as part of PKZip.
- zlib/gzip: standard reference implementation used in a huge amount of software, owing to public availability of the source code and a license allowing inclusion into other software.
  - jzlib: Rewrite/re-implementation/port of the zlib encoder into pure Java and distributed under a BSD license. (Fully featured replacement for java.util.zip).
  - PasZLIB: Translation/port of the zlib code into Pascal source code by Jacques Nomssi-Nzali.
  - gziplite: Minimalist rework of gzip / gunzip with minimal memory requirement, also supporting on-the-fly data compression/decompression (no need to bufferize all input) and input/output to/from memory.
  - pako: full featured zlib port to JavaScript, optimized for high speed. Works in browsers and node.js.
- miniz – Public domain Deflate/Inflate implementation with a zlib-compatible API in a single .C source file
- lodepng by Lode Vandevenne. A BSD-licensed single file PNG file reader with built-in C++ Inflate implementation and no external dependencies.
- KZIP/PNGOUT: an encoder by the game-programmer Ken Silverman using “an exhaustive search of all patterns” and "[an] advanced block splitter”.
- PuZip: designed for Commodore 64/C128 computers. PuZip is limited to an 8kB LZ77 window size, with only the store (type 00) and fixed Huffman (type 01) methods.
- BigSpeed Deflate: “Tiny in-memory compression library” available as a MS Windows DLL limited to

32kB blocks at a time and three compression settings.

- BJWFlate & DeflOpt/DeflOpt: Ben Jos Walbeehm's utilities "designed to attempt to squeeze every possible byte out of the files it compresses". Note that the author has stopped development on BJWFlate (but not DeflOpt) in March 2004.
- Crypto++: contains a public domain implementation in C++ aimed mainly at reducing potential security vulnerabilities. The author, Wei Dai states "This code is less clever, but hopefully more understandable and maintainable [than zlib]".
- DeflateStream - an implementation of a stream that performs DEFLATE compression, it is packaged with the Base Class Library included with the .NET Framework.
- ParallelDeflateOutputStream – an open source stream that implements a parallel (multi-thread) deflating stream, for use in .NET programs.
- DotNetCompression – a managed C# implementation of DEFLATE/ZLIB/GZIP that is optimized for high speed, conforms to the streaming API of System.IO.Compression and includes assemblies for .NET Framework, .NET Compact Framework, Xamarin.iOS, Xamarin.Android, Xamarin.Mac, Windows Phone, Xbox 360, Silverlight, Mono and as a Portable Class Library.
- 7-Zip/AdvanceCOMP: written by Igor Pavlov in C++, this version is freely licensed and tends to achieve higher compression than zlib at the expense of CPU usage. Has an option to use the DEFLATE64 storage format.
- deflate.s7i/gzip.s7i, a pure-Seed7 implementation of Deflate and gzip compression, by Thomas Mertes. Made available under the GNU LGPL license.
- PuTTY 'sshzlib.c': a standalone implementation, capable of full decode, but static tree only creation, by Simon Tatham. MIT licensed.
- Halibut 'deflate.c': a standalone implementation capable of full decode. Forked from PuTTY's 'sshzlib.c', but extended to write dynamic Huffman trees and provides Adler-32 and CRC-32 checksum support.
- Plan 9 from Bell Labs operating system's libflate implements deflate compression.
- Hyperbac: uses its own proprietary lossless compression library (written in C++ and Assembly) with an option to implement the DEFLATE64 storage format.
- zip.js: JavaScript implementation.
- Zopfli: C implementation by Google that achieves highest compression at the expense of CPU usage. Apache licensed.

AdvanceCOMP uses the higher compression ratio version of Deflate as implemented by 7-Zip (or optionally Zopfli in recent versions) to enable recompression of gzip, PNG, MNG and ZIP files with the possibility of achieving smaller file sizes than zlib is able to at maximum settings. An even more effective (but also more user-input-demanding and CPU intensive) Deflate encoder is employed inside Ken Silverman's KZIP and PNGOUT utilities, although recent versions of AdvanceCOMP have surpassed KZIP and PNGOUT when using AdvanceCOMP's Zopfli mode.

3.2 Hardware encoders

- AHA361-PCIX/AHA362-PCIX from Comtech AHA. Comtech produced a PCI-X card (PCI-ID: 193f:0001) capable of compressing streams using Deflate at a rate of up to 3.0 Gbit/s (375 MB/s) for incoming uncompressed data. Accompanying the Linux kernel driver for the AHA361-PCIX is an "ahagzip" utility and customised "mod_deflate_aha" capable of using the hardware compression from Apache. The hardware is based on a Xilinx Virtex FPGA and four custom AHA3601 ASICs. The AHA361/AHA362 boards are limited to only handling static Huffman blocks and require software to be modified to add support — the cards were not able to support the full Deflate specification, meaning they could only reliably decode their own output (a stream that did not contain any dynamic Huffman type 2 blocks).
- StorCompress 300/MX3 from Indra Networks. This is a range of PCI (PCI-ID: 17b4:0011) or PCI-X cards featuring between one and six compression engines with claimed processing speeds of up to 3.6 Gbit/s (450 MB/s). A version of the cards are available with the separate brand *WebEnhance* specifically designed for web-serving use rather than SAN or backup use; a PCIe revision, the MX4E is also produced.
- AHA363-PCIe/AHA364-PCIe/AHA367-PCIe. In 2008, Comtech started producing two PCIe cards (PCI-ID: 193f:0363/193f:0364) with a new hardware AHA3610 encoder chip. The new chip was designed to be capable of a sustained 2.5Gbit/s. Using two of these chips, the AHA363-PCIe board can process Deflate at a rate of up to 5.0 Gbit/s (625 MB/s) using the two channels (two compression and two decompression). The AHA364-PCIe variant is an encode-only version of the card designed for outgoing load balancers and instead has multiple register sets to allow 32 independent *virtual* compres-

sion channels feeding two physical compression engines. Linux, Microsoft Windows, and OpenSolaris kernel device drivers are available for both of the new cards, along with a modified zlib system library so that dynamically linked applications can automatically use the hardware support without internal modification. The AHA367-PCIe board (PCI-ID: 193f:0367) is similar to the AHA363-PCIe but uses four AHA3610 chips for a sustained compression rate of 10 Gbit/s (1250 MB/s). Unlike the AHA362-PCIX, the decompression engines on the AHA363-PCIe and AHA367-PCIe boards are fully deflate compliant.

- Nitrox and Octeon processors from Cavium, Inc. contain high-speed hardware deflate and inflate engines compatible with both ZLIB and GZIP with some devices able to handle multiple simultaneous data streams.

4 Decoder/decompressor

Inflate is the decoding process that takes a Deflate bit stream for decompression and correctly produces the original full-size data or file.

4.1 Inflate-only implementations

The normal intent with an alternative Inflate implementation is highly optimised decoding speed, or extremely predictable RAM usage for micro-controller embedded systems.

- Assembly
  - 6502 inflate, written by Piotr Fusik in 6502 assembly language.
  - Elektronika MK-90 inflate, the above 6502 program ported by Piotr Piatek to the PDP-11 architecture.
  - SAMflate, written by Andrew Collier in Z80 assembly language with optional memory paging support for the SAM Coupé, and made available under the BSD/GPL/LGPL/DFSG licenses.
  - inflate.asm, a fast and efficient implementation in M68000 machine language, written by Keir Fraser and released into the Public Domain.
- C/C++
  - kunzip by Michael Kohn and unrelated to "KZIP". Comes with C source-code under the GNU LGPL license. Used in the GIMP installer.
  - puff.c (zlib), a small, unencumbered, single-file reference implementation included in the /contrib/puff directory of the zlib distribution.
  - tinf written by Jørgen Ibsen in ANSI C and comes with zlib license. Adds about 2k code.
  - tinfl.c (miniz), Public domain Inflate implementation contained entirely in a single C function.
- PCDEZIP, Bob Flanders and Michael Holmes, published in PC Magazine 1994–01–11.
- inflate.cl by John Foderaro. Self-standing Common Lisp decoder distributed with a GNU LGPL license.
- inflate.s7i/gzip.s7i, a pure-Seed7 implementation of Deflate and gzip decompression, by Thomas Mertes. Made available under the GNU LGPL license.
- pyflate, a pure-Python stand-alone Deflate (gzip) and bzip2 decoder by Paul Sladen. Written for research/prototyping and made available under the BSD/GPL/LGPL/DFSG licenses.
- deflatelua, a pure-Lua implementation of Deflate and gzip/zlib decompression, by David Manura.
- inflate a pure-Javascript implementation of Inflate by Chris Dickinson
- pako: JavaScript speed-optimized port of zlib. Contains separate build with inflate only.

4.2 Hardware decoders

- Serial Inflate GPU from BitSim. Hardware implementation of Inflate. Part of BitSim's *BADGE* (Bitsim Accelerated Display Graphics Engine) controller offering for embedded systems.

5 See also

- List of archive formats
- List of file archivers
- Comparison of file archivers

6 References

[1] L. Peter Deutsch (May 1996). *DEFLATE Compressed Data Format Specification version 1.3*. IETF. p. 1. sec. Abstract. RFC 1951. https://tools.ietf.org/html/rfc1951#section-Abstract. Retrieved 2014-04-23.

[2] US patent 5051745, Katz, Phillip W., "String searcher, and compressor using same", published 1991-09-24, issued 1991-09-24

[3] David, Salomon (2007). *Data Compression: The Complete Reference* (4 ed.). Springer. p. 241. ISBN 978-1-84628-602-5.

[4] Binary Essence - Deflate64

[5] Binary Essence - "Calgary Corpus" compression comparisons

[6] 7-Zip Manual and Documentation - compression Method

[7] History of Lossless Data Compression Algorithms - Deflate64

[8] zlib FAQ - Does zlib support the new "Deflate64" format introduced by PKWare?

7 External links

- PKWARE, Inc.'s appnote.txt, *.ZIP File Format Specification*; Section 10, *X. Deflating - Method 8*.
- RFC 1951 – *Deflate Compressed Data Format Specification version 1.3*
- zlib Home Page
- *An Explanation of the Deflate Algorithm* by Antaeus Feldspar.
- *Extended Application of Suffix Trees to Data Compression* An excellent algorithm to implement Deflate by Jesper Larsson

8 Text and image sources, contributors, and licenses

8.1 Text

- **DEFLATE** *Source:* https://en.wikipedia.org/wiki/DEFLATE?oldid=672745787 *Contributors:* Damian Yerrick, Tobias Hoevekamp, Zundark, Ap, Olivier, Edward, J-Wiki, Den fjättrade ankan~enwiki, Timwi, Rvalles, Fibonacci, RedWolf, Diberri, Enochlau, DocWatson42, Inkling, OverlordQ, Kate, Sladen, Vague Rant, Bo Lindbergh, Andrejj, Plugwash, Evice, K12u, Minghong, Wrs1864, Paul1337, Danhash, Cristan, Pmberry, Deeahbz, Aottley, Marudubshinki, Graham87, Josh Parris, Ketiltrout, Rjwilmsi, Dieter Schmeer, Yar Kramer, FlaBot, Chobot, 121a0012, YurikBot, Hydrargyrum, CambridgeBayWeather, Sangwine, Brandon, Off!, Unforgiven24, JLaTondre, Wilsynet, DmitriyV, SmackBot, Mmernex, Slashme, KelleyCook, JorgePeixoto, Chris the speller, Kurykh, Jacob Poon, Sommers, Cybercobra, MagnaMopus, OS2Warp, 0xF, Jac16888, BobC32, Boemanneke, Thijs!bot, Bobblehead, Remram44, Widefox, Sterrys, CountingPine, SpeckMade, Raise exception, Mikeakohn, Piotr433, Borber, Mrh30, DrSlony, Hqb, Alexbrn, Aspects, Int21h, Goodone121, DanielPharos, AgnosticPreachersKid, Viktorkallas, IsmaelLuceno, Addbot, AndersBot, OlEnglish, Luckas-bot, DeluxNate, Craigster0, KamikazeBot, 4th-otaku, AnomieBOT, Pur-ja, Blenheimears, FrescoBot, Citrin.ru, Dimafogo, Noloader, SlowByte, Tolly4bolly, Maschen, Richgel999, MelbourneStar, Helpful Pixie Bot, BG19bot, Alexander Philippou, EoRdE6, Pink kitty111, Nazlax10 and Anonymous: 79

8.2 Images

- **File:Question_book-new.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*

  Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:* Tkgd2007
- **File:Symbol_template_class.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/5/5c/Symbol_template_class.svg *License:* Public domain *Contributors:* ? *Original artist:* ?

8.3 Content license

- Creative Commons Attribution-Share Alike 3.0