Registry: Zarr Codecs

CheckList

Numcodecs

- [x] Base64
- [x] Categorize
- [x] Delta
- [x] FixedScaleOffset
- [x] Blosc
- [x] GZIP
- [x] BZ2
- [x] LZMA
- [x] LZ4
- [x] Zlib (DEFLATE)
- [x] ZStandard (ZSTD)
- [x] PackBits
- [x] Pickle
- [x] Quantize
- [x] Shuffle
- [x] ZFPY
- [x] Astype
- [x] Bitround
- [x] JSON

Others

- [x] Deflate64 / Enhanced Deflate
- [x] Snappy
- [x] LZS
- [x] LZSS
- [x] LZO
- [x] LZFSE
- [x] ZFP
- [x] LZRW
- [x] LZX
- [x] LZWL
- [x] LZ4HC
- [x] LZW
- [x] LZF
- [x] GZIP

ImageCodecs

- [x] AEC
- [x] APNG
- [x] AVIF
- [x] Brotli
- [x] Brunsli

Base64

| CodecID - `base64` |- Provides Base64 encoding and decoding functions as defined by RFC 2045 (section 6.8). and used for Encoding and Decoding an object into ASCII format. Can handle both encode and decode operations seemlessly because of fast speeds. This codec operates directly on byte streams, and not character streams, it is hard-coded to only encode/decode character encodings which are compatible with the lower 127 ASCII chart (ISO-8859-1, Windows-1252, UTF-8, etc).

Related Links: Wikipedia Base64

Categorize

| CodecID - `categorize` |- Filters encoding categorical string data as integers. When provided with a set of strings it converts and represents them as integers .i.e **categorize them**. **Example**: **converts** (['male', 'female', 'female', 'male', 'unexpected'], dtype=object) **to** ([2, 1, 1, 2, 0], dtype=uint8)

Related Links: Source Code

Delta

| CedecID - `delta` |- Codec to encode data as the difference between adjacent values. Suitable for array with uniform element distribution .i.e having same distance from each other in magnitude. **Example**: **Converts** ([100, 102, 104, 106, 108, 110, 112, 114, 116, 118]) **to** ([100, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int8)

Related Links: Wikipedia Source Code

FixedScaleOffset

| CodecID - `fixedscaleoffset` |- Simplified version of the scale-offset filter available in HDF5. Applies the transformation **`(x - offset) * scale`** to all chunks. Results are rounded to the nearest integer but are not packed according to the minimum number of bits.

Related Links: Source Code Defination

PackBits

| CodecID - `packbits` |- Codec to pack elements of a boolean array into bits in a uint8 array. A PackBits data stream consists of packets with a one-byte header followed by data. Apple introduced the PackBits format and it is used in TIFF-files. **Example: Converts** ([True, False, False, True], dtype=bool) **To** ([4, 144], dtype=uint8)

Related Links: Source Code Wikipedia

Pickle

| CodecID - `pickle` |- Codec to encode data as as pickled bytes. Useful for encoding an array of Python string objects. To use the codec, **`f = codecs.Pickle()`**

Related Links: Source Code Information

Quantize

| CodecID - `quantize` |- Lossy filter to reduce the precision of floating point data. To use this codec: **codec = numcodecs.Quantize(digits=1, dtype='f8')**

Related Links: Source Code

Shuffle

| CodecID - `shuffle` |- Codec providing shuffle

Related Links: Source Code

ZFPY

| CodecID - `zfpy` |- Codec providing compression using zfpy via the Python standard library.

Related Links: Source Code

Astype

| CodecID - `astype` |- Filter to convert data between different types.

Related Links: Source Code

Bitround

| CodecID - `bitround` |- Floating-point bit rounding codec Drops a specified number of bits from the floating point mantissa, leaving an array more amenable to compression. The number of bits to keep should be determined by an information analysis of the data to be compressed.

Related Links: Source Code

JSON

| CodecID - `json2` |- Codec to encode data as JSON. Useful for encoding an array of Python objects.

Related Links: Source Code

Zlib (DEFLATE)

| CodecID - `zlib` | - | Zlib uses a combination of LZ77 and Huffman coding to compress data.| While the decompression algorithm is the inflate method, which takes a deflate bit stream for decompression and correctly produces the original full-size data or file. Compresses web pages and CSS files before sending them to the browser. And therefore drastically reduces transfer time since the files are much smaller. Zlib provides a good balance of speed and compression

efficiency which makes it a widely used CoDec. The Dictionary size used is 32 KB.

â€“ Related links: https://zlib.net/ https://en.wikipedia.org/wiki/Zlib https://github.com/madler/zlib

Deflate64 / Enhanced Deflate

|â€“ Deflate64 is a proprietary variant of Deflate and uses the same algorithm. |- â€“ Used to compress web pages and CSS files before sending them to the browser. â€“ Deflate64 takes longer compression time, But with a slightly higher compression ratio, than Deflate. â€“ Dictionary size is increased from 32 KB to 64 KB as compared to normal Deflate.

â€“ Related links: https://github.com/brianhelba/zipfile-deflate64
https://en.wikipedia.org/wiki/Deflate#Deflate64/Enhanced_Deflate

ZStandard (ZSTD)

| â€“ The Linux kernel has included Zstandard as a compression method for the btrfs and squashfs filesystems. | - â€“ Offers much faster compression and decompression and 2x faster speed and 12-15% smaller files as compared to HHVM binary and bytecode. â€“ Zstd package includes parallel (multi-threaded) implementations of both compression and decompression.

â€“ Related links: https://en.wikipedia.org/wiki/Zstd https://github.com/facebook/zstd https://facebook.github.io/zstd/

Blosc

| â€“ A compressor that uses the blocking technique to reduce activity on the memory bus as much as possible. In short, this blocking technique works by dividing datasets in blocks that are small enough to fit in the caches of modern processors and perform compression/decompression there. |- â€“ Blosc is meant to support all platforms where a C89 compliant C compiler can be found, such as Intel (Linux, Mac OSX and Windows) and ARM (Linux). â€“ Designed to transfer data to the processor cache faster than the traditional, non-compressed. Blosc compressor not only reduces the size of large datasets but also accelerates memory related computations. â€“ Capable of using multi threading capabilities of CPU, in order to make the compression and decompression process fast.

â€“ Related links: https://github.com/Blosc/c-blosc https://www.blosc.org/pages/blosc-in-depth/

Brotli

| â€“ Brotli is a generic-purpose lossless compression algorithm that compresses data using a combination of a modern variant of the LZ77 algorithm, Huffman coding and 2nd order context modeling. |- â€“ Primarily used to compress HTTP content in web servers and content delivery networks that results in making internet websites load faster. â€“ Supported and used by major web browsers due to better compression capabilities than gzip. It is similar in speed as compared to deflate but offers more dense compression. â€“ Compression efficiency. JavaScript files compressed are roughly 15% smaller, CSS files are around 16% smaller and HTML files are around 20% smaller.

â€“ Related links: https://github.com/google/brotli https://en.wikipedia.org/wiki/Brotli

Snappy

| â€“ A compression format and program library introduced by Google and defined as a raw stream format, plus a higher-level "framing format" that can be used as a file format. |- â€“ Majorly used in compressing plain text, HTML, PNGs and JPEGs â€“ Compression efficiency. Compression ratio of 1.5-1.7x for plain text, about 2-4x for HTML, and 1.0x for PNGs, JPEGs and other already-compressed data. â€“ The compression speeds are very fast which is achieved by losing on the compression ratios, Therefore the resulting output is 20-100% larger than that of other libraries.

â€“ Related links: https://en.wikipedia.org/wiki/Snappy_(compression) https://github.com/google/snappy

LZMA

| CodecID - `lzma` |- | LZMA, A lossless data compression algorithm that uses a dictionary compression algorithm (a variant of LZ77 with huge dictionary sizes and special support for repeatedly used match distances) using a complex model to make a probability prediction of each bit. Mainly used in Unix Operating system and kind of the same data compression algorithm as Zstd. Compression is really high as compared to other algorithms. Uses .lzma as file extension and variable dictionary size.

â€“ Related links: Wikipedia Github

LZS

| â€“ LZS uses a combination of fixed Huffman coding and sliding window compression algorithm. It uses the last 2 KB of uncompressed data as a sliding-window dictionary. An LZS compressor looks for matches between the data to be compressed and the last 2 KB of data. If it finds a match, it encodes an offset/length reference to the dictionary. |- â€“ It was originally developed for tape compression, and subsequently adapted for hard disk compression and later specified as a compression algorithm for various network protocols. â€“ Compression and decompression speeds are fairly good as compared to other CoDec. â€“ Dictionary size is variable and is similar to LZ77 CoDec.

â€“ Related links: Wikipedia GitHub

LZSS

| â€“ A lossless data compression algorithm derived from LZ77, That uses a dictionary coding technique. It attempts to replace a string of symbols with a reference to a dictionary location of the same string. |- â€“ Apple's Mac OS X uses LZSS as one of the compression methods for kernel code. â€“ Many popular archivers like PKZip, ARJ, RAR, ZOO, LHarc use LZSS as the primary compression algorithm.

â€“ Related links: Wikipedia GitHub

BZ2

| CodecID - `bz2` | - BZ2 or BZip2 is a free and open-source compression technique that uses Burrowsâ€“Wheeler algorithm and only compresses single files and not a file archiver. | Mostly used in big data applications with cluster computing frameworks like Hadoop and Apache Spark. | More effective than the older LZW and Deflate compression algorithms, but is considerably slower. LZMA is generally more space-efficient than bzip2 at the expense of even slower compression speed, while having much faster decompression. |Compression ratio varies between 10% to 15% as compared to the best CoDec algorithm.

â€“ Related links: Wikipedia Python-BZ2 GitHub

LZ4,

| â€“ The LZ4 algorithm works by representing the data as a series of sequences. Each sequence begins with a one-byte token that is broken into two 4-bit fields. The first field represents the number of literal bytes that are to be copied to the output. |- â€“ This algorithm provides a good combination of speed and compression. But not very useful in the latest test cases. â€“ Provides a smaller or we can say worse compression ratio than LZO algorithm, and even more worse when compared to DEFLATE. â€“ Uses .lz4 as file extension.

â€“ Related links: GitHub Wikipedia

LZ4HC,

| â€“ LZ4HC is a "high-compression" variant of LZ4 in which the compressor finds more than one match between current and past data and looks for the best match to ensure the output is small. |- â€“ The Apache Hadoop system uses this algorithm for fast compression. LZ4 was also implemented natively in the Linux kernel. â€“ LZ4HC has a high compression ratio but low compression speed as compared to LZ4. â€“ LZ4HC compression makes it possible to load individual Assets from an Asset Bundle quickly and using less memory than LZMA compressed Asset Bundles.

â€“ Related LInks: Github Doc

LZW,

| â€“ LZW compression uses a table-based lookup algorithm that identifies repeated sequences in the data and adds them to the code table. Also the Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents. |- â€“ Most commonly used to compress GIF image format served from Web sites and the TIFF image format. â€“ The advantage of LZW is its simplicity, fast execution.

â€“ Related Links: Doc Github

LZX

| â€“ LZX is a LZ77 family compression algorithm. Like LZ77, Itâ€™s also a dictionary codec and uses a sliding window during compression. |- â€“ Mainly used in Amiga LZX, Microsoft Cabinet files, Microsoft Compressed HTML Help (CHM) files, Microsoft Reader (LIT) files, CompactOS NTFS file compression etc. â€“ As LZX is used widely, the compression and decompression speed may vary as per its Instances of use.

â€“ Related Links: Wikipedia

LZWL

| â€“ LZWL is a syllable-based variant of the character-based LZW compression algorithm and works over the alphabet of syllables therefore this algorithm can be used with words also. |- â€“ Being a variant of LZW compression, uses are also the same. â€“ Compression speeds are fast and even more fast in selective cases.

â€“ Related Links: Wikipedia

LZF,

| â€“ A Java library with no Huffman-encoding is used for encoding and decoding data in LZF format. |- â€“ This algorithm is optimized for speed with an average compression ratio. â€“ When compared to standard Deflate algo LZF can give 5 to 6 times faster compression and 2 times faster decompression. â€“ Compression rate is lower since no Huffman-encoding is used.

LZO

| â€“ LZO is a block compression algorithm as it compresses and decompresses blocks of data. Block size must be the same for compression and decompression. LZO compresses a block of data into matches (a sliding dictionary) and runs of non-matching literals to produce good results on highly redundant data |- â€“ Higher compression and decompression speeds as compared to DEFLATE compression. â€“ Requires an additional buffer during compression (of size 8 kB or 64 kB, depending on compression level) â€“ Allows the user to adjust the balance between compression ratio and compression speed, without affecting the speed of decompression.

â€“ Related Links: Github Wikipedia Website

LZFSE

| â€“ LZFSE (Lempelâ€“Ziv Finite State Entropy) is an open source lossless data compression algorithm created by Apple Inc. It was released with a simpler algorithm called LZVN. |- â€“ LZFSE is used for compressing a payload for iOS, macOS, watchOS, and tvOS. â€“ LZFSE compresses with a ratio comparable to that of zlib (DEFLATE) and decompresses two to three times faster while using fewer resources, therefore offering higher energy efficiency than zlib. â€“ LZFSE is similar in speed to ZSTD (level 6), but has a slightly worse ratio.

â€“ Related Links: Wikipedia GitHub

LZJB

| â€“ LZJB is a lossless data compression algorithm to compress crash dumps and data in ZFS. Basically it is the improved version of LZRW1 algorithm |- â€“ Pending due to Less info available.

â€“ Related links: Wikipedia

GZIP

| CodecID - `gzip` |- | Gzip is a file format and a software application used for file compression and decompression. gzip is based on the DEFLATE algorithm, which is a combination of Huffman coding and LZ77. DEFLATE was intended as a replacement for LZW and other patent-encumbered data compression algorithms which, at the time, limited the usability of compress and other popular archivers. | The zlib stream format, DEFLATE, and the gzip file format were standardized respectively as RFC 1950, RFC 1951, and RFC 1952.

-- Related Links: Wikipedia GitHub

ZFP

| ZFP was designed to achieve high compression ratios and therefore uses lossy but optionally error-bounded compression. | - | Bit-for-bit lossless compression of integer and floating-point arrays is also supported. | It provides compressed-array classes that support high throughput read and write random access to individual array elements and also supports serial and parallel (OpenMP and CUDA) compression of whole arrays.

Related Links: Official Site Github

LZRW

| LZRW(Lempelâ€“Ziv Ross Williams) refers to variants of the LZ77 lossless data compression algorithm. |- It's an improved version with better compression speeds due to the use of hash tables and other techniques. â€“ There are many variants of LZRW i.e LZRW1[2], LZRW1-A, LZRW2, LZRW3, LZRW3-A, LZRW4, LZRW5 â€“ Both the LZRW1-A and LZRW3 use the same hash table lookups to identify matches, therefore provides (almost) identical compression results.

â€“ Related links: Comparison Wikipedia

AEC

| Acoustic Echo Cancellation (AEC) is used to efficiently process the microphone audio so that audio coming from the loudspeaker is removed from the microphone signal. |- Acoustic echo cancellation (AEC) technology is a common addition to today's conference phones. The technology recognizes and eliminates the echo as quickly as possible without affecting the audio quality.

â€“ Related links: Wikipedia Link

APNG

| APNG i.e Animated Portable Network Graphics is an extended File format verison of PNG. |- It permits animated images that work similarly to animated GIF files. It supports 24-bit images and 8-bit transparency that is not available for GIFs. Provides backward compatibility with non-animated PNG files.

Related Links: Source Code Wikipedia

AVIF

| AV1 Image File Format (AVIF) is an image file format specification for storing images or image sequences compressed with AV1 in the HEIF container format. |- It competes with HEIC, which uses the same container format built upon ISOBMFF, but HEVC for compression. AVIF gives better compression efficiency than JPEG in terms of better detail preservation, less blocking artifacts and less color bleeding around hard edges in composites of natural images, text, and graphics.

Related Links: Source Code Wikipedia

Brunsli

| Brunsli works as a transcoder converting between JPEG and JPEG XL. |- Brunsli results in 22% decrease in file size while allowing the original JPEG to be recovered byte-by-byte.

Related Links: Source Code