/\*\*

\* The `zlib` module provides compression functionality implemented using Gzip,

\* Deflate/Inflate, and Brotli.

\*

\* To access it:

\*

\* ```js

\* const zlib = require('zlib');

\* ```

\*

\* Compression and decompression are built around the Node.js `Streams API`.

\*

\* Compressing or decompressing a stream (such as a file) can be accomplished by

\* piping the source stream through a `zlib` `Transform` stream into a destination

\* stream:

\*

\* ```js

\* const { createGzip } = require('zlib');

\* const { pipeline } = require('stream');

\* const {

\* createReadStream,

\* createWriteStream

\* } = require('fs');

\*

\* const gzip = createGzip();

\* const source = createReadStream('input.txt');

\* const destination = createWriteStream('input.txt.gz');

\*

\* pipeline(source, gzip, destination, (err) => {

\* if (err) {

\* console.error('An error occurred:', err);

\* process.exitCode = 1;

\* }

\* });

\*

\* // Or, Promisified

\*

\* const { promisify } = require('util');

\* const pipe = promisify(pipeline);

\*

\* async function do\_gzip(input, output) {

\* const gzip = createGzip();

\* const source = createReadStream(input);

\* const destination = createWriteStream(output);

\* await pipe(source, gzip, destination);

\* }

\*

\* do\_gzip('input.txt', 'input.txt.gz')

\* .catch((err) => {

\* console.error('An error occurred:', err);

\* process.exitCode = 1;

\* });

\* ```

\*

\* It is also possible to compress or decompress data in a single step:

\*

\* ```js

\* const { deflate, unzip } = require('zlib');

\*

\* const input = '.................................';

\* deflate(input, (err, buffer) => {

\* if (err) {

\* console.error('An error occurred:', err);

\* process.exitCode = 1;

\* }

\* console.log(buffer.toString('base64'));

\* });

\*

\* const buffer = Buffer.from('eJzT0yMAAGTvBe8=', 'base64');

\* unzip(buffer, (err, buffer) => {

\* if (err) {

\* console.error('An error occurred:', err);

\* process.exitCode = 1;

\* }

\* console.log(buffer.toString());

\* });

\*

\* // Or, Promisified

\*

\* const { promisify } = require('util');

\* const do\_unzip = promisify(unzip);

\*

\* do\_unzip(buffer)

\* .then((buf) => console.log(buf.toString()))

\* .catch((err) => {

\* console.error('An error occurred:', err);

\* process.exitCode = 1;

\* });

\* ```

\* @since v0.5.8

\* @see [source](https://github.com/nodejs/node/blob/v17.0.0/lib/zlib.js)

\*/

declare module 'zlib' {

import \* as stream from 'node:stream';

interface ZlibOptions {

/\*\*

\* @default constants.Z\_NO\_FLUSH

\*/

flush?: number | undefined;

/\*\*

\* @default constants.Z\_FINISH

\*/

finishFlush?: number | undefined;

/\*\*

\* @default 16\*1024

\*/

chunkSize?: number | undefined;

windowBits?: number | undefined;

level?: number | undefined; // compression only

memLevel?: number | undefined; // compression only

strategy?: number | undefined; // compression only

dictionary?: NodeJS.ArrayBufferView | ArrayBuffer | undefined; // deflate/inflate only, empty dictionary by default

info?: boolean | undefined;

maxOutputLength?: number | undefined;

}

interface BrotliOptions {

/\*\*

\* @default constants.BROTLI\_OPERATION\_PROCESS

\*/

flush?: number | undefined;

/\*\*

\* @default constants.BROTLI\_OPERATION\_FINISH

\*/

finishFlush?: number | undefined;

/\*\*

\* @default 16\*1024

\*/

chunkSize?: number | undefined;

params?:

| {

/\*\*

\* Each key is a `constants.BROTLI\_\*` constant.

\*/

[key: number]: boolean | number;

}

| undefined;

maxOutputLength?: number | undefined;

}

interface Zlib {

/\*\* @deprecated Use bytesWritten instead. \*/

readonly bytesRead: number;

readonly bytesWritten: number;

shell?: boolean | string | undefined;

close(callback?: () => void): void;

flush(kind?: number, callback?: () => void): void;

flush(callback?: () => void): void;

}

interface ZlibParams {

params(level: number, strategy: number, callback: () => void): void;

}

interface ZlibReset {

reset(): void;

}

interface BrotliCompress extends stream.Transform, Zlib {}

interface BrotliDecompress extends stream.Transform, Zlib {}

interface Gzip extends stream.Transform, Zlib {}

interface Gunzip extends stream.Transform, Zlib {}

interface Deflate extends stream.Transform, Zlib, ZlibReset, ZlibParams {}

interface Inflate extends stream.Transform, Zlib, ZlibReset {}

interface DeflateRaw extends stream.Transform, Zlib, ZlibReset, ZlibParams {}

interface InflateRaw extends stream.Transform, Zlib, ZlibReset {}

interface Unzip extends stream.Transform, Zlib {}

/\*\*

\* Creates and returns a new `BrotliCompress` object.

\* @since v11.7.0, v10.16.0

\*/

function createBrotliCompress(options?: BrotliOptions): BrotliCompress;

/\*\*

\* Creates and returns a new `BrotliDecompress` object.

\* @since v11.7.0, v10.16.0

\*/

function createBrotliDecompress(options?: BrotliOptions): BrotliDecompress;

/\*\*

\* Creates and returns a new `Gzip` object.

\* See `example`.

\* @since v0.5.8

\*/

function createGzip(options?: ZlibOptions): Gzip;

/\*\*

\* Creates and returns a new `Gunzip` object.

\* @since v0.5.8

\*/

function createGunzip(options?: ZlibOptions): Gunzip;

/\*\*

\* Creates and returns a new `Deflate` object.

\* @since v0.5.8

\*/

function createDeflate(options?: ZlibOptions): Deflate;

/\*\*

\* Creates and returns a new `Inflate` object.

\* @since v0.5.8

\*/

function createInflate(options?: ZlibOptions): Inflate;

/\*\*

\* Creates and returns a new `DeflateRaw` object.

\*

\* An upgrade of zlib from 1.2.8 to 1.2.11 changed behavior when `windowBits`is set to 8 for raw deflate streams. zlib would automatically set `windowBits`to 9 if was initially set to 8\. Newer

\* versions of zlib will throw an exception,

\* so Node.js restored the original behavior of upgrading a value of 8 to 9,

\* since passing `windowBits = 9` to zlib actually results in a compressed stream

\* that effectively uses an 8-bit window only.

\* @since v0.5.8

\*/

function createDeflateRaw(options?: ZlibOptions): DeflateRaw;

/\*\*

\* Creates and returns a new `InflateRaw` object.

\* @since v0.5.8

\*/

function createInflateRaw(options?: ZlibOptions): InflateRaw;

/\*\*

\* Creates and returns a new `Unzip` object.

\* @since v0.5.8

\*/

function createUnzip(options?: ZlibOptions): Unzip;

type InputType = string | ArrayBuffer | NodeJS.ArrayBufferView;

type CompressCallback = (error: Error | null, result: Buffer) => void;

/\*\*

\* @since v11.7.0, v10.16.0

\*/

function brotliCompress(buf: InputType, options: BrotliOptions, callback: CompressCallback): void;

function brotliCompress(buf: InputType, callback: CompressCallback): void;

namespace brotliCompress {

function \_\_promisify\_\_(buffer: InputType, options?: BrotliOptions): Promise<Buffer>;

}

/\*\*

\* Compress a chunk of data with `BrotliCompress`.

\* @since v11.7.0, v10.16.0

\*/

function brotliCompressSync(buf: InputType, options?: BrotliOptions): Buffer;

/\*\*

\* @since v11.7.0, v10.16.0

\*/

function brotliDecompress(buf: InputType, options: BrotliOptions, callback: CompressCallback): void;

function brotliDecompress(buf: InputType, callback: CompressCallback): void;

namespace brotliDecompress {

function \_\_promisify\_\_(buffer: InputType, options?: BrotliOptions): Promise<Buffer>;

}

/\*\*

\* Decompress a chunk of data with `BrotliDecompress`.

\* @since v11.7.0, v10.16.0

\*/

function brotliDecompressSync(buf: InputType, options?: BrotliOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function deflate(buf: InputType, callback: CompressCallback): void;

function deflate(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace deflate {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Compress a chunk of data with `Deflate`.

\* @since v0.11.12

\*/

function deflateSync(buf: InputType, options?: ZlibOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function deflateRaw(buf: InputType, callback: CompressCallback): void;

function deflateRaw(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace deflateRaw {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Compress a chunk of data with `DeflateRaw`.

\* @since v0.11.12

\*/

function deflateRawSync(buf: InputType, options?: ZlibOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function gzip(buf: InputType, callback: CompressCallback): void;

function gzip(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace gzip {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Compress a chunk of data with `Gzip`.

\* @since v0.11.12

\*/

function gzipSync(buf: InputType, options?: ZlibOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function gunzip(buf: InputType, callback: CompressCallback): void;

function gunzip(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace gunzip {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Decompress a chunk of data with `Gunzip`.

\* @since v0.11.12

\*/

function gunzipSync(buf: InputType, options?: ZlibOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function inflate(buf: InputType, callback: CompressCallback): void;

function inflate(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace inflate {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Decompress a chunk of data with `Inflate`.

\* @since v0.11.12

\*/

function inflateSync(buf: InputType, options?: ZlibOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function inflateRaw(buf: InputType, callback: CompressCallback): void;

function inflateRaw(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace inflateRaw {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Decompress a chunk of data with `InflateRaw`.

\* @since v0.11.12

\*/

function inflateRawSync(buf: InputType, options?: ZlibOptions): Buffer;

/\*\*

\* @since v0.6.0

\*/

function unzip(buf: InputType, callback: CompressCallback): void;

function unzip(buf: InputType, options: ZlibOptions, callback: CompressCallback): void;

namespace unzip {

function \_\_promisify\_\_(buffer: InputType, options?: ZlibOptions): Promise<Buffer>;

}

/\*\*

\* Decompress a chunk of data with `Unzip`.

\* @since v0.11.12

\*/

function unzipSync(buf: InputType, options?: ZlibOptions): Buffer;

namespace constants {

const BROTLI\_DECODE: number;

const BROTLI\_DECODER\_ERROR\_ALLOC\_BLOCK\_TYPE\_TREES: number;

const BROTLI\_DECODER\_ERROR\_ALLOC\_CONTEXT\_MAP: number;

const BROTLI\_DECODER\_ERROR\_ALLOC\_CONTEXT\_MODES: number;

const BROTLI\_DECODER\_ERROR\_ALLOC\_RING\_BUFFER\_1: number;

const BROTLI\_DECODER\_ERROR\_ALLOC\_RING\_BUFFER\_2: number;

const BROTLI\_DECODER\_ERROR\_ALLOC\_TREE\_GROUPS: number;

const BROTLI\_DECODER\_ERROR\_DICTIONARY\_NOT\_SET: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_BLOCK\_LENGTH\_1: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_BLOCK\_LENGTH\_2: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_CL\_SPACE: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_CONTEXT\_MAP\_REPEAT: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_DICTIONARY: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_DISTANCE: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_EXUBERANT\_META\_NIBBLE: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_EXUBERANT\_NIBBLE: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_HUFFMAN\_SPACE: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_PADDING\_1: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_PADDING\_2: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_RESERVED: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_SIMPLE\_HUFFMAN\_ALPHABET: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_SIMPLE\_HUFFMAN\_SAME: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_TRANSFORM: number;

const BROTLI\_DECODER\_ERROR\_FORMAT\_WINDOW\_BITS: number;

const BROTLI\_DECODER\_ERROR\_INVALID\_ARGUMENTS: number;

const BROTLI\_DECODER\_ERROR\_UNREACHABLE: number;

const BROTLI\_DECODER\_NEEDS\_MORE\_INPUT: number;

const BROTLI\_DECODER\_NEEDS\_MORE\_OUTPUT: number;

const BROTLI\_DECODER\_NO\_ERROR: number;

const BROTLI\_DECODER\_PARAM\_DISABLE\_RING\_BUFFER\_REALLOCATION: number;

const BROTLI\_DECODER\_PARAM\_LARGE\_WINDOW: number;

const BROTLI\_DECODER\_RESULT\_ERROR: number;

const BROTLI\_DECODER\_RESULT\_NEEDS\_MORE\_INPUT: number;

const BROTLI\_DECODER\_RESULT\_NEEDS\_MORE\_OUTPUT: number;

const BROTLI\_DECODER\_RESULT\_SUCCESS: number;

const BROTLI\_DECODER\_SUCCESS: number;

const BROTLI\_DEFAULT\_MODE: number;

const BROTLI\_DEFAULT\_QUALITY: number;

const BROTLI\_DEFAULT\_WINDOW: number;

const BROTLI\_ENCODE: number;

const BROTLI\_LARGE\_MAX\_WINDOW\_BITS: number;

const BROTLI\_MAX\_INPUT\_BLOCK\_BITS: number;

const BROTLI\_MAX\_QUALITY: number;

const BROTLI\_MAX\_WINDOW\_BITS: number;

const BROTLI\_MIN\_INPUT\_BLOCK\_BITS: number;

const BROTLI\_MIN\_QUALITY: number;

const BROTLI\_MIN\_WINDOW\_BITS: number;

const BROTLI\_MODE\_FONT: number;

const BROTLI\_MODE\_GENERIC: number;

const BROTLI\_MODE\_TEXT: number;

const BROTLI\_OPERATION\_EMIT\_METADATA: number;

const BROTLI\_OPERATION\_FINISH: number;

const BROTLI\_OPERATION\_FLUSH: number;

const BROTLI\_OPERATION\_PROCESS: number;

const BROTLI\_PARAM\_DISABLE\_LITERAL\_CONTEXT\_MODELING: number;

const BROTLI\_PARAM\_LARGE\_WINDOW: number;

const BROTLI\_PARAM\_LGBLOCK: number;

const BROTLI\_PARAM\_LGWIN: number;

const BROTLI\_PARAM\_MODE: number;

const BROTLI\_PARAM\_NDIRECT: number;

const BROTLI\_PARAM\_NPOSTFIX: number;

const BROTLI\_PARAM\_QUALITY: number;

const BROTLI\_PARAM\_SIZE\_HINT: number;

const DEFLATE: number;

const DEFLATERAW: number;

const GUNZIP: number;

const GZIP: number;

const INFLATE: number;

const INFLATERAW: number;

const UNZIP: number;

// Allowed flush values.

const Z\_NO\_FLUSH: number;

const Z\_PARTIAL\_FLUSH: number;

const Z\_SYNC\_FLUSH: number;

const Z\_FULL\_FLUSH: number;

const Z\_FINISH: number;

const Z\_BLOCK: number;

const Z\_TREES: number;

// Return codes for the compression/decompression functions.

// Negative values are errors, positive values are used for special but normal events.

const Z\_OK: number;

const Z\_STREAM\_END: number;

const Z\_NEED\_DICT: number;

const Z\_ERRNO: number;

const Z\_STREAM\_ERROR: number;

const Z\_DATA\_ERROR: number;

const Z\_MEM\_ERROR: number;

const Z\_BUF\_ERROR: number;

const Z\_VERSION\_ERROR: number;

// Compression levels.

const Z\_NO\_COMPRESSION: number;

const Z\_BEST\_SPEED: number;

const Z\_BEST\_COMPRESSION: number;

const Z\_DEFAULT\_COMPRESSION: number;

// Compression strategy.

const Z\_FILTERED: number;

const Z\_HUFFMAN\_ONLY: number;

const Z\_RLE: number;

const Z\_FIXED: number;

const Z\_DEFAULT\_STRATEGY: number;

const Z\_DEFAULT\_WINDOWBITS: number;

const Z\_MIN\_WINDOWBITS: number;

const Z\_MAX\_WINDOWBITS: number;

const Z\_MIN\_CHUNK: number;

const Z\_MAX\_CHUNK: number;

const Z\_DEFAULT\_CHUNK: number;

const Z\_MIN\_MEMLEVEL: number;

const Z\_MAX\_MEMLEVEL: number;

const Z\_DEFAULT\_MEMLEVEL: number;

const Z\_MIN\_LEVEL: number;

const Z\_MAX\_LEVEL: number;

const Z\_DEFAULT\_LEVEL: number;

const ZLIB\_VERNUM: number;

}

// Allowed flush values.

/\*\* @deprecated Use `constants.Z\_NO\_FLUSH` \*/

const Z\_NO\_FLUSH: number;

/\*\* @deprecated Use `constants.Z\_PARTIAL\_FLUSH` \*/

const Z\_PARTIAL\_FLUSH: number;

/\*\* @deprecated Use `constants.Z\_SYNC\_FLUSH` \*/

const Z\_SYNC\_FLUSH: number;

/\*\* @deprecated Use `constants.Z\_FULL\_FLUSH` \*/

const Z\_FULL\_FLUSH: number;

/\*\* @deprecated Use `constants.Z\_FINISH` \*/

const Z\_FINISH: number;

/\*\* @deprecated Use `constants.Z\_BLOCK` \*/

const Z\_BLOCK: number;

/\*\* @deprecated Use `constants.Z\_TREES` \*/

const Z\_TREES: number;

// Return codes for the compression/decompression functions.

// Negative values are errors, positive values are used for special but normal events.

/\*\* @deprecated Use `constants.Z\_OK` \*/

const Z\_OK: number;

/\*\* @deprecated Use `constants.Z\_STREAM\_END` \*/

const Z\_STREAM\_END: number;

/\*\* @deprecated Use `constants.Z\_NEED\_DICT` \*/

const Z\_NEED\_DICT: number;

/\*\* @deprecated Use `constants.Z\_ERRNO` \*/

const Z\_ERRNO: number;

/\*\* @deprecated Use `constants.Z\_STREAM\_ERROR` \*/

const Z\_STREAM\_ERROR: number;

/\*\* @deprecated Use `constants.Z\_DATA\_ERROR` \*/

const Z\_DATA\_ERROR: number;

/\*\* @deprecated Use `constants.Z\_MEM\_ERROR` \*/

const Z\_MEM\_ERROR: number;

/\*\* @deprecated Use `constants.Z\_BUF\_ERROR` \*/

const Z\_BUF\_ERROR: number;

/\*\* @deprecated Use `constants.Z\_VERSION\_ERROR` \*/

const Z\_VERSION\_ERROR: number;

// Compression levels.

/\*\* @deprecated Use `constants.Z\_NO\_COMPRESSION` \*/

const Z\_NO\_COMPRESSION: number;

/\*\* @deprecated Use `constants.Z\_BEST\_SPEED` \*/

const Z\_BEST\_SPEED: number;

/\*\* @deprecated Use `constants.Z\_BEST\_COMPRESSION` \*/

const Z\_BEST\_COMPRESSION: number;

/\*\* @deprecated Use `constants.Z\_DEFAULT\_COMPRESSION` \*/

const Z\_DEFAULT\_COMPRESSION: number;

// Compression strategy.

/\*\* @deprecated Use `constants.Z\_FILTERED` \*/

const Z\_FILTERED: number;

/\*\* @deprecated Use `constants.Z\_HUFFMAN\_ONLY` \*/

const Z\_HUFFMAN\_ONLY: number;

/\*\* @deprecated Use `constants.Z\_RLE` \*/

const Z\_RLE: number;

/\*\* @deprecated Use `constants.Z\_FIXED` \*/

const Z\_FIXED: number;

/\*\* @deprecated Use `constants.Z\_DEFAULT\_STRATEGY` \*/

const Z\_DEFAULT\_STRATEGY: number;

/\*\* @deprecated \*/

const Z\_BINARY: number;

/\*\* @deprecated \*/

const Z\_TEXT: number;

/\*\* @deprecated \*/

const Z\_ASCII: number;

/\*\* @deprecated \*/

const Z\_UNKNOWN: number;

/\*\* @deprecated \*/

const Z\_DEFLATED: number;

}

declare module 'node:zlib' {

export \* from 'zlib';

}