

[Discover Packages](#) > [Standard library](#) > [compress](#) > [flate](#) 

# flate


package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: 9 |

Imported by: 4,997

## Details

 Valid [go.mod](#) file  Redistributable license  Tagged version  Stable version [Learn more](#)

## Repository

[cs.opensource.google/go/go](https://cs.opensource.google/go/go)

## Links

 [Report a Vulnerability](#) Documentation 

## <> Documentation

### Overview

Package flate implements the DEFLATE compressed data format, described in [RFC 1951](#). The gzip and zlib packages implement access to DEFLATE-based file formats.

[▶ Example \(Dictionary\)](#)[▶ Example \(Reset\)](#)[▶ Example \(Synchronization\)](#)

### Index

#### Constants

`func NewReader(r io.Reader) io.ReadCloser``func NewReaderDict(r io.Reader, dict []byte) io.ReadCloser``type CorruptInputError``func (e CorruptInputError) Error() string``type InternalError``func (e InternalError) Error() string``type ReadError` **DEPRECATED**`func (e *ReadError) Error() string``type Reader`

type Resetter

type WriteError DEPRECATED

```
func (e *WriteError) Error() string
```

type Writer

```
func NewWriter(w io.Writer, level int) (*Writer, error)
```

```
func NewWriterDict(w io.Writer, level int, dict []byte) (*Writer, error)
```

```
func (w *Writer) Close() error
```

```
func (w *Writer) Flush() error
```

```
func (w *Writer) Reset(dst io.Writer)
```

```
func (w *Writer) Write(data []byte) (n int, err error)
```

## Examples

[Package \(Dictionary\)](#)

[Package \(Reset\)](#)

[Package \(Synchronization\)](#)

## Constants

[View Source](#)

```
const (  
    NoCompression      = 0  
    BestSpeed           = 1  
    BestCompression    = 9  
    DefaultCompression = -1  
  
    // HuffmanOnly disables Lempel-Ziv match searching and only performs Huffman  
    // entropy encoding. This mode is useful in compressing data that has  
    // already been compressed with an LZ style algorithm (e.g. Snappy or LZ4)  
    // that lacks an entropy encoder. Compression gains are achieved when  
    // certain bytes in the input stream occur more frequently than others.  
    //  
    // Note that HuffmanOnly produces a compressed output that is  
    // RFC 1951 compliant. That is, any valid DEFLATE decompressor will  
    // continue to be able to decompress this output.  
    HuffmanOnly = -2  
)
```

## Variables

This section is empty.

## Functions

func [NewReader](#)

```
func NewReader(r io.Reader) io.ReadCloser
```

NewReader returns a new ReadCloser that can be used to read the uncompressed version of r. If r does not also implement io.ByteReader, the decompressor may read more data than necessary from r. The reader returns io.EOF after the final block in the DEFLATE stream has been encountered. Any trailing data after the final block is ignored.

The ReadCloser returned by NewReader also implements Resetter.

## func NewReaderDict

```
func NewReaderDict(r io.Reader, dict []byte) io.ReadCloser
```

NewReaderDict is like NewReader but initializes the reader with a preset dictionary. The returned Reader behaves as if the uncompressed data stream started with the given dictionary, which has already been read. NewReaderDict is typically used to read data compressed by NewWriterDict.

The ReadCloser returned by NewReader also implements Resetter.

## Types

### type CorruptInputError

```
type CorruptInputError int64
```

A CorruptInputError reports the presence of corrupt input at a given offset.

### func (CorruptInputError) Error

```
func (e CorruptInputError) Error() string
```

### type InternalError

```
type InternalError string
```

An InternalError reports an error in the flate code itself.

### func (InternalError) Error

```
func (e InternalError) Error() string
```

### type ReadError DEPRECATED [Show](#)

### type Reader

```
type Reader interface {  
    io.Reader  
    io.ByteReader  
}
```

The actual read interface needed by `NewReader`. If the passed in `io.Reader` does not also have `ReadByte`, the `NewReader` will introduce its own buffering.

## type `Resetter`

added in go1.4

```
type Resetter interface {  
    // Reset discards any buffered data and resets the Resetter as if it was  
    // newly initialized with the given reader.  
    Reset(r io.Reader, dict []byte) error  
}
```

`Resetter` resets a `ReadCloser` returned by `NewReader` or `NewReaderDict` to switch to a new underlying `Reader`. This permits reusing a `ReadCloser` instead of allocating a new one.

## type `WriteError` DEPRECATED [Show](#)

## type `Writer`

```
type Writer struct {  
    // contains filtered or unexported fields  
}
```

A `Writer` takes data written to it and writes the compressed form of that data to an underlying writer (see `NewWriter`).

## func `NewWriter`

```
func NewWriter(w io.Writer, level int) (*Writer, error)
```

`NewWriter` returns a new `Writer` compressing data at the given level. Following `zlib`, levels range from 1 (`BestSpeed`) to 9 (`BestCompression`); higher levels typically run slower but compress more. Level 0 (`NoCompression`) does not attempt any compression; it only adds the necessary DEFLATE framing. Level -1 (`DefaultCompression`) uses the default compression level. Level -2 (`HuffmanOnly`) will use Huffman compression only, giving a very fast compression for all types of input, but sacrificing considerable compression efficiency.

If level is in the range [-2, 9] then the error returned will be nil. Otherwise the error returned will be non-nil.

## func `NewWriterDict`

```
func NewWriterDict(w io.Writer, level int, dict []byte) (*Writer, error)
```

`NewWriterDict` is like `NewWriter` but initializes the new `Writer` with a preset dictionary. The returned `Writer` behaves as if the dictionary had been written to it without producing any compressed output. The compressed data written to `w` can only be decompressed by a `Reader` initialized with the same dictionary.

## func (\*Writer) `Close`

```
func (w *Writer) Close() error
```

Close flushes and closes the writer.

## func (\*Writer) Flush

```
func (w *Writer) Flush() error
```

Flush flushes any pending data to the underlying writer. It is useful mainly in compressed network protocols, to ensure that a remote reader has enough data to reconstruct a packet. Flush does not return until the data has been written. Calling Flush when there is no pending data still causes the Writer to emit a sync marker of at least 4 bytes. If the underlying writer returns an error, Flush returns that error.

In the terminology of the zlib library, Flush is equivalent to Z\_SYNC\_FLUSH.

## func (\*Writer) Reset

added in go1.2

```
func (w *Writer) Reset(dst io.Writer)
```

Reset discards the writer's state and makes it equivalent to the result of NewWriter or NewWriterDict called with dst and w's level and dictionary.

## func (\*Writer) Write

```
func (w *Writer) Write(data []byte) (n int, err error)
```

Write writes data to w, which will eventually write the compressed form of data to its underlying writer.



## Source Files

[View all](#)

[deflate.go](#)  
[deflatefast.go](#)  
[dict\\_decoder.go](#)

[huffman\\_bit\\_writer.go](#)  
[huffman\\_code.go](#)  
[inflate.go](#)

[token.go](#)

### Why Go

[Use Cases](#)

[Case Studies](#)

### Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

### Packages

[Standard Library](#)

[About Go Packages](#)

### About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

## Connect

[Twitter](#)

[GitHub](#)

[Slack](#)

[r/golang](#)

[Meetup](#)

[Golang Weekly](#)

---

[Copyright](#)

[Terms of Service](#)

[Privacy Policy](#)

[Report an Issue](#)



Google