zip

package standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 17 | Imported by: 11,480

Details Valid go.mod file Redistributable license Tagged version Stable version

Learn more

Repository cs.opensource.google/go/go

Links Report a Vulnerability

Documentation

Documentation

Overview

Package zip provides support for reading and writing ZIP archives.

See the ZIP specification for details.

This package does not support disk spanning.

A note about ZIP64:

To be backwards compatible the FileHeader has both 32 and 64 bit Size fields. The 64 bit fields will always contain the correct value and for normal archives both fields will be the same. For files requiring the ZIP64 format the 32 bit fields will be 0xffffffff and the 64 bit fields must be used instead.

Index

Examples

Constants

View Source

```
const (
    Store   uint16 = 0 // no compression
    Deflate uint16 = 8 // DEFLATE compressed
)
```

Compression methods.

Variables

View Source

```
var (
    ErrFormat       = errors.New("zip: not a valid zip file")
    ErrAlgorithm    = errors.New("zip: unsupported compression algorithm")
```

```
    ErrChecksum     = errors.New("zip: checksum error")
    ErrInsecurePath = errors.New("zip: insecure file path")
)
```

Functions

func RegisterCompressor

added in go1.2

```
func RegisterCompressor(method uint16, comp Compressor)
```

RegisterCompressor registers custom compressors for a specified method ID. The common methods Store and Deflate are built in.

func RegisterDecompressor

added in go1.2

```
func RegisterDecompressor(method uint16, dcomp Decompressor)
```

RegisterDecompressor allows custom decompressors for a specified method ID. The common methods Store and Deflate are built in.

Types

type Compressor

added in go1.2

```
type Compressor func(w io.Writer) (io.WriteCloser, error)
```

A Compressor returns a new compressing writer, writing to w. The WriteCloser's Close method must be used to flush pending data to w. The Compressor itself must be safe to invoke from multiple goroutines simultaneously, but each returned writer will be used only by one goroutine at a time.

type Decompressor

added in go1.2

```
type Decompressor func(r io.Reader) io.ReadCloser
```

A Decompressor returns a new decompressing reader, reading from r. The ReadCloser's Close method must be used to release associated resources. The Decompressor itself must be safe to invoke from multiple goroutines simultaneously, but each returned reader will be used only by one goroutine at a time.

type File

```
type File struct {
    FileHeader
    // contains filtered or unexported fields
}
```

A File is a single file in a ZIP archive. The file information is in the embedded FileHeader. The file content can be accessed by calling Open.

func (*File) DataOffset

added in go1.2

```
func (f *File) DataOffset() (offset int64, err error)
```

DataOffset returns the offset of the file's possibly-compressed data, relative to the beginning of the zip file.

Most callers should instead use Open, which transparently decompresses data and verifies checksums.

func (*File) Open

```
func (f *File) Open() (io.ReadCloser, error)
```

Open returns a ReadCloser that provides access to the File's contents. Multiple files may be read concurrently.

func (*File) OpenRaw

added in go1.17

```
func (f *File) OpenRaw() (io.Reader, error)
```

OpenRaw returns a Reader that provides access to the File's contents without decompression.

type FileHeader

```
type FileHeader struct {
    // Name is the name of the file.
    //
    // It must be a relative path, not start with a drive letter (such as "C:"),
    // and must use forward slashes instead of back slashes. A trailing slash
    // indicates that this file is a directory and should have no data.
    Name string

    // Comment is any arbitrary user-defined string shorter than 64KiB.
    Comment string

    // NonUTF8 indicates that Name and Comment are not encoded in UTF-8.
    //
    // By specification, the only other encoding permitted should be CP-437,
    // but historically many ZIP readers interpret Name and Comment as whatever
    // the system's local character encoding happens to be.
    //
    // This flag should only be set if the user intends to encode a non-portable
    // ZIP file for a specific localized region. Otherwise, the Writer
    // automatically sets the ZIP format's UTF-8 flag for valid UTF-8 strings.
    NonUTF8 bool

    CreatorVersion uint16
    ReaderVersion  uint16
    Flags          uint16
```

```
    // Method is the compression method. If zero, Store is used.
    Method uint16

    // Modified is the modified time of the file.
    //
    // When reading, an extended timestamp is preferred over the legacy MS-DOS
    // date field, and the offset between the times is used as the timezone.
    // If only the MS-DOS date is present, the timezone is assumed to be UTC.
    //
    // When writing, an extended timestamp (which is timezone-agnostic) is
    // always emitted. The legacy MS-DOS date field is encoded according to the
    // location of the Modified time.
    Modified time.Time

    // ModifiedTime is an MS-DOS-encoded time.
    //
    // Deprecated: Use Modified instead.
    ModifiedTime uint16

    // ModifiedDate is an MS-DOS-encoded date.
    //
    // Deprecated: Use Modified instead.
    ModifiedDate uint16

    // CRC32 is the CRC32 checksum of the file content.
    CRC32 uint32

    // CompressedSize is the compressed size of the file in bytes.
    // If either the uncompressed or compressed size of the file
    // does not fit in 32 bits, CompressedSize is set to ^uint32(0).
    //
    // Deprecated: Use CompressedSize64 instead.
    CompressedSize uint32

    // UncompressedSize is the compressed size of the file in bytes.
    // If either the uncompressed or compressed size of the file
    // does not fit in 32 bits, CompressedSize is set to ^uint32(0).
    //
    // Deprecated: Use UncompressedSize64 instead.
    UncompressedSize uint32

    // CompressedSize64 is the compressed size of the file in bytes.
    CompressedSize64 uint64

    // UncompressedSize64 is the uncompressed size of the file in bytes.
    UncompressedSize64 uint64

    Extra         []byte
    ExternalAttrs uint32 // Meaning depends on CreatorVersion
}
```

FileHeader describes a file within a ZIP file. See the ZIP specification for details.

func FileInfoHeader

```
func FileInfoHeader(fi fs.FileInfo) (*FileHeader, error)
```

FileInfoHeader creates a partially-populated FileHeader from an fs.FileInfo. Because fs.FileInfo's Name method returns only the base name of the file it describes, it may be necessary to modify the Name field of the returned header to provide the full path name of the file. If compression is desired, callers should set the FileHeader.Method field; it is unset by default.

func (*FileHeader) FileInfo

```
func (h *FileHeader) FileInfo() fs.FileInfo
```

FileInfo returns an fs.FileInfo for the FileHeader.

func (*FileHeader) ModTime DEPRECATED Show

func (*FileHeader) Mode

```
func (h *FileHeader) Mode() (mode fs.FileMode)
```

Mode returns the permission and mode bits for the FileHeader.

func (*FileHeader) SetModTime DEPRECATED Show

func (*FileHeader) SetMode

```
func (h *FileHeader) SetMode(mode fs.FileMode)
```

SetMode changes the permission and mode bits for the FileHeader.

type ReadCloser

```
type ReadCloser struct {
    Reader
    // contains filtered or unexported fields
}
```

A ReadCloser is a Reader that must be closed when no longer needed.

func OpenReader

```
func OpenReader(name string) (*ReadCloser, error)
```

OpenReader will open the Zip file specified by name and return a ReadCloser.

func (*ReadCloser) Close

```
func (rc *ReadCloser) Close() error
```

Close closes the Zip file, rendering it unusable for I/O.

type Reader

```
type Reader struct {
    File    []*File
    Comment string
    // contains filtered or unexported fields
}
```

A Reader serves content from a ZIP archive.

▶ Example

func NewReader

```
func NewReader(r io.ReaderAt, size int64) (*Reader, error)
```

NewReader returns a new Reader reading from r, which is assumed to have the given size in bytes.

If any file inside the archive uses a non-local name (as defined by filepath.IsLocal) or a name containing backslashes and the GODEBUG environment variable contains `zipinsecurepath=0`, NewReader returns the reader with an ErrInsecurePath error. A future version of Go may introduce this behavior by default. Programs that want to accept non-local names can ignore the ErrInsecurePath error and use the returned reader.

func (*Reader) Open

added in go1.16

```
func (r *Reader) Open(name string) (fs.File, error)
```

Open opens the named file in the ZIP archive, using the semantics of fs.FS.Open: paths are always slash separated, with no leading / or ../ elements.

func (*Reader) RegisterDecompressor

added in go1.6

```
func (z *Reader) RegisterDecompressor(method uint16, dcomp Decompressor)
```

RegisterDecompressor registers or overrides a custom decompressor for a specific method ID. If a decompressor for a given method is not found, Reader will default to looking up the decompressor at the package level.

type Writer

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

```
}
```

Writer implements a zip file writer.

▶ Example

func NewWriter

```
func NewWriter(w io.Writer) *Writer
```

NewWriter returns a new Writer writing a zip file to w.

func (*Writer) Close

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

Close finishes writing the zip file by writing the central directory. It does not close the underlying writer.

func (*Writer) Copy

added in go1.17

```
func (w *Writer) Copy(f *File) error
```

Copy copies the file f (obtained from a Reader) into w. It copies the raw form directly bypassing decompression, compression, and validation.

func (*Writer) Create

```
func (w *Writer) Create(name string) (io.Writer, error)
```

Create adds a file to the zip file using the provided name. It returns a Writer to which the file contents should be written. The file contents will be compressed using the Deflate method. The name must be a relative path: it must not start with a drive letter (e.g. C:) or leading slash, and only forward slashes are allowed. To create a directory instead of a file, add a trailing slash to the name. The file's contents must be written to the io.Writer before the next call to Create, CreateHeader, or Close.

func (*Writer) CreateHeader

```
func (w *Writer) CreateHeader(fh *FileHeader) (io.Writer, error)
```

CreateHeader adds a file to the zip archive using the provided FileHeader for the file metadata. Writer takes ownership of fh and may mutate its fields. The caller must not modify fh after calling CreateHeader.

This returns a Writer to which the file contents should be written. The file's contents must be written to the io.Writer before the next call to Create, CreateHeader, CreateRaw, or Close.

func (*Writer) CreateRaw

added in go1.17

```
func (w *Writer) CreateRaw(fh *FileHeader) (io.Writer, error)
```

CreateRaw adds a file to the zip archive using the provided FileHeader and returns a Writer to which the file contents should be written. The file's contents must be written to the io.Writer before the next call to Create, CreateHeader, CreateRaw, or Close.

In contrast to CreateHeader, the bytes passed to Writer are not compressed.

func (*Writer) Flush

added in go1.4

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

Flush flushes any buffered data to the underlying writer. Calling Flush is not normally necessary; calling Close is sufficient.

func (*Writer) RegisterCompressor

added in go1.6

```
func (w *Writer) RegisterCompressor(method uint16, comp Compressor)
```

RegisterCompressor registers or overrides a custom compressor for a specific method ID. If a compressor for a given method is not found, Writer will default to looking up the compressor at the package level.

▶ Example

func (*Writer) SetComment

added in go1.10

```
func (w *Writer) SetComment(comment string) error
```

SetComment sets the end-of-central-directory comment field. It can only be called before Close.

func (*Writer) SetOffset

added in go1.5

```
func (w *Writer) SetOffset(n int64)
```

SetOffset sets the offset of the beginning of the zip data within the underlying writer. It should be used when the zip data is appended to an existing file, such as a binary executable. It must be called before any data is written.

Source Files

View all

reader.go
register.go
struct.go
writer.go