

[Discover Packages](#) > [Standard library](#) > [archive](#) > [tar](#) 


tar

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [18](#) |Imported by: [16,164](#)

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

Rendered for [linux/amd64](#) 

Overview

Package tar implements access to tar archives.

Tape archives (tar) are a file format for storing a sequence of files that can be read and written in a streaming manner. This package aims to cover most variations of the format, including those produced by GNU and BSD tar tools.

► [Example \(Minimal\)](#)

Index

[Constants](#)[Variables](#)[type Format](#)[func \(f Format\) String\(\) string](#)[type Header](#)[func FileInfoHeader\(fi fs.FileInfo, link string\) \(*Header, error\)](#)[func \(h *Header\) FileInfo\(\) fs.FileInfo](#)[type Reader](#)[func NewReader\(r io.Reader\) *Reader](#)[func \(tr *Reader\) Next\(\) \(*Header, error\)](#)[func \(tr *Reader\) Read\(b \[\]byte\) \(int, error\)](#)[type Writer](#)[func NewWriter\(w io.Writer\) *Writer](#)

```
func (tw *Writer) Close() error
func (tw *Writer) Flush() error
func (tw *Writer) Write(b []byte) (int, error)
func (tw *Writer) WriteHeader(hdr *Header) error
```

Examples

[Package \(Minimal\)](#)

Constants

[View Source](#)

```
const (
    // Type '0' indicates a regular file.
    TypeReg = '0'

    // Deprecated: Use TypeReg instead.
    TypeRegA = '\x00'

    // Type '1' to '6' are header-only flags and may not have a data body.
    TypeLink      = '1' // Hard link
    TypeSymlink   = '2' // Symbolic link
    TypeChar      = '3' // Character device node
    TypeBlock     = '4' // Block device node
    TypeDir       = '5' // Directory
    TypeFifo      = '6' // FIFO node

    // Type '7' is reserved.
    TypeCont = '7'

    // Type 'x' is used by the PAX format to store key-value records that
    // are only relevant to the next file.
    // This package transparently handles these types.
    TypeXHeader = 'x'

    // Type 'g' is used by the PAX format to store key-value records that
    // are relevant to all subsequent files.
    // This package only supports parsing and composing such headers,
    // but does not currently support persisting the global state across files.
    TypeXGlobalHeader = 'g'

    // Type 'S' indicates a sparse file in the GNU format.
    TypeGNUSparse = 'S'

    // Types 'L' and 'K' are used by the GNU format for a meta file
    // used to store the path or link name for the next file.
    // This package transparently handles these types.
    TypeGNULongName = 'L'
    TypeGNULongLink = 'K'
)
```

Type flags for Header.Typeflag.

Variables

[View Source](#)

```
var (  
    ErrHeader          = errors.New("archive/tar: invalid tar header")  
    ErrWriteTooLong    = errors.New("archive/tar: write too long")  
    ErrFieldTooLong    = errors.New("archive/tar: header field too long")  
    ErrWriteAfterClose = errors.New("archive/tar: write after close")  
    ErrInsecurePath    = errors.New("archive/tar: insecure file path")  
)
```

Functions

This section is empty.

Types

type **Format** added in go1.10

```
type Format int
```

Format represents the tar archive format.

The original tar format was introduced in Unix V7. Since then, there have been multiple competing formats attempting to standardize or extend the V7 format to overcome its limitations. The most common formats are the USTAR, PAX, and GNU formats, each with their own advantages and limitations.

The following table captures the capabilities of each format:

	USTAR	PAX	GNU
Name	256B	unlimited	unlimited
Linkname	100B	unlimited	unlimited
Size	uint33	unlimited	uint89
Mode	uint21	uint21	uint57
Uid/Gid	uint21	unlimited	uint57
Uname/Gname	32B	unlimited	32B
ModTime	uint33	unlimited	int89
AccessTime	n/a	unlimited	int89
ChangeTime	n/a	unlimited	int89
Devmajor/Devminor	uint21	uint21	uint57
string encoding	ASCII	UTF-8	binary
sub-second times	no	yes	no
sparse files	no	yes	yes

The table's upper portion shows the Header fields, where each format reports the maximum number of bytes allowed for each string field and the integer type used to store each numeric field (where timestamps are stored as the number of seconds since the Unix epoch).

The table's lower portion shows specialized features of each format, such as supported string encodings, support for sub-second timestamps, or support for sparse files.

The Writer currently provides no support for sparse files.

```
const (  
  
    // FormatUnknown indicates that the format is unknown.  
    FormatUnknown Format  
  
    // FormatUSTAR represents the USTAR header format defined in POSIX.1-1988.  
    //  
    // While this format is compatible with most tar readers,  
    // the format has several limitations making it unsuitable for some usages.  
    // Most notably, it cannot support sparse files, files larger than 8GiB,  
    // filenames larger than 256 characters, and non-ASCII filenames.  
    //  
    // Reference:  
    // http://pubs.opengroup.org/onlinepubs/9699919799/utilities/pax.html#tag\_20\_92\_13  
    FormatUSTAR  
  
    // FormatPAX represents the PAX header format defined in POSIX.1-2001.  
    //  
    // PAX extends USTAR by writing a special file with Typeflag TypeXHeader  
    // preceding the original header. This file contains a set of key-value  
    // records, which are used to overcome USTAR's shortcomings, in addition to  
    // providing the ability to have sub-second resolution for timestamps.  
    //  
    // Some newer formats add their own extensions to PAX by defining their  
    // own keys and assigning certain semantic meaning to the associated values.  
    // For example, sparse file support in PAX is implemented using keys  
    // defined by the GNU manual (e.g., "GNU.sparse.map").  
    //  
    // Reference:  
    // http://pubs.opengroup.org/onlinepubs/009695399/utilities/pax.html  
    FormatPAX  
  
    // FormatGNU represents the GNU header format.  
    //  
    // The GNU header format is older than the USTAR and PAX standards and  
    // is not compatible with them. The GNU format supports  
    // arbitrary file sizes, filenames of arbitrary encoding and length,  
    // sparse files, and other features.  
    //  
    // It is recommended that PAX be chosen over GNU unless the target  
    // application can only parse GNU formatted archives.  
    //  
    // Reference:  
    // https://www.gnu.org/software/tar/manual/html\_node/Standard.html  
    FormatGNU  
  
)
```

Constants to identify various tar formats.

func (Format) String

added in go1.10

```
func (f Format) String() string
```

type Header

```
type Header struct {
    // Typeflag is the type of header entry.
    // The zero value is automatically promoted to either TypeReg or TypeDir
    // depending on the presence of a trailing slash in Name.
    Typeflag byte

    Name      string // Name of file entry
    Linkname  string // Target name of link (valid for TypeLink or TypeSymlink)

    Size  int64 // Logical file size in bytes
    Mode  int64 // Permission and mode bits
    Uid   int   // User ID of owner
    Gid   int   // Group ID of owner
    Uname string // User name of owner
    Gname string // Group name of owner

    // If the Format is unspecified, then Writer.WriteHeader rounds ModTime
    // to the nearest second and ignores the AccessTime and ChangeTime fields.
    //
    // To use AccessTime or ChangeTime, specify the Format as PAX or GNU.
    // To use sub-second resolution, specify the Format as PAX.
    ModTime    time.Time // Modification time
    AccessTime time.Time // Access time (requires either PAX or GNU support)
    ChangeTime time.Time // Change time (requires either PAX or GNU support)

    Devmajor int64 // Major device number (valid for TypeChar or TypeBlock)
    Devminor int64 // Minor device number (valid for TypeChar or TypeBlock)

    // Xattrs stores extended attributes as PAX records under the
    // "SCHILY.xattr." namespace.
    //
    // The following are semantically equivalent:
    // h.Xattrs[key] = value
    // h.PAXRecords["SCHILY.xattr."+key] = value
    //
    // When Writer.WriteHeader is called, the contents of Xattrs will take
    // precedence over those in PAXRecords.
    //
    // Deprecated: Use PAXRecords instead.
    Xattrs map[string]string

    // PAXRecords is a map of PAX extended header records.
    //
    // User-defined records should have keys of the following form:
```

```

//  VENDOR.keyword
// Where VENDOR is some namespace in all uppercase, and keyword may
// not contain the '=' character (e.g., "GOLANG.pkg.version").
// The key and value should be non-empty UTF-8 strings.
//
// When Writer.WriteHeader is called, PAX records derived from the
// other fields in Header take precedence over PAXRecords.
PAXRecords map[string]string

// Format specifies the format of the tar header.
//
// This is set by Reader.Next as a best-effort guess at the format.
// Since the Reader liberally reads some non-compliant files,
// it is possible for this to be FormatUnknown.
//
// If the format is unspecified when Writer.WriteHeader is called,
// then it uses the first format (in the order of USTAR, PAX, GNU)
// capable of encoding this Header (see Format).
Format Format
}

```

A Header represents a single header in a tar archive. Some fields may not be populated.

For forward compatibility, users that retrieve a Header from Reader.Next, mutate it in some ways, and then pass it back to Writer.WriteHeader should do so by creating a new Header and copying the fields that they are interested in preserving.

func FileInfoHeader

added in go1.1

```
func FileInfoHeader(fi fs.FileInfo, link string) (*Header, error)
```

FileInfoHeader creates a partially-populated Header from fi. If fi describes a symlink, FileInfoHeader records link as the link target. If fi describes a directory, a slash is appended to the name.

Since fs.FileInfo's Name method only returns the base name of the file it describes, it may be necessary to modify Header.Name to provide the full path name of the file.

func (*Header) FileInfo

added in go1.1

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

FileInfo returns an fs.FileInfo for the Header.

type Reader

```

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

```

Reader provides sequential access to the contents of a tar archive. Reader.Next advances to the next file in the archive (including the first), and then Reader can be treated as an io.Reader to access the file's data.

func NewReader

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

NewReader creates a new Reader reading from r.

func (*Reader) Next

```
func (tr *Reader) Next() (*Header, error)
```

Next advances to the next entry in the tar archive. The Header.Size determines how many bytes can be read for the next file. Any remaining data in the current file is automatically discarded. At the end of the archive, Next returns the error io.EOF.

If Next encounters a non-local name (as defined by [filepath.IsLocal](#)) and the GODEBUG environment variable contains ``tarinsecurepath=0``, Next returns the header 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 header.

func (*Reader) Read

```
func (tr *Reader) Read(b []byte) (int, error)
```

Read reads from the current file in the tar archive. It returns (0, io.EOF) when it reaches the end of that file, until Next is called to advance to the next file.

If the current file is sparse, then the regions marked as a hole are read back as NUL-bytes.

Calling Read on special types like TypeLink, TypeSymlink, TypeChar, TypeBlock, TypeDir, and TypeFifo returns (0, io.EOF) regardless of what the Header.Size claims.

type Writer

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

Writer provides sequential writing of a tar archive. Write.WriteHeader begins a new file with the provided Header, and then Writer can be treated as an io.Writer to supply that file's data.

func NewWriter

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

NewWriter creates a new Writer writing to w.

func (*Writer) Close

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

Close closes the tar archive by flushing the padding, and writing the footer. If the current file (from a prior call to WriteHeader) is not fully written, then this returns an error.

func (*Writer) Flush

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

Flush finishes writing the current file's block padding. The current file must be fully written before Flush can be called.

This is unnecessary as the next call to WriteHeader or Close will implicitly flush out the file's padding.

func (*Writer) Write

```
func (tw *Writer) Write(b []byte) (int, error)
```

Write writes to the current file in the tar archive. Write returns the error ErrWriteTooLong if more than Header.Size bytes are written after WriteHeader.

Calling Write on special types like TypeLink, TypeSymlink, TypeChar, TypeBlock, TypeDir, and TypeFifo returns (0, ErrWriteTooLong) regardless of what the Header.Size claims.

func (*Writer) WriteHeader

```
func (tw *Writer) WriteHeader(hdr *Header) error
```

WriteHeader writes hdr and prepares to accept the file's contents. The Header.Size determines how many bytes can be written for the next file. If the current file is not fully written, then this returns an error. This implicitly flushes any padding necessary before writing the header.

Source Files

[View all](#) 

[common.go](#)
[format.go](#)
[reader.go](#)

[stat_actime1.go](#)
[stat_unix.go](#)
[strconv.go](#)

[writer.go](#)

Why Go

Use Cases

Case Studies

Get Started

Playground

Tour

Packages

Standard Library

About Go Packages

About

Download

Blog

[Stack Overflow](#)

[Issue Tracker](#)

[Help](#)

[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