bytes

Package bytes go1.15.2 Latest

Published: **Sep 9, 2020** | License: BSD-3-Clause | Standard library

Overview

Package bytes implements functions for the manipulation of byte slices. It is analogous to the facilities of the strings package.

Constants

```
const MinRead = 512
```

MinRead is the minimum slice size passed to a Read call by Buffer.ReadFrom. As long as the Buffer has at least MinRead bytes beyond what is required to hold the contents of r, ReadFrom will not grow the underlying buffer.

Variables

```
var ErrTooLarge = errors.New("bytes.Buffer: too large")
```

ErrTooLarge is passed to panic if memory cannot be allocated to store data in a buffer.

func Compare

```
func Compare(a, b []byte) int
```

Compare returns an integer comparing two byte slices lexicographically. The result will be 0 if a==b, -1 if a < b, and +1 if a > b. A nil argument is equivalent to an empty slice.

func Contains

```
func Contains(b, subslice []byte) bool
```

Contains reports whether subslice is within b.

func ContainsAny

```
func ContainsAny(b []byte, chars string) bool
```

ContainsAny reports whether any of the UTF-8-encoded code points in chars are within b.

func ContainsRune

```
func ContainsRune(b []byte, r rune) bool
```

ContainsRune reports whether the rune is contained in the UTF-8-encoded byte slice b.

func Count

```
func Count(s, sep []byte) int
```

Count counts the number of non-overlapping instances of sep in s. If sep is an empty slice, Count returns 1 + the number of UTF-8-encoded code points in s.

func Equal

```
func Equal(a, b []byte) bool
```

Equal reports whether a and b are the same length and contain the same bytes. A nil argument is equivalent to an empty slice.

func EqualFold

```
func EqualFold(s, t []byte) bool
```

EqualFold reports whether s and t, interpreted as UTF-8 strings, are equal under Unicode case-folding, which is a more general form of case-insensitivity.

func Fields

```
func Fields(s []byte) [][]byte
```

Fields interprets s as a sequence of UTF-8-encoded code points. It splits the slice s around each instance of one or more consecutive white space characters, as defined by unicode.IsSpace, returning a slice of subslices of s or an empty slice if s contains only white space.

func FieldsFunc

```
func FieldsFunc(s []byte, f func(rune) bool) [][]byte
```

FieldsFunc interprets s as a sequence of UTF-8-encoded code points. It splits the slice s at each run of code points c satisfying f(c) and returns a slice of subslices of s. If all code points in s satisfy f(c), or len(s) == 0, an empty slice is returned.

FieldsFunc makes no guarantees about the order in which it calls f(c) and assumes that f always returns the same value for a given c.

func HasPrefix

```
func HasPrefix(s, prefix []byte) bool
```

HasPrefix tests whether the byte slice s begins with prefix.

func HasSuffix

```
func HasSuffix(s, suffix []byte) bool
```

HasSuffix tests whether the byte slice s ends with suffix.

func Index

```
func Index(s, sep []byte) int
```

Index returns the index of the first instance of sep in s, or -1 if sep is not present in s.

func IndexAny

```
func IndexAny(s []byte, chars string) int
```

IndexAny interprets s as a sequence of UTF-8-encoded Unicode code points. It returns the byte index of the first occurrence in s of any of the Unicode code points in chars. It returns -1 if chars is empty or if there is no code point in common.

func IndexByte

```
func IndexByte(b []byte, c byte) int
```

IndexByte returns the index of the first instance of c in b, or -1 if c is not present in b.

func IndexFunc

```
func IndexFunc(s []byte, f func(r rune) bool) int
```

IndexFunc interprets s as a sequence of UTF-8-encoded code points. It returns the byte index in s of the first Unicode code point satisfying f(c), or -1 if none do.

func IndexRune

```
func IndexRune(s []byte, r rune) int
```

IndexRune interprets s as a sequence of UTF-8-encoded code points. It returns the byte index of the first occurrence in s of the given rune. It returns -1 if rune is not present in s. If r is utf8.RuneError, it

returns the first instance of any invalid UTF-8 byte sequence.

func Join

```
func Join(s [][]byte, sep []byte) []byte
```

Join concatenates the elements of s to create a new byte slice. The separator sep is placed between elements in the resulting slice.

func LastIndex

```
func LastIndex(s, sep []byte) int
```

LastIndex returns the index of the last instance of sep in s, or -1 if sep is not present in s.

func LastIndexAny

```
func LastIndexAny(s []byte, chars string) int
```

LastIndexAny interprets s as a sequence of UTF-8-encoded Unicode code points. It returns the byte index of the last occurrence in s of any of the Unicode code points in chars. It returns -1 if chars is empty or if there is no code point in common.

func LastIndexByte

```
func LastIndexByte(s []byte, c byte) int
```

LastIndexByte returns the index of the last instance of c in s, or -1 if c is not present in s.

func LastIndexFunc

```
func LastIndexFunc(s []byte, f func(r rune) bool) int
```

LastIndexFunc interprets s as a sequence of UTF-8-encoded code points. It returns the byte index in s of the last Unicode code point satisfying f(c), or -1 if none do.

func Map

```
func Map(mapping func(r rune) rune, s []byte) []byte
```

Map returns a copy of the byte slice s with all its characters modified according to the mapping function. If mapping returns a negative value, the character is dropped from the byte slice with no replacement. The characters in s and the output are interpreted as UTF-8-encoded code points.

func Repeat

```
func Repeat(b []byte, count int) []byte
```

Repeat returns a new byte slice consisting of count copies of b.

It panics if count is negative or if the result of (len(b) * count) overflows.

func Replace

```
func Replace(s, old, new []byte, n int) []byte
```

Replace returns a copy of the slice s with the first n non-overlapping instances of old replaced by new. If old is empty, it matches at the beginning of the slice and after each UTF-8 sequence, yielding up to k+1 replacements for a k-rune slice. If n < 0, there is no limit on the number of replacements.

func ReplaceAll

```
func ReplaceAll(s, old, new []byte) []byte
```

ReplaceAll returns a copy of the slice s with all non-overlapping instances of old replaced by new. If old is empty, it matches at the beginning of the slice and after each UTF-8 sequence, yielding up to k+1 replacements for a k-rune slice.

func Runes

```
func Runes(s []byte) []rune
```

Runes interprets s as a sequence of UTF-8-encoded code points. It returns a slice of runes (Unicode code points) equivalent to s.

func Split

```
func Split(s, sep []byte) [][]byte
```

Split slices s into all subslices separated by sep and returns a slice of the subslices between those separators. If sep is empty, Split splits after each UTF-8 sequence. It is equivalent to SplitN with a count of -1.

func SplitAfter

```
func SplitAfter(s, sep []byte) [][]byte
```

SplitAfter slices s into all subslices after each instance of sep and returns a slice of those subslices. If sep is empty, SplitAfter splits after each UTF-8 sequence. It is equivalent to SplitAfterN with a count of -1.

func SplitAfterN

```
func SplitAfterN(s, sep []byte, n int) [][]byte
```

SplitAfterN slices s into subslices after each instance of sep and returns a slice of those subslices. If sep is empty, SplitAfterN splits after each UTF-8 sequence. The count determines the number of subslices to return:

```
n > 0: at most n subslices; the last subslice will be the unsplit remainder.
n == 0: the result is nil (zero subslices)
n < 0: all subslices
```

func SplitN

```
func SplitN(s, sep []byte, n int) [][]byte
```

SplitN slices s into subslices separated by sep and returns a slice of the subslices between those separators. If sep is empty, SplitN splits after each UTF-8 sequence. The count determines the number of subslices to return:

```
n > 0: at most n subslices; the last subslice will be the unsplit remainder.
n == 0: the result is nil (zero subslices)
n < 0: all subslices
```

func Title

```
func Title(s []byte) []byte
```

Title treats s as UTF-8-encoded bytes and returns a copy with all Unicode letters that begin words mapped to their title case.

BUG(rsc): The rule Title uses for word boundaries does not handle Unicode punctuation properly.

func ToLower

```
func ToLower(s []byte) []byte
```

ToLower returns a copy of the byte slice s with all Unicode letters mapped to their lower case.

func ToLowerSpecial

```
func ToLowerSpecial(c unicode.SpecialCase, s []byte) []byte
```

ToLowerSpecial treats s as UTF-8-encoded bytes and returns a copy with all the Unicode letters mapped to their lower case, giving priority to the special casing rules.

func ToTitle

```
func ToTitle(s []byte) []byte
```

ToTitle treats s as UTF-8-encoded bytes and returns a copy with all the Unicode letters mapped to their title case.

func ToTitleSpecial

```
func ToTitleSpecial(c unicode.SpecialCase, s []byte) []byte
```

ToTitleSpecial treats s as UTF-8-encoded bytes and returns a copy with all the Unicode letters mapped to their title case, giving priority to the special casing rules.

func ToUpper

```
func ToUpper(s []byte) []byte
```

ToUpper returns a copy of the byte slice s with all Unicode letters mapped to their upper case.

func ToUpperSpecial

```
func ToUpperSpecial(c unicode.SpecialCase, s []byte) []byte
```

ToUpperSpecial treats s as UTF-8-encoded bytes and returns a copy with all the Unicode letters mapped to their upper case, giving priority to the special casing rules.

func ToValidUTF8

```
func ToValidUTF8(s, replacement []byte) []byte
```

ToValidUTF8 treats s as UTF-8-encoded bytes and returns a copy with each run of bytes representing invalid UTF-8 replaced with the bytes in replacement, which may be empty.

func Trim

```
func Trim(s []byte, cutset string) []byte
```

Trim returns a subslice of s by slicing off all leading and trailing UTF-8-encoded code points contained in cutset.

func TrimFunc

```
func TrimFunc(s []byte, f func(r rune) bool) []byte
```

TrimFunc returns a subslice of s by slicing off all leading and trailing UTF-8-encoded code points c that satisfy f(c).

func TrimLeft

```
func TrimLeft(s []byte, cutset string) []byte
```

TrimLeft returns a subslice of s by slicing off all leading UTF-8-encoded code points contained in cutset.

func TrimLeftFunc

```
func TrimLeftFunc(s []byte, f func(r rune) bool) []byte
```

TrimLeftFunc treats s as UTF-8-encoded bytes and returns a subslice of s by slicing off all leading UTF-8-encoded code points c that satisfy f(c).

func TrimPrefix

```
func TrimPrefix(s, prefix []byte) []byte
```

TrimPrefix returns s without the provided leading prefix string. If s doesn't start with prefix, s is returned unchanged.

func TrimRight

```
func TrimRight(s []byte, cutset string) []byte
```

TrimRight returns a subslice of s by slicing off all trailing UTF-8-encoded code points that are contained in cutset.

func TrimRightFunc

```
func TrimRightFunc(s []byte, f func(r rune) bool) []byte
```

TrimRightFunc returns a subslice of s by slicing off all trailing UTF-8-encoded code points c that satisfy f(c).

func TrimSpace

```
func TrimSpace(s []byte) []byte
```

TrimSpace returns a subslice of s by slicing off all leading and trailing white space, as defined by Unicode.

func TrimSuffix

```
func TrimSuffix(s, suffix []byte) []byte
```

TrimSuffix returns s without the provided trailing suffix string. If s doesn't end with suffix, s is returned unchanged.

type Buffer

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

A Buffer is a variable-sized buffer of bytes with Read and Write methods. The zero value for Buffer is an empty buffer ready to use.

func NewBuffer

```
func NewBuffer(buf []byte) *Buffer
```

NewBuffer creates and initializes a new Buffer using buf as its initial contents. The new Buffer takes ownership of buf, and the caller should not use buf after this call. NewBuffer is intended to prepare a Buffer to read existing data. It can also be used to set the initial size of the internal buffer for writing. To do that, buf should have the desired capacity but a length of zero.

In most cases, new(Buffer) (or just declaring a Buffer variable) is sufficient to initialize a Buffer.

func NewBufferString

```
func NewBufferString(s string) *Buffer
```

NewBufferString creates and initializes a new Buffer using string s as its initial contents. It is intended to prepare a buffer to read an existing string.

In most cases, new(Buffer) (or just declaring a Buffer variable) is sufficient to initialize a Buffer.

func (*Buffer) Bytes

```
func (b *Buffer) Bytes() []byte
```

Bytes returns a slice of length b.Len() holding the unread portion of the buffer. The slice is valid for use only until the next buffer modification (that is, only until the next call to a method like Read, Write, Reset, or Truncate). The slice aliases the buffer content at least until the next buffer modification, so immediate changes to the slice will affect the result of future reads.

func (*Buffer) Cap

```
func (b *Buffer) Cap() int
```

Cap returns the capacity of the buffer's underlying byte slice, that is, the total space allocated for the buffer's data.

func (*Buffer) Grow

```
func (b *Buffer) Grow(n int)
```

Grow grows the buffer's capacity, if necessary, to guarantee space for another n bytes. After Grow(n), at least n bytes can be written to the buffer without another allocation. If n is negative, Grow will panic. If the buffer can't grow it will panic with ErrTooLarge.

func (*Buffer) Len

```
func (b *Buffer) Len() int
```

Len returns the number of bytes of the unread portion of the buffer; b.Len() == len(b.Bytes()).

func (*Buffer) Next

```
func (b *Buffer) Next(n int) []byte
```

Next returns a slice containing the next n bytes from the buffer, advancing the buffer as if the bytes had been returned by Read. If there are fewer than n bytes in the buffer, Next returns the entire buffer. The slice is only valid until the next call to a read or write method.

func (*Buffer) Read

```
func (b *Buffer) Read(p []byte) (n int, err error)
```

Read reads the next len(p) bytes from the buffer or until the buffer is drained. The return value n is the number of bytes read. If the buffer has no data to return, err is io.EOF (unless len(p) is zero); otherwise it is nil.

func (*Buffer) ReadByte

```
func (b *Buffer) ReadByte() (byte, error)
```

ReadByte reads and returns the next byte from the buffer. If no byte is available, it returns error io.EOF.

func (*Buffer) ReadBytes

```
func (b *Buffer) ReadBytes(delim byte) (line []byte, err error)
```

ReadBytes reads until the first occurrence of delim in the input, returning a slice containing the data up to and including the delimiter. If ReadBytes encounters an error before finding a delimiter, it returns the data read before the error and the error itself (often io.EOF). ReadBytes returns err != nil if and only if the returned data does not end in delim.

func (*Buffer) ReadFrom

```
func (b *Buffer) ReadFrom(r io.Reader) (n int64, err error)
```

ReadFrom reads data from r until EOF and appends it to the buffer, growing the buffer as needed. The return value n is the number of bytes read. Any error except io.EOF encountered during the read is also returned. If the buffer becomes too large, ReadFrom will panic with ErrTooLarge.

func (*Buffer) ReadRune

```
func (b *Buffer) ReadRune() (r rune, size int, err error)
```

ReadRune reads and returns the next UTF-8-encoded Unicode code point from the buffer. If no bytes are available, the error returned is io.EOF. If the bytes are an erroneous UTF-8 encoding, it consumes one byte and returns U+FFFD, 1.

func (*Buffer) ReadString

```
func (b *Buffer) ReadString(delim byte) (line string, err error)
```

ReadString reads until the first occurrence of delim in the input, returning a string containing the data up to and including the delimiter. If ReadString encounters an error before finding a delimiter, it returns the data read before the error and the error itself (often io.EOF). ReadString returns err != nil if and only if the returned data does not end in delim.

func (*Buffer) Reset

```
func (b *Buffer) Reset()
```

Reset resets the buffer to be empty, but it retains the underlying storage for use by future writes. Reset is the same as Truncate(0).

func (*Buffer) String

```
func (b *Buffer) String() string
```

String returns the contents of the unread portion of the buffer as a string. If the Buffer is a nil pointer, it returns "<nil>".

To build strings more efficiently, see the strings.Builder type.

func (*Buffer) Truncate

```
func (b *Buffer) Truncate(n int)
```

Truncate discards all but the first n unread bytes from the buffer but continues to use the same allocated storage. It panics if n is negative or greater than the length of the buffer.

func (*Buffer) UnreadByte

```
func (b *Buffer) UnreadByte() error
```

UnreadByte unreads the last byte returned by the most recent successful read operation that read at least one byte. If a write has happened since the last read, if the last read returned an error, or if the read read zero bytes, UnreadByte returns an error.

func (*Buffer) UnreadRune

```
func (b *Buffer) UnreadRune() error
```

UnreadRune unreads the last rune returned by ReadRune. If the most recent read or write operation on the buffer was not a successful ReadRune, UnreadRune returns an error. (In this regard it is stricter than UnreadByte, which will unread the last byte from any read operation.)

func (*Buffer) Write

```
func (b *Buffer) Write(p []byte) (n int, err error)
```

Write appends the contents of p to the buffer, growing the buffer as needed. The return value n is the length of p; err is always nil. If the buffer becomes too large, Write will panic with ErrTooLarge.

func (*Buffer) WriteByte

```
func (b *Buffer) WriteByte(c byte) error
```

WriteByte appends the byte c to the buffer, growing the buffer as needed. The returned error is always nil, but is included to match bufio.Writer's WriteByte. If the buffer becomes too large, WriteByte will panic with ErrTooLarge.

func (*Buffer) WriteRune

```
func (b *Buffer) WriteRune(r rune) (n int, err error)
```

WriteRune appends the UTF-8 encoding of Unicode code point r to the buffer, returning its length and an error, which is always nil but is included to match bufio.Writer's WriteRune. The buffer is grown as needed; if it becomes too large, WriteRune will panic with ErrTooLarge.

func (*Buffer) WriteString

```
func (b *Buffer) WriteString(s string) (n int, err error)
```

WriteString appends the contents of s to the buffer, growing the buffer as needed. The return value n is the length of s; err is always nil. If the buffer becomes too large, WriteString will panic with ErrTooLarge.

func (*Buffer) WriteTo

```
func (b *Buffer) WriteTo(w io.Writer) (n int64, err error)
```

WriteTo writes data to w until the buffer is drained or an error occurs. The return value n is the number of bytes written; it always fits into an int, but it is int64 to match the io.WriterTo interface. Any error encountered during the write is also returned.

type Reader

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

A Reader implements the io.Reader, io.ReaderAt, io.WriterTo, io.Seeker, io.ByteScanner, and io.RuneScanner interfaces by reading from a byte slice. Unlike a Buffer, a Reader is read-only and supports seeking. The zero value for Reader operates like a Reader of an empty slice.

func NewReader

```
func NewReader(b []byte) *Reader
```

NewReader returns a new Reader reading from b.

func (*Reader) Len

```
func (r *Reader) Len() int
```

Len returns the number of bytes of the unread portion of the slice.

func (*Reader) Read

```
func (r *Reader) Read(b []byte) (n int, err error)
```

Read implements the io.Reader interface.

func (*Reader) ReadAt

```
func (r *Reader) ReadAt(b []byte, off int64) (n int, err error)
```

ReadAt implements the io.ReaderAt interface.

func (*Reader) ReadByte

```
func (r *Reader) ReadByte() (byte, error)
```

ReadByte implements the io.ByteReader interface.

func (*Reader) ReadRune

```
func (r *Reader) ReadRune() (ch rune, size int, err error)
```

ReadRune implements the io.RuneReader interface.

func (*Reader) Reset

```
func (r *Reader) Reset(b []byte)
```

Reset resets the Reader to be reading from b.

func (*Reader) Seek

```
func (r *Reader) Seek(offset int64, whence int) (int64, error)
```

Seek implements the io.Seeker interface.

func (*Reader) Size

```
func (r *Reader) Size() int64
```

Size returns the original length of the underlying byte slice. Size is the number of bytes available for reading via ReadAt. The returned value is always the same and is not affected by calls to any other method.

func (*Reader) UnreadByte

```
func (r *Reader) UnreadByte() error
```

UnreadByte complements ReadByte in implementing the io.ByteScanner interface.

func (*Reader) UnreadRune

```
func (r *Reader) UnreadRune() error
```

UnreadRune complements ReadRune in implementing the io.RuneScanner interface.

func (*Reader) WriteTo

```
func (r *Reader) WriteTo(w io.Writer) (n int64, err error)
```

WriteTo implements the io.WriterTo interface.

BUGs

- The rule Title uses for word boundaries does not handle Unicode punctuation properly.