binary

package standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 5 | Imported by: 139,534

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 binary implements simple translation between numbers and byte sequences and encoding and decoding of varints.

Numbers are translated by reading and writing fixed-size values. A fixed-size value is either a fixed-size arithmetic type (bool, int8, uint8, int16, float32, complex64, ...) or an array or struct containing only fixed-size values.

The varint functions encode and decode single integer values using a variable-length encoding; smaller values require fewer bytes. For a specification, see https://developers.google.com/protocol-buffers/docs/encoding.

This package favors simplicity over efficiency. Clients that require high-performance serialization, especially for large data structures, should look at more advanced solutions such as the encoding/gob package or protocol buffers.

Index

Constants
Variables
func AppendUvarint(buf []byte, x uint64) []byte
func AppendVarint(buf []byte, x int64) []byte
func PutUvarint(buf []byte, x uint64) int
func PutVarint(buf []byte, x int64) int
func Read(r io.Reader, order ByteOrder, data any) error

func ReadUvarint(r io.ByteReader) (uint64, error)
func ReadVarint(r io.ByteReader) (int64, error)
func Size(v any) int
func Uvarint(buf []byte) (uint64, int)
func Varint(buf []byte) (int64, int)
func Write(w io.Writer, order ByteOrder, data any) error
type AppendByteOrder
type ByteOrder

Examples

ByteOrder (Get)
ByteOrder (Put)
PutUvarint
PutVarint
Read
Read (Multi)
Uvarint
Varint
Write
Write (Multi)

Constants

View Source

```
const (
    MaxVarintLen16 = 3
    MaxVarintLen32 = 5
    MaxVarintLen64 = 10
)
```

MaxVarintLenN is the maximum length of a varint-encoded N-bit integer.

Variables

View Source

```
var BigEndian bigEndian
```

BigEndian is the big-endian implementation of ByteOrder and AppendByteOrder.

View Source

```
var LittleEndian littleEndian
```

LittleEndian is the little-endian implementation of ByteOrder and AppendByteOrder.

Functions

func AppendUvarint

added in go1.19

```
func AppendUvarint(buf []byte, x uint64) []byte
```

AppendUvarint appends the varint-encoded form of x, as generated by PutUvarint, to buf and returns the extended buffer.

func AppendVarint

added in go1.19

```
func AppendVarint(buf []byte, x int64) []byte
```

AppendVarint appends the varint-encoded form of x, as generated by PutVarint, to buf and returns the extended buffer.

func PutUvarint

```
func PutUvarint(buf []byte, x uint64) int
```

PutUvarint encodes a uint64 into buf and returns the number of bytes written. If the buffer is too small, PutUvarint will panic.

▶ Example

func PutVarint

```
func PutVarint(buf []byte, x int64) int
```

PutVarint encodes an int64 into buf and returns the number of bytes written. If the buffer is too small, PutVarint will panic.

▶ Example

func Read

```
func Read(r io.Reader, order ByteOrder, data any) error
```

Read reads structured binary data from r into data. Data must be a pointer to a fixed-size value or a slice of fixed-size values. Bytes read from r are decoded using the specified byte order and written to successive fields of the data. When decoding boolean values, a zero byte is decoded as false, and any other non-zero byte is decoded as true. When reading into structs, the field data for fields with blank (_) field names is skipped; i.e., blank field names may be used for padding. When reading into a struct, all non-blank fields must be exported or Read may panic.

The error is EOF only if no bytes were read. If an EOF happens after reading some but not all the bytes, Read returns ErrUnexpectedEOF.

▶ Example

▶ Example (Multi)

func ReadUvarint

```
func ReadUvarint(r io.ByteReader) (uint64, error)
```

ReadUvarint reads an encoded unsigned integer from r and returns it as a uint64. The error is EOF only if no bytes were read. If an EOF happens after reading some but not all the bytes, ReadUvarint returns io.ErrUnexpectedEOF.

func ReadVarint

```
func ReadVarint(r io.ByteReader) (int64, error)
```

ReadVarint reads an encoded signed integer from r and returns it as an int64. The error is EOF only if no bytes were read. If an EOF happens after reading some but not all the bytes, ReadVarint returns io.ErrUnexpectedEOF.

func Size

```
func Size(v any) int
```

Size returns how many bytes Write would generate to encode the value v, which must be a fixed-size value or a slice of fixed-size values, or a pointer to such data. If v is neither of these, Size returns -1.

func Uvarint

```
func Uvarint(buf []byte) (uint64, int)
```

Uvarint decodes a uint64 from buf and returns that value and the number of bytes read (> 0). If an error occurred, the value is 0 and the number of bytes n is <= 0 meaning:

```
n == 0: buf too small
n  < 0: value larger than 64 bits (overflow)
        and -n is the number of bytes read
```

▶ Example

func Varint

```
func Varint(buf []byte) (int64, int)
```

Varint decodes an int64 from buf and returns that value and the number of bytes read (> 0). If an error occurred, the value is 0 and the number of bytes n is <= 0 with the following meaning:

```
n == 0: buf too small
n  < 0: value larger than 64 bits (overflow)
        and -n is the number of bytes read
```

▶ Example

func Write

```
func Write(w io.Writer, order ByteOrder, data any) error
```

Write writes the binary representation of data into w. Data must be a fixed-size value or a slice of fixed-size values, or a pointer to such data. Boolean values encode as one byte: 1 for true, and 0 for false. Bytes written to w are encoded using the specified byte order and read from successive fields of the data. When writing structs, zero values are written for fields with blank (_) field names.

▶ Example

▶ Example (Multi)

Types

type AppendByteOrder

added in go1.19

```
type AppendByteOrder interface {
    AppendUint16([]byte, uint16) []byte
    AppendUint32([]byte, uint32) []byte
    AppendUint64([]byte, uint64) []byte
    String() string
}
```

AppendByteOrder specifies how to append 16-, 32-, or 64-bit unsigned integers into a byte slice.

type ByteOrder

```
type ByteOrder interface {
    Uint16([]byte) uint16
    Uint32([]byte) uint32
    Uint64([]byte) uint64
    PutUint16([]byte, uint16)
    PutUint32([]byte, uint32)
    PutUint64([]byte, uint64)
    String() string
}
```

A ByteOrder specifies how to convert byte slices into 16-, 32-, or 64-bit unsigned integers.

▶ Example (Get)

▶ Example (Put)

Source Files

View all

binary.go varint.go

Google