

[Discover Packages](#) > [Standard library](#) > [text](#) > [scanner](#) 

scanner





package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: 6 |

Imported by: 2,025

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 scanner provides a scanner and tokenizer for UTF-8-encoded text. It takes an `io.Reader` providing the source, which then can be tokenized through repeated calls to the `Scan` function. For compatibility with existing tools, the NUL character is not allowed. If the first character in the source is a UTF-8 encoded byte order mark (BOM), it is discarded.

By default, a Scanner skips white space and Go comments and recognizes all literals as defined by the Go language specification. It may be customized to recognize only a subset of those literals and to recognize different identifier and white space characters.

[▶ Example](#)[▶ Example \(IsIdentRune\)](#)[▶ Example \(Mode\)](#)[▶ Example \(Whitespace\)](#)

Index

[Constants](#)

```
func TokenString(tok rune) string
type Position
    func (pos *Position) IsValid() bool
    func (pos Position) String() string
type Scanner
    func (s *Scanner) Init(src io.Reader) *Scanner
    func (s *Scanner) Next() rune
    func (s *Scanner) Peek() rune
    func (s *Scanner) Pos() (pos Position)
    func (s *Scanner) Scan() rune
    func (s *Scanner) TokenText() string
```

Examples

Package

Package (IsIdentRune)

Package (Mode)

Package (Whitespace)

Constants

[View Source](#)

```
const (
    ScanIdents      = 1 << -Ident
    ScanInts        = 1 << -Int
    ScanFloats      = 1 << -Float // includes Ints and hexadecimal floats
    ScanChars       = 1 << -Char
    ScanStrings     = 1 << -String
    ScanRawStrings  = 1 << -RawString
    ScanComments    = 1 << -Comment
    SkipComments    = 1 << -skipComment // if set with ScanComments, comments become whitespace
    GoTokens        = ScanIdents | ScanFloats | ScanChars | ScanStrings | ScanRawStrings
)
```

Predefined mode bits to control recognition of tokens. For instance, to configure a Scanner such that it only recognizes (Go) identifiers, integers, and skips comments, set the Scanner's Mode field to:

```
ScanIdents | ScanInts | SkipComments
```

With the exceptions of comments, which are skipped if SkipComments is set, unrecognized tokens are not ignored. Instead, the scanner simply returns the respective individual characters (or possibly sub-tokens). For instance, if the mode is ScanIdents (not ScanStrings), the string "foo" is scanned as the token sequence "" Ident "".

Use GoTokens to configure the Scanner such that it accepts all Go literal tokens including Go identifiers. Comments will be skipped.

[View Source](#)

```
const (  
    EOF = -(iota + 1)  
    Ident  
    Int  
    Float  
    Char  
    String  
    RawString  
    Comment  
)
```

The result of Scan is one of these tokens or a Unicode character.

[View Source](#)

```
const GoWhitespace = 1<<'\\t' | 1<<'\\n' | 1<<'\\r' | 1<<' '
```

GoWhitespace is the default value for the Scanner's Whitespace field. Its value selects Go's white space characters.

Variables

This section is empty.

Functions

func TokenString

```
func TokenString(tok rune) string
```

TokenString returns a printable string for a token or Unicode character.

Types

type Position

```
type Position struct {  
    Filename string // filename, if any  
    Offset   int    // byte offset, starting at 0  
    Line     int    // line number, starting at 1  
    Column   int    // column number, starting at 1 (character count per line)  
}
```

Position is a value that represents a source position. A position is valid if Line > 0.

func (*Position) IsValid

```
func (pos *Position) IsValid() bool
```

IsValid reports whether the position is valid.

func (Position) String

```
func (pos Position) String() string
```

type Scanner

```
type Scanner struct {

    // Error is called for each error encountered. If no Error
    // function is set, the error is reported to os.Stderr.
    Error func(s *Scanner, msg string)

    // ErrorCount is incremented by one for each error encountered.
    ErrorCount int

    // The Mode field controls which tokens are recognized. For instance,
    // to recognize Ints, set the ScanInts bit in Mode. The field may be
    // changed at any time.
    Mode uint

    // The Whitespace field controls which characters are recognized
    // as white space. To recognize a character ch <= ' ' as white space,
    // set the ch'th bit in Whitespace (the Scanner's behavior is undefined
    // for values ch > ' '). The field may be changed at any time.
    Whitespace uint64

    // IsIdentRune is a predicate controlling the characters accepted
    // as the ith rune in an identifier. The set of valid characters
    // must not intersect with the set of white space characters.
    // If no IsIdentRune function is set, regular Go identifiers are
    // accepted instead. The field may be changed at any time.
    IsIdentRune func(ch rune, i int) bool

    // Start position of most recently scanned token; set by Scan.
    // Calling Init or Next invalidates the position (Line == 0).
    // The Filename field is always left untouched by the Scanner.
    // If an error is reported (via Error) and Position is invalid,
    // the scanner is not inside a token. Call Pos to obtain an error
    // position in that case, or to obtain the position immediately
    // after the most recently scanned token.
    Position
    // contains filtered or unexported fields
}
```

A Scanner implements reading of Unicode characters and tokens from an io.Reader.

func (*Scanner) Init

```
func (s *Scanner) Init(src io.Reader) *Scanner
```

Init initializes a Scanner with a new source and returns s. Error is set to nil, ErrorCount is set to 0, Mode is set to GoTokens, and Whitespace is set to GoWhitespace.

func (*Scanner) Next

```
func (s *Scanner) Next() rune
```

Next reads and returns the next Unicode character. It returns EOF at the end of the source. It reports a read error by calling s.Error, if not nil; otherwise it prints an error message to os.Stderr. Next does not update the Scanner's Position field; use Pos() to get the current position.

func (*Scanner) Peek

```
func (s *Scanner) Peek() rune
```

Peek returns the next Unicode character in the source without advancing the scanner. It returns EOF if the scanner's position is at the last character of the source.

func (*Scanner) Pos

```
func (s *Scanner) Pos() (pos Position)
```

Pos returns the position of the character immediately after the character or token returned by the last call to Next or Scan. Use the Scanner's Position field for the start position of the most recently scanned token.

func (*Scanner) Scan

```
func (s *Scanner) Scan() rune
```

Scan reads the next token or Unicode character from source and returns it. It only recognizes tokens t for which the respective Mode bit (1<<-t) is set. It returns EOF at the end of the source. It reports scanner errors (read and token errors) by calling s.Error, if not nil; otherwise it prints an error message to os.Stderr.

func (*Scanner) TokenText

```
func (s *Scanner) TokenText() string
```

TokenText returns the string corresponding to the most recently scanned token. Valid after calling Scan and in calls of Scanner.Error.



Source Files

[View all](#)

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