log

Package log go1.15.2 Latest

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

Overview

Package log implements a simple logging package. It defines a type, Logger, with methods for formatting output. It also has a predefined 'standard' Logger accessible through helper functions Print[f|ln], Fatal[f|ln], and Panic[f|ln], which are easier to use than creating a Logger manually. That logger writes to standard error and prints the date and time of each logged message. Every log message is output on a separate line: if the message being printed does not end in a newline, the logger will add one. The Fatal functions call os.Exit(1) after writing the log message. The Panic functions call panic after writing the log message.

Constants

```
const (
    Ldate         = 1 << iota     // the date in the local time zone: 2009/01/23
    Ltime                         // the time in the local time zone: 01:23:23
    Lmicroseconds                 // microsecond resolution: 01:23:23.123123.  assume
    Llongfile                     // full file name and line number: /a/b/c/d.go:23
    Lshortfile                    // final file name element and line number: d.go:23
    LUTC                          // if Ldate or Ltime is set, use UTC rather than th
    Lmsgprefix                    // move the "prefix" from the beginning of the line
    LstdFlags     = Ldate | Ltime // initial values for the standard logger
)
```

These flags define which text to prefix to each log entry generated by the Logger. Bits are or'ed together to control what's printed. With the exception of the Lmsgprefix flag, there is no control over the order they appear (the order listed here) or the format they present (as described in the comments). The prefix is followed by a colon only when Llongfile or Lshortfile is specified. For example, flags Ldate | Ltime (or LstdFlags) produce,

```
2009/01/23 01:23:23 message
```

while flags Ldate | Ltime | Lmicroseconds | Llongfile produce,

```
2009/01/23 01:23:23.123123 /a/b/c/d.go:23: message
```

func Fatal

```
func Fatal(v ...interface{})
```

Fatal is equivalent to Print() followed by a call to os.Exit(1).

func Fatalf

```
func Fatalf(format string, v ...interface{})
```

Fatalf is equivalent to Printf() followed by a call to os.Exit(1).

func Fatalln

```
func Fatalln(v ...interface{})
```

Fatalln is equivalent to Println() followed by a call to os.Exit(1).

func Flags

```
func Flags() int
```

Flags returns the output flags for the standard logger. The flag bits are Ldate, Ltime, and so on.

func Output

```
func Output(calldepth int, s string) error
```

Output writes the output for a logging event. The string s contains the text to print after the prefix specified by the flags of the Logger. A newline is appended if the last character of s is not already a newline. Calldepth is the count of the number of frames to skip when computing the file name and line number if Llongfile or Lshortfile is set; a value of 1 will print the details for the caller of Output.

func Panic

```
func Panic(v ...interface{})
```

Panic is equivalent to Print() followed by a call to panic().

func Panicf

```
func Panicf(format string, v ...interface{})
```

Panicf is equivalent to Printf() followed by a call to panic().

func Panicln

```
func Panicln(v ...interface{})
```

Panicln is equivalent to Println() followed by a call to panic().

func Prefix

```
func Prefix() string
```

Prefix returns the output prefix for the standard logger.

func Print

```
func Print(v ...interface{})
```

Print calls Output to print to the standard logger. Arguments are handled in the manner of fmt.Print.

func Printf

```
func Printf(format string, v ...interface{})
```

Printf calls Output to print to the standard logger. Arguments are handled in the manner of fmt.Printf.

func Println

```
func Println(v ...interface{})
```

Println calls Output to print to the standard logger. Arguments are handled in the manner of fmt.Println.

func SetFlags

```
func SetFlags(flag int)
```

SetFlags sets the output flags for the standard logger. The flag bits are Ldate, Ltime, and so on.

func SetOutput

```
func SetOutput(w io.Writer)
```

SetOutput sets the output destination for the standard logger.

func SetPrefix

```
func SetPrefix(prefix string)
```

SetPrefix sets the output prefix for the standard logger.

func Writer

```
func Writer() io.Writer
```

Writer returns the output destination for the standard logger.

type Logger

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

A Logger represents an active logging object that generates lines of output to an io.Writer. Each logging operation makes a single call to the Writer's Write method. A Logger can be used simultaneously from multiple goroutines; it guarantees to serialize access to the Writer.

func New

```
func New(out io.Writer, prefix string, flag int) *Logger
```

New creates a new Logger. The out variable sets the destination to which log data will be written. The prefix appears at the beginning of each generated log line, or after the log header if the Lmsgprefix flag is provided. The flag argument defines the logging properties.

func (*Logger) Fatal

```
func (l *Logger) Fatal(v ...interface{})
```

Fatal is equivalent to l.Print() followed by a call to os.Exit(1).

func (*Logger) Fatalf

```
func (l *Logger) Fatalf(format string, v ...interface{})
```

Fatalf is equivalent to l.Printf() followed by a call to os.Exit(1).

func (*Logger) Fatalln

```
func (l *Logger) Fatalln(v ...interface{})
```

Fatalln is equivalent to l.Println() followed by a call to os.Exit(1).

func (*Logger) Flags

```
func (l *Logger) Flags() int
```

Flags returns the output flags for the logger. The flag bits are Ldate, Ltime, and so on.

func (*Logger) Output

```
func (l *Logger) Output(calldepth int, s string) error
```

Output writes the output for a logging event. The string s contains the text to print after the prefix specified by the flags of the Logger. A newline is appended if the last character of s is not already a newline. Calldepth is used to recover the PC and is provided for generality, although at the moment on all pre-defined paths it will be 2.

func (*Logger) Panic

```
func (l *Logger) Panic(v ...interface{})
```

Panic is equivalent to l.Print() followed by a call to panic().

func (*Logger) Panicf

```
func (l *Logger) Panicf(format string, v ...interface{})
```

Panicf is equivalent to l.Printf() followed by a call to panic().

func (*Logger) Panicln

```
func (l *Logger) Panicln(v ...interface{})
```

Panicln is equivalent to l.Println() followed by a call to panic().

func (*Logger) Prefix

```
func (l *Logger) Prefix() string
```

Prefix returns the output prefix for the logger.

func (*Logger) Print

```
func (l *Logger) Print(v ...interface{})
```

Print calls l.Output to print to the logger. Arguments are handled in the manner of fmt.Print.

func (*Logger) Printf

```
func (l *Logger) Printf(format string, v ...interface{})
```

Printf calls l.Output to print to the logger. Arguments are handled in the manner of fmt.Printf.

func (*Logger) Println

```
func (l *Logger) Println(v ...interface{})
```

Println calls l.Output to print to the logger. Arguments are handled in the manner of fmt.Println.

func (*Logger) SetFlags

```
func (l *Logger) SetFlags(flag int)
```

SetFlags sets the output flags for the logger. The flag bits are Ldate, Ltime, and so on.

func (*Logger) SetOutput

```
func (l *Logger) SetOutput(w io.Writer)
```

SetOutput sets the output destination for the logger.

func (*Logger) SetPrefix

```
func (l *Logger) SetPrefix(prefix string)
```

SetPrefix sets the output prefix for the logger.

func (*Logger) Writer

```
func (l *Logger) Writer() io.Writer
```

Writer returns the output destination for the logger.