listing.go

2013-03-10 22:10:43 MDT

golist is a Go utility for producing readable Go source listings using markdown. There are two rules it uses in producing these markdown listings:

1. lines beginning with a double slash are treated as markdown text.
2. all other lines are indented with a tab; according to markdown's syntax, this should produce a code listing.

Currently, the only output formats supported are writing to standard out or a markdown file.

```
package main

import (
    "bufio"
    "flag"
    "fmt"
    "io"
    "io/ioutil"
    "os"
    "path/filepath"
    "regexp"
    "time"
)

const DefaultDateFormat = "2006-01-02 15:04:05 MST"

var (
    CommentLine     = regexp.MustCompile("^\\s*//\\s*")
    DateFormat      = DefaultDateFormat
    InputFormats    map[string]SourceTransformer
    OutputFormats   map[string]OutputWriter
    OutputDirectory string
)
```

A SourceTransformer converts the source code to desired form. For example, it might convert the source to markdown, which can then be passed to a conversion function.

```
type SourceTransformer func(string) (string, error)
```

An OutputWriter takes markdown source and an output file name, and handles its output, whether writing to a file or displaying to screen.

```
type OutputWriter func(string, string) error

func init() {
    InputFormats = make(map[string]SourceTransformer, 0)
    InputFormats["markdown"] = SourceToMarkdown
    InputFormats["tex"] = SourceToLatex

    OutputFormats = make(map[string]OutputWriter, 0)
    OutputFormats["-"] = ScreenWriter
    OutputFormats["html"] = HtmlWriter
    OutputFormats["latex"] = PandocTexWriter
    OutputFormats["md"] = MarkdownWriter
    OutputFormats["pdf"] = PdfWriter
    OutputFormats["tex"] = TexWriter
}
```

SourceToMarkdown takes a file and returns a string containing the source converted to markdown.

```
func SourceToMarkdown(filename string) (markdown string, err error) {
    file, err := os.Open(filename)
    if err != nil {
        return
    }
    defer file.Close()
    buf := bufio.NewReader(file)

    var (
        line      string
        longLine  bool
        lineBytes []byte
        isPrefix  bool
        comment   = true
    )

    markdown += "## " + filename + "\n"
    printDate := time.Now().Format(DateFormat)
    markdown += "<small>" + printDate + "</small>\n\n"

    for {
        err = nil
        lineBytes, isPrefix, err = buf.ReadLine()
```

```
        if io.EOF == err {
            err = nil
            break
        } else if err != nil {
            break
        } else if isPrefix {
            line += string(lineBytes)

            longLine = true
            continue
        } else if longLine {
            line += string(lineBytes)
            longLine = false
        } else {
            line = string(lineBytes)
        }

        if CommentLine.MatchString(line) {
            if !comment {
                markdown += "\n"
            }
            markdown += CommentLine.ReplaceAllString(line, "")
            comment = true
        } else {
```

The comment flag is used to trigger a newline before a codeblock; in some markdown implementations, not doing this will cause the code block to not be displayed properly.

```
            if comment {
                markdown += "  \n"
                comment = false
            }
            markdown += "\t"
            markdown += line
        }
        markdown += "\n"
    }
    return
}

func main() {
    fDateFormat := flag.String("t", DefaultDateFormat,
        "specify a format for the listing date")
    fOutputFormat := flag.String("o", "-", "output format")
```

```
    fOutputDir := flag.String("d", ".",
        "directory listings should be saved in.")
    flag.Parse()

    DateFormat = *fDateFormat
    OutputDirectory = *fOutputDir

    var transformer SourceTransformer

    outHandler, ok := OutputFormats[*fOutputFormat]
    if !ok {
        fmt.Printf("[!] %s is not a supported output format.\n",
            *fOutputFormat)
        fmt.Println("Supported formats:")
        fmt.Println("\t-        write markdown to standard output")
        fmt.Println("\thtml     produce an HTML listing")
        fmt.Println("\tlatex    produce a LaTeX listing")
        fmt.Println("\tmd       write markdown to file")
        fmt.Println("\tpdf      produce a PDF listing")
        fmt.Println("\ttex      produce a TeX listing")
        os.Exit(1)
    }

    if *fOutputFormat != "tex" {
        transformer = InputFormats["markdown"]
    } else {
        transformer = InputFormats["tex"]
    }

    for _, sourceFile := range flag.Args() {
        out, err := transformer(sourceFile)
        if err != nil {
            fmt.Fprintf(os.Stderr,
                "[!] couldn't convert %s to listing: %s\n",
                sourceFile, err.Error())
            continue
        }
        if err := outHandler(out, sourceFile); err != nil {
            fmt.Fprintf(os.Stderr,
                "[!] couldn't convert %s to listing: %s\n",
                sourceFile, err.Error())
        }
    }

}
```

GetOutFile joins the output directory with the filename.

```
func GetOutFile(filename string) string {
    return filepath.Join(OutputDirectory, filename)
}
```

ScreenWriter prints the markdown to standard output.

```
func ScreenWriter(markdown string, filename string) (err error) {
    _, err = fmt.Println(markdown)
    return
}
```

MarkdownWriter writes the transformed listing to a file.

```
func MarkdownWriter(listing string, filename string) (err error) {
    outFile := GetOutFile(filename + ".md")
    err = ioutil.WriteFile(outFile, []byte(listing), 0644)
    return
}
```