listing.go

2013-03-10 19:55:08 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"
    "regexp"
    "time"
)

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

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

An output writer 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() {
    OutputFormats = make(map[string]OutputWriter, 0)
    OutputFormats["-"] = ScreenWriter
    OutputFormats["html"] = HtmlWriter
```

```
    OutputFormats["md"] = MarkdownWriter
    OutputFormats["pdf"] = PdfWriter
}
```

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   bool
    )

    markdown += "## " + filename + "\n"
    printDate := time.Now().Format(DateFormat)
    markdown += "### " + printDate + "\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")
    flag.Parse()

    DateFormat = *fDateFormat
    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("\tmd       write markdown to file")
        os.Exit(1)
    }

    for _, sourceFile := range flag.Args() {
```

```
        md, err := SourceToMarkdown(sourceFile)
        if err != nil {
            fmt.Fprintf(os.Stderr,
                "[!] couldn't convert %s to markdown: %s\n",
                sourceFile, err.Error())
            continue
        }
        if err := outHandler(md, sourceFile); err != nil {
            fmt.Fprintf(os.Stderr,
                "[!] couldn't convert %s to markdown: %s\n",
                sourceFile, err.Error())
        }
    }

}
```

ScreenWriter prints the markdown to standard output.

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

MarkdownWriter writes the markdown listing to a file.

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