

# Qzlogs for GO Lang Windows

## ログ支援・開発ツールライブラリ

フリー・シンプル版

Win64 DLL版

logsplgo\_x64.dll

logsplgo\_x64.lib

サポートプラットフォーム

Windows 10 1909 以降

対応プログラミング言語

Go Lang 1.15.6 windows/amd64 Windows64 Windows 10 20H2 で確認しております

開発元 QuiZ Lab. 2021. 01. 25

バージョン 0.5.1.2

### 機能概要

- ・ 一行ログ出力
- ・ ダンプ機能 (UTF8文字列ダンプ) NULLを含むバイナリダンプは未サポート  
代わりに、logPutHexでバイトデータを列挙できます
- ・ 1文字logPut機能
- ・ ビットイメージ表示機能 UINT16, UINT32, Unicodeスカラ値からUTF8ビットイメージ
- ・ win32エラー日本語メッセージ取得、win32エラー日本語メッセージ・ログ記録
- ・ MessageBox 表示のみ、MessageBox Ok/Cancel
- ・ v 0.5.1.0以降より、マイクロ秒計測のプロファイラ関数を追加しました。

シンプル版では、ログの日付変更ローテート、指定サイズローテート、各種文字コード指定出力機能はありません。出力文字コードはUNICODE(**UTF8**)プロジェクトのみの対応となります。一度に出力できる文字列サイズは8100バイトになります。UNICODE(UTF16)だと1文字2バイト使いますので、4050文字程度の出力上限値になります。UTF8に書き変わったUTF16サロゲート文字は4バイトのため、これよりも少なくなる可能性があります。また、学習用を想定しておりますので、マルチスレッディングからの呼び出しは考慮されておりません。Webサーバからの利用などは動作保証できかねます。今後、マルチスレッドサーバ用のデバッグツールも対応したいと考えております。

プログラミングをはじめたばかりの方は、どのタイミングでどうプログラムが動くのか掴みにくい傾向にあります。ログを張りめぐらせば、どう動作しているのかいち早く理解することができます。ぜひ、プログラミング初心者の方に利用していただければと思います。

### ツールの構成

#### 1. win64 DLLバージョン

使い方は簡単です。 初期化、終了処理の間にログ出力ファンクションを呼び出すだけです。

## 目 次

0. 利用イメージ Go Lang (Windows 10)	3
1. Qzlogsのセットアップ	4
1.1. すべてのプロジェクトから参照するインストール	4
1.8. Qzlogsを特定のプロジェクトのみで利用する場合	10
2. 公開メソッド・マニュアル	14
3. Write Flag パラメータオプション説明	19
4. デバッグ時のQzlogsの使い方のヒントなど	20
5. 実装例など	21
5.1. ビット文字列表示メソッド makeBitStringUSHORT16(), makeBitStringUINT32(), makeBitStringUnicodeScalarValue()	21
5.2. Win32エラーコードとエラー文字のロギング	23
5.3. メソッドの 出入口にログを書きしておくテクニック	24
5.4. logSleepSw = 1 の簡易ログ停止機能	26
5.5. UTF8 文字列ダンプ時の禁則処理の説明	28
6. サンプルプログラム	29
7. 仕様	30

お詫び 一部、ドキュメント内の画像文字列と、リリースプログラムの出力結果が異なる場合がございます。開発途上でドキュメント作成開始したため、多少の齟齬が残る場合がございます。謹んでお詫び申し上げます。

## 改版履歴

2020/12/20 初版 0.5.0.0 (Base Python版)

2021/01/25 改版 0.5.1, 2 変更履歴はChangeLog.txt参照

### ■ライセンス

個人が、学習目的、オープンソース開発に利用する場合は利用に制限はありません。  
完全フリーウェアとしてご利用ください。

企業または個人が対価を求めるプロジェクトでご利用される場合は、基本フリーウェアとしてご利用頂けますが、必ずご一報ください。

本ソフトウェアツールを利用して発生した損害につきましては、いかなる場合に於いても保障することはできません。すべて利用する側の責任でご利用ください。また動作の完全性も保証はできかねます。下記にもあるように、バグ対応のメンテナンスは積極的に行いますので、ぜひご利用されて、フィードバックをいただけると有難く存じます。

### ■作者への連絡など


バグ、ご要望などがあれば。バグについては、積極的にメンテナンスする予定です。ただし、その他のお問い合わせにつきましては必ずお返事できるかはお約束出来かねます。

netbsdmania@gmail.com

### ■ Go言語について

私も、本ログ・ライブラリを作成するために学習したばかりの初心者でございます。Go言語のセオリーにそぐわない箇所も多々あるかと思いますが、ぜひ識者様のご助言など頂けましたら、次回以降に反映させたい所存です。

## 0. 利用イメージ Go Lang (Windows 10)



```
qztest01.go - メモ帳
ファイル(E) 編集(E) 書式(Q) 表示(V) ヘルプ(H)
package main

import "qzlogs"

func main() {
    qzlogs.LogInit("D:\\PrgFiles\\go\\prj\\qzlogstest\\cmd", //ログファイル出力先フォルダ
        "QZ_TEST01_", //ログファイル名識別用prefix
        1, //writeFlg 1行ログ、先頭の日付時間などのOption
        1, //CrLfOpt 0...CR+LF 1...LF 2...CR
        0, //LogSleepSw 0...ログ出力 1...ログ・スリープ(記録中止)
        0, //LogLevel 1-5 レベルに合致したログメソッドのみ出力する
        1) //utf8bom 0...UTF8 Bomなし 1...UTF8 Bomあり

    qzlogs.Log("hoge uga piyo dasa!")

    qzlogs.LogTerm()
}
```

13 行, 68 列 100% Windows (CRLF) UTF-8

※ サンプルの配置位置を D:\work に変更しました。インストール時は、そのように置き換えてご確認ください。 go/binにパスが通っていれば問題は出ないはずですが、問題があれば、どうぞご指摘お願いいたします。

GO言語の環境構築につきましては、事前に本家サイトをご確認のうえ、構成してください。

# 1. Qzlogsのセットアップ

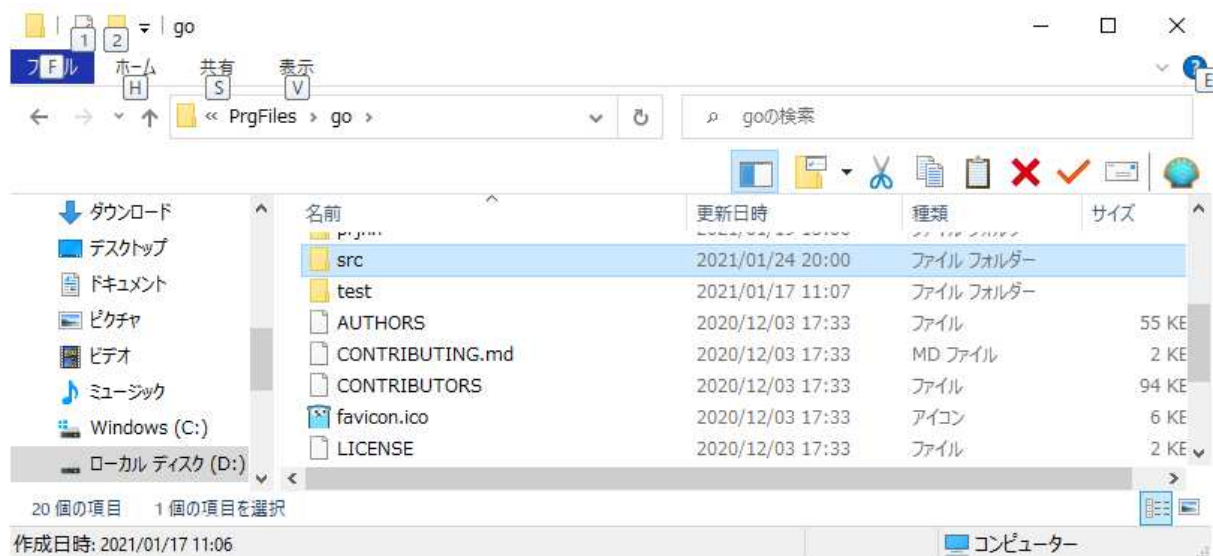
## 1.1. すべてのプロジェクトから参照するインストール

Goのインストール・ルートに移動します

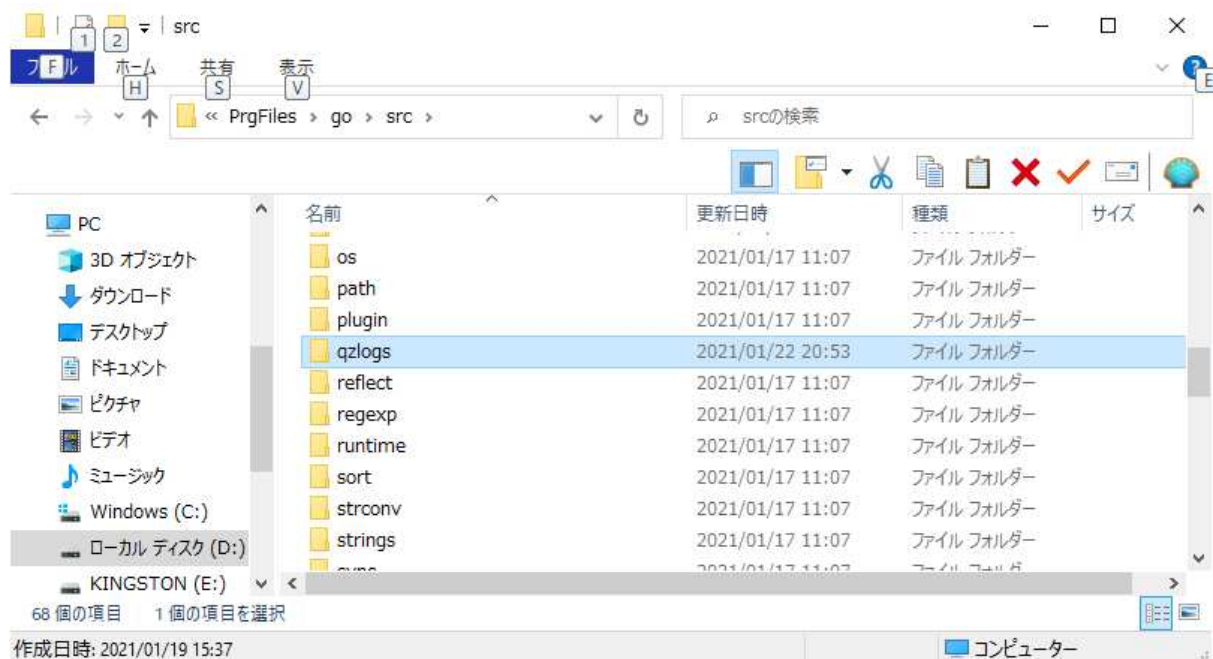
GOはインストール、動作確認済みの前提です、インストールについては、Go Langの公認サイトをご確認ください)環境にパッケージフォルダをコピーするだけです・・・

zipファイルを解凍し、qzlogsフォルダをフォルダごと GOインストール先:/src/にコピーする作業になります。

1.2.Go->srcまで、エクスプローラで開きます



1.3. Windowsエクスプローラを使い、展開した先のqzlogsフォルダをGo/src以下に貼り付けます。



1.4. qzlogsフォルダにある、qzlogsUTF8.go ファイルをメモ帳で開き、以下の内容を修正します  
注意 dllはフルパスで指定してください。パスは正確に。

このパスは、利用される方がGoをインストールした場所に依存しますので、正確に変更してください。

The screenshot shows a Windows File Explorer window with the address bar set to 'PrgFiles > go > src > qzlogs'. The file list contains three items:

名前	更新日時	種類	サイズ
logsplgo_x64.dll	2021/01/24 18:22	アプリケーション拡張	49 KB
logsplgo_x64.lib	2021/01/24 18:22	Object File Libr...	6 KB
qzlogsUTF8.go	2021/01/24 19:05	GO ファイル	8 KB

Below the file explorer is a code editor window showing the Go code for 'qzlogsUTF8.go'. The code includes a comment and a path placeholder that is highlighted in blue in the original image:

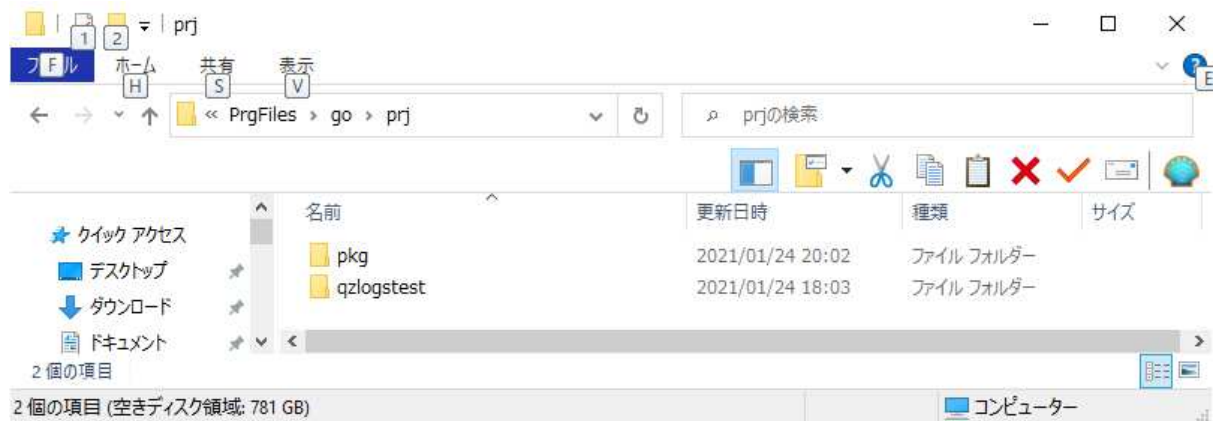
```
// 以下の行のパス位置は、このパッケージをインストールした場所に依存します。正確に書き換えてご利用ください。
// また logsplgo_x64.dllも必ず同じパス位置に配置してください。
qzlogsU8 = syscall.NewLazyDLL(D:\\PrgFiles\\go\\src\\qzlogs\\logsplgo_x64.dll")
logInitRaw = qzlogsU8.NewProc("logInit")
logTermRaw = qzlogsU8.NewProc("logTerm")
logRaw = qzlogsU8.NewProc("log")
logDebugRaw = qzlogsU8.NewProc("log_debug")
logInfoRaw = qzlogsU8.NewProc("log_info")
logWarningRaw = qzlogsU8.NewProc("log_warning")
logErrorRaw = qzlogsU8.NewProc("log_error")
logCriticalRaw = qzlogsU8.NewProc("log_critical")

logDumpUTF8Raw = qzlogsU8.NewProc("logDumpUTF8")
logPutRaw = qzlogsU8.NewProc("logPut")
logPutHexRaw = qzlogsU8.NewProc("logPutHex")
makeBitStringsUINT32Raw = qzlogsU8.NewProc("makeBitStringsUINT32")
makeBitStringsUSHORT16Raw = qzlogsU8.NewProc("makeBitStringsUSHORT16")
makeBitStringsUnicodeScalarValueRaw = qzlogsU8.NewProc("makeBitStringsUnicodeScalarValue")
msgBoxRaw = qzlogsU8.NewProc("msgBox")
msgBoxOkCancelRaw = qzlogsU8.NewProc("msgBoxOkCancel")
getLastErrRaw = qzlogsU8.NewProc("GetLastErr")
profileTimeStartRaw = qzlogsU8.NewProc("profileTimeStart")
profileTimeStopRaw = qzlogsU8.NewProc("profileTimeStop")
```

利用される方が今、コピーした作業場所を指定してください

1.5.書き換えたら、サンプル・プログラムをコピーします。zipを解凍した中に、samples/qzlogtestフォルダがありますが、qzlogtestフォルダをgo/prj内にコピーします。prjは新規フォルダとして作成してください。

※ サンプルパッケージは、go version [Enter]で動作していれば、他の場所でもかまいません  
たとえば D:\work\qzlogtest みたいに他の場所にコピーします



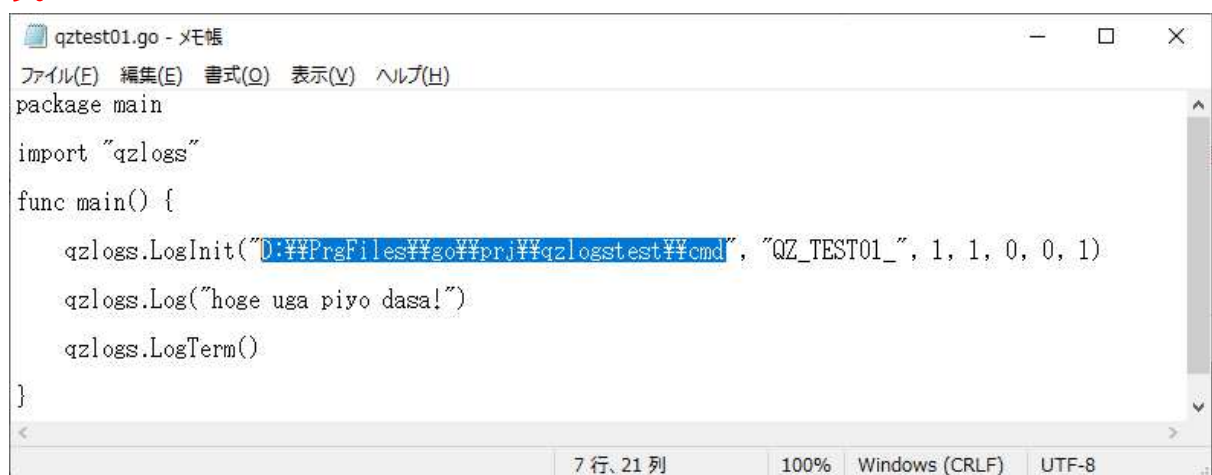
まず、qztest01のサンプルをビルドしてみます。  
ビルドする場合は cmdフォルダに入ります。



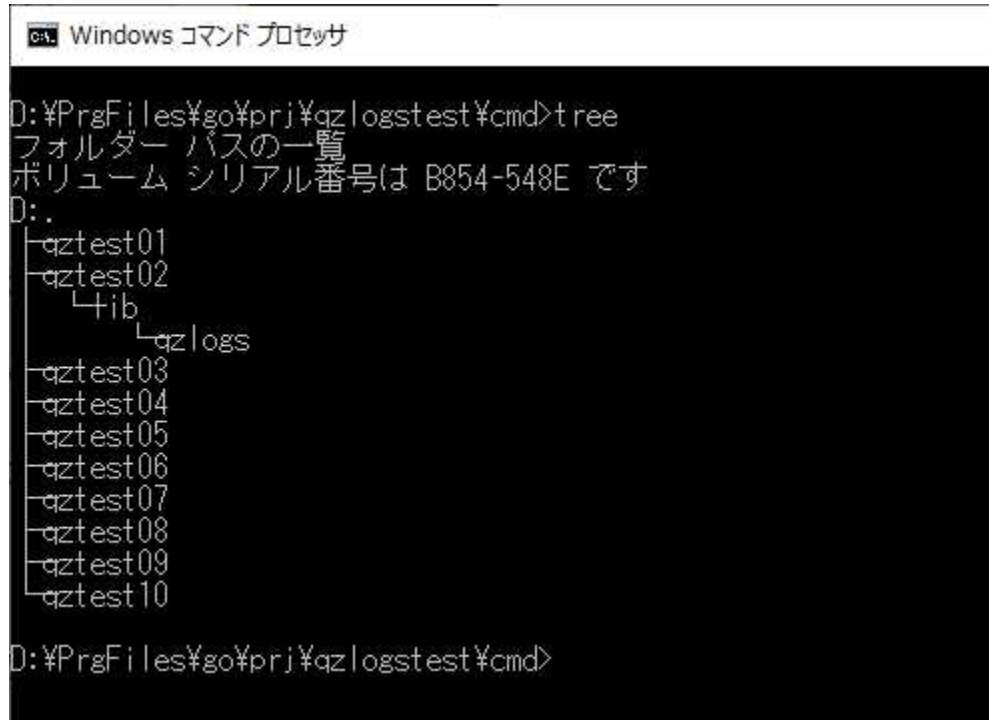
次に、qztest01フォルダ内のqztest01.goファイルをメモ帳やutf8対応エディタで開き、ログ・出力パスを、ダウンロードした方がインストールした先書き換えます。

cmd位置からビルドをするので、最後のフォルダ・パスはcmdにしてください。

**例 あなたがインストールしたgoが C:\go だとしたら "C:\go\prj\qzlogtest\cmd"になります。**



## ディレクトリ構成図



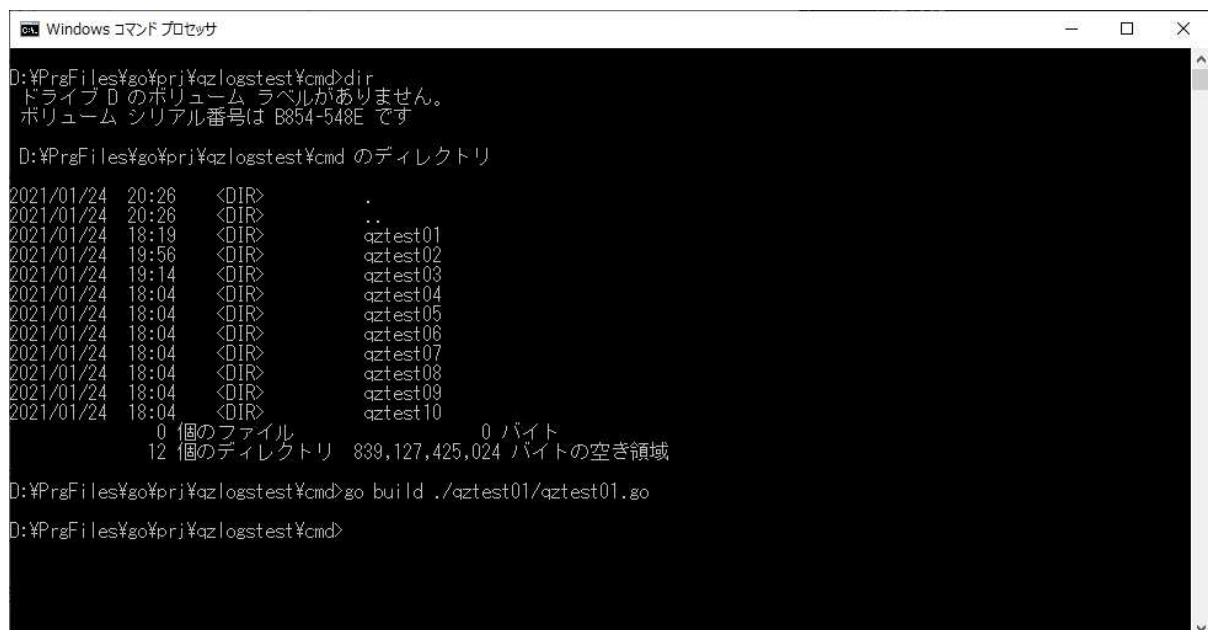
```
Windows コマンド プロセッサ
D:\PrgFiles\go\prj\qz\logstest\cmd>tree
フォルダー パスの一覧
ボリューム シリアル番号は B854-548E です
D:
├── qztest01
├── qztest02
│   ├── lib
│   └── qzlogs
├── qztest03
├── qztest04
├── qztest05
├── qztest06
├── qztest07
├── qztest08
├── qztest09
└── qztest10

D:\PrgFiles\go\prj\qz\logstest\cmd>
```

ログ・出力パスを書き換えたら、ビルドします。

go build ./qztest01/qztest01.go [Enter]とコマンドプロンプトから実行します。

実行後の状態。



```
Windows コマンド プロセッサ
D:\PrgFiles\go\prj\qz\logstest\cmd>dir
ドライブ D のボリューム ラベルがありません。
ボリューム シリアル番号は B854-548E です

D:\PrgFiles\go\prj\qz\logstest\cmd のディレクトリ
2021/01/24  20:26    <DIR>        .
2021/01/24  20:26    <DIR>        ..
2021/01/24  18:19    <DIR>        qztest01
2021/01/24  19:56    <DIR>        qztest02
2021/01/24  19:14    <DIR>        qztest03
2021/01/24  18:04    <DIR>        qztest04
2021/01/24  18:04    <DIR>        qztest05
2021/01/24  18:04    <DIR>        qztest06
2021/01/24  18:04    <DIR>        qztest07
2021/01/24  18:04    <DIR>        qztest08
2021/01/24  18:04    <DIR>        qztest09
2021/01/24  18:04    <DIR>        qztest10
                0 個のファイル                0 バイト
                12 個のディレクトリ  839,127,425,024 バイトの空き領域

D:\PrgFiles\go\prj\qz\logstest\cmd>go build ./qztest01/qztest01.go
D:\PrgFiles\go\prj\qz\logstest\cmd>
```



エラーがなければ、qztest01.exeが作成されます。

```
Windows コマンド プロセッサ

D:\PrgFiles¥go¥prj¥qzlogstest¥cmd>dir
ドライブ D のボリューム ラベルがありません。
ボリューム シリアル番号は B854-548E です

D:\PrgFiles¥go¥prj¥qzlogstest¥cmd のディレクトリ

2021/01/24  20:26    <DIR>          .
2021/01/24  20:26    <DIR>          ..
2021/01/24  18:19    <DIR>          qztest01
2021/01/24  20:26    1,387,008 qztest01.exe
2021/01/24  19:56    <DIR>          qztest02
2021/01/24  19:14    <DIR>          qztest03
```

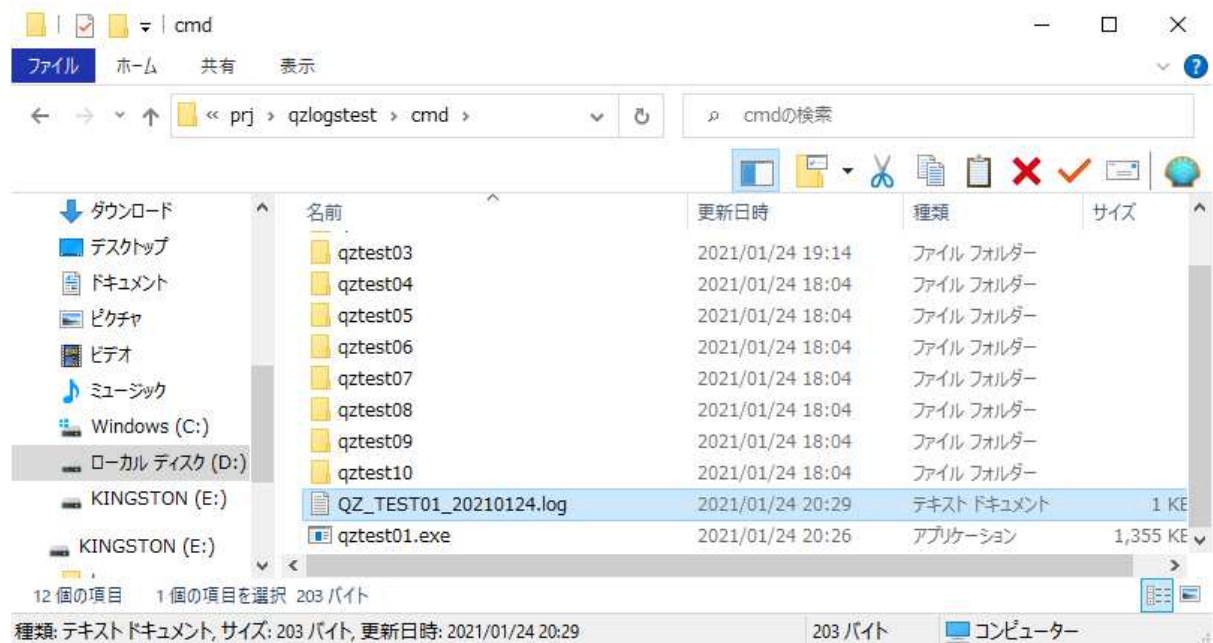
実行します。

```
Windows コマンド プロセッサ

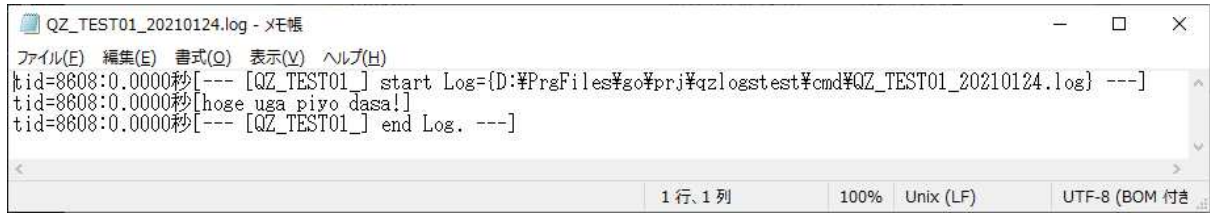
D:\PrgFiles¥go¥prj¥qzlogstest¥cmd>qztest01

D:\PrgFiles¥go¥prj¥qzlogstest¥cmd>
```

1.6. エクスプローラーで、ログファイルを見てみます



## 1.7.メモ帳で開きます



上記状態のファイルが見れたら、動作テスト完了です。

**※ビルドエラーになる場合は 1.4.のdllをロードするパス設定が誤っている場合があります。見直してください。**

基本的な利用方法は、初期化 ログ出力メソッド利用 終了処理 となります。

logInitは、初期化パラメータをすべてここで渡す初期化方法です。初期化は必ずこのメソッドでログ用のとファイル・ハンドルを開いておく必要があります。

1. string path...ログを出力するパス位置。(ファイル名は必要ありません) string prefix...ログ出力するファイルの先頭識別文字列
2. string prefix....ログファイル名の先頭文字列です。識別用に必ず入力します
3. enum WriteFlg...ログ出力する際の、日付、時間、スレッドID, 処理時間などのオプションを選択します。詳細はマニュアル、ログ出力オプションを参照してください。(P12 3. 参照)
4. enum CrLfOpt ...改行コードの指定です。 0...CR+LF 1... CR, 2...LF となります。
5. enum LogSleepSw...ログ出力の開始、停止スイッチです。0...ログ出力 1...ログ・スリープ (記録しません)
6. LogLevelは 1-6の値で、出力するメソッドを制御します。数値に対応するログメソッド詳細は、2. 公開メソッド・マニュアル P7を参照
7. utf8 BOM は 出力するログファイルに、UTF8の  
0...BOMをつけない  
1...BOMを付ける

もし、ログ出力パスを間違えると・・・



こんなダイアログが出てきます。有効なディスク領域で、フォルダがない場合は、フォルダを新規作成してログ出力されます。

## 1.8. Qzlogsを特定のプロジェクトのみで利用する場合

サンプル qztest02

まず、他のプロジェクトに影響させないために、1.1. のインストールは行いません。もしインストール済みの場合は、`$go/src/qzlogs` ←のフォルダ名を`qzlogs_sleep` とかにリネームしておきます。また他のプロジェクトからも利用したいときに `qzlogs`の名前に戻します。この点ご注意ください。

1.9. goには標準的なディレクトリ構成があります。かなり沢山のオプションがあり、複雑化しますが、ここでは、最小の構成で説明を進めていきます。

以下のディレクトリ構成になります。

G0インストール先/prj/qzlogstest---cmd--qztest02---qztest02.goと

```
cmd/qztest02/lib/qzlogs-----logsplgo_x64.dll
cmd/qztest02/lib/qzlogs-----logsplgo_x64.lib
cmd/qztest02/lib/qzlogs-----qzlogsUTF8.go
```

解凍したqzlogsフォルダをlib/に貼り付けます

1.4. で修正した場所のdllロードパスをこの位置のパスに書き換えます。

注意、先にインストールし、`qzlogs_sleep`にリネームした場所の  
(`go/src/qzlogs_sleep`)内のパスには手をつけません。

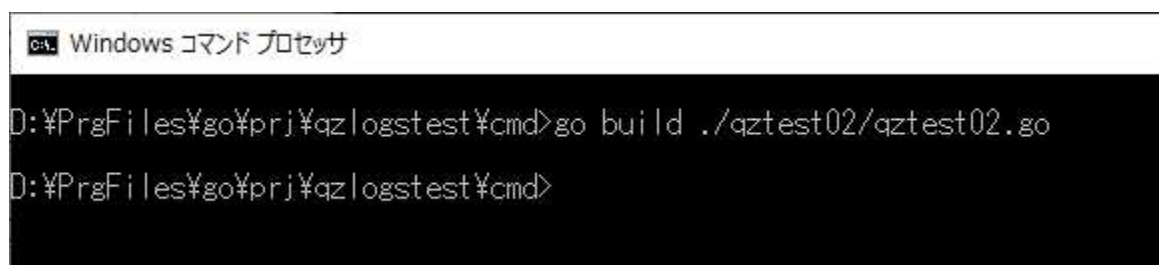
新たに、今コピーしたqzlogsフォルダの中のqzlogsUTF8.goを開き、パスを修正します。  
私の環境はD:\PrgFiles\go がG0のインストール場所なので、これを基本に説明します。

go build 位置を やはり cmdとします。



1.10. ビルドコマンドは、この位置から打ちます

go build ./qztest02/qztest02.go [Enter]



ビルド成功例

失敗する場合。書き換えたパスが間違っているばあいのエラーメッセージ

わざと存在しないドライブ x:にしてみました

書き換えたのは、qztest02アプリのみ利用するパッケージの

D:\PrgFiles\go\prj\qzlogstest\cmd\qztest02\lib\qzlogs内にあるqzlogsUTF8.goです。

```
qzlogsUTF8.go - X86
ファイル(E) 編集(E) 書式(O) 表示(V) ヘルプ(H)

import (
    "syscall"
    "unsafe"
)

var (
    // 以下の行のパス位置は、このパッケージをインストールした場所に依存します。正確に書き換えてご利用ください。
    // また logspigo_x64.dll, logspigo_x64.libも必ず同じパス位置に配置してください。
    qzlogsU8 = syscall.NewLazyDLL("D:\PrgFiles\go\src\qzlogs\logspigo_x64.dll")
    logInitRaw = qzlogsU8.NewProc("log_init")
    logTermRaw = qzlogsU8.NewProc("log_term")
    logRaw = qzlogsU8.NewProc("log")
    logDebugRaw = qzlogsU8.NewProc("log_debug")
    logInfoRaw = qzlogsU8.NewProc("log_info")
    logWarningRaw = qzlogsU8.NewProc("log_warning")
    logErrorRaw = qzlogsU8.NewProc("log_error")
    logCriticalRaw = qzlogsU8.NewProc("log_critical")

    logDumpUTF8Raw = qzlogsU8.NewProc("logDumpUTF8")
    logPutRaw = qzlogsU8.NewProc("logPut")
    logPutHexRaw = qzlogsU8.NewProc("logPutHex")
    makeBitStringUINT32Raw = qzlogsU8.NewProc("makeBitStringUINT32")
    makeBitStringUSHORT16Raw = qzlogsU8.NewProc("makeBitStringUSHORT16")
    makeBitStringUnicodeScalarValueRaw = qzlogsU8.NewProc("makeBitStringUnicodeScalarValue")
    msgBoxRaw = qzlogsU8.NewProc("msgBox")
    msgBoxOkCancelRaw = qzlogsU8.NewProc("msgBoxOkCancel")
    getLastErrRaw = qzlogsU8.NewProc("GetLastErr")
    profileTimeStartRaw = qzlogsU8.NewProc("profileTimeStart")
    profileTimeStopRaw = qzlogsU8.NewProc("profileTimeStop")

    w32_reset_errCodeRaw = qzlogsU8.NewProc("w32_reset_errCode")
)

28行, 40列 100% Windows (CRLF) UTF-8
```

コンパイルは通りますが・・・

```
選択Windows コマンド プロセッサ

D:\PrgFiles\go\prj\qzlogstest\cmd>go build ./qztest02/qztest02.go
D:\PrgFiles\go\prj\qzlogstest\cmd>go build ./qztest02/qztest02.go
D:\PrgFiles\go\prj\qzlogstest\cmd>
```

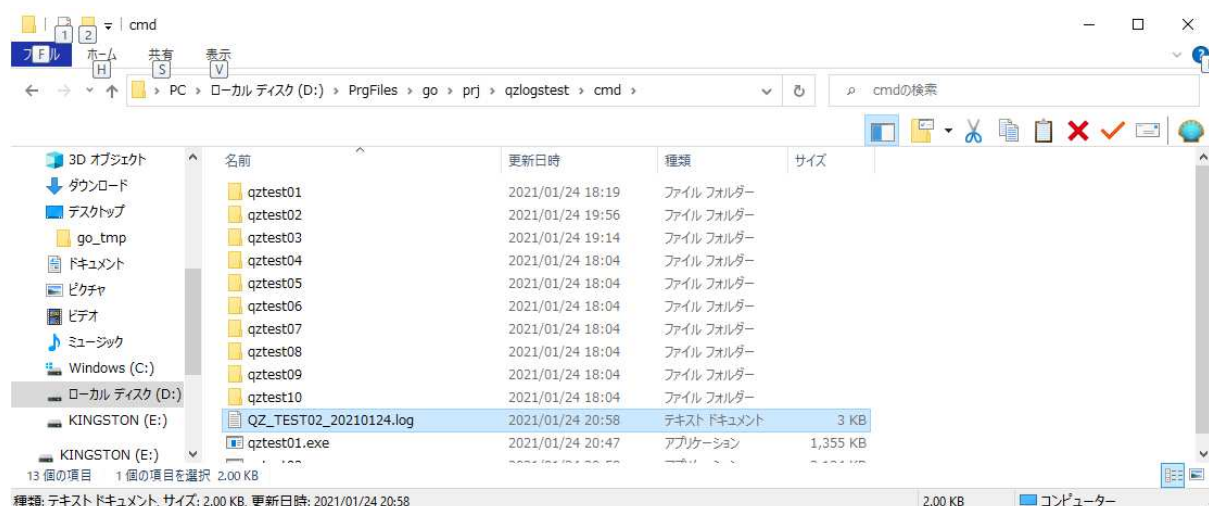
実行時にこんなエラーになります

```
Windows コマンド プロセッサ

D:\PrgFiles\go\prj\qzlogstest\cmd>go build ./qztest02/qztest02.go
D:\PrgFiles\go\prj\qzlogstest\cmd>go build ./qztest02/qztest02.go
D:\PrgFiles\go\prj\qzlogstest\cmd>qztest02
panic: Failed to load x:\PrgFiles\go\prj\qzlogstest\cmd\qztest02\lib\qzlogs\logspigo_x64.dll: The specified module could not be found.

goroutine 1 [running]:
syscall.(*LazyProc).mustFind(...)
    D:/PrgFiles/Go/src/syscall/dll_windows.go:320
syscall.(*LazyProc).Addr(...)
    D:/PrgFiles/Go/src/syscall/dll_windows.go:327
D:/PrgFiles/go/prj/qzlogstest/cmd/qztest02/lib/qzlogs.LogInit(0x8a5a2d, 0x21, 0x8a0060, 0xa, 0x8000, 0x1, 0x0, 0x0, 0x0, 0x0, 0x0, ...)
    D:/PrgFiles/Go/prj/qzlogstest/cmd/qztest02/lib/qzlogs/qzlogsUTF8.go:54 +0x18c
main.main()
    D:/PrgFiles/Go/prj/qzlogstest/cmd/qztest02/qztest02.go:12 +0x79
D:\PrgFiles\go\prj\qzlogstest\cmd>
```

正しくパスが通れば、実行後にQZ\_TEST02\_xxx.logファイルが作成されます。



メモ帳で開くと、ログが記録されています



qztest02では、新規追加された、プロファイラ関数がサポートされています。sleepしないと時間が経過しないので、サンプル内ですこしだけsleepしています。ログの先頭行の表示はwriteFlgパラメータを変更することで変えることが可能です。

## 2. 公開メソッド・マニュアル

(ATL COM, Win32/64 DLL版共に同名で実装されています)

ただし、GO版は、GOの言語仕様から、先頭が大文字になるようにラップしています

No	公開メソッド名	機能	引数	備考
1	int LogInit  ret 0...エラー ret 1....正常初期化	logsplgo_x64.dll logsplgo_x64.lib 初期化	<ol style="list-style-type: none"> <li>1. string path</li> <li>2. string prefix</li> <li>3. enum WriteFlg</li> <li>4. enum CrLfOpt</li> <li>5. enum LogSleepSw</li> <li>6. enum LogLevel</li> <li>7. enum utf8BOM</li> </ol> <p>※enumは数値型に置き換えてください</p>	<p>pathは、ログを吐き出す位置の指定、ファイル名はprefix+日付で自動設定されるため書かないこと。</p> <p>prefixは、ログのファイル名識別用のユーザー定義自由文字列。</p> <p>WriteFlgは、log()メソッドの先頭に書かれる、スレッドID,や時間、日付のフォーマット指定など。41. ※1 参照</p> <p>CrLfOptは 0...CR+LF 1...CR 2...LF の指定となる</p> <p>LogSleepSw 0...ログ出力 1...ログを出力しない</p> <p>LogLevelは 1-6の値で、出力するメソッドを制御します</p> <p>utf8 BOM は UTF8の 0...BOMをつけない 1...BOMを付ける</p>
2	LogTerm	終了処理。 logInit系と必ず対に呼び出す必要がある	なし	



N o	公開メソッド名	機能	引数	備考
3	Log	1行ログ出力  printf format パラメータ対応	1. string data  <b>注意</b> <b>_log()メソッドは、ログ録 レベル数値を 無視します。 いつでも書いてしまいま す。</b>	最大サイズは8100バイト  _が付くのは、C++テンプレート系とリンクするとコンフリクトしたため。かつ、なるべく短い名前で作成したかった。
4	LogDebug	一行ログ出力 printf format パラメータ対応  LogLevelが5 以上でログ記録	1. string data	動作仕様は_log()と同じ これ以下はLogLevelを評価する LOG_ALL = 0x00000006 の場合全てのログを記録します。  このログの出力スイッチ LOG_DEBUG =0x00000005
5	LogInfo	一行ログ出力 printf format パラメータ対応  LogLevelが4 以上でログ記録	1. string data	このログの出力スイッチ LOG_INFO = 0x00000004
6	LogWarning	一行ログ出力 printf format/ パラメータ対応  LogLevelが3 以上でログ記録	1. string data	このログの出力スイッチ LOG_WARNING =0x00000003
7	LogError	一行ログ出力 printf format パラメータ対応  LogLevelが2 以上でログ記録	1. string data	このログの出力スイッチ LOG_ERROR = 0x00000002

No	公開メソッド名	機能	引数	備考
8	LogCritical	一行ログ出力 printf format パラメータ対応 LogLevelが1 以上でログ記録	1. string data	このログの出力スイッチ LOG_CRITICAL= 0x00000001
9	LogSys ※非公開	システムが利用する一行ログ。ユーザ操作不可。		
10	LogDumpUTF8	ダンプ形式の出力	1. string data int size 2. string prefix	最大サイズは8100バイト int sizeはUNICODE文字数なので実質4050バイト上限となる。  prefixは短い文字列のダンプ識別ユーザ定義
11	LogPut  ※ UTF8文字列は、1-4バイトで表現されるため、nextLineCntがぴつたりと指定位置で改行されるとは限りません	1文字ずつシ ンプルにログ出力する	1. string data 2. int size 3. bool header 4. int nextLineCnt	最大サイズは8100バイト int sizeはUNICODE文字数なので実質4050バイト上限となる。  headerは、trueにセットすると、見やすくするための区切文字列を挿入する  nextLineCntはn文字での改行コードを挿入します。10-80の間で有効となります。
12	LogPutHex	バイトデータ 列挙  0xXXの16進 値で列挙しま す	1. string data 2. int size 3. int header 4. int nextLineCnt	最大サイズは8100バイト  headerは、trueにセットすると、見やすくするための区切文字列を挿入する  nextLineCntはn文字での改行コードを挿入します。10-80の間で有効となります。



N o	公開メソッド名	機能	引数	備考
13	W32ErrLog	w32エラー文字列を日本語でログする	1. file_fnc 2. int line	<p>file_fncは、エラーが発生した箇所の指定。どのファイルのどのメソッドか入力する</p> <p>lineは、ユーザプログラムのエラーチェックしたい行番号を入力</p>
14	W32ResetErrCode	Win32内部エラーをリセット		W32ErrLogを利用する前に呼び出します。前に、エラーがあると、そのerrorCodeを保持しているからです
15	MakeBitStringUSHORT16	USHORT16の値をビット文字列に分解する	1. USHORT x 2. string 3. prefix	<p>x 分解する符号なし16ビット</p> <p>prefix 識別文字列、UTF16文字などをセット</p> <p>注、サロゲートペアは4バイトなので9番で処理する。</p>
16	MakeBitStringUINT32	UINT32の値をビット文字列に分解する	1. ULONG x 2. string prefix	<p>x 分解する符号なし32ビット</p> <p>prefix 識別文字列 UTF32文字などをセット</p>

No	公開メソッド名	機能	引数	備考
17	MakeBitStringUnicodeScalarValue	U+xxxxのユニコードスカラー値を、UTF16, UTF8のビット文字列に分解する	<ol style="list-style-type: none"> <li>1. ULONG scalar</li> <li>2. string prefix</li> </ol>	scalar 分解するU+XXXXユニコード、スカラー値  prefix 識別文字列、文字などをセット
18	MsgBox	デバッグ用メッセージボックス簡易表示	<ol style="list-style-type: none"> <li>1. string ダイアログ内文字列</li> <li>2. string タイトル文字列</li> </ol>	
20	int MsgBoxOkCancel	デバッグ用メッセージボックス簡易表示  [OK] [CANCEL] 選択オプション処理可能	<ol style="list-style-type: none"> <li>3. string ダイアログ内文字列</li> <li>4. string タイトル文字列</li> </ol>	ret ... 1 の場合[OK]が押された  ret ...2の場合 [CANCEL]が押された  ret...0の場合 バッファサイズオーバー
21	ProfileTimeStart v 0.5.1.1 追加機能	マイクロ秒のプロファイル計測開始関数	<ol style="list-style-type: none"> <li>1. string prefix</li> </ol>	計測終了時に、prefix文字列が表示されます <b>注意 この計測関数は、入れ子で使うことができません。入れ子状態で使うと、結果は未定義になります。例としては、関数の開始でProfileTimeStart関数の終了前にProfileTimeStopを使ってください。この場合でも、関数途中のエラーリターンなどがあると計測できません。</b>
22	ProfileTimeStop v 0.5.1.1 追加機能	マイクロ秒プロファイルの計測終了。計測時間をオープンしているログに記録します	なし	

※ C言語とGO言語間のポインタ評価ができない理由と安全性から、W32\_err\_fmtMsg関数は除外されています。

## 2. Write Flag パラメータオプション説明

No.	C/C++定義名	値(int)	説明
1	_NO_DATETIME	0x0	_log()の行頭になにも出力しない
2	_START_MILLISEC	0x1	OS起動時からの時間(ミリ秒)を出力
3	_TIME_SEC	0x2	PC時計の秒を出力
4	_TIME_SEC_MILI	0x3	PC時計の秒+ミリ秒を出力
5	_TIME_MIN	0x4	PC時計の分を出力
6	_TIME_MIN_MILI	0x5	PC時計の分+ミリ秒を出力
7	_TIME_MIN_SEC	0x6	PC時計の分+秒を出力
8	_TIME_MIN_SEC_MILI	0x7	PC時計の分+秒+ミリ秒を出力
9	_TIME_HOUR	0x8	PC時計の時間(Hour)のみを出力
10	_TIME_HOUR_MILI	0x9	PC時計の時間(Hour)+ミリ秒を出力
11	_DAY_DAY	0x10	PCカレンダーの日付のみ出力
12	_DAY_MON	0x20	PCカレンダーの月のみ出力
13	_DAY_YEAR	0x40	PCカレンダーの年のみ出力
14	_DAY_YEAR_MON_DAY	0x80	PCカレンダーの年月日を出力
15	_YYMMDD	0x100	PCカレンダーの日付をYYMMDDで出力
16	_YYMMDD_SLA	0x200	PCカレンダーの日付をYY/MM/DDで出力
17	_YYMMDD_HIF	0x400	PCカレンダーの日付をYY-MM-DDで出力
18	_YYMMDD_TIME	0x800	PCカレンダーの日付をYYMMDD+時間を出力
19	_YYMMDD_TIME_SLA	0x1000	PCカレンダーの日付をYY/MM/DD+時間を出力
20	_YYMMDD_TIME_HIF	0x2000	PCカレンダーの日付をYY-MM-DD+時間を出力
21	_YYMMDD_TIME_MILI	0x4000	PCカレンダーの日付をYYMMDD+時間+ミリ秒を出力
22	_YYMMDD_TIME_MILI_SEC	0x8000	PCカレンダーの日付をYY/MM/DD+時間+ミリ秒を出力

### 3. デバッグ時のQzlogsの使い方のヒントなど

個人的に、VC C/C++プロジェクトをデバッグする際、私はいつもReleaseモードで書き始めます。ああ、これはGo Langでした。ログに吐き出せば値が何か? どこまで実行されているかすぐわかるからです。

以下の疑似コードは、どこまで動作しているかの調査コード例です。  
例えばログに 2が残らなければ、最初のprintfで問題があると判断できるわけです。

```
qzlogs.Log("1")
```

```
fmt.Println(...
```

```
qzlogs.Log("2")
```

```
fmt.Println(...
```

```
qzlogs.Log("3")
```

```
fmt.Println(...
```

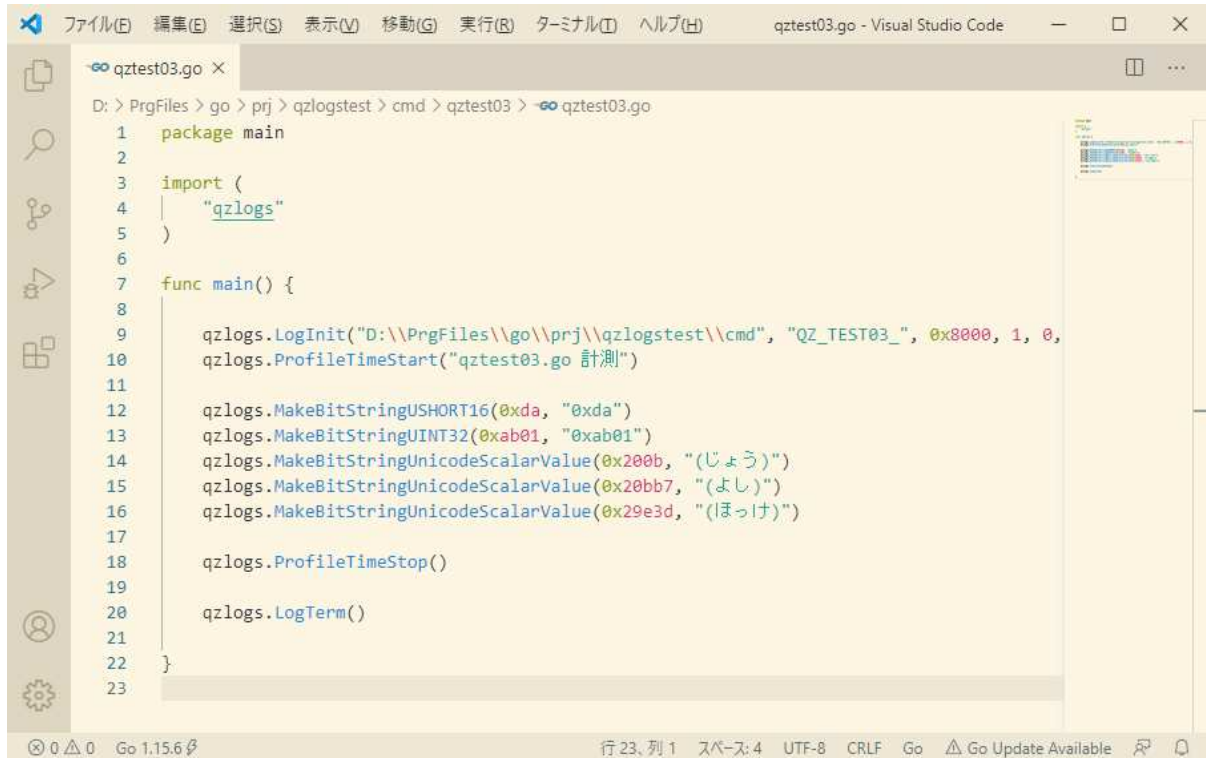
Qzlogsライブラリを使い、GO Langプログラミングをお楽しみください。

## 4. 実装例など

### 5.1. ビット文字列表示メソッド

**MakeBitStringUSHORT16(), MakeBitStringUINT32(),  
MakeBitStringUnicodeScalarValue()**

サンプル qztest03.go



```
1 package main
2
3 import (
4     "qzlogs"
5 )
6
7 func main() {
8
9     qzlogs.LogInit("D:\\PrgFiles\\go\\prj\\qzlogstest\\cmd", "QZ_TEST03_", 0x8000, 1, 0,
10     qzlogs.ProfileTimeStart("qztest03.go 計測"))
11
12     qzlogs.MakeBitStringUSHORT16(0xda, "0xda")
13     qzlogs.MakeBitStringUINT32(0xab01, "0xab01")
14     qzlogs.MakeBitStringUnicodeScalarValue(0x200b, "(じょう)")
15     qzlogs.MakeBitStringUnicodeScalarValue(0x20bb7, "(よし)")
16     qzlogs.MakeBitStringUnicodeScalarValue(0x29e3d, "(ほっけ)")
17
18     qzlogs.ProfileTimeStop()
19
20     qzlogs.LogTerm()
21 }
22
23
```

書かれた、ログのイメージ。

```

QZ_TEST03_20210125.log - メモ帳
ファイル(E) 編集(E) 書式(O) 表示(V) ヘルプ(H)
2021/01/25-14:06:20:0[--- [QZ_TEST03_] start Log={D:\PrgFiles\go\prj\qzlogstest\cmd\QZ_TEST03_20210125.log} ---]
+-----+
| 0 | 1 | [0xda]
+-----+
| da | 00 | HEX=0x00da
+-----+
|11011010|00000000| UTF16_LE Bit Pattern!
+-----+
+-----+
| 0 | 1 | 2 | 3 | []
+-----+
| 01 | ab | 00 | 00 | HEX=0x0000ab01
+-----+
|00000001|10101011|00000000|00000000| UTF32_LE Bit Pattern!
+-----+
+-----+
| * | 0 | 1 | 2 | [(じょう)]
+-----+
|*****| 0x00 | 0x20 | 0x0b | SCALAR-HEX(UINT32)=0x0000200b
+-----+
|32 24| 21 16| 8| 0| [(じょう)]
+-----+
|*****|00000000|00100000|00001011| UNICODE(SCALAR) Bit Pattern!
+-----+
| * | 0xe2 | 0x80 | 0x8b | DECODE-HEX(UINT32)=0x008b80e2
+-----+
|32 24| 16| 8| 0| [(じょう)]
+-----+
|*****|11100010|10000000|10001011| UTF8 Decode(3 bytes char)!
+-----+
+-----+
| 0 | 1 | 2 | 3 | [(よし)]
+-----+
|*****| 0x02 | 0x0b | 0xb7 | SCALAR-HEX(UINT32)=0x00020bb7
+-----+
|32 24| 21 16| 8| 0| [(よし)]
+-----+
|*****|***00010|00001011|10110111| UNICODE(SCALAR) Bit Pattern!
+-----+
| 0xf0 | 0xa0 | 0xae | 0xb7 | DECODE-HEX(UINT32)=0xb7aea0f0
+-----+
|32 24| 16| 8| 0| [(よし)]
+-----+
|11110000|10100000|10101110|10110111| UTF8 Decode(4 bytes char)!
+-----+
+-----+
| 0 | 1 | 2 | 3 | [(ほっけ)]
+-----+

```

あまり役にたたないかも知れませんが、ビット・イメージをすぐにダンプできますので、なにかの時にご利用ください。

## 5.2. Win32エラーコードとエラー文字のロギング

Windows OS内で発生した、W32エラー文字列を取得します。

これは、使うタイミングが難しいと思いますが、なにかのときに役に立ててください。


qzttest05.go

```
ファイル(F) 編集(E) 選択(S) 表示(V) 移動(G) 実行(R) ターミナル(T) ヘルプ(H) qztest05.go - Visual Studio Code
qztest05.go x
D:\> PrgFiles > go > prj > qzlogstest > cmd > qztest05 > -o qztest05.go > main
1 package main
2
3 import (
4     "fmt"
5     "qzlogs"
6     "runtime"
7 )
8
9 func main() {
10
11     qzlogs.LogInit("D:\\PrgFiles\\go\\prj\\qzlogstest\\cmd", "QZ_TEST05_", 1, 1, 0, 0, 1)
12     qzlogs.ProfileTimeStart("qztest05.go 計測")
13
14     _, file, line, ok := runtime.Caller(0)
15     if !ok {
16         fmt.Println("スタックトレースの取得失敗")
17         // 後始末
18         qzlogs.ProfileTimeStop()
19         qzlogs.LogTerm()
20         return
21     }
22     msg := fmt.Sprintf("%s main()", file)
23     qzlogs.W32ErrLog(msg, line)
24
25     // この文字列は、サロゲート文字対応のエディタでないと化けます
26     strDat := "g帛梳煤燐燐f復殘a碯矜蠶繭艾繭繭猱繭燐雞一厠自へ何俾僂僂僂僂僂爰浴几刳刳刳刳刳斗車々及味薯啣啣"
27     for i := 0; i < 2000; i++ {
28         qzlogs.Log(strDat)
29     }
30
31     qzlogs.ProfileTimeStop()
32     qzlogs.LogTerm()
33
34 }
35
```

## 5.3. メソッドの 出入口にログを書いておくテクニック

開発初期で、実装コードが不安定なとき、すごく役に立ちます。どこまで動作しているか一目瞭然です。

サンプル qztest02.go



```
9
10 func main() {
11
12     qzlogs.LogInit("D:\\PrgFiles\\go\\prj\\qzlogstest\\cmd", "QZ_TEST02_", 0x8000, 1, 0, 0, 1)
13     qzlogs.ProfileTimeStart("qztest02.go 計測")
14
15     hoge()
16     saitama()
17     musako()
18     omiya()
19     chiba()
20     azabu()
21     kawaguchi()
22     mount()
23
24     qzlogs.ProfileTimeStop()
25
26     qzlogs.LogTerm()
27
28 }
29
30 func hoge() {
31     qzlogs.Log("hoge() In!")
32
33     _, file, line, ok := runtime.Caller(0)
34     if !ok {
35         fmt.Println("スタックトレースの取得失敗")
36         return
37     }
38     msg := fmt.Sprintf("file=%s 関数 hoge() line=%d でエラー発生(嘘ですが)", file, line)
39     qzlogs.Log(msg)
40
41     qzlogs.Log("hoge() Out!")
42 }
43
```

Sample2.py ログの結果。



```
QZ_TEST02_20210125.log - メモ帳
ファイル(E) 編集(E) 書式(O) 表示(V) ヘルプ(H)
2021/01/25-14:11:23:0[--- [QZ_TEST02_] start Log={D:\PrgFiles\go\prj\qzlogstest\cmd\QZ_TEST02_20210125.log} ---]
2021/01/25-14:11:23:0[hoge() In!]
2021/01/25-14:11:23:0[file=D:\PrgFiles\go\prj\qzlogstest\cmd\qztest02\qztest02.go 関数 hoge() line=34 でエラー発生(嘘ですが)]
2021/01/25-14:11:23:0[hoge() Out!]
2021/01/25-14:11:23:0[saitama() In!]
2021/01/25-14:11:23:0[さいたまの浦和は凄いですよ。皆年収2千万の人ばかりらしいです]
2021/01/25-14:11:23:0[saitama() Out!]
2021/01/25-14:11:23:0[musako() In!]
2021/01/25-14:11:23:0[ムサコは凄いですよ。地上200m以上に住んでいる人ばかりです。でも駅から近いのにホームまで時間がかかります]
2021/01/25-14:11:23:0[musako() Out!]
2021/01/25-14:11:23:0[omiya() In!]
2021/01/25-14:11:23:0[大宮こそキングオブさいたま！新都心はまさに未来都市。でも銀座からタクシーで帰ると奥さんに叱られます]
2021/01/25-14:11:23:0[omiya() Out!]
2021/01/25-14:11:23:0[chiba() In!]
2021/01/25-14:11:23:0[おおっと、千葉こそ最強。千葉なのに『東京ディズニーランド』があるぜ。海も無いさいたまにだけは負けないぜ]
2021/01/25-14:11:23:0[chiba() Out!]
2021/01/25-14:11:23:0[azabu() In!]
2021/01/25-14:11:23:0[アデクシ、麻布から出たことはありませんことよ。おほほほほほほ〜]
2021/01/25-14:11:23:0[azabu() Out!]
2021/01/25-14:11:23:0[kawaguchi() In!]
2021/01/25-14:11:23:0[あふっ、見栄さえはらなきゃ川口最強。荒川こえるだけで赤羽だからね。]
2021/01/25-14:11:23:0[kawaguchi() Out!]
2021/01/25-14:11:23:0[mount() In!]
2021/01/25-14:11:23:0[住んでるところでマウントするのやめて(*'ω'*) ここでわざとsleepしますね]
2021/01/25-14:11:23:0[開発で疲れて、わるふざけが過ぎました。ごめんなさい Cool Bar!]]
2021/01/25-14:11:23:0[mount() Out!]
2021/01/25-14:11:23:0[qztest02.go 計測] 経過時間=0.202000[ms]]
2021/01/25-14:11:23:0[--- [QZ_TEST02_] end Log. ---]
```




ログが記録されているところまでは動作しているということなので、ログが書かれていないメソッドのバグ取りを行います。

上記例で、omiya()の内部でエラー発生の場合は、omiya() Out! が記録されなくなります。  
omiya()関数内部のバグ取りを行えば良いということです。

#### 5.4. logSleepSw = 1 の簡易ログ停止機能

Qzlogsは、ログが必要なくなった時点で、ログ記録を停止できます。内部的な仕様として、各メソッドの引数があれば、それをスタックに積んでからのリターンとなります。これが、ログ停止機能のオーバーヘッドとなります。このオーバーヘッドが気になる場合、または、重大なエラー箇所だけはログ記録を残したい場合などでは、各言語のコンパイル制御文で対処してください。

logInitの5番目のパラメータを1にすると logSleepSw がONになり、ログ記録を停止します。



The screenshot shows a Notepad window titled "QZ\_TEST07\_20210125.log - メモ帳". The menu bar includes "ファイル(F)", "編集(E)", "書式(O)", "表示(V)", and "ヘルプ(H)". The text content of the log is: "##### LOG\_SLEEP Switch ON! No save logs... logInit() No.5 Param 0 Set to start Log! #####". The status bar at the bottom indicates "1行, 1列", "100%", "Windows (CRLF)", and "UTF-8 (BOM 付き)".

qztest07.qo

```
D:\> PrgFiles > go > prj > qzlogstest > cmd > qztest07 > qztest07.go > main
```

```
1 package main
2
3 import (
4     "qzlogs"
5 )
6
7 func main() {
8
9     // ログ・出力する初期化
10    //qzlogs.LogInit("D:\\PrgFiles\\gol\\prj\\qzlogstest\\cmd", "QZ_TEST07_", 0x8000, 1, 0, 0, 1)
11
12    // ログ・スリープする初期化
13    qzlogs.LogInit("D:\\PrgFiles\\gol\\prj\\qzlogstest\\cmd", "QZ_TEST07_", 0x8000, 1, 1, 0, 1)
14
15    qzlogs.ProfileTimeStart("qztest07.go 計測")
16
17    // この文字列は、サロゲート文字対応のエディタでないと化けます
18    strDat := "g昚梲燄齋焚f複硯a穉矜蠟繡支蔣藏愷綰院鷄一衆自へ伺憊僂僂僂僂僂僂浴几劔劓辺勳斗車ㇿ𐤅𐤆𐤇𐤈𐤉𐤊𐤋𐤌𐤍𐤎𐤏𐤐𐤑𐤒𐤓𐤔𐤕𐤖𐤗𐤘𐤙𐤚𐤛𐤜𐤝𐤞𐤟𐤠𐤡𐤢𐤣𐤤𐤥𐤦𐤧𐤨𐤩𐪀𐪁𐪂𐪃𐪄𐪅𐪆𐪇𐪈𐪉𐪊𐪋𐪌𐪍𐪎𐪏𐪐𐪑𐪒𐪓𐪔𐪕𐪖𐪗𐪘𐪙𐪚𐪛𐪜𐪝𐪞𐪟𐪠𐪡𐪢𐪣𐪤𐪥𐪦𐪧𐪨𐪩𐫀𐫁𐫂𐫃𐫄𐫅𐫆𐫇𐫈𐫉𐫊𐫋𐫌𐫍𐫎𐫏𐫐𐫑𐫒𐫓𐫔𐫕𐫖𐫗𐫘𐫙𐫚𐫛𐫜𐫝𐫞𐫟𐫠𐫡𐫢𐫣𐫤𐫦𐫥𐫧𐫨𐫩𐬀𐬁𐬂𐬃𐬄𐬅𐬆𐬇𐬈𐬉𐬊𐬋𐬌𐬍𐬎𐬏𐬐𐬑𐬒𐬓𐬔𐬕𐬖𐬗𐬘𐬙𐬚𐬛𐬜𐬝𐬞𐬟𐬠𐬡𐬢𐬣𐬤𐬥𐬦𐬧𐬨𐬩𐬪𐬫𐬬𐬭𐬮𐬯𐬰𐬱𐬲𐬳𐬴𐬵𐬶𐬷𐬸𐬹𐬺𐬻𐬼𐬽𐬾𐬿𐍀𐍁𐍂𐍃𐍄𐍅𐍆𐍇𐍈𐍉𐍊𐍋𐍌𐍍𐍎𐍏𐍐𐍑𐍒𐍓𐍔𐍕𐍖𐍗𐍘𐍙𐍚𐍛𐍜𐍝𐍞𐍟𐍠𐍡𐍢𐍣𐍤𐍥𐍦𐍧𐍨𐍩𐍪𐍫𐍬𐍭𐍮𐍯𐍰𐍱𐍲𐍳𐍴𐍵𐍶𐍷𐍸𐍹𐍺𐍻𐍼𐍽𐍾𐍿𐎀𐎁𐎂𐎃𐎄𐎅𐎆𐎇𐎈𐎉𐎊𐎋𐎌𐎍𐎎𐎏𐎐𐎑𐎒𐎓𐎔𐎕𐎖𐎗𐎘𐎙𐎚𐎛𐎜𐎝𐎞𐎟𐎠𐎡𐎢𐎣𐎤𐎥𐎦𐎧𐎨𐎩𐎪𐎫𐎬𐎭𐎮𐎯𐎰𐎱𐎲𐎳𐎴𐎵𐎶𐎷𐎸𐎹𐎺𐎻𐎼𐎽𐎾𐎿�0�1�2�3�4�5�6�7�8�9𐐀𐐁𐐂𐐃𐐄𐐅𐐆𐐇𐐈𐐉𐐊𐐋𐐌𐐍𐐎𐐏𐐐𐐑𐐒𐐓𐐔𐐕𐐖𐐗𐐘𐐙𐐚𐐛𐐜𐐝𐐞𐐟𐐠𐐡𐐢𐐣𐐤𐐥𐐦𐐧𐐨𐐩𐐪𐐫𐐬𐐭𐐮𐐯𐐰𐐱𐐲𐐳𐐴𐐵𐐶𐐷𐐸𐐹𐐺𐐻𐐼𐐽𐐾𐐿𐑀𐑁𐑂𐑃𐑄𐑅𐑆𐑇𐑈𐑉𐑊𐑋𐑌𐑍𐑎𐑏𐑐𐑐𐑑𐑒𐑓𐑔𐑕𐑖𐑗𐑘𐑙𐑚𐑛𐑜𐑝𐑞𐑟𐑠𐑡𐑢𐑣𐑤𐑥𐑦𐑧𐑨𐑩𐑪𐑫𐑬𐑭𐑮𐑯𐑰𐑱𐑲𐑳𐑴𐑵𐑶𐑷𐑸𐑹𐑺𐑻𐑼𐑽𐑾𐑿𐒀𐒁𐒂𐒃𐒄𐒅𐒆𐒇𐒈𐒉𐒊𐒋𐒌𐒍𐒎𐒏𐒐𐒑𐒒𐒓𐒔𐒕𐒖𐒗𐒘𐒙𐒚𐒛𐒜𐒝𐒞𐒟𐒠𐒡𐒢𐒣𐒤𐒥𐒦𐒧𐒨𐒩𐒪𐒫𐒬𐒭𐒮𐒯𐒰𐒱𐒲𐒳𐒴𐒵𐒶𐒷𐒸𐒹𐓀𐓁𐓂𐓃𐓄𐓅𐓆𐓇𐓈𐓉𐓊𐓋𐓌𐓍𐓎𐓏𐓐𐓑𐓒𐓓𐓔𐓕𐓖𐓗𐓘𐓙𐓚𐓛𐓜𐓝𐓞𐓟𐓠𐓡𐓢𐓣𐓤𐓥𐓦𐓧𐓨𐓩𐓪𐓫𐓬𐓭𐓮𐓯𐓰𐓱𐓲𐓳𐓴𐓵𐓶𐓷𐓸𐓹𐔀𐔁𐔂𐔃𐔄𐔅𐔆𐔇𐔈𐔉𐔊𐔋𐔌𐔍𐔎𐔏𐔐𐔑𐔒𐔓𐔔𐔕𐔖𐔗𐔘𐔙𐔚𐔛𐔜𐔝𐔞𐔟𐔠𐔡𐔢𐔣𐔤𐔥𐔦𐔧𐔨𐔩𐔪𐔫𐔬𐔭𐔮𐔯𐔰𐔱𐔲𐔳𐔴𐔵𐔶𐔷𐔸𐔹𐔺𐔻𐔼𐔽𐔾𐔿𐕀𐕁𐕂𐕃𐕄𐕅𐕆𐕇𐕈𐕉𐕊𐕋𐕌𐕍𐕎𐕏𐕐𐕑𐕒𐕓𐕔𐕕𐕖𐕗𐕘𐕙𐕚𐕛𐕜𐕝𐕞𐕟𐕠𐕡𐕢𐕣𐕤𐕥𐕦𐕧𐕨𐕩𐕪𐕫𐕬𐕭𐕮𐕯𐕰𐕱𐕲𐕳𐕴𐕵𐕶𐕷𐕸𐕹𐕺𐕻𐕼𐕽𐕾𐕿𐖀𐖁𐖂𐖃𐖄𐖅𐖆𐖇𐖈𐖉𐖊𐖋𐖌𐖍𐖎𐖏𐖐𐖑𐖒𐖓𐖔𐖕𐖖𐖗𐖘𐖙𐖚𐖛𐖜𐖝𐖞𐖟𐖠𐖡𐖢𐖣𐖤𐖥𐖦𐖧𐖨𐖩𐖪𐖫𐖬𐖭𐖮𐖯𐖰𐖱𐖲𐖳𐖴𐖵𐖶𐖷𐖸𐖹𐖺𐖻𐖼𐖽𐖾𐖿𐗀𐗁𐗂𐗃𐗄𐗅𐗆𐗇𐗈𐗉𐗊𐗋𐗌𐗍𐗎𐗏𐗐𐗑𐗒𐗓𐗔𐗕𐗖𐗗𐗘𐗙𐗚𐗛𐗜𐗝𐗞𐗟𐗠𐗡𐗢𐗣𐗤𐗥𐗦𐗧𐗨𐗩𐗪𐗫𐗬𐗭𐗮𐗯𐗰𐗱𐗲𐗳𐗴𐗵𐗶𐗷𐗸𐗹𐗺𐗻𐗼𐗽𐗾𐗿𐘀𐘁𐘂𐘃𐘄𐘅𐘆𐘇𐘈𐘉𐘊𐘋𐘌𐘍𐘎𐘏𐘐𐘑𐒀𐒁𐒂𐒃𐒄𐒅𐒆𐒇𐒈𐒉𐒊𐒋𐒌𐒍𐒎𐒏𐒐𐒑𐒒𐒓𐒔𐒕𐒖𐒗𐒘𐒙𐒚𐒛𐒜𐒝𐒞𐒟𐒠𐒡𐒢𐒣𐒤𐒥𐒦𐒧𐒨𐒩𐒪𐒫𐒬𐒭𐒮𐒯𐒰𐒱𐒲𐒳𐒴𐒵𐒶𐒷𐒸𐒹𐓀𐓁𐓂𐓃𐓄𐓅𐓆𐓇𐓈𐓉𐓊𐓋𐓌𐓍𐓎𐓏𐓐𐓑𐓒𐓓
```

[illegible]

## 5.5. UTF8 文字列ダンプ時の禁則処理の説明

UTF8文字列は、1バイトから4バイトまでを使います。なので、ダンプ表示の際に、先頭が、ダンプ中心直前と、最後のバイト位置に3バイト以上の表示はできません。その際に、

先頭位置の識別として **ハ** キャラクタを表示します。これが、書けなかったマルチバイト文字の先頭位置の印となります。その他、4バイトの(UTF16)サロゲート文字などは、漢字を表示したあとで、**..** 半角のドットが二つ追加されます。3バイトの漢字は漢字の表示のあとでドットが一つ追加されます。

2バイト文字は、文字表示のあとでドットがひとつ追加されます。

上段 サロゲート漢字ダンプ例

中断 3バイト日本語ひらがなダンプ例

下段 2バイト記号文字ダンプ例

(このログ内容はPython3 バージョンのもので)

```
PY_SMP6_20201220.log - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)
2020-12-20 12:33:13:[--- [PY_SMP6_] start Log={C:\dev\VS2017\Python\PY_SMP6_20201220.log} ---]
2020-12-20 12:33:13:[aaa丈土壕b嬢叱岨嶮梟梳様齋齋復碇碇籽璽]

addr={00000236158CF310} prefix={サロゲート} charSet={UTF8N(BomSet)} size(BYTE)=72
00 01 02 03 04 05 06 07 |08 09 0A 0B 0C 0D 0E 0F |01234567|89ABCDEF| Dec Hex
-----
81 81 81 f0 a0 80 8b f0 |a1 88 bd f0 a1 91 ae 82 |aaa丈土壕..b| 000016 0x0010
f0 a1 a2 bd f0 a0 ae 91 |f0 a1 9a b4 f0 a1 b8 b4 |嬢..叱..岨..嶮..| 000032 0x0020
f0 a3 87 84 f0 a3 97 84 |f0 a3 9c bf f0 a3 9d a3 |梟..梳..様..齋..| 000048 0x0030
f0 a3 b3 be f0 a4 9f b1 |f0 a5 92 8e f0 a5 94 8e |齋..復..碇..碇..| 000064 0x0040
f0 a5 9d b1 f0 a5 a7 84 |籽..璽..| 000072 0x0048
-----

addr={00000236158CF310} prefix={漢字} charSet={UTF8N(BomSet)} size(BYTE)=45
00 01 02 03 04 05 06 07 |08 09 0A 0B 0C 0D 0E 0F |01234567|89ABCDEF| Dec Hex
-----
e3 81 82 e3 81 84 e3 81 |86 e3 81 88 e3 81 8a e3 |あ..い..う..え..お..| 000016 0x0010
81 8b e3 81 8d e3 81 8f |e3 81 91 e3 81 93 e3 81 |かぎ..く..け..こ..さ..| 000032 0x0020
95 e3 81 97 e3 81 99 e3 |81 9b e3 81 9d |..し..す..せ..そ..| 000045 0x002d
-----

addr={00000236158CF310} prefix={UTF8 2バイト} charSet={UTF8N(BomSet)} size(BYTE)=161
00 01 02 03 04 05 06 07 |08 09 0A 0B 0C 0D 0E 0F |01234567|89ABCDEF| Dec Hex
-----
81 c3 80 c3 81 c3 82 c3 |83 c3 84 c3 85 c3 86 c3 |aÀ.Á.Â.Ã|.Ä.Å.Æ.Ç| 000016 0x0010
8f c3 88 c3 89 c3 8a c3 |8b c3 8c c3 8d c3 8e c3 |.É.Ê.Ë.Ê|.Ï.Î.Ï.Î| 000032 0x0020
87 c3 90 c3 91 c3 92 c3 |93 c3 94 c3 95 c3 96 c3 |.Ë.Ë.Ë.Ë|.Ë.Ë.Ë.Ë| 000048 0x0030
97 c3 98 c3 99 c3 9a c3 |9b c3 9c c3 9d c3 9e c3 |.Ë.Ë.Ë.Ë|.Ë.Ë.Ë.Ë| 000064 0x0040
9f c3 a0 c3 a1 c3 a2 c3 |a3 c3 a4 c3 a5 c3 a6 c3 |.ä.ä.ä.ä|.ä.ä.ä.ä| 000080 0x0050
a7 c3 a8 c3 a9 c3 aa c3 |ab c3 ac c3 ad c3 ae c3 |.è.è.è.è|.è.è.è.è| 000096 0x0060
af c3 b0 c3 b1 c3 b2 c3 |b3 c3 b4 c3 b5 c3 b6 c3 |.ö.ö.ö.ö|.ö.ö.ö.ö| 000112 0x0070
b7 c3 b8 c3 b9 c3 ba c3 |bb c3 bc c3 bd c3 be c3 |.ø.ø.ø.ø|.ø.ø.ø.ø| 000128 0x0080
bf c4 80 c4 81 c4 82 c4 |83 c4 84 c4 85 c4 86 c4 |.À.ä.Ä.ä|.Ä.ä.ä.ä| 000144 0x0090
87 c4 88 c4 89 c4 8a c4 |8b c4 8c c4 8d c4 8e c4 |.â.â.â.â|.â.â.â.â| 000160 0x00a0
8f |. | 000181 0x00a1
-----

2020-12-20 12:33:13:[----- [PY_SMP6_] end Log. -----]
```

## 6. サンプルプログラム

6.1. qztest01	導入サンプル
6.2. qztest02	ローカルlibパッケージ利用、メソッド・出入り口例サンプル
6.3. qztest03	ビット・ストリング表示イメージサンプル
6.4. qztest04	LogDumpUTF8 1-4バイト文字のダンプサンプル
6.5. qztest05	W32エラーメッセージログ記録サンプル
6.6. qztest06	LogPut, LogPutHexサンプル
6.7. qztest07	ログ・スリープ・スイッチ使い方サンプル
7.8. qztest08	ログ、出力レベル別の使い方サンプル
7.9. qztest09	MsgBox, MsgBoxOkCancel利用例サンプル
7.10. qztest10	LogDumpUTF8 もろもろ

## 7. 仕様

### 7.1. ダンプ文字列最大長 8190バイト

(UTF8は可変 1-4バイトで文字を表現しますので、ダンプしたデータに依存します)

### 7.2. 一行文字最大長 8190バイト

なお prefix文字は、1024バイト上限です

### 7.3. 制限を超えるバイト数のロギング

繰り返し文などで、最大長を超えない固定サイズを処理してください

### 7.4. スレッド

Goが利用しているメインスレッドをそのまま利用

マルチ・スレッドからのログメソッド呼び出しの動作は不定です

今後、マルチ・スレッドログサーバーの開発も視野にいております

本ソフトウェアは、あくまでも言語学習用と捉えてください

### 7.5. フォルダの自動作成

LogInitで指定した、ログ記録パスが存在しない場合自動作成。ただし物理的にその位置が存在しない場合はエラー

### 7.6. ログ・ローテート

シンプル版ではサポートされていません

### 7.7. ログ・ディチェンジ

シンプル版ではサポートされていません

### 7.8. 指定サイズ超過ローテート

シンプル版ではサポートされていません

## qzlogtest01.go

```

1  package main
2
3  import (
4      "qzlogs"
5      "strconv"
6  )
7
8  func main() {
9
10     qzlogs.LogInit("D:\\work\\qzlogtest\\cmd", // ログファイル出力先フォルダ
11         "QZ_TEST01_", // ログファイル名識別用prefix
12         1, // writeFlg 1行ログ、先頭の日付時間などのOption
13         1, // CrLfOpt 0...CR+LF 1...LF 2...CR
14         0, // LogSleepSw 0...ログ出力 1...ログ・スリープ(記録中止)
15         0, // LogLevel 1-5 レベルに合致したログメソッドのみ出力する
16         1) // utf8bom 0...UTF8 Bomなし 1...UTF8 Bomあり
17
18     qzlogs.ProfileTimeStart("qzlogtest01")
19
20     // パフォーマンス・テスト 2万行 Log 書き込み
21     //[SYSLOG][--INFO--] [qzlogtest01] 経過時間=0.077000[秒]
22     //tid=20024:0.000000[ms][--- [QZ_TEST01_] end Log. ---]
23     // Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz 3.00 GHz
24     // 最初のCPUテストはPython版よりも速く動作した
25     //
26     //[SYSLOG][--INFO--] [qzlogtest01] 経過時間=0.173000[秒]
27     //tid=11552:0.000000[秒][--- [QZ_TEST01_] end Log. ---]
28     //[SYSLOG][--INFO--] [qzlogtest01] 経過時間=0.173000[秒]
29     //tid=11552:0.000000[秒][--- [QZ_TEST01_] end Log. ---]
30     // なんと、Python3版のほうがパフォーマンスが良い' strconv.Itoaが遅いのか
31     // 以下、i7 同マシンのPython3 バージョン計測
32     //#[SYSLOG][--INFO--] [Sample.py] 経過時間=0.118000[秒]
33     //#2021-01-29 12:11:06:[----- [PY_SMP_] end Log. -----]
34     // # i7-4770 CPU @ 3.40GHz 3.40 GHz (第4世代) 8スレッド だと0.118000秒でした、そんなに変わらないですね
35     //
36     //[SYSLOG][--INFO--] [qzlogtest01] 経過時間=0.210000[秒]
37     //tid=8252:0.000000[秒][--- [QZ_TEST01_] end Log. ---]
38     // Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz 2.50 GHz 古いNote PC HDD搭載機
39     for i := 0; i < 20000; i++ {
40         qzlogs.Log("hoge! = " + strconv.Itoa(i))
41     }
42
43     qzlogs.ProfileTimeStop()
44     qzlogs.LogTerm()
45
46 }
47

```