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、これよりも少なくる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

ドキュメント　バージョン　0.5.0.0　2021/01/15

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)

※　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。

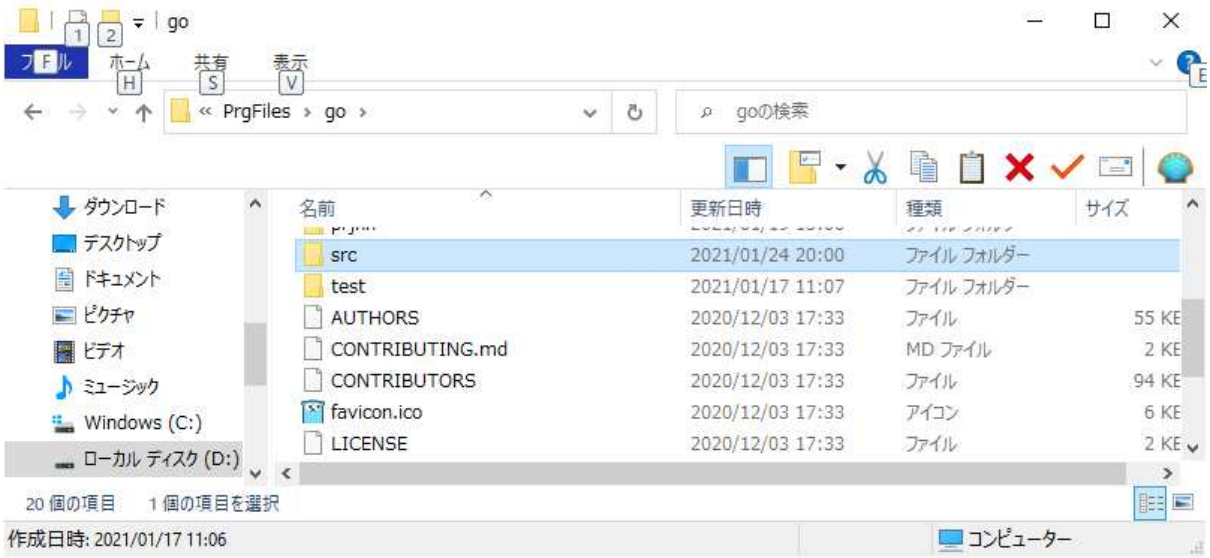
1. Qzlogsのセットアップ

1.1. すべてのプロジェクトから参照するインストール

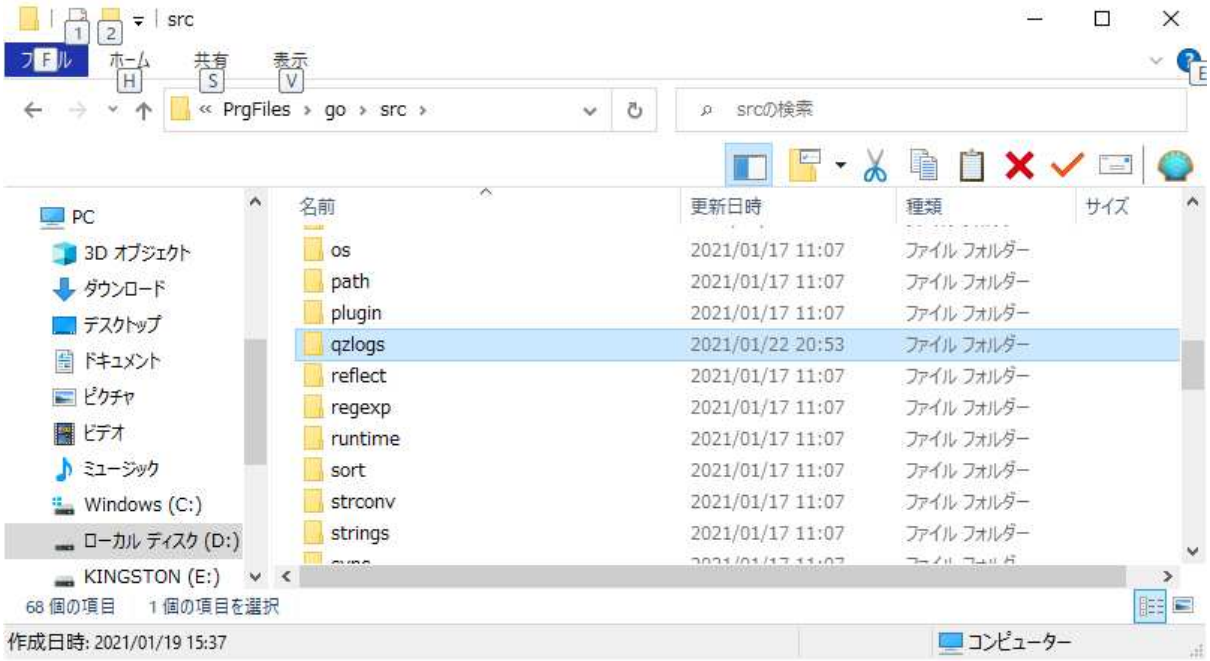
Goのインストール・ルートに移動します
GOはインストール、動作確認済みの前提です、インストールについては、Go Langの公認サイトをご確認ください)環境にパッケージフォルダをコピーするだけですが・・・

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

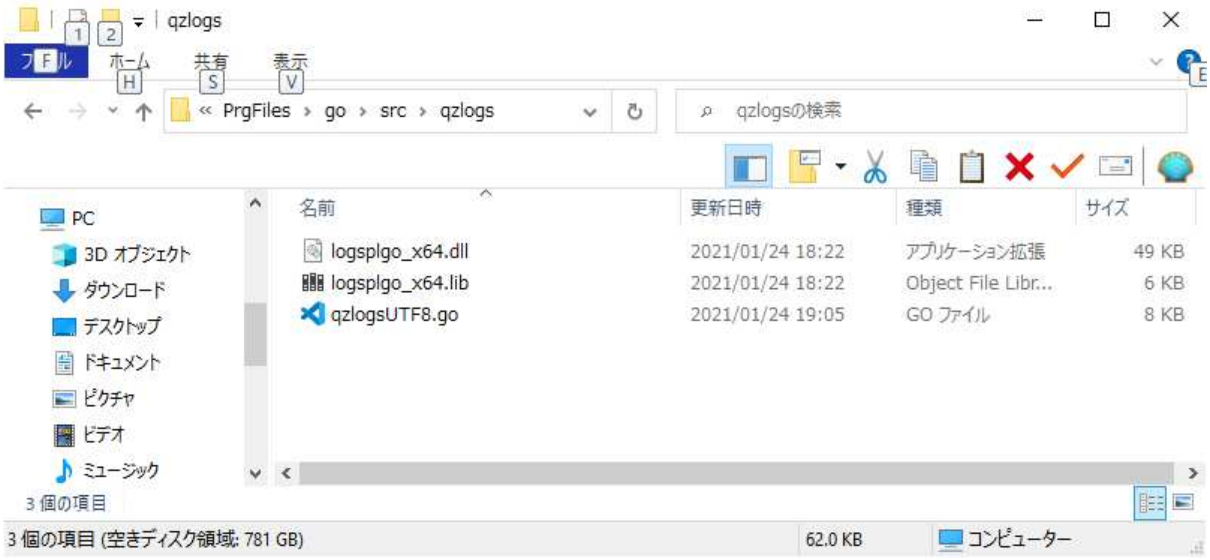


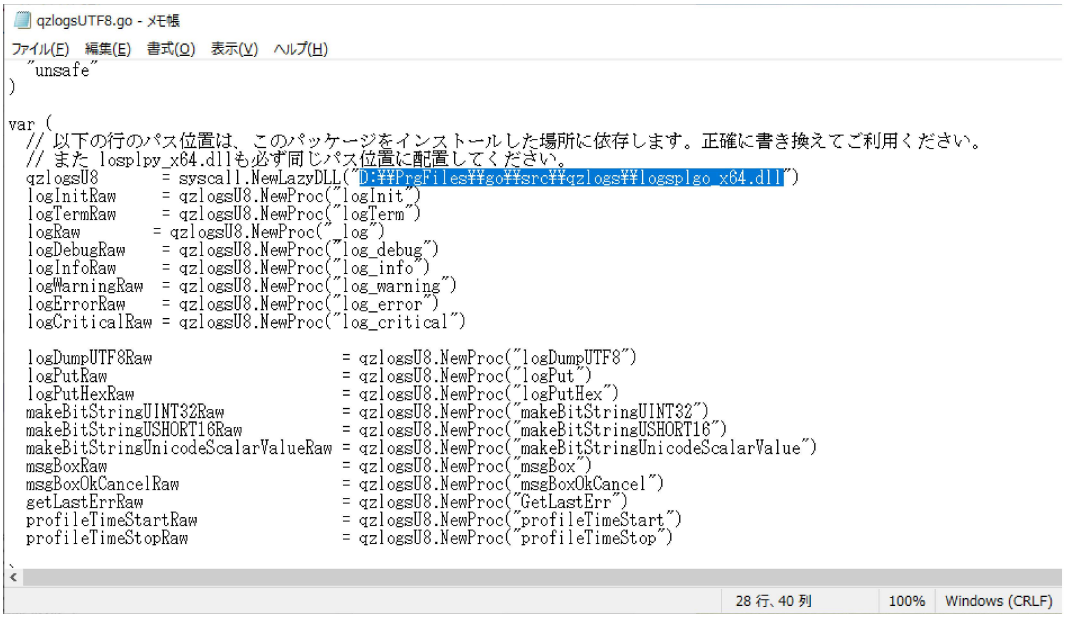
1.3.　　Windowsエクスプローラを使い、展開した先のqzlogsフォルダをGo/src以下に貼り付けます。



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。

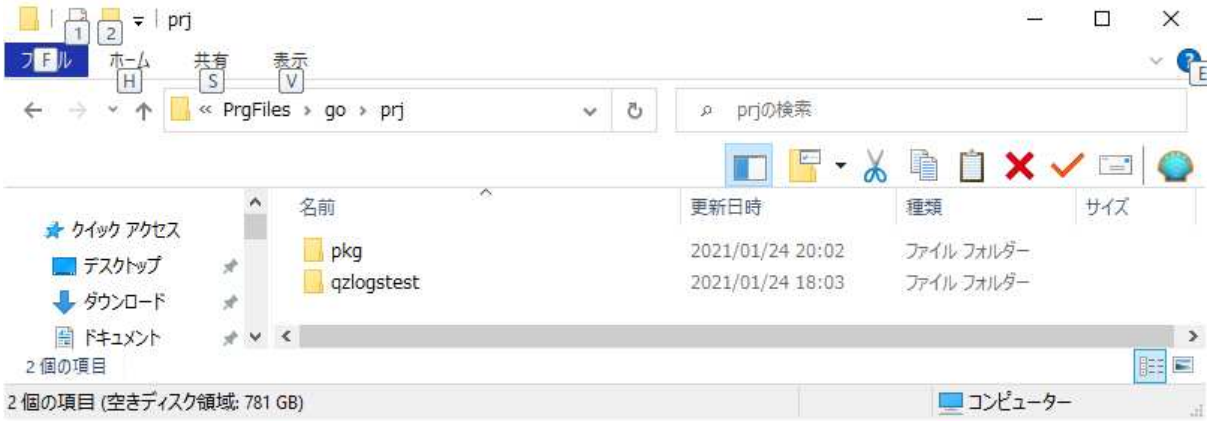
このパスは、利用される方がGoをインストールした場所に依存しますので、正確に変更してください。





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\qzlogstest みたいに他の場所にコピーします

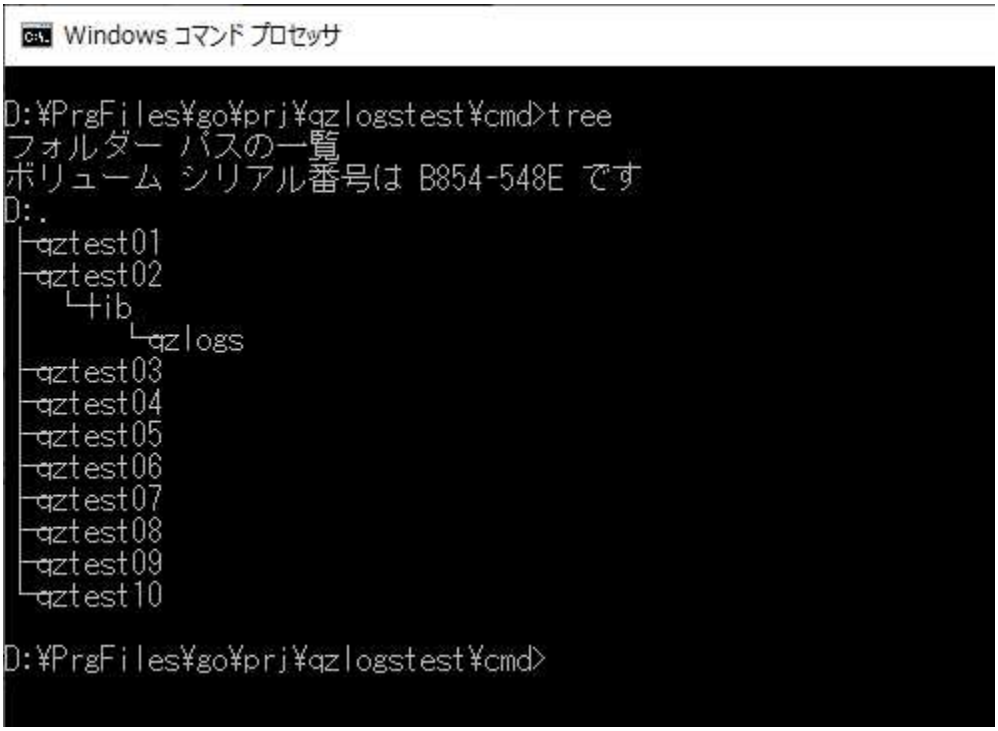


まず、qztest01のサンプルをビルドしてみます。
ビルドする場合は　cmdフォルダに入ります。

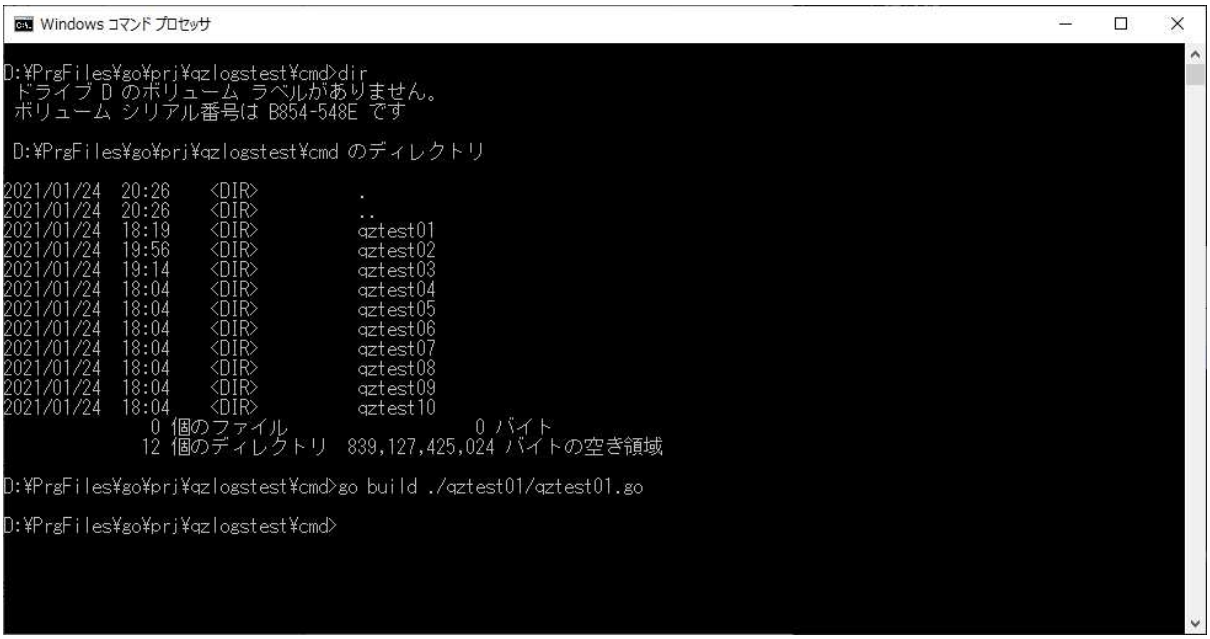
次に、qztest01フォルダ内のqztest01.goファイルをメモ帳やutf8対応エディタで開き、ログ・出力パスを、ダウンロードした方がインストールした先に書き換えます。

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\\qzlogstest\\cmd”になります。

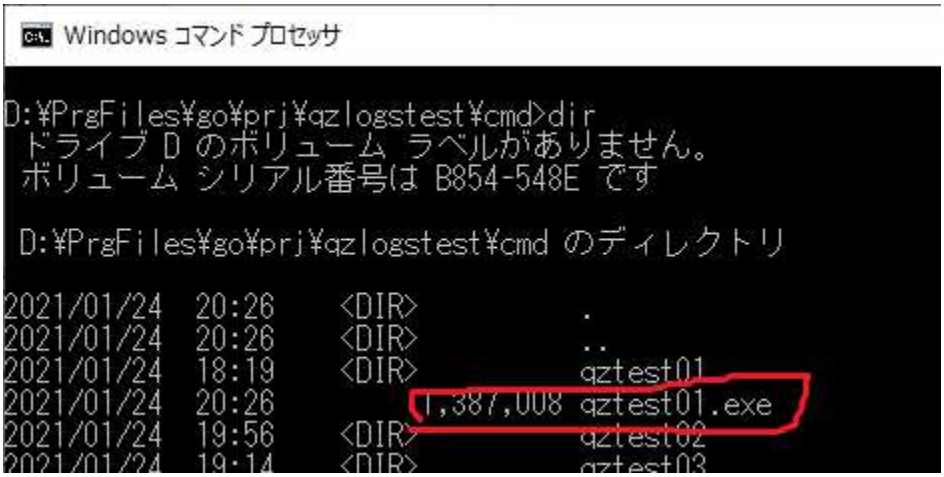
ディレクトリ構成図



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。

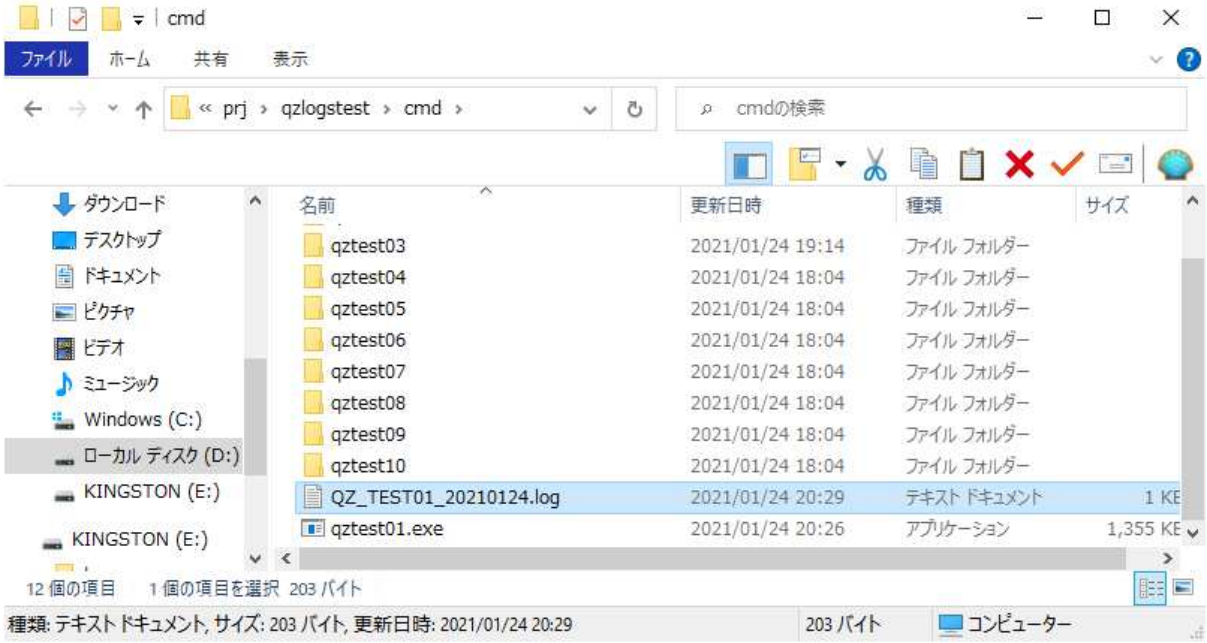


エラーがなければ、qztest01.exeが作成されます。

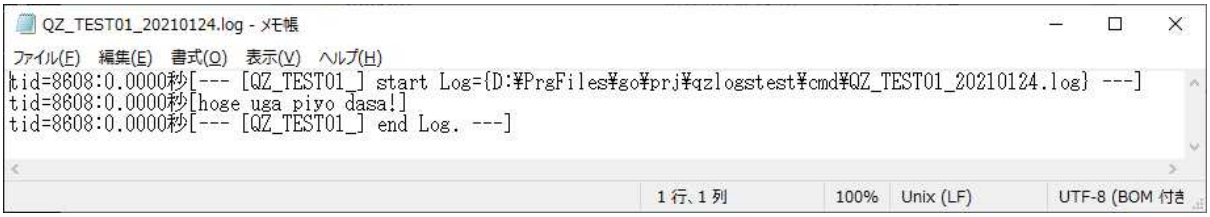


実行します。

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...ログ出力するファィ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. enumWriteFlg...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. enumCrLfOpt ...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。この点をご注意ください。

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。

GO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。

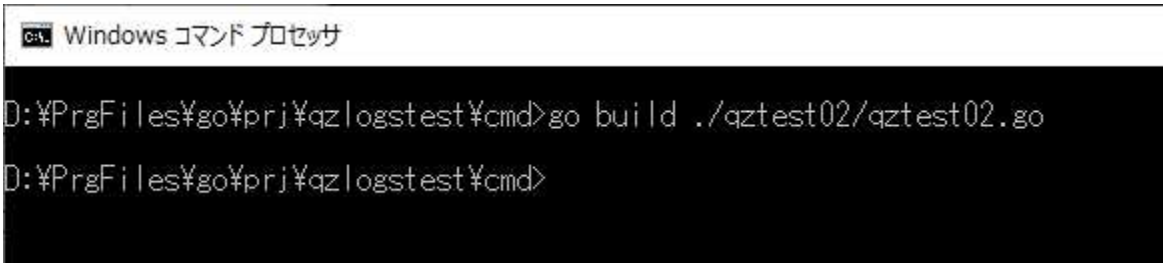
新たに、今コピーしたqzlogsフォルダの中のqzlogsUTF8.goを開き、パスを修正します。

私の環境はD:¥PrgFiles¥go　がGO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。

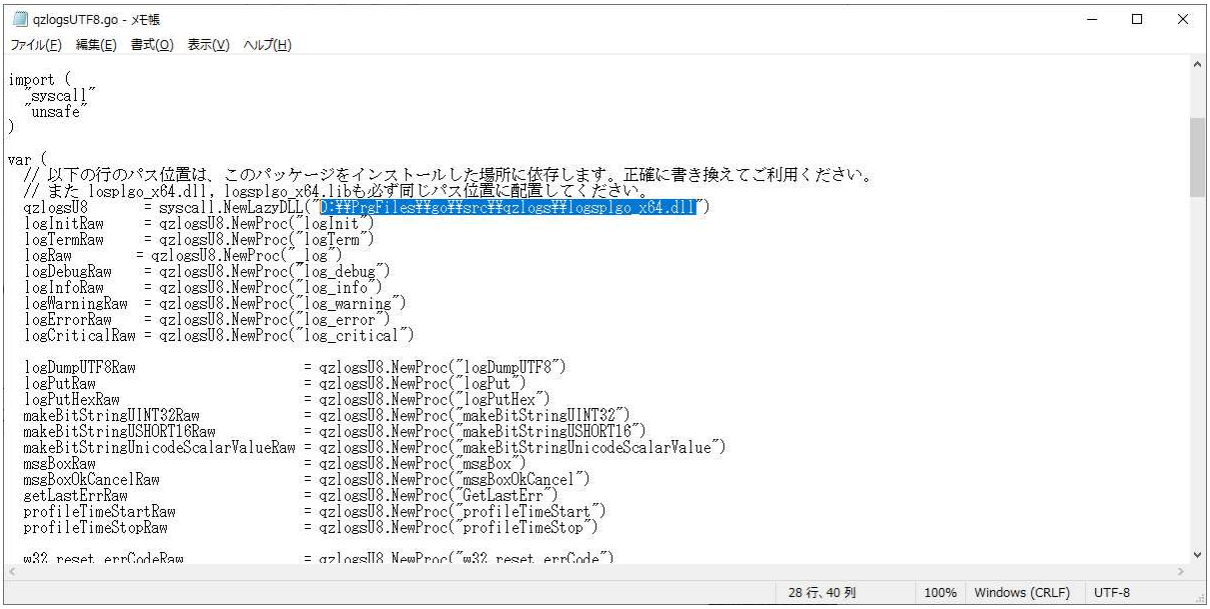
1.10.　ビルドコマンドは、この位置から打ちます

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

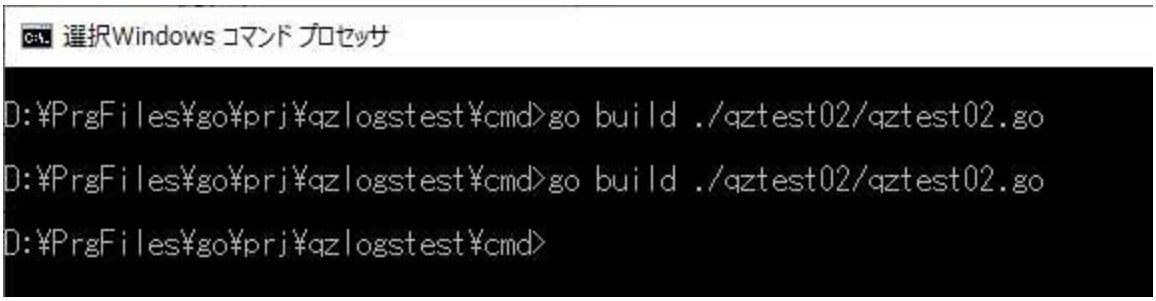


ビルド成功例

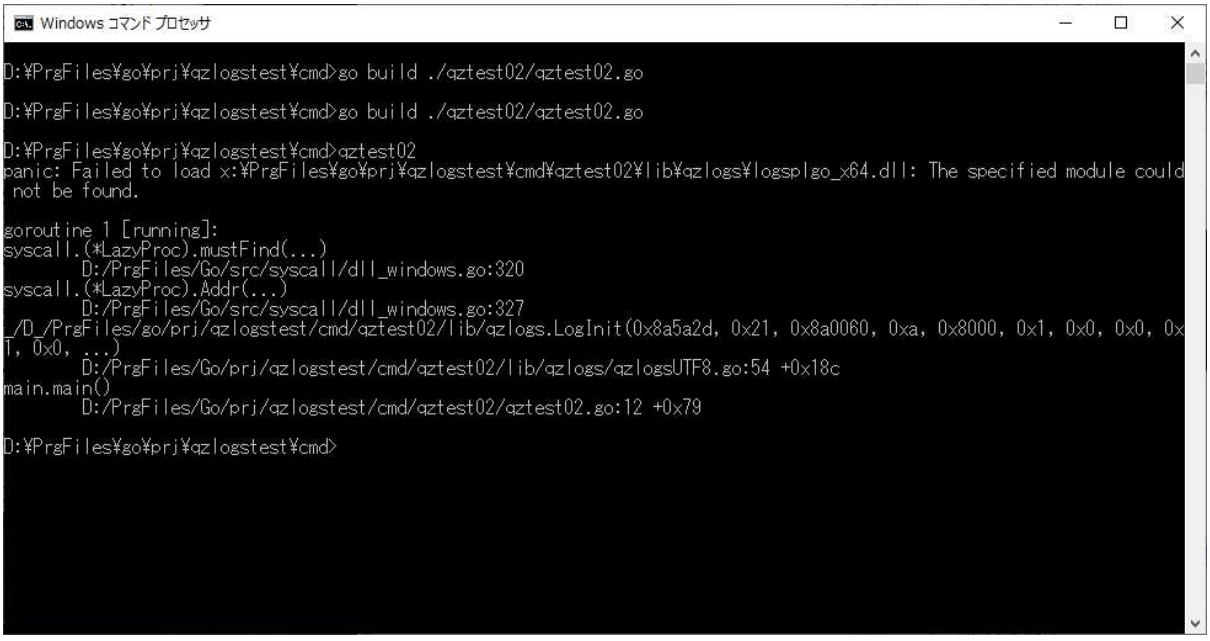
失敗する場合。書き換えた パスが間違っているばあいのエラーメッセージ
わざと存在しないドライブ　x:にしてみました
書き換えたのは、qztest02アプリのみ利用するパッケージの
D:¥PrgFiles¥go¥prj¥qzlogstest¥cmd¥qztest02¥lib¥qzlogs内にあるqzlogsUTF8.goです。



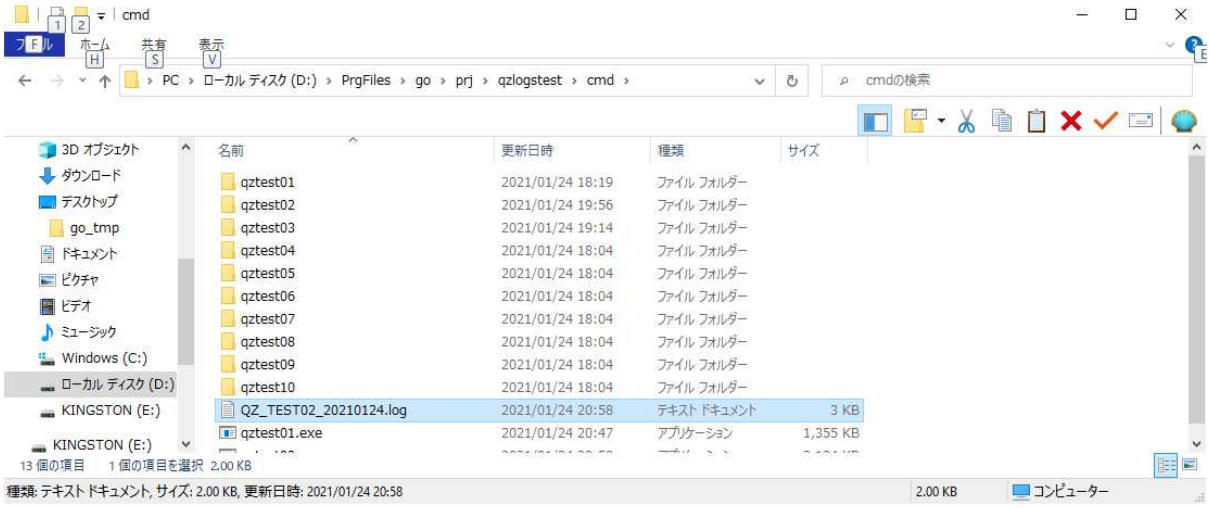
コンパイルは通りますが・・・



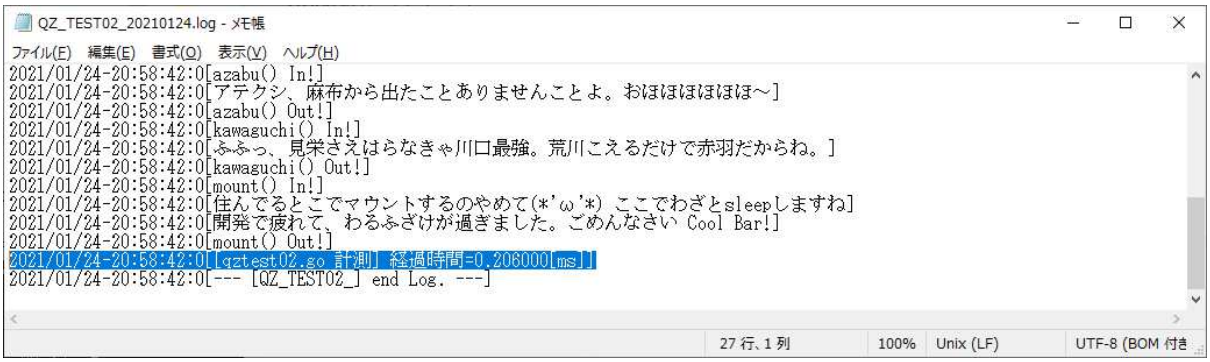
実行時にこんなエラーになります



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	1. string path 2. string prefix 3. enum WriteFlg 4. enum CrLfOpt 5. enum LogSleepSw 6. enum LogLevel 7. enum utf8BOM ※enumは数値型に置き換えてください	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。 prefixは、ログのファイル名識別用のユーザ定義自由文字列。 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 CrLfOptは 0...CR+LF 1...CR 2...LF　の指定となる LogSleepSw 0...ログ出力 1...ログを出力しない LogLevelは 1-6の値で、出力するメソッドを制御します utf8 BOM は　UTF8の 0...BOMをつけない 1...BOMを付ける
2	LogTerm	終了処理。 logInit系と必ず対に呼び出す必要がある	なし	

No	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 **注意 _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。**	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、netxtLineCntがぴっ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。

No	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	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か入力する lineは、ユーザブログラムのエラーチェッしたい行番号を入力
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	x 分解する符号なし16ビット prefix 識別文字列、UTF16文字などをセット 注、サロゲートペアは4バイトなので9番で処理する。
16	MakeBitStringUINT32	UINT32の値をビット文字列に分解する	1. ULONG x 2. string prefix	x 分解する符号なし32ビット prefix 識別文字列 UTF32文字などをセット

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値を、UTF16, UTF8のビット文字列に分解する	1. ULONG scalar 2. string prefix	scalar 分解するU+XXXXユニコード、スカラ値 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	1. string ダイアログ内文字列 2. string タイトル文字列	
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	3. string ダイアログ内文字列 4. string タイトル文字列	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	1. string prefix	計測終了時に、prefix文字列が表示されます **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。**
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_MILISEC	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prinf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”)

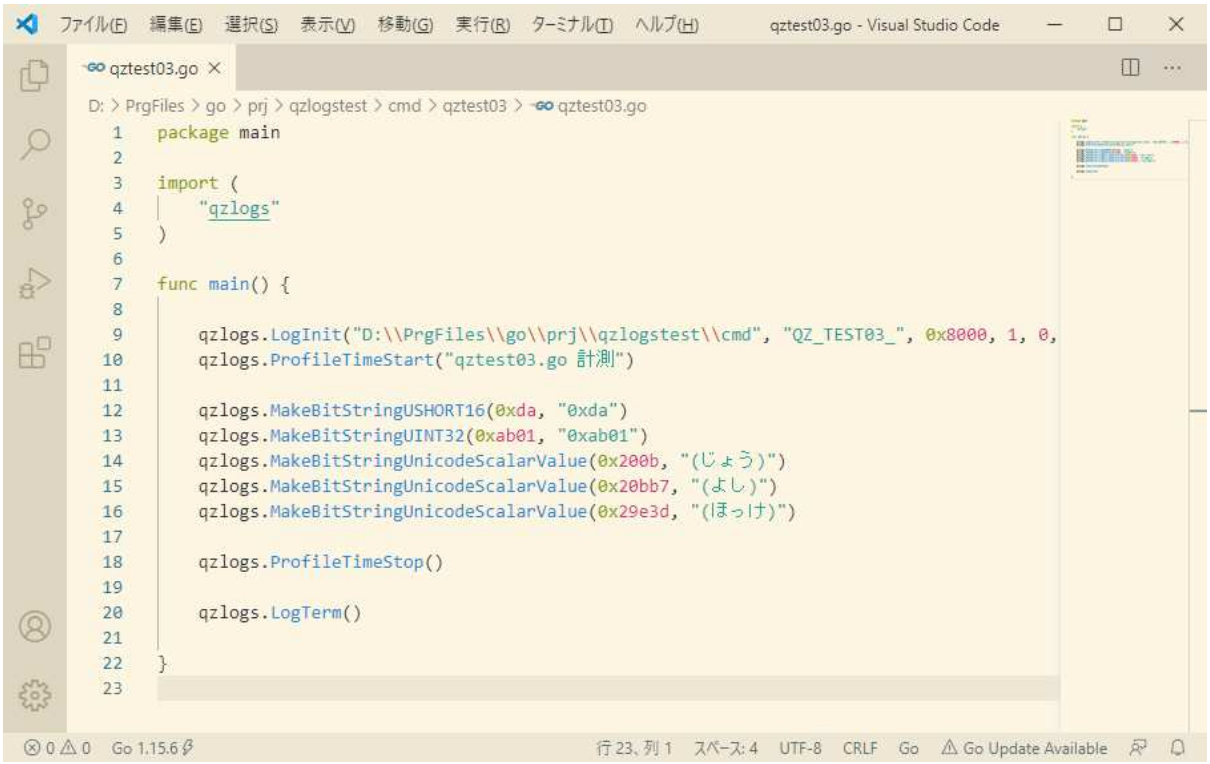
fmt.println(....

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。

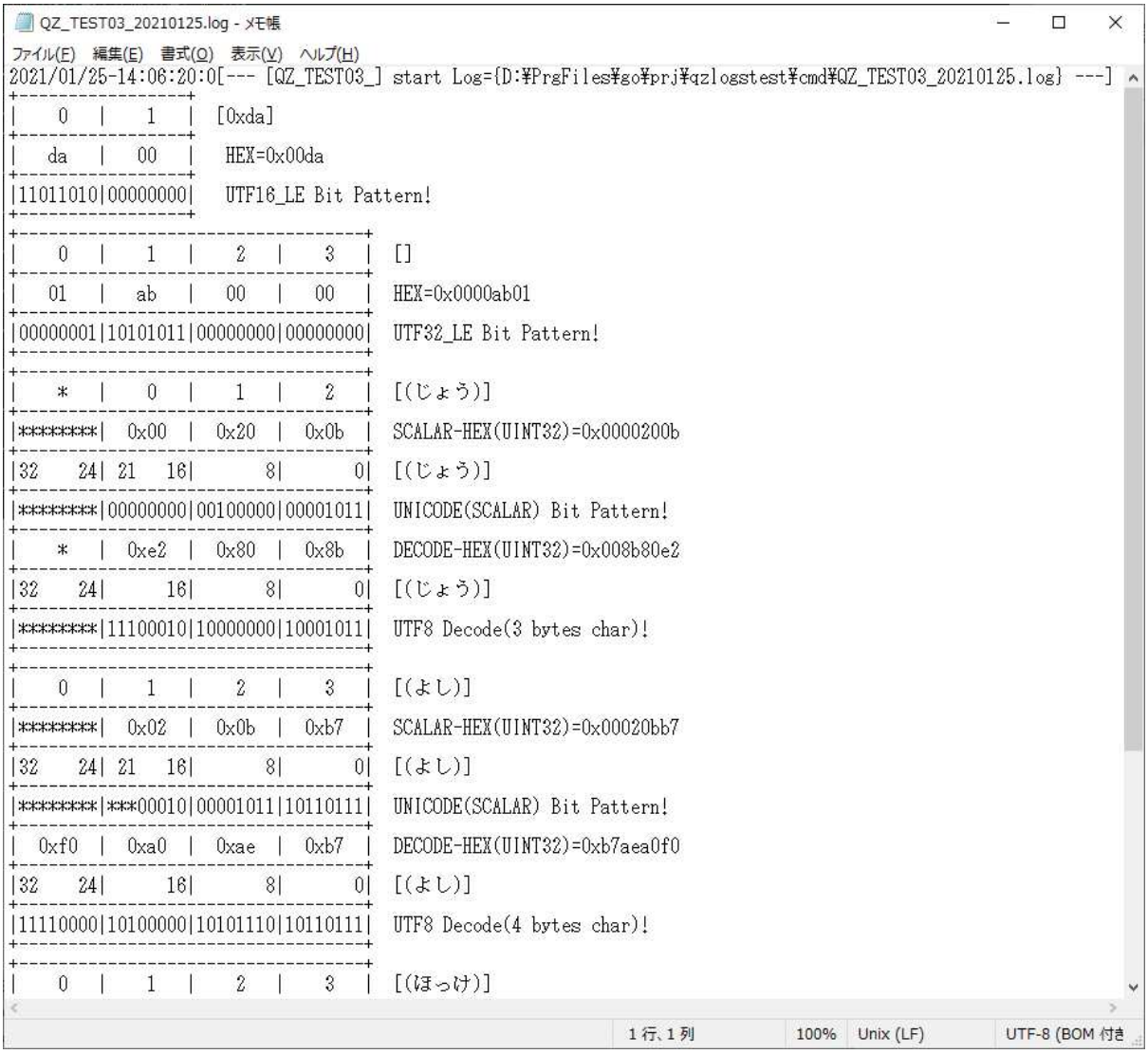
4.　実装例など

5.1. ビット文字列表示メソッド MakeBitStringUSHORT16(), MakeBitStringUINT32(), MakeBitStringUnicodeScalarValue()

サンプル　qztest03.go



書かれた、ログのイメージ。



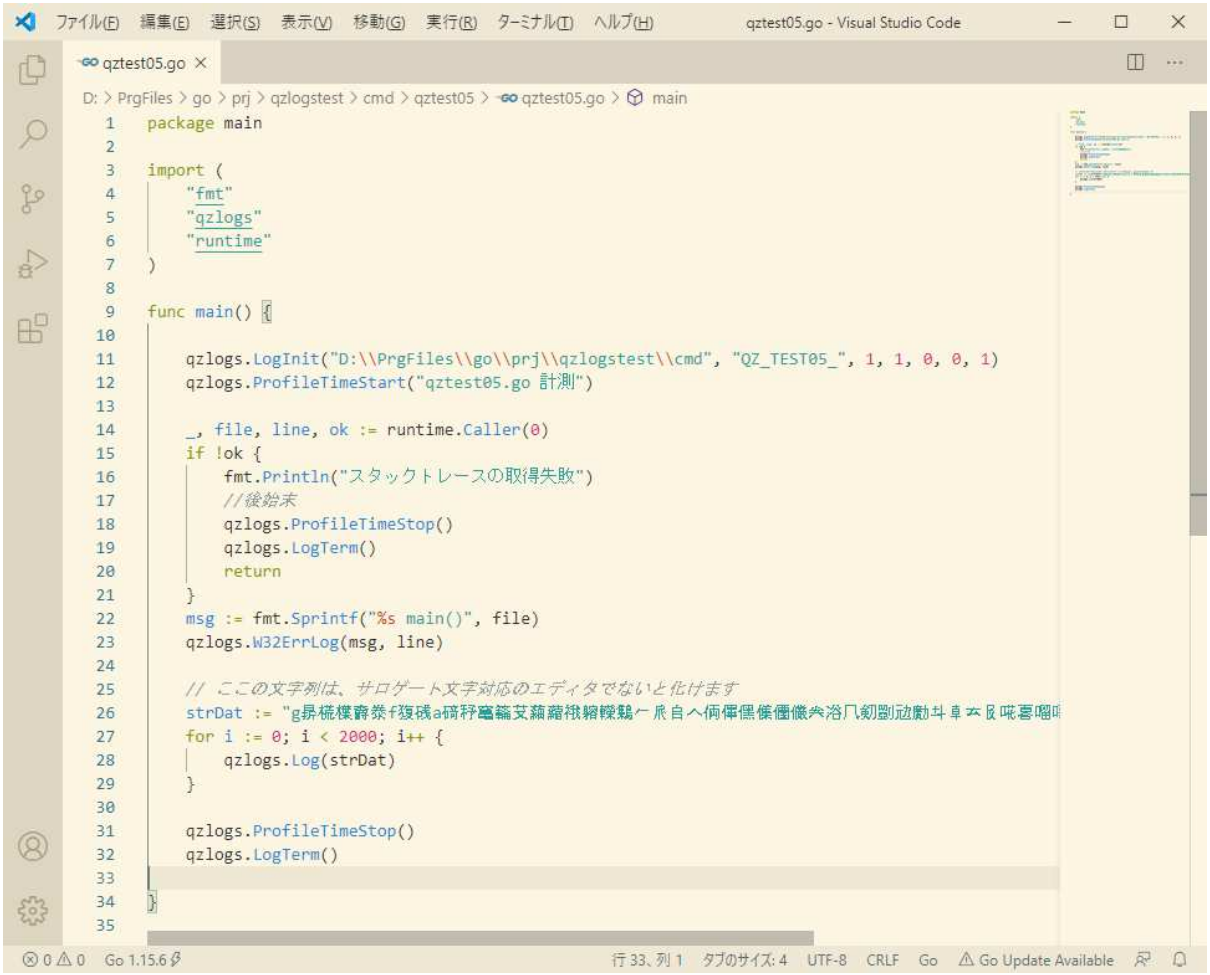
```
QZ_TEST03_20210125.log - メモ帳
ファイル(F)  編集(E)  書式(O)  表示(V)  ヘルプ(H)
2021/01/25-14:06:20:0[--- [QZ_TEST03_] start Log={D:¥PrgFiles¥go¥prj¥qzlogstest¥cmd¥QZ_TEST03_20210125.log} ---]
+----------+
|    0     |    1     |  [0xda]
+----------+
|   da     |   00     |   HEX=0x00da
+----------+
|11011010|00000000|   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)]
1 行、1 列   100%   Unix (LF)   UTF-8 (BOM 付き
```

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。

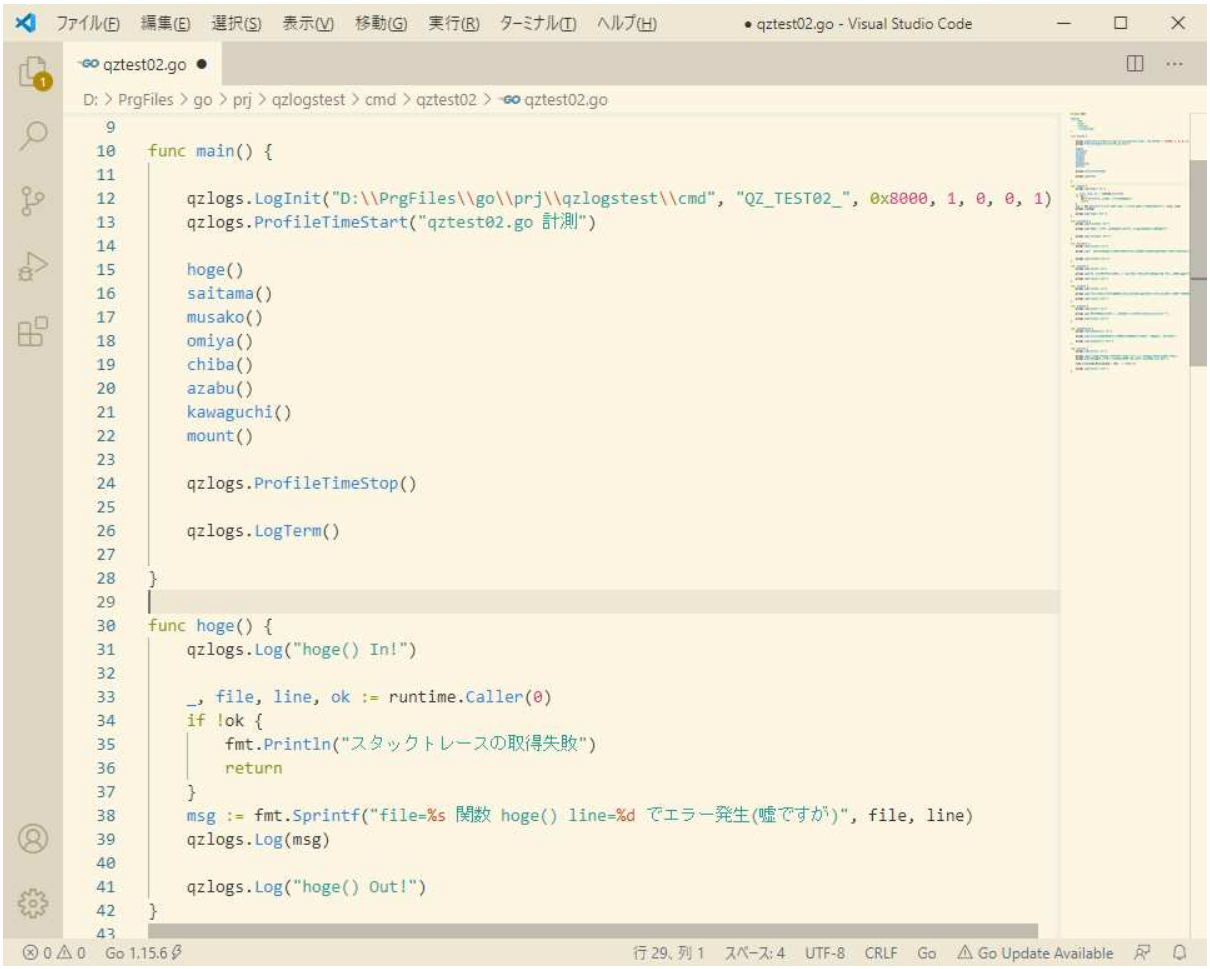
qzttest05.go



5.3. メソッドの　出入口にログを書いておくテクニック

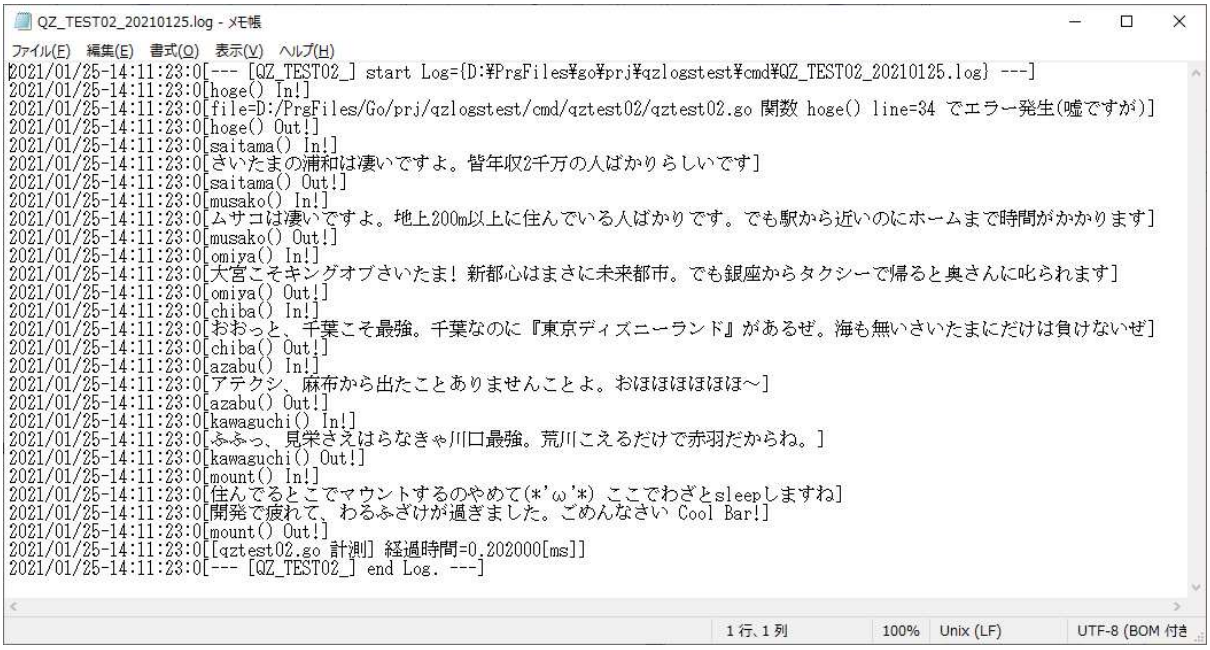
開発初期で、実装コードが不安定なとき、すごく役に立ちます。どこまで動作しているか一目瞭然です。

サンプル　qztest02.go



```go
func main() {

    qzlogs.LogInit("D:\\PrgFiles\\go\\prj\\qzlogstest\\cmd", "QZ_TEST02_", 0x8000, 1, 0, 0, 1)
    qzlogs.ProfileTimeStart("qztest02.go 計測")

    hoge()
    saitama()
    musako()
    omiya()
    chiba()
    azabu()
    kawaguchi()
    mount()

    qzlogs.ProfileTimeStop()

    qzlogs.LogTerm()

}
func hoge() {
    qzlogs.Log("hoge() In!")

    _, file, line, ok := runtime.Caller(0)
    if !ok {
        fmt.Println("スタックトレースの取得失敗")
        return
    }
    msg := fmt.Sprintf("file=%s 関数 hoge() line=%d でエラー発生(嘘ですが)", file, line)
    qzlogs.Log(msg)

    qzlogs.Log("hoge() Out!")
}
```

Sample2.py ログの結果。



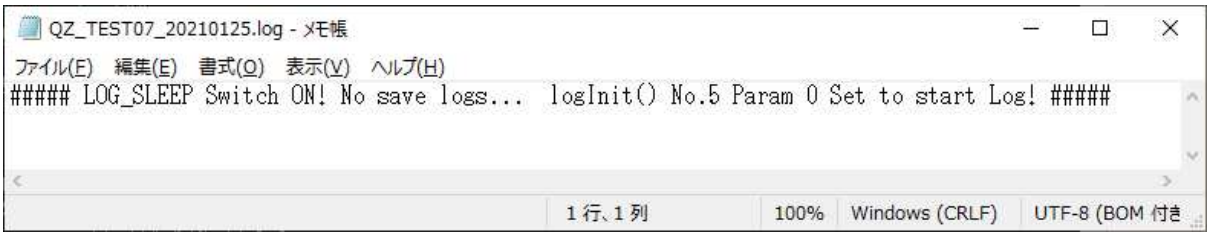
```
2021/01/25-14:11:23:0[--- [QZ_TEST02_] start Log={D:¥PrgFiles¥go¥prj¥qzlogstest¥cmd¥QZ_TEST02_20210125.log} ---]
2021/01/25-14:11:23:0[hoge() In!]
2021/01/25-14:11:23:0[file=D:/PrgFiles/Go/prj/qzlogstest/cmd/qztest02/qztest02.go 関数 hoge() line=34 でエラー発生(嘘ですが)]
2021/01/25-14:11:23:0[hoge() Out!]
2021/01/25-14:11:23:0[saitama() 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[アテクシ、麻布から出たことありませんことよ。おほほほほほ～]
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[ふふっ、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言うことなので、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。

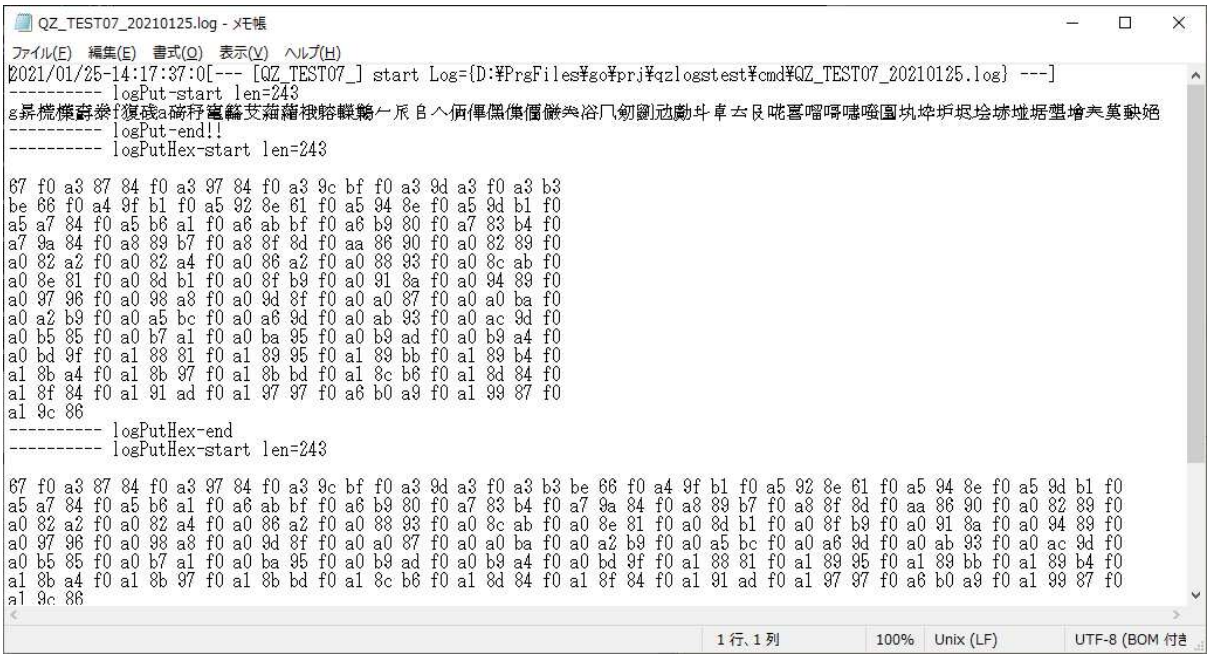
5.4. logSleepSw = 1 の簡易ログ停止機能

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。



qztest07.go

ちなみに、5番目のパラメータを0にするとこうなります。



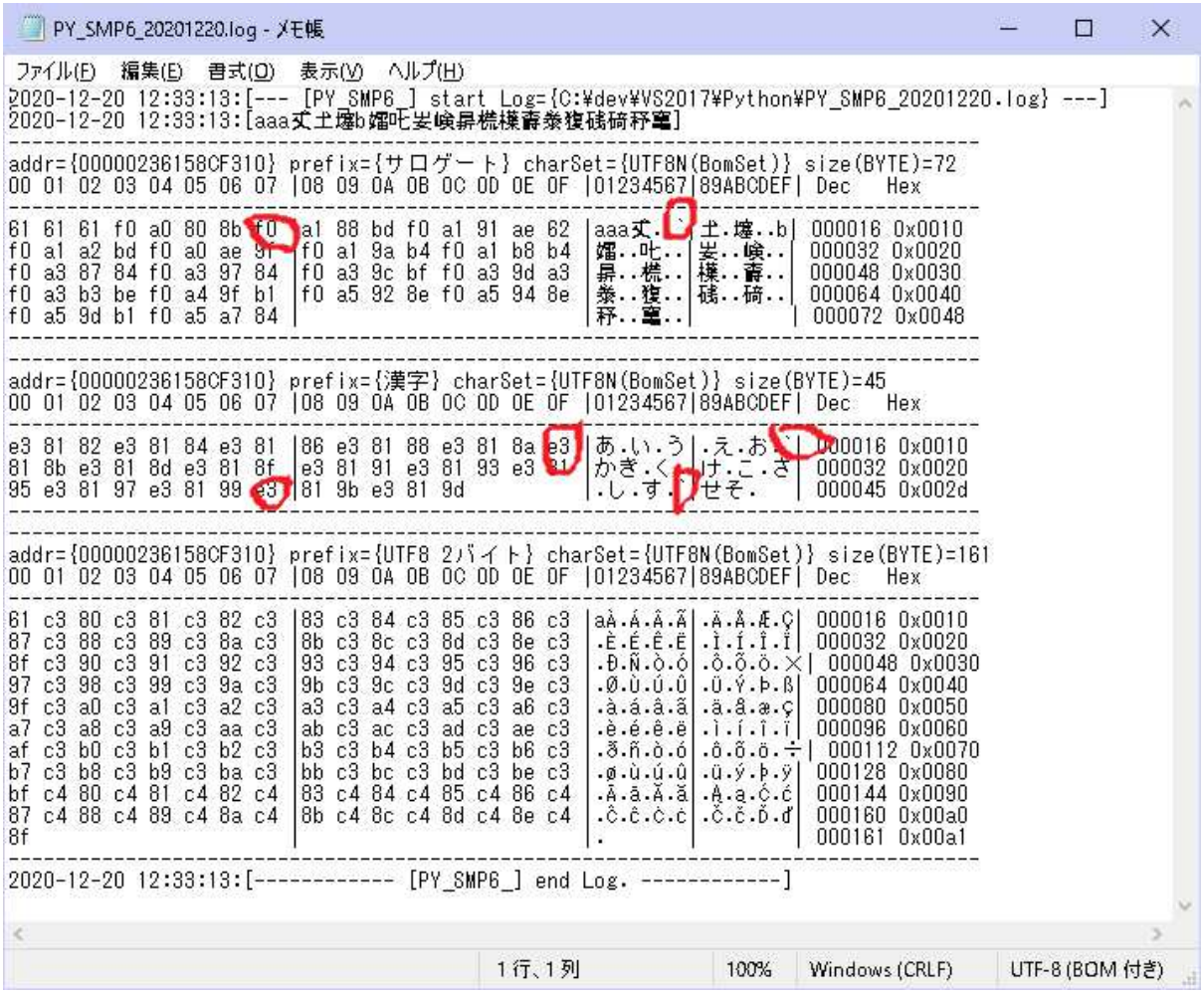
5.5. UTF8 文字列ダンプ時の禁則処理の説明

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 ,`, 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す)



```
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,`土壜b媚叱岌嗹昇槻様蕎叀複硲碕秤竃]
------------------------------------------------------------------------------
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
------------------------------------------------------------------------------
61 61 61 f0 a0 80 8b f0 |a1 88 bd f0 a1 91 ae 62 |aaa丈.`.|土.壜..b| 000016 0x0010
f0 a1 a2 bd f0 a0 ae 9f |f0 a1 9a b4 f0 a1 b8 b4 |媚..叱..|岌..嗹..| 000032 0x0020
f0 a3 87 84 f0 a3 97 84 |f0 a3 9c bf f0 a3 9d a3 |昇..槻..|様..蕎..| 000048 0x0030
f0 a3 b3 be f0 a4 9f b1 |f0 a5 92 8e f0 a5 94 8e |叀..複..|硲..碕..| 000064 0x0040
f0 a5 9d b1 f0 a5 a7 84 |                        |秤..竃..|        | 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.き.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
------------------------------------------------------------------------------
61 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
87 c3 88 c3 89 c3 8a c3 |8b c3 8c c3 8d c3 8e c3 |.È.É.Ê.Ë|.Ì.Í.Î.Ï| 000032 0x0020
8f c3 90 c3 91 c3 92 c3 |93 c3 94 c3 95 c3 96 c3 |.Ð.Ñ.Ò.Ó|.Ô.Õ.Ö.×| 000048 0x0030
97 c3 98 c3 99 c3 9a c3 |9b c3 9c c3 9d c3 9e c3 |.Ø.Ù.Ú.Û|.Ü.Ý.Þ.ß| 000064 0x0040
9f c3 a0 c3 a1 c3 a2 c3 |a3 c3 a4 c3 a5 c3 a6 c3 |.à.á.â.ã|.ä.å.æ.ç| 000080 0x0050
a7 c3 a8 c3 a9 c3 aa c3 |ab c3 ac c3 ad c3 ae c3 |.è.é.ê.ë|.ì.í.î.ï| 000096 0x0060
af c3 b0 c3 b1 c3 b2 c3 |b3 c3 b4 c3 b5 c3 b6 c3 |.ð.ñ.ò.ó|.ô.õ.ö.÷| 000112 0x0070
b7 c3 b8 c3 b9 c3 ba c3 |bb c3 bc c3 bd c3 be c3 |.ø.ù.ú.û|.ü.ý.þ.ÿ| 000128 0x0080
bf c4 80 c4 81 c4 82 c4 |83 c4 84 c4 85 c4 86 c4 |.Ā.ā.Ă.ă|.Ą.ą.Ć.ć| 000144 0x0090
87 c4 88 c4 89 c4 8a c4 |8b c4 8c c4 8d c4 8e c4 |.Ĉ.ĉ.Ċ.ċ|.Č.č.Ď.ď| 000160 0x00a0
8f                      |                        |.       |        | 000161 0x00a1
------------------------------------------------------------------------------
2020-12-20 12:33:13:[------------ [PY_SMP6_] end Log. ------------]
                                    1 行、1 列   100%   Windows (CRLF)   UTF-8 (BOM 付き)
```

6. サンプルプログラム

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れております
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