Qzlogs for Rust 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
logsplrs_x64.dll
logsplrs_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
rustc 1.49.0 (e1884a8e3 2020-12-29) Windows64 Windows 10 20H2 で確認しております
開発元　　QuiZ Lab. 2021.02.02
バージョン　0.5.2.1

注意　Build Tools for Visual Studio 2019の別途先行インストールが必須です。
Go,Rustともにコンパイル言語なので必要となります。
※ BuildTooslインストールマニュアル.pdf 参照

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(log_dump_raw())
  log_put_hexでもバイトデータを列挙できます
- 1文字log_put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 msg_box, msg_box_okcancell
- 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

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）
2021/02/02　　改版　0.5.2,1 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。

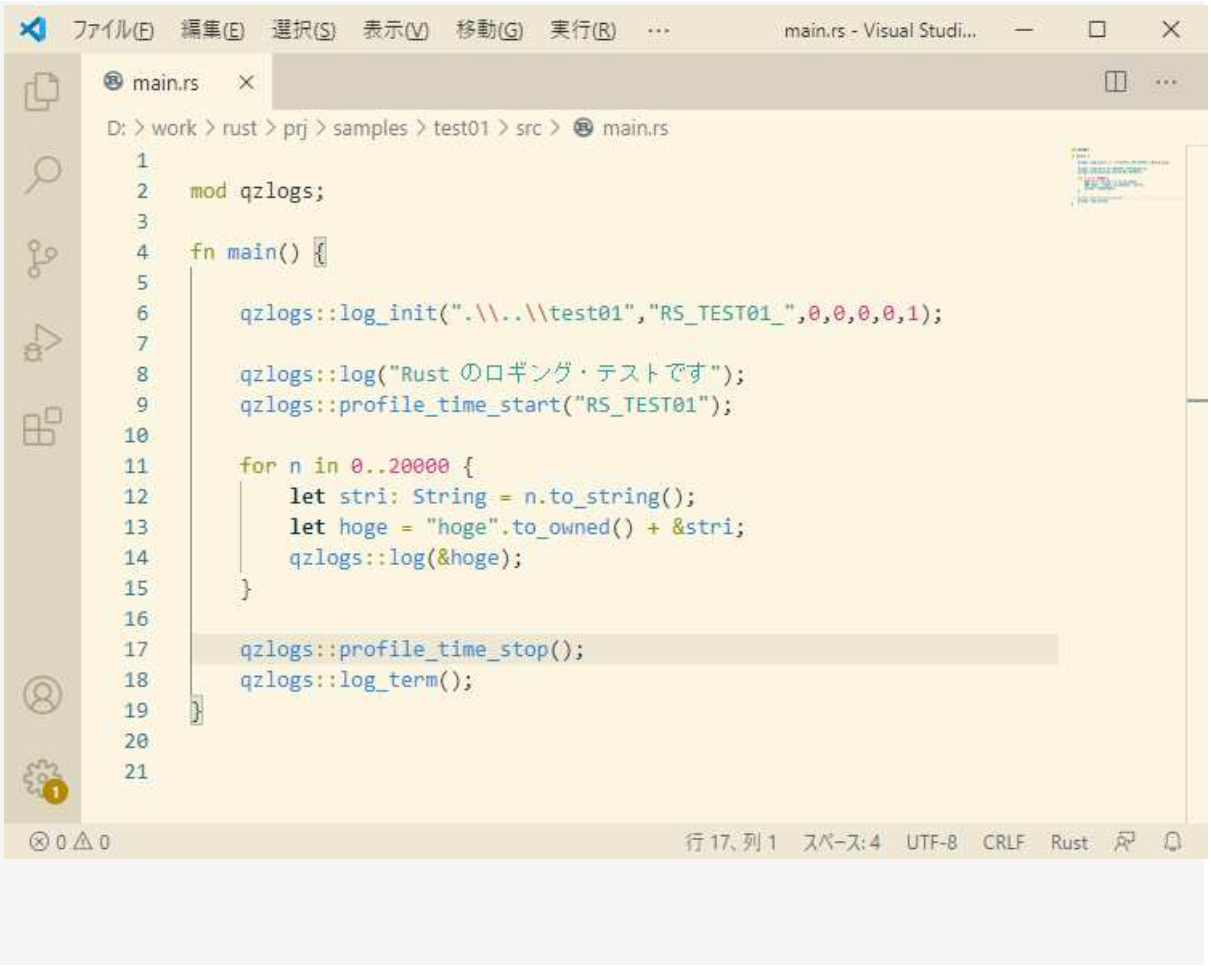
本ソフトウェアツールを利用して発生した損害につきましては、いかなる場合に於いても保障することはできません。すべて利用する側の責任でご利用ください。また動作の完全性も保証はできかねます。下記にもあるように、バグ対応のメンテナンスは積極的に行いますので、ぜひご利用されて、フィードバックをいただけると有難く存じます。

■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

■ Rust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。Rust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 Rust (Windows 10)

cargo new test01 で作成されたフォルダ /src 内にqzlogs.rs qzlogs.libを配置



rust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

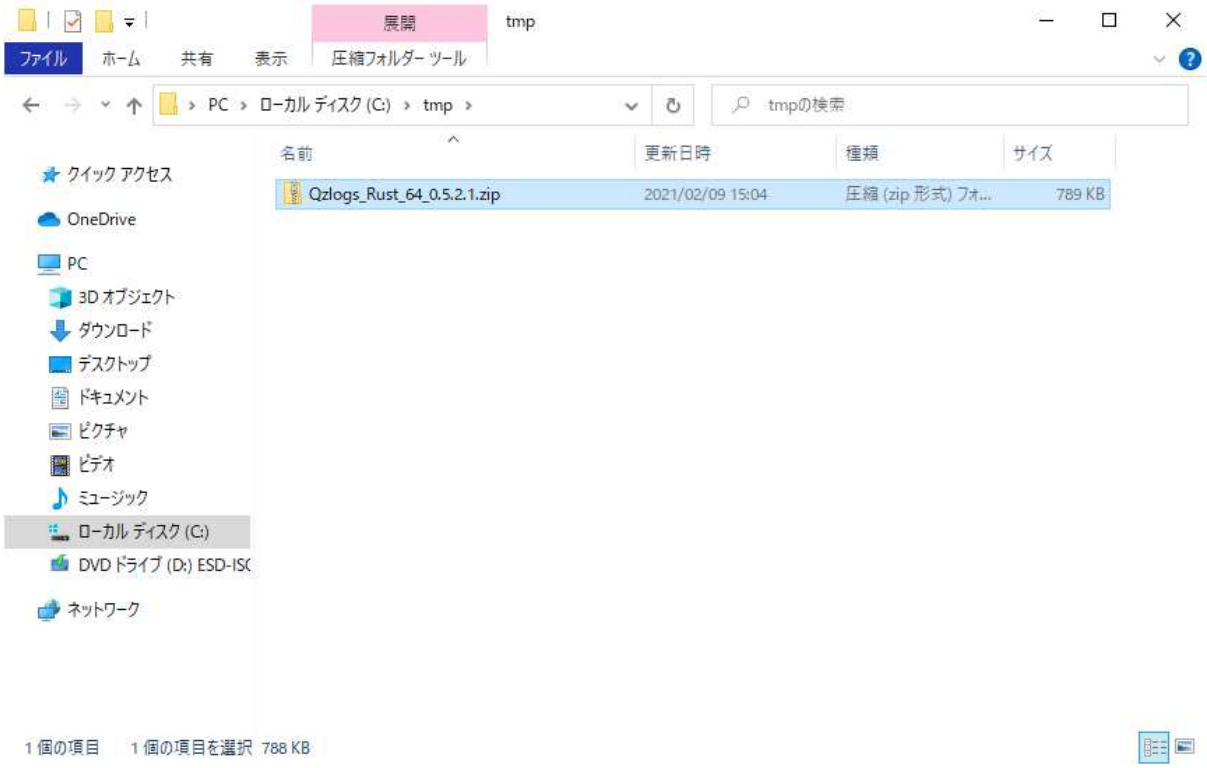
1.1. cargo new で作成された srcフォルダにQzlogsを配置する例

rustのインストール・ルートに移動します
rust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、rustの公認サイトをご確認ください) テスト環境にパッケージフォルダをコピーするだけですが・・・

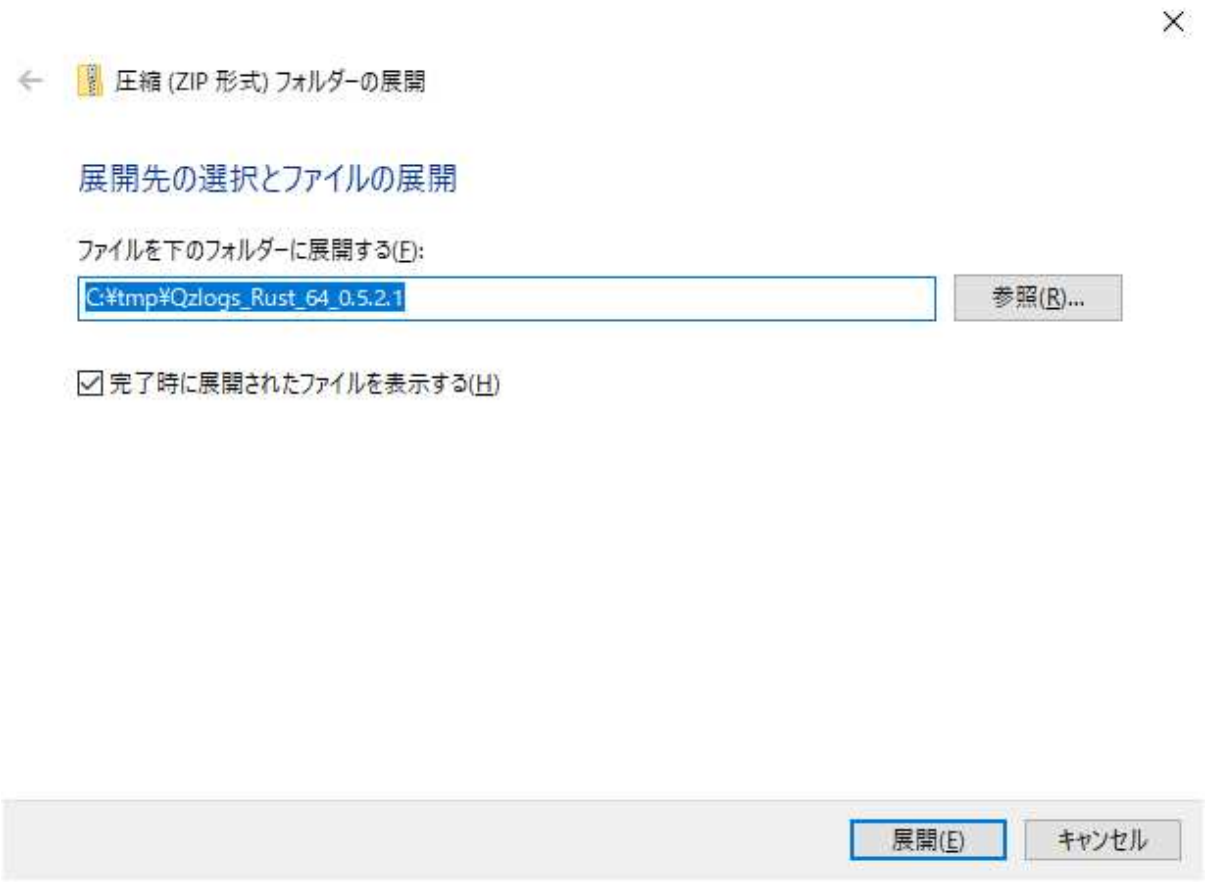
zipファイルを解凍し、qzlogsフォルダをフォルダごと　rustの作業場所にコピーする作業になります。

1.2.展開場所をエクスプローラで開きます

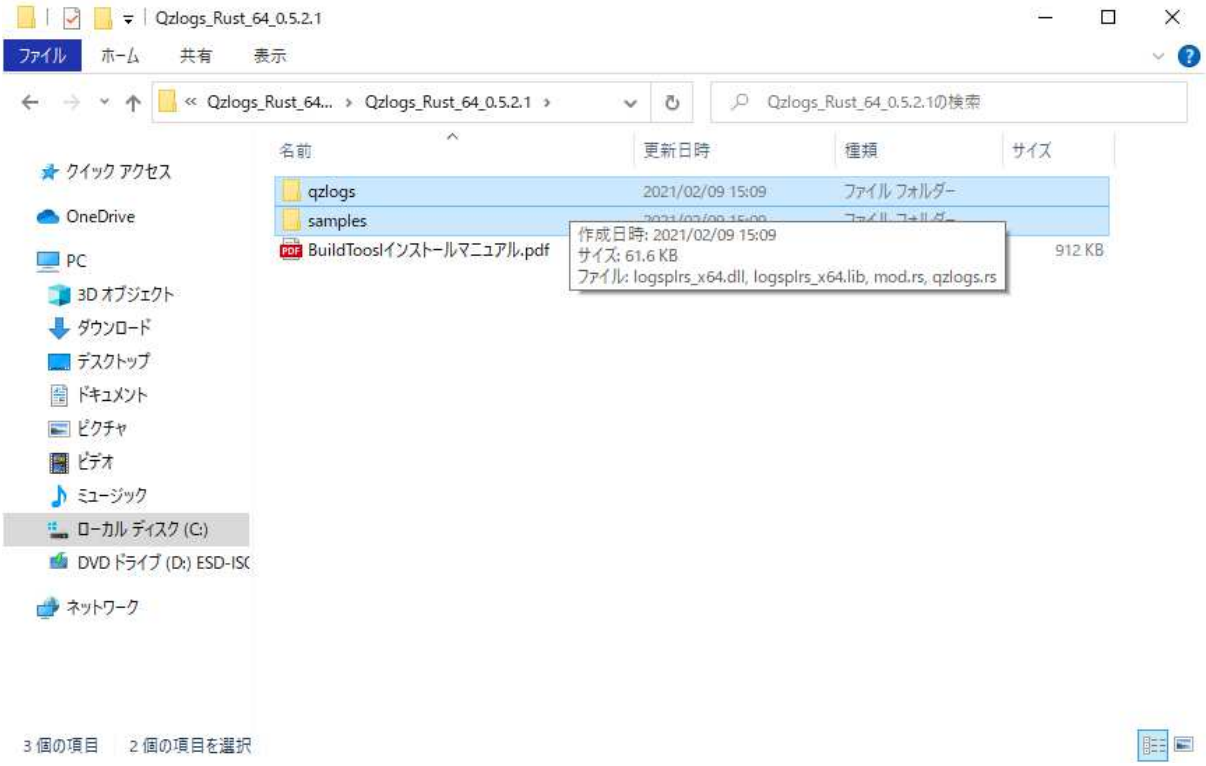
1.3. zipファイルを展開します



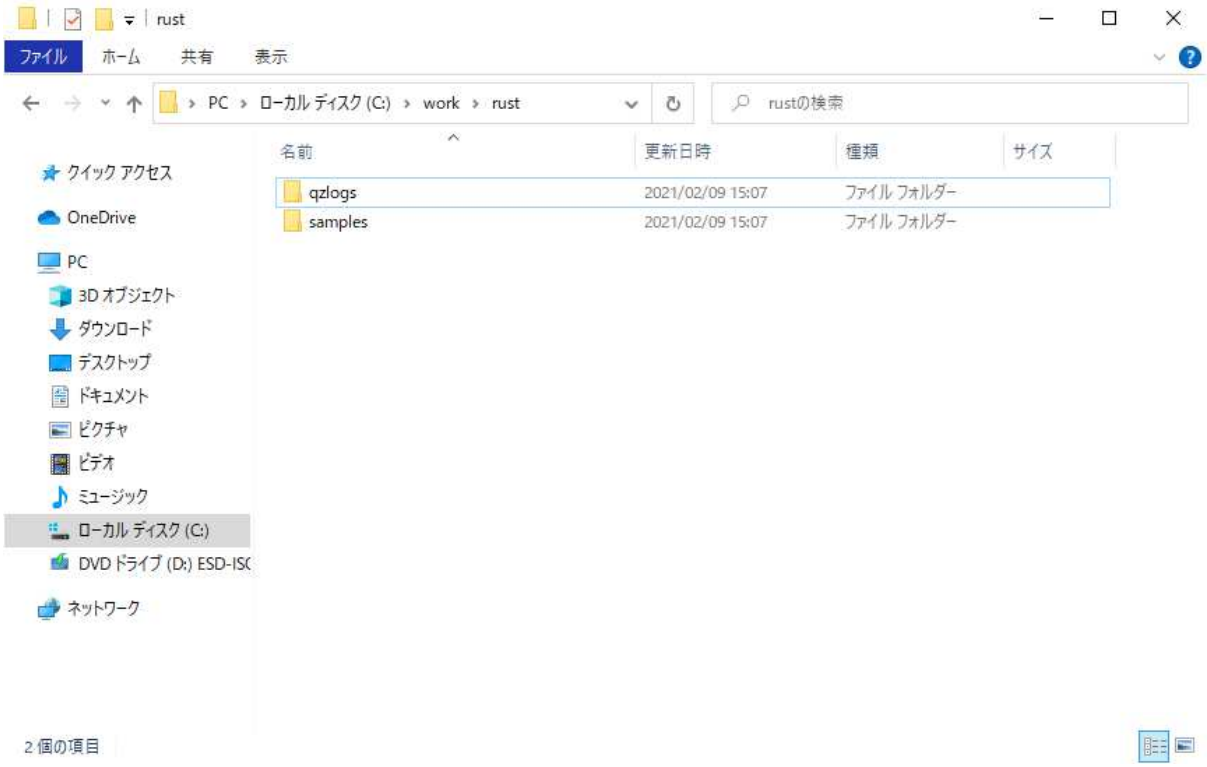
ここでは、C:\tmp に展開しています



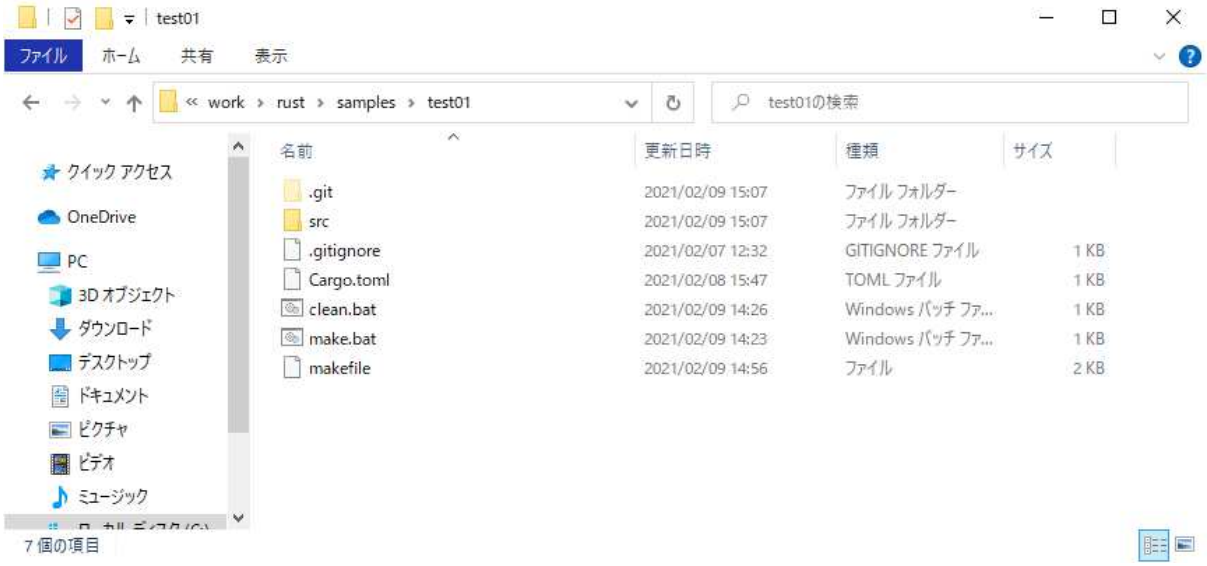
展開したフォルダを掘り下げると、qzlogs, samples フォルダがみつかります
このふたつのフォルダを、rustプログラムをコンパイルする作業場所にコピーします



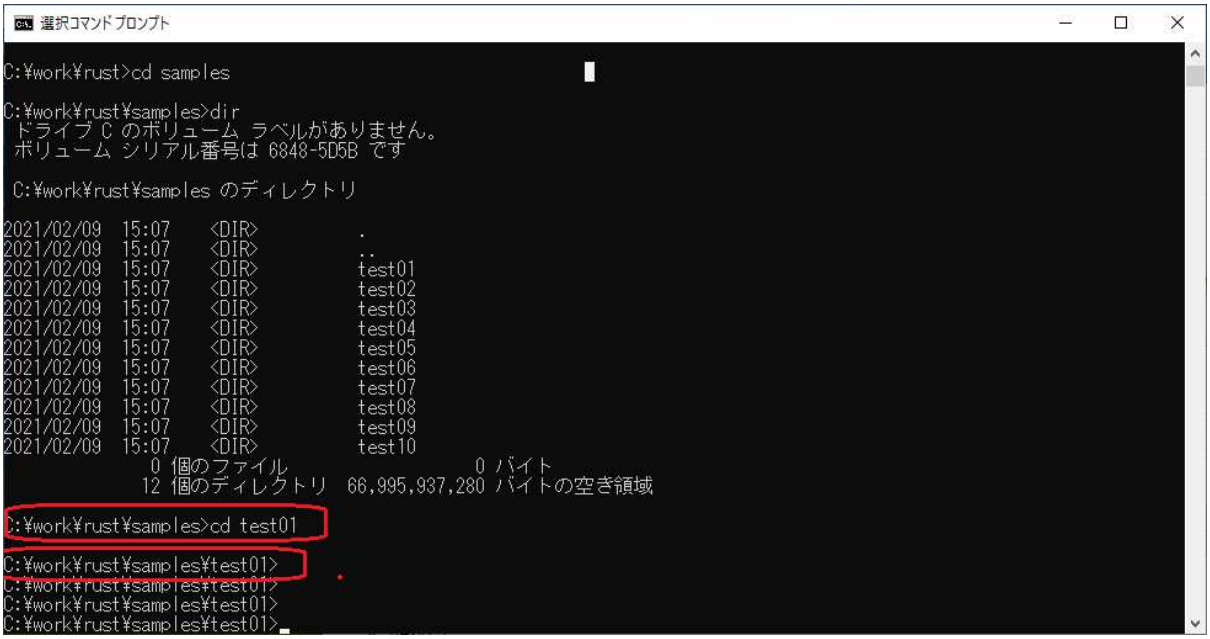
C:\tmp から、作業場所の　C:\work\rustに貼り付けました



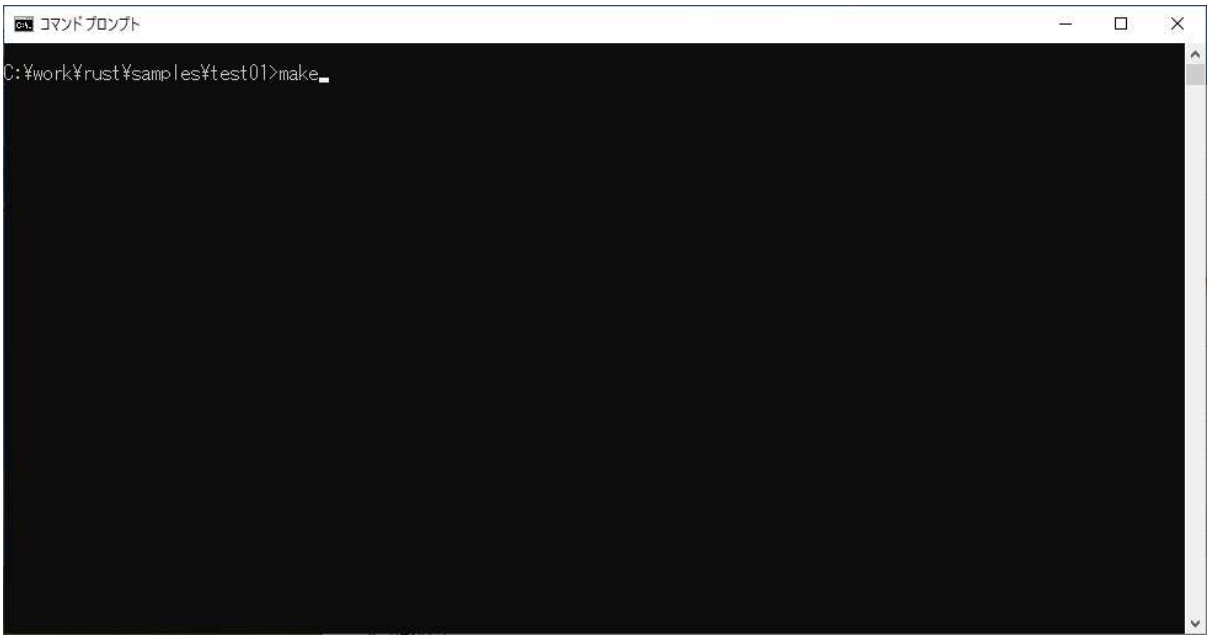
samplesをエクスプローラで掘り下げます。test01の中を表示させます



コマンド・プロンプトを起動し、cd コマンドで、test01の中に入ります。



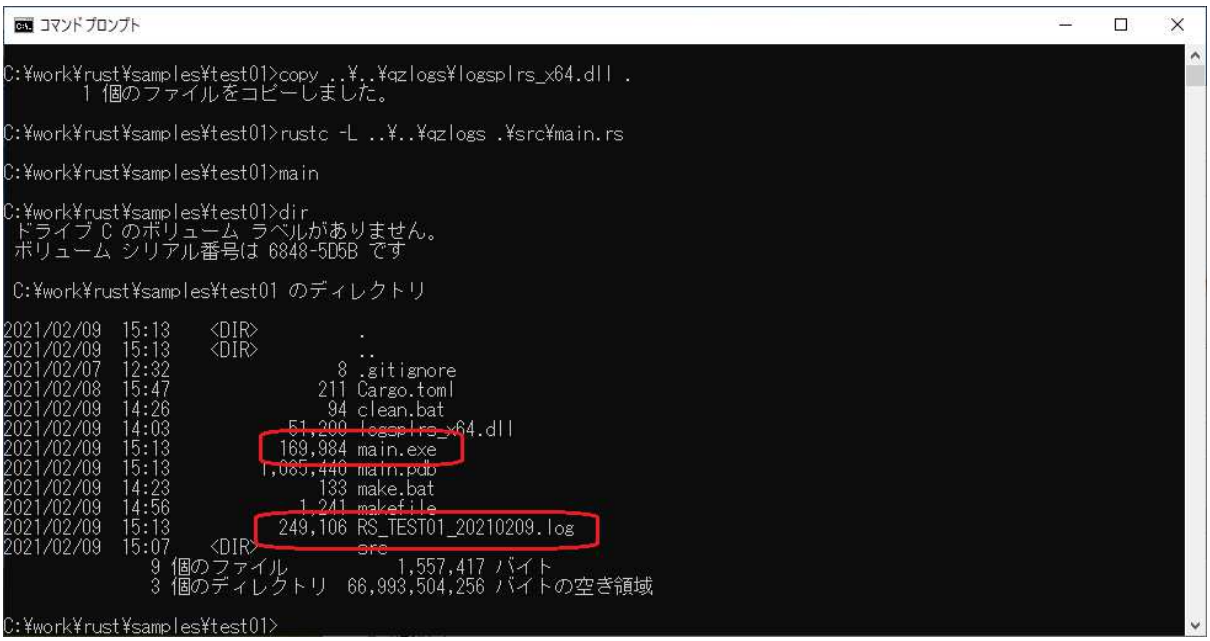
make.bat というサンプルプログラムビルドバッチがありますので、これを実行します。



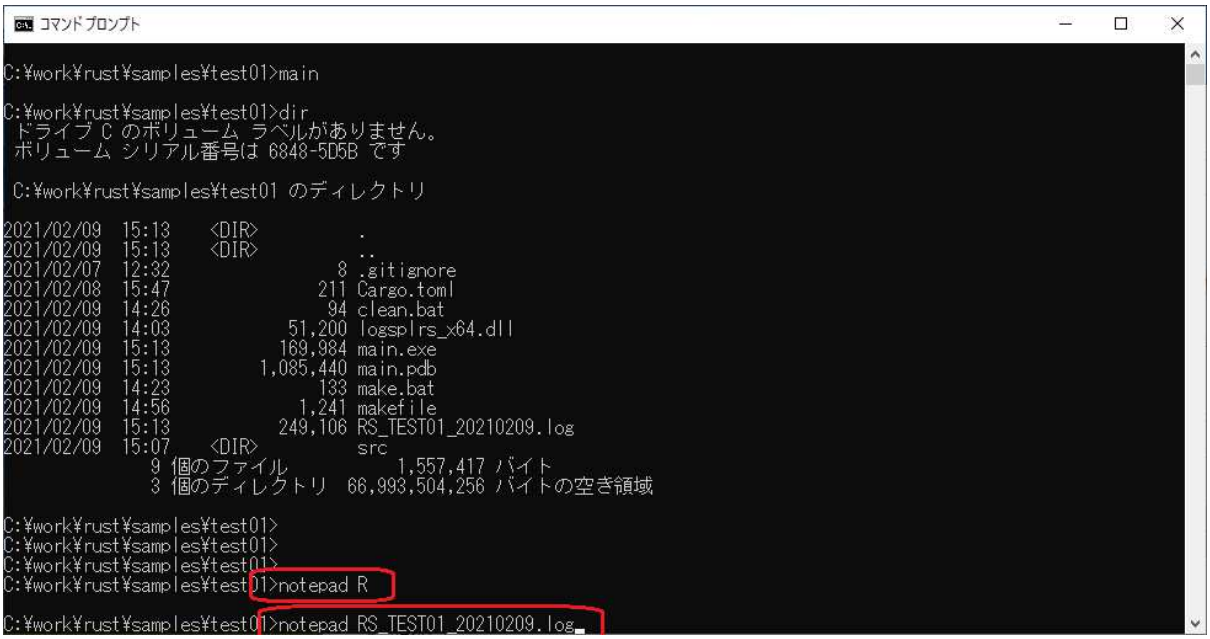
注意　コマンドプロンプトでは、日本語を混ぜる場合はCP65001　utf8に文字コードを決めて保存してください。

本サンプルは、Windowsのコマンドプロンプトでバッチおよびmakefileを実行する際にchcp65001を呼び出して、強制的に UTF8にしております。

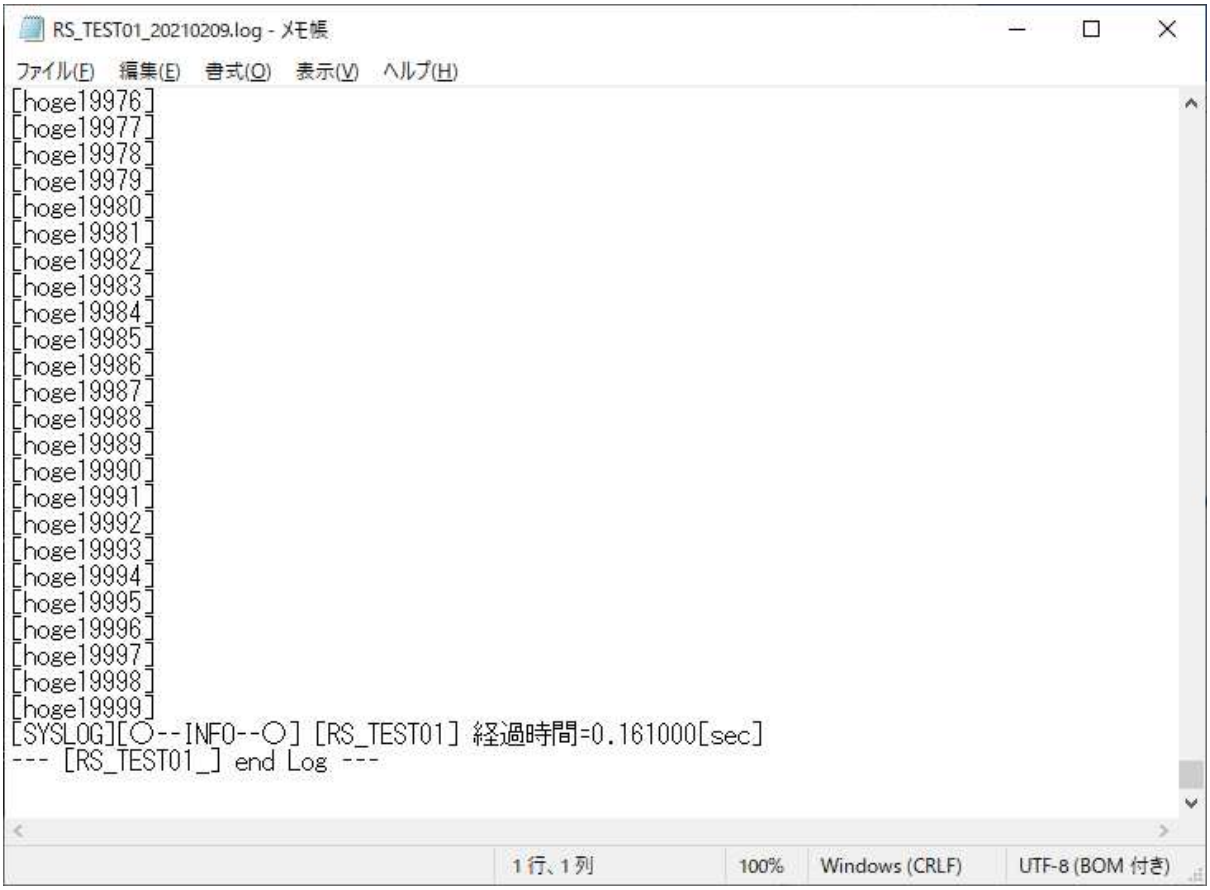
実行が成功すれば、main.exeと、RS_TEST_今日の日付.logが作成されます



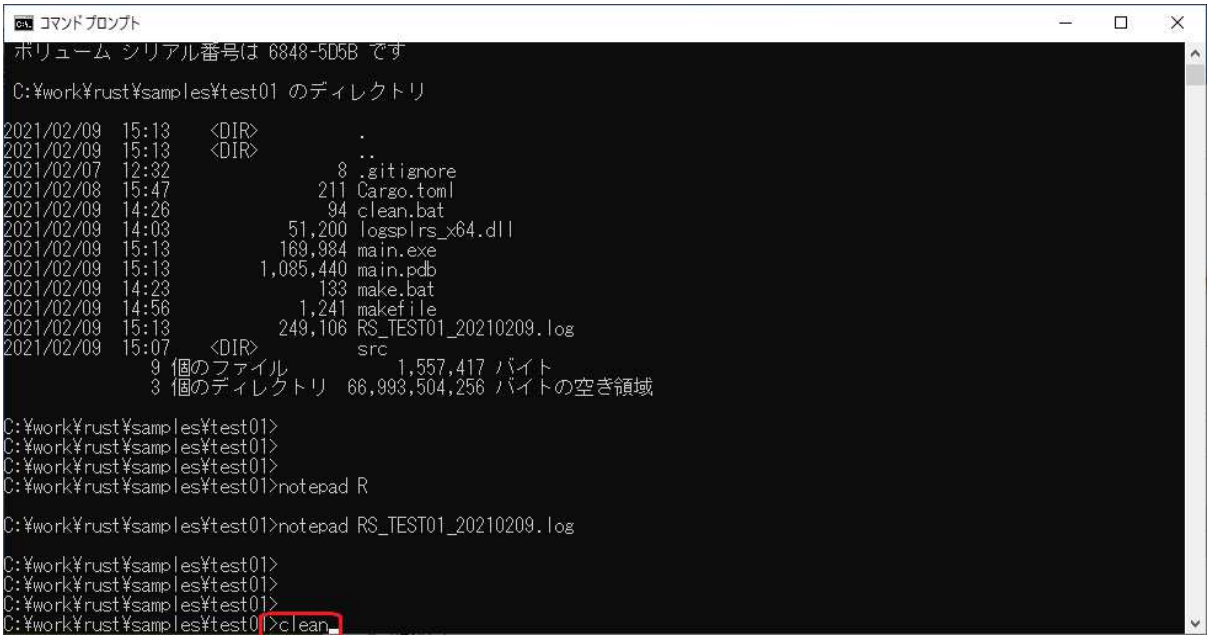
notepad R　と入力して、TABキーを押すと、RS_TEST_XXXXX.logのファイル名が補完されセットされますので、[Enter]キーで実行すると、メモ帳でlogが表示されます



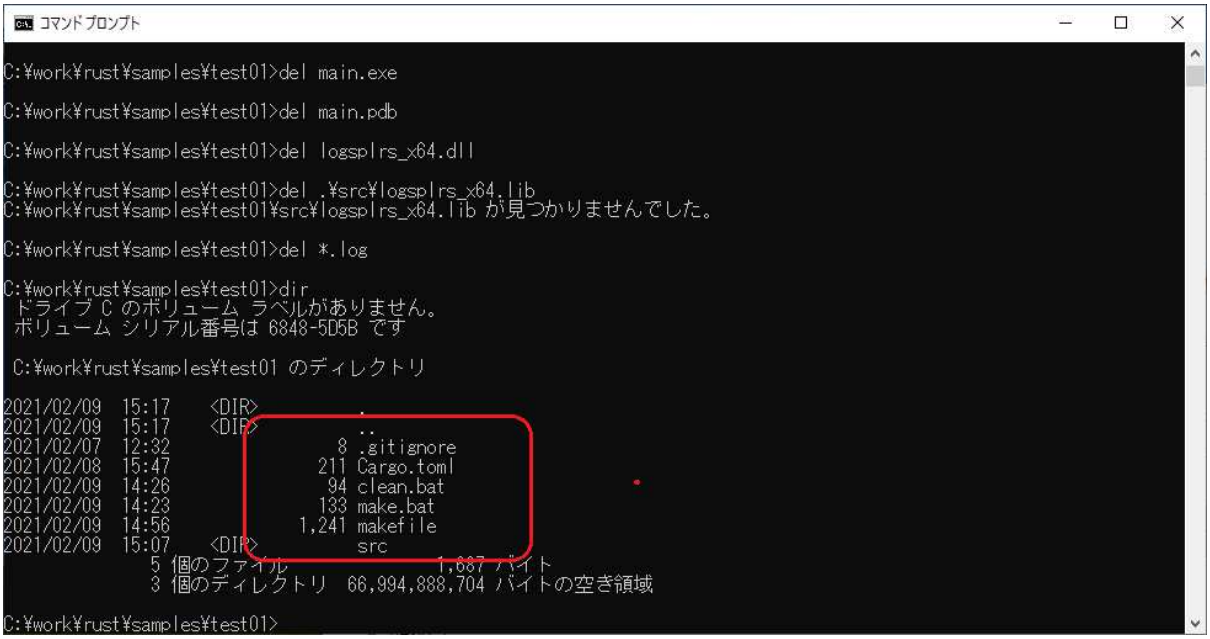
logを開いて、ファイルの最後まで移動したところ。このサンプルは2万行書き出します。



初期状態に戻すバッチ　clean.bat　があります。実行すると、exeファイルや一時的にコピーしたdll, libファイルも削除され、初期環境に戻ります。再度 make.batを実行すれば環境はビルド可能な状態に戻ります



clean.batを実行した直後の状態



1.2. nmake ユーティリティーを使ってサンプルをビルドする

nmake.exeを使ってのサンプル・プログラムのビルド

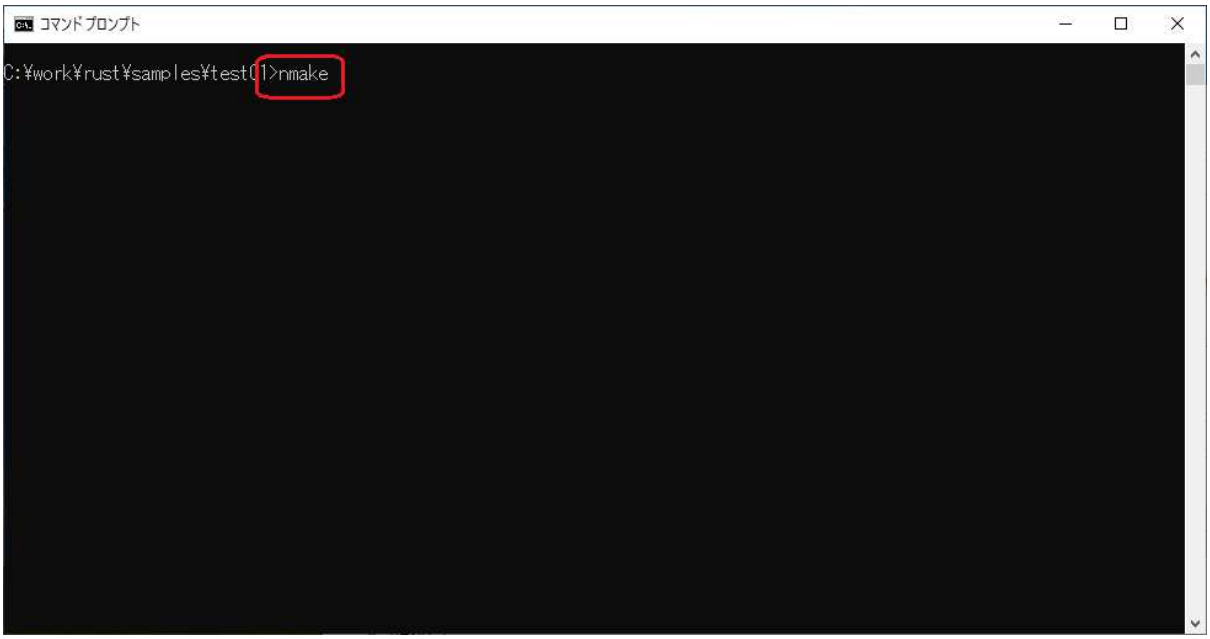
(注意　環境変数または、ファイルのコピーが必須です)

makefileの文字コードは、utf8にしております。

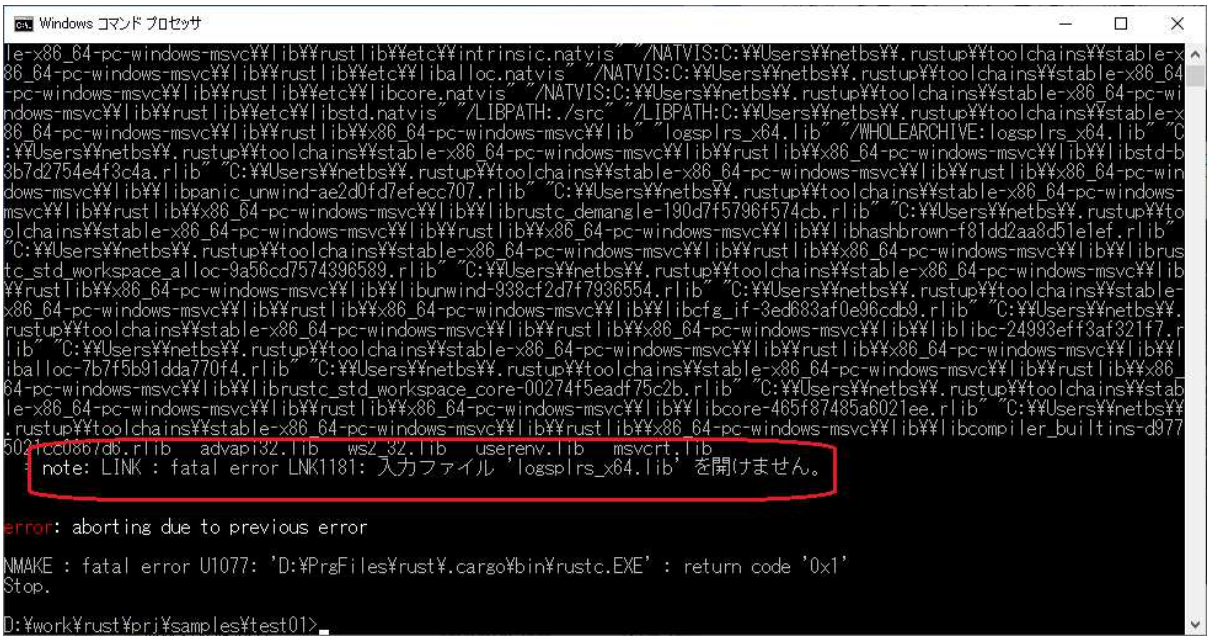
漢字が混在しなければ、utf8でも問題は出ないと思いますが、コメント等日本語が混在する場合は文字化けの発生が考えられます。　ASCII文字とutf8のASCIIは互換性がありますが、マルチバイト文字だと判断ができない場合があります。makefile内に漢字は混ぜないほうが無難です。

nmake.exeは、別途資料　**BuildToolsインストールマニュアル.pdf**　をご確認ください。nmake.exeのインストール位置、パスを通す方法と、パスが通った場所にコピーして利用する方法が記述されています。

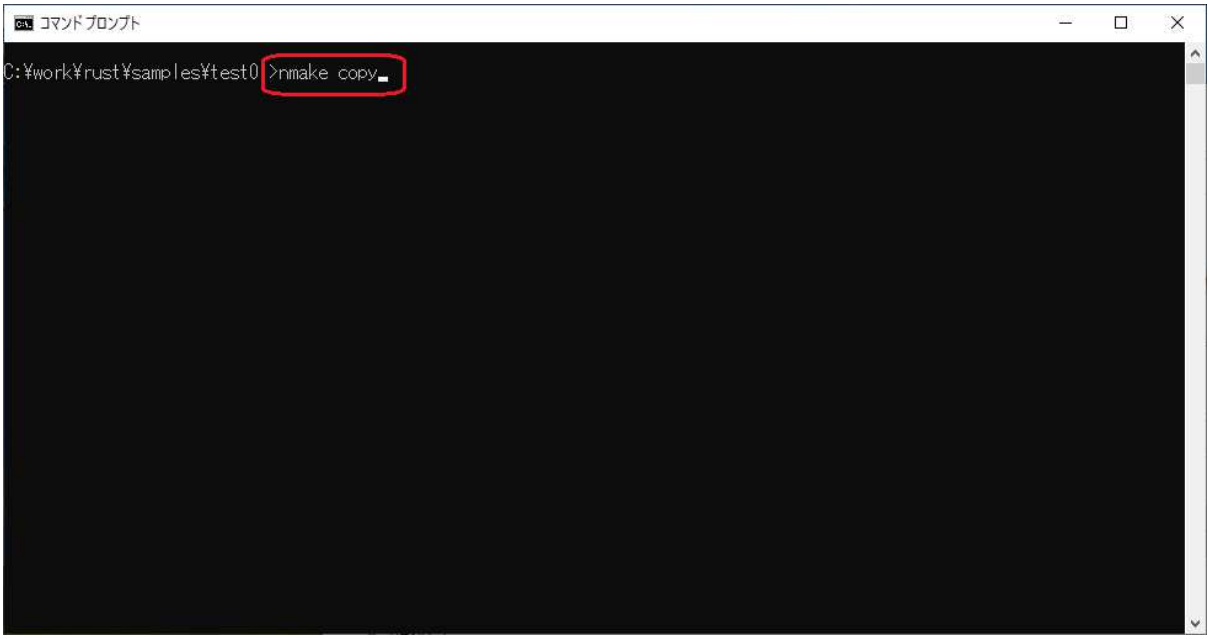
上記、nmake.exeを使う準備が整っていれば、namkeコマンドも使えます



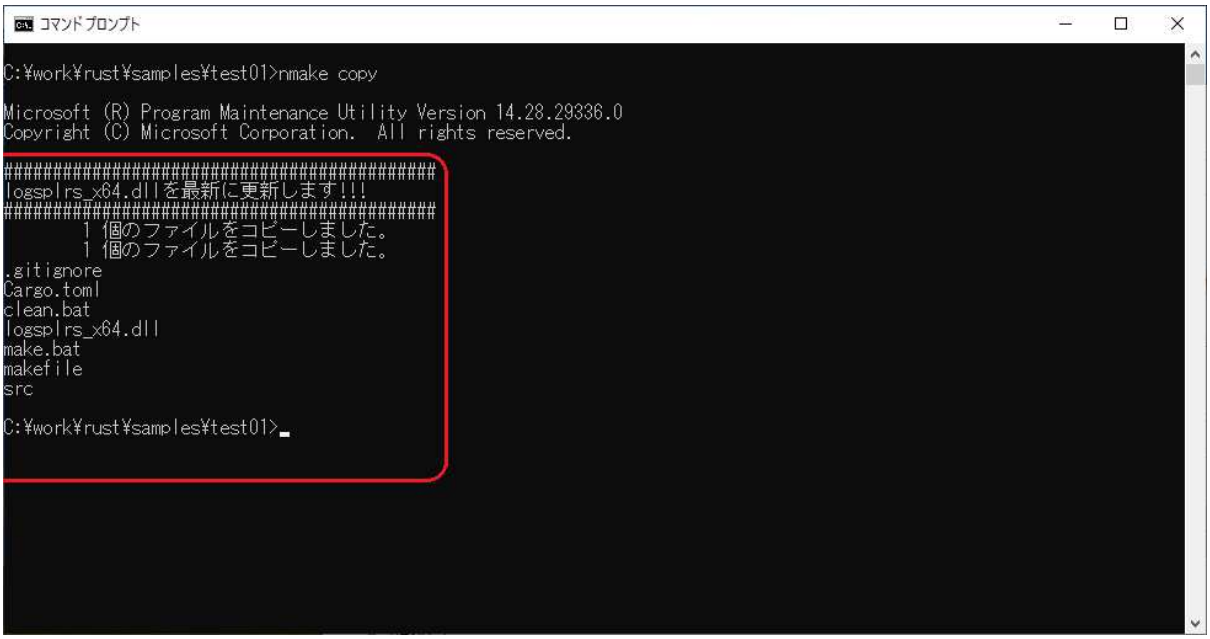
nmakeを実行すると、リンカーエラーが出ます。これは、Qzlogsのdll,　libファイルがテストプログラム環境に存在しないからです　**注　cp932(Shift_JIS)だと　logsplrs_x64.libを開けませんが化けます rustコンパイラの出力がutf8だからです**



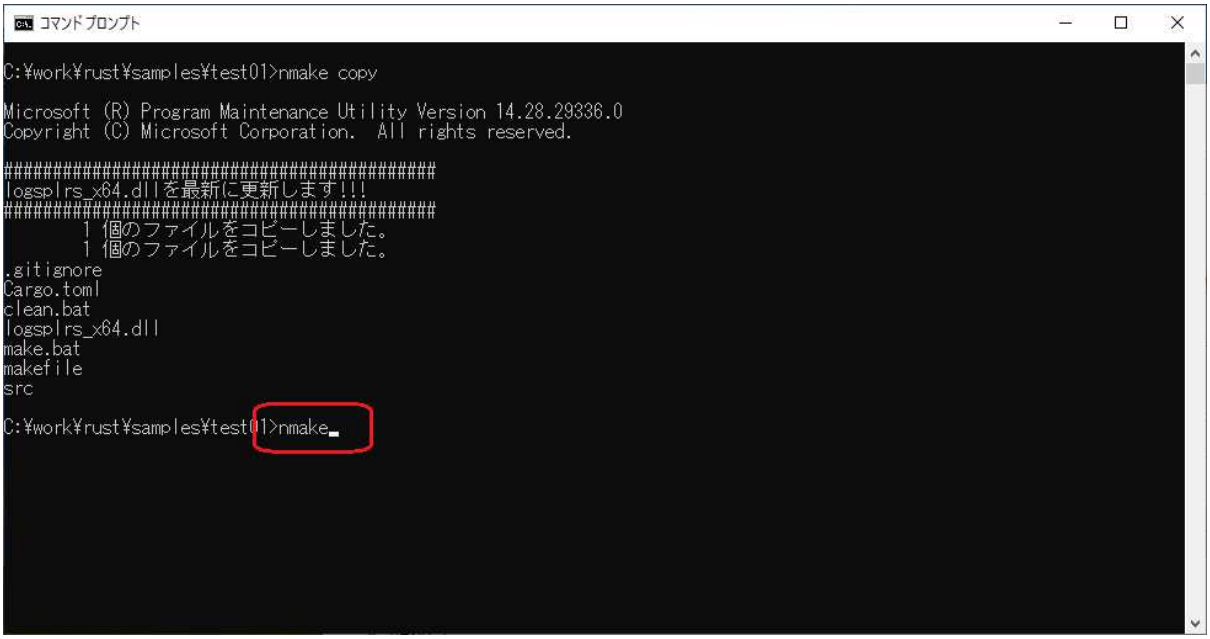
nmake copy [Enter] を実行します　makefileに書かれた、ライブラリファイルをコピーするコマンドメニューです



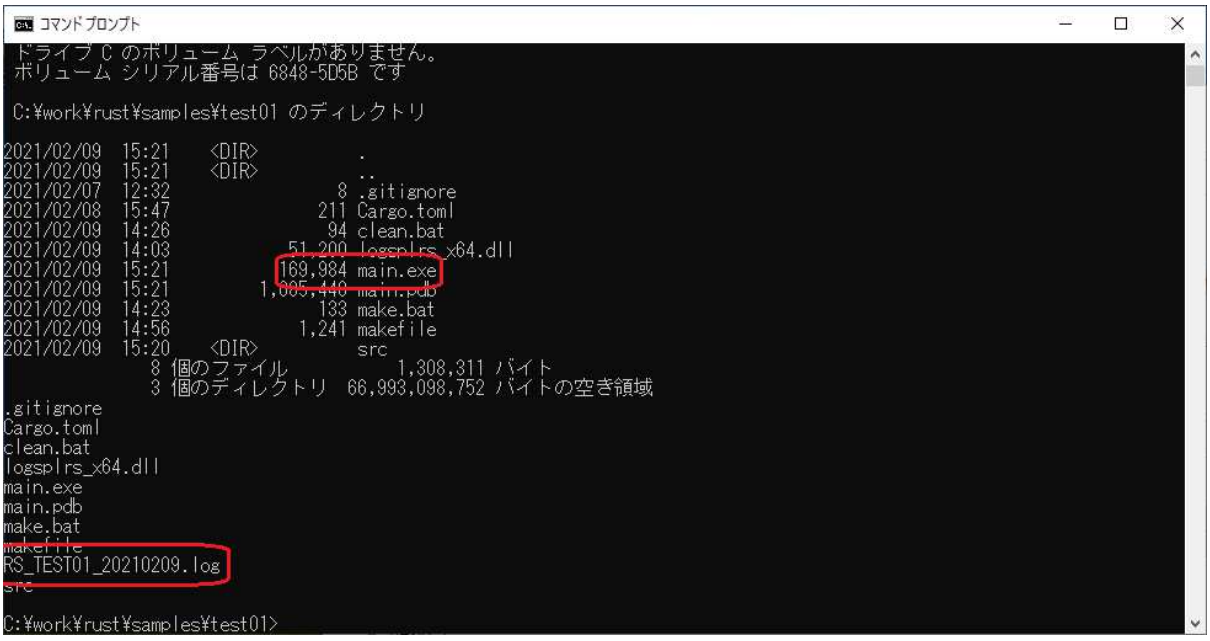
コピーが成功すれば、以下の状態になります



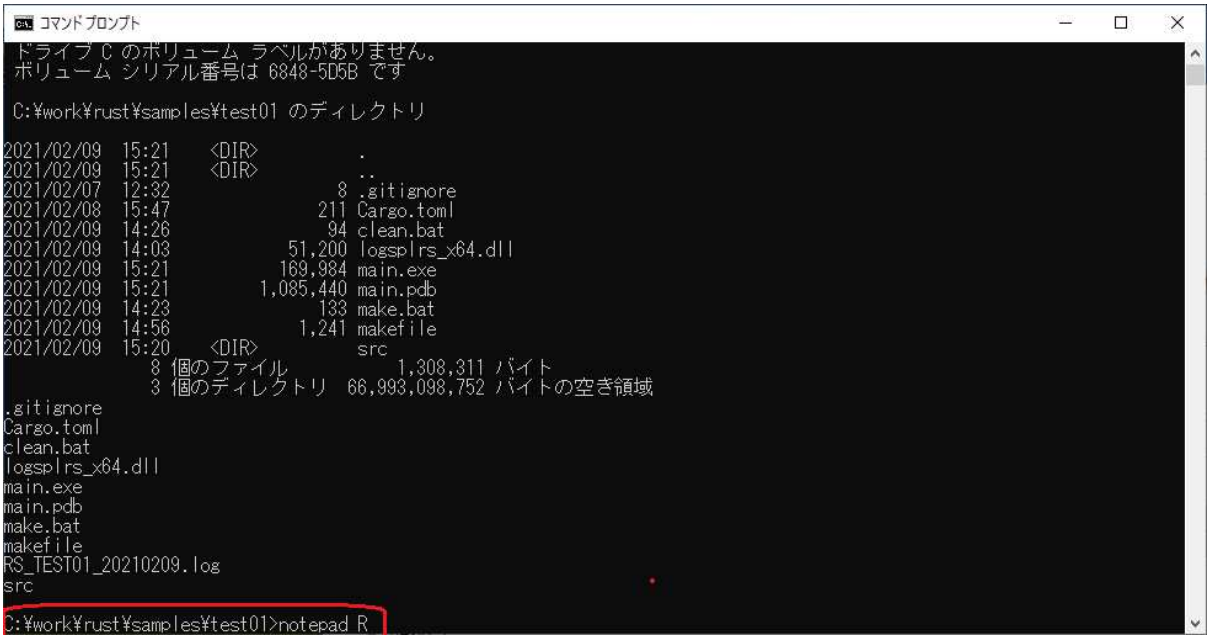
再度ビルドコマンドを実行します

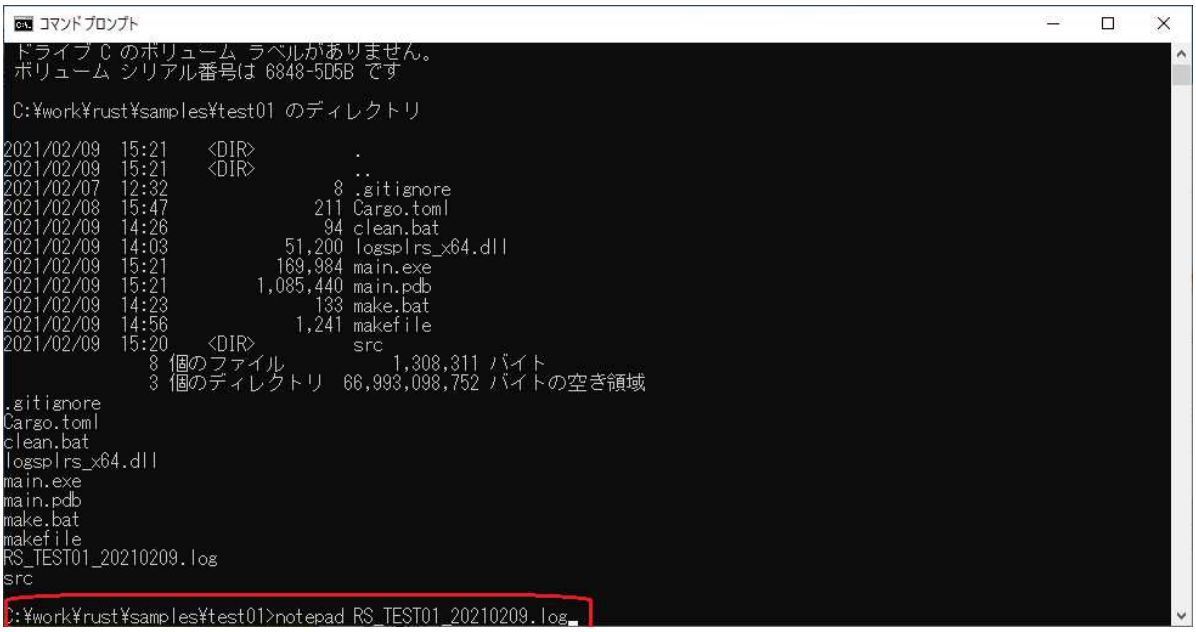


ビルドが成功すると、以下の状態になります

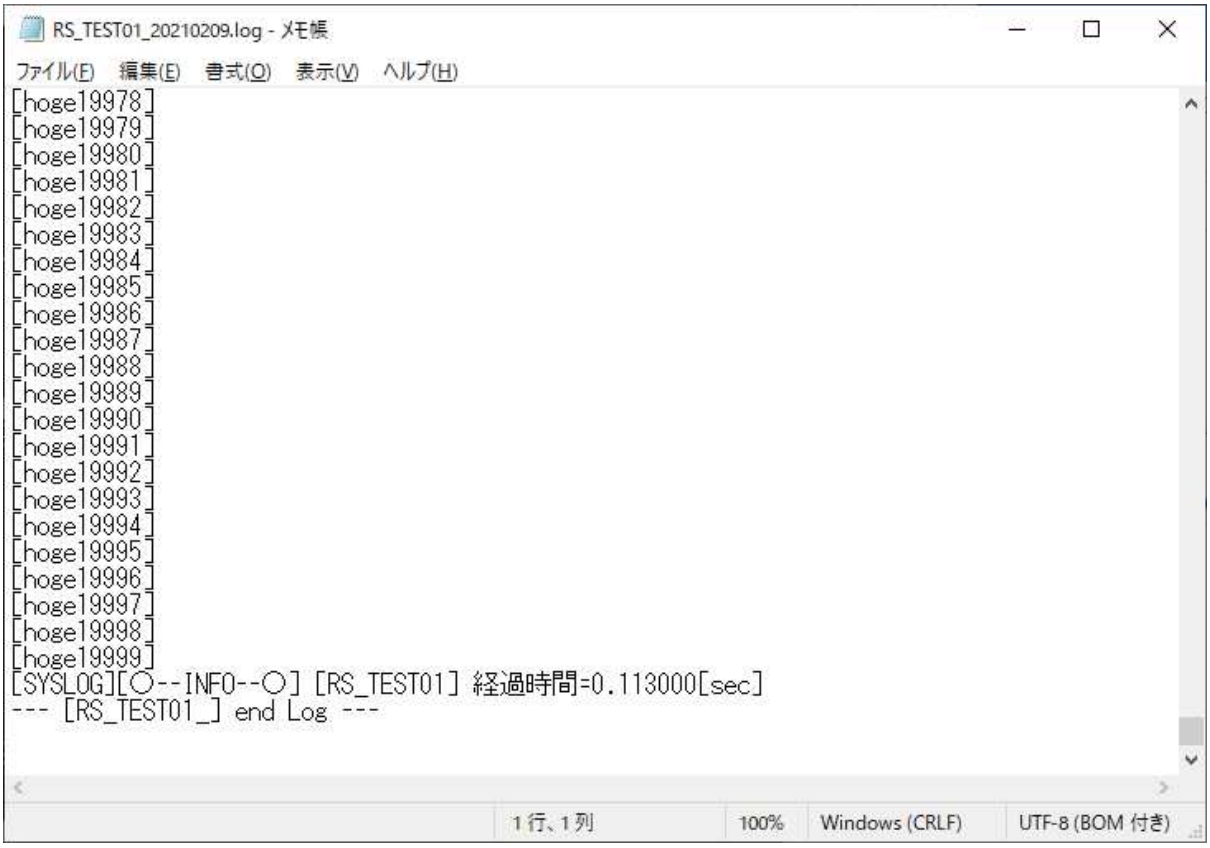


先ほどの例と同様に　notepad R [TABキー]を押してファイル名補完をして開きます

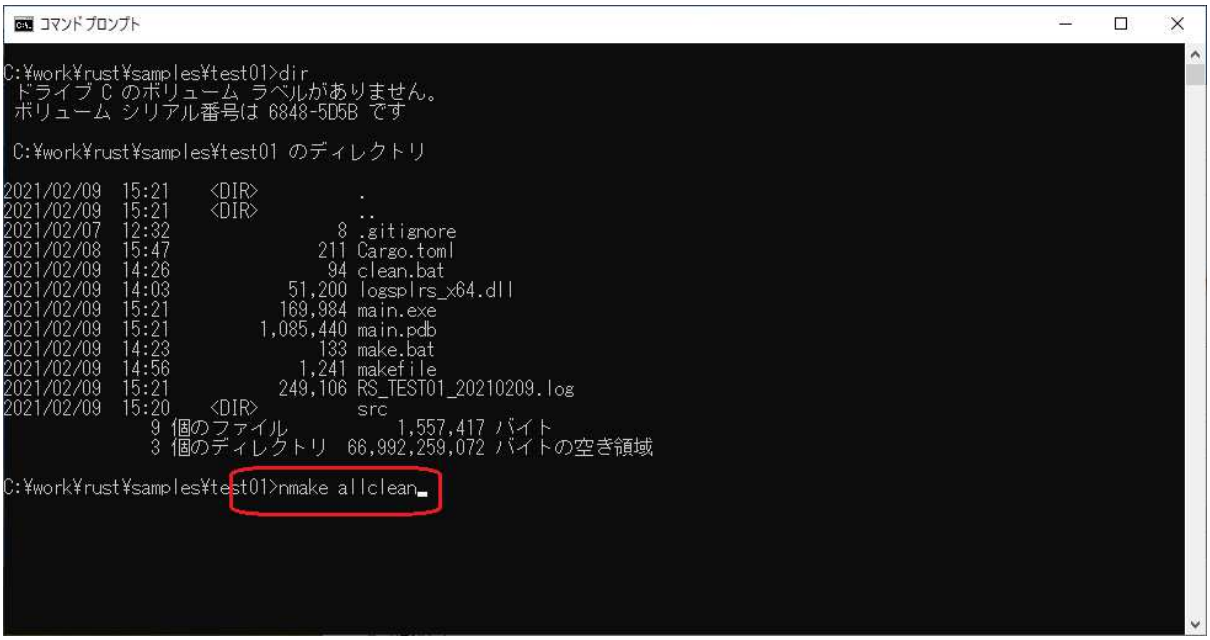




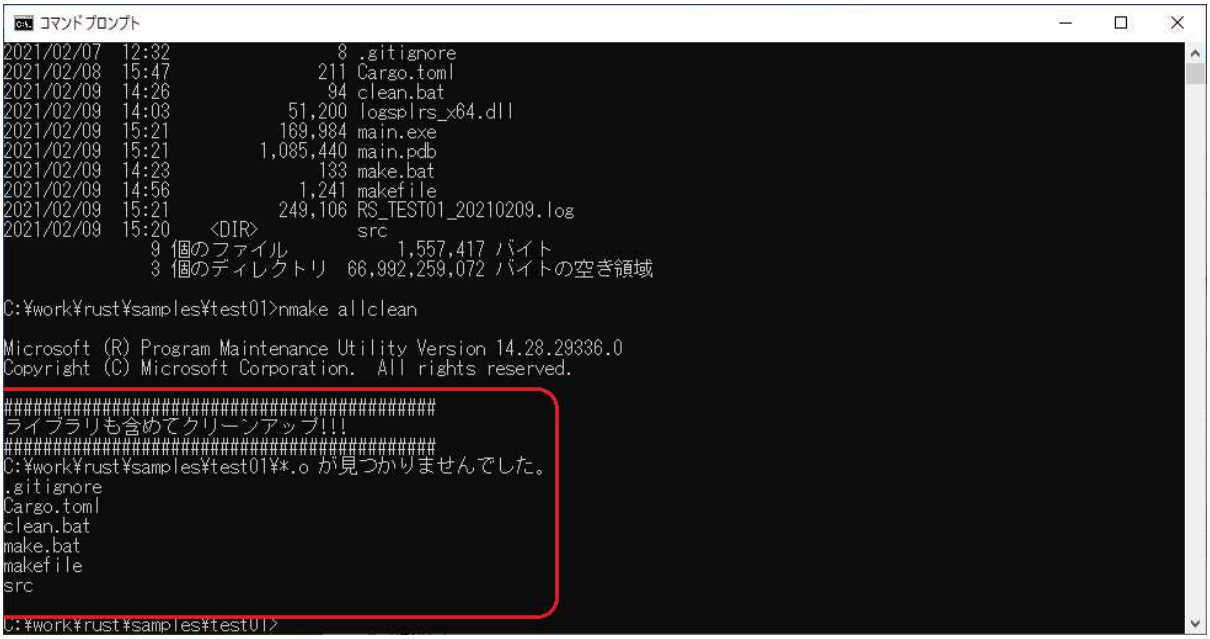
ログの終端を表示しています



nmake allclean を実行すると clean.bat を実行した状態と同じになります



nmake allcleanを実行後



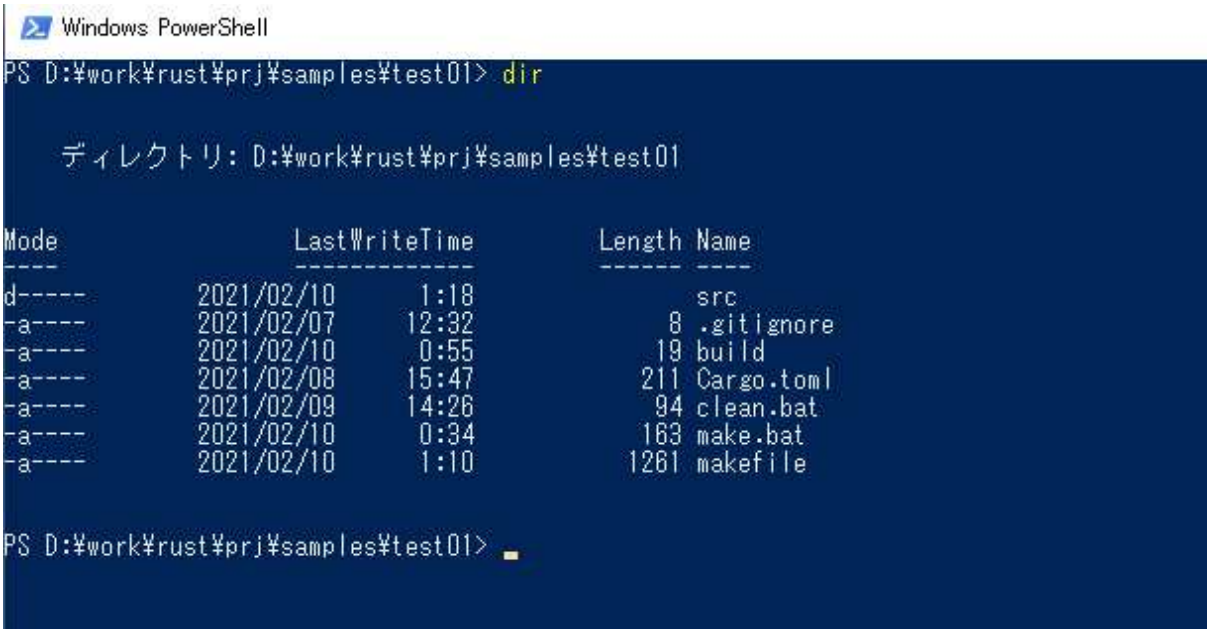
以下、その他のサンプルについても testXX フォルダに移動して、これまでの説明と同様にサンプルを作成してください。

注意
makefileとバッチファイルをカスタマイズする場合は、文字コードは Shift_JIS CP932で保存します。これ以外の文字コードだと動作は未定義です。VS Codeで開くとutf8だと勘違いされますので、メモ帳で開くのが無難です

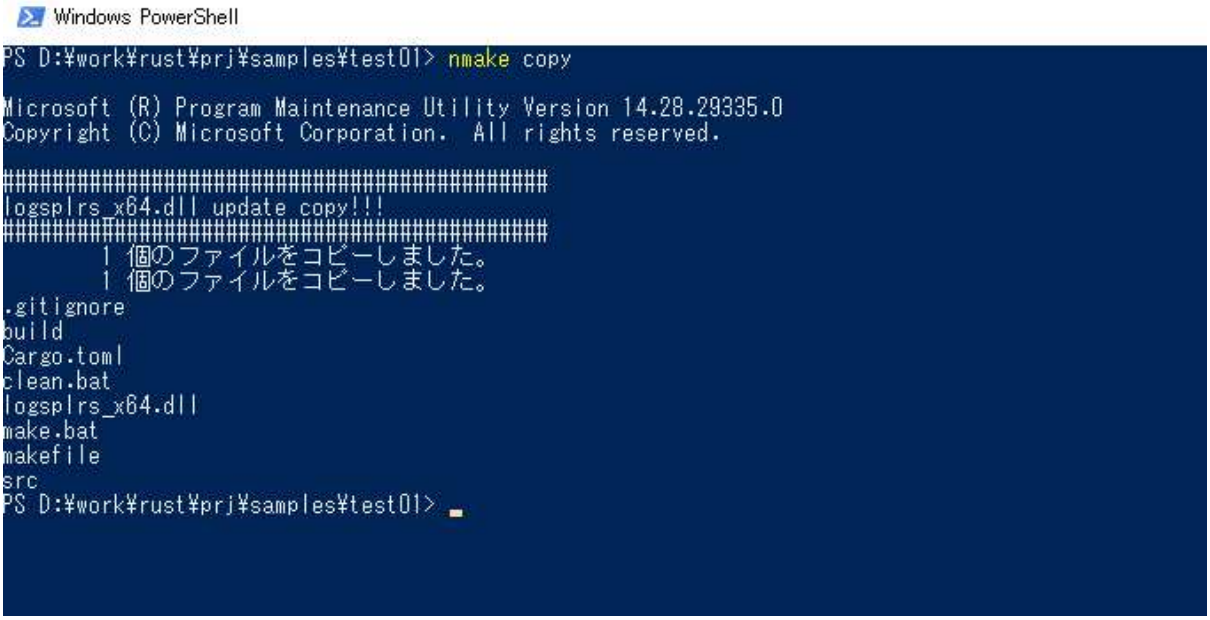
test03以降は、quizパッケージがsrcフォルダ内に追加された環境になります。
make.bat makefile ともに コピー先、ビルド対象位置が test01,test02とはすこしだけ異なりますのでご注意ください

1.3. PowerShellを利用する

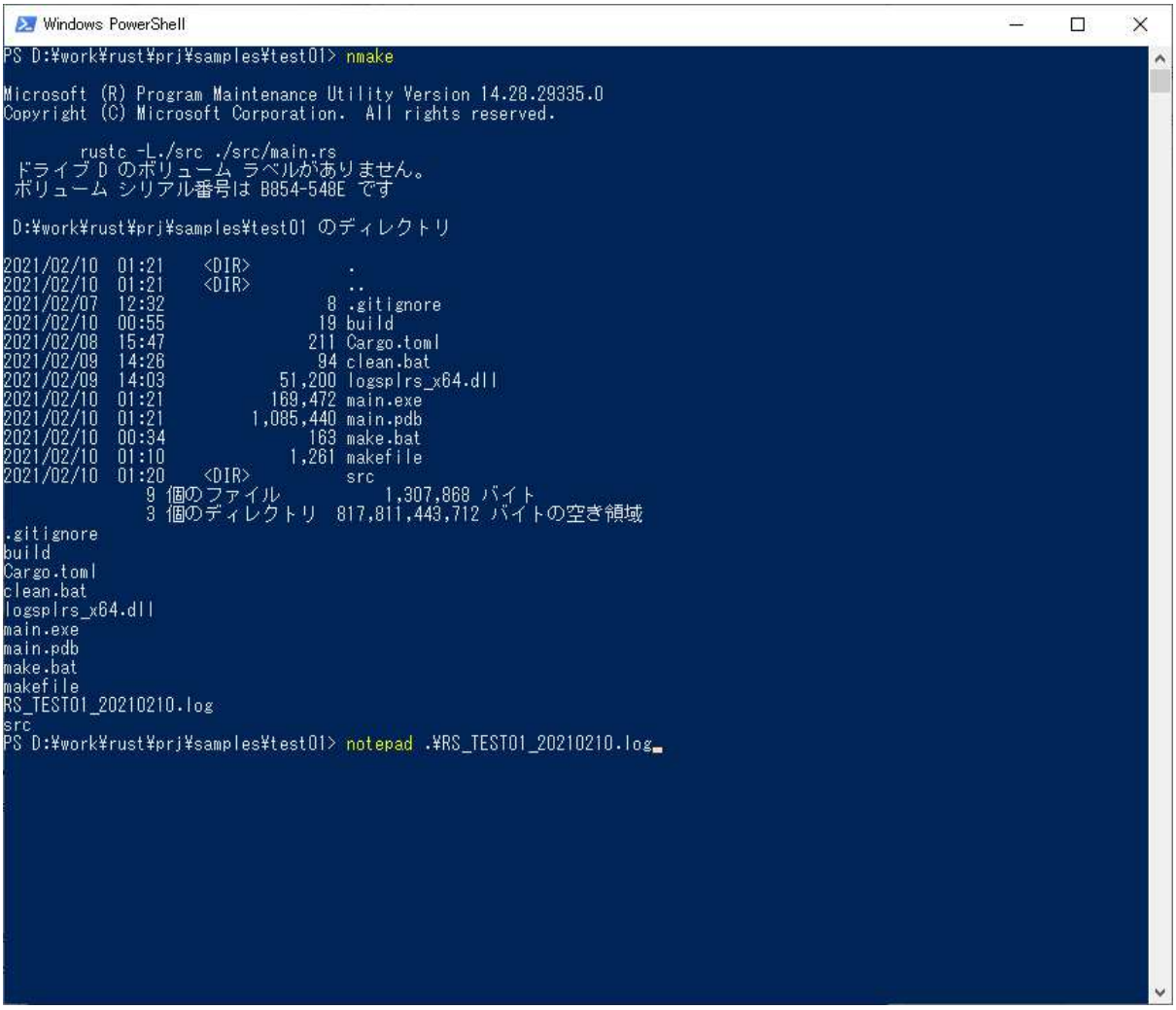
PowerShellでも同様にビルド可能です



nmake copy 実行　dll, libファイルを配置



ビルド成功　ログを見ます



最後の行を表示

PowerShellについても、文字コードは cp65001 (utf8)でご利用ください
学習後、ユーザが作成する新規プロジェクトはご自由に設定してください。Qzlogsの設定に縛られることはありません。

2. Qzlogsの初期化方法

2.1. v0.5.2.1 より前の初期化方法

初期化方法が v0.5.2.1より2パターンになりました。

いままで通りの log_int()パターン

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

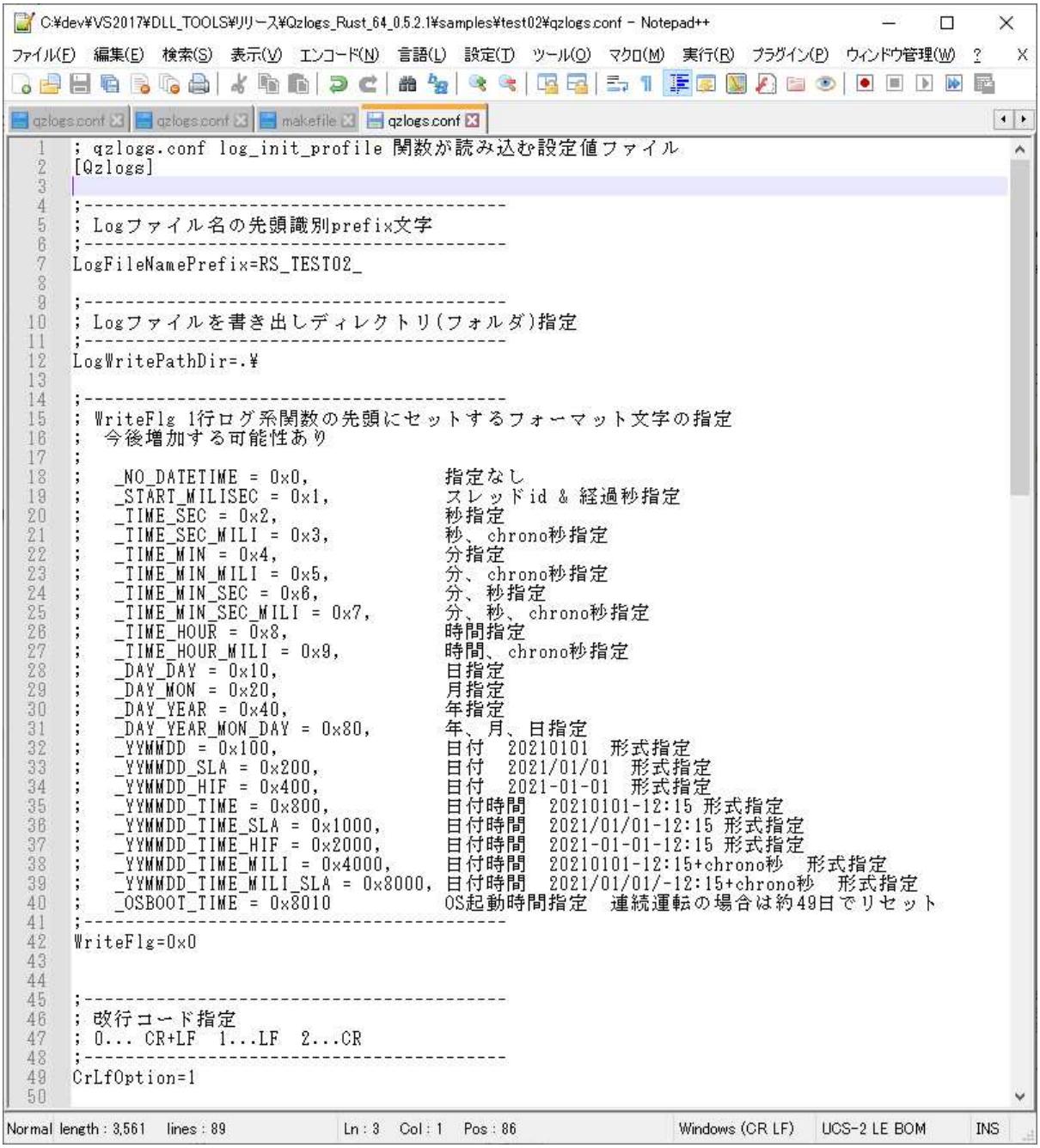
3. enumWriteFlg...ログ出力する際の、日付、時間、スレッドID, 処理時間などのオプションを
   選択します。詳細はマニュアル、ログ出力オプションを参照してください。
   (P12 3. 参照)
4. enumCrLfOpt ...改行コードの指定です。　0...CR+LF 1... CR, 2...LF となります。
5. enum LogSleepSw...ログ出力の開始、停止スイッチです。0...ログ出力　1...ログ・スリープ
   (記録しません)
6. LogLevelは 1-6の値で、出力するメソッドを制御します。数値に対応するログメソッド詳細
   は、2. 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。

2.2. 新しい初期化 プロファイル情報から初期化する
log_init_profile(qzlogs.confファイルの存在するパスを渡す)

新しい初期化は、プログラムにパラメータを埋め込む事無く、テキストファイルに設定した動作パラメータでlog動作します
P25のqzlogs.confの説明と、test01を除く、各サンプルに　qzlogs.conf　ファイルとして保存されています。
qzlogs.conf内部にはコメントがされていますので、コメントに従い設定値を与えてください
ファイル名は、ユーザが好きな名前を与えることができます。
hoge.ini でも、piyo.dat でもかまいません。管理しやすいと思われる名前をご利用ください



3. 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、rust版は、仕様から、スネークケース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 log_init ret 0...エラー ret 1....正常初期化	logsplrs_x64.dll logsplrs_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

No	公開メソッド名	機能	引数	備考
2	bool log_init_profile ret 0...エラー ret 1....正常初期化 v0.5.2.0追加機能	logsplrs_x64.dll logsplrs_x64.lib qzlogs.conf ファイルによる初期化	1. string conf path 設定ファイルのフルパス、ファイル名を含む文字列で呼び出します。	各、設定値の詳細はサンプル qzlogs.conf 参照のこと
2	log_term	終了処理。logInit系と必ず対に呼び出す必要がある	なし	
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、ログ記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	log_debug	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

No	公開メソッド名	機能	引数	備考
5	log_info	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	log_warning	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	log_error	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
8	log_critical	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	log_sys ※非公開	システムが利用する一行ログ。ユーザ操作不可。		
10	log_dump_utf8	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

No	公開メソッド名	機能	引数	備考
11	log_dump_raw v 0.5.2.0追加 **※バイナリダンプでは、右側のキャラクタ表示位置でマルチバイトは表示しません。Ascii文字のみの表示対応です**	ダンプ形式の出力 NULLを含むバイナリダンプ用	3. []byte data int size 4. 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
12	log_put **※ 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口出力する	1. string data 2. int size 3. bool header int nextLineCnt	最大サイズは8100バイトint sizeはUNICODE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。
13	log_put_hex	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。
14	w32_err_log	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、ユーザプログラムのエラーチェッしたい行番号を入力

No	公開メソッド名	機能	引数	備考
15	w32_reset_err_code	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
16	makebit_string_us16	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。
17	makebit_string_u32	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文字などをセト

No	公開メソッド名	機能	引数	備考
18	makebit_string_unicode_scalar_value	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
19	msg_box	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 msg_box_okcancel	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	profile_time_start 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. 5string 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	profile_time_stop 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Rust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。

4. log_init_profile関数　qzlogs.conf説明

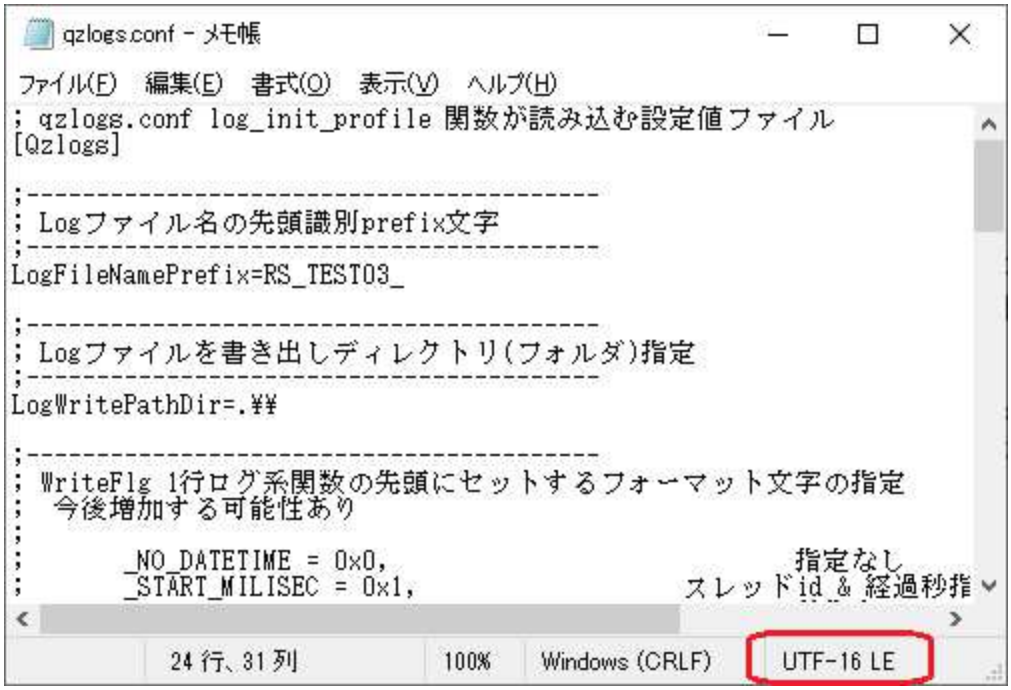
2.1. log_init_profile関数は、ログ初期化を　qzlogs.conf (ファイル名は自由に定義可)
　　から各設定値を読み込み、qzlogs動作定義としてセットします。

　qzlogs.conf はどこの場所にあってもかまいません。ただ、ネットワークドライブのUNCパスはうけつけません。　\\SERVER_NAME\disk\tmp みたいな記述はだめです。　\\SERVER_NAME\disk を、エクスプローラで、ネットワークドライブの割り当てを行い、ドライブレターを XとかZとかにセットすれば、ネットワークドライブ上のqzlogs.confを読みだせます。　上記の例で言うと、X:\\tmp となります。

あとは、qzlogsのRustラップしたプログラムと、logsplrs_x64.dll, logsplrs_x64.lib ファイルが読み込めれば動作します。

本関数の便利なところは、プログラムをrustコンパイルしなおさなくとも、ログの出力定義を変更できるところです。

以下は、qzlogs.conf ファイルの内容を説明しています。直接エディタで編集してください。**保存時の文字コードの注意ですが、Windowsは内部でUNICODE (UTF16を利用しています。UTF16_LE　で保存するようにしてください。**
メモ帳で開くと、UTF-16 LEが確認できます。
makefileやmake.bat, rustのプログラムはutf8で保存されていますので、ご注意ください。



と

また、初期化関数は、本関数または、log_init()のどちらかひとつのみの利用となります。ふたつの関数で初期化した場合は、最初に初期化成功した関数の定義で動作します。2回目の初期化は無視されます。

qzlogs.confの　セミコロンは　コメント行になります。

◇　LogFileNamePrefix 定義

```
;----------------------------------------
; Logファイル名の先頭識別prefix文字
;----------------------------------------
LogFileNamePrefix=TEST02_
```

◇　LogWritePathDir　定義

```
;----------------------------------------
; Logファイルを書き出しディレクトリ(フォルダ)指定
;----------------------------------------
LogWritePathDir=D:\PrgFiles\rust\prj\samples\test02
```

◇　WriteFlg　定義

```
;----------------------------------------
; WriteFlg 1行ログ系関数の先頭にセットするフォーマット文字の指定
;  今後増加する可能性あり
;
;          _NO_DATETIME = 0x0,                    指定なし
;          _START_MILISEC = 0x1,                  スレッドid & 経過秒指定
;          _TIME_SEC = 0x2,                       秒指定
;          _TIME_SEC_MILI = 0x3,                  秒、chrono秒指定
;          _TIME_MIN = 0x4,                       分指定
;          _TIME_MIN_MILI = 0x5,                  分、chrono秒指定
;          _TIME_MIN_SEC = 0x6,                   分、秒指定
;          _TIME_MIN_SEC_MILI = 0x7,              分、秒、chrono秒指定
;          _TIME_HOUR = 0x8,                      時間指定
;          _TIME_HOUR_MILI = 0x9,                 時間、chrono秒指定
;          _DAY_DAY = 0x10,                       日指定
;          _DAY_MON = 0x20,                       月指定
;          _DAY_YEAR = 0x40,                      年指定
;          _DAY_YEAR_MON_DAY = 0x80,              年、月、日指定
;          _YYMMDD = 0x100,                       日付　20210101　形式指定
;          _YYMMDD_SLA = 0x200,                   日付　2021/01/01　形式指定
;          _YYMMDD_HIF = 0x400,                   日付　2021-01-01　形式指定
;          _YYMMDD_TIME = 0x800,                  日付時間　20210101-12:15 形式
指定
```

;　　_YYMMDD_TIME_SLA = 0x1000,　　日付時間　2021/01/01-12:15 形式指定
;　　_YYMMDD_TIME_HIF = 0x2000,　　日付時間 2021-01-01-12:15 形式指定
;　　_YYMMDD_TIME_MILI = 0x4000,　　日付時間 20210101-12:15+chrono秒　形式指定
;　　_YYMMDD_TIME_MILI_SLA = 0x8000,　日付時間 2021/01/01/-12:15+chrono秒　形式指定
;　　_OSBOOT_TIME = 0x8010　　OS起動時間指定　連続運転の場合は約49日でリセット
;----------
WriteFlg=0x8010

◇　CrLfOption　定義
;----------
; 改行コード指定
; 0... CR+LF 1...LF 2...CR
;----------
CrLfOption=1

◇　LogOutPutLevel　定義
;----------
; ログ出力レベル定義
; 5...LOG_DEBUG出力 4...LOG_INFO出力　3...LOG_WARNING出力
; 2...LOG_ERROR出力 1...LOG_CRITICAL出力
;----------
LogOutPutLevel=5

◇　LogSleepSwitch　定義
;----------
; ログ Run 停止スイッチ
; 0...Run　1...Sleep
;----------
LogSleepSwitch=0

◇　BomSetFlg　定義
;----------
; Bom セットフラグ　0...UTF8にbomセットしない　1...UTF8 Bomをセットする
;----------
BomSetFlg=0

5. 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
23	_OSBOOT_TIME	0x8010	OS起動時間指定　連続運転の場合は約49日でリセット

6. 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。VCの場合、デバッグとリリースでは動作が完全同一になりません。デバッグで動作してもリリースで動かないなんてことがよくあります。そんなとき、デバッグモードは使わずにシンプルに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ので、デバッグモードは不要になりました。rustに関して言えば、コンパイル時の最適化の違い、デバッグモードでコンパイルする事になりますが、そのあたりのrustc コンパイラの仕様はrust本家サイトのドキュメントをご確認ください。

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”)

println!(....

qzlogs::log(“2”)

println!(...

qzlogs::log(“3”)

println!(....
```

Qzlogsライブラリを使い、rustプログラミングをお楽しみください。

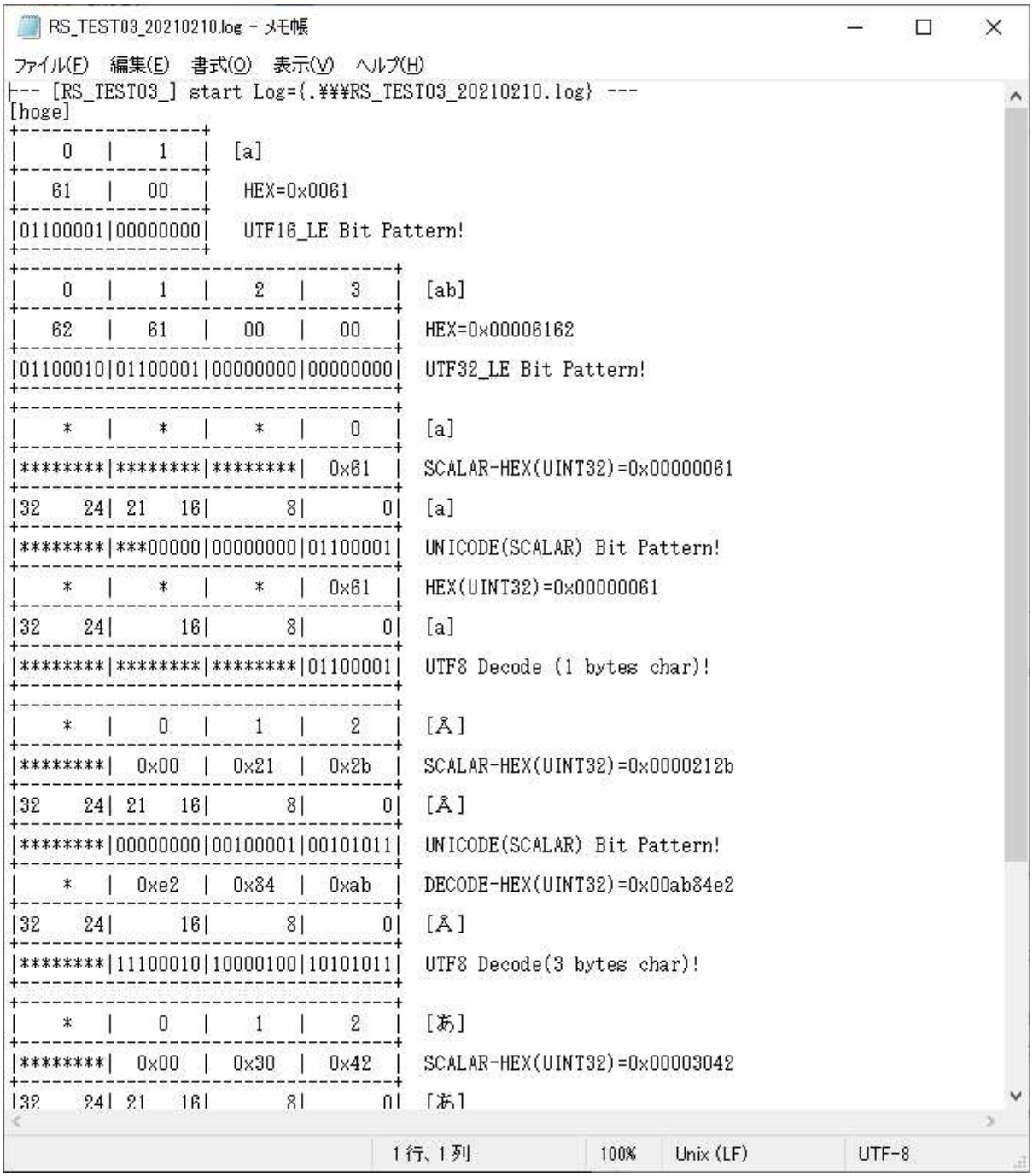
7. 実装例など

7.1. ビット文字列表示メソッド

MakeBitStringUSHORT16(), MakeBitStringUINT32(), MakeBitStringUnicodeScalarValue()

サンプル test03 main.rs

書かれた、ログのイメージ。



```
RS_TEST03_20210210.log - メモ帳
ファイル(F)  編集(E)  書式(O)  表示(V)  ヘルプ(H)
+--- [RS_TEST03_] start Log=(.¥¥¥RS_TEST03_20210210.log) ---
[hoge]
+-----------------+
|    0    |    1    |  [a]
+-----------------+
|   61    |   00    |   HEX=0x0061
+-----------------+
|01100001|00000000|   UTF16_LE Bit Pattern!
+-----------------+
+-----------------------------------+
|    0    |    1    |    2    |    3    |  [ab]
+-----------------------------------+
|   62    |   61    |   00    |   00    |  HEX=0x00006162
+-----------------------------------+
|01100010|01100001|00000000|00000000|  UTF32_LE Bit Pattern!
+-----------------------------------+
+-----------------------------------+
|    *    |    *    |    *    |    0    |  [a]
+-----------------------------------+
|********|********|********|  0x61   |  SCALAR-HEX(UINT32)=0x00000061
+-----------------------------------+
|32    24| 21   16|       8|       0|  [a]
+-----------------------------------+
|********|***00000|00000000|01100001|  UNICODE(SCALAR) Bit Pattern!
+-----------------------------------+
|    *    |    *    |    *    |  0x61   |  HEX(UINT32)=0x00000061
+-----------------------------------+
|32    24|      16|       8|       0|  [a]
+-----------------------------------+
|********|********|********|01100001|  UTF8 Decode (1 bytes char)!
+-----------------------------------+
+-----------------------------------+
|    *    |    0    |    1    |    2    |  [Å]
+-----------------------------------+
|********|  0x00   |  0x21   |  0x2b   |  SCALAR-HEX(UINT32)=0x0000212b
+-----------------------------------+
|32    24| 21   16|       8|       0|  [Å]
+-----------------------------------+
|********|00000000|00100001|00101011|  UNICODE(SCALAR) Bit Pattern!
+-----------------------------------+
|    *    |  0xe2   |  0x84   |  0xab   |  DECODE-HEX(UINT32)=0x00ab84e2
+-----------------------------------+
|32    24|      16|       8|       0|  [Å]
+-----------------------------------+
|********|11100010|10000100|10101011|  UTF8 Decode(3 bytes char)!
+-----------------------------------+
+-----------------------------------+
|    *    |    0    |    1    |    2    |  [あ]
+-----------------------------------+
|********|  0x00   |  0x30   |  0x42   |  SCALAR-HEX(UINT32)=0x00003042
+-----------------------------------+
|32    24| 21   16|       8|       0|  [あ]
1 行、1 列    100%    Unix (LF)    UTF-8
```

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。

7.2. Win32エラーコードとエラー文字のロギング

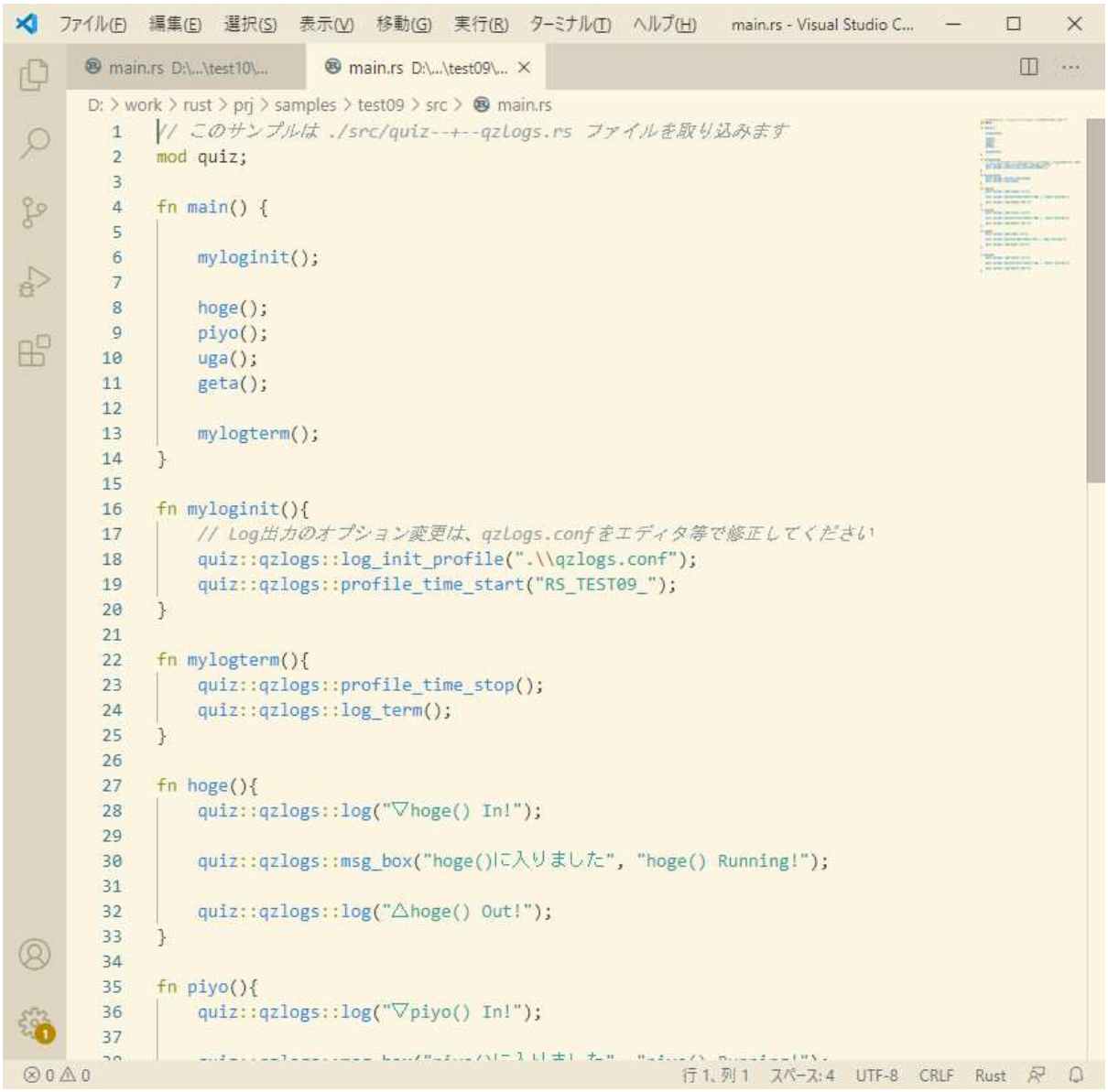
Windows OS内で発生した、W32エラー文字列を取得します。
これは、使うタイミングが難しいと思いますが、なにかのときに役に立ててください。

test05 main.rs

7.3. 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　test09 main.rs



```rust
// このサンプルは ./src/quiz--+--qzlogs.rs ファイルを取り込みます
mod quiz;

fn main() {

    myloginit();

    hoge();
    piyo();
    uga();
    geta();

    mylogterm();
}

fn myloginit(){
    // log出力のオプション変更は、qzlogs.confをエディタ等で修正してください
    quiz::qzlogs::log_init_profile(".\\qzlogs.conf");
    quiz::qzlogs::profile_time_start("RS_TEST09_");
}

fn mylogterm(){
    quiz::qzlogs::profile_time_stop();
    quiz::qzlogs::log_term();
}

fn hoge(){
    quiz::qzlogs::log("▽hoge() In!");

    quiz::qzlogs::msg_box("hoge()に入りました", "hoge() Running!");

    quiz::qzlogs::log("△hoge() Out!");
}

fn piyo(){
    quiz::qzlogs::log("▽piyo() In!");
```

ログの結果

```
--- [RS_TEST09_] start Log=(.¥¥RS_TEST09_20210210.log) ---
[▽hoge() In!]
[△hoge() Out!]
[▽piyo() In!]
[△piyo() Out!]
[▽uga() In!]
[△uga() Out!]
[▽geta() In!]
[△geta() Out!]
[SYSLOG][○--INFO--○] [RS_TEST09_] 経過時間=2.108000[sec]
--- [RS_TEST09_] 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。

7.4. 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。
初期リリースの初期化　log_initの場合は
log_init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。

log_init_profileの場合は　qzlogs.conf 内の LogSleepSwitch=0にセットすればLogを出力します。

test07 main.rs

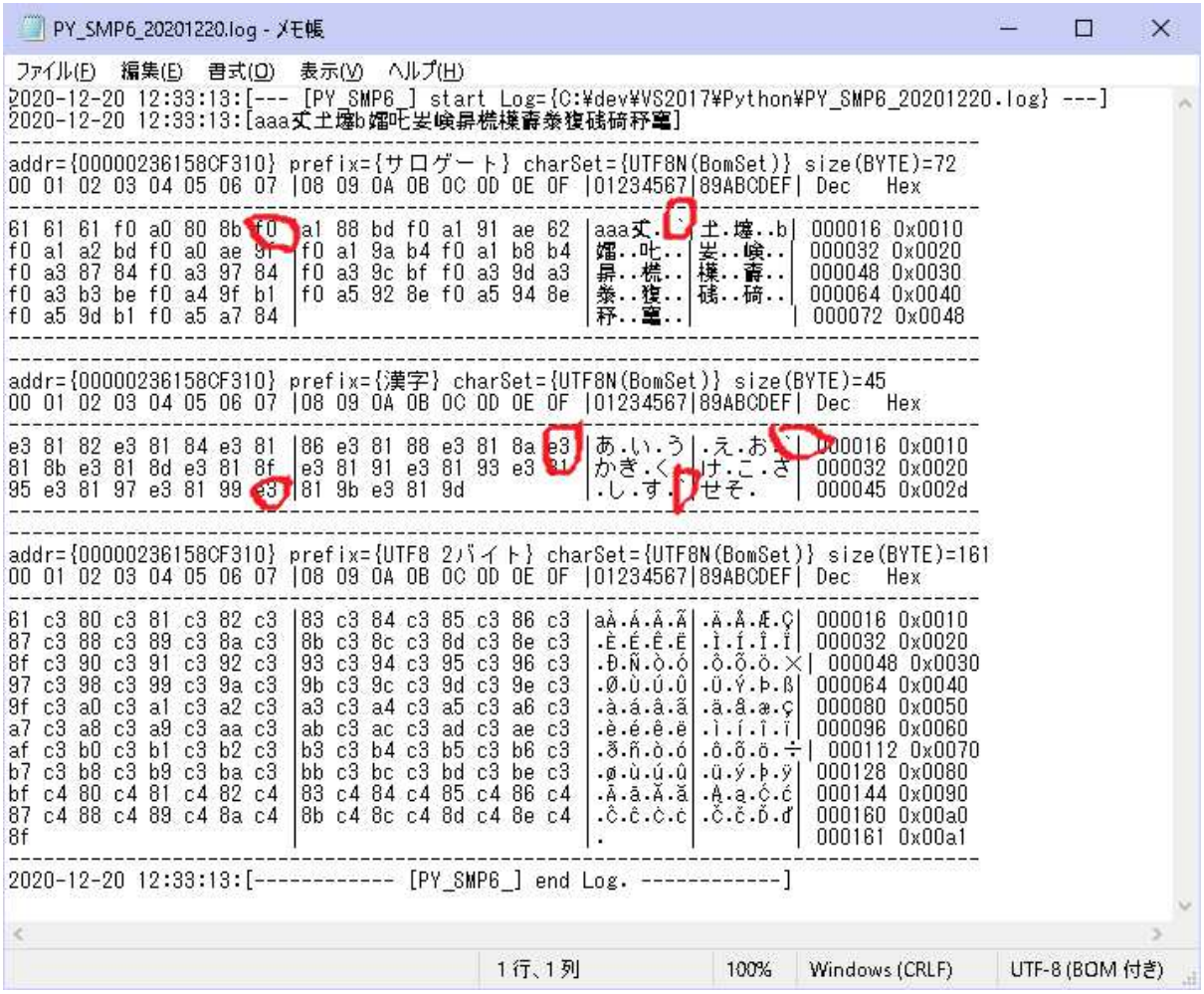
7.5. UTF8 文字列ダンプ時の禁則処理の説明

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)



8. サンプルプログラム

9. 仕様

9.1. 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)

9.2. 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

9.3. 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

9.4. スレッド
　　rust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

9.5. フォルダの自動作成
　　log_init, log_init_profile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

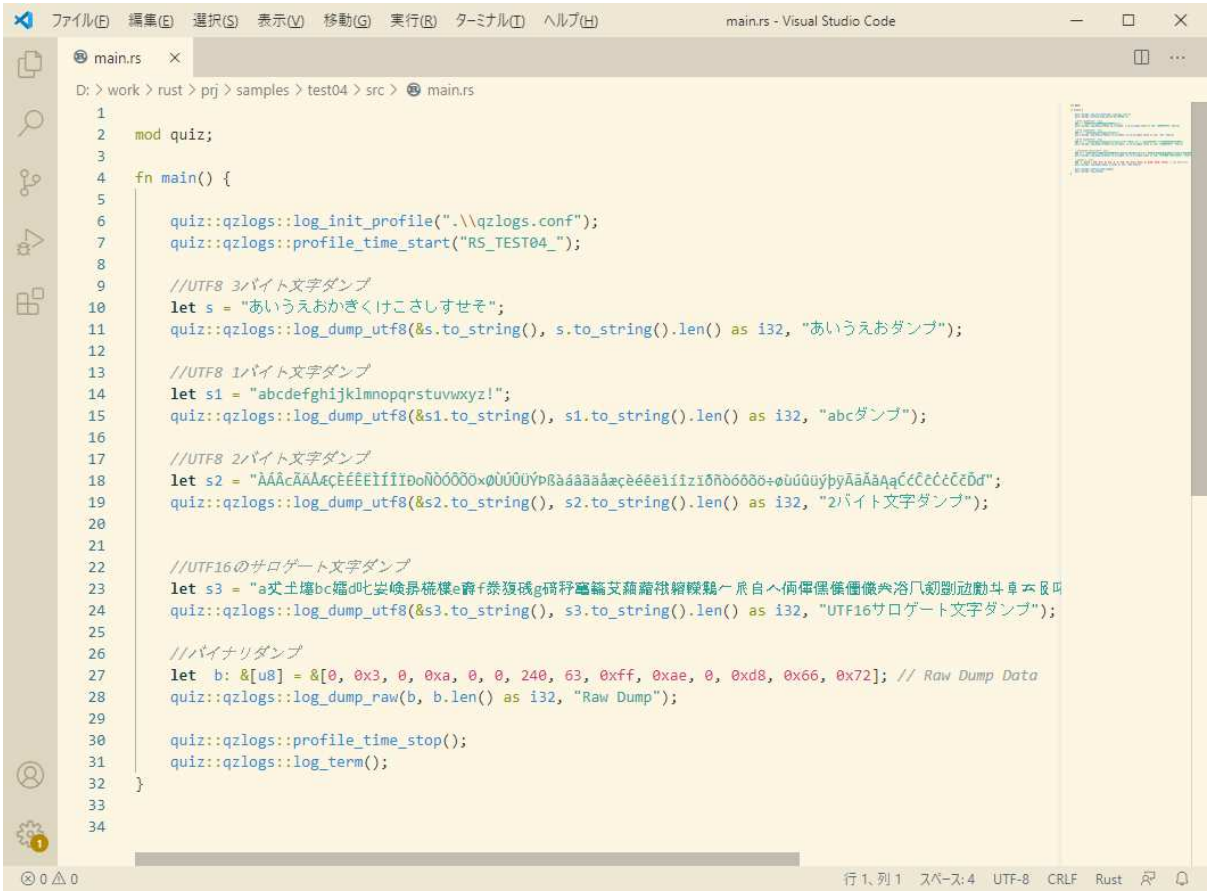
9.6. ログ・ローテート
　　シンプル版ではサポートされていません

9.7. 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

9.8. 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

10. サンプルプログラム抜粋

サンプル　test04　main.rs



11.　Rustのプログラム配置、フォルダに配置したrsファイルの利用など

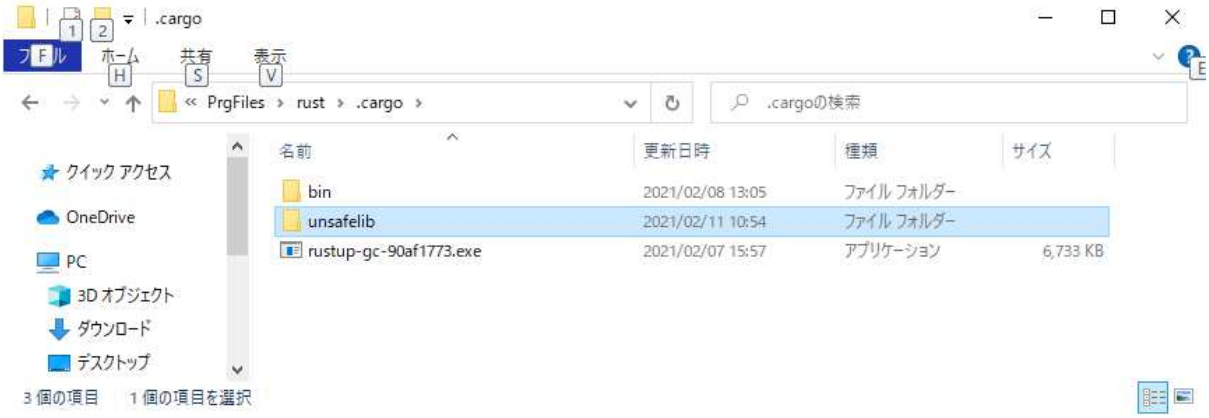
11.1. test01, test02サンプルは、main.rsと同じフォルダにqzlogs.rsファイルを配置し
mod qzlogs: でモジュール利用宣言をしています。
この場合　qzlogs::log_init_profile(...)の呼び出し方法となります

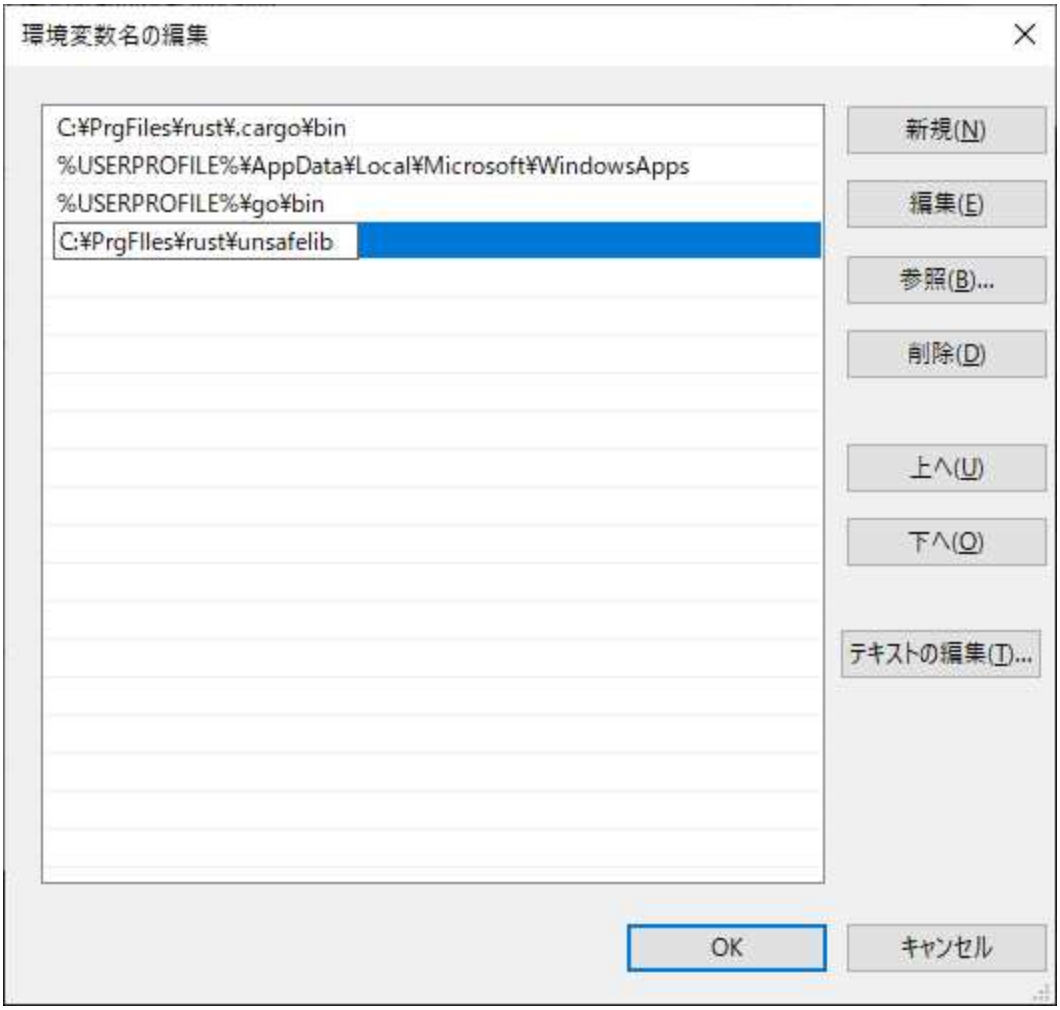
11.2. test03以降は、 /src/quiz/qzlogs.rs に保持されたqzlogs.rsを利用しています。
関数呼び出しで quiz::qzlogs::log_init_profile(...) と、quiz:: が追加されて呼び出すことになります

11.3. /src/quiz/qzlogs.rsを呼び出す場合、 mod.rsがquizフォルダに存在しなければなりません。また mod.rsで　pub mod qzlogs; の行が必須になります。これはrustの仕様です

11.4. 独自のプログラムを拡張する場合は、cargo new your_prject_name で作成したプロジェクトのフォルダに、上記の説明を参考に、Qzlogsの配置をしてください

11.5. logsplrs_x64.dll, logsplrs_x64.libファイルを /srcなどのフォルダに含めたくない場合は
rustのインストール先に unsafelib フォルダを作成し、ここにdll, libファイルをコピーし、さらにパスを設定します。PC　右クリック-->システムの詳細設定-->システムのプロパティ-->ユーザの環境変数-->pathを設定





これで、実行時に dllを使うのですが、どこからmain.rsをコンパイルしたmain.exeを読んでも logsplrs_x64.dllが見つからないと言われなくなります。

注意としては、main.rsをコンパイルするときに .libファイルつまり -> logsplrs_x64.libが必要になります。
rustc -L C:\PrgFiles\rust\unsafelib の -L ライブラリリンクパスの指定を忘れないようにしてください。↑はあなたがインストールしたunsafelib フォルダのパスになります

makefileを書き換えます
前の定義の先頭に # を打ち、コメント化し、新たにライブラリ・リンクパスを記述してください

あとは、使われる方が工夫してご利用なさってください。
あなたとrustの素晴らしい時間が持てる事をお祈りします!

11.6. makefile コマンドメニュー

nmake clean	ごみそうじ
nmake copy	libファイルを /src または /src/quizにコピー dllファイルをカレントサンプルフォルダにコピー
nmake allclean	※ライブラリファイルも含めてごみそうじ
nmake [Enter]	サンプルmain.exeをビルドする
nmake help	コマンド・メニューを表示する

※は隠しコマンドでhelpで表示されません