

Syachi PokemonGDS Network Library

Install Document

更新履歴

更新日	更新内容
2010/02/15	新規作成
2010/03/05	<ul style="list-style-type: none">• 全体の DS という記述を Nitro/TWL に変更• ディレクトリ構成に Nitro/TWL 用サンプル関連のファイルとディレクトリを追加• ディレクトリ構成に Windows 用で使用する OpenSSL ライブラリのディレクトリを追加• 「通信ライブラリについて」の項目を追加。「旧ポケモン GDS 通信ライブラリとの違いについて」をこの項目内に移動• 「旧ポケモン GDS 通信ライブラリとの違いについて」の文言を一部加筆修正• 「DS でのサンプルとライブラリのビルド方法」のプラットフォームごとに poke_net.h のマクロ定義を変更する記述を削除• Nitro/TWL 用サンプルのビルド説明を追記• Nitro/TWL 用サンプルの説明項目を追加
2010/03/17	<ul style="list-style-type: none">• ライブラリバージョン履歴を更新• Win 用サンプルのディレクトリ構成を修正
2010/03/25	<ul style="list-style-type: none">• ライブラリバージョン履歴を更新し、別ページ(次ページ)へ移動

ライブラリバージョン履歴

この情報は man/syachi_poke_net.chm のトップページにも表示されます。

バージョン	更新日	更新内容
0.10	2010/02/15	・ 初回リリース
0.20	2010/03/05	・ SSL 通信を実装 ・ SSL 通信の実装に伴い、前バージョンまで通信時にしていた暗号化／復号化を廃止 ・ POKE_NET_GDS_Initialize 関数の引数を変更
0.21	2010/03/17	・ GDS 構造体の一部変更への対応
0.22	2010/03/25	・ GDS 構造体の一部変更への対応 ・ OS_Alloc, OS_Free を使用しないように修正

□ ディレクトリ構成と各ファイルの説明

poke_gds_syachi	
└ Makefile	メイクファイル (dbs, poke_net, gds_sample を一括でビルドします)
└ depend.mk	上記メイクファイルで使用するファイル
└ man	各種マニュアルディレクトリ
└ syachi_poke_net_inst.pdf	このファイル。GDS 通信ライブラリのインストールマニュアル
└ syachi_poke_net.chm	GDS 通信ライブラリのヘルプ
└ poke_net	ポケモン GDS 通信ライブラリディレクトリ
└ Makefile	メイクファイル
└ poke_net.c	Nitro/TWL/WIN 通信部共通ソース
└ poke_net.h	Nitro/TWL/WIN 通信部共通ヘッダ
└ poke_net_common.h	Nitro/TWL/WIN/LINUX 共通リクエストヘッダ
└ poke_net_ds.c	Nitro/TWL 通信部ソース
└ poke_net_ds_ssl.c	Nitro/TWL での SSL 関連ソース
└ poke_net_ds_ssl_cert.c	Nitro/TWL 用のルート CA データ定義ソース
└ poke_net_ds_ssl.h	Nitro/TWL での SSL 関連ヘッダ
└ poke_net_dscomp.h	Nitro/TWL/WIN/LINUX 変数定義緩衝ヘッダ
└ poke_net_gds.c	Nitro/TWL/WIN GDS 関係ソース
└ poke_net_gds_debug.c	Nitro/TWL/WIN GDS 関係デバッグ用ソース
└ poke_net_gds.h	Nitro/TWL/WIN GDS 関係ヘッダ
└ poke_net_opt.c	Nitro/TWL/WIN/LINUX 共通暗号化ソース
└ poke_net_opt.h	Nitro/TWL/WIN/LINUX 共通暗号化ヘッダ
└ poke_net_win.cpp	WIN 通信部ソース
└ gds_header	GDS 関係ヘッダディレクトリ(任天堂様からのご提供)

(次ページに続く)

(前ページより)

└ dbs	TwIDWC に含まれているサンプルデモ共通で使用されているライブラリ
└ Makefile	メイクファイル
└ include	ヘッダファイルディレクトリ
└ src	ソースファイルディレクトリ
└ gds_sample	Nitro/TWL 用のサンプルディレクトリ
└ Makefile	メイクファイル
└ include	ヘッダファイルディレクトリ
└ src	ソースファイルディレクトリ
└ TestPokeNet	Windows 用サンプルコンソールディレクトリ
└ TestPokeNet.vcproj	プロジェクトファイル
└ main.cpp	サンプルソース
└ main.h	サンプルソース
└ Nitro	CRC 計算ルーチンディレクトリ(中のソースは NitroSDK のもの)
└ Release	
└ TestPokeNet.exe	実行プログラムファイル
└ openssl	Windows 用で使用する OpenSSL ライブラリディレクトリ
└ openssl	ヘッダファイルディレクトリ
└ lib	VisualStudio 用スタティックリンクライブラリディレクトリ

□ 通信ライブラリについて

○通信プロトコル

オリジナルデータフォーマットによるリクエストレスポンス型です。

トランスポートプロトコルとして TCP を使用します。

SSL によるサーバ認証・暗号化通信を行います。

(ライブラリに隠蔽されていますので、クライアント側で特に気にすることなくご利用いただけます。)

○通信モード

SSL 通信を行いますので、Nitro/TWL での通信は全て同期モードで行われます。ご注意ください。

○旧ポケモンGDS通信ライブラリとの違い

- ・インストール方法に違いはありません。
- ・初期化方法に違いはありません。旧ライブラリと同様の手順で初期化を行い、各リクエストメソッドをコールしてください。
- ・提供される機能(インタフェース)に大幅な変更があります。
詳しくは man/syachi_poke_net.chm の各リクエストメソッドの説明を参照してください。
また、サンプルとして gds_sample (Nitro/TWL) や TestPokeNet (Win) 内のソースもご活用ください。
- ・ビルドの際に poke_net.h をプラットフォームごとに修正する必要がなくなりました。

□ Nitro/TWL でのサンプルとライブラリのビルド方法

○サンプル及び通信ライブラリのビルドに必要なライブラリと確認済みバージョン

ライブラリ	バージョン
TWLSDK	5.3 patch1
TWLDWC	5.3 patch1
TWLSYSTEM	2.1.1
TWLWiFi	3.3

○通信ライブラリのビルド

DWC のサンプル等がビルド出来る状態(環境変数設定等)で、poke_net ディレクトリ内の Makefile を使って Make します。

以下はハイブリッドでビルドした場合の、通信ライブラリの作成されるパスです。

デバッグビルド・・・poke_net¥lib¥ARM9-TS.HYB¥Debug¥libgds.TWL.HYB.a

リリースビルド・・・poke_net¥lib¥ARM9-TS.HYB¥Release¥libgds.TWL.HYB.a

○サンプルのビルド

上記ライブラリのビルドを行い、次に dbs/Makefile を使用して dbs をビルドします。

(dbs・・・TwIDWC に含まれているサンプルデモ共通で使用されているライブラリ)

以下はハイブリッドでビルドした場合の、dbs ライブラリの作成されるパスです。

・デバッグビルド・・・dbs¥lib¥ARM9-TS.HYB¥Debug¥libdwcdbs.TWL.HYB.a

・リリースビルド・・・dbs¥lib¥ARM9-TS.HYB¥Release¥libdwcdbs.TWL.HYB.a

その後に、gds_sample/Makefile を使用してサンプルをビルドしてください。

以下はハイブリッドでビルドした場合の、サンプル実行ファイルの作成されるパスです。

・デバッグビルド・・・gds_sample¥bin¥ARM9-TS.HYB¥Debug¥gds_sample.srl

・リリースビルド・・・gds_sample¥bin¥ARM9-TS.HYB¥Release¥gds_sample.srl

○全体をまとめてビルド

poke_gds_syachi 直下の Makefile を使用してメイクしてください。

pbs, poke_net, gds_sample が一括でビルドされます。

また、以下のオプションを付けることでデバッグビルドを選択出来ます。(これは全ての Makefile で有効です)

例 1. デバッグビルド

```
$ make TWL_DEBUG=True
```

例 2. リリースビルド

```
$ make
```

○GDS 通信を行うために必要なファイル

Nitro/TWL での開発で最終的に必要なものは上記ビルド方法で生成されたライブラリ

(ハイブリッドでビルドした場合は libgds.TWL.HYB.a)と、poke_net ディレクトリ内の各種ヘッダファイルとなります。

実際にインクルードするファイルは poke_net/poke_net_gds.h です。

□ Nitro/TWL 用サンプルについて

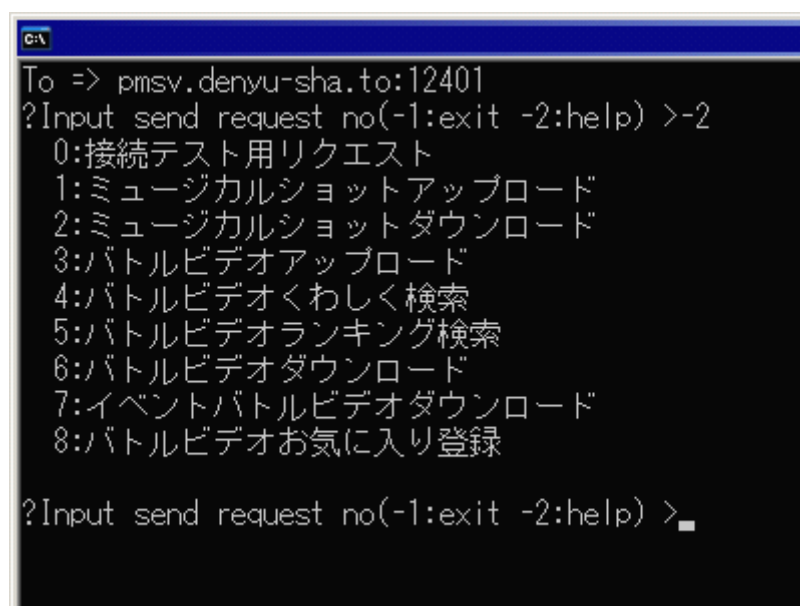
1. 起動します。
2. 「SELECT AP」画面が表示されます。
本体に WiFi 接続設定が保存されている番号を選択します。DWC 認証が開始されます。
3. 認証が完了すると、「SELECT USE PID」画面が表示されます。
使用する PID(プロファイル ID)を選択してください。
 - ・ USE DEFAULT PID・・・2345678 が使用されます。
 - ・ INPUT PID ・・・自由に値を設定出来ます。1～9999999 の間で設定してください。
 - ・ USE DEBUG PID ・・・99999999 が使用されます。この値はサーバ側での一切のチェックをパスします。
(不正データが登録出来ます)
4. PID を設定すると、「MENU」画面が表示されます。各項目を選択し、サーバとの通信を開始してください。
各項目と Windows 用でのメニューとの対応表は以下の通りです。
また、各機能は Windows 用と同様ですので、機能の詳細については次項「Windows 用サンプルコンソールについて」の機能一覧をご覧ください。

・「test request」	・・・ 接続テスト用リクエスト
・「musical update」	・・・ ミュージカルショットアップロード
・「musical download」	・・・ ミュージカルショットダウンロード
・「battle upload」	・・・ バトルビデオアップロード
・「battle search」	・・・ バトルビデオくわしく検索
・「battle rank search」	・・・ バトルビデオランキング検索
・「battle download」	・・・ バトルビデオダウンロード
・「event battle download」	・・・ イベントバトルビデオダウンロード
・「battle bookmark」	・・・ バトルビデオお気に入り登録

□ Windows 用サンプルコンソールについて

- TestPokeNet/Release/TestPokeNet.exe は接続テスト用としてご活用ください。
- ソースコードは poke_net を使用する際のサンプルとしてご参照いただければと思います。
また、poke_net/poke_net_gds_debug.c 内で、各レスポンス構造体をパースし出力していますので、
こちらも合わせてご参照いただければと思います。

【 TestPokeNet.exe の使用方法 】



```

C:\
To => pmsv.denyu-sha.to:12401
?Input send request no(-1:exit -2:help) >-2
 0:接続テスト用リクエスト
 1:ミュージカルショットアップロード
 2:ミュージカルショットダウンロード
 3:バトルビデオアップロード
 4:バトルビデオくわしく検索
 5:バトルビデオランキング検索
 6:バトルビデオダウンロード
 7:イベントバトルビデオダウンロード
 8:バトルビデオお気に入り登録

?Input send request no(-1:exit -2:help) >_

```

図 1. Win 用サンプルコンソールの画面

起動するとコンソール画面が表示されます。

この状態で - 2 を入力すると、図 1 のようにリクエスト番号の一覧が表示されます。(ここでのリクエスト番号と実際の通信で使用されるリクエスト番号は違います。)

一覧の各リクエストの左側に表示されている番号を入力すると、通信が開始されます。(追加パラメータの入力を求められる項目もあります。)

終了する場合は、ウィンドウを閉じるか - 1 を入力してください。

(次ページに機能一覧)

Windows 用サンプルコンソールの機能一覧は以下の通り

リクエスト番号	機能名称	機能
0	接続テスト用リクエスト	入力したメッセージを送信します。サーバから送信メッセージがそのまま返ってきます。
1	ミュージカルショットアップロード	ミュージカルショットをアップロードします。
2	ミュージカルショットダウンロード	ミュージカルショットをポケモン番号で検索し、そのリストを取得します。
3	バトルビデオアップロード	バトルビデオデータをアップロードします。
4	バトルビデオくわしく検索	このサンプルでは便宜的にポケモン番号とランキング種別で検索し、そのリストを取得します。
5	バトルビデオランキング検索	最新 30 件／バトルサブウェイランキング／通信対戦ランキングで検索し、そのリストを取得します。
6	バトルビデオダウンロード	番号指定でバトルビデオの本体データを取得します。番号は暗号化されていない値を指定してください。
7	イベントバトルビデオダウンロード	番号指定でイベントバトルビデオの本体データを取得します。番号は暗号化されていない値を指定してください。
8	バトルビデオお気に入り登録	番号指定でバトルビデオをお気に入りに登録します。番号は暗号化されていない値を指定してください。

謝辞

この製品には、OpenSSL Toolkit で使用するために OpenSSL Project によって開発されたソフトウェアが組み込まれています。(http://www.openssl.org/)

この製品には、Eric Young 氏 (eay@cryptsoft.com) によって作成された暗号ソフトウェアが含まれています。

この製品には、Tim Hudson 氏 (tjh@cryptsoft.com) によって作成されたソフトウェアが含まれています。

※ 上記の記載は Windows 版通信ライブラリでの OpenSSL の利用に伴う記載です。

Nitro/TWL 版通信ライブラリ、及びそのサンプルプログラムには一切関係はありません。

□ その他情報

○テストサーバのアドレスと使用ポート

- ・ アドレス・・・pkgdtest.nintendo.co.jp
- ・ ポート ……12401

ライブラリの初期化時に上記アドレスとポートを引数として指定してください。

※ポートの開放等を忘れないようにしてください。

○Nitro/TWL のスレッド優先

Nitro/TWL 用の通信ライブラリでは内部でスレッドを使用しています。

スレッドの優先レベルは初期化の後に POKE_NET_GDS_SetThreadLevel()を用いて変更出来ます。

スレッドはリクエストを発行した時に作られ、レスポンスを受け取ったりエラー終了すると、終了します。

□ まず初めに試していただきたいこと

テストサーバへの接続が正常に行われるかどうかを試験するために、試していただきたいことがございます。

まず初めに、

前述の Windows 用サンプルコンソールのリクエスト番号 0 番（接続テスト用リクエスト）を試行してください。

送信したメッセージがレスポンスデータとして返ってくれば、接続は正常に行われています。

□ GDS 通信ライブラリでのレスポンス処理について

TestPokeNet/main.cpp 内でも行われている GDS 通信ライブラリにおけるレスポンス処理についての補足説明です。

各リクエストを発行後、POKE_NET_GDS_GetStatus()の値が POKE_NET_GDS_STATUS_FINISHED となり通信が正常終了した後に、レスポンス処理を行います。この状態では既に通信スレッドは終了していますので、レスポンスデータを読むことが可能となっています。逆にリクエスト後～通信が完了するまではスレッド内から、リクエスト時に_response で指定したメモリ領域へアクセスを行う可能性がありますので注意してください。

- POKE_NET_GDS_GetResponse()
- POKE_NET_GDS_GetResponseSize()

上記 2 つの関数を用いてレスポンス格納先のアドレスとレスポンス容量を取得出来ます。

POKE_NET_GDS_GetResponse()は POKE_NET_RESPONSE 構造体のアドレスが返ってきます。

POKE_NET_GDS_GetResponseSize()は POKE_NET_RESPONSE 構造体も含めたレスポンス容量が返ってきます。

次に取得した POKE_NET_RESPONSE 構造体の Result メンバを見て処理が成功したかどうかを確認します。ReqCode メンバにはどのリクエストに対するレスポンスかを表すリクエストコードが返ってきます。

例えばミュージカルショットの登録処理が成功していた場合、Result メンバには

POKE_NET_GDS_RESPONSE_RESULT_MUSICALSHOT_REGIST_SUCCESS が格納されています。(各リクエストメソッドと発行されるリクエストの関係については syachi_poke_net.chm の各リクエストメソッドの説明を参照してください。)

Result メンバの値が成功を指している場合、Param メンバにはそれぞれのリクエストに対するレスポンス構造体が返ってきています。

上のミュージカルショットの登録の例であれば、Param メンバの値は

POKE_NET_GDS_RESPONSE_MUSICALSHOT_REGIST 構造体の先頭アドレスとなります。

Result メンバの値が成功以外を指している場合、Param メンバにデータは返ってきませんので注意してください。(また、成功しても Param メンバにデータを返さないリクエストがあります。その場合の返答は POKE_NET_RESPONSE 構造体のみとなります。)

また、poke_net/poke_net_gds_debug.c ファイル内では、各レスポンス構造体をパースして出力する処理が記述されていますので、参考にしてください。

※通信の異常によりレスポンス容量がおかしくなっていた場合等はライブラリ側でチェックが行われます。

※POKE_NET_GDS_GetResponse()で返ってくるアドレスはリクエスト発行時に指定したレスポンス保存先のアドレスとは異なるアドレス(レスポンスヘッダ分シークしたアドレス)が返ってきますので注意してください。