

# Nintendo Wi-Fi Connection

## TWL-DWC プログラミングマニュアル

Ver 2.0.10

任天堂株式会社発行

このドキュメントの内容は、機密情報であるため、**厳重な取り扱い、管理を行ってください。**

# 目次

1	はじめに .....	7
2	TWL-DWCにおけるユーザー管理 .....	8
2.1	Wi-Fiユーザー情報の管理 .....	8
2.1.1	ユーザーIDとプレイヤーID .....	8
2.1.2	ユーザーIDとプレイヤーIDの違い .....	9
2.1.3	ゲーム毎のプレイヤー情報: ログインID .....	9
2.1.4	ゲームで保存するWi-Fi認証用の情報 .....	10
2.2	友達管理概要 .....	11
2.2.1	友達関係の構築 .....	11
2.2.2	DSワイヤレス通信での友達関係の構築 .....	11
2.2.3	友達コードでの友達関係の構築 .....	12
2.2.4	ゲームで保存する友達の情報 .....	12
2.3	例外処理 .....	13
2.3.1	DS本体とDSカードの関連付けの消去 .....	13
3	TWL-DWCの初期化 .....	14
4	ユーザーデータの作成 .....	16
5	接続処理 .....	18
5.1	インターネットに接続する .....	18
5.2	インターネットから切断する .....	19
5.3	Wi-Fiコネクションサーバへ接続する .....	19
6	友達リストの作成／友達情報の作成 .....	22
6.1	DSワイヤレス通信で友達情報を交換する .....	22
6.2	友達コードを交換する .....	23
6.3	友達リストの同期処理 .....	24
6.4	友達情報タイプ .....	26
6.5	友達の状態を取得する .....	27
7	マッチメイク .....	29
7.1	友達無指定ピアマッチメイク .....	29
7.2	友達指定ピアマッチメイク .....	30
7.3	マッチメイク候補プレイヤーを評価する .....	32
7.4	サーバクライアント型マッチメイク .....	33
7.5	グループIDを用いた再接続 .....	35
7.6	締め切り .....	37
7.7	ConnectAttemptCallback .....	38
7.7.1	DWCEvalPlayerCallbackとDWCCConnectAttemptCallbackの違い .....	39
7.8	サーバ交代 .....	40

7.9	マッチメイクの高速化.....	41
7.10	マッチメイク指標キーに使用できないキー名 .....	42
8	データ送受信.....	43
8.1	ピア・ツー・ピアのデータ送受信.....	43
8.2	接続形態 .....	45
8.3	コネクションを切断する .....	46
8.4	DWC_InitFriendsMatch関数で指定するバッファサイズの見安 .....	47
8.5	遅延とパケットロスのエミュレーション .....	48
8.6	データ送受信量の目安.....	48
9	通信エラー .....	50
9.1	エラー処理 .....	50
9.2	エラーコード一覧.....	51
10	ネットワークストレージ対応.....	53
11	NITRO-DWC 3.X以前のバージョンとの相違点 .....	56
11.1	マッチメイク関連.....	56

## 表

表 7-1	マッチメイク指標キーに使用できないキー名一覧.....	42
表 8-1	通信内容とバッファサイズの見安 .....	47
表 8-2	通信データ内訳 .....	48

## 図

図 2-1	DS本体とDSカードのユーザーIDの保存状態 .....	8
図 2-2	複数のDS本体と複数のDSカードで使ったイメージ .....	8
図 2-3	インターネット上のデータの持ち方の関係 .....	9
図 2-4	ログインIDの構成 .....	9
図 2-5	Wi-Fi認証用の用語の全体図 .....	10
図 2-6	DSワイヤレス通信での友達関係作成のイメージ.....	11
図 2-7	友達コードでの友達関係作成のイメージ.....	12

---

コード

コード 3-1	DWC初期化.....	14
コード 4-1	ユーザーデータの作成 .....	16
コード 4-2	ユーザーデータのセーブ.....	17
コード 5-1	インターネットへの接続.....	18
コード 5-2	インターネットからの切断 .....	19
コード 5-3	Wi-Fiコネクションサーバへの接続 .....	20
コード 6-1	DSワイヤレス通信による友達情報の交換 .....	23
コード 6-2	友達コードの交換 .....	24
コード 6-3	友達リストの同期処理 .....	26
コード 6-4	友達情報タイプの取得 .....	27
コード 6-5	友達の状態取得.....	28
コード 7-1	友達無指定ピアマッチメイク .....	30
コード 7-2	友達指定ピアマッチメイク .....	32
コード 7-3	マッチメイク候補プレイヤーの評価.....	33
コード 7-4	サーバクライアント型マッチメイク .....	35
コード 7-5	グループIDの取得.....	35
コード 7-6	グループIDからの再接続.....	36
コード 7-7	締め切り処理 .....	37
コード 7-8	ConnectAttemptCallbackの例 .....	39
コード 8-1	データ送受信のための準備 .....	44
コード 8-2	データ送信 .....	45
コード 8-3	遅延とパケットロスのエミュレーション.....	48
コード 9-1	エラー処理 .....	51
コード 10-1	ストレージサーバへのアクセス .....	55

## 改訂履歴

版	改訂日	改 訂 内 容	承認者	担当者
2.0.10	2009-12-21	「7.7 ConnectAttemptCallback」、「7.8 サーバ交代」においてサーバクライアントマッチメイクではサーバ交代が発生しないと記述されていましたが、発生すると修正しました。		
2.0.9	2009-11-25	「11 NITRO-DWC 3.X以前のバージョンとの相違点」において、一定時間通信の無い相手を自動的に切断する仕様について追記しました。		
2.0.8	2009-09-01	「3 TWL-DWCの初期化」、「4 ユーザーデータの作成」、「5.3 Wi-Fiコネクションサーバへ接続する」において、DWC_Init関数が開発用と製品用の2関数に分かれたこと、またそれに伴いDWC_CreateUserData関数及びDWC_InitFriendsMatch関数の引数が変わった事に対応しました。 「コード 6-1 DS ワイヤレス通信による友達情報の交換」及び「コード 6-2 友達コードの交換」で s_userdata 変数の扱いに誤りがありましたので修正しました。		
2.0.7	2009-06-30	「8.4 DWC_InitFriendsMatch 関数で指定するバッファサイズの目安」で Reliable 通信におけるバッファサイズの目安の計算式が誤っていたので修正しました。		
2.0.6	2009-05-25	「8.1 ピア・ツー・ピアのデータ送受信」に、デバッグ出力によりパケットが遅延して見えることがある旨を追記しました。		
2.0.5	2009-04-08	「5.1 インターネットに接続する」で、文末が、「(コード 5-1)」であるべきところ、「(コード 5-)」となっていたので修正しました。		
2.0.4	2009-03-23	用語統一のため、「プロフィール ID」と書かれていた部分を「GS プロファイル ID」に修正。		
2.0.3	2009-02-02	「7.7 ConnectAttemptCallback」と「7.8 サーバ交代」にサーバクライアントマッチメイクではサーバ交代が発生しない旨を追記。 「コード 7-8 ConnectAttemptCallback の例」を DWC_GetAIDList 関数の仕様変更に合わせて修正。		
2.0.2	2008-12-18	「7.8 サーバ交代」誤植を修正（接続形態（詳しくは「」を参照して下さい）→ 接続形態（詳しくは「8.2 接続形態」を参照して下さい））		
2.0.1	2008-11-28	GHTTP ライブラリの非公開化に伴い、「9 HTTP 通信機能」を削除。		
2.0.0	2008-10-10	NITRO-DWC から TWL-DWC への移行に対応。 TWL-DWC 5.0 でマッチメイクに関して行われた仕様変更に対応。		
1.4.3	2007-07-21	「コード 7-3 マッチメイク候補プレイヤーの評価」誤記を修正（s_int_key → &s_int_key）		
1.4.2	2007-02-15	「コード 6-1 DS ワイヤレス通信による友達情報の交換」誤記を修正（s_friendData → ownFriendData）		
1.4.1	2006-08-09	「8.4 DWC_InitFriendsMatch関数で指定するバッファサイズの目安」と「表 8-2 通信データ内訳」の誤記を修正		
1.4.0	2006-06-19	「9.1 エラー処理」エラーコードの表示条件を変更		
1.3.0	2006-06-06	「2.1.3 ゲーム毎のプレイヤー情報：ログインID」【仮ログインIDの重複例】を修正 「3 TWL-DWCの初期化」使用メモリサイズを 200kbyte → 230kbyte に変更 「7.10 マッチメイク指標キーに使用できないキー名」を追加 その他の変更（語句の統一、修正等）		
1.2.0	2006-03-10	「2 TWL-DWCにおけるユーザー管理」を追加		

		「7.9 マッチメイクの高速化」を追加 「8.6 データ送受信量の目安」を追加 その他の変更（段落の見直し、語句の変更等）		
1.1.0	2006-01-30	「コード 6-3 友達リストの同期処理」更新 「コード 6-4 友達情報タイプの取得」誤植を修正（stablished→established） 「コード 7-3 マッチメイク候補プレイヤーの評価」誤植を修正（"anymatch"→ "anymatch_test"） 「10ネットワークストレージ対応」データロード関数を、新規追加関数に変更		
1.0.0	2005-12-27	初版作成。		

# 1 はじめに

TWL-DWC(以下 DWC)は、Nintendo Wi-Fi Connection の「カンタン・あんしん・無料」の理念を実現するため、下記の内容を目標に構築されています。

- インターネットに接続するための煩雑な情報をユーザーから遮蔽し、ユーザーが簡単に使用できる
- インターネットに接続していない状態で、あらかじめ「ワイヤレス通信」または「友達コード交換」等で友達関係を構築しておくことでインターネットに接続したときに友達と簡単に通信できる
- ユーザーが DS カードを手放し、他のユーザーの手に渡った場合、インターネットに関する情報を他のユーザーが簡単に参照できないようにセキュリティを保つ

## 2 TWL-DWC におけるユーザー管理

### 2.1 Wi-Fi ユーザー情報の管理

Nintendo Wi-Fi Connectionの認証（以下、Wi-Fi認証）に必要な情報には、ユーザーID、プレイヤーID、パスワードがあります。これらの情報はDS本体とDSカードを対にして管理します（図 2-1）。

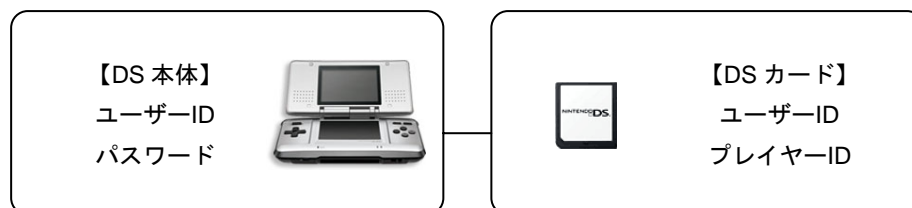


図 2-1 DS 本体と DS カードのユーザーID の保存状態

- DS 本体には、Wi-Fi 認証に使用するユーザーID とパスワードを保存します。
- DS カードには、Wi-Fi 認証に使用するユーザーID とプレイヤーID を保存します。

Nintendo Wi-Fi Connectionでは、これらの情報を使って、インターネット上のサーバに接続します。DSカードに保存されたユーザーIDがDS本体に保存されているものと違う場合、インターネット上に保存されたデータにはアクセス出来ません。こうすることで、不正なデータアクセスを防ぎます（図 2-2）。

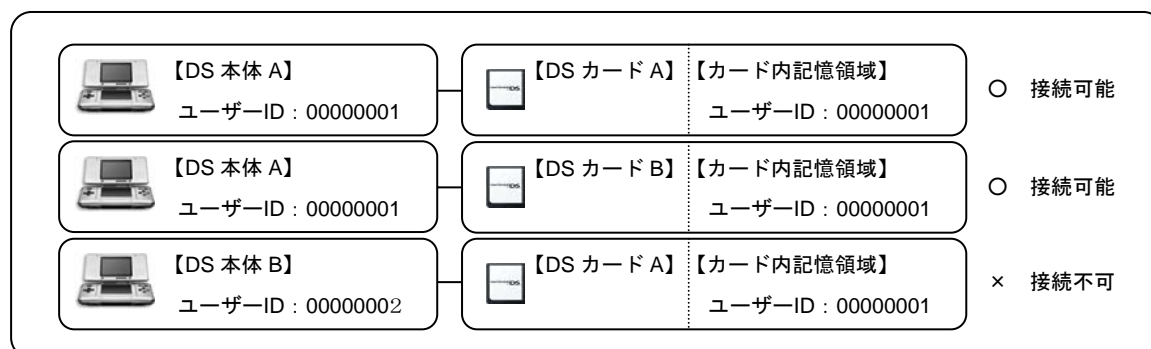


図 2-2 複数の DS 本体と複数の DS カードで使ったイメージ

#### 2.1.1 ユーザーID とプレイヤーID

ユーザーID は、出来る限り固有になるようにインターネットに接続されていないときに生成されます。その後、インターネットに接続し、認証サーバに確認して登録されると正式なユーザーID となります。もし認証サーバに確認した結果、ユーザーID が重複していて使用できない場合は、重複していないユーザーID が割り当てられます。

※ユーザーID をできるだけ固有にするために、本体の MAC アドレスの一部を利用します。そのため、違う本体間で同じユーザーID が生成されることはありませんが、ユーザーID を移動させた場合・ユーザーID を再生成した場合などに重複の可能性があります。



プレイヤーID は、32bits の乱数です。インターネットのサーバ上のデータは「ユーザーID+プレイヤーID+イニシャルコード」単位で管理されますので、プレイヤーID は、同一ユーザーID 及び同一イニシャルコードに対して固有であれば問題ありません。これも、もし重複した場合は、認証時に重複していないプレイヤーID が割り当てられます。

### 2.1.2 ユーザーID とプレイヤーID の違い

ユーザーID は、DS 本体に対して発行するので、同じ DS 本体を使用するユーザーは、いくつものゲームで一つのユーザーID を使用することになります。

プレイヤーIDは、DSカードに対して発行するので、同じDS本体（ユーザーID）かつ同じイニシャルコードのゲームに対して、違うプレイヤーIDを使用することになります。（図 2-3）

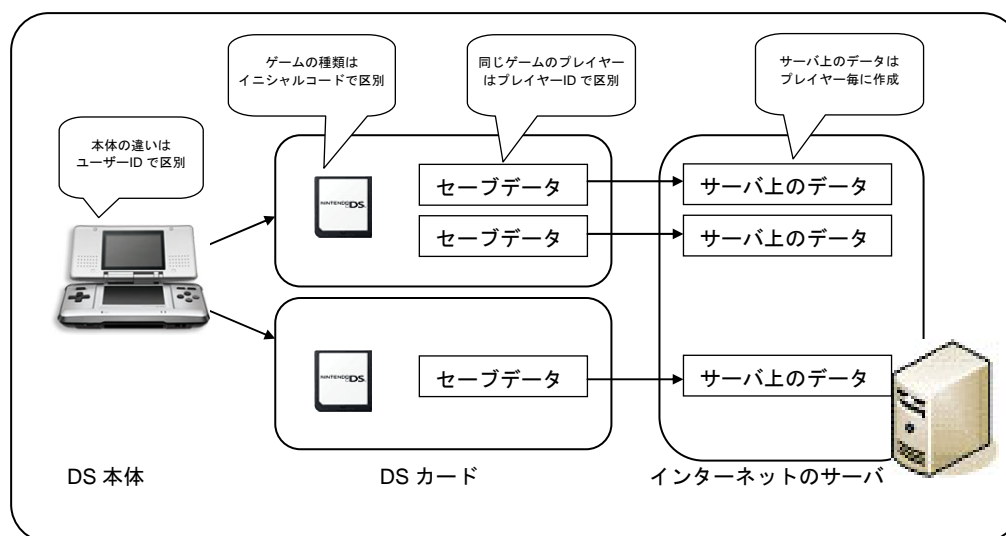


図 2-3 インターネット上のデータの持ち方の関係

### 2.1.3 ゲーム毎のプレイヤー情報：ログイン ID

「ユーザーID+プレイヤーID+イニシャルコード」をまとめたものを「ログインID」と呼びます（図 2-4）。また、インターネットのサーバ上に保存するユーザー情報の単位を「プロフィール」、プロフィールをサーバ上で管理するためのIDを「GSプロフィールID」と呼びます。

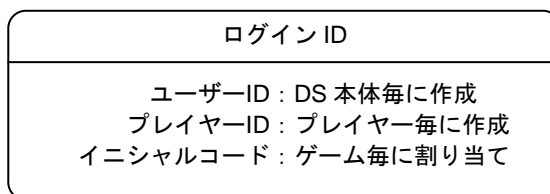


図 2-4 ログイン ID の構成

DWC 内部では、インターネットのサーバ上で他のユーザーのプロファイルを検索するために、ログイン ID または GS プロファイル ID を使用します。

前述のとおり、ログイン ID はインターネットに接続していない状態で生成されます（仮ログイン ID）。高い確率でそのまま使用可能ですが、仮ログイン ID がそのまま使えない場合もあります。その場合は、重複していない承認されたログイン ID（認証済ログイン ID）が生成されます。認証済ログイン ID に対して GS プロ

ファイル ID が 1 対 1 で割り当てられます。

#### 【仮ログイン ID の重複例】

- 認証されていないユーザーID でログイン ID を作成した場合、既に他者がそのユーザーID を認証サーバに登録していて、かつ同じゲーム、同じプレイヤーID でログイン ID を作成していた場合。
- 複数の DS 本体で、同じ未認証ユーザーID を使って、同じゲーム、同じプレイヤーID でログイン ID を作成した場合。

### 2.1.4 ゲームで保存する Wi-Fi 認証用の情報

ゲームでは、これらの Wi-Fi 認証用の情報を DS カードのバックアップに保存しなければなりません。

**この認証用の情報のサイズは、64 バイトです。**

Wi-Fi 認証用の情報には、仮のログイン ID、認証済ログイン ID、GS プロファイル ID などが含まれています。この情報は、DWC で作成・更新されますので、開発者が内容の詳細を把握する必要はありません。

また、一つの DS カードで複数のプレイヤーがプレイヤーデータを保存できる場合、プレイヤーごとに Wi-Fi 認証用の情報を保存しなければなりません。

これまでの、Wi-Fi 認証に関する用語は、図 2-5 のようになります。

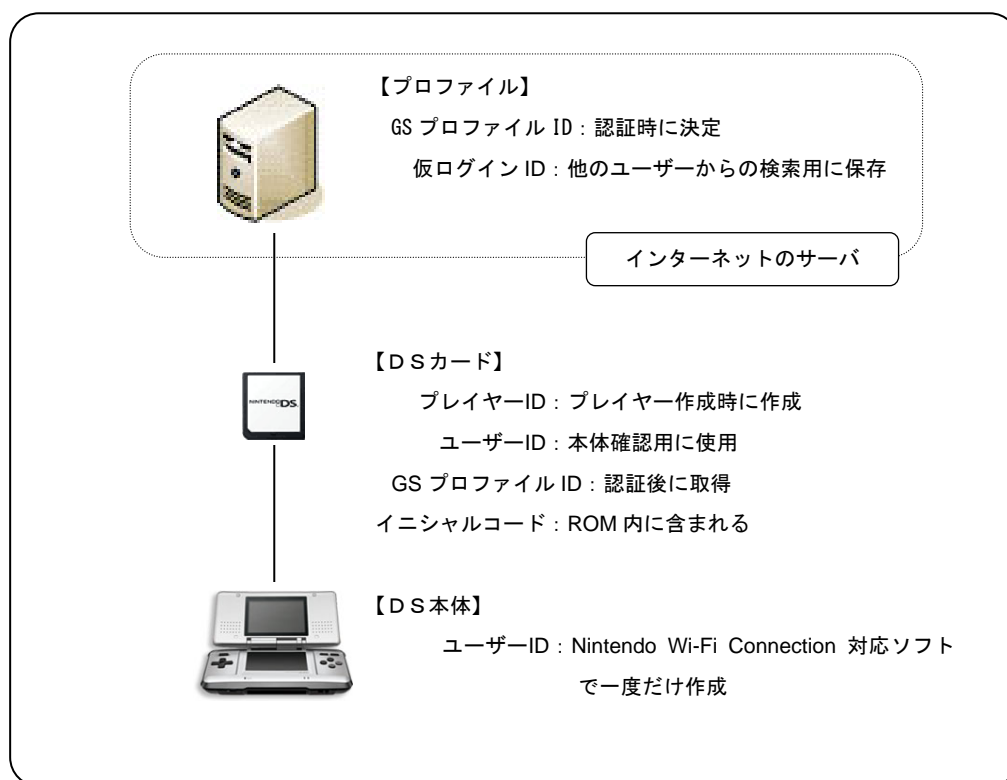


図 2-5 Wi-Fi 認証用の用語の全体図

## 2.2 友達管理概要

### 2.2.1 友達関係の構築

DWC では、友達と簡単に通信を開始できるようにするために、友達関係の構築をインターネット上のサーバで行います。友達関係の構築は、ユーザー情報を交換することで実現します。構築された友達関係は、各ユーザーのプロファイルに保存されます。

友達関係を作成するために交換するユーザー情報として、以下の方法があります。

- DS ワイヤレス通信を用いる方法

ログイン ID または GS プロファイル ID を交換します。

一度もログインしていない場合は、ログインIDを使用します。それぞれがローカルで作ったままの状態でも、非常に一意性が高いため、そのまま使える確率が高いのですが、確実ではありません。しかし、現時点では、 $1/2^{75}$ 以下の確率であり、特別な対策は必要ありません。

既に一度でもログインしている場合は、GS プロファイル ID を用います。必ず相手を特定できるため、確実に友達関係を作成することが出来ます。

- 友達コードを用いる方法

GS プロファイル ID にエラーチェック用の情報を入れた「友達コード」を交換します。

GS プロファイル ID を用いるためには、必ず一度はインターネットに接続する必要があります。また、入力を間違える可能性があるため、入力の確認ややり直しができるインタフェースを作成する必要があります。

これらの交換用の情報は、DWC で作成することができます。DWC には、DS カードに保存された Wi-Fi 認証用の情報から、もっとも適切と思われる情報を自動的に作成する機能があります。

### 2.2.2 DS ワイヤレス通信での友達関係の構築

DSワイヤレス通信を行った際に相手と情報を交換しておくことで、自動的に友達関係が結べる仕組みを提供します。交換する情報は、ユーザーデータに格納されているログインIDまたはGSプロフィールIDから作成します。（図 2-6）**ただし、DWCではこの情報をDSワイヤレス通信で交換することはサポートしていません。作成した情報の交換は、アプリケーション側で行ってください。**

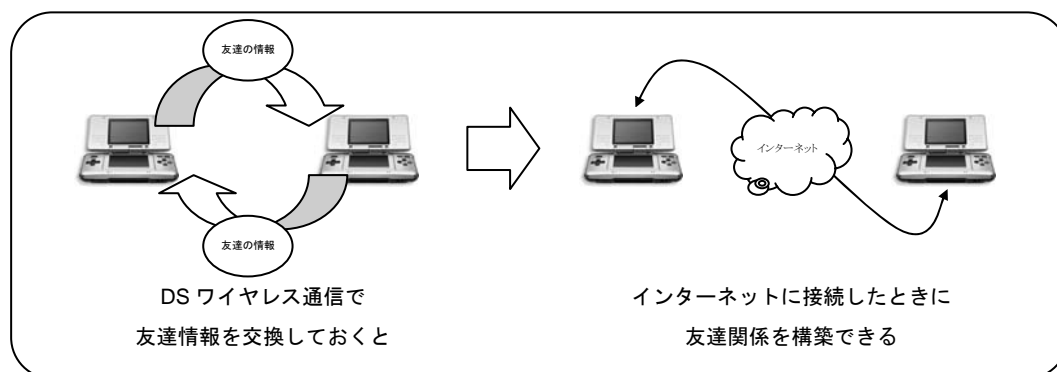


図 2-6 DS ワイヤレス通信での友達関係作成のイメージ

## 2.2.3 友達コードでの友達関係の構築

友達関係を結ぶための相手ユーザーが特定できる情報を「友達コード」と呼びます。

この友達コードを交換する事で、友達関係を作成できる仕組みを提供します。（図 2-7）

この友達コードは、人が扱うため不必要に長くは問題があります。そのため、ログイン ID ではなく、一度インターネットに接続することで得られる GS プロファイル ID を使って作成します。

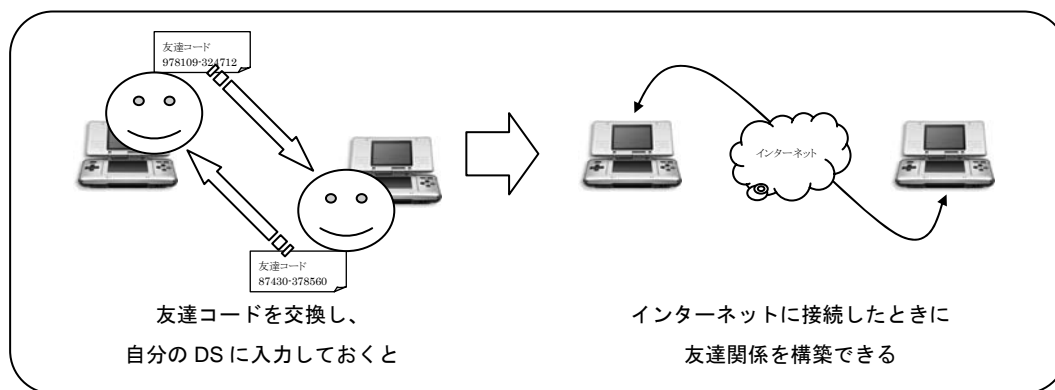


図 2-7 友達コードでの友達関係作成のイメージ

**友達コードは、12桁の数字で表現されます。**ゲームでは、以下の点に注意してください。

- 友達コード発行のためのユーザーインターフェースを作成する必要があります。インターネットに一度も接続されていない場合は発行できませんので、メッセージを表示する必要があります。
- 友達コード入力のためのユーザーインターフェースを作成する必要があります。このユーザーインターフェースでは、友達コードが間違えている場合も想定して、入力されたものを保存し、何度でも修正できるようにする必要があります。

## 2.2.4 ゲームで保存する友達の情報

ゲームでは、交換した友達情報をゲームで友達関係として管理したい最大人数分、バックアップに保存する必要があります。これは、インターネットに接続していない状態で、ユーザーが友達関係を編集できるようにするためです。また、実際のゲームでは管理している友達に関連する情報（ニックネームや対戦戦績など）もあわせて保存する必要があります。

DWC では、ログイン ID、GS プロファイル ID、友達コードの種類を意識せず、友達情報として扱うようにしています。

**DWC で使用する友達情報の保存には、一人につき 12 バイト必要です。**

## 2.3 例外処理

---

### 2.3.1 DS 本体と DS カードの関連付けの消去

---

Nintendo Wi-Fi Connection では、セキュリティのために DS 本体と DS カードをセットにして使用します。そのため、DS 本体を買い換えたり、故障したときに、Nintendo Wi-Fi Connection に接続できなくなり、ユーザーの利便性が著しく落ちる場合があります。

この問題に対応するために DWC では、プロフィールに保存している情報を破棄することで、DS カードの関連付け情報を削除できる仕組みを提供しています。この場合、インターネット上の友達関係はすべて削除されますので、ユーザーに警告し、ユーザーが誤って情報を削除してしまわないようなインタフェースを作成してください。

もし、インターネット上の友達関係を削除した場合でも、削除したユーザーの DS カードには相手の友達情報が残っています。この情報を利用して、相手に新しい友達コードを伝えることで、友達関係を復帰させることもできます。この場合、削除したユーザーが友達にもう一度登録をしてもらうように促す必要がありますので、各アプリケーションで手順を具体的にユーザーに伝えるメッセージを加えてください。

具体的な処理としては、現在 DS カードに保存している関連付けを消去し、新たな関連付けを作成したい場合は、新規にユーザーデータを作成し、以前まで使用していたユーザーデータを破棄することで対応してください。また、ユーザーデータが更新されても、DS カードに保存している友達リストの友達関係は成立したままの状態になります。もし、友達リストを残す仕様にしていて、かつ友達関係が成立した状態であることをユーザーにわかるようにしている場合、友達情報の友達関係成立フラグをクリアするようにしてください。

※ フロー図を「Nintendo Wi-Fi Connection プログラミングガイドライン」に載せていますので、ご確認ください。

### 3 TWL-DWC の初期化

DWC の初期化は、すべての DWC 関数を呼び出す前に必ず行ってください。例外として、DWC\_Debug\_DWCInitError 関数、DWC\_SetReportLevel 関数初期化関数より前に呼び出すことができます。

DWC の初期化を行う DWC\_InitForDevelopment/DWC\_InitForProduction 関数では、以下の処理を行います。

- 開発用と製品用どちらの認証サーバを使用するかの設定
- DWC ライブラリが使用するメモリ確保関数の設定
- DS 本体に格納されるユーザー認証のための情報を生成
- DS 本体のバックアップに格納されている接続先情報などの正当性のチェックと修正

```
void init_dwc( void )
{
    int ret;
    ret = DWC_InitForDevelopment("dwctest", 'NTRJ', AllocFunc, FreeFunc );
    //ret = DWC_InitForProduction("dwctest", 'NTRJ', AllocFunc, FreeFunc );

    // DWC初期化
    if ( ret == DWC_INIT_RESULT_DESTROY_OTHER_SETTING )
        disp_init_warning_msg(); // 警告メッセージ表示
}

// メモリ確保関数
void* AllocFunc( DWCAIlocType name, u32 size, int align )
{
    void * ptr;
    OSIntrMode old;
    (void)name;
    (void)align;

    old = OS_DisableInterrupts();
    ptr = OS_AllocFromMain( size );
    OS_RestoreInterrupts( old );

    return ptr;
}

// メモリ解放関数
void FreeFunc( DWCAIlocType name, void* ptr, u32 size )
{
    OSIntrMode old;
    (void)name;
    (void)size;

    if ( !ptr ) return;

    old = OS_DisableInterrupts();

    OS_FreeToMain( ptr );
    OS_RestoreInterrupts( old );
}
```

コード 3-1 DWC 初期化

開発向けの認証サーバを使用する場合は、DWC\_InitForDevelopment 関数で初期化し、製品向けの認証サーバを使用する場合は、DWC\_InitForProduction で初期化します。どちらの認証サーバを使用するかによって、ダウンロードライブラリなどで使用するサーバの切り替えも行われます。詳細は DWC 初期化関数の関数リファレンスを参照下さい。

DS カードのバックアップデータを削除するシーケンスを実装する場合は、それと同じタイミングで呼び出すことを推奨します。

DWC が必要とするメモリについて参考情報として、四人でマッチメイクを行う場合で 230KByte 程度です。

マッチメイクの最大人数を一人減らすたびに必要なメモリは 20Kbyte (DWC\_InitFriendsMatch関数の引数、sendBufSize, recvBufSizeがデフォルトの 8Kbyteずつの場合) 程度減ります。

## 4 ユーザーデータの作成

DWC では、ユーザーデータを基にして下記の典型的な処理を行います。

- ユーザー認証
- 友達関係の作成

※インターネットに接続しない場合でも、DS ワイヤレス通信での友達関係を構築するためにユーザーデータは必要になります。

ユーザーデータがまだ作成されていない場合やユーザーデータが破損している場合 DWC\_CreateUserData 関数を用いてユーザーデータを作成し DS カードのバックアップに保存してください。

DWCUserData 構造体を保存するメモリはアプリケーションで確保してください。ひとつの DS カードで複数のプレイヤーをサポートする場合、人数分のユーザーデータが必要になります。

既に作成している場合は、バックアップからメモリに読み込んだ後、DWC\_CheckUserData関数を用いて、ユーザーデータの正当性を確認してください（コード 4-1）。

```
B00L create_userdata( void )
{
    // バックアップデータがあって、かつそこにユーザーデータがあれば、
    // 全てロードしてTRUEを返す
    if ( DTUDs_CheckBackup() )
    {
        (void)DTUD_LoadBackup( 0, &s_PlayerInfo, sizeof(DTUDPlayerInfo) );

        OS_TPrintf("Load From Backup¥n");

        if ( DWC_CheckUserData( &s_PlayerInfo.userData ) )
        {
            DWC_ReportUserData( &s_PlayerInfo.userData );
            return TRUE;
        }
    }

    // 有効なユーザーデータがセーブされていなかった場合
    OS_TPrintf("no Backup UserData¥n");

    // ユーザーデータを作成する
    DWC_CreateUserData( &s_PlayerInfo.userData );

    OS_TPrintf("Create UserData. ¥n");
    DWC_ReportUserData( &s_PlayerInfo.userData );

    return FALSE;
}
```

### コード 4-1 ユーザーデータの作成

DWC\_CheckDirtyFlag関数を使用すると、DSカードへユーザーデータを保存する必要があるかどうかを判定できます。ユーザーデータをバックアップへ保存する前に、必ずDWC\_ClearDirtyFlag関数でDirtyFlagをクリアするようにしてください（コード 4-2）。

```
void check_and_save_userdata( void )
{
    if ( DWC_CheckDirtyFlag( &s_PlayerInfo.userData ) )
```



```
{
    DWC_ClearDirtyFlag( &s_PlayerInfo.userData );
    DTUD_SaveBackup( 0, &s_PlayerInfo.userData, sizeof(DWCUserData) );
}
```

#### コード 4-2 ユーザーデータのセーブ

インターネットに接続する前に以下の手順でユーザーデータの確認を行ってください。

- DWC\_CheckHasProfile 関数で、ユーザーデータがインターネットに接続しプロファイルを取得しているかどうかを確認してください。取得していない場合はユーザーデータが更新され、DS 本体と DS カードがセットで扱われるようになります。
- DWC\_CheckValidConsole 関数で、DS 本体と DS カードが正しく使用されているかを確認してください。DS 本体と DS カードが正しくない場合、認証に失敗するためインターネットに接続できません。

※ フロー図を「Nintendo Wi-Fi Connection プログラミングガイドライン」に載せていますので、ご確認ください。

## 5 接続処理

DWC では、インターネットに接続する処理を下記の二つのフェーズに分けて行えるようにしています。

- インターネットに接続する（Wi-Fi 接続し IP アドレスを取得する）
- Wi-Fi コネクションサーバ（以下サーバ）へ接続する

初めてインターネットに接続する際に、任天堂の認証サーバから DS 本体に対してユーザーID を発行します。このユーザーID は、発行された DS 本体のバックアップに保存されます。

その後、サーバへ接続する際に、DWC で作成したユーザーデータにユーザーID とプレイヤーID を保存し、プロファイルを作成します。作成されたプロファイルに対応する GS プロファイル ID がユーザーデータに格納されます。

### 5.1 インターネットに接続する

初めてインターネットに接続し、IP アドレスを取得する際に、任天堂の認証サーバから DS 本体に対してユーザーID を発行します。また、インターネット接続時に接続テストサーバへ TCP での通信ができることを確認し、インターネットへの接続が正常に行われているかを確認します。

これらの処理は、DWC\_\*Inet関数で自動的に行われます（コード 5-1）。

```
static DWCInetControl s_ConnCtrl; // インターネット接続が切断されるまで保持する
BOOL connect_to_inet( void )
{
    // インターネット接続初期化処理
    DWC_InitInet( &s_ConnCtrl );

    // 接続開始
    DWC_ConnectInetAsync();

    // 接続処理
    while ( !DWC_CheckInet() )
    {
        DWC_ProcessInet();
        // Vblank待ち処理
        // 接続処理中は、メインスレッドよりも優先順位の低いスレッドに
        // 処理時間を渡す必要があるため、OS_WaitIrq関数を使用してくだ
        // さい。
        GameWaitVBlankIntr();
    }

    // 接続結果確認
    if ( DWC_GetInetStatus() != DWC_CONNECTINET_STATE_CONNECTED )
    {
        handle_error();
        return FALSE;
    }
    // 接続完了
    :
}
```

コード 5-1 インターネットへの接続

## 5.2 インターネットから切断する

DWC\_CleanupNet\*関数を呼び出すことで、インターネットから切断されます（コード 5-2）。

もし、通信エラーが発生し自動的に切断された場合でも、ライブラリ用のメモリを解放する必要があるため、必ず呼び出してください。

```
void di sconnect_func( void )
{
    while ( !DWC_CleanupNetAsync() )
    {
        GameWaitVBlankIntr();
    }
    :
}
```

コード 5-2 インターネットからの切断

## 5.3 Wi-Fi コネクションサーバへ接続する

Wi-Fiコネクションサーバ（以下サーバ）へ接続する際には、DWC\_InitFriendsMatch関数を使用して、マッチメイク／友達関連機能の初期化を行います（コード 5-3）。

この関数の引数には、ユーザーデータへのポインタ、GameSpy 社から付与されるプロダクト ID とシークレットキー、DS 同士の通信で使用される送受信バッファサイズ、友達リストへのポインタと友達リストの最大要素数を渡します。

ここで指定された制御オブジェクトは、DWC\_ShutdownFriendsMatch 関数が呼び出されるまで、DWC 内で使用されます。

送受信バッファサイズの詳細については、「8 データ送受信」に記載しています。下記のサンプルプログラムのように 0 を指定した場合は、デフォルトの 8Kbyteが使用されます。

友達リストは、DWCFriendData構造体で友達情報の配列です。友達リストと友達情報の詳細については、「6 友達リストの作成／友達情報の作成」に記載しています。

次に、実際にサーバへ接続するために、DWC\_LoginAsync関数を使用します（コード 5-3）。

この関数の最初の引数はゲーム内スクリーンネームです。ここでゲーム中に使用するプレイヤー名があれば必ずそれを指定してください。ゲーム内スクリーンネームは認証サーバに送信され、不適切な名前でないかどうかチェックされます。

チェックの結果はDWC\_GetIngamesnCheckResult関数で取得できます（コード 5-3）。

2 つ目の引数は現在は使われていないので、NULL を渡してください。残りの引数はログイン完了コールバックとそのパラメータです。

この関数を呼出した後は、ログイン処理を進めるために毎ゲームフレーム程度の頻度でDWC\_ProcessFriendsMatch関数を呼び出してください（コード 5-3）。

DWC\_ProcessFriendsMatch 関数は、以後 DWC\_ShutdownFriendsMatch 関数を呼ぶまで、マッチメイク／友達関連機能の全ての通信処理を実行します。

ログイン完了後、他のホストへの接続を開始する間などのアプリケーションで意図したネットワーク処理がない場合でも、友達リストの更新などの通信処理が発生する可能性があるため、必ず呼び出すようにしてください。

```

static BOOL s_logged = FALSE;

void connect_to_wifi_connection( void )
{
    DWC_InitFriendsMatch( DTUD_GetUserData(),
                          GAME_PRODUCTID, GAME_SECRET_KEY,
                          0, 0,
                          DTUD_GetFriendList(), FRIEND_LIST_LEN );

    // 認証用関数を使ってログイン
    s_logged = FALSE;
    if ( !DWC_LoginAsync( L"なまえ", NULL, cb_login, NULL ) )
    {
        // 接続処理開始の失敗。
        return;
    }

    // 接続完了のポーリング
    while ( !s_logged )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // 接続処理完了
    if ( DWC_GetIngamesnCheckResult() == DWC_INGAMESN_INVALID )
    {
        // 不適切なゲーム内スクリーンネームを検出した場合の特別な処理
        display_gamesn_warning();
    }
    :
}

// ログイン完了コールバック
void cb_login( void )
{
    if (error == DWC_ERROR_NONE)
    {
        check_and_save_userdata();
        s_logged = TRUE;
    }
}

```

### コード 5-3 Wi-Fi コネクションサーバへの接続

DWC\_ShutdownFriendsMatch 関数は、マッチメイク／友達関連機能を終了し、ライブラリ内部で確保していたメモリを解放します。

DWC\_InitFriendsMatch 関数で設定したユーザーデータで初めてサーバに接続するときに、DS 本体と DS カードはペアとして扱われるようになります。ペアとして扱われるようになった場合、ユーザーデータを保存した DS カードと他の DS 本体を使い、サーバへ接続しようとした場合、接続に成功しなくなりますので、注意してください。

また、初めてサーバに接続するときに、ユーザーデータは必ず更新されます。アプリケーションでは、ログ

インコールバックを使用して、ログイン完了時にユーザーデータの更新を `DWC_CheckDirtyFlag` 関数を使用して確認し、もし必要がある場合は、DS カードに保存するようにしてください。

## 6 友達リストの作成／友達情報の作成

DWC では、プレイヤー同士の友達関係を構築するために、以下の二つの手順を用意しています。

- DS ワイヤレス通信で友達情報を交換する
- 友達コードを交換する

### 6.1 DS ワイヤレス通信で友達情報を交換する

DSワイヤレス通信では、自分のユーザーデータを元にDWC\_CreateExchangeToken関数を用いて交換用の友達情報を作成し、それを他のプレイヤーと交換します（コード 6-1）。

受信した友達情報は、アプリケーションで友達リストに保存して下さい。

```
DWCUserData s_userdata;
DWCFriendData s_friendList[ FRIEND_LIST_LEN ];

// 友達情報を交換する
void exchange_friend_data( void )
{
    int i, j;

    DWCFriendData ownFriendData;
    DWCFriendData recvFriendList[ FRIEND_LIST_LEN ];

    // 自分のユーザーデータから送信用の友達情報を作成する
    DWC_CreateExchangeToken( &s_userdata, &ownFriendData );

    // MP通信で友達情報を送受信する
    MP_start( (u16 *)&ownFriendData, (u16 *)recvFriendList );
    :

    // 受信した友達情報を友達リストの空いているところへ保存する。
    // 既に持っている同じ友達情報ならば保存しない。
    for ( i = 0; i < num_recv_data; ++i )
    {
        int index;
        for ( j = 0, index = -1; j < FRIEND_LIST_LEN; ++j )
        {
            if ( DWC_IsValidFriendData( &s_friendList[ j ] ) )
            {
                // 友達リストが有効なデータなら、受信した友達情報と
                // 同じでないか調べ、同じなら保存しない
                if ( DWC_IsEqualFriendData( &recvFriendList[ i ],
                                            &s_friendList[ j ] ) )
                {
                    break;
                }
            }
            else
            {
                // 友達リストの空きインデックスを記録しておく
                if ( index == -1 ) index = j;
            }
        }

        // 有効な被りのない友達情報を、友達リストに保存する
        if ( j >= FRIEND_LIST_LEN && index >= 0 )
        {

```

```

        s_friendList[ index ] = recvFriendList[ i ];
    }
}
:
}

```

コード 6-1 DS ワイヤレス通信による友達情報の交換

## 6.2 友達コードを交換する

一度でも Wi-Fi コネクションサーバに接続したプレイヤーは、そのプレイヤーに固有の GS プロファイル ID が割り当てられ、ユーザーデータ内に保存されています。

GSプロファイルIDを持っているプレイヤーは、GSプロファイルIDにエラーチェック用の情報を加えた「友達コード」を作成することができます（コード 6-2）。これを 10 進 12 桁の数値として表示し、他のプレイヤーと交換し、入力することで友達データの交換を簡単に行うことができます。

入力された友達コードは、DWC\_CreateFriendKeyToken関数で友達情報に変換し、友達リストに保存して下さい（コード 6-2）。

入力された友達コードの正当性はDWC\_CheckFriendKey関数でチェックすることができますが、それでも間違っている可能性はあるため、何度でも修正できるようにするユーザーインターフェースを用意して下さい（コード 6-2）。

```

DWCUserData s_userdata;
DWCFriendData s_friendList[ FRIEND_LIST_LEN ];

// 友達コードを表示する
void disp_friend_key( void )
{
    u64 friend_key;

    // 自分のユーザーデータから友達コードを作成する
    if ( ( friend_key = DWC_CreateFriendKey( &s_userdata ) ) != 0 )
    {
        // 友達コードを表示する
        disp_message( "FRIEND CODE : %ld", friend_key );
    }
    else
    {
        // 友達コードがないことを表示する
        disp_message( "FRIEND CODE : not available" );
    }
    :
}

// 友達コードから友達情報を作成し、友達リストに登録する
BOOL register_friend_key( void )
{
    u64 friend_key;
    DWCFriendData friendData;

    while ( 1 )
    {
        char friend_key_string[ 13 ];

        // 友達コードをユーザーに手入力してもらう
        input_friend_key( friend_key_string );
    }
}

```

```
// 入力された友達コードの文字列をu64の数値に変換する
friend_key = charToU64( friend_key_string );

// 友達コードの正当性をチェックし、正しければ先に進む
// 間違っていたらメッセージを表示して、もう一度入力させる
if ( DWC_CheckFriendKey( &s_userdata, friend_key ) ) break;
else display_warning_message();
}

// 正当な友達コードから友達情報を作成する
DWC_CreateFriendKeyToken( &friendData, friend_key );

{
    int index;
    // MP通信と同じやり方で友達リストの空きと被りを探し、
    // 友達情報を登録する
    :
    s_friendList[ index ] = friendData;
    :
}
}
```

コード 6-2 友達コードの交換

## 6.3 友達リストの同期処理

アプリケーションが保持している友達リスト（以下ローカル友達リスト）をインターネット上で有効にするためには、DWC\_UpdateServersAsync関数を呼び出してGameSpy社のサーバ上に格納されている友達リスト（以下サーバ友達リスト）を更新する必要があります（コード 6-3）。

この同期処理を行うためには、まず DWC\_LoginAsync 関数によるログインまでを完了させておく必要があります。

引数には、プレイヤー名（過去の仕様です。NULL を指定して下さい）、友達リスト同期処理完了コールバックとそのパラメータ、後述する友達状態変化通知コールバックとそのパラメータ、友達リスト削除コールバックとそのパラメータを指定します。

友達リストの同期処理の主な内容は、ローカル友達リストにあってサーバ友達リストにない友達に友達関係構築要求を送信することと、サーバ友達リストにあってローカル友達リストにない友達情報をサーバ友達リストから削除することです。

友達関係構築要求を送った相手がその時オフライン状態だったとしても、この要求は GameSpy サーバ上に保存され、相手が次回 DWC\_LoginAsync 関数によるログインが完了した直後に届けられます。そして、相手もこちらの情報をローカル友達リストに持っていた場合のみ、友達関係が成立します。

ただしこれは、こちらが相手を友達として登録できただけです。友達関係構築要求を受信した相手は、自動的に同様の手順を踏んで、こちらを友達として登録します。

注意しなければならないのは、友達リスト同期処理完了コールバックが呼び出されるのは、ローカル／サーバ両友達リストを全てチェックし、必要な友達関係構築要求の送信と、不要な友達情報の削除を全て終えた時であるということです。コールバックが返ってきたからといって、全ての友達関係が成立したというわけではありません。

友達リスト同期処理完了コールバックの引数 isChanged が TRUE の場合は、ローカル友達リスト中のいずれかの友達情報が更新されたことを示しており、ローカル友達リストをセーブする必要があります。友達リスト同期処理中以外に友達関係が成立した場合は、DWC\_SetBuddyFriendCallback 関数で設定した友達関係成



立コールバックが呼び出されます。

また、友達リストの同期処理の過程では、リスト内に同じ友達の友達情報を複数発見した場合には、1つを残して自動的に削除します。そして削除した友達情報の友達リスト内でのインデックスと、同じ情報と判定された友達のインデックスを引数として、削除するごとにコールバックが呼び出されます。

```
BOOL s_update      = FALSE;
BOOL s_updateFriendList = FALSE;

void sync_friend_list( void )
{
    // 友達関係成立コールバック設定
    DWC_SetBuddyFriendCallback( cb_buddyFriend, NULL );

    // ローカル友達リストとサーバ友達リストの同期処理
    if ( !DWC_UpdateServersAsync( NULL,
                                   cb_updateServers, NULL,
                                   NULL, NULL,
                                   cb_deleteFriend, NULL ) )
    {
        // 同期処理開始の失敗
        return;
    }

    while ( !s_update )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    while ( 1 )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }

        // 友達リストは不定期に更新されるため、適当な時点で以下の処理を行ない、
        // 更新されたローカル友達リストをまとめてセーブしてください。
        if ( s_updateFriendList )
        {
            // 友達リストが更新されていたらセーブする
            s_updateFriendList = FALSE;
            save_friendList();
        }

        game_loop();

        GameWaitVBlankIntr();
    }
}
```

```

    }
    :
}

// 友達リスト同期処理完了コールバック
void cb_updateServers( DWCError error, BOOL isChanged, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        // 友達リスト同期処理成功
        s_update = TRUE;

        // 友達リストが変更されていたらセーブする必要がある
        if ( isChanged ) s_updateFriendList = TRUE;
    }
}

// 友達リスト削除コールバック
void cb_deleteFriend( int deletedIndex, int srcIndex, void* param )
{
    OS_TPrintf( "friend[%d] was deleted (equal friend[%d]).\n",
                deletedIndex, srcIndex );
    s_updateFriendList = TRUE;
}

// 友達関係成立コールバック
void cb_buddyFriend( int index, void* param )
{
    OS_TPrintf( "Got friendship with friend[%d].\n", index );
    s_updateFriendList = TRUE;
}

```

コード 6-3 友達リストの同期処理

## 6.4 友達情報タイプ

友達情報にはデータタイプが設定されており、DWC\_GetFriendDataType関数でそれを取得することができます（コード 6-4）。データタイプは下記の通りです。

- DWC\_FRIENDDATA\_NODATA                      友達情報は格納されていない
- DWC\_FRIENDDATA\_LOGIN\_ID                   一度も Wi-Fi コネクションに接続していない状態の ID
- DWC\_FRIENDDATA\_FRIEND\_KEY                友達コード
- DWC\_FRIENDDATA\_GS\_PROFILE\_ID            GS プロファイル ID

データタイプが DWC\_FRIENDDATA\_LOGIN\_ID の時は、相手がまだ GS プロファイル ID を取得していない時に DS ワイヤレス通信で取得した友達情報であることを示しています。

その後、相手が GS プロファイル ID を取得したのちに、自分が友達リスト同期処理を完了させると、データタイプが DWC\_FRIENDDATA\_GS\_PROFILE\_ID に変化します。

データタイプが DWC\_FRIENDDATA\_FRIEND\_KEY の時は、友達コードで登録された GS プロファイル ID で、まだ友達関係が成立していないことを示しています。友達関係が成立すれば、データタイプが DWC\_FRIENDDATA\_GS\_PROFILE\_ID に変化します。

また、友達情報から友達関係が成立しているかどうかを、DWC\_IsBuddyFriendData関数を用いて調べることができます（コード 6-4）。

```

void disp_friendList( void )
{

```

```

int i;

for ( i = 0; i < FRIEND_LIST_LEN; ++i )
{
    // 友達情報タイプ取得
    int type = DWC_GetFriendDataType( &s_friendList[ i ] );
    OS_TPrintf( "friend[%d] type %d.¥n", type );

    if ( type == DWC_FRIENDDATA_GS_PROFILE_ID )
    {
        // GSプロフィールIDの場合は友達関係を表示する
        if ( DWC_IsBuddyFriendData( &s_friendList[ i ] ) )
        {
            OS_TPrintf( "Friendship is established.¥n" );
        }
        else
        {
            OS_TPrintf( "Friendship is not yet established.¥n" );
        }
    }
}
:
}

```

コード 6-4 友達情報タイプの取得

## 6.5 友達の状態を取得する

Wi-Fi コネクションに参加するプレイヤーは、全員自分の状態というものを保持し、それは GameSpy 社のサーバによって管理されています。アプリケーションから参照できるプレイヤーの状態には以下の二種類があります。

- 通信状態
- ステータス文字列、もしくはバイナリデータ

通信状態は、DWC\_STATUS\_\*定数で定義されており、DWC が自動的に設定します。

ステータス文字列、もしくはバイナリデータは、DWC\_SetOwnStatusString /DWC\_SetOwnStatusData関数を用いて、アプリケーションがセットすることができます（コード 6-5）。

セットすることができる文字列は NULL で終端されている必要があり、NULL 終端も含めて 256 文字までに制限されています（バイナリデータは関数内で文字列に変換され、データサイズ×1.5 倍ほどの文字数になります）。

'/'と'¥'は識別文字としてライブラリが使用するため、文字列中には使用しないでください。

友達関係が成立していれば、友達の現在の状態を取得することができます。DWC\_UpdateServersAsync関数の引数で友達状態変化通知コールバックを指定していれば、友達の状態に変化がある度にコールバックが呼び出され、友達の状態を知ることができます（コード 6-5）。

また、友達の状態取得用にDWC\_GetFriendStatus\*関数群も用意されています（コード 6-5）。

これらの関数群は、DWC が保持している友達の状態リストにアクセスするため、通信は発生しません。しかし、数 100us 単位の処理時間がかかるため、短い期間に何度も呼び出す場合はご注意ください。

プレイヤーの状態は、通信中に突然電源を切った場合などは、数分間は前の状態が残ります。

```

void sync_friend_list( void )
{

```

```
int i;

// ローカル友達リストとサーバ友達リストの同期処理
if ( !DWC_UpdateServersAsync( NULL,
                              cb_updateServers, NULL,
                              cb_friendStatus, NULL,
                              NULL, NULL ) )
{
    // 同期処理開始の失敗
    return;
}
:

// 友達リスト同期処理完了
:

// 自分のステータス文字列をセットする
DWC_SetOwnStatusString( "Location=city, level=1" );
:

for ( i = 0; i < FRIEND_LIST_LEN; ++i )
{
    if ( DWC_IsValidFriendData( &friendList[ i ] ) )
    {
        u8    status;
        char* statusString;

        // 有効な友達情報ならその友達の状態を取得する
        status = DWC_GetFriendStatus( &friendList[ i ], statusString );

        // 友達の状態を表示する
        disp_friend_status( status, statusString );
    }
}
:

// 友達状態変化通知コールバック
void cb_friendStatus( int index, u8 status, const char* statusString, void*
param )
{
    OS_TPrintf( "Friend[%d] status -> %d (statusString : %s).\n",
                index, status, statusString );
}
```

コード 6-5 友達の状態取得

## 7 マッチメイク

DWC ではマッチメイクを行うための仕組みとして、ピア型とサーバクライアント型の二つを用意しています。

ピア型マッチメイクは、各 DS をサーバ、クライアントと分けずにマッチメイクを行う方法で、更に下記の二つに分類されます。

- 友達無指定ピアマッチメイク
- 友達指定ピアマッチメイク

### 7.1 友達無指定ピアマッチメイク

不特定多数のプレイヤーとのマッチメイクを行ないます。

友達無指定ピアマッチメイクを開始するためには、DWC\_ConnectToAnybodyAsync関数を使用します（コード 7-1）。この関数の引数には、接続形態(詳しくは「8.2 接続形態」を参照して下さい)、自分を含めた最大接続希望人数、マッチメイクに条件を付加するためのフィルタ文字列、マッチメイク完了コールバックとそのパラメータ、新規接続クライアント通知コールバックとそのパラメータ、後述するプレイヤー評価コールバックとそのパラメータ、マッチメイク条件判定コールバックとそのパラメータ及びマッチメイク条件値を渡します。

マッチメイク完了コールバックは、自分がサーバホストとの接続に成功した時に呼ばれる他、自分が所属しているグループに新たにクライアントホストが加わった場合にも呼ばれます。新規接続クライアント通知コールバックは、自分が所属しているグループに、新たにクライアントホストが接続を開始した時に呼び出されます。

フィルタ文字列は、マッチメイクの候補として検索するプレイヤーを絞り込むために用います。このフィルタ文字列内で使用するためのマッチメイク指標キー（以下の例では、キー名"str\_key"と"int\_key"）は、予めDWC\_AddMatchKey\*関数を使って登録しておく必要があります（コード 7-1）。キー名はライブラリ内で保存しますが、キーの値はライブラリ内ではポインタのみ保存するので、マッチメイクが完了するまでは保持するようにして下さい。また、キー名には使用できないものがあります。詳しくは「7.10 マッチメイク指標キーに使用できないキー名」を参照してください。

```
static BOOL s_matched = FALSE;
static BOOL s_canceled = FALSE;
static const char* s_str_key = "anymatch_test";
static const int s_int_key = 10;

void do_anybody_match( void )
{
    // マッチメイク指標キーをセット
    DWC_AddMatchKeyString( 0, "str_key", s_str_key );
    DWC_AddMatchKeyInt( 0, "int_key", &s_int_key );

    // 友達無指定マッチメイク開始
    DWC_ConnectToAnybodyAsync( DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                                "str_key = 'anymatch_test' and int_key = 10",
                                cb_sc_match, NULL,
                                cb_sc_new, NULL,
                                NULL, NULL,
                                NULL, NULL, NULL );
}
```

```

// マッチメイク完了のポーリング
while ( !s_matched )
{
    DWC_ProcessFriendsMatch();

    if ( DWC_GetLastErrorEx( NULL, NULL ) )
    {
        // エラー発生。
        handle_error();
        return;
    }

    GameWaitVBlankIntr();
}

// マッチメイク完了
:
}

// マッチメイク完了コールバック
void cb_sc_match( DWCErrors error,
                 BOOL cancel, BOOL self, BOOL isServer,
                 int index, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( !cancel )
        {
            // 接続成功
            s_matched = TRUE;
        }
        else if ( self || isServer )
        {
            // 自分がマッチメイクをキャンセルした、もしくは自分が
            // クライアントホストで、サーバホストがマッチメイクをキャンセルした
            s_cancelled = TRUE;
        }
        // 新規接続クライアントがマッチメイクをキャンセルしても何もしない
    }
}

// 新規接続クライアント通知コールバック
void cb_sc_new( int index, void* param )
{
    printf( "Newcomer : friend[%d]. ¥n", index );
}

```

コード 7-1 友達無指定ピアマッチメイク

## 7.2 友達指定ピアマッチメイク

友達リストに登録された友達とのマッチメイクを行ないます。友達指定ピアマッチメイクを開始するためには、DWC\_ConnectToFriendsAsync関数を使用します（コード 7-2）。

この関数の引数には、接続形態（詳しくは「8.2 接続形態」を参照して下さい）、マッチメイクしたい友達の友達リストインデックス配列（以下インデックスリスト）、インデックスリストの要素数、自分を含めた最大接続希望人数、マッチメイク完了コールバックとそのパラメータ、新規接続クライアント通知コールバックとそのパラメータ、後述するプレイヤー評価コールバックとそのパラメータ、マッチメイク条件判定コールバックとそのパラメータ及びマッチメイク条件値を渡します。

マッチメイク完了コールバックは、自分がサーバホストとの接続に成功した時に呼ばれる他、自分が所属しているグループに新たにクライアントホストが加わった場合にも呼ばれます。新規接続クライアント通知コールバックは、自分が所属しているグループに、新たにクライアントホストが接続を開始した時に呼び出されます。

インデックスリストに NULL を指定すれば、友達リスト中の全ての友達をマッチメイクの候補とします。友達指定ピアマッチメイク時に参照される友達リストは、DWC\_InitFriendsMatch 関数で指定されたものになります。

```
static BOOL s_matched = FALSE;
static BOOL s_cancelled = FALSE;

void do_friend_match( void )
{
    // 友達指定マッチメイク開始
    DWC_ConnectToFriendsAsync(DWC_TOPOLOGY_TYPE_FULLMESH, NULL, 0, 4,
                              cb_sc_match, NULL,
                              cb_sc_new, NULL,
                              NULL, NULL,
                              NULL, NULL, NULL );

    // マッチメイク完了のポーリング
    while ( !s_matched )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // マッチメイク完了
    :
}

// マッチメイク完了コールバック
void cb_sc_match( DWCErr error, BOOL cancel, BOOL self, BOOL isServer, int
index, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( !cancel )
        {
            // 接続成功
            s_matched = TRUE;
        }
        else if ( self || isServer )
        {
            // 自分がマッチメイクをキャンセルした、もしくは自分が
            // クライアントホストで、サーバホストがマッチメイクをキャンセルした
            s_cancelled = TRUE;
        }
        // 新規接続クライアントがマッチメイクをキャンセルしても何もしない
    }
}
```

```
// 新規接続クライアント通知コールバック
void cb_sc_new( int index, void* param )
{
    printf( "Newcomer : friend[%d].¥n", index );
}
```

コード 7-2 友達指定ピアマッチメイク

## 7.3 マッチメイク候補プレイヤーを評価する

ピア型マッチメイクにおいては、マッチメイク候補として検索された複数のプレイヤーをゲーム独自の指標で評価し、マッチメイク候補としての優先順位付けを行うことができます。ピア型マッチメイク開始関数の引数に評価コールバックを指定すると、マッチメイク時に候補プレイヤーを見つける度に、指定された評価コールバックが呼び出されます。

このコールバック内では、DWC\_AddMatchKey\*関数で登録されているマッチメイク指標キーを、DWC\_GetMatch\*Value関数を用いて参照することができます（コード 7-3）。その値を元にプレイヤーを評価し、評価値を戻り値として返してください。評価値が 0 以下のプレイヤーはマッチメイクの対象から外されます。

ただし、常に評価値が最高のプレイヤーがマッチメイクの相手として選ばれるわけではなく、評価値が高いプレイヤーほど選ばれやすくなります。

```
static const char* s_str_key = "anymatch_test";
static const int s_int_key = 10;

void do_anybody_match( void )
{
    // マッチメイク指標キーをセット
    DWC_AddMatchKeyString( 0, "str_key", s_str_key );
    DWC_AddMatchKeyInt( 0, "int_key", &s_int_key );

    // 友達無指定マッチメイク開始
    DWC_ConnectToAnybodyAsync(DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                              "str_key = 'anymatch_test'",
                              cb_sc_match, NULL,
                              cb_sc_new, NULL,
                              cb_eval, NULL,
                              NULL, NULL, NULL );
}

// プレイヤー評価コールバック
int cb_eval( int index, void* param )
{
    int eval_int;

    // マッチメイク指標キー int_key の値を取得する
    eval_int = DWC_GetMatchIntValue(index, "int_key", -1);

    if ( eval_int >= 0 )
    {
        // 自分とどれだけ近い値かを評価値とする
        return MATH_ABS( s_int_key - eval_int ) + 1;
    }
    else
    {
        // キー int_key を持たないプレイヤーとはマッチメイクしない
        return 0;
    }
}
```



```
}  
}
```

### コード 7-3 マッチメイク候補プレイヤーの評価

接続形態(詳しくは「8.2 接続形態」を参照して下さい)が異なる相手とはマッチメイクが成立しませんが、この場合でも評価コールバックは呼ばれてしまいます。この時、評価コールバックの戻り値はマッチメイク結果に影響しません。

## 7.4 サーバクライアント型マッチメイク

サーバクライアント型マッチメイクは、各ホストをサーバとクライアントに役割を明確に分ける友達同士のマッチメイク方法です。サーバホストは、接続形態(詳しくは「8.2 接続形態」を参照して下さい)、自分を含めた最大接続希望人数、マッチメイク完了コールバックとそのパラメータ、新規接続クライアント通知コールバックとそのパラメータ、後述するマッチメイク条件判定コールバックとそのパラメータ及びマッチメイク条件値を指定して、DWC\_SetupGameServer関数を呼び出し、クライアントホストが接続に来るのを待ちます(コード 7-4)。

クライアントホストは、接続形態、接続したい友達の友達リストインデックス、マッチメイク完了コールバックとそのパラメータ、新規接続クライアント通知コールバックとそのパラメータ、後述するマッチメイク条件判定コールバックとそのパラメータ及びマッチメイク条件値を引数に指定して、DWC\_ConnectToGameServerAsync関数を呼び出せば、その友達がサーバホストとしてマッチメイクを開始していれば接続に行くことができます(コード 7-4)。

サーバクライアント型マッチメイク完了時には、サーバホストからは接続された全クライアントホストが相互に友達の関係ですが、クライアントホスト同士はサーバホストを介した友達の友達の関係である可能性があります。

マッチメイク完了コールバックは、自分がサーバホストとの接続に成功した時に呼ばれる他、自分が所属しているグループに新たにクライアントホストが加わった場合にも呼ばれます。

新規接続クライアント通知コールバックは、自分が所属しているグループに、新たにクライアントホストが接続を開始した時に呼び出されます。

```
static BOOL s_matched = FALSE;  
  
void do_server_match( void )  
{  
    // サーバホストとしてマッチメイクを開始する  
    DWC_SetupGameServer( DWC_TOPOLOGY_TYPE_FULLMESH, 4,  
                        cb_sc_match, (void *)CB_CONNECT_SERVER,  
                        cb_sc_new, NULL,  
                        NULL, NULL, NULL );  
  
    while ( 1 )  
    {  
        DWC_ProcessFriendsMatch();  
  
        if ( DWC_GetLastErrorEx( NULL, NULL ) )  
        {  
            // エラー発生。  
            handle_error();  
            return;  
        }  
    }  
}
```

```
    if ( s_matched )
    {
        // 新規接続クライアントとの接続が完了した場合
        init_new_connection();
        s_matched = FALSE;
    }

    GameWaitVBlankIntr();
}
:
}

void do_client_match( void )
{
    // クライアントホストとしてマッチメイクを開始する
    DWC_ConnectToGameServerAsync(DWC_TOPOLOGY_TYPE_FULLMESH, 0,
                                cb_sc_match, (void *)CB_CONNECT_CLIENT,
                                cb_sc_new, NULL,
                                NULL, NULL, NULL );

    // マッチメイク完了のポーリング
    while ( !s_matched )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // マッチメイク完了
    :
}

// マッチメイク完了コールバック
void cb_sc_match( DWCErr error, BOOL cancel, BOOL self, BOOL isServer, int
index, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( !cancel )
        {
            // 接続成功
            s_matched = TRUE;
        }
        else if ( self || isServer )
        {
            // 自分がマッチメイクをキャンセルした、もしくは自分が
            // クライアントホストで、サーバホストがマッチメイクをキャンセルした
            s_cancelled = TRUE;
        }
        // 新規接続クライアントがマッチメイクをキャンセルしても何もしない
    }
}

// 新規接続クライアント通知コールバック
void cb_sc_new( int index, void* param )
```

```
{  
    printf( "Newcomer : friend[%d].¥n", index );  
}
```

コード 7-4 サーバクライアント型マッチメイク

## 7.5 グループ ID を用いた再接続

ピア型マッチメイク・サーバクライアント型マッチメイクの双方に、一度自分が参加したグループから回線切断等で抜けてしまった後、同じグループに再度参加するための機能があります。

マッチメイク完了コールバックが成功で呼ばれたタイミング以後、参加したグループを識別するためのグループIDをDWC\_GetGroupID関数で取得することができるようになります(コード 7-5)。

```
static u32 s_groupID = 0 // 最後に参加したグループID  
  
// マッチメイク完了コールバック  
void cb_sc_match( DWCErrror error, B00L cancel, B00L self, B00L isServer, int  
index, void* param )  
{  
    if ( error == DWC_ERROR_NONE )  
    {  
        if ( !cancel )  
        {  
            // 接続成功  
            s_matched = TRUE;  
  
            // このタイミングでグループIDを取得  
            s_groupID = DWC_GetGroupID();  
        }  
        else if ( self || isServer )  
        {  
            // 自分がマッチメイクをキャンセルした、もしくは自分が  
            // クライアントホストで、サーバホストがマッチメイクをキャンセルした  
            s_cancelled = TRUE;  
        }  
        // 新規接続クライアントがマッチメイクをキャンセルしても何もしない  
    }  
}
```

コード 7-5 グループ ID の取得

自分が参加したことのあるグループに、DWC\_GetGroupID 関数で取得したグループ ID を元に再接続するには、DWC\_ConnectToGameServerByGroupID 関数を使用します。

DWC\_ConnectToGameServerByGroupIDには、接続形態(詳しくは「8.2 接続形態」を参照して下さい)、グループID、マッチメイク完了コールバックとそのパラメータ、新規接続クライアント通知コールバックとそのパラメータ、後述するマッチメイク条件判定コールバックとそのパラメータ及びマッチメイク条件値を引数に指定して呼び出します(コード 7-6)。

マッチメイク完了コールバックは、自分がサーバホストとの接続に成功した時に呼ばれる他、自分が所属しているグループに新たにクライアントホストが加わった場合にも呼ばれます。

新規接続クライアント通知コールバックは、自分が所属しているグループに、新たにクライアントホストが接続を開始した時に呼び出されます。

```
static u32 s_groupID = 0 // 最後に参加したグループID
static BOOL s_matched = FALSE;

void do_groupID_match( void )
{
    // グループIDからの再接続で、マッチメイクを開始する
    DWC_ConnectToGameServerByGroupID(DWC_TOPOLOGY_TYPE_FULLMESH,
                                      s_groupID,
                                      cb_sc_match, NULL,
                                      cb_sc_new, NULL,
                                      NULL, NULL, NULL );

    // マッチメイク完了のポーリング
    while ( !s_matched )
    {
        DWC_ProcessFriendsMatch();
        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }
        GameWaitVBBlankIntr();
    }

    // マッチメイク完了
    :
}

// マッチメイク完了コールバック
void cb_sc_match( DWCErr error, BOOL cancel, BOOL self, BOOL isServer, int
index, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( !cancel )
        {
            // 接続成功
            s_matched = TRUE;
        }
        else if ( self || isServer )
        {
            // 自分がマッチメイクをキャンセルした、もしくは自分が
            // クライアントホストで、サーバホストがマッチメイクをキャンセルした
            s_cancelled = TRUE;
        }
        // 新規接続クライアントがマッチメイクをキャンセルしても何もしない
    }
}

// 新規接続クライアント通知コールバック
void cb_sc_new( int index, void* param )
{
    printf( "Newcomer : friend[%d].¥n", index );
}
```

#### コード 7-6 グループ ID からの再接続

グループ ID からの再接続は、指定のグループに既に人が存在していない場合や、指定のグループが定員一杯である場合は失敗します。

## 7.6 締め切り

マッチメイクの規定人数に達すると、それ以上新規のクライアントが参加することはできませんが、規定人数に達していない場合でも、新規クライアントの参加を締め切ることが可能です。また、締め切り状態から受付状態に戻すことも可能です。

締め切り状態を変更するには、全参加ホストが同じタイミングで、設定する締め切り状態、締め切り状態変更コールバックとそのパラメータを指定して、DWC\_RequestSuspendMatchAsync関数を呼び出します（コード 7-7）。

全員が同一タイミングで同じ締め切り状態を設定し DWC\_RequestSuspendMatchAsync 関数を呼び出している場合に限り、締め切り処理が完了します。

例えば、ゲーム設定画面から実際のゲーム画面に移る時に必ず締め切り処理を行う等、参加者全員が同じ締め切り状態引数で DWC\_RequestSuspendMatchAsync 関数呼び出しを行う流れにする必要があります。

現在設定されている締め切り状態は、DWC\_GetSuspendMatch 関数で取得することが可能です。

```
static BOOL s_suspendCompleted = FALSE;

void do_suspend(BOOL suspend)
{
    DWC_RequestSuspendMatchAsync(suspend, suspendCallback, NULL);

    // 締め切り状態変更完了のポーリング
    while ( !s_suspendCompleted )
    {
        DWC_ProcessFriendsMatch();
        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }
        GameWaitVBlankIntr();
    }
    // 締め切り状態変更完了
}

// 締め切り完了時に呼ばれるコールバック
static void suspendCallback(DWCSuspendResult result, BOOL suspend,
                           void* data)
{
    if(result == DWC_SUSPEND_SUCCESS)
    {
        DWCDemoPrintf("締め切り状態が[%s]. に変わりました\n",
                      suspend ? "TRUE" : "FALSE");
    }
    else
    {
        // エラー
    }
    s_suspendCompleted = TRUE;
}
```

コード 7-7 締め切り処理

## 7.7 ConnectAttemptCallback

マッチメイクを開始する関数、

- DWC\_ConnectToAnybodyAsync
- DWC\_ConnectToFriendsAsync
- DWC\_SetupGameServer
- DWC\_ConnectToGameServerAsync
- DWC\_ConnectToGameServerByGroupID

の引数connectionUserData(サイズはDWC\_CONNECTION\_USERDATA\_LEN)には、アプリケーションが任意の値を設定することができます。(コード 7-8)

この値は、自分がマッチメイクに参加できるかどうかを決めるために、新規クライアントホストを受け付ける役を行っているサーバホストに送信されます。

友達指定・友達無指定マッチメイクではサーバホストが誰になるかは、アプリケーションでは意識する必要はありません。全員がサーバホストである可能性があると考えるとよいでしょう。サーバクライアントマッチメイクでは DWC\_SetupGameServer 関数を呼び出したホストが最初のサーバホストとなりますが、その後サーバホストが抜けたときにサーバ交代が発生します。

サーバホストでは、新規クライアントホストの connectionUserData を伴った DWCCConnectAttemptCallback の呼び出しを受けます。

この時点では、新規クライアントホストの受付は決定していません。DWCCConnectAttemptCallback を受けたホストは、コールバック引数に渡された新規クライアントホストの connectionUserData を元に判断を行い、受け付ける場合は TRUE を返します。

コールバックが TRUE で終了した場合、新規接続クライアントホストの受け入れが決定し、その他の接続済みクライアントホストとの接続処理が開始します。この接続処理の際に、新規接続クライアントホストの connectionUserData が全クライアントホストに通知され、共有されます。

既に接続済みのクライアントホスト(自分を含めて、AID が決定しているホスト)の connectionUserData は、DWC\_GetConnectionUserData 関数で取得することができます。この関数は、DWCCConnectAttemptCallback 内での受け入れ判断材料として使用されることを想定しています。

```
#define MALE 0
#define FEMALE 1
#define MAX_MALE_COUNT 2
#define MAX_FEMALE_COUNT 2

:
{
    u8 userData[DWC_CONNECTION_USERDATA_LEN];
    :

    // 友達無指定マッチメイク開始
    userData[0] = MALE;
    DWC_ConnectToAnybodyAsync( DWC_TOPOLOGY_TYPE_FULLMESH, 4,
                                NULL,
                                cb_sc_match, NULL,
                                cb_sc_new, NULL,
                                NULL, NULL,
                                ConnectAttemptCallback, NULL, userData );
    :
}
```

```

}

/*
 * DWC_ConnectToXXXのconnectionUserDataの[0]に、男性か女性かを示す値を設定して
 * おいて、男性・女性それぞれの上限人数を制限する場合。
 */
static BOOL ConnectAttemptCallback(u8* newClientUserData, void* param)
{
    int numAid;
    u8* aidList;
    u8 userData[DWC_CONNECTION_USERDATA_LEN];
    int i;
    int maleCount = 0;    // 男性の数
    int femaleCount = 0;  // 女性の数

    (void) param;

    numAid = DWC_GetAidList(&aidList);

    // numAid が取得できた場合、リストの中に自分も入っているの
    // ループ内で処理する。
    // numAid が取得できなかった場合、自分のデータは別途取得する
    if(numAid == 0)
    {
        if(male == USER_TYPE_MALE) ++maleCount;
        else ++femaleCount;
    }
    else
    {
        for (i = 0; i < numAid; i++)
        {
            DWC_GetConnectionUserData(aidList[i], userData);
            if (userData[0] == USER_TYPE_MALE)
                maleCount++;
            else if (userData[0] == USER_TYPE_FEMALE)
                femaleCount++;
        }
    }

    DWCDemoPrintf("male: %d female: %d\n", maleCount, femaleCount);

    if (data[0] == USER_TYPE_MALE && maleCount < USER_MAX_MALE)
        return TRUE;
    else if (data[0] == USER_TYPE_FEMALE && femaleCount < USER_MAX_FEMALE)
        return TRUE;
    else
        return FALSE;
}

```

コード 7-8 ConnectAttemptCallback の例

### 7.7.1 DWCEvalPlayerCallback と DWCCConnectAttemptCallback の違い

DWCEvalPlayerCallback も DWCCConnectAttemptCallback もマッチメイク対象の絞り込みのための仕組みですが、両者には以下のような違いがあります。

- DWCEvalPlayerCallback はサーバクライアントマッチメイクでは使用できない。  
反対に、DWCCConnectAttemptCallback はすべてのマッチメイクで使用できます。
- 絞り込み判断を行うホストが違う

- DWCEvalPlayerCallback  
新規参加クライアントホスト側で呼び出され、判断を行います。
- DWCCConnectAttemptCallback  
新規参加クライアントホストを受け付けるサーバホスト側で呼び出され、判断を行います。
- 絞り込み判断のための情報の違い
  - DWCEvalPlayerCallback  
新規クライアントホスト側でコールバックが呼び出されます。この時、コールバックに渡される引数を用いて `DWC_GetMatchIntValue` 関数 / `DWC_GetMatchStringValue` 関数を呼び出し、接続先サーバホストの情報を取得することができます。接続先サーバに既に接続している他のホストの情報を取得することはできません。
  - DWCCConnectAttemptCallback  
新規参加クライアントホストを受け付けるサーバホスト側で呼び出されます。この時、新規参加クライアントホスト及び既に参加している全ホストについて、それぞれが `DWC_SetupGameServer` 関数/`DWC_ConnectToXxx` 関数に渡した、`connectionUserData` を取得することができます。( `DWC_GetConnectionUserData` 及び `DWCCConnectAttemptCallback` の引数を使用する )
- 絞り込みタイミング
  - DWCEvalPlayerCallback  
新規クライアントホストが、これから参加するサーバホストを選択する際に呼び出されます。絞込条件により除外された相手に接続に行くことはありません。
  - DWCCConnectAttemptCallback  
サーバホストに、新規クライアントホストが接続を試みようとしている段階で呼び出されます。絞込条件により除外されたクライアントホストが、絞り込みを行ったサーバホストのマッチメイクに参加することはありません。

簡単にまとめると、以下のようになります。

- DWCEvalPlayerCallback
  - 接続先サーバホストの情報のみで、参加の可否が判断できる場合
  - サーバ・クライアントマッチメイク以外のマッチメイクでのみ使用可能
- DWCCConnectAttemptCallback
  - 参加ホスト全員の情報に応じて、動的に参加の可否を判断する必要がある場合
  - すべてのマッチメイクタイプで使用可能

## 7.8 サーバ交代

接続グループ中のサーバホストが切断してしまった場合、残っている別のホストのうち一人がサーバホスト役を引き継ぎます。これにより、最初にサーバホスト役であった人が抜けてしまっても、そのグループに新たなホストが参加することが可能になります。

サーバ交代が行われるのは、接続形態（詳しくは「8.2 接続形態」を参照して下さい）がハイブリッド型とフルメッシュ型の場合のみで、スター型のサーバホストが切断してしまった場合、他の全てのクライアントホ



ストはエラー終了します。

また、ハイブリッド型、フルメッシュ型でも、サーバ交代が発生したときは、既に新しいサーバホストと直接接続されたホストのみで継続され、それ以外のホストは全て切断されます。

サーバ交代は全てのマッチメイクタイプで発生します。

このサーバ交代機能により、友達指定マッチメイクにおいて、交代した後のサーバホストと他のクライアントホストとの友達関係が"友達の友達の友達"以上になる可能性が出てきます。

友達指定マッチメイクでは、全員が"友達の友達"の範囲に納まるように以下のような処理が行われます。

1. 最初のサーバホストについては、新規接続クライアントホストがサーバホストと友達であれば参加可能。
2. 一度でもサーバ交代が起こった後は、そのグループに新規クライアントホストは参加できなくなる。ただし、グループが自分一人だけになった場合は参加可能になる。

2.の状態になっているサーバホストについては、DWC\_GetFriendStatus 系 API で得られる返り値が DWC\_STATUS\_PLAYING になります。

サーバクライアントマッチメイクの場合は、一度サーバ交代が発生するとその後新規クライアントホストは参加できなくなります。

## 7.9 マッチメイクの高速化

友達無指定ピアマッチメイクにおいて、マッチメイキングサーバよりマッチメイク候補リストを取得する際にフィルタ機能を用いる事でマッチメイクの高速化を図ることができます（コード 7-1）。

マッチメイキングサーバに置かれているマッチメイク候補リストは、様々な条件が混在したリストとなっています。そのため無条件にリストを取得し、評価コールバック内でのマッチメイク候補の絞込みを行うと、マッチメイク不成立の可能性が高くなり、リストの取り直しとマッチメイクの繰り返しでタイムロスが発生します。

フィルタ機能を使用することによって、取得するマッチメイク候補リストをマッチメイク成立可能な候補リストにする事ができるため、不成立の可能性が下がり、マッチメイクの高速化を図ることができます。

また、同様に評価コールバック内での極端な絞込み（同レベル対戦、同地区対戦等の候補自体が少ないと考えられる条件）を行うと、マッチメイクの成功率が下がり、タイムロスが発生します。

マッチメイクの高速化を図る場合、次の点を考慮するようにしてください。

- フィルタ機能を使用し、取得するマッチメイク候補リストを成立可能な候補リストにする
- 評価コールバック内では極端な絞込みをせず、極力成立するような仕様とする

## 7.10 マッチメイク指標キーに使用できないキー名

DWC\_AddMatchKey\*関数によって登録されるマッチメイク指標キーには、ライブラリとサーバで使用されているため使用できないキー名が存在します。以下のキー名は使用しないようにしてください。

表 7-1 マッチメイク指標キーに使用できないキー名一覧

country	region	hostname	gamename	gamever	hostport
mapname	gametype	gamevariant	numplayers	numteams	maxplayers
gamemode	teamplay	fraglimit	teamfraglimit	timeelapsed	timelimit
roundtime	roundelapsed	password	groupid	player_	score_
skill_	ping_	team_	deaths_	pid_	team_t
score_t	dwc_pid	dwc_mtype	dwc_mresv	dwc_mver	dwc_eval

## 8 データ送受信

### 8.1 ピア・ツー・ピアのデータ送受信

マッチメイクが完了すると、参加しているホスト同士は相互に接続を確立された状態になっています。マッチメイク開始時に設定した接続形態の設定により、どのホスト間に直接接続が存在するかは異なります。

マッチメイクで一人参加が完了すると全ホスト間に相互接続が存在する形態、サーバとクライアント間のみ相互接続が存在する形態、及びその中間(最初はサーバクライアント間のみ、後にバックグラウンドで完全な相互接続を構築していく) 形態の3つの形態が存在します。

このマッチメイクされたグループで各ホストと直接通信を行うには、いくつか準備が必要です。

まず各ホストからのデータを受信するための受信バッファの設定です。これにはDWC\_SetRecvBuffer関数を使用しますが、引数のaidには、各ホストの識別番号であるAIDを指定します（コード 8-1）。

AID には 0～（ネットワーク構成台数-1）の数値が用いられます。例えば4人でマッチメイクを完了した場合なら、0, 1, 2, 3 の4台が存在し、AID が 1 の人がネットワークから抜ければ、残りは 0, 2, 3 となります。

受信バッファを設定する前に届いたデータは破棄されます。

次に送受信コールバックの設定を、DWC\_SetUserSendCallback関数、DWC\_SetUserRecvCallback関数で行うことができます（コード 8-1）。

受信コールバックは、他のホストからデータを受信した時に呼び出されます。送信コールバックは、送信指定したデータの送信が完了した直後に呼び出されます。

ここで言う送信の完了とは、低レイヤーの送信関数にデータを渡し終えたというだけで、相手側にデータが到着したという意味ではありません。

もう1つ、自分、もしくは他のホストが正式な接続切断の手続きを踏んで、ネットワークから離脱した場合に呼び出される、接続クローズコールバックを設定することができます。これにはDWC\_SetConnectionClosedCallback関数を用います（コード 8-1）。

```
static u8 s_RecvBuffer[ 4 ][ SIZE_RECV_BUFFER ];

void prepare_communication( void )
{
    u8* pAidList;
    int num = DWC_GetAidList( &pAidList );
    int i;

    for ( i = 0; i < num; ++i )
    {
        if ( pAidList[i] == DWC_GetMyAID() )
        {
            continue;
        }

        // 自分のAID以外の受信バッファをセットする
        DWC_SetRecvBuffer( pAidList[i],
                           &s_RecvBuffer[pAidList[i]], SIZE_RECV_BUFFER );
    }
}
```

```

    }

    // 送信コールバックの設定
    DWC_SetUserSendCallback( cb_send, NULL);

    // 受信コールバックの設定
    DWC_SetUserRecvCallback( cb_recv, NULL);

    // コネクションクローズコールバックの設定
    DWC_SetConnectionClosedCallback( cb_closed, NULL );
}

// データ送信コールバック
void cb_send( int size, u8 aid, void* param )
{
    printf( "to aid = %d size = %d\n", aid, size);
}

// データ受信コールバック
void cb_recv( u8 aid, u8* buffer, int size, void* param )
{
    printf( "from aid = %d size = %d buffer[0] = %X\n",
            aid, size, buffer[0] );
}

// コネクションクローズコールバック
void cb_closed( DWCError error, BOOL isLocal, BOOL isServer, u8 aid, int
index, void* param)
{
    if ( error == DWC_ERROR_NONE )
    {
        if ( isLocal )
        {
            printf( "Closed connection to aid %d
                    (friendListIndex = %d).\n", aid, index );
        }
        else
        {
            printf( "Connection to aid %d
                    (friendListIndex = %d) was closed.\n", aid, index );
        }
    }
}
}

```

#### コード 8-1 データ送受信のための準備

データの送信には、Reliable 送信、Unreliable 送信の二種類があり、どちらも UDP 通信を用いていますが、Reliable 送信は TCP 通信のようにパケットロスがなく、パケットの到着順が入れ替わることもない代わりに、パケットの到着を送信ごとに確認するので送信完了に時間がかかります。

Unreliable 送信は UDP 通信そのもので、上記のどちらの問題も起こりえますが、データの到着確認も再送も行わないので高速です。

DWC よりも低位のレイヤーでデータ送信が滞った場合、DWC\_InitFriendsMatch 関数でサイズを指定した送信バッファにデータが溜まっていきます。そして送信バッファに空きが足りない時に Reliable 送信しようとすると、送りきれないデータはそのまま保留され、送信バッファに空きができ次第 DWC\_ProcessFriendsMatch 関数内から送信されます。

またこれとは別に、一度に送信できる最大データサイズ（デフォルト 1465Byte）というものも決まっています。このサイズ以上のデータを送信しようとした場合も、送信データが分割され、送信が保留されます。

この最大サイズは `DWC_SetSendSplitMax` 関数で変更することができますが、様々な設定の通信機器に対応するためには、これ以上のサイズには設定しないでください。

このように送信データが保留されている間は、送信バッファを破棄しないでください。また、保留されている間は次のデータを送信できません。

送信バッファの空きや、送信先AIDが有効かどうかを含めて、Reliable送信が可能かどうかは、`DWC_IsSendableReliable`関数を用いてチェックすることができます（コード 8-2）。

Unreliable 送信時は、前述の最大データサイズを越えて一度に送信しようとすると、送信できずに `FALSE` を返します。

```
static u8 s_SendBuffer[ SIZE_SEND_BUFFER ];

void send_data( void )
{
    // 接続中の全てのホストにデータをUnreliable送信する
    // 自分のAIDを渡しても無視されます。
    DWC_SendUnreliableBtmap( DWC_GetAIDBtmap(),
                             s_SendBuffer, SIZE_SEND_BUFFER );
    :

    // 今AID=0のホストにReliable送信が可能かどうか調べる
    if ( !DWC_IsSendableReliable( 0 ) ) return;

    // 特定のホストにデータをReliable送信する。
    DWC_SendReliableBtmap( 0, s_SendBuffer, SIZE_SEND_BUFFER );
    :
}
```

### コード 8-2 データ送信

DWC は随時デバッグにデバッグ出力を行っています。デバッグ出力には多くの時間がかかることもあり、このデバッグ出力と送受信が重なってしまった場合はパケットが遅延したようにみえます。これを防ぐには `DWC_SetReportLevel` 関数でデバッグ出力を抑制してください。もしくは、デバックプリント処理内部で呼び出している `OS_TVPrintf` 関数は `Weak` シンボルでコンパイルされますので、独自にこの関数を定義することでログ出力の負荷を抑えることも可能です。

## 8.2 接続形態

新規参加者のマッチメイクが完了した後の Reliable/Unreliable 通信を行う経路の接続形態には、ハイブリッド型、スター型、フルメッシュ型の3つの形態があります。

Unreliable 通信は、全ホスト間で相互に行うことができます。Reliable 通信は、接続形態によって送信可能な相手に制限があります。

接続形態は、マッチメイクを開始する関数、

`DWC_ConnectToAnybodysync` 関数、`DWC_ConnectToFriendsAsync` 関数、`DWC_SetupGameServer` 関数、`DWC_ConnectToGameServerAsync` 関数、`DWC_ConnectToGameServerByGroupID` 関数の引数で設定することができます。

これらの関数でマッチメイクを開始する時に、希望の接続形態を設定してください。

- フルメッシュ型(`DWC_TOPOLOGY_TYPE_FULLMESH`)

新規参加ホストのマッチメイクが完了した時点で、全ホスト間に相互接続が存在し、相互に Reliable/Unreliable 通信を行うことができます。

- スター型(DWC\_TOPOLOGY\_TYPE\_STAR)

サーバホストとクライアントホストの間에만相互接続が存在します。サーバホストとクライアントホスト間は Reliable 通信を行うことができますが、クライアントホスト同士の間では Reliable 通信を行うことはできません。

Unreliable 通信は、全ホスト間で相互に行うことができます。

クライアントホスト間の Unreliable 通信は、サーバホストが自動で中継します。

- ハイブリッド型(DWC\_TOPOLOGY\_TYPE\_HYBRID)

新規参加ホストのマッチメイクが完了した時点では、サーバホストと新規接続クライアントホストの間에만相互接続が存在します。マッチメイク完了後、バックグラウンドでクライアントホスト間の相互接続が構築されていきます。

直接の相互接続があるホストとは Reliable 通信が行えます。

クライアントホスト同士の間での接続が確立したかどうかは、ライブラリからは直接通知されません。

後述する方法で、Reliable 通信が可能な相手かどうかを調べる必要があります。

(サーバホストとクライアントホスト間には必ず相互接続があるため、Reliable 通信が可能です)

Unreliable 通信は、全ホスト間で相互に行うことができます。

クライアントホスト間の Unreliable 通信は、直接接続が無い場合サーバホストが自動で中継します。

Reliable 通信が可能なホストの AID 一覧は、DWC\_GetDirectConnectedAIDBitmap 関数で取得することができます。直接の相互接続が無い相手ホストに対する DWC\_IsSendableReliable は必ず FALSE を返します。

フルメッシュ型は、マッチメイク完了時点で全ホスト間での相互接続を確立するため、最もマッチメイクに時間がかかります。また参加人数が増えるほど、マッチメイク完了までの時間が増加します。

スター型及びハイブリッド型では、新規クライアントホストとサーバホスト間の接続が確立するとマッチメイクが完了するため、フルメッシュ型より高速にマッチメイクを完了することができます。

## 8.3 コネクションを切断する

グループ中の全てのホストとコネクションを切断するには、DWC\_CloseAllConnectionsHard 関数を呼び出してください。クローズ処理が実行されたら、この関数を抜ける前に DWC\_SetConnectionClosedCallback 関数で設定したコネクションクローズコールバックが呼び出されます。

同時にクローズの通知が接続されていた他のホストにも通知され、コネクションクローズコールバックが呼び出されます。

既に接続中のホストがない場合でもこの関数を呼び出せます。この場合は、マッチメイクに使用したメモリ領域が残っていればそれを解放し、通信状態をオンライン状態に戻します。

この関数を呼び出しても、Wi-Fi コネクションサーバとの接続は切断されません。

## 8.4 DWC\_InitFriendsMatch 関数で指定するバッファサイズの見安

DWC\_InitFriendsMatch 関数で指定するバッファサイズは DWC が内部で使用するバッファサイズとなります。送信バッファは Reliable 通信で送信したデータのうち ACK が返ってきていないものを保持するために使用されます。受信バッファは正しい順番で到着しなかったデータを保持しておくために使用されます。

Reliable 通信の場合、ネットワークの瞬断に対応できるだけの容量が必要になるためゲームの仕様上許容される瞬断時間分の送受信バッファが必要となります。Unreliable 通信では、基本的に送受信バッファは使われませんが、DWC がピア・ツー・ピア接続時に内部で Reliable 通信を行っているため、送信バッファに最低 1KByte、受信バッファに最低 128Byte が必要となります。

表 8-1 通信内容とバッファサイズの見安

通信内容		バッファサイズの見安	備考
Reliable 通信	送信バッファサイズ	ゲームの仕様上許容される瞬断時間（秒）× 1 秒あたりの Reliable データ数 × Reliable データサイズ（7 × 送信データの分割数 + 送信データサイズ + 15）	最低 1KByte
	受信バッファサイズ		最低 128Byte
Unreliable 通信	送信バッファサイズ	Unreliable 通信の最大データサイズ + 2Byte	最低 1KByte
	受信バッファサイズ	最低 128Byte	

※ 送信データの分割数は、一度に送信できる最大データサイズ（DWC\_SetSendSplitMax 関数で設定される、デフォルトは 1465Byte）を超えた場合に送信データが分割される数の事を示します。

仕様上許容される瞬断時間が 1 秒、通信頻度が 3 フレームに 1 回、一度に送信できる最大データサイズ 64Byte のゲームにおいて、100Byte のデータを Reliable 通信で送る場合、必要な送受信バッファサイズは

$$1（秒） \times （60（フレーム） \div 3） \times （7 \times 2（分割） + 100（Byte） + 15） = 2580（Byte）$$

となります。

## 8.5 遅延とパケットロスのエミュレーション

DWC では、送受信データの遅延とパケットロスをエミュレーションすることができます。

ただし送信遅延に関しては、送信データを指定時間、別バッファにコピーして置いておくため、接続をクローズした時にそのデータが破棄され、相手に届きません。そのため受信遅延のみの使用を推奨します。

それぞれ、下記のようにパケットロス率（単位は%）と、遅延時間（単位はミリ秒）と、対象とするDSのAIDを指定します（コード 8-3）。

```
void set_trans_emulation( void )
{
    DWC_SetSendDrop( 30, 0 );
    DWC_SetRecvDrop( 30, 0 );

    DWC_SetSendDelay( 300, 0 );
    DWC_SetRecvDelay( 300, 0 );
    :
}
```

コード 8-3 遅延とパケットロスのエミュレーション

## 8.6 データ送受信量の目安

Reliable 通信と Unreliable 通信における通信量は下表の内容になります。

表 8-2 通信データ内訳

送信データ項目	送信データサイズ			
Preamble	192bit (24Byte)			
MAC	24Byte			
LLC	8Byte			
IP	20Byte			
UDP	8Byte			
DATA	Reliable 通信			Unreliable 通信
	ヘッダ送信	データ送信	受信確認	データ送信
	15Byte	7 + XXXByte	5Byte	XXXByte
FCS	4Byte			
B (パケットの衝突回避用ランダム時間)	MAX 600usec			

※ Reliable 通信にはデータ送信の前後にヘッダ送信と受信確認が送受信されます。

各送信におけるデータ送信時間は  $\text{Preamble} + (\text{MAC} + \text{LLC} + \text{IP} + \text{UDP} + \text{DATA} + \text{FCS}) \times 4 + \text{B} [\text{usec}]$  から求める事ができますが、帯域の状況による再送や送信パケット数、衝突回避のための送信待機等、様々な要因によって送信時間が変化するため、正確なデータ送受信量を計算する事は困難です。

そのため、本項目では実験により得られたデータ送受信量を目安として記載しています。

実験は、Reliable 通信と Unreliable 通信、AP の機種やメーカー、電波の使用率、送信サイズ、送信頻度等の条件を変化させ、スループットや CPU 負荷、パケットロス率を測定する事で行いました。その結果、おおよそ下記の内容が判明しています。



- ヘッダ分や無線通信のバックオフ時間（通信間の空き時間、パケットの衝突回避用ランダム時間等を含む）があるため、送信頻度（発生させるパケット数）の影響が大きい
  - 4 台メッシュ型、送信頻度 3 フレーム、ノイズ電波 10%以下の条件では送信サイズ 120～150byte が限界となる
  - 4 台メッシュ型、送信頻度 3 フレーム、ノイズ電波 50%程度の条件では送信サイズ 100～120byte が限界となる
  - Reliable 通信ではトラフィック混雑状態（ネットワークの混雑によって繰り返される再送で混雑が続く状態）が起こりやすく、1 度起こると復帰までに時間がかかる。
- ※ ノイズ電波は、他の DS から WMTesTool を使用して発生させています。

以上の実験結果を元に、弊社発売のタイトルでは下記のように通信を行っています。

- 「4 台メッシュ型、Unreliable 通信」  
N フレーム目 : 相手 1 に送信  
N+1 フレーム目 : 送信しない  
N+2 フレーム目 : 相手 2 に送信  
N+3 フレーム目 : 送信しない  
N+4 フレーム目 : 相手 3 に送信  
N+5 フレーム目 : 送信しない  
: (以下繰り返し)  
各 60～104byte の通信
- 「4 台サーバクライアント型、Reliable 通信」  
送信頻度を 3 フレーム、通信サイズが通常 1～40byte、最大 256byte

実際のゲーム開発においては、下記のようなケースを想定した設計を行うことが望めます。

- ネットワーク環境  
インターネットによる伝送遅延、パケットロス（国内、国外）  
伝送遅延は、国内でほぼ一定ですが、国外との通信では大きくなる傾向があります。
- 無線環境での輻輳状態  
無線環境ではパケット数が多い場合に有線と比べて輻輳状態がおきやすくなります。

送受信するパケットのサイズや、単位時間に送受信するパケット数も、上記の点を踏まえてゲームごとにチューニングしてください。尚、ゲームの内容を保障するために、回線の品質によって通信相手を限定することがやむを得ない場合の対処については「Nintendo Wi-Fi Connection プログラミングガイドライン」を参照してください。

## 9 通信エラー

DWC では、各 DWC モジュールのエラー処理を一括して行うためのシステムを用意しています。

これにより、DWC のエラーとアプリケーションのエラーを同じように扱うことができます。

### 9.1 エラー処理

DWCのエラーの状態は、DWC\_GetLastErrorEx関数で取得することができます（コード 9-1）。

戻り値はエラーの種別、引数はエラーコードとエラー処理タイプ格納先のポインタとなります。

エラーコードは 0 または負の数になります。表示する場合、符号を反転して正の数にした値を表示してください。ただし、復帰可能なエラーで Wi-Fi コネクションから切断しない場合、表示の必要はありません。

エラー処理タイプはエラー発生後に必要な復帰処理を示しており、この値ごとに定型的なエラー処理を作成することができます。

エラー状態に入るとDWCは、ほとんど全ての関数を受け付けなくなります。エラー状態から復帰するためには、DWC\_ClearError関数を呼び出してください（コード 9-1）。

```
void main_loop( void )
{
    while ( 1 )
    {
        DWC_ProcessFriendsMatch();

        handle_error(); // エラー処理

        GameWaitBlankIntr();
    }
    :
}

// エラー処理
void handle_error( void )
{
    int dwcError, gameError;

    dwcError = handle_dwc_error();
    gameError = handle_game_error();
    :
}

int handle_dwc_error( void )
{
    int errcode;
    DWCErr err;
    DWCErrType errtype;

    // エラーを取得する
    err = DWC_GetLastErrorEx( &errcode, &errtype );

    // エラーがなければ何もせずに戻る
    if ( err == DWC_ERROR_NONE ) return 0;
```

```
// エラーをクリア
DWC_ClearError();

// エラーメッセージを表示する
disp_error_message( err );
// エラーコードが0以下なら正の数にして表示する
if ( errcode <= 0 ) disp_message( "%d", -1*errcode );

if ( errtype == DWC_ETYPE_SHUTDOWN_FM )
{
    // Fri endsMatch処理を終了する
    DWC_ShutdownFri endsMatch();
}
else if ( errtype == DWC_ETYPE_DISCONNECT )
{
    // Fri endsMatch処理の終了とインターネット接続のClearupを行なう
    DWC_ShutdownFri endsMatch();
    disconnect_func();
}
else if ( errtype == DWC_ETYPE_FATAL )
{
    // Fatal Errorなので電源切断を促した後は何もできなくする
    while (1);
}
// 軽いエラーの場合はエラークリアだけでFri endsMatch処理を再開できる

return err;
}
```

コード 9-1 エラー処理

## 9.2 エラーコード一覧

マッチメイク／友達関連処理において発生する主要なエラーコードは下記の通りです。

下三桁が 010, 020 のエラーは、GameSpy 社のサーバがメンテナンスなどで不安定な状態にある時に発生しやすくなっています。

- 61010 GameSpy 社の GP サーバへのログイン中に、GP サーバとの間で通信エラーが発生しました。
- 61020 GameSpy 社の GP サーバへのログイン中に、GP サーバとの間で通信エラーが発生しました。
- 61070 GameSpy 社の GP サーバへのログイン中に、ログインのタイムアウトエラーが発生しました。
- 71010 友達リスト同期処理中に、GameSpy 社の GP サーバとの間で通信エラーが発生しました。
- 80430 サーバクライアントマッチメイクのクライアント DS において、接続先のサーバ DS、もしくはそれに接続されているクライアント DS が電源切断するなどして、接続に失敗しました。
- 81010 マッチメイク中に GameSpy 社の GP サーバとの間で通信エラーが発生しました。
- 81020 マッチメイク中に GameSpy 社の GP サーバとの間で通信エラーが発生しました。
- 84020 マッチメイク中に GameSpy 社のマスターサーバからの通信が途絶えました。マスターサーバがダウンしているか、ファイヤウォールが UDP をブロックしています。
- 85020 マッチメイク中に GameSpy 社のマスターサーバとの間で通信エラーが発生しました。
- 85030 マッチメイク中に GameSpy 社のマスターサーバの DNS に失敗しました。下三桁が 030 のエラーは全て DNS エラーを表しています。
- 86420 1 回のマッチメイク中に一定回数 NAT ネゴシエーションに失敗しました。ルータなどに問題がある可能性があります。サーバクライアントマッチメイクでは接続を開始したクライアント DS のみに発生し、1 回でも NAT ネゴシエーションに失敗するとこのエラーになります。
- 97003 マッチメイク完了後、DWC より低位のレイヤーでソケットエラーが発生しました。

下四桁が 1010, 1020 のエラーと 85020 のエラーは、NitroWiFi 1.0 RC2 以前の WiFi ライブラリにおいて GameSpy 社のサーバとの間の TCP の送信が滞ると多発することが確認されています。

## 10 ネットワークストレージ対応

DWC では、GameSpy 社によってネットワーク上に用意されているストレージサーバに、データを格納することができます。

このストレージサーバにアクセスするには、まずDWC\_LoginAsync関数によるログインまでを完了させておく必要があります。その上で、更にストレージサーバへのログインをDWC\_LoginToStorageServerAsync関数を用いて行います（コード 10-1）。

ストレージサーバにセーブできるデータは、Public か Private の属性を持たせることができます。

DWC\_SavePublicDataAsync関数を用いてデータをセーブすれば、Public属性となり、他のプレイヤーからも参照できるデータとなります（コード 10-1）。

一方、DWC\_SavePrivateDataAsync関数を用いてデータをセーブすれば、Private属性となり、他のプレイヤーからは参照できないデータとなります（コード 10-1）。

ストレージサーバからデータをロードする場合は、自分のPublicデータをロードするDWC\_LoadOwnPublicDataAsync関数とPrivateデータをロードするDWC\_LoadOwnPrivateDataAsync関数、友達リスト内の友達のデータをロードするDWC\_LoadOthersDataAsync関数を使うことができます（コード 10-1）。友達は友達リストのインデックスで指定します。

セーブ・ロード完了時には、DWC\_SetStorageServerCallback関数で設定したコールバック関数が呼び出されます（コード 10-1）。

各コールバック関数は必ずセーブ・ロード関数を呼び出した順に呼び出されます。

セーブデータとして指定できるのは、key/value の組み合わせの文字列で、"¥¥name¥¥mario¥¥stage¥¥3"のように'¥¥'で区切ってキーとその値の組を繰り返します。

このデータを指定した場合、ストレージサーバ上のデータベースには、"name"というキーの値が"mario"、"stage"というキーの値が"3"として、全て文字列で登録されます。

データをロードする時は、取得したいキーを"¥¥name¥¥stage"のように'¥¥'で区切って指定します。

この場合、ロードコールバックで取得できる文字列は"¥¥name¥¥mario¥¥stage¥¥3"という形になります。

存在しないキーや、友達が Private 属性でセーブしたキーのみをロードしようとした場合は、コールバックの引数 success は FALSE になります。ただし、複数のキーを指定したうちの、一部だけが上記のようなキーであった場合は、ロードデータにそのキーが含まれないだけで、引数 success は TRUE になります。

ストレージサーバの処理が全て終了したら、DWC\_LogoutFromStorageServer関数でストレージサーバからログアウトしてください（コード 10-1）。

```
static int s_cb_level = 0;
static BOOL s_storage_logged = FALSE;

void access_net_storage( void )
{
    // ストレージサーバへのログイン
    if ( !DWC_LoginToStorageServerAsync( cb_storage_login, NULL ) )
    {
        OS_TPrintf( "DWC_LoginToStorageServerAsync() failed. %n" );
        return;
    }
}
```

```
}

// ストレージサーバへのログイン完了待ち
while ( !s_storage_logged )
{
    DWC_ProcessFriendsMatch();

    if ( DWC_GetLastErrorEx( NULL, NULL ) )
    {
        // エラー発生。
        handle_error();
        return;
    }

    GameWaitVBlankIntr();
}

// ストレージサーバへのセーブ・ロード完了コールバック設定
DWC_SetStorageServerCallback( cb_save_storage, cb_load_storage );

// パブリックデータをセーブする
s_cb_level++;
if ( !DWC_SavePublicDataAsync( "%¥name¥¥mario¥¥stage¥¥3", NULL ) )
{
    OS_TPrintf( "DWC_SavePublicDataAsync() failed. ¥n" );
    return;
}

// プライベートデータをセーブする
s_cb_level++;
if ( !DWC_SavePrivateDataAsync( "¥¥id¥¥100", NULL ) )
{
    OS_TPrintf( "DWC_SavePrivateDataAsync() failed. ¥n" );
    return;
}

// セーブ完了待ち
while ( s_cb_level > 0 )
{
    DWC_ProcessFriendsMatch();

    if ( DWC_GetLastErrorEx( NULL, NULL ) )
    {
        // エラー発生。
        handle_error();
        return;
    }

    GameWaitVBlankIntr();
}

// 自分のPublicセーブデータをロードする
s_cb_level++;
if ( !DWC_LoadOwnPublicDataAsync( "¥¥name", NULL ) )
{
    OS_TPrintf( "DWC_LoadOwnPublicDataAsync() failed. ¥n" );
    return;
}

// 自分のPrivateセーブデータをロードする
s_cb_level++;
if ( !DWC_LoadOwnPrivateDataAsync( "¥¥id", NULL ) )
{
```

```

        OS_TPrintf( "DWC_LoadOwnPrivateDataAsync() failed. %n" );
        return;
    }

    // 他人のセーブデータをロードする
    s_cb_level++;
    if ( !DWC_LoadOthersDataAsync( "%sname", 0, NULL ) )
    {
        OS_TPrintf( "DWC_LoadOthersDataAsync() failed. %n" );
        return;
    }

    // ロード完了待ち
    while ( s_cb_level > 0 )
    {
        DWC_ProcessFriendsMatch();

        if ( DWC_GetLastErrorEx( NULL, NULL ) )
        {
            // エラー発生。
            handle_error();
            return;
        }

        GameWaitVBlankIntr();
    }

    // ストレージサーバからのログアウト
    DWC_LogoutFromStorageServer();
    :
}

// ストレージサーバ ログイン完了コールバック
void cb_storage_login( DWCError error, void* param )
{
    if ( error == DWC_ERROR_NONE )
    {
        s_storage_logged = TRUE;
        s_cb_level = 0;
    }
}

// ストレージサーバ セーブ完了コールバック
void cb_save_storage( BOOL success, BOOL isPublic, void* param )
{
    OS_TPrintf( "result %d, isPublic %d. %n", success, isPublic );
    s_cb_level--;
}

// ストレージサーバ ロード完了コールバック
void cb_load_storage( BOOL success, int index, char* data, int len, void*
param )
{
    OS_TPrintf( "result %d, index %d, data '%s', len %d %n",
                success, index, data, len );
    s_cb_level--;
}

```

#### コード 10-1 ストレージサーバへのアクセス

## 11 NITRO-DWC 3.X 以前のバージョンとの相違点

この章では TWL-DWC 5.0 以降と NITRO-DWC 3.X 以前のバージョンとの相違点・注意点について記載します。

### 11.1 マッチメイク関連

- 友達指定・無指定マッチメイクのコールバックが、サーバクライアントマッチメイクと同じ、DWCMatchedSCCallback/DWCNewClientCallback に変更されました。  
友達指定・無指定マッチメイクは、DWC 3.X 以前では規定人数が集まるまでマッチメイク完了コールバックが呼ばれませんでした。本バージョンの DWC とでは、サーバクライアントマッチメイクと同じく、一人参加する毎にマッチメイク完了コールバックが呼ばれます。
- グループ ID での再接続機能により一端抜けてしまったグループに再度参加することができるようになりました。
- 友達指定マッチメイクの、distantFriend 引数が無くなりました。参加受付中は DWC 3.X 以前の distantFriend=TRUE と同じ挙動になります。
- マッチメイク開始関数(DWC\_ConnectToAnybodyAsync/DWC\_ConnectToFriendsAsync/DWC\_SetupGameServer/DWC\_ConnectToGameServerAsync/DWC\_ConnectToGameServerByGroupID)に、接続形態設定の引数が追加されました。  
スター型やハイブリッド型に設定することで、人数が多い場合に DWC 3.X 以前より高速にマッチメイクを完了することができます。  
接続形態の設定により、相互に reliable 通信を行えない場合が存在します。
- 全てのマッチメイクタイプで、マッチメイク受け入れ判断のための DWCCConnectAttemptCallback が追加されました。
- マッチメイクオプションに関する API が廃止されました。
  - DWC\_SetMatchOption
  - DWC\_GetMatchOption
  - DWC\_GetMOMinCompState
  - DWC\_GetMOSCCConnectBlockState
  - DWC\_ClearMOSCCConnectBlock
  - DWC\_StopSCMatchAsync  
上記 API については、RequestSuspendMatchAsync 関数による締め切り機能で代用できる仕様をアプリケーションで検討してください。以下に、RequestSuspendMatchAsync 関数による締め切り機能で代用する方法の一例を示します。
  - DWC\_MATCH\_OPTION\_MIN\_COMPLETE  
マッチメイク完了は、一人接続されるごとにコールバックで通知されます。一定時間立っても最大人数に達しない場合、RequestSuspendMatchAsync 関数による締め切りを行って、最大人数以下でマッチメイクを完了することが出来ます。
  - DWC\_MATCH\_OPTION\_SC\_CONNECT\_BLOCK



RequestSuspendMatchAsync 関数による締め切り機能では、  
DWC\_MATCH\_OPTION\_SC\_CONNECT\_BLOCK と完全に同じ動作を行うことはできません。

- `DWC_StopSCMatchAsync`  
DWC\_RequestSuspendMatchAsync 関数による締め切り機能により代用できます。
- `DWCUserSendCallback`, `DWCUserRecvCallback`, `DWCUserRecvTimeoutCallback`, `DWCUserPingCallback` にアプリケーション側から任意のパラメータを渡すための (void\* param) が追加されました。
- 一定時間通信の無い相手を自動的に切断するようになりました。この時間はデフォルトで 20 秒が設定されていて、`DWC_SetConnectionKeepAliveTime` 関数で指定することができます。DWC ライブラリ内で自動的に接続維持のための通信は行いませんので、アプリケーションは本関数で設定するタイムアウト時間より短い間隔で、全接続ホストに対して何らかの reliable/unreliable 送信を行う必要があります。

© 2005–2009 Nintendo

任天堂株式会社の許諾を得ることなく、本書に記載されている内容の一部あるいは全部を無断で複製・複写・転写・頒布・貸与することを禁じます。