

すれちがい通信ライブラリ解説

Ver 1.2.2

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	4
1.1	概要	4
2	WXCライブラリの動作	6
2.1	全体の構成	6
2.1.1	WMDライバ	7
2.1.2	スケジューラ	7
2.1.3	プロトコル制御	7
2.2	ライブラリ操作手順	8
2.2.1	初期化	8
2.2.2	起動	8
2.2.3	コールバックの制御	9
2.2.4	終了	10
3	共通すれちがい通信中継所	11
3.1	すれちがい中継所の基本動作について	11
3.2	WXCライブラリの対応について	11
3.3	すれちがいデータの容量制限について	11
3.4	無効データ(ゼロデータ)への対応について	12
3.5	連続すれちがい通信への対応について	12

コード

表 1	ライブラリの初期化	8
表 2	GGIDとデータバッファの登録	8
表 3	ライブラリの起動	8
表 4	システムコールバックの実装例	9
表 5	ユーザコールバックの実装例	10
表 6	ライブラリの終了	10

図

図 1-1	一般的なDSワイヤレスプレイの接続シーケンス	4
図 1-2	すれちがい通信の接続シーケンス	5
図 2-1	WXCライブラリの全体動作概要	6
図 2-2	親子状態の変化と通信成立のタイミング	7

改訂履歴

版	改訂日	改 訂 内 容	担当者
1.2.2	2009-12-10	3.2 北米向けタイトルに関する記述追加	奥畑
1.2.1	2009-09-29	2.2.1 記述追加 (ビーコンで送信される実際の GGID に関する補足)	吉崎
1.2.0	2009-07-29	「3 共通すれちがい通信中継所」を新設	吉崎
1.1.1	2009-05-14	TwlSDK 収録による変更	北瀬
1.1.0	2006-08-10	関数名の変更 (WXC_RegisterData から WXC_RegisterCommonData へ)	吉崎
1.0.0	2005-09-22	初版	吉崎

1 はじめに

このドキュメントでは、ニンテンドーDS の環境で実現されるいわゆる「すれちがい通信」の動作概要と、それに必要な機能をモジュール化したライブラリの詳細について説明します。すれちがい通信ライブラリは Wireless Xing(Crossing) Communication ライブラリまたは略して WXC ライブラリと呼ばれます。

1.1 概要

本ドキュメントおよび WXC ライブラリでは、ユーザアプリケーションの事前設定に従って以下の全ての手順を自動的に実行する機能を総称して「すれちがい通信」と呼びます。

1. 同種アプリケーションの探索および通信確立
2. ブロック転送による 1 対 1 のデータ交換

上記手順の進捗は全て WXC ライブラリ内部で自動的に管理されるので、ユーザアプリケーションにおいては事前設定および完了後の処理のみ考慮すればよく、特にワイヤレス制御を意識する必要はありません。

一般的な DS ワイヤレスプレイのゲームでは各个体が前もって親機・子機の役割を決めておきワイヤレスを開始する必要があります。また、子機側では探索によって発見した親機群から今回希望する接続対象を選択する必要があります。これらの選択決定処理はたいていユーザインタフェースによる手作業として実装されます。このような接続シーケンスの概略を下図 1-1 に示します。

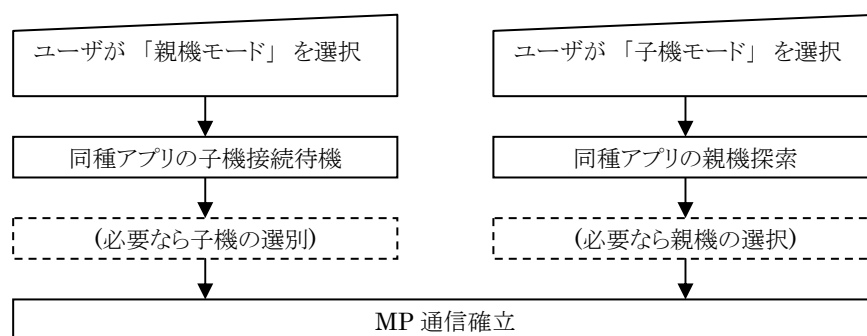


図 1-1 一般的な DS ワイヤレスプレイの接続シーケンス

すれちがい通信では、前述の選択処理がライブラリによって実行されます。

親機・子機両方の状態をランダムに繰り返すことによって個体間で親機・子機の役割が自動的に振り分けられ 1 対のワイヤレス通信が成立するようになります。また、すれちがい通信では同種アプリケーションなら任意の相手と接続することを前提としているので、受信ビーコン内の GGID を判定することによって子機側での接続対象選択もライブラリが処理します。

この接続シーケンスの概略を下図 1-2 に示します。

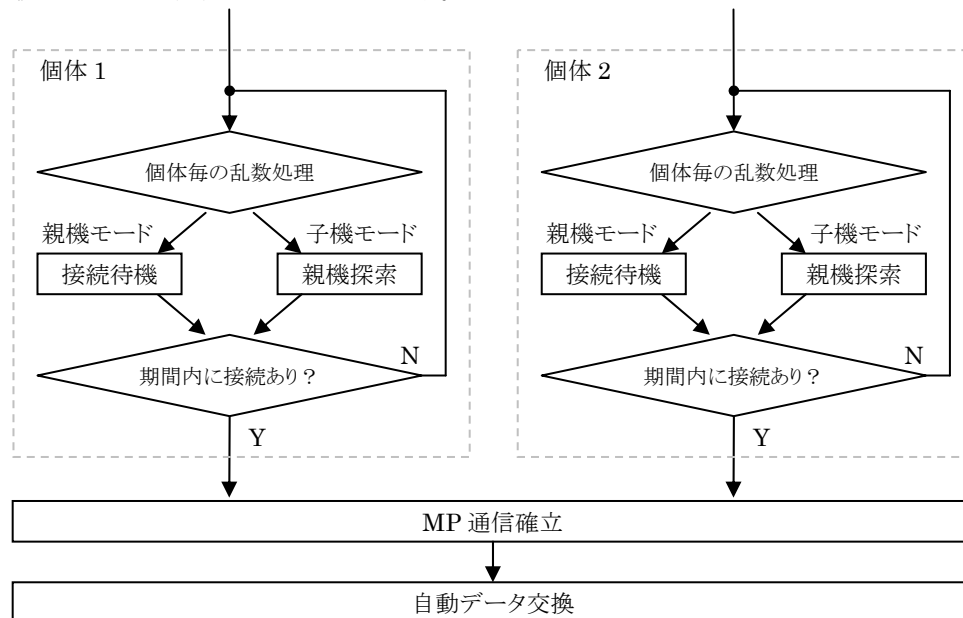


図 1-2 すれちがい通信の接続シーケンス

2 WXCライブラリの動作

この項では WXC ライブラリの内部構成とその動作および使用方法について説明します。

2.1 全体の構成

WXC ライブラリの動作の流れはおおよそ下図のとおりです。

内部処理は大きく分けてスケジューラ、WMドライバ、プロトコル制御の3つのサブモジュールから構成されます。

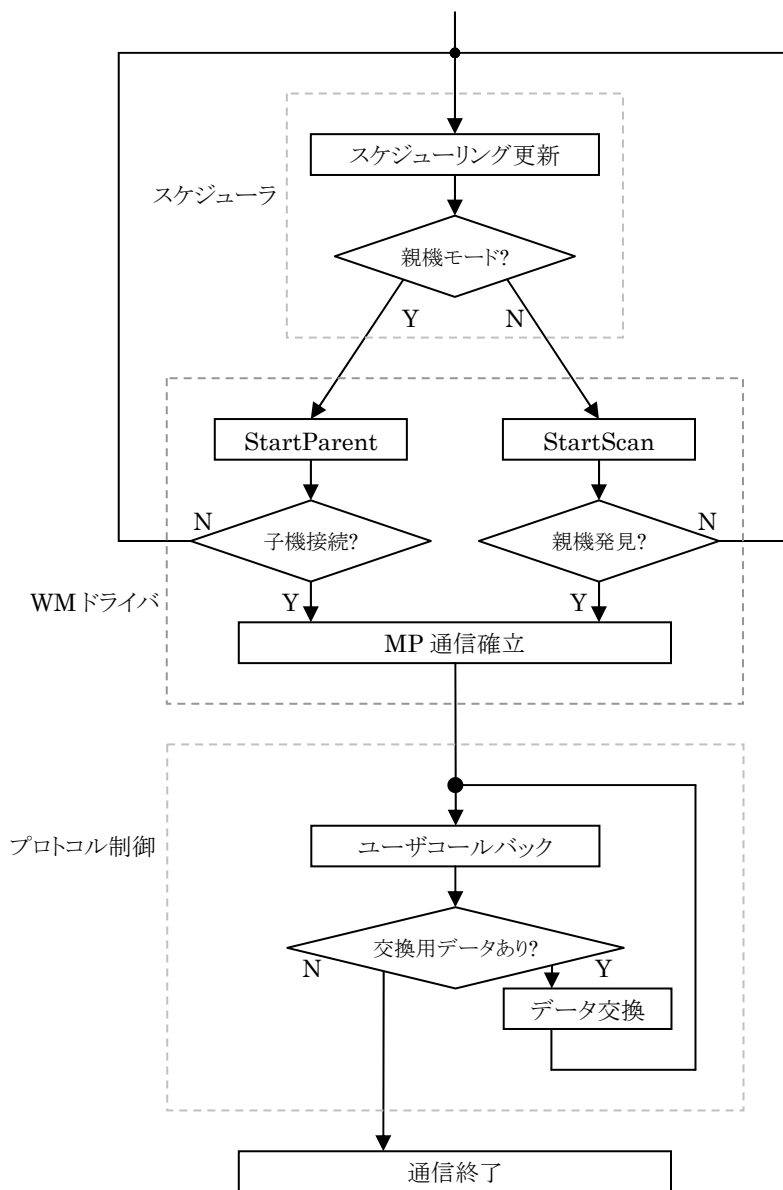


図 2-1 WXC ライブラリの全体動作概要

2.1.1 WMDライバ

WXC ライブラリは内部で WM ライブラリを使用しています。

WXC_Start 関数の呼び出しによって WM の駆動を開始し、スケジューラの判断に従ってワイヤレスステートを IDLE・MP_PARENT・MP_CHILD のいずれかへ自動的に状態遷移します。

WXC_End 関数の呼び出しによって WM の駆動は終了します。

2.1.2 スケジューラ

WXC ライブラリは親子の状態遷移を繰り返しますが、異なる個体間でこの状態遷移の周期が一致したり似通って同期してしまったりすると、子機からの探索タイミングが競合して親子の対が成立しません。

この発生頻度をなるべく低く抑えるよう、スケジューラは個体ごと時間ごとになるべくランダムな親子期間を決定します。

スケジューラによる親子状態の遷移と個体間での通信成否の例を下図に示します。

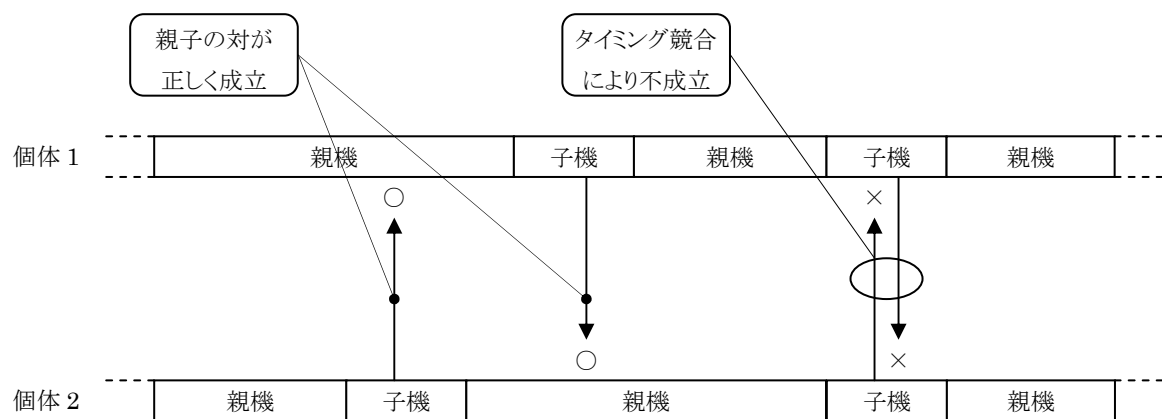


図 2-2 親子状態の変化と通信成立のタイミング

2.1.3 プロトコル制御

WXC ライブラリの MP 通信は、双方向で同時にデータ転送を実行してデータを交換するシーケンスになります。

データ転送は MB ライブラリや WBT ライブラリに採用されている受信ビットセット管理方式のブロック転送で、双方が確かに受信完了したことを確認できた時点で交換成功とみなします。通信シーケンス上、親機・子機の違いはなく対象的な処理となります。

このシーケンスには親機・子機側の両者でお互いの送受信データサイズを把握しているという前提条件があり、相手から受信しうる最長のデータを格納できる十分な受信バッファをアプリケーション側で設定しておく必要があります。

（いずれかが受信サイズを超える場合は「交換」が成立しないため、双方とも失敗とされます。）

2.2 ライブラリ操作手順

ここでは、WXC ライブラリの一般的な使用方法を解説します。

2.2.1 初期化

最初に WXC_Init 関数を呼び出し、WXC ライブラリを初期化します。

WXC ライブラリには、WXC_WORK_SIZE バイトのワークメモリと 1 個の DMA チャンネルを必要とします。

また、内部状態の変化と各種イベント発生を通知するシステムコールバックもここで指定します。

```
/* ライブラリ内部状態初期化 */  
WXC_Init(OS_Alloc(WXC_WORK_SIZE), system_callback, 2);
```

表 1 ライブラリの初期化

ここでは内部ワークメモリと各種サブモジュールの初期化のみ行われ、すれちがい通信そのものは開始しません。

次に WXC_RegisterCommonData 関数を呼び出し、GGID と送受信バッファを関連付けてライブラリに登録します。

```
/* データブロックの登録．十分な受信バッファを用意しておくこと． */  
WXC_RegisterCommonData(APP_GGID, user_callback,  
    send_buffer, sizeof(send_buffer),  
    recv_buffer, sizeof(recv_buffer));
```

表 2 GGID とデータバッファの登録

これにより、ライブラリは同じ GGID のすれちがい通信アプリケーションへ探索とデータ交換を行うようになります。

なお、ここで登録するユーザコールバックはシステムコールバックとは異なるものであることに注意してください。

なお、WXC ライブラリが MP 通信の親機として発信するビーコンではこの GGID の値がそのまま使用されることはなく、共通すれちがい通信を示す特別なビット(WXC_GGID_COMMON_BIT:0x80000000)が加算された値になります。

WmTestTool ツールなどで実際の電波状況を直接観測される場合は、この点にご注意ください。

2.2.2 起動

GGID とデータを登録したら WXC_Start 関数を呼び出して実際にすれちがい通信を開始します。

これ以降の進捗は全てシステムコールバックおよびユーザコールバックへ通知されるので、アプリケーション側でライブラリの内部状態を特に考慮する必要はありません。

```
/* ライブラリのワイヤレスを起動 */  
WXC_Start();
```

表 3 ライブラリの起動

2.2.3 コールバックの制御

コールバックには、WXC_Init 関数で指定するシステムコールバックと WXC_RegisterCommonData 関数で指定するユーザコールバックがあります。

システムコールバックには、WXC ライブラリ全体に影響する状態変化が通知されます。

通知される内容は、アプリケーション側で特に不要なら無視してかまいません。

```
static void system_callback(WXCStateCode state, void *arg)
{
    switch (state)
    {
        /* 以下は状態変更 API の呼び出しの中から通知され、特に使用しません。 */
        case WXC_STATE_READY: /* WXC_Init 関数呼び出しの内部から発生。 */
        case WXC_STATE_ACTIVE: /* WXC_Start 関数呼び出しの内部から発生。 */
        case WXC_STATE_ENDING: /* WXC_End 関数呼び出しの内部から発生。 */
            break;
        case WXC_STATE_END: /* WXC_End 関数による終了処理完了時に発生。 */
            /*
             * arg は WXC_Init 関数で割り当てた内部ワークメモリ。
             * この時点でワークメモリはユーザに解放されているので、
             * 動的に確保したならここで破棄できます。
             */
            {
                void *system_work = arg;
                OS_Free(system_work);
            }
            break;
        case WXC_STATE_CONNECTED: /* 新規の接続が発生した場合に発生。 */
            /*
             * arg は接続対象を示す WXCUserInfo 構造体へのポインタです。
             * 何か特殊な制御をする場合にのみここで処理を記述します。
             */
            {
                const WXCUserInfo * user = (const WXCUserInfo *)arg;
                OS_TPrintf("connected(%2d):" user->aid);
            }
            break;
    }
}
```

表 4 システムコールバックの実装例

ユーザコールバックには、データ交換が完了したときに通知が発生します。
引数には、アプリケーションで指定した受信バッファがデータ格納済みの状態で与えられます。

```
static void user_callback(WXCStateCode stat, void *arg)
{
    /* stat は、現状では常に WXC_STATE_EXCHANGE_DONE になります */
#pragma unused(stat)
    /* arg は受信データを格納した WXCBlockDataFormat 構造体のポインタです. */
    const WXCBlockDataFormat * block = (const WXCBlockDataFormat *)arg;
    u8      *recv_data = (u8 *)block->buffer;
    u32      recv_length = block->length;
    /*
     * ここでアプリケーション固有の処理を行います.
     * この場で WXC_End 関数を呼び出すことも可能です.
     */
}
```

表 5 ユーザコールバックの実装例

2.2.4 終了

すれちがい通信をこれ以上続ける必要が無くなれば、WXC_End 関数を呼び出してライブラリの終了を指示します。
この呼び出しのあと実際に終了処理が全て完了した時点で WXC の内部状態は WXC_STATE_END に変化し、システムコールバックへ通知されます。

```
/* ライブラリのワイヤレスを終了 */
WXC_End();
/*
 * 終了処理はただちに完了しません。
 * システムコールバックで完了通知を待つか
 * 定期的に WXC の内部状態をチェックします。
 */
while (WXC_GetStateCode() != WXC_STATE_END)
{
    OS_Sleep(100);
}
```

表 6 ライブラリの終了

3 共通すれちがい通信中継所

WXC ライブラリを使用するゲームタイトル同士がデータ交換の機会をより多く得られるように、「共通すれちがい通信中継所」(以下、すれちがい中継所)が設置されている場所があります。WXC ライブラリを使用するゲームタイトルは、すれちがい中継所とデータ交換しても誤動作しないようにいくつかの注意事項を考慮しておく必要があります。この項では、すれちがい中継所のおおまかな動作原理とおもな注意事項について解説します。

3.1 すれちがい中継所の基本動作について

すれちがい中継所は、WXC 使用ゲームタイトルが付近を通りかかると自動的にデータ交換を実行し、得られたデータはいったん保持しておき、次に通りかかる同種のゲームタイトルとのデータ交換にそのまま使用します。この動作を繰り返すことにより、すれちがいデータの中継所として振舞います。

中継所のこのような挙動を考慮し、ゲームタイトルでは以下 2 点にご注意ください。

- (1) 「中継」という機能の性質上、1 対 1 での純粋な交換にはならない
通常のすれちがい通信では、ゲームタイトル同士での 1 対 1 のデータ交換が起こりますが、中継所を介したすれちがい通信では、1 対 1 のデータ交換は起こりません。ですので、特定のペアでないと交換が意味をなさない、Wi-Fi フレンドコードの交換などには利用できません。
- (2) 中継所を介したすれちがい通信によって、すれちがいデータが複製されたり消失したりする可能性がある
すれちがいデータが複製されたり(3.4で後述)、すれちがいデータが消失(中継所内部仕様である 1 時間に 1 回のセーブタイミングなどに起因)したりすることがあります。そのため、すれちがいデータの複製・消失がゲームの世界観や設定を壊さないかどうかをご確認いただくだけでなく、交換されるすれちがいデータがゲーム内で 2 つ存在することや、交換されるすれちがいデータが消失することによって、誤動作を起こす可能性がないようご注意ください。

3.2 WXCライブラリの対応について

中継所に対応するためには、WXC ライブラリをご使用いただく必要があります。

以下の 3 点は、中継所対応タイトルが WXC ライブラリを組み込む上の注意事項になります。

- (1) 必ず弊社提供の WXC ライブラリを使用すること。
- (2) 新設の WXC_RegisterCommonData 関数を必ず使用し、旧仕様の WXC_RegisterData 関数は使用しないこと。
- (3) SDK 付属の中継所テストプログラム(\$TwlSDK/bin/ARM9-TS/Rom/RelayStation.srl)が正しく応答するように、動作確認すること。
- (4) [北米向けタイトルのみ] SDK 付属の DS Download Station テストプログラム(\$TwlSDK/bin/ARM9-TS/Rom/DSDownloadStation.srl)が正しく応答するように、動作確認すること。

これらを守って実装していただいた上で、さらに下記の項目を考慮した設計であることが必要です。

3.3 すれちがいデータの容量制限について

すれちがいデータのデータ長は 32K byte までです。それより大きいサイズのすれちがいデータは WXC ライブラリによ

って制限されております。中継所とのすれちがい通信だけでなく、ゲームタイトル同士の「通常のすれちがい通信」でもこのサイズを超えるデータは受け取りませんのでご注意ください。

3.4 無効データ(ゼロデータ)への対応について

無効データ(ゼロデータ)とは、中継所にとって初めてすれちがう種類のゲームタイトルに対して初回のみ送られるデータです。中継所はその種類のゲームタイトルと交換できる持ち合わせのデータがないため、初回だけサイズが 0 バイトのすれちがいデータを使用してデータ交換します。これは対応タイトルにとって無効なすれちがいデータとなりますので、中継所対応タイトルはこの無効データを受け取ったときの状況を想定し、処理を記述しておく必要があります。

ゼロデータに対応する処理として、たとえば以下のような手法が考えられます。

- (1) ゲームがすれちがい通信によって無効データを受け取ったとき、ゲームの ROM 内から別のデータを読み出し、それを受け取ったように見せかける(すなわち、ユーザからは、誰かとの正常なすれちがい通信が行なわれたように見える)。
- (2) ゲーム内でセットした「相手に送るためのすれちがいデータ」(以下、送信データ)を中継所の無効データと交換するが、交換した旨をゲーム内で一切表示せず、交換が行なわれなかったように見せかける。その際、交換終了後、ゲーム側には同じ送信データを再びセットしておく(すなわち、ユーザからは、すれちがい通信が行なわれていないように見える)。

ただしこの方法では、ゲーム内に送信データを残したまま中継所にも同じ送信データが保存されることになるため、例えば以下のような手順で同一アイテムが複製される可能性があります。

1. 送信データにアイテム甲をセットし、中継所の無効データとすれちがい通信で交換する。
ゲーム内ではすれちがい通信が起こっていないように見せてアイテム甲をセットしたままにしておくが、中継所はアイテム甲を受け取ったものとみなすため、同じアイテム甲が双方に 1 個ずつ存在する状態になる。
2. 送信データにセットされているアイテム甲を外し、アイテム乙をセットする。
3. セットしたアイテム乙を、中継所のアイテム甲とすれちがい通信で交換する。
ゲーム内では手順 1 で送信していたアイテム甲を再び中継所から受け取り、すでに手順 2 で外しておいた分とあわせてアイテム甲は 2 個に増える。

この現象は、このゲームタイトルが初めて中継所とすれちがい、かつ中継所内に登録されるときにしか起こらないので、起こるとしても確率は低いと考えられます。ですので、アイテムの複製の可能性を考慮した上で、「すれちがいデータの中にユニークな識別番号を含ませて、自分自身のすれちがいデータを受け取ることを防止する」などとしてアイテムの複製を防ぐという対処法が考えられます。

3.5 連続すれちがい通信への対応について

「連続すれちがい通信」とは、同じプレイヤーが 2 回以上連続で中継所とすれちがう状況を指します。このとき、プレイヤーは自分でセットした送信データを自分で受け取ることになります。

たとえば、「アイテムやメッセージを受け取るが、その受け取ったデータをそのまま次に送る送信データとしてセットできない」というタイプのすれちがい通信を実装しているゲームタイトルの場合、このゲームタイトル同士でデータ交換を行うぶんには仕様としてどうやっても自分のデータを自分で受け取る状態は作り出せません。

しかし、中継所は「受け取ったすれちがいデータを常に単純に送り返す」仕様になっておりますので、中継所とのデータ交換時にはこの不可能な状況が起こりえてしまうことになります。中継所対応のタイトルを作成される開発者の方は、この点にご留意いただき、プレイヤーが自分でセットしたデータを自分で受け取る現象によってゲームが誤動作しないように開発していただけるよう、お願いいたします。

自分の送信データを自分で再び受け取らないようにするためには、[3.4](#)で記述しましたように、すれちがいデータの中にユニークな識別子を仕込むことで、自分が送ったデータかどうかを判別する方法も考えられます。ゲームのすれちがい通信の仕様に合わせてご検討ください。

© 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。