

# すれちがい通信ライブラリ解説

Ver 1.0.0

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	WMドライバ.....	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

## 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

## 改訂履歷

版	改訂日	改 訂 内 容	担当者
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およびそのパッケージは 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、NITRO-SDK とは別の単体パッケージとして公開されています。

以降、このドキュメントでの解説においては NITRO-SDK のインストール先を \$NitroSDK と表記し、WXC ライブラリのインストール先を \$WXC と表記します。

## 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個体が前もって親機・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。

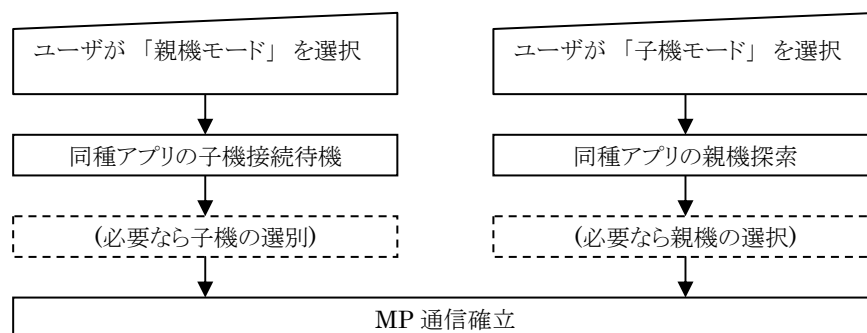


図 1-1 一般的な DS ワイヤレスプレイの接続シーケンス

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。

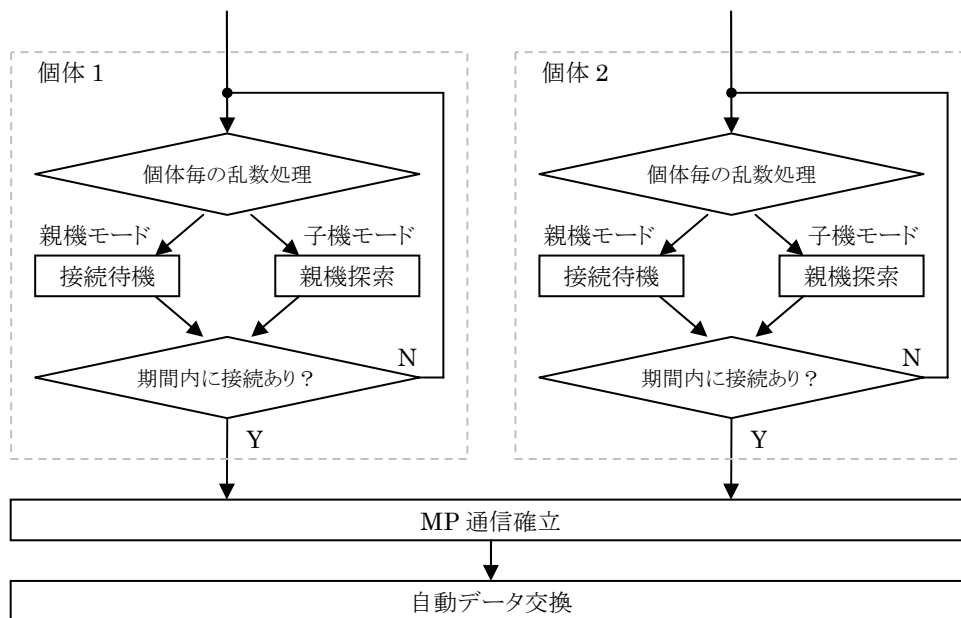


図 1-2 すれちがい通信の接続シーケンス

## 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。

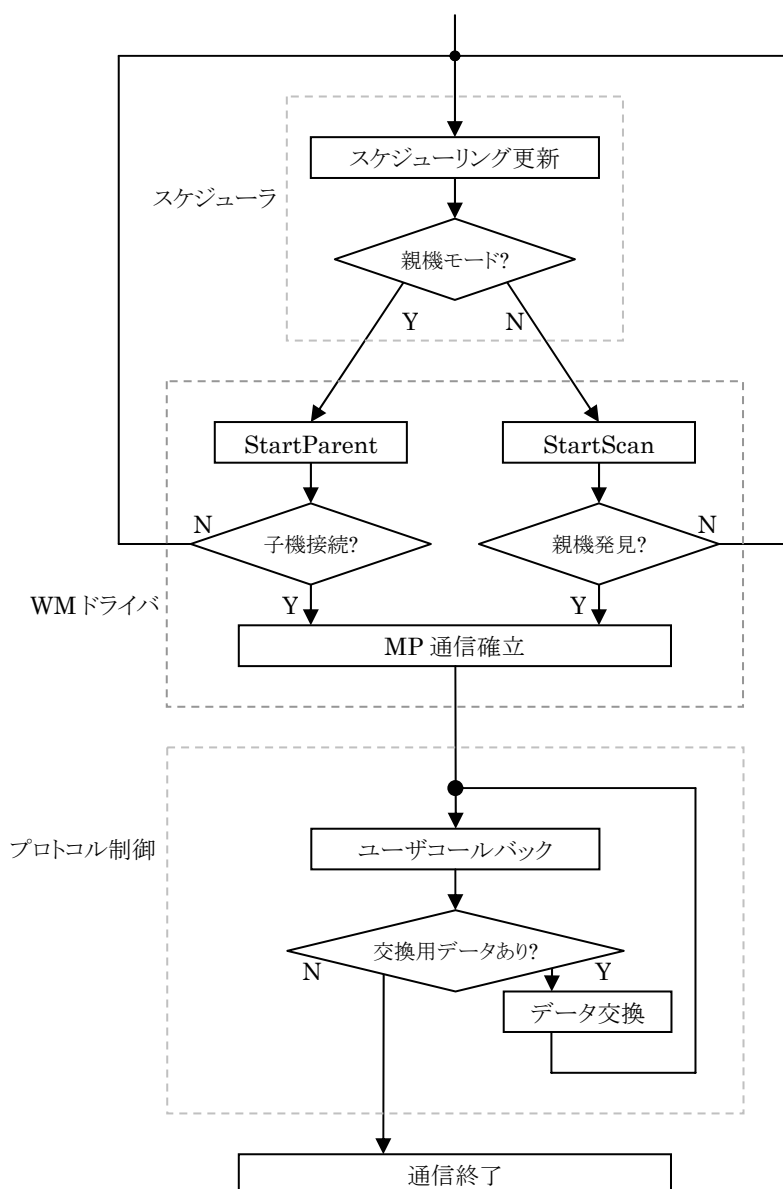


図 2-1 WXC ライブラリの全体動作概要

### 2.1.1 WMドライバ

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。

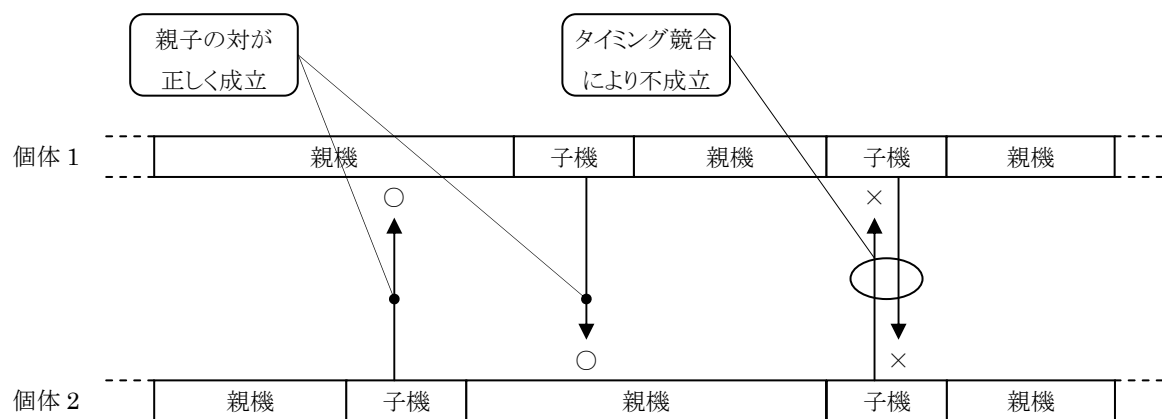


図 2-2 親子状態の変化と通信成立のタイミング

### 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・子機の区別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\_RegisterData 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ておきます。

```
/* 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_RegisterData(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。

### 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\_RegisterData 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

© 2005 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。