TwlSDK

# TWL-SDK 移行ガイド

## NITRO-SDK から乗り換え時の注意事項

2009-05-20

任天堂株式会社発行

# 改訂履歴

| 改訂日 | 改 訂 内 容 |
|---|---|
| 2009-05-20 | 比較対象の TWLSDK を TWL-SDK 5.1 から TWL-SDK 5.2 に変更<br>5.1.2 OS_CreateExtraHeap()、OS_AddExtraAreaToHeap()の説明を追加 |
| 2009-03-10 | 比較対象の TWLSDK を TWL-SDK 5.1　PR, RC から TWL-SDK 5.1 に変更<br>3.1, 5.1 Wi-Fi 関連の記述を削除<br>4.7 WM_Initialize()内部の挙動の説明を追加<br>4.12.3 CodecMode について説明を追加<br>4.12.4 CardRegion の設定を追加<br>5.2 クローンブートの対応についてを削除<br>7.6.3 RomSize と RomFootPadding を追加 |
| 2009-02-23 | 2.11 MATH の MD5 関数の非推奨化を追加<br>5.1 HYBRID ROM に移植時のメモリ増加量を約 15KByte に変更<br>5.1.1 カートリッジライブラリ(CTRDG)の無効化を追加<br>5.1.2 メモリの先頭領域の使用を追加 |
| 2009-02-06 | 3.1 ライブラリが利用するメモリ使用量の増加を追加<br>5.1 ライブラリが利用するメモリ使用量の増加を修正 |
| 2009-01-07 | 4.12.2 WramMapping で指定する値を変更 |
| 2008-12-18 | 3.1 Static モジュールの先頭アドレスの一時的な変更を削除<br>7.7.1 FS エラーの追加を 7.5.8 に移動<br>7.7.5 FS_ReadFile()時のメモリのアライメントによる高速化を削除<br>7.7.6 FS_ReadDirectory()の速度の低下の削除<br>7.7.7 ファイル ID を使用する関数の削除 |
| 2008-11-17 | 初版。 |
| | |

# 1 はじめに

このドキュメントではこれまで NITRO-SDK で開発されてきた方に向けて、TWL-SDK に移行する際に注意すべき話題について取り扱います。

既に手元にある NITRO-SDK のプロジェクトを TWL-SDK にビルドし直して動かすことを目的にしています。とりあえず動くということを目的としているため、新機能の紹介はしておりません。TWL-SDK に移行した際に、修正が必要となる部分のみ取り上げています。

変更点は NITRO-SDK 4.2 plus8 パッチと TWL-SDK 5.2 で比較しています。

NITRO / HYBRID / LIMITEDと呼ばれる異なるタイプのROMごとに注意事項を記載します。 ROMのタイプについて詳細は、TWL-SDK アプリ開発ガイドをご参照ください。

# 2 すべてのアプリケーションに共通の注意点

NITRO / HYBRID / LIMITED のどの ROM に移植する際にも、共通の注意点を以下に挙げていきます。

## 2.1 NITROSDK_ROOTの環境変数の修正

NITRO-SDKで使用していた環境変数NITROSDK_ROOTは、TWL-SDKでは、TWLSDK_ROOTに変更され、新しくTWLSDK_PLATFORMでプラットフォームを設定する必要があります。詳細は、Quick Start Guideを参照してください。

## 2.2 OS_GetConsoleType()の戻り値にOS_CONSOLE_TWLを追加

OS_GetConsoleType()の戻り値として、TWL-SDK では OS_CONSOLE_TWL が追加されています。TWL 実機上で TWL モードとして動いているアプリは、OS_CONSOLE_TWL が戻ります。TWL 実機上で NITRO モードとして動いているアプリは、DEBUG と RELEASE モードの場合のみ OS_CONSOLE_TWL が戻りますが、FINALROM では OS_CONSOLE_NITRO が戻ります。

## 2.3 無線を停止せずにスリープした場合、OS_Panic()で止まります

NITRO-SDK では、無線を停止せずにスリープしても動いていましたが、TWL-SDK では OS_Panic()で停止するようになりました。これは無線を動かしたままスリープに遷移するのを禁止したガイドラインに対応したものです。

## 2.4 デバッガー用のライブラリのリンクによるメモリ使用量の増加

IS-TWL-DEBUGGER ソフトウェアと IS-NITRO-DEBUGGER ソフトウェアの両方をインストールしていた場合、両方のデバッガー用ライブラリがリンクされ、それによってアプリケーションが使用できるメモリが減ることになります。

## 2.5 デバッグ出力の処理時間の増加

IS-TWL-DEBUGGER を使用した場合、IS-NITRO-DEBUGGER の場合に比較して、OS_Printf()等のデバッグ

出力に必要な時間が約 2 倍に増えています。

## 2.6 WCサンプルライブラリの削除

WM や MB、WBT ライブラリのサンプルデモで使用されていた WC ライブラリを全て WH ライブラリに切り替えました。この移植に伴い、WC ライブラリを削除しました。

## 2.7 STD関数の高速化

STD_GetStringLength()、STD_CopyLString()、STD_CompareString()、STD_CompareNString()の関数が高速化されています。

## 2.8 STD_TSPrintf()の%n書式のサポート廃止

セキュリティ上の問題があるため、STD_TSNPrintf()，STD_TSPrintf()，STD_TVSNPrintf()，STD_TVSPrintf()の%n 書式のサポートを廃止しました。

## 2.9 MATH_FFTReal(), MATH_IFFTRead()の不具合修正

NITRO-SDK では、MATH_FFTRead()，MATH_IFFTRead()に不具合がありましたが、TWL-SDK では修正を行っています。

## 2.10 SVC_IsMmemExpanded ()の削除

NITRO-SDK に含まれていた SVC_IsMmemExpanded()関数は TWL-SDK では削除されました。メインメモリのサイズを取得する際は、OS_GetConsoleType()をお使いください。

## 2.11 MATHのMD5 関数の非推奨化

NITRO-SDKに含まれていたMATHのハッシュ関数のうち、MD5 はセキュリティ上に弱点があるため、非推奨になりました。そのためTWL-SDKでは関数リファレンスから削除されています。ハッシュ関数が必要であればSHA-1 をご使用下さい。詳細は関数リファレンスの「数学演算(MATH)」→「概要」→「ハッシュ・メッセージダイジェスト」をご参照ください。MD5 のヘッダと関数は互換性のために残されています。

# 3 NITRO ROMへ移行する際の注意点

TWL-SDK において、NITRO ROM を作成する場合にのみ必要な注意点を挙げます。

## 3.1 ライブラリが利用するメモリ使用量の増加

TWL-SDK のライブラリが消費するメモリは、機能追加と不具合修正のため、NITRO-SDK と比較すると増加しています。そのため、NITRO-SDK からアプリケーションを移植した場合に、アプリケーションが使えるメモリ領域は減少します。一例として TWL-SDK 5.2 を使用し、過去の DS プロジェクトを移植した場合には、ライブラリが消費するメモリは

約 9Kbyte 増加します。

NITRO-SDK の全ての関数をリンクしていた場合約 11KByte 増加します。Thumb コードを使用した場合は約 7Kbyte 増加します。但しライブラリ内にある inline 関数が増えていた場合、複数回コールされた分メモリ使用量が増加しますので、実際消費するメモリ量はこれより大きくなることがあります。

# 4 HYBRID/LIMITED ROM共通の注意点

TWL-SDK において、HYBRID/LIMITED ROM に移行する際の共通の注意点を挙げます。

## 4.1 TWLモードでのROMへのアクセス速度の低下(カードアプリ)

カードアプリにおいては、TWLモードのROMのアクセス速度は、NITROモードのROMのアクセス速度に比較して低下しています。これは特にHYBRIDアプリにおいて、NITROモードで起動した場合と、TWLモードで起動した場合とでアプリの挙動が異なる原因となりますので、ご注意ください。詳細は関数リファレンスの「ファイルシステム(FS)」→「概要」→「ROMアーカイブ」をご参照ください。

## 4.2 FS_Init()の前にOS_EnableIrq()を呼び出す必要性

TWL モードで動作する場合においては、FS_Init()の前に OS_EnableIrq()を呼んでおく必要があります。呼んでいない場合は、FS_Init()内で OS_TPanic()で止まります。NITRO ROM や HYBRID ROM を NITRO モードで動かした場合には必要ありません。

## 4.3 libsyscall.twl.aとrom_header.LTD.sbinの追加

NITRO アプリにおいては、弊社窓口が libsyscall.a と rom_header.template.sbin をアプリごとに配布しており、これは TWL アプリにおいても同じ対応になります。TWL アプリでは新たに libsyscall.twl.a と rom_header.LTD.sbin を追加する必要があります。これらのファイルは TWL-SDK の$TwlSDK/lib/ARM9-TS/etc/libsyscall.twl.a、$TwlSDK/tools/bin/rom_header.LTD.sbin を使用してください。

## 4.4 Staticモジュール先頭アドレスの修正

Static モジュールの先頭アドレスが、0x02004000 に変更されています。この先頭 16Kbyte の領域はシステムのパラメータ領域として使用しているため、開発者が使用することはできません。注意が必要な点として HYBRID アプリにおいて NITRO モードで起動した場合にも同じような影響をうけるため、使用可能なアリーナ領域が 16KByte 減ることになります。また NITRO-SDK のいくつかのサンプルデモの lsf ファイルは、Static の Address が 0x02000000 となっているため、これらの lsf ファイルを利用している場合、移行する際にこの値を 0x02004000 または、$(ADDRESS_STATIC) に修正する必要があります

## 4.5 NTRバナーをTWLバナーに変更

NITROで使用していたバナーはNTRバナーと呼ばれるものですが、TWL対応アプリではTWLバナーに作りかえる必要があります。具体的には、makebannerツールではなく、makebanner.TWLツールでバナーを作りなおす必要があります。かならずしもアニメーションバナーにする必要があるわけではありません。詳細は、関数リファレンスの「ツー

ル」→「概要」→「バナー」をご参照ください。

## 4.6 DSダウンロードプレイの子機について

DS ダウンロードプレイ子機として起動できるのは NITRO ROM のみです。HYBRID ROM や LIMITED ROM はブートできません。DS ダウンロードプレイ親機は全ての ROM・全ての起動モードで利用できます。

## 4.7 無線通信設定OFFの取得

TWL 本体では無線通信の設定を OFF にすることができます。OFF になった本体では、アプリケーションは WM_Initialize()で WM_ERRCODE_WM_DISABLE を返して失敗します。WM_Initialize()内部では、WM_Init()が WM_ERRCODE_WM_DISABLE を返して失敗しています。もしユーザーに無線通信設定を変更させたい場合は、本体設定の無線通信設定を OS_IsAvailableWireless()で取得し、OS_JumpToWirelessSetting()で本体設定の無線通信項目に遷移してください。ただし HYBRID アプリの NITRO モードでは設定の取得や、遷移はできません。また TWL 本体で無線通信の設定が OFF でかつ NITRO モードで動いている場合は WM_Initialize()や他の無線ライブラリがエラーを返すことはありませんが、無線通信機能は動かないため、通信相手が見つからない状態になります。

## 4.8 ARM7 コンポーネント ichneumonの変更

HYBRID, LIMITED アプリにおいては、NITRO で使用できた VRAM 上にワイヤレスコードを置いた ichenumon コンポーネントに該当するものは提供しておりません。HYBRID においては、mangoose, LIMITED においては racoon コンポーネントをご使用ください。

## 4.9 OS_SpinWait()を使う上での注意

OS_SpinWait()を使う場合において、TWL モードではクロックが NITRO の頃に 2 倍になっていることに注意する必要があります。特に HYBRID アプリにおいては、NITRO モードで動かした場合と、TWL モードで動かした場合とでは関数から返るまでの時間が変わることになります。NITRO モードと TWL モードで同じ時間待ちたい場合は OS_SpinWaitSysCycles()を使ってください。

## 4.10 ペアレンタルコントロールの設定

マスターROM を作成するには、SRL ファイルに MasterEditorTWL でペアレンタルコントロールの設定をする必要があります。詳細は、「TWL マスターエディタ 解説書」をご参照ください。

## 4.11 CTRDG, VIBライブラリの使用について

TWL 本体で動作しないカートリッジ(CTRDG)や振動カートリッジ(VIB)関数ですが、これらは HYBRID, LIMITED アプリでリンクしていても問題はありません。TWL 本体においてこれらの関数は、カートリッジがささっていないように振舞います。

## 4.12 RSFファイルの修正

カードアプリに関しては $TwlSDK/include/twl/specfiles/ROM-TS.rsf を元に、TitleName, MakerCode,

RemasterVersion, RomSize, RomFootPadding, RomHeaderTemplate, BannerFileを設定してください。ここでは、特に注意すべき点についてのみ挙げます。詳細は、関数リファレンスの「ツール」→「ROMイメージ関連」→「makerom.TWL」をご参照ください。

### 4.12.1 RomSpeedTypeの修正(カードアプリ)

これまでのプロジェクトで RomSpeedType にマスク ROM(MROM)を使用していた場合、HYBRID アプリ、LIMITED アプリでは使用できなくなりますので、ワンタイム PROM(1TROM)に変更してください。この変更によりROM へのアクセス速度が低下しますのでアプリの動作にご注意ください。

### 4.12.2 WramMappingの設定

WramMapping を HYBRID アプリであれば MAP2_TS_HYB、LIMITED アプリであれば MAP2_TS_LTD にする必要があります。

### 4.12.3 CodecModeの設定

TWL には、マイク入力やスピーカー出力、タッチスクリーンへの入力などを制御する CODEC というハードウェアが搭載されています。RSFファイルにはTWLモードで動作した場合のCODECを設定します。NTR(CODEC-DSモード)を設定すれば、CODEC がNITRO互換で動作しますが、カメラを使用できなくなります。カメラを使用する場合は、TWL(CODEC-TWLモード)に設定する必要がありますが、その場合NITRO ROMとの互換性がなくなり、プログラムの修正が必要となる場合があります。CODEC-TWLモードの詳細は、関数リファレンスの「サウンド(SND)」→「概要」→「TWL サウンド拡張機能 概要」をご参照ください。

### 4.12.4 CardRegionの設定(カードアプリ)

ソフトのリージョンを指定します。ソフトのリージョンと本体のリージョンが一致しない場合、 そのソフトは ランチャー上で認識されませんのでご注意ください。またサンプルデモに設定されている ALL はデモ専用のものであり、この設定でマスター ROM を提出することはできません。

# 5 HYBRID ROMへ移行する際の注意点

TWL-SDK において、HYBRID ROM に移行する際の注意点を挙げます。

## 5.1 ライブラリが利用するメモリ使用量の増加

3.1 と関連していますが、TWL-SDK のライブラリが消費するメモリは、機能追加と不具合修正のため、NITRO-SDKと比較すると増加しています。HYBRID ROM ではそれに加えて、TWL モードで動作するための処理が追加されているため、さらに多くのメモリを使用します。そのため、NITRO-SDK から HYBRID ROM アプリケーションに移植した場合に、アプリケーションが使えるメモリ領域は減少します。一例として TWL-SDK 5.2 を使用し、過去の DS プロジェクトを移植した場合には、ライブラリが消費するメモリは約 15Kbyte 増加します。

NITRO-SDK の全ての関数をリンクしていた場合約 19Kbyte 増えます。Thumb コードを使用した場合は、約13Kbyte 増加します。但しライブラリ内にある inline 関数が増えていた場合、複数回コールされた分メモリ使用量が増加しますので、実際消費するメモリ量はこれより大きくなることがあります。

また 4.4 より HYBRID ROM ではメモリ先頭 16Kbyte がシステムのために予約されています。そのため、アプリケーションが使えるメモリ領域は上記のライブラリの増加分とシステム用の先頭 16Kbyte の合計分減少することになります。

以下にメモリが足りない場合の対策方法をご説明します。

### 5.1.1 カートリッジライブラリ(CTRDG)の無効化

OS_Init()からは、カートリッジライブラリの初期化であるCTRDG_Init()を常に呼ぶようになっており、その分のメモリが消費されています。常にカートリッジライブラリを使用しない場合は、Weak化されたCTRDG_Init()関数を、CTRDG_DummyInit()を内部で呼ぶ関数で定義しなおすことでその分のメモリを約 3Kbyte減らすことができます。詳細は関数リファレンスの「カートリッジモジュール(CTRDG)」→「初期化」→「CTRDG_DummyInit」をご参照ください。

### 5.1.2 メモリの先頭領域の使用

システムに予約された先頭 16Kbyte の領域は TWL モードのみで使用しますので、NITRO モードではアプリケーションが使用することができます。具体的には 0x02000000 ～ 0x02004000 の領域を

if (OS_IsRunOnTwl() == FALSE) においてのみご使用ください。

また、この領域をヒープとして作成する場合はOS_CreateExtraHeap()を、既存のヒープ領域に追加する場合はOS_AddExtraAreaToHeap()を使用してください。詳細は関数リファレンスの「オペレーティングシステム(OS)」→「概要」→「メモリ割り当て（overview)」をご参照ください。

# 6 LIMITED ROMへ移行する際の注意点

TWL-SDK において、LIMITED ROM に移行する際の注意点を挙げます。

## 6.1 クローンブートの非対応

LIMITED ROM ではクローンブートは非対応です。クローンブートを使用する場合は HYBRID ROM を選択して下さい。

# 7 NANDアプリへ移行する際の注意点

TWL-SDK において、NAND アプリに移行する際の注意点を挙げます。NAND アプリは TWL でしか動作しないため、基本的には LIMITED ROM として作成します。HYBRID ROM でも作成可能ですが、これはクローンブートのサポートを目的としています。

NANDアプリの開発について、詳しくは「TwlSDK NAND アプリ開発マニュアル」( NandAppManual.pdf )をご参照ください。

## 7.1 NAライブラリのリンク

NAND アプリにするには NA ライブラリをリンクする必要があります。lib/ARM9-TS/*/libna.TWL*.a をリンクするよう

にしてください。

## 7.2 OS_ResetSystem()をOS_RebootSystem()に置き換え

NAND アプリでソフトウェアリセットを使用する場合は、OS_ResetSystem()ではなく、OS_RebootSystem()を使用します。

## 7.3 OS_GetConsoleType()の返り値

NAND アプリでは、OS_GetConsoleType()の返り値と OS_CONSOLE_DEV_MASK の論理積が OS_CONSOLE_DEV_NAND を返します。

## 7.4 電子取説の用意

NAND アプリには紙媒体の取扱説明書が付属しないため、電子取説を用意し、タイトル画面付近から選択可能にしてください。詳細はニンテンドーDS/TWL プログラミングガイドラインまたは TWL-ManualTools をご参照ください。

## 7.5 セーブデータについて

カードアプリの場合はゲームカードのバックアップデバイスにセーブデータを保存していましたが、NAND アプリでは本体 NAND メモリにセーブデータを保存します。以下に NAND メモリに移行する際の注意点を挙げます。

### 7.5.1 CARD関数の削除

NAND アプリではゲームカードのバックアップデバイスを使いませんので、CARD 関数は削除してください。

### 7.5.2 FS関数の使用

NANDアプリではセーブデータにはFS関数を使用し、「dataPub」または「dataPrv」のアーカイブにアクセスします。詳細は「TwlSDK NAND アプリ開発マニュアル」( NandAppManual.pdf ) の 「7.1 セーブデータ領域の使用」を参照してください。

### 7.5.3 Privateセーブデータのユーザーによる初期化

NAND アプリでは SD カードへのコピーができますが、private セーブデータは含まれないため、SD カードにコピーされた NAND アプリを書き戻すときには private セーブデータはクリアされます。このため、アプリケーションは常に private セーブデータのみがクリアされる状況を想定する必要があります。

### 7.5.4 セーブデータに指定するサイズについて

セーブデータのサイズはRSFファイルによって指定します(7.6.2 参照)。セーブデータ領域内にはファイルシステムの管理領域などが含まれるため、アプリケーションで実際に使用できる領域は指定する値より小さな値になります。セーブデータ領域サイズごとの実際に利用可能なサイズの値については関数リファレンスの「ツール」→「ROMイメージ関連」→「セーブデータサイズ一覧」を参照してください。

### 7.5.5 Flashメモリに比べてアクセス速度の向上

カードアプリに使用していた Flash メモリと比較すると、NAND メモリの読み書きの速度は高速になります。詳細は、TWL プログラミングマニュアル 23.NAND フラッシュメモリをご参照ください。

### 7.5.6 セーブ中の電源断について

セーブデータを書き込んでいる最中に電源ボタンが押された場合、FS 関数は書き込みを中断し FS_RESULT_CANCELED を返します。電源ボタンが押された時点までに書かれたデータは正常にセーブデータに反映されます。ファイルテーブルが破壊される心配はありません。

### 7.5.7 NANDの書き込み寿命の考慮について

ゲームカードのバックアップメモリにおいては、書き換え寿命を考慮する必要がありましたが、本体 NAND メモリにおいては、基本的に考慮する必要がありません。但し過度な消去や書き込み(毎秒セーブする、1 歩歩くごとにセーブする、等)は避けるようにしてください。自動でセーブを行う場合は、保存データのサイズが 3 分あたり 128KB 以下となるような頻度で行ってください。もしくは通常はメインメモリ上でデータを操作し、電源が切断される場合に、PM_AppendPreExitCallback() 関数等を利用して、シャットダウン時にセーブするようにしてください。但しこの場合でも終了処理は 3 秒以内に終える必要がありますので、その時間内に終了できる範囲のセーブデータを保存してください。詳細はニンテンドーDS/TWL プログラミングガイドラインをご参照ください。

### 7.5.8 対処すべき新規FSエラー

ROM アーカイブへのアクセスと異なり NAND アーカイブへ FS 関数でアクセスする際に以下のエラーに対処する必要があります。

#### 7.5.8.1 FS_RESULT_MEDIA_FATAL

致命的なデバイス異常を検知した場合などに発生します。このアーカイブに対する処理の継続は困難なため、本体の修理を促す必要があります。

#### 7.5.8.2 FS_RESULT_BAD_FORMAT

ファイルシステムの破損を検知した場合などに発生します。このアーカイブに対する処理の継続は困難なため、NA_FormatTitleArchive()でアーカイブを再フォーマットする必要があります。

## 7.6 RSFファイルの修正

NANDアプリに関しては$TwlSDK/include/twl/specfiles/ROM-TS_nand.rsfを元にTitleName, MakerCode, RemasterVersion, RomHeaderTemplate, BannerFileを設定してください。ここでは、特に注意すべき点についてのみ挙げます。詳細は、関数リファレンスの「ツール」→「ROMイメージ関連」→「makerom.TWL」をご参照ください。

### 7.6.1 InitialCodeの設定

RSF ファイルの AppendProperty セクションの InitalCode にアプリごとに割り振られた ASCII4 文字のイニシャルコードを設定してください。

### 7.6.2 Privateセーブデータ、Publicセーブデータのサイズの指定

PrivateSaveDataSize, PublicSaveDataSize に、それぞれ使用するセーブデータのサイズを指定してください。

### 7.6.3 RomSizeとRomFootPaddingの修正

NAND アプリでは RomSize の指定は無視され、最適な値が自動的に選択されます。また RomFootPadding は必ず FALSE を指定してください。

## 7.7 FS関数を使用してROMアーカイブにアクセスする際の注意点

NAND アプリにおいては従来のカードアプリと比較して、物理的なメディアやフォーマットが異なるため、FS 関数を使用して ROM アーカイブにアクセスする際にいくつか注意する点があります。

### 7.7.1 NANDの読み出し時間にランダムな遅延時間の追加

本体 NAND メモリは、個体差や経年劣化による性能低下により、アクセス速度が不定となります。「不定なアクセス速度」を擬似的にシミュレーションする目的で Debug および Release ビルド時には読み出しアクセス時間にランダムな遅延時間が追加されます。パフォーマンス計測等に支障がある場合は、FinalRom ビルドをご使用ください。

### 7.7.2 NANDアプリのROMアーカイブは高速アクセス可能

NAND アプリの ROM アーカイブはカードアプリでワンタイム PROM を使用した場合と比較して、読み出し速度は約 2 倍近く高速になります。7.7.1 で挙げた問題がありますので、遅延時間が長くなることがありますが速度がカードアプリ程遅くなることはありません。

### 7.7.3 FS_Init()でのDMAチャンネル設定不可

カードアプリにおいては、FS_Init()で DMA チャンネルを設定できますが、NAND アプリでは設定することはできません。これは NAND に対して DMA 転送が使用できないためです。

### 7.7.4 RTC, SNDEX, 無線の処理の遅延

FS 関数(NA 関数を含む)を使用して NAND にアクセスしている場合、FS 関数の処理が完了するまで RTC, SNDEX, 無線の処理が遅延します。別スレッドで RTC_GetTime()などの同期関数を動かした場合は、FS 関数の処理が終わるまでブロックされます。この問題を回避するためには、RTC_GetTimeAsync()などの非同期関数を使用してください。またこの問題は将来の SDK で修正を予定しています。

### 7.7.5 MP通信によるファイルアクセスの速度低下

カードアプリでは ROM アーカイブにアクセスした場合、ARM9 によって処理するため、ARM7 で行っている MP 通信の影響を受けることはありませんが、NAND アプリでは、NAND アーカイブへのアクセスを MP 通信と同じ ARM7 を使用してアクセスしているため、MP 通信の影響を受けやすくなります。MP 通信を行っている場合、ファイルへのアクセス速度が MP 通信を行っていない場合と比較して遅くなります。

記載されている会社名、製品名等は、各社の登録商標または商標です。