

アーカイブフォーマット解説書

アーカイブのフォーマットについての説明

2008-05-30

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	5
2	アーカイブの基本構造	5
3	アーカイブヘッダ	6
3.1	アーカイブヘッダの構造	6
3.1.1	signature	6
3.1.2	byteOrder	6
3.1.3	version	6
3.1.4	fileSize	6
3.1.5	headerSize	6
3.1.6	dataBlocks	7
4	ファイルアロケーションテーブルブロック	7
4.1	ファイルアロケーションテーブルブロックの構造	7
4.1.1	kind	7
4.1.2	size	7
4.1.3	numFiles	7
4.1.4	allocationTable	7
5	ファイル名テーブルブロック	8
5.1	ファイル名テーブルブロックの構造	8
5.1.1	kind	8
5.1.2	size	8
5.1.3	ディレクトリテーブル	9
5.1.4	エントリ名テーブル	9
5.1.5	アライメント用のパディング	10
5.2	ファイル名テーブルの例	10
6	ファイルイメージブロック	11
6.1	kind	11
6.1.1	size	11
6.1.2	fileImage	11

表

表 3-1	アーカイブヘッダの構造	6
表 4-1	ファイルアロケーションテーブルブロック	7
表 4-2	ファイルアロケーションエントリ	8
表 5-1	ファイル名テーブルブロック	8
表 5-2	ディレクトリテーブルエントリ	9
表 5-3	ファイルエントリ構造	9
表 5-4	ディレクトリエントリ構造	9



図 2-1	アーカイブ構造	5
図 5-1	ファイル名テーブル	10

改訂履歴

改訂日	改訂内容
2008-05-30	NITRO-System の名称変更による修正 (NITRO-System から TWL-System に変更)。
2008-04-08	<ul style="list-style-type: none"> ・改訂履歴の書式を変更。 ・TWL-SDK に対応。
2007-11-26	ファイルのアライメント指定が行われたときの説明を追記。
2005-01-05	NITRO という表記をニンテンドーDS に統一。
2004-06-10	初版。

1 はじめに

アーカイブは、いくつかのファイルが1つに結合されたファイルです。アーカイブにはファイルだけでなく、階層ディレクトリ情報も含めることが可能となっており、ファイルID(インデックス値)またはパス名を指定することにより、アーカイブ内の個々のファイルにアクセスする事ができるようになっています。

2 アーカイブの基本構造

アーカイブはTWL-Systemのバイナリファイル規約に従っており、図 2-1 のような構造になっています。

アーカイブの先頭にはアーカイブヘッダが置かれます。その後に、ファイルアロケーションテーブル、ファイル名テーブル、ファイルイメージの順にデータブロックが続きます。

ファイルアロケーションテーブル、及びファイル名テーブルの構造は、TWL-SDK の ROM ファイルシステムの構造と基本的に同じ構造となっています。

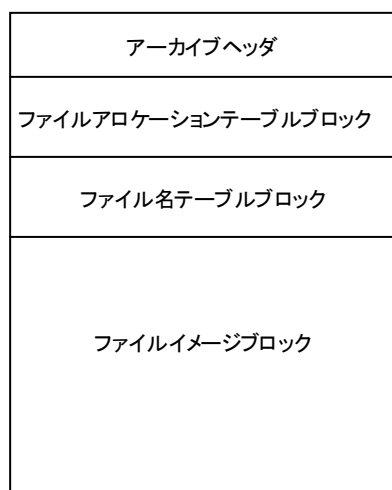


図 2-1 アーカイブ構造

3 アーカイブヘッダ

アーカイブヘッダは、必ずアーカイブファイルの先頭にあります。アーカイブヘッダには、アーカイブファイル全体に対する情報が入っています。

3.1 アーカイブヘッダの構造

アーカイブヘッダは、表 3-1 に示すような構造となっています。

表 3-1 アーカイブヘッダの構造

タイプ	パラメータ名	意味	サイズ
char[4]	signature	ファイル・シグネチャ。('N', 'A', 'R', 'C')	4 バイト
u16	byteOrder	バイトオーダーマーク。(0xffeff)	2 バイト
u16	version	アーカイブフォーマットのバージョン番号。(0x0100)	2 バイト
u32	fileSize	アーカイブファイルの大きさ。	4 バイト
u16	headerSize	アーカイブヘッダの大きさ。(16)	2 バイト
u16	dataBlocks	データブロックの数。(3)	2 バイト

3.1.1 signature

signature には、バイナリファイルの種類を判定する為のファイル・シグネチャが格納されます。ファイル・シグネチャには、'N', 'A', 'R', 'C' の4文字が格納されます。シグネチャの4文字は、エンディアンに関係なく、必ずこの順番に格納されます。

3.1.2 byteOrder

byteOrder には、エンディアン判定用のバイトオーダーマーク(Zero Width No Break Space)0xffeff が格納されます。TWL およびニンテンドーDS はリトルエンディアンで動作しますので、0xff,0xfe の順で格納されます。TWL-System のアーカイバである nnsarc.exe は、必ずリトルエンディアンでアーカイブを作成します。

3.1.3 version

version には、アーカイブフォーマットのバージョン番号が格納されます。上位バイトと下位バイトは、それぞれメジャーバージョン(整数値)と、マイナーバージョン(小数値)となっています。現在のバージョンは、1.0 です。0x0100 という値が格納されます。

3.1.4 fileSize

fileSize には、アーカイブ全体の大きさが格納されます。この大きさには、アーカイブヘッダの大きさも含まれます。

3.1.5 headerSize

headerSize には、アーカイブヘッダの大きさである 16 が格納されます。将来のバージョンアップにより、アーカイブヘッダの大きさが変更される可能性がありますので、ヘッダの大きさが 16 であると仮定しないで下さい。

3.1.6 dataBlocks

dataBlocks には、アーカイブに含まれるデータブロックの数が格納されます。現在のバージョンでは、必ず3つのデータブロックが存在しますので、3が格納されます。なお将来、新しいデータブロックが追加される可能性があります。

4 ファイルアロケーションテーブルブロック

ファイルアロケーションテーブルブロックには、アーカイブ内の各ファイルの中身が、アーカイブのどの位置に格納されているかを示す情報が収められています。

4.1 ファイルアロケーションテーブルブロックの構造

ファイルアロケーションテーブルブロックは、表 4-1 のような構造となっています。ファイルアロケーションテーブル内の各エントリには、ファイルIDと呼ばれる番号が割り振られています。ファイルIDは格納順に 0x0000 からカウントアップされ、最大 0xffffまで割り振られます。アロケーションテーブルの配列の大きさはファイル数と一致し、この配列の添え字がファイルIDと一致します。

表 4-1 ファイルアロケーションテーブルブロック

タイプ	パラメータ名	意味	サイズ
u32	kind	データブロックの種類。('FATB')	4 バイト
u32	size	データブロックの大きさ。	4 バイト
u16	numFiles	ファイルの個数。	2 バイト
u16	reserved	予約。	2 バイト
	allocationTable	ファイルアロケーションテーブル。 (1エントリにつき8バイト)	8×n バイト

4.1.1 kind

kind には、データブロックの種類を表す4バイトのコードが格納されます。この4バイトには、コード 'FATB' が格納されます。なお、アーカイブはリトルエンディアンですので、アルファベットは逆順に格納されています。

4.1.2 size

size には、ファイルアロケーションテーブルが格納されているデータブロックの大きさが格納されます。

4.1.3 numFiles

numFiles には、ファイルアロケーションテーブルに含まれているファイルの個数が格納されます。この値がアーカイブに格納されているファイルの数となります。

4.1.4 allocationTable

ファイルアロケーションテーブルは、ファイルアロケーションエントリの配列となっています。1つのファイルエントリにつき、8バイトの大きさがあります。各エントリにはファイルID と呼ばれる番号が割り振られています。ファイルIDは、格納順に

0x0000 からカウントアップされます。最大値は 0xffff となります。配列の大きさはファイルの数と一致し、配列の添え字がファイルIDと一致します。

表 4-2 ファイルアロケーションエントリ

タイプ	パラメータ名	意味	サイズ
u32	fileTop	ファイルの先頭オフセット。	4 バイト
u32	fileBottom	ファイルの最終オフセット+1。	4 バイト

ファイルへの先頭、最終位置を示すオフセットは、ファイルイメージブロックの `fileImage` パラメータの位置(ファイルイメージブロックの先頭から8バイト目)を0とする値が格納されます。ファイルのサイズを求める場合は、以下の計算式で求められます。

```
u32 fileSize = fileBottom - fileTop;
```

5 ファイル名テーブルブロック

ファイル名テーブルブロックには、パス名からファイルIDを取得するための情報が格納されています。ファイル名テーブルブロックは、ディレクトリテーブルとエントリ名テーブルから構成され、階層ディレクトリをサポートしています。

5.1 ファイル名テーブルブロックの構造

ファイル名テーブルブロックは、表 5-1 に示すような構造となっています。

表 5-1 ファイル名テーブルブロック

タイプ	パラメータ名	意味	サイズ
u32	kind	データブロックの種類。('FNTB')	4 バイト
u32	size	データブロックの大きさ。	4 バイト
	directoryTable	ディレクトリテーブル。	n バイト
	entryNameTable	エントリ名テーブル。	m バイト
	padding	アライメント用のパディング	

5.1.1 kind

kind には、データブロックの種類を表す4バイトのコードが格納されます。この4バイトには、コード 'FNTB' が格納されます。なお、アーカイブはリトルエンディアンですので、アルファベットは逆順に格納されています。

5.1.2 size

size には、ファイル名テーブルが格納されているデータブロックの大きさが格納されます。

5.1.3 ディレクトリテーブル

ディレクトリテーブルは、表 5-2 に示すデータ構造の配列となっています。各エントリにはディレクトリIDと呼ばれる番号が割り振られています。ディレクトリIDは、格納順にカウントアップされます。ファイルIDと区別するため、ディレクトリIDは 0xf000 から 0xffffまでの数値を使用します。この仕様から、ファイルの最大格納可能数は 61440 個、ディレクトリの最大格納数は 4096 個となります。

表 5-2 ディレクトリテーブルエントリ

タイプ	パラメータ名	意味	サイズ
u32	dirEntryStart	エントリ名の検索位置。	4 バイト
u16	dirEntryFileID	先頭エントリのファイルID。	2 バイト
u16	dirParentID	親ディレクトリのID。(またはディレクトリエントリ数。)	2 バイト

dirEntryStart は、そのディレクトリに含まれる最初のエントリ(ファイルかディレクトリ)を示し、ディレクトリテーブルの先頭(ファイル名テーブルブロックの先頭から8バイト目)を0としたオフセット値が格納されます。

ディレクトリエントリ配列の要素数はディレクトリの数と一致し、配列の添え字がディレクトリIDから 0xf000 を減じた値と一致します。ディレクトリIDが 0xf000 であるディレクトリは、ルートディレクトリとなります。ルートディレクトリの場合のみ、親ディレクトリIDのメンバにディレクトリエントリ数が格納されます。

5.1.4 エントリ名テーブル

エントリ名テーブルは、2種類の可変長のデータの集合体です。そのエントリがファイルであるかディレクトリであるかによってデータ構造を使い分けます。

表 5-3 ファイルエントリ構造

タイプ	パラメータ名	意味	サイズ
u8	entryNameLength	ファイル名の長さ。(上位1ビットはエントリタイプを示す。)	1 バイト
char	entryName[n]	ファイル名。(n = entryNameLength)	n バイト

表 5-4 ディレクトリエントリ構造

タイプ	パラメータ名	意味	サイズ
u8	entryNameLength	ファイル名の長さ。(上位1ビットはエントリタイプを示す。)	1 バイト
char	entryName[n]	ファイル名。(n = entryNameLength)	n バイト
u16	directoryID	ディレクトリID。	2 バイト

同一ディレクトリ内に含まれるエントリは、連続した領域に配置されており、連続したファイルIDが割り振られます。ディレクトリ内の最終エントリの次には、エントリ名の長さが0であるファイルエントリ('¥0') が置かれます。

エントリ名の長さは、entryNameLength の下位7ビットで表される為、エントリ名は 1byte 文字で換算して最大 127 文字までとなります。

entryNameLength の最上位ビットは、エントリのタイプを表します。最上位ビットが0の場合はファイルエントリであり、最上位ビットが1の場合は、ディレクトリエントリとなります。

5.1.5 アライメント用のパディング

ファイル名テーブルブロックの最後には、次に続くファイルイメージブロック内の先頭に格納されているファイルを設定された境界にアライメントするためのパディングが付加されます。パディングは 0xff の値で埋められます。

5.2 ファイル名テーブルの例

図 5-1 は、以下の 3 つのファイルが格納された場合のファイル名テーブルの状態を示しています。

```
/sprite.bin
/screen.bin
/model/player.nmd
```

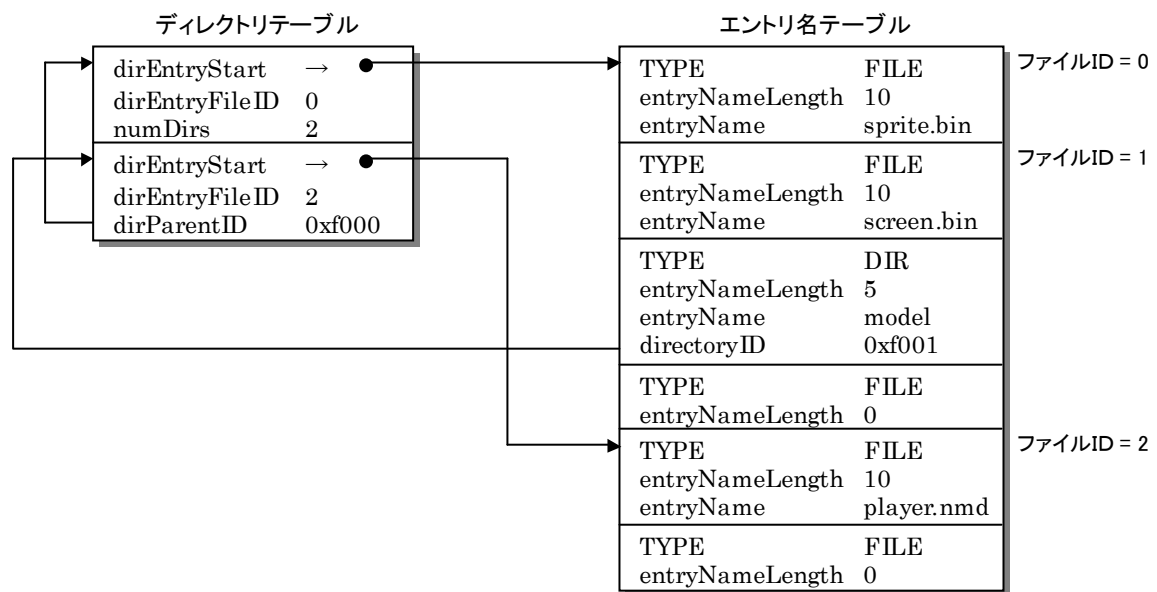


図 5-1 ファイル名テーブル

6 ファイルイメージブロック

ファイルイメージブロックには、アーカイブされているファイルのイメージが格納されています。各ファイルイメージの先頭位置と最終位置は、ファイルアロケーションテーブルの各エントリにより示されています。

ファイルイメージブロック

タイプ	パラメータ名	意味	サイズ
u32	kind	データブロックの種類。('FIMG')	4 バイト
u32	size	データブロックの大きさ。	4 バイト
	fileImage	ファイルイメージ。	n バイト

6.1 kind

kind には、データブロックの種類を表す4バイトのコードが格納されます。この4バイトには、コード'FIMG'が格納されます。なお、アーカイブはリトルエンディアンですので、アルファベットは逆順に格納されています。

6.1.1 size

size には、ファイルイメージが格納されているデータブロックの大きさが格納されます。

6.1.2 fileImage

fileImage には、アーカイブファイルに含まれる全ファイルのイメージがパックされた形で格納されます。格納されるファイルイメージは、4バイト、8 バイト、16 バイト、32 バイトのいずれかの境界で整列(アライメント)されます。各ファイルの先頭をアライメントの境界に合わせるために、各ファイルの間にはパディングが挿入される場合があります。パディングは 0xff の値で埋められます。

© 2004-2008 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。