tar の構造

tar アーカイブファイルは 1 つ 512 バイトのブロックが複数個合わさって構成される。 アーカイブされたそれぞれのファイルは 1 つのヘッダーブロックと、それに続く 0 個以上のファイルの内容を含むブロックからなる。 アーカイブファイルの終わりは end-of-archive を表すためにバイナリーゼロで埋められたブロックが 2 つ付加される。

header 1 (512 バイト)
content 1 (2048 バイト)
header 2 (512 バイト)
content 2 (1024 バイト)
header 3 (512 バイト)
content 3 (1536 バイト)
バイナリーゼロ (1024 バイト)

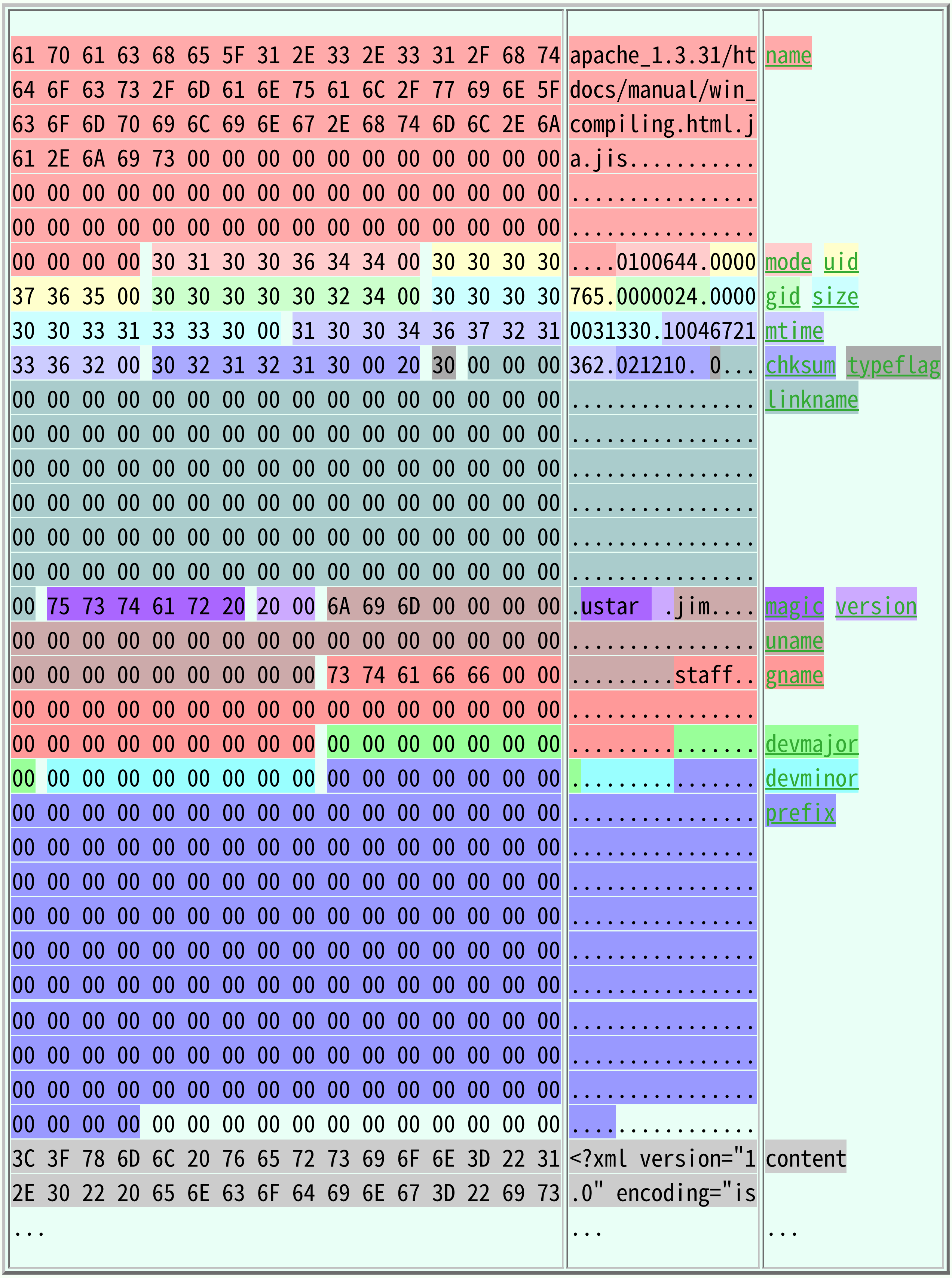
ヘッダーブロックは以下に示される構造からなる。

tar ヘッダーブロック

フィールド名	オフセット(バイト)	長さ(バイト)
name	0	100
mode	100	8
uid	108	8
gid	116	8
size	124	12
mtime	136	12
chksum	148	8
typeflag	156	1
linkname	157	100
magic	257	6
version	263	2
uname	265	32
gname	297	32
devmajor	329	8
devminor	337	8
prefix	345	155

長さとオフセットはすべて 10 進数で表している。

tar ファイルの中身



```
61 70 61 63 68 65 5F 31 2E 33 2E 33 31 2F 68 74  apache_1.3.31/ht  name
64 6F 63 73 2F 6D 61 6E 75 61 6C 2F 77 69 6E 5F  docs/manual/win_
63 6F 6D 70 69 6C 69 6E 67 2E 68 74 6D 6C 2E 6A  compiling.html.j
61 2E 6A 69 73 00 00 00 00 00 00 00 00 00 00 00  a.jis...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 30 31 30 30 36 34 34 00 30 30 30 30  ....0100644.0000  mode uid
37 36 35 00 30 30 30 30 30 32 34 00 30 30 30 30  765.0000024.0000  gid size
30 30 33 31 33 33 30 00 31 30 30 34 36 37 32 31  0031330.10046721  mtime
33 36 32 00 30 32 31 32 31 30 00 20 30 00 00 00  362.021210. 0...  chksum typeflag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  linkname
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 75 73 74 61 72 20 20 00 6A 69 6D 00 00 00 00  .ustar  .jim....  magic version
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  uname
00 00 00 00 00 00 00 00 00 73 74 61 66 66 00 00  .........staff..  gname
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  devmajor
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  devminor
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................  prefix
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
3C 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31  <?xml version="1  content
2E 30 22 20 65 6E 63 6F 64 69 6E 67 3D 22 69 73  .0" encoding="is
...                                               ...               ...
```

ヘッダーブロック内で用いられるシンボル定数はヘッダーファイル <tar.h> で以下のように定義されている。

```
#define TMAGIC    "ustar" /* ustar and a null */
#define TMAGLEN   6
#define TVERSION  "00"    /* 00 and no null */
#define TVERSLEN  2

/* Values used in typeflag field */
#define REGTYPE  '0'     /* Regular file  */
#define AREGTYPE '\0'    /* Regular file  */
```

```
#define LNKTYPE  '1'     /* Link          */
#define SYMTYPE  '2'     /* Reserved      */
#define CHRTYPE  '3'     /* Character special */
#define BLKTYPE  '4'     /* Block special */
#define DIRTYPE  '5'     /* Directory     */
#define FIFOTYPE '6'     /* FIFO special  */
#define CONTTYPE '7'     /* Reserved      */

/* Bits used in the mode field - values in octal */
#define TSUID    04000   /* Set UID on execution */
#define TSGID    02000   /* Set GID on execution */
#define TSVTX    01000   /* File permissions */
#define TUREAD   00400   /* Read by owner */
#define TUWRITE  00200   /* Write by owner */
#define TUEXEC   00100   /* Execute/Search by owner */
#define TGREAD   00040   /* Read by group */
#define TGWRITE  00020   /* Write by group */
#define TGEXEC   00010   /* Execute/Search by group */
#define TOREAD   00004   /* Read by other */
#define TOWRITE  00002   /* Write by other */
#define TOEXEC   00001   /* Execute/Search by other */
```

---

name

`name` フィールドは最後の文字を含むすべての文字が非 NULL 文字のとき (つまり 100 文字全部を使い切っているとき) 以外は NULL 終端文字列である。 `name` フィールドはファイルのパスを提供する。 アーカイブに含まれるファイルの構造的な位置関係はパス名から決定できる。 パス名が与えられたスペースに収まらない場合、アーカイブ作成ユーティリティーはユーザーにエラーを通知しファイルのいかなる部分(ヘッダーとデータ)もメディアに格納しようとすべきでない。

mode

`mode` フィールドは 9 ビットのファイルパーミッションと 3 ビットの setUID、setGID、TSVTX モードを与える。 これらのモードビットを設定するときに適切な権限が必要な場合や、アーカイブからファイルを取り出すユーザーが適切な権限を持たない場合、ユーザーが権限を持たないモードビットは無視される。

uid, gid

`uid` と `gid` フィールドはそれぞれファイルのユーザー ID とグループ ID を表している。

size

`size` フィールドはファイルサイズのバイト数である。 `typeflag` フィールドが LINKTYPE か SYMTYPE に設定されている場合 `size` フィールドは 0 と設定される。 `typeflag` フィールドが DIRTYPE に設定されている場合 `size` フィールドはそのレコードタイプの定義で述べられているものと解釈される。 LINKTYPE、SYMTYPE、DIRTYPE にはデータが含まれない。 `typeflag` フィールドが CHRTYPE、BLKTYPE、FIFOTYPE に設定された場合、`size` フィールドの意味はここでは定義しない。アーカイブファイルにはデータブロックは格納されない。 `typeflag` フィールドが FIFOTYPE の場合 `size` フィール

ドはファイル読み込み時には無視すべきである。 `typeflag` が他の値に設定された場合、ヘッダーに続くブロック数は(`size`+511)/512であり、それ以降の部分は無視する。

mtime

`mtime` フィールドはアーカイブされた時点でのファイルの最終更新時刻である。 これはstat()関数で得られる最終更新時刻の8進数の値でありISO/IEC 646の文字で表す。

chksum

`chksum` フィールドはヘッダーブロックの全バイトの和を8進数で表した値でありISO/IEC 646 IRVの文字で表される。 ヘッダー内の各バイトは符号なし(unsigned)の値として扱われる。 チェックサムの値は0に初期化された符号なし整数(unsigned integer)に加算して求められる。 この整数の精度は少なくとも17ビットであるとする。 チェックサムを計算しているときには `chksum` フィールドはすべて空として扱われる。

typeflag

`typeflag` フィールドは1文字からなりアーカイブの種類を決定する。 特定の実装がタイプを認識できなかったり、ユーザーがタイプを生成するのに適切な権限を持たない場合、ファイルは `size` フィールドが意味を持つように定義されているとき(この場合にはメディア上にデータブロックが生成される)には通常のファイルと同じように伸長されるべきである(`size` の定義を見ること)。 通常ファイルへの変換が行われた場合、アーカイブ伸長ユーティリティーは変換が行われたというエラーを通知すべきである。 `typeflag` フィールドはすべてISO/IEC 646 IRVの文字で記述される:

'0'

通常のファイルを表す。 後方互換性のためアーカイブからファイルを抽出するときに `typeflag` のバイナリーゼロ'\0'は通常ファイルとして認識されるべきである。 ここで記述されているフォーマットを持つアーカイブファイルは `typeflag` 値がISO/IEC 646 IRV '0'を持つ通常ファイルを作成するであろう。

'1'

以前にアーカイブされた(すべてのタイプの)他のファイルへのファイルリンクを表す。 そのようなファイルは同じデバイスやファイルシリアルナンバーを持つファイルで識別される。 リンクしている名前は、100バイト未満の場合null終端の付いた `lihkname` フィールドで決定される。

'2'

デバイスやファイルシリアルナンバーの異なった(すべてのタイプの)他のファイルへのリンクを表す。 これはデバイスやファイルシリアルナンバーの異なるファイルへのリンクをサポートするシステムのために提供される。 この拡張が存在しない場合にはタイプ'1'のファイルとして扱われる。

'3', '4'

それぞれキャラクター型スペシャルファイルとブロック型スペシャルファイルを表す。 この場合 `devmajor` と `devminor` フィールドがデバイスを決定する情報を持つであろう。そのフォーマットについてはここでは決定しない。 実装はデバイス指定をそれら自身のローカル指定にマップしてもよいし、エントリー自体を無視してもよい。

'5'

ディレクトリーやサブディレクトリーを表す。 ディスク割り当てをディレクトリーベースで行っているシステムでは `size` フィールドはディレクトリーが持つことができる最大バイトにすべきである (その値は一番近いディスクブロックアロケーションユニットに丸められるであろう)。 0 の `size` フィールドはそのような制限がないことを表す。 このような方法での制限をサポートしないシステムでは `size` フィールドを無視すべきである。

'6'

FIFO スペシャルファイルを表す。 FIFO ファイルのアーカイビングはこのファイルの実体だけをアーカイブしその内容はアーカイブしないことに注意。

'7'

実装がファイルをあるハイパフォーマンスな属性と関連付けるためにリザーブしておく。 そのような拡張を持たない実装はこのファイルを通常のファイル (タイプ '0') として扱うべきである。

'A' - 'Z'

A から Z の文字はカスタム実装用にリザーブしておく。 他のすべての文字は将来のバージョンのためにリザーブしておく。

linkname

`linkname` フィールドは最後の文字を含むすべての文字が非 NULL 文字のとき (つまり 100 文字全部を使い切っているとき) 以外は NULL 終端文字列である。 `linkname` フィールドはパス名を生成するときに `prefix` を用いない。そのため `linkname` は 100 文字に制限される。 リンク名が与えられたスペースに収まらない場合、アーカイブ作成ユーティリティーはユーザーにエラーを通知し、メディアにリンク名を格納しようとすべきではない。

magic

`magic` フィールドは NULL 終端文字列である。 `magic` フィールドはアーカイブがここに記述されているフォーマットで出力されたかを表す。 このフィールドが TMAGIC を含む場合、`uname` と `gname` フィールドにはそれぞれファイルの所有者とグループが ISO/IEC 646 IRV 形式で (もし必要ならば適切なサイズに切り詰めて) 含まれるべきである。 ファイルが適切な権限付きで、保護機能を保持するユーティリティーで復元された場合、パスワードファイルとグループファイルがこれらの名前でスキャンされるべきである。 もし該当する名前が見つかったならば、これらのファイルに含まれるユーザー ID とグループ ID は `uid` と `gid` フィールドに含まれる値よりも優先して使われるべきである。

version

`version` フィールドは "00" (ゼロゼロ) の 2 バイトからなる。

uname, gname

`uname` と `gname` フィールドは NULL 終端文字列である。 詳しくは `magic` フィールドを参照のこと。

devmajor, devminor

デバイスを定義する情報を持つ。 詳しくは `typeflag` フィールドを参照のこと。

prefix

`prefix` フィールドは最後の文字を含むすべての文字が非 NULL 文字のとき (つまり 155 文字全部を使い切っているとき) 以外は NULL 終端文字列である。 `name` と `prefix` フィールドはファイルのパスを提供する。 ファイルの構造的な位置関係はパス名から決定できる (パス名はパスのプリフィックスと、スラッシュ文字＋ファイル名のサフィックスからなる)。 `prefix` が空文字列ではない場合 (1 文字目が NULL ではない場合)、(最初の NULL 文字に至るまでの) `prefix` とスラッシュ文字と `name` を連結して新しいパス名を作成するか、あるいは `name` が単独で使われる。 どちらの場合も `name` は最初の NULL 文字が終端となる。 `prefix` が空文字列の場合は単に無視されるだけである。 この方法を用いると 256 文字までのパス名をサポートできる。 パス名が与えられたスペースに収まらない場合、tar 作成ユーティリティーはユーザーにエラーを通知しファイルのいかなる部分(ヘッダーやデータ)もメディアに格納しようとすべきでない。

すべての文字は ISO/IEC 646 の符号化文字集合により表わされる。 実装間での移植性を高めるために、名前には最上位ビットが 0 の 8 ビット文字である ポータブルファイル名文字集合を用いるべきである。 もし実装がファイル名やユーザー名、グループ名にポータブルファイル名文字集合外の文字の使用を許可する場合、変換用にそれらの文字に実装で定義された 1 つ以上のエンコーディングが提供されるべきである。 しかし、アーカイブ伸長用ユーティリティーはこの仕様書に書いてある方法でアクセスできないファイル名はローカルのシステム上に作成すべきではない。 もし不正なファイル名を作成するファイル名がメディア上に見つかった場合には、実装はファイルからのデータをその名前で保存するか否かを決めることができる。 伸長用ユーティリティーはエラーを出力しこれらのファイルを無視するという選択をするかもしれない。

ヘッダーブロックの各フィールドは連続しておりパディングは用いない。アーカイブメディア上の各文字は連続的に格納されている。

ISO/IEC 646

-	0x2	0x3	0x4	0x5	0x6	0x7
0	SP	0	*1	P	*1	p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3	# £	3	C	S	c	s
4	¤ $	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	*1	k	*1
C	,	<	L	*1	l	*1

D	-	=	M	*1	m	*1
E	.	>	N	*1	n	*1
F	/	?	O	_	o	DEL

0x23 は # か £ かを各文字集合で選択する。
0x24 は ¤ か $ かを各文字集合で選択する。
*1 の部分は各文字集合で定義する。

ISO/IEC 646 IRV

-	0x2	0x3	0x4	0x5	0x6	0x7
0	SP	0	@	P	`	p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3	#	3	C	S	c	s
4	$	4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
A	*	:	J	Z	j	z
B	+	;	K	[	k	{
C	,	<	L	\	l	|
D	-	=	M	]	m	}
E	.	>	N	^	n	~
F	/	?	O	_	o	DEL

ASCII と言われるもの。

Portable Filename Character Set

-	0x2	0x3	0x4	0x5	0x6	0x7
0		0		P		p
1		1	A	Q	a	q
2		2	B	R	b	r
3		3	C	S	c	s
4		4	D	T	d	t
5		5	E	U	e	u

6		6	F	V	f	v
7		7	G	W	g	w
8		8	H	X	h	x
9		9	I	Y	i	y
A			J	Z	j	z
B			K		k	
C			L		l	
D	-		M		m	
E	.		N		n	
F			O	_	o	

ただし - (ハイフン) から始まるファイル名は不可。

目次へ