Linuxサウンド処理基盤
ALSA プログラミング入門

My Linux シリーズ

ハイレゾ音源 WAVE, AIFF, FLAC対応 PCオーディオ・プレーヤを作る

音羽 良
[著]

CQ出版社

第2章 ALSAアプリケーション・プログラミング・インターフェース概要

第1節 ALSAの構成概要

■第1項 ALSAの全体構造

ALSA（Advance Linux Sound Architecture）は，Linuxにおける標準的なサウンド処理基盤です．Linuxでは，UNIXにおける考え方と同様に，OSの中核となるソフトウェア要素の集合を「カーネル」と呼びます．カーネルは，ハードウェアとのインターフェースやメモリ管理，タスク・スケジュール，ファイル管理などを行います．カーネル・プログラムの動作する領域をカーネル空間と言い，一方アプリケーション・プログラムなどのカーネル以外のソフトウェア要素が動作する領域をユーザー空間と言います．

ALSAには，カーネル・プログラムとアプリケーション・プログラムの間を仲介するプログラミング・インターフェースがライブラリの形式で用意されており，ALSAライブラリAPI（Application Programming Interface）と呼ばれています．このALSAライブラリAPIを使えば，カーネルやハードウェアの詳細を意識しないで，オーディオ処理アプリケーションを作成できます．アプリケーション・プログラムから見たALSAのアーキテクチャの概念を**図2-1**に示します．

● ALSAカーネル・ドライバのバージョン確認

Linuxコマンド`cat /proc/asound/version`を実行すると，**リスト2-1**に示すようにALSAカーネル・ドライバのバージョンを表示できます．

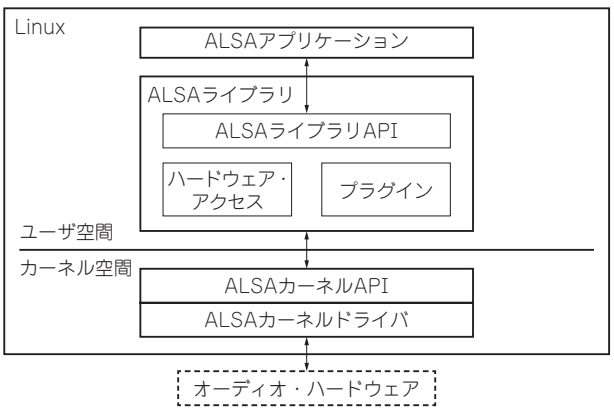

図2-1
ALSAアーキテクチャ階層構成
ALSAライブラリ経由でハードウェアの詳細仕様に依存しないアプリケーションを作成できる

リスト2-1 ALSAカーネル・ドライバのバージョン確認

```
$ cat /proc/asound/version ⏎

Advanced Linux Sound Architecture Driver Version k3.13.0-76-generic.
```

Note

- ALSAプロジェクトは，アプリケーション・プログラマに対して，次に示す理由によりALSAカーネル・ドライバと直接インターフェースするカーネルAPIよりは，ライブラリAPIの使用を推奨しています．
  （a）ALSAライブラリは，カーネルAPIの機能特性を100%提供することに加えてアプリケーション・コードをより簡単・明瞭にする使い勝手を提供していること
  （b）将来の修復や互換コードは，カーネル・ドライバの代わりにライブラリ・コードに設定される可能性があること

■第2項　ALSAのハードウェア・デバイス構成

ALSAライブラリを適用したアプリケーション・プログラムを作成する際に有用な，ALSAの主要概念を説明します．

これら概念の説明は，抽象的になる傾向があるので，ここではLinuxのユーティリティ・コマンド，およびALSAが提供するユーティリティ・プログラム利用して，各概念の具体例を確認しながら進めていきます．なお，以下に示す具体例では，USB DACを接続したPCオーディオ・システムにおける確認結果を示します．

● ALSAユーティリティ

UbuntuなどのALSAをサウンド処理の基盤とするLinuxでは，通常次のようなコマンドライン・ユーティリティ・プログラムが付属提供されます．

```
alsactl         ：サウンド・カード設定管理ユーティリティ
aplay/arecord   ：再生/録音ユーティリティ（.wav，.voc，.auファイル）
amixer          ：ミキサ
alsamixer       ：CUI（Character User Interface）ミキサ
amidi           ：MIDI送受信ユーティリティ
alsaloop        ：PCM ループバック
speaker-test    ：スピーカ試験用トーン・ジェネレータ
iecset          ：IEC958 ステータス・ビット表示/設定ユーティリティ
```

これらユーティリティ・プログラムの中，本書ではaplayをサウンド再生に関わる機能の検証用に使用します．aplayには多くのコマンド・オプションが用意されていますが，本書で使用するオプションは次の3つです．

```
-l, --list-devices ：サウンド・カードおよびディジタル・オーディオ・デバイスを一覧表示するオプション
-D, --device=NAME  ：PCMデバイスをデバイス名で指定するオプション
-v, --verbose      ：PCMデバイスの構造と設定を表示するオプション
```

● 試験音源

本書では，サウンド再生に関わる種々の動作確認のために，次に示す仕様の音叉音を模擬した試験音源を適用します．

▶ ファイル・フォーマット

WAVE，FLAC，AIFF

第3章 ALSAライブラリによるPCMサウンド再生の要点

第1節　PCMサウンド再生処理の流れ

■第1項　再生におけるALSAとアプリケーションの役割

ALSAライブラリを使用したPCMサウンド再生処理のフローは，**図3-1**のようになります．**図3-1**でハッチングを施した処理は，ALSAライブラリの提供するAPI関数（以降，ALSA API）が主体的に行います．一方白抜きの処理は，アプリケーションもしくは，アプリケーションがALSA以外の汎用ユーティリティ・プログラムを使用して行います．各処理の概要は，次のようになります．

▶サウンド・フォーマット情報の取得処理

WAVE形式，FLAC形式等のサウンド・ファイルのデータを解析し，サウンド・フォーマットに関する各種情報（標本化速度，チャネル数，量子化ビット数など）を取得します．

▶PCMデバイスのオープン処理

ALSA APIにより，PCMデバイスをオープンします．

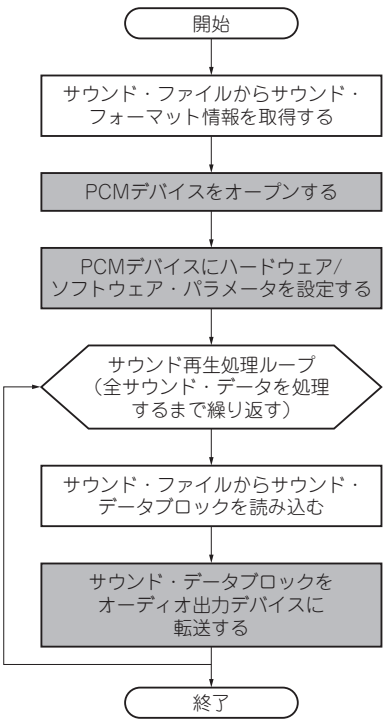

図3-1　PCMサウンド再生処理フロー
灰色部分の処理にはALSA APIを適用する

▶ ハードウェア/ソフトウェア・パラメータの設定処理

ALSA APIにより，サンプル・フォーマットや標本化速度などのPCMデバイス仕様に適合するようにパラメータを設定します．

▶ サウンド・データ読込み処理

サウンド・ファイルからブロック単位でデータを読み込みます．このブロック単位のサウンド・データの塊を，便宜上データブロックと呼ぶことにします．

▶ サウンド・データ出力処理

ALSA APIにより，サウンド・データブロックを出力デバイスに転送します．

第2節　PCMデバイスのオープン/クローズ

PCMデバイスのオープンは，**表3-1**に示すALSA APIにより実行します．例えば，コード`snd_pcm_open(&handle, "plughw:0,0", SND_PCM_STREAM_PLAYBACK, 0);`は，標準PCMデバイス`"plughw:0,0"`を再生ストリーム用としてブロック・モードでオープンし，PCMハンドルを取得することを示します．`mode`を`0`に設定するブロック・モードはデフォルトであり，オープン対象のリソースがすでに別のアプリケーションで使用されている場合，そのリソースが解放されるまで呼び出し元をブロックします．他方，`mode`を`1`またはマクロ定数`SND_PCM_NONBLOCK`に設定する非ブロック・モードは，いかなる場合も呼び出し元をブロックせず，リソースが使用できないときは，負のエラー・コード（`-EBUSY`）を戻します．

いったん，PCMデバイスをオープンすると，以降同デバイスには，取得したPCMハンドルでアクセスすることが可能になります．その意味で，PCMデバイス・オープン処理は，標準的なファイル・オープン処理により，ファイル・ハンドルを取得するのと類似の概念として理解できます．

オープンしているPCMデバイスをクローズするには，**表3-2**に示すALSA APIを実行します．

表3-1　PCMデバイスをオープンするALSA API（snd_pcm_open）

API	`int snd_pcm_open( snd_pcm_t **pcm, const char *name, snd_pcm_stream_t stream, int mode )`
説明	PCMデバイスをオープンする
引き数	`pcm` APIの実行により戻されるPCMハンドル `name` PCMハンドルのASCII識別名 `stream` ストリームの指定 `mode` オープン・モード
戻り値	成功時は`0`を，失敗時は負のエラー・コードを戻す
型定義	`typedef struct _snd_pcm snd_pcm_t`（アプリケーションからは不透明なPCMハンドルの構造体）
列挙型（stream）	`enum snd_pcm_stream_t { SND_PCM_STREAM_PLAYBACK = 0, SND_PCM_STREAM_CAPTURE, SND_PCM_STREAM_LAST = SND_PCM_STREAM_CAPTURE }`

表3-2　PCMデバイスをクローズするALSA API（snd_pcm_close）

API	`int snd_pcm_close (snd_pcm_t *pcm)`
説明	PCMデバイスをクローズして関連する全てのリソースを解放する
引き数	`pcm` PCMハンドル
戻り値	成功時は`0`を，失敗時は負のエラー・コードを戻す

第3節　PCMデバイス関連のパラメータ設定

■第1項　パラメータ構成空間

ALSAライブラリでは，ディジタル・サウンドの特性を規定するパラメータ（標本化速度，チャネル数，サンプル・フォーマットなど）構成を多次元の構成空間として取り扱います．例えば，ある1つの次元が，標本化速度に対応し，別の1つの次元がサンプル・フォーマットに対応するといった概念です．

一般的に，特定のサウンド・カードでは，構成空間に属するパラメータの全ての組み合わせを実現することが不可能な場合もあれば，各パラメータを独立に設定できない可能性もあります．ALSAライブラリを用いれば，このような複雑で微妙なパラメータを構成空間から順次自動的に設定することができます．

PCMデバイス関連のパラメータは，ハードウェア・パラメータとソフトウェア・パラメータに大別されます．以下，順に説明します．

●構成コンテナの割り当て

PCMデバイスの構成パラメータを設定する前に，パラメータ情報の受け皿（いわゆる「コンテナ」）となる変数にリソースを割り当てる必要があります．そのためには，**表3-3**と**表3-4**に示すALSAライブラリのマクロ定義を使用します．

これらのマクロは定義が示すように，実体はCの標準関数alloca，およびmemsetを実行します．

■第2項　ハードウェア・パラメータの設定

ALSAライブラリでハードウェア・パラメータを設定する手順は**図3-2**のようになります．

最初にPCMデバイスに設定可能な全パラメータを含むハードウェア構成空間を定義するために，**表3-5**に示すALSA APIを実行します．また，必須の手順ではありませんが，PCMデバイスで再標本化することによる標本化速度変換を可能にするかどうかを設定するためには，**表3-6**に示すALSA APIを適用することができます．

●アクセス・タイプ

次に，ALSAライブラリでは，サウンド・データを転送する際の転送方式およびインターリーブ/非インターリーブの区分を示すアクセス・タイプを指定する必要があります．アクセス・タイプの設定には，**表3-7**に

表3-3　ハードウェア構成空間コンテナにリソースを割当てるALSA マクロ（snd_pcm_hw_params_alloca）

マクロ	#define　snd_pcm_hw_params_alloca(ptr)　__snd_alloca(ptr, snd_pcm_hw_params) #define　__snd_alloca(ptr, type)　do { *ptr = (type##_t *) alloca(type##_sizeof()); memset(*ptr, 0, type##_sizeof()); } while (0)
説明	PCMデバイスのハードウェア構成空間コンテナにリソースを割り当てる
引き数	ptr　戻されるコンテナへのポインタ
型定義	typedef struct　_snd_pcm_hw_params　snd_pcm_hw_params_t（アプリケーションからは不透過なハードウェア構成空間コンテナの構造体）

表3-4　ソフトウェア構成コンテナにリソースを割当てるALSA マクロ（snd_pcm_sw_params_alloca）

マクロ	#define snd_pcm_sw_params_alloca(ptr)　__snd_alloca(ptr, snd_pcm_sw_params)
説明	PCMデバイスのソフトウェア構成コンテナにリソースを割り当てる
引き数	ptr　戻されるコンテナへのポインタ
型定義	typedef struct　_snd_pcm_sw_params　snd_pcm_sw_params_t（PCMソフトウェア構成コンテナ）

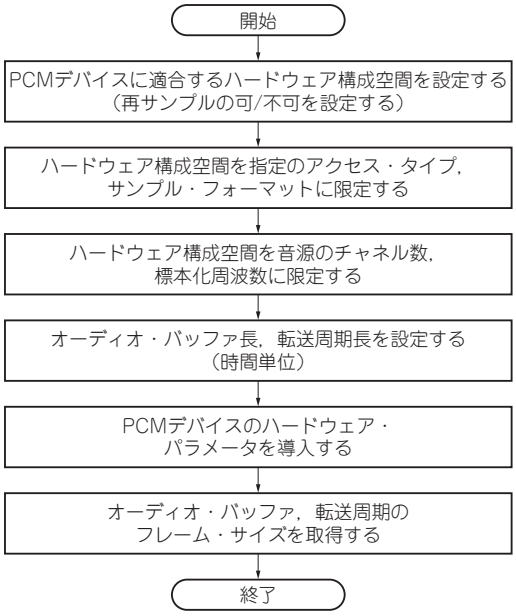

図3-2　ハードウェア・パラメータ設定手順例
ALSAライブラリを用いれば，複雑で微妙なパラメータを順次設定できる

表3-5　全ハードウェア構成空間を定義するALSA API(snd_pcm_hw_params_any)

API	int snd_pcm_hw_params_any (snd_pcm_t *pcm, snd_pcm_hw_params_t *params)
説明	デバイスpcmに設定可能な全パラメータで構成空間paramsを充填する
引き数	pcm　PCMハンドル params　ハードウェア構成空間
戻り値	成功時は0を，失敗時は負のエラー・コードを戻す

表3-6　標本化速度変換の可不可を設定するALSA API(snd_pcm_hw_params_set_rate_resample)

API	int snd_pcm_hw_params_set_rate_resample (snd_pcm_t *pcm, snd_pcm_hw_params_t *params, unsigned int val)
説明	デバイスpcmの構成空間を実際のハードウェアの標本化周波数に限定するか否かを設定する
引き数	pcm PCMハンドル，params ハードウェア構成空間，val 0＝標本化速度変換不可，1＝標本化速度変換可能(デフォルト)
戻り値	成功時は0を，失敗時は負のエラー・コードを戻す

表3-7　アクセス・タイプを設定するALSA API(snd_pcm_hw_params_set_access)

API	int snd_pcm_hw_params_set_access (snd_pcm_t *pcm, snd_pcm_hw_params_t *params, snd_pcm_access_t access)
説明	ハードウェア構成空間を指定のアクセス・タイプに限定する
引き数	pcm　PCMハンドル params　ハードウェア構成空間 access　アクセス・タイプ
戻り値	成功時は0を，失敗時(構成空間が空の場合)は負のエラー・コードを戻す

列挙型(access)	enum snd_pcm_access_t {
	SND_PCM_ACCESS_MMAP_INTERLEAVED = 0, SND_PCM_ACCESS_MMAP_NONINTERLEAVED,
	SND_PCM_ACCESS_MMAP_COMPLEX, SND_PCM_ACCESS_RW_INTERLEAVED,
	SND_PCM_ACCESS_RW_NONINTERLEAVED, SND_PCM_ACCESS_LAST = SND_PCM_ACCESS_RW_NONINTERLEAVED
	}

第5章 WAVE再生プログラム

第1節 WAVEファイル・フォーマット

■第1項 WAVEフォーマットのデータ構造

● RIFFチャンク

WAVEフォーマットは，1991年にIBM社とMicrosoft社から，マルチメディア向けのファイル・フォーマットRIFF（Resource Interchange File Format）に包含される仕様の1つとして初版が公表されました．RIFFは，チャンク（chunk）と呼ばれる**表5-1**に示す構成要素の集合として定義されます．

Note

- **表5-1**のフィールド名称は，IBMおよびMicrosoftの源泉仕様書に準拠しています．本書ではこれ以降，WAVEフォーマットの仕様を示すためにこのフィールド名を適宜使用します．

● 標準PCMのWAVEフォーマット

RIFFの1つである，WAVE（正式にはWaveform オーディオ・ファイル・フォーマット）は，初版では標準的なPCMサウンド・データを含むフォーマットとして，**表5-2**のようなチャンク構造として規定されました．

Note

- WAVEフォーマットは，RIFFチャンクから見るとチャンクが入れ子構造になっています．下位のチャンクをサブチャンクと称して区分する場合もありますが，本書ではチャンクという名称に統一します．
- WAVEフォーマットの仕様では，フォーマット・チャンクおよびデータ・チャンクは必須のチャンクであり，かつ前者は後者に先行することを必要条件としていますが，他にファイル内の物理的な位置関係は，特に規定されていません．従って，ファイル上でフォーマット・チャンクの直後に連続してデータ・チャンクが配置されているとは限らないので，WAVEデータを取り扱うプログラム処理を記述する場合に留意する必要があります．
- 標準PCMとは非PCM形式，または後述するハイレゾ対応に拡張されたPCMと区別するための用語です．IBM，Microsoftの初版仕様書上では“Microsoft PCM format”と記述されていました．
- フォーマット・チャンクのフィールドwBlockAlignの説明にある「サンプル・フレーム」は，第1章で紹介したサウンド・データのひとまとまりのデータ単位のことです．源泉仕様書とは別の説明表現になっていますが，後述する実例プログラムとの関係など，本書内での説明の一貫性を考慮して，このような表現にしています．

● WAVEファイル上の保存形式

16ビットのステレオPCMサウンドのWAVEファイル上での保存形式例を**図5-1**に示します．このデ

表5-1　チャンク構造

チャンク構成要素（フィールド名）	概　要
チャンクID（ckID）	チャンク・データの内容を識別する4文字コード．アプリケーション・プログラムは，認知しないチャンクIDを読み飛ばすことができる
チャンク・サイズ（ckSize）	チャンク・データのバイト・サイズを識別する符号なしの32ビット整数値．この値はチャンクIDフィールド，チャンク・サイズ・フィールドのサイズ，およびチャンク・データ末尾に必要に応じて補填する1バイト（チャンク・データ・フィールドの項を参照）を含まない
チャンク・データ（ckData）	チャンクの実際のデータに相当する2値データ．チャンク・データの先頭は，RIFFファイルの先頭から2バイト境界（すなわち偶数バイト・サイズ）に整列される．もしもチャンク・サイズが奇数の場合，チャンク・データの末尾に値ゼロのデータを1バイト補填して整列する

表5-2　標準PCM WAVEフォーマットのチャンク構造

フィールド	フィールド長（バイト）	説　明
ckID	4	チャンクID：'RIFF'
ckSize	4	チャンク・サイズ：4＋24＋{8＋n＋(0 or 1)}　←フォーマット・チャンク全体のサイズ　←WAVEデータ・チャンク全体のサイズ
formType	4	RIFFの枠組に基づくファイル・フォーマット区分．WAVEに対しては'wave'
〈フォーマット・チャンク〉		
ckID	4	チャンクID：'fmt'
ckSize	4	チャンク・サイズ：16
wFormatTag	2	データ・フォーマット種を示す値．PCMの場合は0x0001
wChannels	2	チャネル数N_c
dwSamplesPerSec	4	標本化速度F_s［標本数/秒］
dwAvgBytesPerSec	4	平均データ転送速度$F_s \times L \times N_c$［バイト/秒］　L：各サンプルのバイト長
wBlockAlign	2	サンプル・フレーム長$L \times N_c$［バイト］
wBitsPerSample	2	量子化ビット数$8 \times L$［ビット/標本］
〈WAVEデータ・チャンク〉		
ckID	4	チャンクID：'data'
ckSize	4	チャンク・サイズ：$n = L \times N_c \times N_s$　N_s：全フレーム数
サウンド・データ	n	ディジタル・サウンド・データ（PCM符号は，2の補数表現）
補填バイト	0または1	nが奇数の場合，1バイトを加えて偶数バイトにする

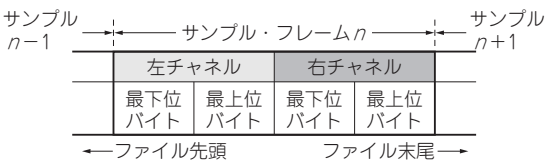

図5-1　WAVEファイル上のデータ保存形式例
サウンド・データはリトル・エンディアン方式で保存される

タ形式の特性は次のようになります．

- 複数チャネルのサウンド・データはインターリーブされます．
- 各サンプル値は符号付き整数値であり，バイト配置順序は最下位バイト（Least Significant Byte）が最初に保存されるリトル・エンディアン方式です．
- サンプルの振幅を示す実効的なビット列は最上位バイト（Most Significant Byte）から詰めて保存され，残りのビットはゼロに設定されます．例えば，12ビットのサウンド・サンプルを2バイトの整数として保存する場合，最下位バイトの最下位ビット側の4ビットはゼロに設定されます．
- 負のサンプル値は，2の補数表現となります．

表5-3　非PCM WAVEフォーマットのチャンク構造

フィールド			フィールド長（バイト）	説　明
ckID			4	チャンクID：'RIFF'
ckSize			4	チャンク・サイズ：4 + 26 + {8 + n + (0 or 1)}
	formType		4	RIFFの枠組に基づくファイル・フォーマット区分．WAVEに対しては'wave'
	〈フォーマット・チャンク〉			
	ckID		4	チャンクID：'fmt'
	ckSize		4	チャンク・サイズ：*18*（値変更）
		wFormatTag	2	データ・フォーマット種を示す値．PCMの場合は0x0001
		wChannels	2	チャネル数N_c
		dwSamplesPerSec	4	標本化速度F_s【単位：標本数/秒】
		dwAvgBytesPerSec	4	平均データ転送速度【単位：バイト/秒】
		wBlockAlign	2	サンプル・フレーム長【単位：バイト】
		wBitsPerSample	2	量子化ビット数【単位：ビット/標本】
		cbSize（追加）	*2*	***非PCMデータ・フォーマットに必要な別途情報のサイズ【単位：バイト】***
	〈WAVEデータ・チャンク〉			
	ckID		4	チャンクID：'data'
	ckSize		4	チャンク・サイズ：n
		サウンド・データ	n	ディジタル・サウンド・データ（PCM符号は，2の補数表現）
		補填バイト	0または1	nが奇数の場合，1バイトを加えて偶数バイトにする

Note

- 初版のWAVEフォーマット仕様では，2チャネルを越えるスピーカ・マッピングは未定義でした．
- WAVEファイルをビッグ・エンディアン系のPCプラットフォーム上のアプリケーションから直接データ処理する場合には，エンディアンの変換処理が必要になります．本書では，リトル・エンディアン系のPCプラットフォームでのアプリケーション作成を前提としているため，エンディアン変換処理は不要として取り扱います．

● 非PCM WAVEフォーマット

1994年にMicrosoftからWAVEフォーマットの改訂内容が開示されました．端的に言って，この改訂は標準PCM以外の非PCMのWAVEデータ・フォーマット種を追加するための仕様追加でした．本書では，標準PCMおよびそのハイレゾ拡張PCMのみを扱うため，この改訂については**表5-3**に示すように，フォーマット・チャンクの変更・追加部分のみを示します．

ただし，この表が示すフォーマット構造は，次に示すハイレゾ対応のWAVEフォーマットの拡張構造に包含されるため，後の実例プログラム・データ構造を規定する際に便宜上利用します．

● ハイレゾ/多チャネル対応の拡張WAVEフォーマット（WAVEFORMATEXTENSIBLE）

2001年にMicrosoft社は，次の各ケースに対応したWAVEフォーマットの拡張版を開示しました．

- PCMサウンド・データの量子化ビット数が16ビット以上の場合．
- チャネル数が2以上の多チャネル構成の設定が必要な場合．

1つ目の改訂事項が，ハイレゾ・オーディオに関係します．ハイレゾPCMサウンド関連の情報に焦点を絞ったWAVEフォーマットの拡張内容を**表5-4**に示します．

この拡張したWAVEフォーマット・ファイルに，サウンドを保存する場合，実際のデータ・フォーマットはSubFormatフィールドで指定します．このフィールドを表現するために，GUID（Globally Unique Identifier）と呼ばれる16バイトのデータ構造が適用されます．GUIDの最初の2バイトはデータ・フォーマ[illegible]示す

表5-4　ハイレゾ/多チャネル対応WAVEフォーマットのチャンク構造

フィールド	フィールド長（バイト）	説　明
ckID	4	チャンクID：'RIFF'
ckSize	4	チャンク・サイズ：4 + 48 + {8 + *n* + (0 or 1)}
formType	4	RIFFの枠組に基づくファイル・フォーマット区分．WAVEに対しては'wave'
〈フォーマット・チャンク〉		
ckID	4	チャンクID：'fmt'
ckSize	4	チャンク・サイズ：40（値変更）
wFormatTag	2	0xFFEに設定（値変更）
wChannels	2	チャネル数
dwSamplesPerSec	4	標本化速度［標本数/秒］
dwAvgBytesPerSec	4	平均データ転送速度［バイト/秒］（定義の厳密化）
wBlockAlign	2	サンプル・フレーム長［バイト］
wBitsPerSample	2	サウンド・コンテナの量子化ビット数［ビット/標本］
cbSize	2	PCMでは拡張情報のサイズ＝22に設定［バイト］（値変更）
wValidBitsPerSamples	2	サウンドの正味の量子化ビット数
（wSamplesPerBlock）	（2）	（圧縮フォーマットに適用．圧縮データ・ブロックに含まれる固定長のサンプル数）
（wReserved）	（2）	（将来の利用に対して予約済）
dwChannelMask	4	スピーカ位置に対するチャネルの対応付けを指定するビット・マスク
SubFormat	16	GUIDデータ構造に基づくデータ・フォーマット
〈WAVEデータ・チャンク〉		
ckID	4	チャンクID：'data'
ckSize	4	チャンク・サイズ：*n*
サウンド・データ	*n*	ディジタル・サウンド・データ（PCM符号は，2の補数表現）
補填バイト	0または1	*n*が奇数の場合，1バイトを加えて偶数バイトにする

（追加）

残りの14バイトは固定の文字列データを示します．ハイレゾPCMデータの場合，データ・フォーマットのフィールドには0x0001を設定します．

Note

- ハイレゾPCMの再生に限定すれば，表5-4で（）付きのフィールドは考慮不要ですが，将来の拡張のための参考情報として表示しています．
- サウンド・データのビット数を示すwBitsPerSampleとwValidBitsPerSamplesの関係はいささか分かりづらいですが，例えば正味20ビットに量子化されたサウンド・データに対しては，wValidBitsPerSamples = 20，wBitsPerSample = 24に設定する必要があります．すなわち，拡張WAVEフォーマットの仕様から，wBitsPerSampleは8の整数倍となることが厳格に要求されるようになりました．Microsoftの仕様書の記述では，この値を「コンテナ・サイズ」と称しています．

■第2項　WAVEファイルのフォーマットを規定するデータ構造

第1項で説明したWAVEファイル・フォーマットに対して，ハイレゾ音源を含むPCMサウンドに関わるデータ構造を**リスト5-1**に示すようにヘッダ・ファイルWaveFormat.hに定義します．

このヘッダ・ファイルの定義の要点は，次のようになります．

- 仕様から，WAVEフォーマットは2バイト単位で整列されるため，その構造を明確にするためにWORD型，DWORD型，FOURCC型を定義します．これらの定義および型名は，IBMおよびMicrosoftの源泉仕様書に準拠しています．
- ハイレゾ音源対応WAVEフォーマットに含まれるGUIDデータ構造を定義します．
- 再生アプリケーション・プログラムが参照するWAVEデータ・フォーマットに関わるデータ構造

My Linux
シリーズ