JPPA-1-2018

BWF-J オーディオファイルフォーマット

第 3.1 版

2018 年 5 月

2000 年　5 月　第 1 版
2004 年　7 月　第 2 版
2009 年 11 月　第 3 版
2018 年　5 月 第 3.1 版

一般社団法人日本ポストプロダクション協会
オーディオ部会 オーディオ基準小委員会 BWF-J WG
http://www.jppanet.or.jp

はじめに

第 3 版発行から 8 年余りが経過し、BWF-J の主力市場である放送業界も大きく変化した。第 3 版の記述の中には現在の市場と合わない部分もあり、一部修正を行い第 3.1 版として発行することにした。

現在、BWF-J フォーマットを採用した録音・編集機器は広く運用されており、放送局を中心に業務用オーディオファイルフォーマットのひとつとして定着している。本規格がさらに利用されることを願うものである。

第 3.1 版の主な改訂内容

1．3.5 インチ MO ディスクの供給及び運用の終了に伴い「解説 A　3.5 インチ MO ディスクの運用規定」を削除した。
2．BWF-J、BWF 等の WAVE ファイルを作成する機器（アプリケーション）の中に、RIFF 規格の fmt chunk と異なるものが存在する。これについての説明を「解説 A」として追加した。
3．bext chunk のバージョンに関する説明を「解説 E」として追加した。

2018 年 5 月
一般社団法人 日本ポストプロダクション協会
技術委員会　オーディオ部会オーディオ基準小委員会
BWF-J WG 議長　丸谷 正利

目　　次

第 1 章　一般事項

1.1　目的

本規格は業務用途デジタルオーディオのファイルフォーマットを規定するものであり、本規格により番組制作、番組交換が円滑に実施できることを目的としている。

1.2　参照規格

BWF-J ファイルフォーマットは EBU の Tech-3285(BWF)を基に機能拡張したものである。したがって、下記の規格は本規格の一部を構成している。Bext chunk 内の UMID フィールドに関しては、SMPTE ST 330 規格を参照すること。

[1] EBU; Tech-3285 version1 Specification of the Broadcast Wave Format (BWF)

[2] SMPTE; ST 330 Unique Material Identifier (UMID)

本規格で使用する文字コードは下記の 2 種類とする。日本語の場合、JIS X 0208 付属書 1（SHIFT-JIS）で規定されていない文字コードを使用すると、他の機器で文字化けなどを生じるので注意する。

[3] ISO/IEC 646 ISO 7-bit coded character set for information interchange (ASCII)

[4] JIS X 0208 付属書 1 (SHIFT-JIS)

1.3　参考文献

BWF 及び BWF-J 規格は RIFF（Resource Interchange File Format）仕様に基づいて策定されている。RIFF の詳細に関しては下記の文献を参照のこと。

[5] Microsoft; Multimedia Programming Interface and Data Specification v1.0 / 1991

[6] Microsoft; Multimedia Data Standards Update / 1994

ラジオ放送用途では、CM 及び番組の搬入基準として制限された運用基準が設けられている。本規格に準拠した機器を運用するユーザーは下記文献を参照されたい。ただし、CM の場合はラジオ CM オンライン送稿システム上で BWF-J ファイルの作成が可能となっている。

[7] 日本民間放送連盟;　ラジオ CM 素材搬入基準

[8] 日本民間放送連盟;　民放連技術規準 R024-2018　音声ファイルによる番組交換

1.4　ファイル名について

1.4.1　ファイル名の長さと拡張子

WAVE ファイル名は拡張子を含めて 128 バイト以内とする。拡張子は“wav”を使用する。

1.4.2　ファイル名の文字コード

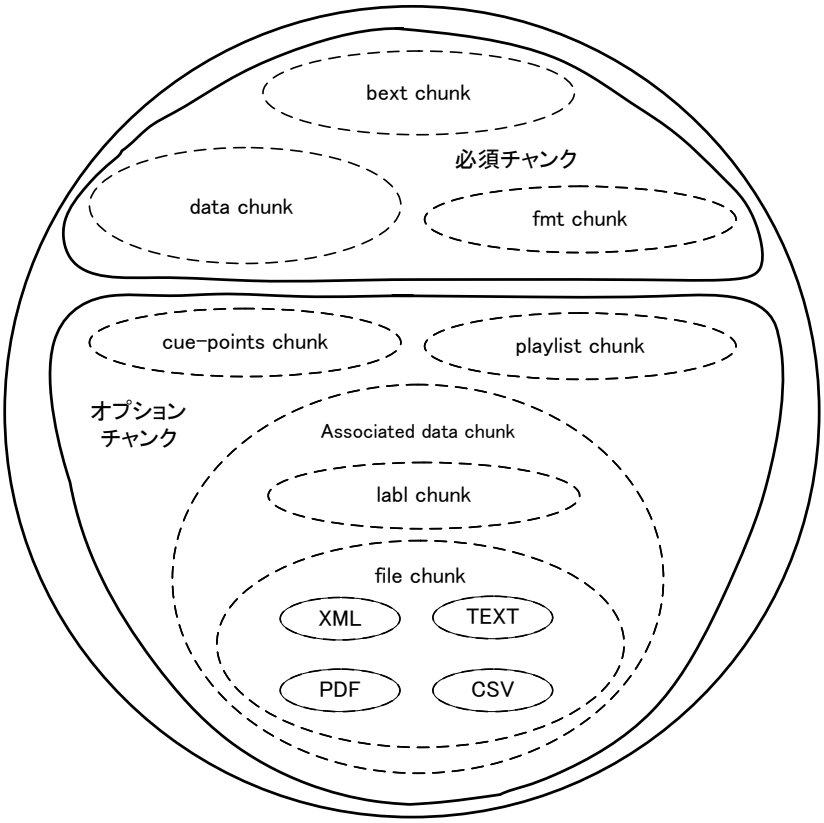
WAVE ファイル内で使用可能な文字コードは 1.2 項で規定しているが、ファイル名に使用する文字コードは機器に依存するので規定しない。ただし、BWF-J ファイルとして外部へ提出する場合は、受け取り側で文字化け等が発生しないように考慮する必要がある。

第 2 章　ファイルフォーマット

BWF-J(Broadcast Wave Format－Japan)は EBU の Tech-3285(BWF)規格を基に、日本国内の業務で必要とする機能拡張を行ったオーディオファイルフォーマットである。BWF 及び BWF-J は RIFF 規格書の WAVE（Waveform Audio File Format）を基に策定されている。

2.1　BWF-J ファイルの概要

BWF-J ファイルは以下の必須チャンクとオプションチャンクから構成する（図－1）。BWF-J ファイルの中に本規格でサポートしていないチャンクが存在した場合は、そのチャンクを無視し削除等は行わない。



図－１　BWF-J（Broadcast Wave Format－Japan）ファイルの構成要素

2.1.1　必須チャンク

以下のチャンクは BWF-J ファイルを構成するために必須となるチャンクである。

- broadcast audio extension chunk (bext chunk)
- format chunk (fmt chunk)
- wave-data chunk (data chunk)

2.1.2　オプションチャンク

以下のチャンクは放送用途で BC$ラベルを使用する際に必要となるチャンクである。BC$ラベル機能をサポートしない場合は不要である。

・cue-points chunk (cue chunk)
・playlist chunk (plst chunk)
・label chunk (labl chunk)
・file chunk (file chunk)

fmt chunk は data chunk より前に位置しなければならないが、bext chunk 及びオプションチャンクの位置は規定しない。

2.2　BWF-J ファイルの構成

BWF-J ファイルの構成は以下のようになる。

```
<WAVE-form> ->
    RIFF('WAVE'
        <fmt chunk>                    // Format of the audio signal: PCM
        <bext chunk>                   // Information on the audio sequence
        <cue-points chunk>             // Optional chunks
        <playlist chunk>               // Optional chunks
        <assoc-data-list> ->    LIST('adtl'     // Optional chunks list
            <labl chunk>               // Optional chunks
            <file chunk> )             // Optional chunks
        )
        <wave-data>                    // Sound data chunk
```

2.2.1　データ型

本規格書で使用しているデータ型について定義する。

<CHAR>	8 ビットまたは 16 ビットの文字記号。
<BYTE>	8 ビットの符号なし整数値。
<WORD>	インテル形式の 16 ビット符号なし整数値。
<DWORD>	インテル形式の 32 ビット符号なし整数値。
<ZSTR>	NULL コードで終端した文字列。
<FOURCC>	4 バイトの ASCII 文字記号。4 バイトに満たない場合はスペースコードを使用して 4 バイトにする。

2.2.2　Broadcast Audio Extension chunk (bext chunk)

下記の要素から構成する。このチャンク内の幾つかのフィールドで使用可能となっている SHIFT-JIS(S-JIS)コードは、日本国内における運用の利便性を図るために規定したものであり、日本国内での運用を前提としている。BWF-J ファイルを国際番組交換ファイルとして使用する場合は EBU/BWF ファイルとの互換性を図るために、bext chunk の文字コードに SHIFT-JIS を用いてはならない。

```
Broadcast_audio_extension typedef struct
    {
    DWORD   ckID;                   // (broadcastextension)ckID=bext
    DWORD   ckSize;                 // size of extension chunk
    BYTE    ckData[ckSize];         // data of the chunk
    }
typedef struct broadcast_audio_extension
    {
    CHAR    Description[256];   // ASCII or S-JIS, Description of the sound sequence
    CHAR    Originator[32];             // ASCII or S-JIS, 制作会社、制作者名
    CHAR    OriginatorReference[32];    // ASCII or S-JIS, 制作ユニークコード
    CHAR    OriginationDate[10];    // ASCII, 制作年月日 yyyy-mm-dd
    CHAR    OriginationTime[8];     // ASCII, 制作時刻 hh:mm:ss
    DWORD   TimeReferenceLow;       // First sample count since midnight, low word
    DWORD   TimeReferenceHigh;      // First sample count since midnight, high word
    WORD    Version;                // BWF-J のバージョン番号
    BYTE    UMID_0;                 // Binary Byte 0 of SMPTE UMID
    ....
    BYTE    UMID_63;                // Binary Byte 63 of SMPTE UMID
    BYTE    Reserved[190];      // 190 Bytes, reserved for future use, set to “NULL”
    CHAR    CodingHistory[];        // ASCII or S-JIS, Coding history
    }
BROADCAST_EXT
```

フィールド定義

Description　256 バイト以下の ASCII 又は SHIFT-JIS で記述した文字列で、WAVE ファイルに関する説明を記述する。データを記述しないか、文字列の長さが 256 バイト未満の場合は、最初の未使用バイトに終端コードとして”NULL”を設定する。

Originator　32 バイト以下の ASCII 又は SHIFT-JIS で記述した文字列で、WAVE ファイルの制作会社、制作者名を記述する。データを記述しないか、文字列の長さが 32 バイト未満の場合は、最初の未使用バイトに終端コードとして”NULL”を設定する。

OriginatorReference　32 バイト以下の ASCII 又は SHIFT-JIS で記述した文字列で、WAVE ファイルの管理に使用する。制作団体によって割り当てられる整理コード（ユニークコード）の記述も可能である。データを記述しないか、文字列の長さが 32 バイト未満の場合は、最初の未使用バイトに終端コードとして”NULL”を設定する。

OriginationDate

10 文字の ASCII 文字列で、wave-data の制作年月日を記述する。
[形式] : yyyy-mm-dd

“yyyy”	＝ 年を表す 4 文字。0000 から 9999 までの値。
“ - ”	＝ セパレータを表す 1 文字。
“mm”	＝ 月を表す 2 文字。01 から 12 までの値。
“ - ”	＝ セパレータを表す 1 文字。
“dd”	＝ 日を表す 2 文字。01 から 31 までの値。

全ての構成要素は必ず記述する。

原則としてハイフン記号(-)を年月日のセパレータとして使用するが、機器側は代替のセパレータとして、アンダースコア(_)、コロン(:)、スペース( )、ピリオド(.)も認識すること。

OriginationTime

8 文字の ASCII 文字列で、wave-data の制作時刻を記述する。制作時刻を記述しない場合はデフォルト値として 00:00:00 を設定する。
[形式] : hh:mm:ss

“hh”	＝ 時を表す 2 文字。00 から 23 までの値。
“ : ”	＝ セパレータを表す 1 文字。
“mm”	＝ 分を表す 2 文字。00 から 59 までの値。
“ : ”	＝ セパレータを表す 1 文字。
“ss”	＝ 秒を表す 2 文字。00 から 59 までの値。

全ての構成要素は必ず記述する。

原則としてコロン記号(:)を時刻表現のセパレータとして使用するが、機器側は代替のセパレータとして、アンダースコア(_)、ハイフン(-)、スペース( )、ピリオド(.)も認識すること。

TimeReferenceLow
TimeReferenceHigh

この 2 つのフィールドでタイムコードのスタート時間を記述する。
データ形式は符号無し 64 ビットで、<TimeReferenceLow>が下位の 32 ビット、<TimeReferenceHigh>が上位の 32 ビットを構成する。値は wave-data の第 1 サンプルに対する真夜中(00 時 00 分 00 秒)からのサンプル数を記述する。1 秒間のサンプル数はサンプリング周波数に依存し、fmt チャンクの<nSamplesPerSec>フィールドで定義され、デフォルト値は ”0” である。

例 : <nSamplesPerSec> = 48000 の場合、<TimeReference>フィールドの 64 ビット値が 10 進数で 48000 であれば、タイムコードのスタート時間は 00 時 00 分 01 秒となる。

Version	本規格のバージョンを示す符号無しバイナリ値。本規格では ”1” を使用する（バージョン 1）。
UMID	SMPTE ST 330 で定義している拡張 UMID（64 バイト）を記述する。未使用の場合は 64 バイトを”0”に設定する。ベーシック UMID（32 バイト）を使用した場合は、残りの 32 バイトを”0”に設定する。 （注） UMID の有効バイト数は、UMID の先頭に記述されている。
Reserved	190 バイトの予約エリア。全て”0”に設定する。
CodingHistory	ASCII または SHIFT-JIS で記述した可変長ブロックの文字列で、複数行を記述する場合は<CR/LF>コードによって改行する。文字列の終端は”NULL”を設定する。 各文字列には、wave-data に適用されたコーディングプロセスを記述する。新しいコーディングプロセスに対しては、新しい文字列を追加する。BWF と互換性のある記述方法については解説 C を参照。

2.2.3 Format chunk (fmt chunk)

wave-data より前に位置し以下の要素から構成する。

（注）RIFF 規格の PCM WAVE ファイルは、fmt chunk フィールドの長さが 16 バイトであるが、これと異なる fmt chunk も存在する。詳細については「解説 A　拡張 fmt chunk について」を参照のこと。

```
<fmt-ck> ->
    fmt (       <common-fields>
                <format-specific-fields> )
<common-fields> ->
    struct
        {
        WORD        wFormatTag;         // Format category, (0x0001:PCM)
        WORD        nChannels;          // Number of channels
        DWORD       nSamplesPerSec;     // Sampling rate
        DWORD       nAvgBytesPerSec;    // For buffer estimation
        WORD        nBlockAlign;        // Data block size
        }
<PCM-format-specific> ->
    struct
        {
        WORD wBitsPerSample;            // Samplesize
        }
```

フィールド定義

wFormatTag	wave-data のフォーマットカテゴリーを示す番号。本規格では値として 0x0001(PCM)を使用する。
nChannels	wave-data のチャンネル数を示す番号。モノラルは値 0x0001、ステレオは値 0x0002 を使用する。
nSamplesPerSec	wave-data のサンプリング周波数を記述する。
nAvgBytesPerSec	wave-data の平均転送バイト数（バイト／秒）を記述する。再生側ではこの値によりバッファの大きさを見積ることが出来る。
nBlockAlign	連続した wave-data のブロックサイズを記述する。再生側では一度に複数（チャンネル数分）の<nBlockAlign>で示されたバイト数のデータを処理する必要がある。したがって、<nBlockAlign>はバッファサイズを確保するために使用することが出来る。
wBitsPerSample	wave-data の量子化ビット数を示す値を記述する。

2.2.4　Wave-data (data chunk)

オーディオデータの本体で、fmt chunk より後ろに位置し以下の要素から構成する。本規格ではリニア PCM 信号（フォーマットカテゴリー1）のみを扱う。

```
<wave-data> ->
    { <data-ck> }
<data-ck> ->
    data        ( <wave-data> )
```

2.2.5　オプションチャンク

Cue-points chunk, Playlist chunk, Associated data chunk（labl chunk, file chunk）を使用する。

2.2.5.1　Cue-points chunk

Cue-point の登録を行う。wave-data に対し 99 個の cue-point を登録することが出来る。登録してある cue-point を実行にするには、playlist chunk の play-segment に記述する。cue-points chunk は以下の要素から構成する。

```
<cue-ck> -> cue (  <dwCuePoints:DWORD>     // Count of cue-points
                   <cue-point>... )        // Cue-point table
<cue-point> ->
```

```
struct
    {
    DWORD       dwName;                 // 32bit cue point number
    DWORD       dwPosition;             // 本規格では未使用とする
    FOURCC      fccChunk;               // Four character code
    DWORD       dwChunkStart;           // 本規格では未使用とする
    DWORD       dwBlockStart;           // 本規格では未使用とする
    DWORD       dwSampleOffset;         // cue point address
    }
```

フィールド定義

dwName　Cue-point 番号を示す任意の 4 バイトデータ。同じ番号（データ）は使用できない。本規格では 32 ビットのデータとして認識する（文字列として認識しない）。<dwName>として 0x00000000 は使用できない。

dwPosition　本規格では未使用。値 ”0” を設定する。使用する場合は<dwSampleOffset>と同じ値とする。

fccChunk　本規格では値として ASCII 文字記号 ”data” を使用する。

dwChunkStart　本規格では未使用。値 “0” を設定する。

dwBlockStart　本規格では未使用。値 “0” を設定する。

dwSampleOffset　本規格では cue-point アドレスとして使用する。wave-data の最初からのサンプル数を値とする。

2.2.5.2 Playlist chunk

本規格では BC$ラベル（第 3 章参照）の実行に使用する。Play-segment に BC$ラベルを持つ cue-point を登録することで wave-data の再生制御を行う。Wave-data に対して 99 個の play-segment を登録することが出来る。

本来の playlist は、play-segment の記述順に実行するものであるが、本規格では BC$ラベルを持つ play-segment は時間軸順に実行する（<dwSampleOffset>値の小さい順に実行）。

```
<playlist-ck> ->    plst(  <dwSegments:DWORD>      // Count of play segments
                           <play-segment>... )     // Play-segment table

<play-segment> ->
    struct
        {
```

```
DWORD       dwName;             // Cue-point 番号
DWORD       dwLength;           // セクションのサンプル数
DWORD       dwLoops;            // セクションの再生回数
}
```

フィールド定義

dwName　Cue-point 番号を示す任意の 4 バイトデータを記述する。同じ番号（データ）は使用出来ない。本規格では 32 ビットのデータとして取り扱う。
<dwName>として 0x00000000 は使用できない。

dwLength　再生するセクションの長さをサンプル数で指定する。BC$ラベルを持つ play-segment ではこの値を無視し、次の play-segment まで BC$ラベルで定義された動作を続ける。

dwLoops　再生するセクションの回数を指定する。“1” で 1 回再生とする。“0”の場合は再生しない。BC$ラベルを持つ play-segment ではこの値を無視し、次の play-segment まで BC$ラベルで定義された動作を続ける。

2.2.5.3　Associated data chunk

Labl chunk と file chunk を使用する。Labl chunk は cue-point に BC$ラベルを付加するために使用する。File chunk は放送運用データなど、異なる形式のファイルを添付（ファイルの格納）する場合に使用する。

```
<assoc-data-list> ->    LIST(   'adtl'
                                <labl-ck>       // Label
                                <file-ck> )     // Media file

<labl-ck> ->            labl(       <dwName:DWORD>
                                    <data:ZSTR> )

<file-ck> ->            file(       <dwName:DWORD>
                                    <dwMedType:DWORD>
                                    <fileData:BYTE>... )
```

フィールド定義

dwName　Cue-point 番号を示す任意の 4 バイトデータを記述する。同じ番号（データ）は使用出来ない。本規格では 32 ビットのデータとして取り扱う。
<dwName>として 0x00000000 は使用できない。

data　“NULL”で終端した文字列。本規格では BC$ラベル名の登録に用い、cue-

	point 番号（<dwName>）に対応した BC$ラベルとして使用する。
dwMedType	未使用。本規格では”NULL”を設定する。
fileData	添付するファイルの本体。CSV、PDF、XML、TEXT の各ファイルを格納することが出来る（3.3 参照）。

第 3 章　CUE 信号と放送運用データ

放送局の運用で必要となる CUE 信号、放送運用データの添付について以下のように規定する。この機能はオプションであり、すべての機器がサポートする必要はないが放送用途では必須となる。この仕様は本規格独自のものであり BWF と互換性は無い。

3.1　CUE 信号

BC$ラベルを使用することで、放送運用で必要となる再生機器の制御及び CUE 信号出力（制御信号出力）を、任意のタイミングで行うことが可能となる。BC$ラベルは cue-points chunk と playlist chunk 及び labl chunk（Associated data chunk）から構成する。

Cue-point は wave-data に対し 99 まで登録することができ、labl chunk は個々の cue-point に対し BC$ラベル名を付加するのに使用する。“BC$”で始まる cue-point ラベルは特別な意味を持ち、この 3 文字で始まるラベルは予約名とする。

１つの BWF-J ファイル内で使用する個々の BC$ラベルについては、BC$NOTE＊ラベルを除いてその使用回数を制限しない。同一 BC$ラベルについて使用回数制限が必要な場合は、ユーザー側の運用で対応する必要がある。

3.2　BC$ラベル

BC$ラベルは再生動作の制御を行う、特別な意味を持つラベルである。BC$ラベルは labl chunk を用いてラベル名の登録と cue-point に対する関連付けを行う。再生動作中に BC$ラベルの付加された cue-point に到達すると、再生機器は BC$ラベルで定義されている動作を実行する。

3.2.1　用語の定義

コマンド	機器の釦操作や外部からの制御信号のこと。
Playlist 動作	Playlist に記述されている play-segment を順に実行する動作。番組や cue-point の編集作業などがこれに該当する。
ON-AIR 動作	番組送出（放送）動作。BC$ラベルを実行しながら番組を再生する。
BC$＊＊TALLY 信号	BC$ラベルを実行することで生じる機器の動作状態を示す出力信号。レベル信号となる（その状態が続いている間、出力する）。
BC$＊＊CUE 信号	BC$ラベルを実行することで出力する制御信号。パルス出力。
本編	番組の本体部分。一般に民間放送の番組は本編と CM から構成されている。

3.2.2　BC$ラベルの動作定義

BC$START	再生開始点ラベル。BC$START は他のラベルと異なり、実行しても機器動作上の変化は発生しない。このラベルの目的は音声ファイル上に“マーク”を打つことにある。Playlist 動作では BC$START ラベルを“マーク”として使用することで、効率よく再生開始点やチェックポイントの“頭だし”が可能となる。

ON-AIR 動作中に BC$START ラベルに到達しても動作上の変化はなく、再生動作を継続する。
Playlist 動作では、このラベルを選択（playlist の play-segment を選択）した状態で PLAY コマンドを受けると、その cue-point から再生動作を開始する。

BC$STANDBY　再生動作の待機（スタンバイ）ラベル。再生動作中に BC$STANDBY ラベルの cue-point に到達すると、その位置で再生動作を停止し待機状態に入る。この時、再生機器は BC$STANDBY-TALLY 信号を出力する。待機中に PLAY コマンドを受け付けると BC$STANDBY-TALLY 信号の出力を終了し、再生動作を開始する。
ON-AIR 動作時は WAVE ファイルの読込みと同時に playlist を検索し、最初の BC$STANDBY ラベルの cue-point（<dwSampleOffset>値が一番小さい cue-point）で待機する（ON-AIR 番組の自動頭出し）。

BC$CM　CM 差し替え制御ラベル。再生動作中に BC$CM ラベルの cue-point に到達すると BC$CM-CUE 信号を出力する。本編の再生動作は継続する。

BC$END　本編の再生終了ラベル。再生動作中に BC$END ラベルの cue-point に到達すると BC$END-CUE 信号を出力する。再生動作は継続する。

BC$STOP　再生動作終了ラベル。再生動作中に BC$STOP ラベルの cue-point に到達すると BC$STOP-CUE 信号を出力し、再生動作を終了する。
BC$STOP ラベルは WAVE ファイルの再生動作を終了するラベルであるが、ON-AIR 動作時は誤操作防止のため、次に再生する番組（WAVE ファイル）を選択できる状態になることが望ましい。待機状態（BC$STANDBY）や playlist の先頭行に戻す処理は、次の番組準備完了と間違う恐れがある。

BC$FILE　CM 一本化ファイル再生用ラベル。CM-BANK のバックアップ再生動作などで使用するラベルで、必ず BC$STANDBY ラベルと対で使用する。
BC$FILE ラベルは以下の複合動作を行う。
１）再生動作中に BC$FILE ラベルの cue-point に到達すると BC$END-CUE 信号を出力する。
２）続けて playlist に記述された次の BC$STANDBY ラベル（cue-point）に移動し待機する。この時 BC$STANDBY-TALLY 信号を出力する。BC$FILE と BC$STANDBY ラベルの間にある他の BC$ラベルは無視する。Playlist の中に次に待機する BC$STANDBY ラベルが見つからなければ再生動作を継続する。

BC$PAUSE　再生動作の一時停止ラベル。再生動作中に BC$PAUSE ラベルの cue-point に

到達すると、再生動作を一時停止する。一時停止中は BC$PAUSE-TALLY 信号を出力する。

BC$NOTE＊　ファイル添付ラベル。WAVE ファイルの中に別のファイル（放送運用データなど）を添付するときに使用する。このラベルは playlist に記述しない。「＊」部には 1～9 の数字を使用（BC$NOTE1～BC$NOTE9）し、最大 9 個のファイルを添付することが出来る（同一ラベル名は使用できない）。
BC$NOTE ラベルの cue-point アドレス値<dwSampleOffset≥は 0x00000000（wave-data の先頭）とする。

BC$UTL＊　汎用信号出力ラベル。再生動作中に BC$UTL＊ラベルの cue-point に到達すると BC$UTL＊-CUE 信号を出力する。本編の再生動作は継続する。「＊」部には 1～4 の数字を使用する（BC$UTL1～BC$UTL4）。ユーザーが任意の機能として使用できる CUE 信号出力である。
BC$UTL ラベルの CUE 信号出力は、運用により動作の意味が異なるので、使用に当たっては事前に関係者間での確認が必要である。

3.3　添付ファイル

BC$FILE ラベルを使用することで、放送運用上必要となる文書ファイルを最大 9 個まで WAVE ファイルに添付することが出来る。添付可能なファイル形式は CSV、PDF、XML、TEXT の 4 種類である。

3.3.1　添付ファイルの種類

CSV 形式　拡張子“csv”を持つテキストファイルで、1 レコードを 1 行で表す。レコード内のデータ項目はカンマ(,)で区切り、レコード間の区切りは改行コード<CR/LF>を使用する。文字データはダブルクォート(“)で囲み、数値データと区別する。ファイルの内容は運用する側が自由に決定する。

PDF 形式　拡張子“pdf”を持つ Portable Document Format 仕様のファイルである。ファイルの内容は運用する側が自由に決定する。

XML 形式　拡張子“xml”を持つ XML マークアップ言語で記述されたファイルである。ファイルの内容は運用する側が自由に決定する。

TEXT 形式　拡張子“txt”を持つプレーンテキストで記述されたファイルである。ファイルの内容は運用する側が自由に決定する。

3.3.2　添付ファイルの作成とファイル名の長さ及び文字コード

WAVE ファイルに添付するファイルは、他のコンピュータシステム（パーソナルコンピュータなど）で作成しても良い。添付ファイルのファイル名は、拡張子を含め 128 バイト以内とする。ファイル名

の文字コードは機器に依存するので規定しない。

3.3.3　添付ファイル本体の文字コード

添付ファイル本体（データ部分）はバイナリデータと同様の扱いとするので、文字コードの制限は無い。

3.3.4　添付ファイルの格納

ファイルの添付は BC$NOTE＊ラベルを使用して行う。添付ファイル本体は file chunk の中の<fileData>フィールドに格納する。<fileData>フィールドの先頭（1 行目）は添付するファイル名をテキスト形式で記述する。使用する文字コードは ASCII 又は SHIFT-JIS を使用する。1 行目は改行コード（CR/LF=0x0D/0x0A）で終了し、改行コード以降のデータが添付ファイルのデータ本体となる。

<fileData>フィールドの記述例：

例 1：添付するファイル名が「ON-AIR-DATA1.csv」の時、<fileData>フィールドの 1 行目には下記のコードが記述される。

HEX 表記：4F 4E 2D 41 49 52 2D 44 41 54 41 31 2E 63 73 76 0D 0A

例 2：添付するファイル名が「ON-AIR-DATA1.pdf」の時、<fileData>フィールドの 1 行目には下記のコードが記述される。

HEX 表記：4F 4E 2D 41 49 52 2D 44 41 54 41 31 2E 70 64 66 0D 0A

3.3.5　添付ファイルの読み出し

添付ファイルの読み出しは file chunk の<fileData>フィールドを別ファイルとして読み出す方法で行う。ただし、file chunk に格納されているファイルはそのまま残すものとする。

<fileData>フィールドの 1 行目には添付ファイル名がテキスト形式で格納されているので、これを読み出したファイル名として使用する（1 行目は改行コード<CR/LF>で終了している）。改行コード以降のデータが、添付ファイルの本体（データ）となる。

（注）添付ファイルの拡張子は 3.3.1 項に記述している 4 種類であるが、本規格第 1 版及び第 2 版では添付ファイルを編集し再格納する場合に、拡張子を 000～999 に変更して使用することが可能となっていた。したがって、旧規格に準拠した機器で作成した WAVE ファイルの中には、拡張子として 000～999 を使用している場合がある。旧規格との互換性を確保するために、<fileData>フィールド 1 行目のファイル名が、拡張子 000～999 の場合も別ファイルとして読み出すこと。読み出したファイルの拡張子は 000～999 をそのまま用いる。

3.3.6　添付ファイルの削除

添付ファイルの削除は BC$NOTE ラベルの削除（cue-point chunk、labl chunk、file chunk の削除）で行う。

3.3.7　添付ファイルの編集と再格納

読み出した添付ファイルを編集し再度格納する場合、同じ BC$NOTE ラベルは使用できない。したがって、未使用の BC$NOTE ラベルを使って格納する。同じ BC$NOTE ラベルを使用する場合は、そのラベルを一度削除（旧添付ファイルの削除）してから改めて使用する。

解説 A　拡張 fmt chunk について

A.1　拡張 fmt chunk

RIFF 規格の PCM オーディオファイル（WAVE、BWF、BWF-J など）の fmt chunk フィールドは 16 バイト構成であるが、中には 40 バイトに拡張されたものがあり、読み込み時にトラブルを生じることがある。この拡張された fmt chunk は、4G バイトを超える音声ファイルを扱えるようにした“Long-form file format”規格で使用されているもので、EBU の RF64 や ITU-R の BW64 がある。

RF64 及び BW64 は、マルチチャンネルオーディオなどの長時間録音を可能にするため、data chunk のアドレスを 64 ビットに拡張している。BWF 及び BWF-J と同じ拡張子“.wav”を使用しているが、RIFF 規格の WAVE ファイルとは互換性がない。RIFF 規格の WAVE ファイルと互換性を保つために、4G バイト以下のファイルでは 32 ビットのアドレスを使用しているが、fmt chunk のフィールド構成が拡張されているのでトラブルになることがある。

A.2　拡張 fmt chunk への対応

トラブルを避けるにはファイルを読み込む側で対応する必要がある。拡張 fmt chunk のフィールド構成は、先頭から 16 バイトまでが RIFF 規格 WAVE ファイルの fmt chunk と同じである。したがって、以下の 2 つの対策が考えられる。機器の運用上最適と思われる方法を選択する。

1）fmt chunk フィールドが 40 バイトで構成されていてもエラーとせず、ファイルを読み込んだ後に調整する。チャンクフィールドは先頭から 16 バイトを有効とし、残りの 24 バイトは無視する。書き出し時もチャンクフィールドは変更しない。

【長所】オリジナル音声ファイルの fmt chunk を変更しない。

【短所】他の RIFF 規格機器でも同じようなトラブルを生じる可能性がある。

2）fmt chunk フィールドが 40 バイトで構成されていてもエラーとせず、ファイルを読み込んだ後に調整する。チャンクフィールドは先頭から 16 バイトを有効とし、残りは破棄する。書き出し時にチャンクフィールドを RIFF 規格に合わせて変更する。

【長所】他の RIFF 規格機器で同じようなトラブルを生じない。

【短所】オリジナル音声ファイルの fmt chunk 拡張データが削除される。

A.3　拡張 fmt chunk の構造

RF64 と BW64 の fmt chunk 仕様を参考として示す。“cbSize = 0x0000”の場合、“extra Data”に有効なデータがないことを示す。

1）EBU RF64 仕様の fmt チャンク（EBU Tech3306）

```
char chunkId[4];                    // 'fmt '
unsigned int32 chunkSize;           // 4 byte size of the 'fmt ' chunk
unsigned int16 formatType;          // WAVE_FORMAT_PCM = 0x0001, etc.
unsigned int16 channelCount;        // 1 = mono, 2 = stereo, etc.
unsigned int32 sampleRate;          // 32000, 44100, 48000, etc.
unsigned int32 bytesPerSecond;      // only important for compressed formats
```

```
unsigned int16 blockAlignment;      // container size (in bytes) of one set of samples
unsigned int16 bitsPerSample;       // valid bits per sample 16, 20 or 24
unsigned int16 cbSize;              // extra information (after cbSize) to store
char extraData[22];                 // extra data of WAVE_FORMAT_EXTENSIBLE when necessary.
```

2）ITU-R BW64 仕様の fmt チャンク（REC ITU-R BS.2088-0）

```
CHAR chunkId[4];                    // ‘fmt ’
DWORD chunkSize;                    // 4 byte size of the ‘fmt ’ chunk
WORD formatTag;                     // WAVE_FORMAT_PCM = 0x0001, etc.
WORD channelCount;                  // 1 = mono, 2 = stereo, etc.
DWORD sampleRate;                   // 32000, 44100, 48000, etc.
DWORD bytesPerSecond;               // only important for compressed formats
WORD blockAlignment;                // container size (in bytes) of one set of samples
WORD bitsPerSample;                 // valid bits per sample 16, 20 or 24
WORD cbSize;                        // extra information (after cbSize) to store should be set to
                                       zero as extraData is not used
CHAR extraData[22];                 // extra data of WAVE_FORMAT_EXTENSIBLE when necessary,
                                       should not be used as cbSize will be zero.
```

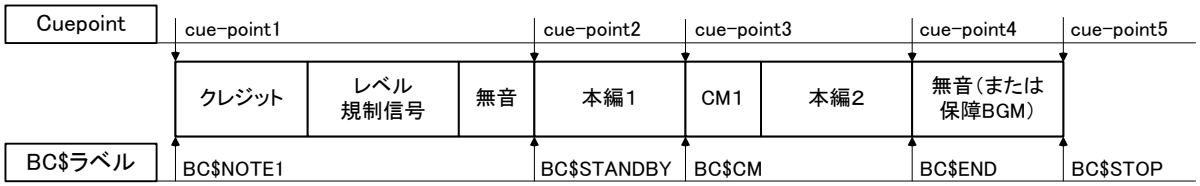
解説 B　BWF-J ファイルの推奨構造

B.1　BWF-J ファイルの推奨ファイル構造

BWF の基になる RIFF 規格書の WAVE ファイルでは、fmt と wave data チャンクの位置関係以外、特にチャンクの記述順序を規定していない。EBU の BWF サンプルファイルでも“bext / fmt / wave-data”という並びと“fmt / wave-data / bext”という並びが混在している。RIFF 規格書に基づき設計されたソフトウェアでは、fmt / wave-data チャンク以外の位置関係がどのようになっていても問題にならないはずであるが、より互換性を確保するために本規格で推奨するファイル構造を示す。

B.2　チャンクの構成順序

例として図－B1 のような BWF-J ファイルの場合のチャンク構造（チャンクの順序）を示す。



図－B1　BWF-J ファイルの構造例

```
RIFF
    'WAVE'              ; WAVE ファイル宣言
    fmt                 ; fmt チャンク
    bext                ; bext チャンク
    wave-data           ; 音声データ本体
    cue                 ; cue-point チャンク
        [cue-point1]    ; dwName で任意の cue-point 番号を指定
        [cue-point2]    ; dwName で任意の cue-point 番号を指定
        [cue-point3]    ; dwName で任意の cue-point 番号を指定
        [cue-point4]    ; dwName で任意の cue-point 番号を指定
        [cue-point5]    ; dwName で任意の cue-point 番号を指定
    plst                    ; playlist チャンク
        [play-segment1]     ; dwName で cue-point2(BC$STANDY)と関連付けを行う
        [play-segment2]     ; dwName で cue-point3(BC$CM)と関連付けを行う
        [play-segment3]     ; dwName で cue-point4(BC$END)と関連付けを行う
        [play-segment4]     ; dwName で cue-point5(BC$STOP)と関連付けを行う
    LIST
        adtl        ; labl チャンク、file チャンク
            [labl1] ; data で BC$ラベルを定義(BC$NOTE1)、dwName で cue-point1 と関連付け
            [labl2] ; data で BC$ラベルを定義(BC$STANDBY)、dwName で cue-point2 と関連付け
```

[labl3] ; data で BC$ラベルを定義(BC$CM)、dwName で cue-point3 と関連付け

[labl3] ; data で BC$ラベルを定義(BC$END)、dwName で cue-point4 と関連付け

[labl4] ; data で BC$ラベルを定義(BC$STOP)、dwName で cue-point5 と関連付け

[file1] ; fileData に添付ファイルを格納、dwName で cue-point1 と関連付け

解説 C　CodingHistory フィールドの使用例

C.1　はじめに

本規格では bext chunk の<CodingHistory>フィールドの記述方法について特に規定していない。これは BWF でも同じであるが、同規格では推奨記述方法として EBU R98-1999 がある。ここでは、R98 で述べている記述方法について解説する。詳細は同規格を参照されたい。

C.2　シンタックス

EBU R98-1999 ではシンタックスとして 5 個のパラメータが決められている（下表）。BWF の場合、coding algorithm として MPEG 圧縮が許可されているが、本規格では PCM と ANALOGUE のみ使用可能である。パラメータ Bit-rate は PCM では使用しない。

Parameter	Variable string <allowed option>	Unit
Coding Algorithm	A=<ANALOGUE, PCM, MPEG1L1, MPEG1L2, MPEG1L3, MPEG2L1, MPEG2L2, MPEG2L3>	
Sampling frequency	F=<11000,22050,24000,32000,44100,48000>	[Hz]
Bit-rate	B=<any bit-rate allowed in MPEG 2 (ISO/IEC 13818-3)>	[kbit/s per channel]
Word Length	W=<8, 12, 14, 16, 18, 20, 22, 24>	[bits]
Mode	M=<mono, stereo, dual-mono, joint-stereo>	
Text, free string	T=<a free ASCII-text string for in house use. This string should contain no commas (ASCII $2C_{hex}$). Examples of the contents: ID-No; codec type; A/D type>	

Notes:

The variable strings should be separated by commas (ASCII $2C_{hex}$). Each row should be terminated by CR/LF

Variable B= is only used for MPEG coding.

Variable W= For MPEG coding, should be used to indicate the word-length of the PCM input to the MPEG coder.

表－C1　パラメータの規定（EBU R98-1999 より抜粋）

C.3　使用例

以下に使用例を示す。T については必要な情報を記述すればよい。例 2 のように、複数行の情報を記述する場合は、行の区切りとして<CR/LF>コードを使用する。

例 1）PCM デジタル収録を行った場合。

A=PCM,F=48000,W=24,M=stereo,T=original;AAA;XXX;SN01234567,<CR/LF>

意味：PCM 録音、ステレオ、サンプリング周波数 48kHz、量子化ビット数 24bit、オリジナル録音で AAA 社のモデル XXX（シリアル No.01234567）を使用。

例 2）アナログ音源から 3 種類の機器を使用してデジタル変換録音した場合。

A=analogue,M=stereo,T=BBB;555;SN01234;38;RMG_SM468,<CR/LF>

A=PCM,F=48000,W=24,M=stereo,T=CCC;YYY;SN98765;A/D,<CR/LF>

A=PCM,F=48000,W=16,M=stereo,T=DDD;ZZZ;SN000999,<CR/LF>

意味：1 行目はアナログ機器とテープに関する記述で、アナログ録音、ステレオ、BBB 社のモデル 555（シリアル No.01234）使用し、再生速度 38cm、テープは RMG 社の SM468 を使用。

2 行目は A/D 変換機器に関する記述で、PCM 変換、サンプリング周波数 48kHz、量子化ビット数 24bit、ステレオ、CCC 社のモデル YYY（シリアル No.98765）を使用して A/D 変換。

3 行目はデジタル録音機器に関する記述で、PCM 録音、サンプリング周波数 48kHz、量子化ビット数 16bit、ステレオ、DDD 社のモデル ZZZ（シリアル No.000999）を使用。

例 2 の場合は、A/D 変換は 24bit であるが、デジタル録音は 16bit で行われたことを示している。

解説 D　用語の説明

Associated data chunk

Cue-points chunk で指定された各 cue-point に対して、関連する情報を付け加えるためのチャンク。例えば、labl chunk を使用すると各 cue-point に対して、より分かりやすい名前を付けることが出来る。また、各 cue-point に対して、関連するテキスト情報や別形式のファイルを付け加えることが出来る。BWF-J ではこの chunk の中の“labl”と“file”が使用できる。

BWF (Broadcast Wave Format)

EBU で策定した放送用途のための WAVE ファイルフォーマット。マイクロソフトと IBM が策定した Resource Interchange File Format (RIFF)の中の Waveform Audio File Format(WAVE)を基に bext chunk の追加など機能拡張を図っている。

BWF-J (Broadcast Wave Format - Japan)

日本ポストプロダクション協会（JPPA）内に設置された BWF-J ワーキング・グループが EBU の BWF を基に日本の放送局、舞台などでの運用に必要な機能拡張を行ったファイルフォーマット。BWF とは上位互換性を持っている。機能拡張は RIFF のチャンクを使用し、ラベルに独自の意味づけを行い実現している。BWF-J 独自のチャンクは持っていない。

Chunk(チャンク)

WAVE ファイルを構成する基本単位で、ファイルは複数のチャンクが順番につながった構造となっている。チャンクはある特定の機能（これをチャンクタイプと言う）を持っており、音声データは data chunk (wave-data) に記録され、音声データの形式は fmt chunk に記述されている。放送用途に必要な基本メタデータは bext chunk に記述されている。WAVE ファイルはチャンクを追加することで機能の拡張を図ることが可能な構造になっている。

BWF-J ファイルは、BWF ファイルに cue-points chunk をはじめとして、幾つかのチャンクを追加し機能の拡張を行っている。チャンクの基本構造は、先頭から順に、チャンクタイプを識別する ASCII コード 4 文字（FOURCC）の“ckID”と、チャンクデータ本体のサイズを示す 32 ビットの“ckSize”が書かれ、それ以降にチャンクデータ本体が格納されている。

Cue-points chunk

Wave-data に対し任意の位置に cue-point を設定するチャンク。本規格では“dwName”フィールドを cue-point 番号として使用している。

File chunk

WAVE ファイルに他の形式のファイルを追加（格納）する。本規格では cue-points chunk、labl chunk と共に用い、CSV/PDF/TEXT/XML 形式で記述したファイルを WAVE ファイルの中に格納する時に使用する（別ファイルの添付）。

Fmt chunk

Fmt chunk (Format chunk) はオーディオデータの信号形式（サンプリング周波数、量子化ビット数、ステレオ／モノラルなど）が記述されているチャンクである。PC 用の再生・編集ソフトの中には、fmt チャンクが WAVE ファイルの先頭になければ読み込みエラーとなることがある。

Labl chunk

<dwName>で措定されている cue-point に、より分かりやすい名前を付けるためのチャンク。本規格では BC$で始まるラベルを予約ラベル名として定義し、特別な意味を持たせている。BC$ラベルの使い方は RIFF の仕様と異なる。

Playlist chunk

WAVE ファイルの再生制御を行うチャンク。本規格では WAVE ファイルに playlist chunk が存在すると、記述されている play-segment を時間軸順（<dwSampleOffset>値の小さい順）に再生する。BC$ラベルは cue-point の設定だけでは実行されず、playlist chunk に記述することで実行可能となる。本規格では BC$ラベルの付加された cue-point の再生回数（<dwLoops>）は 1 と判断し、BC$ラベルで定義された動作を継続する。本来の playlist chunk は、記述してある play-segment 順に再生動作が行われる。

RIFF (Resource Interchange File Format)

IBM と Microsoft が 1991 年に策定した OS/2 と Windows（2.X,3.X）の為のマルチメディアファイル交換インターフェース仕様。この中に WAVE ファイルに関する仕様も記述されている。1994 年に追加仕様が発行されている（参考文献 [5],[6] ）。

WAV ファイル

“WAVE ファイル”のこと。拡張子が“.wav”であることからこのように呼ぶことが多い。Windows の標準音声ファイルであるが、Mac や Linux などの汎用 OS の他、DAW でも取り扱うことができる汎用性の高い音声ファイルである。WAVE の項を参照。WAVE (Waveform Audio File Format)

RIFF 規格書の中に記述されている音声（音楽）のためのフォーマット。このフォーマットをもとに BWF や本規格が策定されている。Windows の標準音声ファイルとなっており、拡張子は“.wav”を使用する。このフォーマットで作成したファイルを“WAVE ファイル”と呼ぶ。

WAVE ファイルは 32 ビットアドレスのため、4G バイトを超える音声を扱うことができない。近年、アドレスを 64 ビットに拡張し 4G バイトを超える音声を扱えるようにした、RF64 及び BW64 などの“Long-form file format”も登場している。これらの音声ファイルも拡張子“.wav”を使用しているが、RIFF 規格の WAVE ファイルと互換性はない。

Wave-data

WAVE ファイルの音声データ部分で単一の wave データ、または複数の wave データを格納できる。また、無音状態を長さで登録することも出来る（Silence chunk）。本規格では単一 WAVE データのリニア PCM のみを扱う。Wave-data の仕様（サンプリング周波数、量子化ビット数、モノラル／ステレオなどの情報）は fmt chunk に記述されている。

解説 E　Bext chunk のバージョン番号

BWF-J が準拠している EBU の BWF（Tech-3285）には 3 つのバージョンがある。バージョン番号は bext chunk の Version フィールドに記述されており、BWF-J はこの中のバージョン 1 に準拠している。ここでは 3 つのバージョンの違いについて説明する。バージョン 1、バージョン 2 とも下位互換性を保っており、各バージョンで定義されているフィールドデータを読み込めば、支障は生じない。

E.1　バージョン 0

Tech3285-1997 で定義されている bext chunk のバージョンで、これがすべてのバージョンの基本となっている。

E.2　バージョン 1

Tech3285-2001 で定義されている bext chunk のバージョンで、BWF-J はこれに準拠している。バージョン 0 では 254 バイトの Reserved フィールドを持っていたが、この中の 64 バイトを使用して UMID フィールドを設けたのがバージョン 1 である。これにより Reserved フィールドは 190 バイトに変更されている。

E.3　バージョン 2

Tech3285-2011 で定義されている bext chunk のバージョンである。バージョン 2 では 190 バイトの Reserved フィールドの中から 10 バイトを使用して、ラウドネス関係の情報を 5 つ追加している。Reserved フィールドは 180 バイトに変更されている。

Version フィールドから CodingHistory までの構造は以下のようになっている。

```
WORD  Version;                /* Version of the BWF; unsigned binary number */
BYTE  UMID_0                  /* Binary byte 0 of SMPTE UMID */
  ....
BYTE  UMID_63                 /* Binary byte 63 of SMPTE UMID */
INT   LoudnessValue;          /* 16 bit signed Integer, Loudness Value of the file in LKFS (multiplied
                                 by 100) */
INT   LoudnessRange;          /* 16 bit signed Integer, Loudness Range of the file in LU (multiplied
                                 by 100) */
INT   MaxTruePeakLevel;       /* 16 bit signed Integer, Maximum True Peak Level of the file
                                 expressed as dBTP (multiplied by 100) */
INT   MaxMomentaryLoudness;   /* 16 bit signed Integer, Highest value of the Momentary Loudness
                                 Level of the file in LKFS (multiplied by 100) */
INT   MaxShortTermLoudness;   /* 16 bit signed Integer, Highest value of the Short-Term Loudness
                                 Level of the file in LKFS (multiplied by 100) */
BYTE  Reserved[180];          /* 180 bytes, reserved for future use, set to “NULL” */
CHAR  CodingHistory[];
```

一般社団法人 日本ポストプロダクション協会
〒160-0014　東京都新宿区内藤町三洋ビル
電話　03-3355-6420
FAX　03-3355-6421
URL　http://www.jppanet.or.jp