STANDARDS AND INFORMATION DOCUMENTS

AES31-2-2019
revision of AES31-2-2012

STANDARDS

AES standard on digital audio – File format for transferring digital audio data between systems of different type and manufacture

AUDIO ENGINEERING SOCIETY, INC.
60 East 42nd Street, New York, New York 10165, US.

AUDIO ES STANDARDS

AES31-2-2019
Revision of AES31-2-2012

AES standard on digital audio – File format for transferring digital audio data between systems of different type and manufacture

Published by
Audio Engineering Society, Inc.
Copyright © 2012, 2019 by the Audio Engineering Society

Abstract

The Broadcast Wave Format is a file format for audio data. It can be used for the seamless exchange of audio material between (i) different broadcast environments and (ii) equipment based on different computer platforms.

As well as the audio data, a BWF file (BWFF) contains the minimum information - or metadata - that is considered necessary for all broadcast applications. The Broadcast Wave Format is based on the Microsoft WAVE audio file format. This specification adds a “Broadcast Audio Extension” chunk to the basic WAVE format.

An optional Extended Broadcast Wave Format (BWF-E) file format is designed to be a compatible extension of the Broadcast Wave Format (BWF) for audio file sizes larger than a conventional Wave file. It extends the maximum size capabilities of the RIFF/WAVE format by increasing its address space to 64 bits where necessary. A set of machine-readable loudness metadata is included.

This revision includes a new annex I to describe a universal 'ubxt' chunk to carry human-readable information in UTF-8 multi-byte characters to support international character sets. This is compatible with EBU v2 broadcast wave files.

Audio Engineering Society Inc. 551 Fifth Avenue, Room 1225, New York, NY 10176, US.
www.aes.org/standards   standards@aes.org.

2019-5-22 printing

Contents

0 Introduction **4**
0.1 General 4
0.2 Data types 4

1 Scope **5**

2 Normative references **5**

3 Definitions and abbreviations **5**

4 BWF file **6**
4.1 Existing Chunks defined as part of the RIFF Format 6
4.2 Additional chunks 6
4.3 Contents of a BWFF 6
4.4 Broadcast audio extension chunk 7

Annex A (normative) RIFF Wave file format **10**
A.1 Waveform audio file format (Wave) 10
A.2 PCM format 11
A.3 Storage of Wave data 13

Annex B (normative) Chunk order **14**

Annex C (normative) Filename conventions **15**
C.1 General 15
C.2 File-name length 15
C.3 File-name extension 15
C.4 File-name character set 15

Annex D (informative) Multi-channel usage **16**
D.1 General 16
D.2 Multi-channel audio data packing 16
D.3 Channel assignments in multi-channel files 17

Annex E (informative) Other audio codings **18**
E.1 General 18
E.2 MPEG files 18

Annex F (normative) Extended file format (BWF-E) **19**
F.1 Introduction 19
F.2 Exceeding the 4-gigabyte limit 19
F.3 Compatibility between BWF and BWF-E 20
F.4 RIFF/WAVE and RF64/WAVE structures 22

Annex G (normative) BWFF versions **24**
G.0 Version 0 24
G.1 Version 1 24
G.2 Version 2 24

Annex H (normative) Loudness parameters **25**
H.1 Treatment of Loudness Parameters 25
H.2 Loudness parameter references 26
H.3 Loudness informative references 26

Annex I (normative) **Universal broadcast audio extension chunk** **27**
I.1 General 27
I.2 Contents of a BWFF with ‘ubxt’ chunk 27
I.3 Universal broadcast audio extension chunk 27

Annex J Bibliography **31**

Foreword
[This foreword is not part of the AES31-2 *Standard on network and file transfer of audio – Audio-file transfer and exchange – File format for transferring digital audio data between systems of different type and manufacture*.]

This document was produced by a writing group of the SC-02-08 Working Group on Audio-File Transfer and Exchange of the SC-02 Subcommittee on Digital Audio under project AES-X066. Contributors to this project included: D. Ackerman, S. Aoki, I. Beynon, D. Brenan, J. Bull, R. Chalmers, B. Devlin, J. Emmett, A. Faust, C. Garza, Y. Grabit, U. Henry, A. Holzinger, P. Jessop, H. Nakashima, M. Yonge, J. Yoshio.

M. Yonge, chair
B. Harris, vice-chair

SC-02-08 Working Group on Audio-File Transfer and Exchange
2006-03-25

Foreword to 2012 edition

This revision incorporates AES31-2 Am.1 2008, *Amendment 1 to AES31-2 - Audio-file transfer and exchange - Part 2: File format for transferring digital audio data between systems of different type and manufacture - Extended file format for audio to exceed 4 GByte* as annex F.

This revision also introduces a means to carry loudness metadata related to the audio content. These files are identified as version 2 and are both forwards and backwards compatible with version 1 files and implementations.

M. Yonge, chair SC-02-08 Working Group on Audio-File Transfer and Exchange
2012-11-19

Foreword to 2019 edition

This revision includes a new annex I to describe a universal 'ubxt' chunk to carry human-readable information in UTF-8 multi-byte characters to support international character sets. The 'ubxt' chunk is always in addition to the basic 'bext' chunk and never used on its own. The document also specifies requirements for implementation, and priority of data when the machine-readable elements in the 'bext' and 'ubxt' chunks are not identical. This revision is intended to be compatible with IEC 62942, currently in development.

This revision replaces and supersedes AES31-2-2012.

M. Yonge, chair SC-02-08 Working Group on Audio-File Transfer and Exchange
2019-02-23

AES standard on digital audio – File format for transferring digital audio data between systems of different type and manufacture

0 Introduction

0.1 General

The broadcast-wave-format file (BWFF) is based on the Microsoft Wave audio file format, which is a type of file specified in the Microsoft resource interchange file format (RIFF). Wave files specifically contain audio data. The basic building block of a RIFF file is a chunk which contains specific information, an identification field, and a size field. A RIFF file contains a number of chunks.

The BWFF specifically includes a <Broadcast Audio Extension> chunk to carry certain metadata important for broadcast and professional use. For reliable interchange, some restrictions apply to the format of the audio data.

The Broadcast Wave Format was first developed using ASCII text for all fields. Later, as the format was further developed, it was proposed to use multi-byte characters to internationalize the format. It was understood that to use multi-byte character sets within the existing format would cause compatibility issues when multi-byte metadata was parsed by applications expecting ASCII text. The separate nature of human-readable and machine-readable metadata was established, and a new "universal" chunk was established to carry internationalized human-readable metadata using multi-byte character sets without interoperability issues. This is described in annex I.

This document contains the specification of the broadcast audio extension chunk and its use with PCM-coded audio data. Basic information on the RIFF format and how it can be extended to other types of audio data is given in annex A and annex F. Details of the PCM Wave format are also given in annex A.

0.2 Data types

The following mnemonics describe the data types used throughout this document. Multi-byte data types are little-endian:

Data Type	Meaning	Equiv. C type
CHAR	8-bit signed integer, representing integer values from –128 to +127	signed char
BYTE	8 bit unsigned integer, representing integer values from 0 to 255	unsigned char
INT	16-bit signed integer, representing integer values from –32768 to +32767	signed short int
WORD	16-bit unsigned integer, representing integer values from 0 to +65535	unsigned short int
LONG	32-bit signed integer, representing integer values from –2,147,483,648 to +2,147,483,647	signed long int
DWORD	32-bit unsigned integer, representing integer values from 0 to +4,294,967,295	unsigned long int

1 Scope

This standard defines a file format for interchanging audio data between compliant equipment. It is primarily intended for audio applications in professional recording, production, post production, and archiving.

It is derived from the EBU Broadcast Wave Format but is also compatible with variant specifications including ITU-R BR.1352-2-2002 and the Japan Post Production Association's BWF-J.

An optional extended format, BWF-E, supports 64-bit addressing to permit file sizes greater than 4 GBytes. Provision is made to support international character sets for human-readable information.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

SMPTE 330M-2000; *SMPTE standard for television - Unique Material Identifier (UMID),* Society of Motion Picture and Television Engineers, White Plains, NY., US.

ISO/IEC 646:1991; *Information technology - ISO-7-bit coded character set for information interchange.* International Standards Organisation, Geneva, Switzerland

ISO 8601; *Data elements and interchange formats - Information interchange - Representation of dates and times* International Standards Organisation, Geneva, Switzerland

ISO/IEC 10646:2012; Information technology - Universal Multiple-Octet Coded Character Set (UCS)

IETF RFC 3629; UTF-8, a transformation format of ISO 10646

3 Definitions and abbreviations

For the purposes of this document, the following terms and definitions apply.

3.1
RIFF
resource interchange file format, a file representation upon which the Wave file format is based

3.2
chunk
data package within RIFF files containing related data

3.3
ASCII
7-bit character code compliant with ISO/IEC 646

3.4
Wave
Audio file format based on the RIFF file structure

3.5
EBU
European Broadcasting Union

3.6
Broadcast Wave Format File
BWFF
Wave file containing the **bext** chunk as described in this standard

3.7
bext
broadcast extension chunk

3.8
SMPTE
Society of Motion Picture and Television Engineers

3.9
UMID
unique material identifier as defined in SMPTE 330M

3.10
Broadcast Wave Format, Extended
BWF-E
an optional extended format that replaces a RIFF header with an RF64 header to support 64-bit addressing to permit file sizes greater than 4 GBytes

3.11
RF64
a structure equivalent to the RIFF file type supporting 64-bit addressing.

4 BWF file

4.1 Existing Chunks defined as part of the RIFF Format

This specification uses a number of RIFF chunks which are already defined (See annex A). These are:

`<fmt-ck>`	Format Chunk
`<wave-data>`	Audio data chunk

4.2 Additional chunks

Additional chunks may be present in the file. Some of these may be outside the scope of this standard. Applications may or may not interpret or make use of these chunks, so the integrity of the data contained in such unknown chunks can not be guaranteed. However, compliant applications should pass on unknown chunks with their contents unchanged.

4.3 Contents of a BWFF

A BWFF shall contain the RIFF “Wave” header and at least the following chunks:

```
<WAVE-form>
RIFF(‘WAVE’
     <fmt-ck>                       /* Format of the audio signal: PCM/MPEG */
     <broadcast_audio_extension>    /* information on the audio sequence */
     <wave-data> )                  /* sound data */
```

4.4 Broadcast audio extension chunk

Extra parameters needed for exchange of material between broadcasters are added in a specific Broadcast Audio Extension, or `"bext"` chunk. The structure of the `bext` chunk shall be defined as follows:

```
typedef struct chunk_header {
     DWORD ckID;              /* (broadcastextension)ckID=bext */
     DWORD ckSize;            /* size of extension chunk */
     BYTE ckData[ckSize];     /* data of the chunk */
}    CHUNK_HEADER;
typedef struct broadcast_audio_extension {
CHAR Description[256];          /* ASCII : "Description of the sound sequence" */
CHAR Originator[32];            /* ASCII : "Name of the originator" */
CHAR OriginatorReference[32];   /* ASCII : "Reference of the originator" */
CHAR OriginationDate[10];       /* ASCII : "yyyy:mm:dd" */
CHAR OriginationTime[8];        /* ASCII : "hh:mm:ss" */
DWORD TimeReferenceLow;         /* First sample count since midnight, low word */
DWORD TimeReferenceHigh;        /* First sample count since midnight, high word */
WORD Version;                   /* Version of the BWF; unsigned binary number. See
                                annex G */
BYTE UMID_0;                    /* Binary byte 0 of SMPTE UMID */
....
BYTE UMID_63;                   /* Binary byte 63 of SMPTE UMID */
WORD LoudnessValue;             /* WORD: Integrated Loudness Value of the file in
                                LUFS (multiplied by 100) see annex H */
WORD LoudnessRange;             /* WORD : Loudness Range of the file in LU
                                (multiplied by 100) see annex H */
WORD MaxTruePeakLevel;          /* WORD: Maximum True Peak Level of the file
                                expressed as dBTP (multiplied by 100) see annex H */
WORD MaxMomentaryLoudness;      /* WORD: Highest value of the Momentary Loudness
                                Level of the file in LUFS (multiplied by 100) see
                                annex H */
WORD MaxShortTermLoudness;      /* WORD: Highest value of the Short-Term Loudness
                                Level of the file in LUFS (multiplied by 100) see
                                annex H */
BYTE Reserved[180];             /* 180 bytes, reserved for future use, set to "NULL"
                                */
CHAR CodingHistory[];           /* ASCII : « History coding » */
} BROADCAST_EXT
```

The content of the fields in the `bext` chunk shall be defined as shown in table 1. Note that in applications where ASCII text is inappropriate for human-readable information - for example when a character set other than ISO 646 is required - it is necessary to carry it by another means, for example, in a dedicated metadata chunk added to the BWFF. See also Annex I, Universal broadcast audio extension chunk (`ubxt`).

All the items except `Description`, `Originator`, `OriginatorReference` and `CodingHistory` should have the same content as that of each corresponding item of the `ubxt` chunk (see annex I), if present. If machine-readable data in the `bext` chunk is updated, the corresponding machine-readable data in the `ubxt` chunk should also be updated identically.

Table 1: `bext` field content definitions

`Description`	Human	ASCII string, 256 characters or less, containing a description of the sequence. If line breaks are used, lines shall be terminated by <CR><LF>. If data is not available or if the length of the string is less than 256 characters, the first unused character shall be a null character (`00`$_{16}$). If line breaks are used, lines shall be terminated by <CR><LF>. To help applications that only display a short description, a summary of the description should be contained in the first 64 characters. The last 192 characters may be used for details.
`Originator`	Human	ASCII string, 32 characters or less, containing the name of the originator of the audio file. If data is not available or if the length of the string is less than 32 characters, the first unused character shall be a null character (`00`$_{16}$).
`OriginatorReference`	Human	ASCII string, 32 characters or less, containing a reference allocated by the originating organization. See references J.11 and J.6. If data is not available or if the length of the string is less than 32 characters, the first unused character shall be a null character (`00`$_{16}$).
`OriginationDate`	Human	ASCII string, 10 characters, containing the date of creation of the audio sequence. The date shall be represented as the year, month, and day of the Gregorian calendar. If data is unavailable, the default value shall be the origin of the modified Julian date (MJD), namely, 1858-11-17. **Format: *CCYY*-*MM*-*DD*** ***CCYY*** = 4 characters for century and year shall contain a value between 0000 and 9999 **-** = 1 character ***MM*** = 2 characters for month shall contain a value between 01 and 12 **-** = 1 character ***DD*** = 2 characters for day of month shall contain a value between 01 and 31 All components shall be present. Hyphen characters, “`-`“, shall be used as separators within the date expression in compliance with ISO 8601. For compatibility with alternative implementations, reproducing equipment should also recognise the following separator characters: “_” underscore “`:`” colon “ “ space “`.`” period.
`OriginationTime`	Human	ASCII string, 8 characters, containing the time of creation of the audio sequence in hours, minutes and seconds. If data is unavailable, the default value shall be 00:00:00 . **Format: *hh*:*mm*:*ss*** ***hh*** = 2 characters for hours shall contain a value between 00 and 23 if time given **:** = 1 character ***mm*** = 2 characters for minutes shall contain a value 00 - 59 if time given **:** = 1 character ***ss*** = 2 characters for seconds shall contain a value between 00 and 59 All components shall be present. Colon characters, “`:`”, shall be used as separators within the time of

		day expression in compliance with ISO 8601. For compatibility with alternative implementations, reproducing equipment should also recognise the following characters: “_” underscore “-” hyphen “ “ space “.” period.
`TimeReference`	Machine	This field shall contain the sample address count [time code] of the sequence. It is a 64-bit unsigned value which contains the sample count since midnight of the first sample in the audio data. The number of samples per second depends on the sample frequency which is defined in the field <**`nSamplesPerSec`**> from the <**`fmt-ck`**>. The default value is zero, corresponding to midnight.
`Version`	Machine	An unsigned binary number indicating the version of the BWF. For Version 1 it shall be set to 0001_{16} and for Version 2 it shall be set to 0002_{16}. See annex G
`UMID`	Machine	64 bytes containing an extended UMID to SMPTE 330M. If a 32-byte basic UMID is used, the last 32 bytes shall be filled with zeros. If no UMID is available, the 64 bytes shall be filled with zeros. NOTE the length of the UMID is coded at the head of the UMID itself
`LoudnessValue`	Machine	A 16-bit signed integer, representing the Integrated Loudness Value of the file in LUFS. See annex H
`LoudnessRange`	Machine	A 16-bit signed integer, representing the Loudness Range of the file in LU. See annex H
`MaxTruePeakLevel`	Machine	A 16-bit signed integer, representing the Maximum True Peak Value of the file in dBTP. See annex H
`MaxMomentaryLoudness`	Machine	A 16-bit signed integer, representing the highest value of the Momentary Loudness Level of the file in LUFS. See annex H
`MaxShortTermLoudness`	Machine	A 16-bit signed integer, representing the highest value of the Short-term Loudness Level of the file in LUFS. See annex H
`Reserved`	Machine	180 bytes reserved for extension. These 180 bytes shall be set to zero.
`CodingHistory`	Human	A variable-size block of ASCII characters comprising 0 or more strings each terminated by <**`CR`**><**`LF`**>. The first unused character shall be a null character (00_{16}). Each string shall contain a description of a coding process applied to the audio data. Each new coding application should add a new string with the appropriate information. See Bibliography, item10

Annex A (normative) RIFF Wave file format

A.1 Waveform audio file format (Wave)

A.1.1 General

The information in this annex follows the specification documents of the Microsoft RIFF file format.

A RIFF file shall be identified by the four ASCII characters **"RIFF"** as the first four octets in the file. The next four octets shall indicate the file length, in octets.

Data contained in a RIFF file is organised into chunks. The first four octets of each chunk shall represent a code to identify that chunk. The next four octets shall indicate the chunk length, in octets. Playback systems shall identify each chunk and ignore any unknown chunks encountered.

A.1.2 Required Wave chunks

A format chunk <**fmt-ck**> shall always occur before the <wave-data>. Both of these chunks shall be present in a Wave file.

```
<WAVE-form> ->
        RIFF ( "WAVE"
                 <fmt-ck>                         /* Format chunk */
                 <wave-data> )                    /* Wave data */
```

The Wave chunks are described in the following sections.

A.1.3 Wave format chunk

The Wave format chunk <**fmt-ck**> specifies the format of the <**wave-data**>.

The <**fmt-ck**> shall be defined as follows:

```
<fmt-ck> ->      fmt( <common-fields>
                 <format-specific-fields> )
<common-fields> ->
         struct{
                 WORD wFormatTag;                 /* Format category */
                 WORD nChannels;                  /* Number of channels */
                 DWORD nSamplesPerSec;            /* Sampling frequency */
                 DWORD nAvgBytesPerSec;           /* For buffer estimation */
                 WORD nBlockAlign;                /* Data block size */
         }
```

The content of the fields in the <**common-fields**> portion of the chunk shall be defined as shown in table A.1.

Table A.1: Format chunk - common fields

Field	Description
`wFormatTag`	A number indicating the Wave format category of the file. The content of the <**`format-specific-fields`**> portion of the format chunk and the interpretation of the waveform data depend on this value.
`nchannels`	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo.
`nSamplesPerSec`	The sampling frequency, in samples per second, at which each channel should be reproduced.
`nAvgBytesPerSec`	The average number of bytes per second at which the waveform data should be transferred. Playback software can estimate the buffer size using this value.
`nBlockAlign`	The block alignment in bytes of the waveform data. Playback software needs to process a multiple of <**`nBlockAlign`**> bytes of data at a time, so the value of <**`nBlockAlign`**> can be used for buffer alignment.

The <**`format-specific-fields`**> shall comprise zero or more bytes of parameters. Which parameters occur depends on the Wave format category. Playback software should allow for (and ignore) any unknown <format-specific-fields> parameters that occur at the end of this field.

A.1.4 Wave format categories

The format category of a Wave file shall be specified by the value of the <**`wFormatTag`**> field of the format chunk. The representation of data in <**`wave-data`**>, and the content of the <**`format-specific-fields`**> of the format chunk, shall depend on the format category.

Currently defined open non-proprietary Wave format categories are shown in table A.2.

Table A.2: Wave format categories

wFormatTag	Value	Format category
`WAVE_FORMAT_PCM`	0001_{16}	Pulse code modulation (PCM) format

NOTE Although other Wave formats exist, only the above formats are at present used with the BWFF. Details of the PCM Wave format are provided in A.2. General information on other Wave formats is given in annex E.

A.2 PCM format

If the <**`wFormatTag`**> field of the <**`fmt-ck`**> is set to WAVE_FORMAT_PCM, then the waveform data shall consist of samples represented in PCM format. For PCM waveform data, the <**`format-specific-fields`**> shall be defined as follows:

```
<PCM-format-specific> ->
        struct{
                WORD nBitsPerSample;           /* Sample size */
        }
```

The <**`nBitsPerSample`**> field shall specify the number of bits of data used to represent each audio sample of each channel. If there are multiple channels, the sample size shall be the same for each channel.

The <**`nBlockAlign`**> field should be equal to the following formula, rounded to the next whole number:

$$\texttt{nChannels} \times \texttt{BytesPerSample}$$

The value of **`BytesPerSample`** shall be calculated by rounding up **`nBitsPerSample`** to the next whole byte. Where the audio sample word is less than an integer number of bytes, the most significant bits of the audio sample shall be placed in the most significant bits of the data word; the unused data bits adjacent to the least-significant bits shall be set to zero.

For PCM data, the <**`nAvgBytesPerSec`**> field of the format chunk shall be equal to the following formula:

$$\texttt{nSamplesPerSec} \times \texttt{nBlockAlign}$$

NOTE: the original Wave specification permits, for example, 20-bit samples from two channels packed into 5 bytes - sharing a single byte for the least significant bits of the two channels. This document specifies a whole number of bytes per audio sample in order to reduce ambiguity in implementations and to achieve maximum interchange compatibility.

A.2.1 Data packing for PCM WAVE files

In a single-channel Wave file, samples shall be stored consecutively.

For stereo Wave files, channel 0 shall represent the left channel, and channel 1 shall represent the right channel. Tables A.3a and A.3b show examples of data packing for 16-bit mono and stereo Wave files:

Table A.3a: Data packing for 16-bit mono PCM

Sample 1		**Sample 2**	
Channel 0 low-order byte	Channel 0 high-order byte	Channel 0 low-order byte	Channel 0 high-order byte

Table A.3b: Data packing for 16-bit stereo PCM

Sample 1			
Channel 0 (left) low-order byte	Channel 0 (left) high-order byte	Channel 1 (right) low-order byte	Channel 1 (right) high-order byte

In multiple-channel Wave files, samples shall be interleaved in channel sequence (see annex D for examples).

A.2.2 Data format of the samples

Each sample is contained in an integer ***i***. The size of ***i*** is the smallest number of bytes required to contain the specified sample size. The least significant byte shall be stored first. The bits that represent the sample amplitude are stored in the most significant bits of ***i***, and the remaining bits shall be set to zero.

For example, if the sample size recorded in <**`nBitsPerSample`**> is 20 bits, then each sample is stored in a three-byte integer. The least significant four bits of the first (least significant) byte is set to zero. The data format and maximum and minimum values for PCM waveform samples of various sizes are shown in table A.4. A corresponding example of 16-bit PCM data values is shown in table A.5.

Table A.4: PCM data format

Sample size	Data format	Maximum value	Minimum value
> 8 bits	Signed integer i	Largest positive value of ***i***	Most negative value of ***i***

Table A.5: PCM data format - 16-bit

Format	Maximum value	Minimum value	Midpoint value
16-bit PCM	32767 (`7FFF`$_{16}$)	-32768 (`-8000`$_{16}$)	0

NOTE legacy word sizes of 8 bits and lower typically used a different scheme

A.2.3 Examples of PCM Wave files

Examples of Format-Chunk settings for four cases are shown in table A.6. The cases are:

A PCM Wave file with 48 kHz sampling rate, mono, 16 bits per sample

B PCM Wave file with 44,1 kHz sampling rate, stereo, 16 bits per sample

C PCM Wave file with 96 kHz sampling rate, stereo, 24 bits per sample

D PCM Wave file with 48 kHz sampling rate, stereo, 20 bits per sample

Table A.6: PCM Wave format chunk examples

	Field	A	B	C	D
Common fields	`wFormatTag`	1	1	1	1
	`nchannels`	1	2	2	2
	`nSamplesPerSec`	48000	44100	96000	48000
	`nAvgBytesPerSec`	96000	176400	576000	288000
	`nBlockAlign`	2	4	6	6
PCM-format-specific	`nBitsPerSample`	16	16	24	20

A.3 Storage of Wave data

The <**`wave-data`**> CHUNK contains the waveform data as little-endian integer values. It shall be defined as follows:

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

Annex B (normative) Chunk order

According to RIFF rules, chunks may be encountered in any order in the file. In the specific case of a Wave file, the <**`fmt-ck`**> shall always be before the <**`data-ck`**>. Other chunks can be in any order. For example, some applications may find it convenient to write metadata chunks after the audio data chunk instead of before it.

Although a particular application may have internal conventions regarding the order in which it expects to write chunks, it should be able to replay files from other systems with chunks in any different order.

NOTE 1 the extended format described in annex F requires the "**`ds64`**" chunk, or its placeholder "**`JUNK`**" chunk, to be the first chunk in the file as a special exception.

NOTE 2 the "**`JUNK`**" chunk is not required for a compliant BWF file, but would be included by applications that wished to accommodate a file size exceeding 4 GBytes.

Annex C (normative) Filename conventions

C.1 General

The general interchange of audio files mean that they must be playable on computer and operating-system types that may be quite different from the originating system. An inappropriate filename could mean that the file cannot be recognised by the destination system. For example, some computer operating systems limit the number of characters in a file name. Others are unable to accommodate multi-byte characters. Some characters have special significance in certain operating systems and should be avoided.

These guidelines are intended to identify best practice for general international interchange.

C.2 File-name length

BWFF file names should not exceed 31 characters, including the file-name extension.

C.3 File-name extension

BWF files shall use the same four-character file-name extension, "**`.wav`**", as a conventional Wave file. This allows the audio content to be played on most computers without additional software. Practical implementations should also accept other extensions, such as "**`.bwf`**", that may have been used in error.

C.4 File-name character set

File names for international interchange should use only ASCII (ISO/IEC 646) 7-bit characters in the range 32 to 126 (decimal).

Character	Decimal value	Hexadecimal value
(Space)	32	20_{16}
...	...	...
~ (tilda)	126	$7E_{16}$

Additionally, the following characters are reserved for special functions on certain file systems and should not be used in file names:

Character	Decimal value	Hexadecimal value
"	34	22_{16}
*	42	$2A_{16}$
/	47	$2F_{16}$
:	58	$3A_{16}$
<	60	$3C_{16}$
>	62	$3E_{16}$
?	63	$3F_{16}$
\	92	$5C_{16}$
|	124	$7C_{16}$

Additionally, the following characters should not be used for the first or last character in a file name:

Character	Decimal value	Hexadecimal value
(Space)	32	20_{16}
. (period)	46	$2E_{16}$

Annex D (informative) Multi-channel usage

D.1 General

Editing systems typically need access to mono files in order to be able to edit between them flexibly. However, some location recorders operate more efficiently when recording multiple channels to a single multi-channel file. Accordingly, many editing application are able to convert multi-channel source recordings to a coherent set of mono files.

D.2 Multi-channel audio data packing

Files with multiple audio channels are constructed by a simple extension of the existing format. The Channel field in the <**`fmt-ck`**> is set to the number of channels and the audio data is packed sequentially, as shown in the following examples:

D.2.1: Data packing for 24-bit mono PCM audio data

Sample 1			Sample 2		
Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte

D.2.2: Data packing for 16-bit stereo (2-channel) PCM audio data

Sample 1				Sample 2			
Channel 1 (Left)		Channel 2 (Right)		Channel 1 (Left)		Channel 2 (Right)	
Low-order byte	High-order byte	Low-order byte	High-order byte	Low-order byte	High-order byte	Low-order byte	High-order byte

D.2.3: Data packing for 24-bit, 4-channel PCM audio data

Sample 1												Sample 2											
Channel 1			Channel 2			Channel 3			Channel 4			Channel 1			Channel 2			Channel 3			Channel 4		
Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte	Low-order byte	Mid-order byte	High-order byte

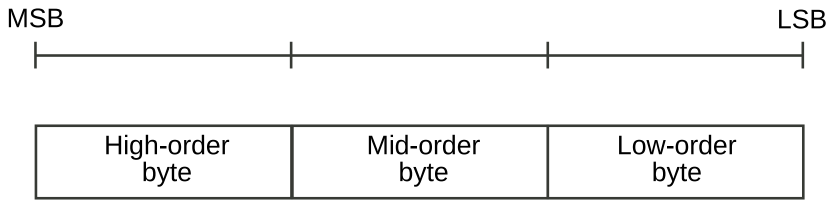


Figure D.1: 24-bit sample packing

D.3 Channel assignments in multi-channel files

Channel usage is not considered in this standard. There are two main application areas.

D.3.1 Distribution and archive

Stereo or surround-sound recordings for archiving or distribution of intermediate elements or finished masters may use the audio channels according to one of a number of predetermined schemes. Examples include:

ITU-R BR.1384-2; *Parameters for international exchange of multi-channel sound recordings with or without accompanying picture*

EBU Technical Recommendation R91-1998; *Track allocations and recording levels for the exchange of multichannel audio signals*

SMPTE S320M-1999; *Channel Assignments and Levels on Multichannel Audio Media* (TV)

SMPTE S323M-2004; *Channel Assignments and Levels on Multichannel Audio Media* (Film)

D.3.2 Production recordings

Production recordings may use the channels in many different ways, described in traditional recording report or in a suitable form of metadata associated with the audio recording. The set of audio channels may accommodate related recordings in mono, LR stereo, MS stereo, or surround sets.

Annex E (informative) Other audio codings

E.1 General

All non-PCM Wave types contain both a <**`fact-ck`**> fact chunk and an extended wave format description within the <**`fmt-ck`**> format chunk.

E.2 MPEG files

Details of MPEG Wave format are given in EBU Tech 3285_S1-1997, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 1 – MPEG audio* (see J.5)

Annex F (normative) Extended file format (BWF-E)

F.1 Introduction

The 32-bit address space of a Wave file limits its maximum size to 4 GB. Some practical computer systems may impose a lower limit of 2 GB. This is not a significant obstacle for mono files at basic rate sampling frequencies, but the limitation becomes increasingly significant as the number of channels in the file is increased or when double- or quadruple-rate sampling frequencies are used.

It is possible to concatenate multiple BWF files using the EBU "**`link`**" chunk. This technique is described in reference 8 of the Bibliography: EBU T3285-S4-2003 *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 4: <link> chunk.*

The extended file format described below is intended to fulfill the longer-term need for single file sizes greater than 4 GByte. It extends the maximum size capabilities of the RIFF/WAVE format by increasing its address space to 64 bits where necessary. The required effort for software implementers is small.

The Broadcast Wave Format, Extended (BWF-E) file format is designed to be a compatible extension of the Broadcast Wave Format (BWF). BWF-E is also designed to be mutually compatible with the EBU T3306 "**`RF64`**" extended format (see Bibliography item 5). Note that EBU T3306 contains additional specifications for channel-mask assignments that are not included in this document.

The extended format is discussed here as an extension to the underlying RIFF/Wave file format. Compliance with this standard will also require the **`bext`** chunk defined in 4.4 and may require other data chunks.

F.2 Exceeding the 4-gigabyte limit

F.2.1 General

The reason for the 4 GByte limit is the 32-bit addressing inherent in the RIFF and WAVE file formats. With 32-bit addressing, a maximum of 4294967296 bytes = 4 GByte can be addressed. See figure F.1.

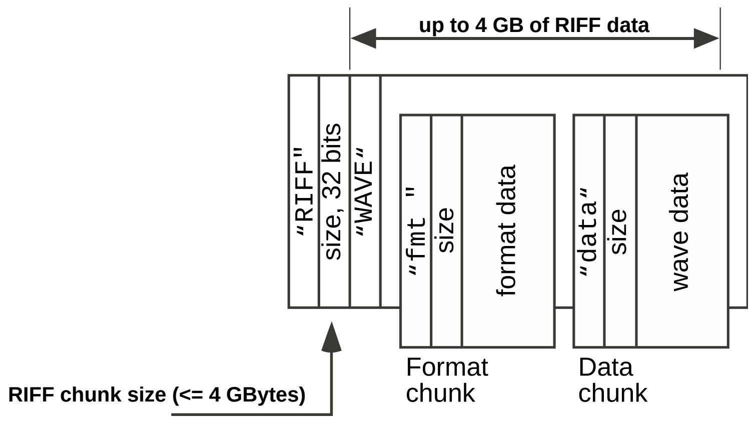


Figure F.1 - Conventional RIFF/WAVE format

To work with larger files, a larger address range is needed. This standard specifies 64-bit addressing. However, simply changing the size of every field in a Wave file to support 64-bit addressing would produce a file that is not compatible with the standard RIFF/WAVE format - an obvious but important observation.

NOTE Additional file-size constraints may be imposed by computer implementations and operating systems

F.2.2 64-bit resource interchange file format (RF64)

This standard defines a 64-bit based Resource Interchange File Format called "RF64". The "RF64" format shall be identical to the RIFF format except for the following changes.

The ID **`RF64`** shall be used instead of '**`RIFF`**' in the first four bytes of the file.

A '**`ds64`**' (data size 64) chunk shall be added. This shall be the first chunk after the **`RF64`** chunk. The **`ds64`** chunk shall contain three 64-bit integer values, which provide substitute values for the equivalent 32-bit fields of the RIFF format:

`riffSize`	shall substitute for the **`RIFF`** size field
`dataSize`	shall substitute for the size field of the **`data`** chunk
`sampleCount`	shall substitute for the sample count value in the **`fact`** chunk

For all three 32-bit fields of the RIFF/WAVE format the following rule shall apply:

If the 32-bit value in the field is not "**`FFFFFFFF`**$_{16}$" then this 32-bit value shall be used.

If the 32-bit value in the field is "**`FFFFFFFF`**$_{16}$" the 64-bit value in the **`ds64`** chunk shall be used instead.

An extended table of **`ChunkSize64`** elements is intended to support additional 64-bit variables. (see G.4.2). Space should be reserved for a minimum of 50 elements.

The complete structure of the "RF64" file format is illustrated in figure 2. See also G.4.2.

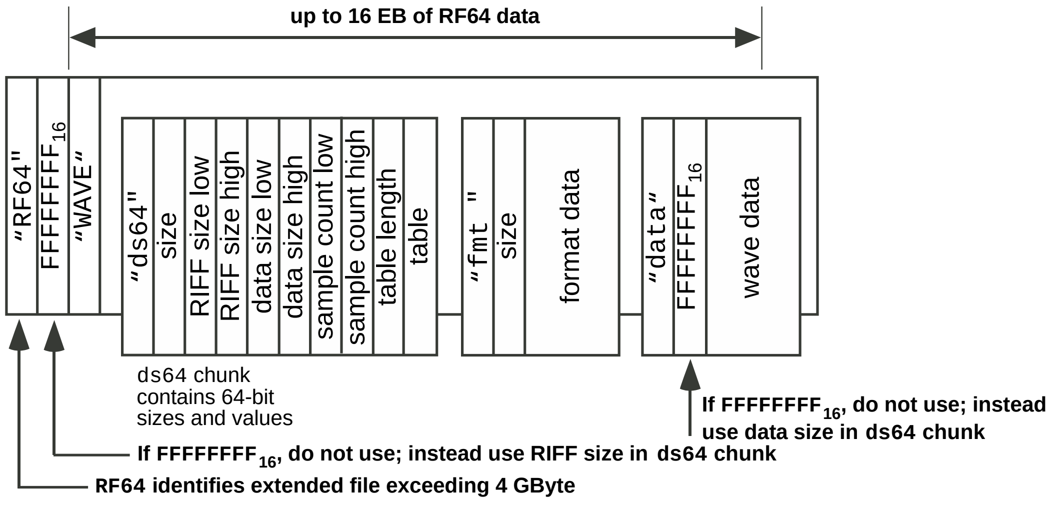

up to 16 EB of RF64 data
"RF64"
FFFFFFFF16
"WAVE"
"ds64"
size
RIFF size low
RIFF size high
data size low
data size high
sample count low
sample count high
table length
table
"fmt "
size
format data
"data"
FFFFFFFF16
wave data
ds64 chunk contains 64-bit sizes and values
If FFFFFFFF16, do not use; instead use data size in ds64 chunk
If FFFFFFFF16, do not use; instead use RIFF size in ds64 chunk
RF64 identifies extended file exceeding 4 GByte

Figure F.2 - Extended RF64/WAVE format

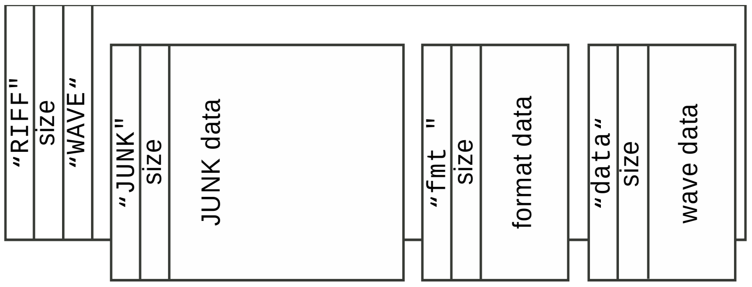
F.3 Compatibility between BWF and BWF-E

F.3.1 General

Many production audio files will be smaller than 4 GByte and they should continue to use the Broadcast Wave Format (BWF).

Because a recording application cannot know in advance whether the audio it is recording will exceed 4 GByte, the recording application must switch from BWF to the extended format (BWF-E) at the 4-GByte file-size limit while the recording continues.

The option to switch from BWF to the extended format is achieved by reserving additional space in the BWF by inserting a '**JUNK**' chunk (see J.1) that is of the same size as a **ds64** chunk (see figure 3). This reserved space has no meaning for the BWF file, but will be used to locate the **ds64** chunk if a transition to BWF-E is necessary.



JUNK chunk as placeholder for 'ds64' chunk in case file size will need to exceed 4 GB

Figure F.3 - Compatible RIFF/WAVE structure

NOTE 1 The **JUNK** chunk is a padding chunk to be used as a placeholder and it will be ignored by any audio application.

NOTE 2 there may be other chunks between the format chunk and the wave data chunk.

F.3.2 Initialisation as BWF

At the beginning of a recording, an operation supporting BWF-E shall create a conventional BWF file with a **JUNK** chunk as the first chunk (see G.3.4.1). While recording, the application shall monitor the RIFF and data sizes.

F.3.3 Transition to BWF-E

If the file size will exceed 4 GByte, the application shall:

- Replace the chunkID '**JUNK**' with '**ds64**'. (This transforms the previous **JUNK** chunk into a **ds64** chunk).
- Insert the RIFF size, **data** chunk size and sample count in the **ds64** chunk
- Set RIFF size, **data** chunk size and sample count in the 32-bit RIFF fields to $\mathbf{FFFFFFFF}_{16}$
- Replace the ID '**RIFF**' with '**RF64**' in the first four bytes of the file
- Continue with the recording.

NOTE Any other chunks that are valid in a RIFF/WAVE file are also valid in a RF64/WAVE file.

F.4 RIFF/WAVE and RF64/WAVE structures

F.4.1 Chunks and structs specific to the RIFF/WAVE format

```
struct RiffChunk          /* declare RiffChunk structure */
{
CHAR chunkId[4];          /* 'RIFF' */
DWORD chunkSize;          /* 4 byte size of the traditional RIFF/WAVE file */
CHAR riffType[4];         /* 'WAVE' */
};
struct JunkChunk          /* declare JunkChunk structure */
{
CHAR chunkId[4];          /* 'JUNK' */
DWORD chunkSize;          /* 4 byte size of the 'JUNK' chunk. This should be 636 to
                          be a place-holder for a 'ds64' chunk with capacity for 50
                          ChunkSize64 table entries */
CHAR chunkData[628*];     /* dummy bytes  */
};
```

NOTE 1 This file declaration and **JUNK** chunk will normally be followed by **fmt**, **bext** and **data** chunks (see Annex A and clause 4).

NOTE 2 The **JUNK** chunk shown here is only necessary if the writing application supports extended file sizes.

NOTE 3 The empty bracket is not standard C/C++ syntax. It is used to show that these arrays have a variable number of elements (which might even be zero).

NOTE 4 [*] Table size will be a multiple of 12 Bytes. A table of size 50 will need 600 bytes.

F.4.2 Chunks and Structs specific to the RF64/WAVE (BWF-E) format

```
struct RF64Chunk             /* declare RF64Chunk structure */
{
CHAR chunkId[4];             /* 'RF64' */
DWORD chunkSize;             /*  0xFFFFFFFF  means  do  not  use  this  data,  use
                             riffSizeHigh and riffSizeLow in 'ds64' chunk instead */
CHAR rf64Type[4];            /* 'WAVE' */
};
struct ChunkSize64           /* declare ChunkSize64 structure */
{
CHAR chunkId[4];             /* chunk ID (i.e. "big1" – this chunk is a big one) */
DWORD chunkSizeLow;          /* low 4 byte chunk size */
DWORD chunkSizeHigh;         /* high 4 byte chunk size */
};
struct DataSize64Chunk       /* declare DataSize64Chunk structure */
{
CHAR chunkId[4];             /* 'ds64' */
DWORD chunkSize;             /* 4 byte size of the 'ds64' chunk */
DWORD riffSizeLow;           /* low 4 byte size of RF64 block */
DWORD riffSizeHigh;          /* high 4 byte size of RF64 block */
DWORD dataSizeLow;           /* low 4 byte size of data chunk */
DWORD dataSizeHigh;          /* high 4 byte size of data chunk */
DWORD sampleCountLow;        /* low 4 byte sample count of fact chunk */
DWORD sampleCountHigh;       /* high 4 byte sample count of fact chunk */
DWORD tableLength;           /* number of valid entries in array "table" */
chunkSize64 table[*];
};
```

NOTE 1 This file declaration and `'ds64'` chunk will normally be followed by `'fmt '`, `'bext'` and `'data'` chunks (see Annex A).

NOTE 2 The array of "`ChunkSize64`" structs is used to store the length of any chunk other than 'data' in the optional part, or 'table' of the `ds64` chunk. Currently, no standard chunk type other than '`data`' is likely to exceed a size of 4 GByte even in extremely large audio files (for example, the EBU '`levl`' chunk will typically exceed 4 GByte only when the 'data' chunk reaches about 512 GByte).

NOTE 3 [*] the size of the table may be more or less than the recommended 50 elements.

Annex G (normative) BWFF versions

G.0 Version 0

Version 0 of the BWF was published by the EBU in 1997. There is no equivalent AES standard.

G.1 Version 1

Version 1 was published in July, 2001 by the EBU as EBU T3285-2001. AES31-2 published in 2006 is directly compatible with this document.

Version 1 differs from Version 0 only in that 64 of the 254 reserved bytes in Version 0 are used to contain a SMPTE UMID and the <Version> field is changed accordingly.

Version 1 is backwards compatible with Version 0. This means that software designed to read Version 0 files will interpret Version 1 files correctly except that it will ignore the UMID field.

The change is also forwards compatible. This means that Version 1 software will be able to read Version 0 files correctly. Ideally, Version 1 software needs to read the <Version> field to determine if a UMID is present. However if the Version number is not read, software will read all zeros in the UMID field in a Version 0 file. This will not be a valid UMID and will be ignored.

G.2 Version 2

Version 2 was published by the EBU as EBU T3285-2011.

It differs from Version 1 only in that 10 of the 190 reserved bytes in Version 1 are used to carry information about the file's loudness and the <Version> field is changed accordingly.

Version 2 is backwards compatible with Versions 1 and 0. This means that software designed to read Version 1 and Version 0 files will interpret the files correctly except that Version 0 software will ignore the UMID and loudness information which may be present and Version 1 software will ignore the loudness information. Therefore, users of such devices will lose metadata unless special precautions are taken. In addition, early BWF-aware devices will be unable to cope with the larger RF64 and MBWF files and may not recognise any of the chunks which have been defined since 2001.

The change is also forwards compatible. This means that Version 2 software will be able to read Version 0 and Version 1 files correctly. Software needs to read the <Version> field to determine if a UMID and loudness metadata are present.

Annex H (normative) Loudness parameters

H.1 Treatment of Loudness Parameters

The loudness parameters specified for carriage in the **`bext`** chunk (see 4.4) are integer representations of floating-point parameters derived elsewhere. A precision of two decimal places is preserved by multiplying the floating-point parameter by 100 before rounding.

The rounding function which shall be used is defined as follows:

integer representation = integer part of (x + sgn(x) · 0.5)

where x is the value to be represented, multiplied by 100

and where sgn() is the signum operator. sgn(x) = -1 if $x < 0$; 0 if $x = 0$; 1 if $x > 0$.

NOTE This rounding method is commonly referred to as “round to nearest, ties away from zero” because where the fractional part of the number is 5 (midway between integers), the rounding is up for positive numbers and down for negative numbers.

EXAMPLES

Negative numbers:

Float value	Calculation	Value carried in **`bext`** (decimal/ hexadecimal)
-22,644	integer[(-22,644 x 100) + sgn(-22,644 x 100) · 0,5]	-2264/ **`F728`**$_{16}$
-22,645	integer[(-22,645 x 100) + sgn(-22,645 x 100) · 0,5]	-2265/ **`F727`**$_{16}$
-22,646	integer[(-22,646 x 100) + sgn(-22,646 x 100) · 0,5]	-2265/ **`F727`**$_{16}$

Positive numbers:

Float value	Calculation	Value carried in **`bext`** (decimal/ hexadecimal)
12,764	integer[(12,764 x 100) + sgn(12,764 x 100) · 0,5]	1276/ **`04FC`**$_{16}$
12,765	integer[(12,765 x 100) + sgn(12,765 x 100) · 0,5]	1277/ **`04FD`**$_{16}$
12,766	integer[(12,766 x 100) + sgn(12,766 x 100) · 0,5]	1277/ **`04FD`**$_{16}$

If any of the loudness parameters are not being used then their 16-bit integer values shall be set to **`7FFF`**$_{16}$, which is a value outside the range of the parameter values.

For **`LoudnessValue`**, **`MaxTruePeakLevel`**, **`MaxMomentaryLoudness`** and **`MaxShortTermLoudness`**, the range of valid values is **`D8F1`**$_{16}$ to **`FFFF`**$_{16}$ (corresponding to the floating-point equivalent values of -99,99 to -0,01) and **`0000`**$_{16}$ to **`270F`**$_{16}$ (0,00 to 99,99). The most significant bit of the 16-bit hexadecimal number is the sign bit; hence, values between **`8000`**$_{16}$ and **`FFFF`**$_{16}$ represent negative numbers.

For **`LoudnessRange`** the range of valid values is **`0000`**$_{16}$ to **`270F`**$_{16}$ (0,00 to 99,99).

When reading the chunk, any parameter values outside their valid ranges shall be ignored.

H.2 Loudness parameter references

The loudness parameters specified for carriage in the `bext` and `ubxt` chunks should be as specified in:

EBU Recommendation R 128, *Loudness normalisation and permitted maximum level of audio signals*, European Broadcasting Union, Geneva, Switzerland

EBU Tech 3341, *Loudness Metering: 'EBU Mode' metering to supplement loudness normalisation in accordance with EBU R 128*, European Broadcasting Union, Geneva, Switzerland

EBU Tech 3342, *Loudness Range: A descriptor to supplement loudness normalisation in accordance with EBU R 128*, European Broadcasting Union, Geneva, Switzerland

H.3 Loudness informative references

ITU-R BS.1770 *Algorithms to measure audio programme loudness and true-peak audio level*, International Telecommuniction Union, Geneva, Switzerland

EBU Tech 3343, *Practical Guidelines for Production and Implementation in accordance with EBU R 128*, European Broadcasting Union, Geneva, Switzerland

EBU Tech 3344, *Practical Guidelines for Distribution of Programmes in accordance with EBU R 128*, European Broadcasting Union, Geneva, Switzerland

Annex I (normative) Universal broadcast audio extension chunk

I.1 General

The **ubxt** chunk is always used as an addition to the **bext** chunk in BWFF.

I.2 Contents of a BWFF with 'ubxt' chunk

A BWFF with **ubxt** shall contain the RIFF "WAVE" header and at least the following chunks:

```
<WAVE-form>
RIFF('WAVE'
     <fmt-ck>                       /* Format of the audio signal: PCM/MPEG */
     <broadcast_audio_extension>    /* information on the audio sequence */
     <universal_broadcast_audio     /* ubxt is required for multibyte language
     extension>                     support only */
     <wave-data> )                  /* sound data */
```

I.3 Universal broadcast audio extension chunk

The information contained in the Broadcast Audio Extension **bext** chunk defined in section 4.4 may additionally be extended by a dedicated chunk called "Universal Broadcast Audio Extension", or "**ubxt**" chunk to express the human-readable information of the **bext** chunk in multi-byte languages. The basic structure of this metadata chunk is the same as that of the **bext** chunk. Four human-readable items, **uDescription**, **uOriginator**, **uOriginatorReference** and **uCodingHistory**, are described in UTF-8 (8-bit UCS Transformation Format) instead of ASCII. The first three items have 8 times the data size of the corresponding items in the **bext** chunk. The structure of the **ubxt** chunk is defined as follows:

```
typedef struct chunk_header {
     DWORD ckID;              /* (universal broadcast extension)ckID=ubxt */
     DWORD ckSize;            /* size of extension chunk */
     BYTE ckData[ckSize];     /* data of the chunk */
}    CHUNK_HEADER;
typedef struct universal_broadcast_audio_extension {
BYTE uDescription[256*8];        /* UTF-8 : "Description of the sound sequence" */
BYTE uOriginator[32*8];          /* UTF-8 : "Name of the originator" */
BYTE                             /* UTF-8 : "Reference of the originator" */
uOriginatorReference[32*8];
CHAR OriginationDate[10];        /* ASCII : "yyyy-mm-dd" */
CHAR OriginationTime[8];         /* ASCII : "hh:mm:ss" */
DWORD TimeReferenceLow;          /* First sample count since midnight, low word */
DWORD TimeReferenceHigh;         /* First sample count since midnight, high word */
WORD Version;                    /* Version of the BWF; unsigned binary number. See
                                 annex F */
BYTE UMID_0;                     /* Binary byte 0 of SMPTE UMID */
....
BYTE UMID_63;                    /* Binary byte 63 of SMPTE UMID */
INT LoudnessValue;               /* Integrated Loudness Value of the file in LKFS
                                 (multiplied by 100) see annex G */
INT LoudnessRange;               /* Loudness Range of the file in LU (multiplied by
                                 100) see annex G */
INT MaxTruePeakLevel;            /* Maximum True Peak Level of the file expressed
                                 as dBTP (multiplied by 100) see annex G */
INT MaxMomentaryLoudness;        /* Highest value of the Momentary Loudness Level
                                 of the file in LKFS (multiplied by 100) see annex
                                 G */
INT MaxShortTermLoudness;        /* Highest value of the Short-Term Loudness Level
                                 of the file in LKFS (multiplied by 100) see annex
                                 G */
BYTE Reserved[180];              /* 180 bytes, reserved for future use, set to
                                 "NULL" */
CHAR uCodingHistory[];           /* UTF-8 : « History coding » */
} UNIV_BROADCAST_EXT
```

The content of the fields in the **ubxt** chunk shall be defined as shown in table I.1.

When a given code value in UTF-8 is out of the subset (as defined in Chapter 12 of ISO/IEC 10646) supported by a piece of processing equipment, the value shall be unchanged and ignored for processing.

All the items except **uDescription**, **uOriginator**, **uOriginatorReference** and **uCodingHistory** shall have the same content as that of each corresponding item of the **bext** chunk (clause 4.4). When data is entered into the human-readable fields of the **ubxt** chunk, the equivalent fields in the **bext** chunk should provide an ASCII equivalent if possible (see note), or a null character. When machine-readable data in the 'ubxt' chunk is updated, the corresponding machine-readable data in the 'bext' chunk shall also be updated identically. If the machine-readable fields of the **bext** and **ubxt** chunks is different for any reason, the content of the **bext** chunk shall be preferred.

Note It is not intended that the operator make an ASCII translation of the original-language UTF-8 string. It is expected that the application will insert an ASCII default string into the 'bext' chunk as an indication that UTF-8 information exists in the 'ubxt' chunk. An example might be, "See data in UBXT chunk".

Table I.1 – ubxt field content definitions

uDescription	Human	UTF-8 string, 2 048 bytes or less, containing a description of the sequence. If line breaks are used, lines shall be terminated by <CR><LF>. If data is not available or if the length of the string is less than 2 048 bytes, the first unused byte shall be a null character (00_{16}).
uOriginator	Human	UTF-8 string, 256 bytes or less, containing the name of the originator of the audio file. If data is not available or if the length of the string is less than 256 bytes, the first unused byte shall be a null character (00_{16}).
uOriginatorReference	Human	UTF-8 string, 256 bytes or less, containing a reference allocated by the originating organization. If data is not available or if the length of the string is less than 256 bytes, the first unused byte shall be a null character (00_{16}).
OriginationDate	Human	10 ASCII characters containing the date of creation of the audio sequence. The format is « “year” - “month” - “day” » with 4 characters for the year and 2 characters per other item. Hyphen characters, “-”, shall be used as separators within the date expression in compliance with ISO 8601. “year” is defined from 0000 to 9999 “month” is define from 01 to 12 “day” is defined from 01 to 31 Some legacy implementations may use “_” underscore, “:” colon, “ ” space, “.” period; reproducing equipment should recognize these separator characters.

`OriginationTime`	Human	8 ASCII characters containing the time of creation of the audio sequence. The format is « “hour” : “minute” : “second” » with 2 characters per item. Colon characters, “:“, shall be used as separators within the time expression in compliance with ISO 8601. If data is unavailable, the default value shall be 00:00:00. “hour” is defined from 00 to 23. “minute” and “second” are defined from 00 to 59. Some legacy implementations may use “_” underscore, “-” hyphen, “ ” space, “.” period; reproducing equipment should recognize these separator characters.
`TimeReference`	Machine	This field shall contain the sample address count [time code] of the sequence. It is a 64-bit unsigned value which contains the sample count since midnight of the first sample in the audio data. The number of samples per second depends on the sample frequency which is defined in the field **`<nSamplesPerSec>`** from the **`<fmt-ck>`**.
`Version`	Machine	An unsigned binary number indicating the version of the BWF. For Version 1 it shall be set to 0001_{16} and for Version 2 it shall be set to 0002_{16}. This is set to $0002_{16.}$
`UMID`	Machine	64 bytes containing an extended UMID to SMPTE ST 330. If a 32-byte basic UMID is used, the last 32 bytes shall be filled with zeros. If no UMID is available, the 64 bytes shall be filled with zeros. NOTE the length of the UMID is coded at the head of the UMID itself.
`LoudnessValue`	Machine	The Integrated Loudness value of the file in LKFS (multiplied by 100). A 16-bit signed integer, being the integer of (100 × the Integrated Loudness value of the file in LKFS) ±0.5. See Annex G for more details on the method of conversion.
`LoudnessRange`	Machine	A 16-bit signed integer, representing the Loudness Range of the file in LU. See annex G.
`MaxTruePeakLevel`	Machine	A 16-bit signed integer, being the integer of (100 × the Maximum True Peak value of the file in dBTP) ±0.5. See Annex G for more details on the method of conversion.
`MaxMomentaryLoudness`	Machine	A 16-bit signed integer, being the integer of (100 × the highest value of the Momentary Loudness Level of the file in LKFS) ±0.5. See Annex G for more details on the method of conversion.
`MaxShortTermLoudness`	Machine	A 16-bit signed integer, being the integer of (100 × the highest value of the Short-term Loudness Level of the file in LKFS) ±0.5. See Annex G for more details on the method of conversion.
`Reserved`	Machine	180 bytes reserved for extension. These 180 bytes shall be set to zero.

`uCodingHistory`	Human	A variable-size block of UTF-8 characters comprising 0 or more strings each terminated by <CR><LF>. The first unused character shall be a null character (00_{16}). Each string shall contain a description of a coding process applied to the audio data. Each new coding application should add a new string with the appropriate information. See EBU R99.

Annex J Bibliography

1 *Microsoft Resource Interchange File Format, RIFF,*
available at http://www.tactilemedia.com/info/MCI_Control_Info.html

2 *Microsoft Windows Multimedia Programmer's Reference*. Redmond, Washington:
Microsoft Press, 1991. ISBN: 1-55615-389-9. Chapter 8 describes the RIFF tagged file structure.

3 **EBU Tech 3285-2001**: *BWF - a format for audio data files in broadcasting. Version 1* European Broadcasting Union, Geneva, Switzerland

4 **EBU Tech 3285-2011**: *BWF - a format for audio data files in broadcasting. Version 2* European Broadcasting Union, Geneva, Switzerland

5 **EBU Tech Document T3306**: 2007-02; *RF64: An extended File Format for Audio*; European Broadcasting Union, Geneva.

6 **ITU-R BR.1352-2-2002**, *File format for the exchange of audio programme materials with metadata on information technology media*. Second edition, first published 1999. European Broadcasting Union, Geneva, Switzerland

7 **EBU Tech 3285_S1-1997**, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 1 – MPEG audio* European Broadcasting Union, Geneva, Switzerland

8 **EBU Tech 3285_S4-2003**, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 4: <link> chunk* European Broadcasting Union, Geneva, Switzerland

9 **EBU Tech 3285_S5-2003**, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 5 <axml> chunk* European Broadcasting Union, Geneva, Switzerland

10 **EBU Recommendation R98-1999** *Format for the <CodingHistory> field in Broadcast Wave Format files* European Broadcasting Union, Geneva, Switzerland

11 **EBU Recommendation R99-1999** *'Unique' Source Identifier (USID) for use in the OriginatorReference field of the Broadcast Wave Format* European Broadcasting Union, Geneva, Switzerland

12 **EBU-N22-1997**, *The Broadcast Wave Format, A format for audio data files in broadcasting* European Broadcasting Union, Geneva, Switzerland

13 **EBU Recommendation R85-1997**. *Use of the Broadcast Wave Format for the exchange of audio data files* European Broadcasting Union, Geneva, Switzerland

14 Japan Post Production Association. *http://www.jppanet.or.jp/*

15 *Sustainability of Digital Formats - Planning for Library of Congress Collections*, National Digital Information Infrastructure and Preservation Program.
http://www.digitalpreservation.gov/formats/fdd/descriptions.shtml

16 **IEC 62942** *File format for professional transfer and exchange of digital audio data*. International Electrotechnical Commission, Geneva, Switzerland (in development)