IEC 62942

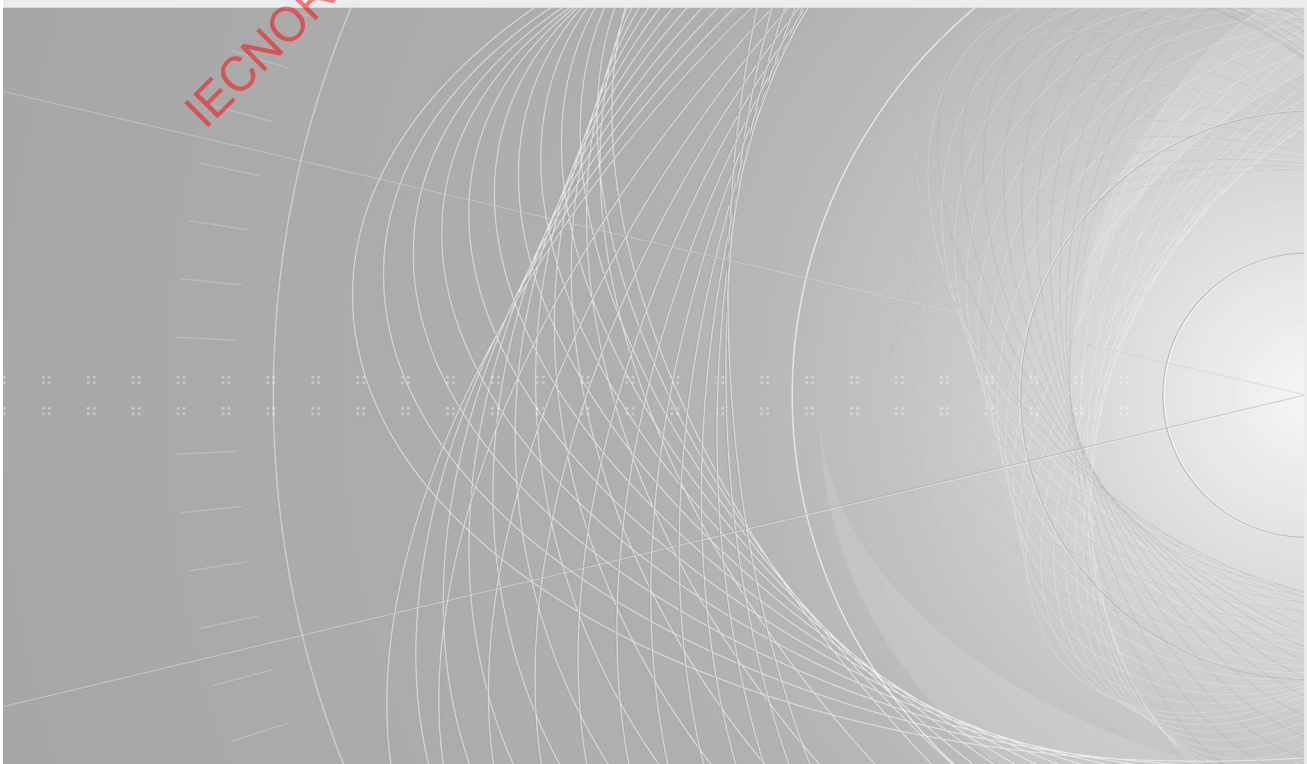
Edition 1.0 2019-12

INTERNATIONAL STANDARD

NORME INTERNATIONALE

File format for professional transfer and exchange of digital audio data

Format de fichier pour le transfert et l'échange professionnels de données audionumériques



IEC 62942:2019-12(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
info@iec.ch
www.iec.ch

IEC 62942

Edition 1.0 2019-12

INTERNATIONAL STANDARD

NORME INTERNATIONALE

File format for professional transfer and exchange of digital audio data

Format de fichier pour le transfert et l'échange professionnels de données audionumériques

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 33.160.30

ISBN 978-2-8322-8696-8

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

FILE FORMAT FOR PROFESSIONAL TRANSFER AND EXCHANGE OF DIGITAL AUDIO DATA

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 62942 has been prepared by technical area 6: Storage media, storage data structures, storage systems and equipment, of IEC technical committee 100: Audio, video and multimedia systems and equipment.

The text of this standard is based on the following documents:

CDV	Report on voting
100/3143/CDV	100/3226/RVC

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

INTRODUCTION

The Broadcast Wave format file (BWFF) is based on the Microsoft WAVE[1] audio file format, which is a type of file specified in the Microsoft resource interchange file format (RIFF) [1][2] 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 K.

---

1 Microsoft® is a registered trademark, and Windows™ is a trademark of Microsoft Corp.. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

2 Numbers in square brackets refer to the Bibliography.

FILE FORMAT FOR PROFESSIONAL TRANSFER AND EXCHANGE OF DIGITAL AUDIO DATA

1 Scope

This document specifies 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 AES31-2 [2] but is also compatible with variant specifications including EBU Tech 3285 [3] to [10], ITU-R BR.1352-3-2007 [11] to [14], and the Japan Post Production Association's BWF-J [15].

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 E. Details of the PCM WAVE format are also given in Annex A.

An optional extended format, BWF-E, supports 64-bit addressing to permit file sizes greater than 4 GB.

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.

ISO/IEC 10646:2017, *Information technology – Universal Coded Character Set (UCS)*

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

SMPTE ST 330-2011, *SMPTE standard for television – Unique Material Identifier (UMID)*

INTERNET ENGINEERING TASK FORCE (IETF). RFC 3629: *UTF-8, a transformation format of ISO 10646* [online]. Edited by F. Yergeau. November 2003 [viewed 2019-11-26]. Available at https://www.rfc-editor.org/rfc/rfc3629.txt

3 Terms and definitions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

3.1
resource interchange file format
RIFF
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
waveform audio file format
WAVE
audio file format based on the RIFF file structure

3.5
Broadcast Wave format file
BWFF
WAVE file containing the bext chunk as described in this document

3.6
broadcast extension chunk
bext
extension chunk to WAVE

3.7
universal broadcast audio extension chunk
ubxt
human-readable information of the bext chunk in multi-byte languages

3.8
UMID
unique material identifier as defined in SMPTE ST 330

3.9
Broadcast Wave format, extended
BWF-E
optional extended format that replaces a RIFF header with an RF64 header to support 64-bit addressing to permit file sizes greater than 4 GB

3.10
RF64
structure equivalent to the RIFF file type supporting 64-bit addressing

3.11
CHAR
8-bit signed integer, representing integer values from −128 to +127

Note 1 to entry: Equivalent C type: "signed char".

3.12
BYTE
8-bit unsigned integer, representing integer values from 0 to 255

Note 1 to entry: Equivalent C type: "unsigned char".

3.13
INT
16-bit signed integer, representing integer values from −32 768 to +32 767

Note 1 to entry: Equivalent C type: "signed short int".

Note 2 to entry: Multi-byte data types are little-endian.

3.14
WORD
16-bit unsigned integer, representing integer values from 0 to +65 535

Note 1 to entry: Equivalent C type: "unsigned short int".

Note 2 to entry: Multi-byte data types are little-endian.

3.15
LONG
32-bit signed integer, representing integer values from −2 147 483 648 to +2 147 483 647

Note 1 to entry: Equivalent C type: "signed long int".

Note 2 to entry: Multi-byte data types are little-endian.

3.16
DWORD
32-bit unsigned integer, representing integer values from 0 to +4 294 967 295

Note 1 to entry: Equivalent C type: "unsigned long int".

Note 2 to entry: Multi-byte data types are little-endian.

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 can be present in the file. Some of these can be outside the scope of this document. Applications may or may not interpret or make use of these chunks, so the integrity of the data contained in such unknown chunks cannot be guaranteed. However, compliant applications should pass on unknown chunks with their contents unchanged (but see also Annex B).

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 */
INT LoudnessValue;                  /* Integrated Loudness Value of the file in
                                    LKFS (multiplied by 100) see Annex H */
INT LoudnessRange;                  /* Loudness Range of the file in LU
                                    (multiplied by 100), see Annex H */
INT MaxTruePeakLevel;               /* Maximum True Peak Level of the file
                                    expressed as dBTP (multiplied by 100), see
                                    Annex H */
INT MaxMomentaryLoudness;           /* Highest value of the Momentary Loudness
                                    Level of the file in LKFS (multiplied by
                                    100), see Annex H */
INT MaxShortTermLoudness;           /* Highest value of the Short-Term Loudness
                                    Level of the file in LKFS (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 K.

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 K), 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 fewer, 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}). 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 fewer, 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 fewer, containing a reference allocated by the originating organization. See Annex I. 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. Format: ***yyyy-mm-dd*** ***yyyy*** = 4 characters for 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 between 00 and 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}. This is set to 0002_{16}. See Annex G.
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 H 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 H.
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 H 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 H 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 H for more details on the method of conversion.
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 Annex J.

4.5 Filename

A BWF file should be saved with a filename that can be interchanged with the widest range of different computer types. See Annex C.

4.6 Channel usage

Audio files are typically mono (1-channel) or stereo (2-channel). For multi-channel usage in Broadcast Wave format files, see Annex D.

4.7 File size

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. For larger file sizes, Annex F describes an extended file format, BWF-E.

Annex A
(normative)

RIFF WAVE file format

A.1 General

The information in this annex follows the specification documents of the Microsoft RIFF file format. See [1].

A.2 Resource Interchange File Format (RIFF)

A.2.1 General

The Resource Interchange File Format (RIFF) is a tagged file structure developed for use on multimedia platforms. This clause describes a "chunk" and "RIFF form".

A.2.2 Chunks

The basic building block of a RIFF file is called a "chunk". Using C syntax, a chunk can be defined as follows:

```
typedef    unsigned long DWORD;
typedef    unsigned char BYTE;
typedef    DWORD FOURCC;           /* Four-character code */
typedef    FOURCC CKID;            /* Four-character-code chunk identifier */
typedef    DWORD CKSIZE;           /* 32-bit unsigned size value */
typedef    struct {                /* Chunk structure */
           CKID ckID;              /* Chunk type identifier */
           CKSIZE    ckSize;       /* Chunk size field (size of ckData) */
           BYTE ckData[ckSize];    /* Chunk data */
           } CK;
```

A "FOURCC" is represented as a sequence of one to four ASCII alphanumeric characters, padded on the right with blank characters (ASCII character value 32) as required, with no embedded blanks.

For example, the four-character code (FOURCC) "FOO " is stored as a sequence of four bytes: "F", "O", "O", " " in ascending addresses. For quick comparisons, a four-character code may also be treated as a 32-bit number.

The three parts of the chunk are described in Table A.1.

Table A.1 – Chunk description

Part	Description
ckID	A four-character code that identifies the representation of the chunk data data. A program reading a RIFF file can skip over any chunk whose chunk ID it doesn't recognize; it simply skips the number of bytes specified by ckSize plus the pad byte, if present.
ckSize	A 32-bit unsigned value identifying the size of ckData. This size value does not include the size of the ckID or ckSize fields or the pad byte at the end of ckData.
ckData	Binary data of fixed or variable size. The start of ckData is word-aligned with respect to the start of the RIFF file. If the chunk size is an odd number of bytes, a pad byte with value zero is written after ckData. The ckSize value does not include the pad byte.

We can represent a chunk with the following notation (in this example, the ckSize and pad byte are implicit):

```
<ckID>      ( <ckData> )
```

A RIFF chunk may contain nested chunks, or subchunks.

All other chunk types store a single element of binary data in <ckData>.

A.2.3 RIFF forms

A RIFF form is a chunk with a "RIFF" chunk ID. The term also refers to a file format that follows the RIFF framework. A "WAVE" is registered to a "RIFF Form Type".

Using the notation for representing a chunk, a RIFF form looks like the following:

```
RIFF    ( <formType>   <ck>... )
```

The first four bytes of a RIFF form make up a chunk ID with values "R", "I", "F", "F". The ckSize field is required, but for simplicity it is omitted from the notation.

The first DWORD of chunk data in the "RIFF" chunk (shown above as <formType>) is a four-character code value identifying the data representation, or *form type*, of the file. Following the form-type code is a series of subchunks. Which subchunks are present depends on the form type.

The definition of a particular RIFF form typically includes the following:

- a unique four-character code identifying the form type;
- a list of required chunks;
- a list of optional chunks;
- possibly, a required order for the chunks.

A.3 Waveform audio file format (WAVE)

A.3.1 General

The WAVE form is defined as follows. Programs shall expect (and shall ignore) any unknown chunks encountered, as with all RIFF forms (but see Annex B). However, <fmt-ck> shall always occur before <wave-data>, and both of these chunks are required in a WAVE file.

```
<WAVE-form> ->
       RIFF ( "WAVE"
              <fmt-ck>                    /* Format chunk */
              [<fact-ck>]                 /* Fact chunk */
              [cue-ck]                    /* Cue points */
              [<playlist-ck>]             /* playlist */
              [<assoc-data-list>]         /* Associated data list */
              <wave-data> )               /* Wave data */
```

The WAVE chunks are described in A.3.2.

NOTE <fact-ck>, <cue-ck>, <playlist-ck> and <assoc-data-list> are not used in this specification.

A.3.2 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.2.

Table A.2 – 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.3.3 WAVE format categories

A.3.3.1 General

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.3.

Table A.3 – 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 Clause A.2. General information on other WAVE formats is given in Annex F.

A.3.3.2 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:

```
nChannels x 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:

```
nSamplesPerSec x 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.3.3.3 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. Table A.4 and Table A.5 show examples of data packing for 16-bit mono and stereo WAVE files.

Table A.4 – 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.5 – 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 E for examples).

A.3.3.4 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.6. A corresponding example of 16-bit PCM data values is shown in Table A.7.

Table A.6 – 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.7 – PCM data format – 16-bit

Format	Maximum value	Minimum value	Midpoint value
16-bit PCM	32 767 (7FFF16)	−32 768 (800016)	0

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

A.3.3.5 Examples of PCM WAVE files

Examples of format chunk settings for four cases are shown in Table A.8.

Table A.8 – PCM WAVE format chunk examples

	Field	A	B	C	D
Common fields	**wFormatTag**	1	1	1	1
	nchannels	1	2	2	2
	nSamplesPerSec	48 000	44 100	96 000	48 000
	nAvgBytesPerSec	96 000	176 400	576 000	288 000
	nBlockAlign	2	4	6	6
PCM-format-specific	**nBitsPerSample**	16	16	24	20

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

A.4 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 can 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 places the "ds64" chunk, or its placeholder "JUNK" chunk, as the first chunk in the file as a special exception.

NOTE 2 The "JUNK" chunk is not needed for a compliant BWF file, but would be included by applications that wished to accommodate a file size exceeding 4 GB.

Annex C
(normative)

Filename conventions

C.1 General

The general interchange of audio files mean that they shall be playable on computers and operating-system types that can 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 could have been used in error.

C.4 File-name character set

File names for international interchange should use only ASCII (ISO/IEC 646 [33]) 7-bit characters in the range 32 to 126 (decimal), as shown in Table C.1.

Table C.1 – Permitted file-name characters

Character	Decimal value	Hexadecimal value
(Space)	32	2016
…	…	…
~ (tilda)	126	7E16

Additionally, the characters listed in Table C.2 are reserved for special functions on certain file systems and should not be used in file names:

Table C.2 – Non-permitted file-name characters

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	$7C_{16}$

Furthermore, the characters listed in Table C.3 should not be used for the first or last character in a file name.

Table C.3 – Non-permitted file-name terminators

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 applications are able to convert multi-channel source recordings into 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.

Audio data in a 24-bit mono file shall be packed as shown in Figure D.1.

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

IEC

Figure D.1 – Data packing for 24-bit mono PCM audio data

Audio data in a 16-bit stereo file shall be packed as shown in Figure D.2.

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

IEC

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

Audio data in a 24-bit, 4-channel file shall be packed as shown in Figure D.3.

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

IEC

Figure D.3 – Data packing for 24-bit, 4-channel PCM audio data

The data bytes in a 24-bit file shall be arranged as shown in Figure D.4.

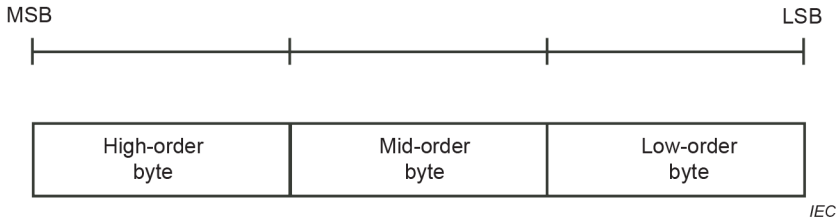


Figure D.4 – 24-bit sample packing

D.3 Channel assignments in multi-channel files

D.3.1 General

Channel usage is not considered in this document. There are two main application areas, stated in D.3.2 and D.3.3.

D.3.2 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 [17], EBU R91-2004 [18], SMPTE ST 2035-2009 [19], SMPTE ST 2035-2009 Amendment 1:2012 [20], and SMPTE ST 323-2004 [21].

D.3.3 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 ITU-R BR.1352-3 Annex 2, and Appendix 1 to Annex 2 [14], and EBU T3285 – Supplement 1 [6].

Annex F
(normative)

Extended file format (BWF-E)

F.1 General

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 EBU T3285-S4-2003 [7].

The extended file format described below is intended to fulfil the longer-term need for single file sizes greater than 4 GB. 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 [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 document will also require the "bext" chunk defined in 4.4 and can require other data chunks.

F.2 Exceeding the 4-GB limit

F.2.1 General

The reason for the 4-GB limit is the 32-bit addressing inherent in the RIFF and WAVE file formats. With 32-bit addressing, a maximum of 4 294 967 296 bytes = 4 GB can be addressed (see Figure F.1).

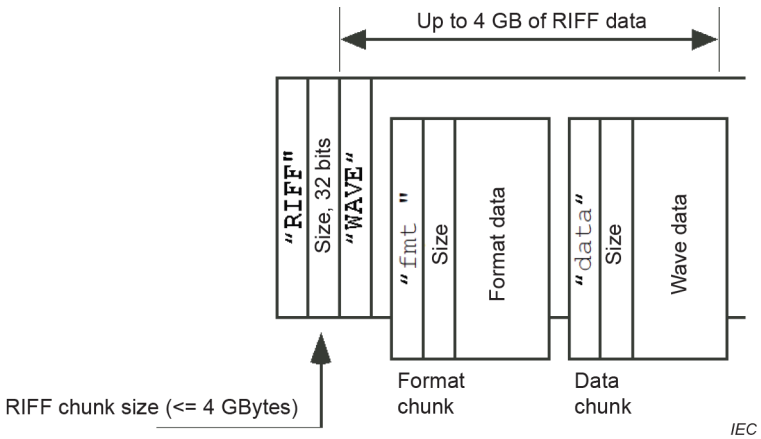


Figure F.1 – Conventional RIFF/WAVE format

To work with larger files, a larger address range is needed. This document 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 can be imposed by computer implementations and operating systems.

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

This subclause 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 F.4.2). Space should be reserved for a minimum of 50 elements.

The complete structure of the RF64 file format is illustrated in Figure F.2. See also F.4.2.

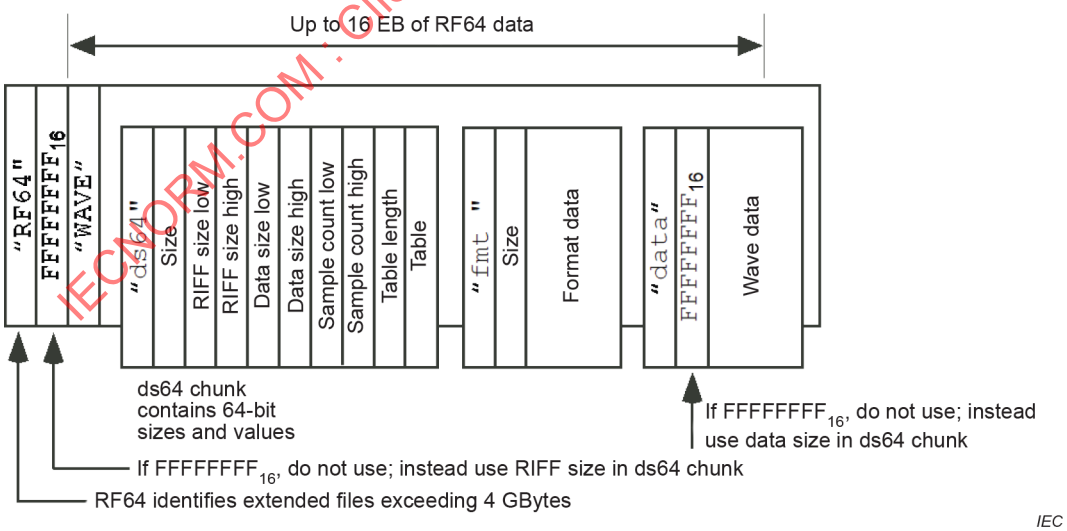


Figure F.2 – Extended RF64/WAVE format

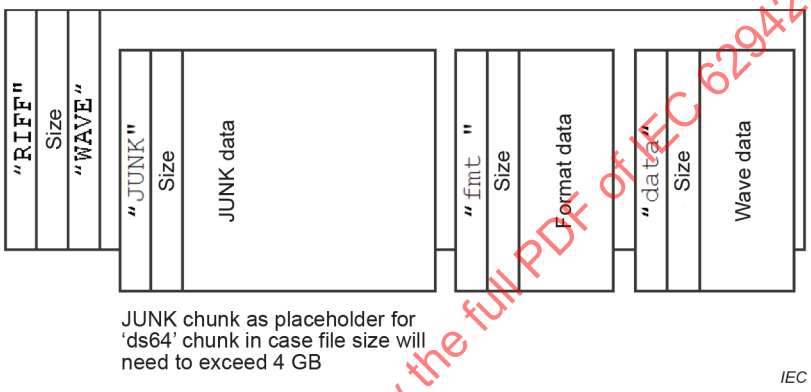
F.3 Compatibility between BWF and BWF-E

F.3.1 General

Many production audio files will be smaller than 4 GB 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 GB, the recording application shall switch from BWF to the extended format (BWF-E) at the 4-GB 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 Annex B) that is of the same size as a 'ds64' chunk (see Figure F.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.



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

There may be other chunks between the format chunk and the WAVE data chunk.

Figure F.3 – Compatible RIFF/WAVE structure

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. 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 GB, 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 $FFFFFFFF_{16}$;
- 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

The structures are as follows.

```
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. See NOTE 4  */
};
```

NOTE 1 This file declaration and 'JUNK' chunk is normally followed by 'fmt ', 'bext', 'ubxt' 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 [*] The table comprises a number of 12-byte entries. A table of size of 50 entries will occupy 600 bytes.

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

The structures are as follows.

```
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;        /* see NOTE 3 */
};
```

NOTE 1 This file declaration and 'ds64' chunk is normally followed by 'fmt ', 'bext', 'ubxt' 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 GB, even in extremely large audio files (for example, the EBU 'levl' chunk will typically exceed 4 GB only when the 'data' chunk reaches about 512 GB).

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

Annex G
(normative)

bext chunk versions

G.1 Version 0

Version 0 of the BWF was published by the EBU T3285-1997. ITU-R BR.1352 [11], ITU-R BR.1352-1 [12], ITU-R BR.1352-2 [13] publications are directly compatible. The version 0 format is not defined in this document.

G.2 Version 1

Version 1 was published in July, 2001 by the EBU as EBU T3285-2001 [3]. AES31-2 published in 2006 and ITU-R BS.1352-3 [14] published in 2007 are directly compatible with EBU T3285-2001. The version 1 format is not defined in this document.

Version 1 differs from version 0 only in that 64 of the 254 reserved bytes in version 0 are used to contain an 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.3 Version 2

Version 2 was published by the EBU as EBU T3285-2011 [3]. AES31-2 published in 2012 [10], and IEC 62942 are directly compatible with 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 that 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 and ubxt chunk (see 4.4 and Clause K.3) 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 (see Table H.1) and down for negative numbers (see Table H.2).

Table H.1 – Rounding negative values

Float value	Calculation	**Value carried in** bext and ubxt **(decimal/ hexadecimal)**
−22 644	integer[(−22,644 × 100) + sgn(−22,644 × 100) · 0,5]	−2264/ $F728_{16}$
−22 645	integer[(−22,645 × 100) + sgn(−22,645 × 100) · 0,5]	−2265/ $F727_{16}$
−22 646	integer[(−22,646 × 100) + sgn(−22,646 × 100) · 0,5]	−2265/ $F727_{16}$

Table H.2 – Rounding positive values

Float value	Calculation	**Value carried in** bext and ubxt **(decimal/ hexadecimal)**
12 764	integer[(12,764 × 100) + sgn(12,764 × 100) · 0,5]	1276/ $04FC_{16}$
12 765	integer[(12,765 × 100) + sgn(12,765 × 100) · 0,5]	1277/ $04FD_{16}$
12 766	integer[(12,766 × 100) + sgn(12,766 × 100) · 0,5]	1277/ $04FD_{16}$

If any of the loudness parameters are not being used, 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 chunk should be as specified in ITU-R BS.1771-1 [23], ITU-R BS.1864 [24], ITU-R BS.2103-1 [25], ITU-R BS.2054-4 [26], and EBU Tech 3342 [27].

Annex I
(informative)

Definition of the format for "Unique" Source Identifier (USID) for use in the <OriginatorReference> field

I.1 USID

The USID in the <OriginatorReference> is generated using several independent randomization sources in order to guarantee its uniqueness in the absence of a single allocation authority. An effective and easy-to-use randomization method is obtained by combining user-, machine- and time-specific information plus a random number. These elements are:

CC	Country code: (2 characters) Based on ISO 3166-1 [22]
OOOO	Organization code: 4 characters
NNNNNNNNNNNN	Serial number: (12 characters extracted from the recorder model and serial number) This should identify the machine's type and serial number
HHMMSS	OriginationTime: (6 characters) From the <OriginationTime> field of the BWF

These elements should be sufficient to identify a particular recording in a "human-useful" form in conjunction with other sources of information, formal and informal.

In addition, the USID contains:

RRRRRRRR	Random number (8 characters) generated locally by the recorder using some reasonably random algorithm.

This element serves to separately identify files such as stereo channels or tracks within multitrack recordings, which are made at the same time.

I.2 Examples of USIDs

Example 1, USID generated by a Tascam DA88 [3], S/N 396FG347A, operated by RAI, Radiotelevisione Italiana, at time: 12:53:24

UDI format:	CCOOOONNNNNNNNNNNNNHHMMSSRRRRRRRR
UDI Example:	ITRAI0DA88396FG34712532498748726

Example 2, USID generated by a xxxxxxx, S/N sssssssss, operated by YLE, Finnish Broadcasting, at time: 08:14:48

UDI format:	CCOOOONNNNNNNNNNNNNHHMMSSRRRRRRRR
UDI Example:	FIYLE0xxxxxxssssss08144887724864

Example 3, USID generated by an Avid ProTools, S/N A123456789F, operated by NHK Tokyo, at time: 00:29:55

UDI format:	CCOOOONNNNNNNNNNNNNHHMMSSRRRRRRRR
UDI Example:	JPJOAKTOOLS456789F00295591683472

[3] Tascan, Avid, and ProTools are examples of applications using products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of these products.

Annex J
(informative)

Specification of the format for <CodingHistory> field

J.1 General

The <CodingHistory> field in the <bext> and <ubxt> chunk is defined as a collection of strings containing a history of the coding processes. A new row should be added whenever the coding history is changed. Each row should contain a string variable for each parameter of the coding. Each row should be terminated by <CR/LF>. A format for the coding history strings is given below.

J.2 Syntax

The syntax of each row should be as shown in Table J.1:

Table J.1 – CodingHistory parameters

Parameter	Variable string <allowed option>
Coding algorithm	A=<ANALOGUE, PCM>
Sampling frequency (Hz)	F=<16000,22050,24000,32000,44100,48000,88200,96000>
Word length (bits)	W=<8, 12, 14, 16, 18, 20, 22, 24>
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 $2C_{16}$). Examples of the contents: ID-No; codec type; A/D type>

The variable strings should be separated by commas (ASCII $2C_{16}$). Each row should be terminated by <CR/LF>.

J.3 Examples of coding history fields

Example 1:

A=PCM,F=48000,W=16,M=stereo,T=original,<CR/LF>

Interpretation of example 1:

The original file is recorded as a linear BWF file with PCM coding with:

– Sampling frequency: 48 kHz
– Coding resolution: 16 bits per sample
– Mode: stereo
– Status: original coding

Example 2 for a digitization process of analogue material[4]:

A=ANALOGUE,M=stereo,T=StuderA816; SN1007; 38; Agfa_PER528,<CR/LF>

A=PCM,F=48000,W=18,M=stereo,T=NVision; NV1000; A/D,<CR/LF>

Interpretation of example 2:

Line 1,

The analogue magnetic tape, type Agfa PER528, was played back on a tape recorder, Studer model A816, serial No. 1007:

- Tape speed: 38 cm/s
- Mode: stereo

Line 2,

The recording was digitized using an A/D converter type NVision NV1000 with:

- Sampling frequency: 48 kHz
- Coding resolution: 18 bits per sample
- Mode: stereo

4 Agfa, Studer, and NVision are examples of applications using products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by IEC of these products.

Annex K
(normative)

Universal broadcast audio extension chunk

K.1 General

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

K.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 */
```

K.3 Universal broadcast audio extension chunk

The information contained in the Broadcast Audio Extension "bext" chunk defined in 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, shall use UTF-8 coding (8-bit UCS Transformation Format) instead of ASCII (see RFC 3629). 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 uOriginatorReference[32*8];   /* UTF-8: "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 */
INT LoudnessValue;                 /* Integrated Loudness Value of the file in
                                   LKFS (multiplied by 100) see Annex H */
INT LoudnessRange;                 /* Loudness Range of the file in LU
                                   (multiplied by 100) see Annex H */
INT MaxTruePeakLevel;              /* Maximum True Peak Level of the file
```

```
                                    expressed as dBTP (multiplied by 100) see
                                    Annex H */
INT MaxMomentaryLoudness;           /* Highest value of the Momentary Loudness
                                    Level of the file in LKFS (multiplied by
                                    100) see Annex H */
INT MaxShortTermLoudness;           /* Highest value of the Short-Term Loudness
                                    Level of the file in LKFS (multiplied by
                                    100) see Annex H */
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 K.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.

When a given code value in UTF-8 is out of the subset (as defined in Clause 12 of ISO/IEC 10646:2017) 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 (see 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, or a null character (see NOTE). 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 are different for any reason, the content of the bext chunk shall be preferred.

NOTE This ASCII equivalent is intended to be default text chosen by the application so that translation by the operator is not required or expected. An example might be: "See data in ubxt chunk".

Table K.1 – `ubxt` field content definitions

uDescription	Human	UTF-8 string, 2 048 bytes or fewer, 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 fewer, 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 fewer, 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 defined from 01 to 12 "day" is defined from 01 to 31 Some legacy implementations can 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 that 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 H 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 H.
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 H 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 H 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 H 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. A standard format for the coding history information is given in Annex J.

Bibliography

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

[2] AES31-2-2012, *AES 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*

[3] EBU Tech 3285-2001, *BWF – a format for audio data files in broadcasting. Version 1*

[4] EBU Tech 3285-2011, *BWF – a format for audio data files in broadcasting. Version 2*

[5] EBU Tech 3306-2007, *RF64: An extended File Format for Audio*

[6] EBU Tech 3285_S1-1997, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 1 – MPEG audio*

[7] EBU Tech 3285_S4-2003, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 4: <link> chunk*

[8] EBU Tech 3285_S5-2003, *Specification of the Broadcast Wave Format, A format for audio data files in broadcasting, Supplement 5 <axml> chunk*

[9] EBU-N22-1997, *The Broadcast Wave Format, A format for audio data files in broadcasting*

[10] EBU Recommendation R85-2004, *Use of the Broadcast Wave Format for the exchange of audio data files*

[11] Rec. ITU-R BR.1352, *File format for the exchange of audio programme materials with metadata on information technology media, BWF Version 0*

[12] Rec. ITU-R BR.1352-1, *File format for the exchange of audio programme materials with metadata on information technology media, BWF Version 0*

[13] Rec. ITU-R BR.1352-2, *File format for the exchange of audio programme materials with metadata on information technology media, BWF Version 0*

[14] Rec. ITU-R BR.1352-3, *File format for the exchange of audio programme materials with metadata on information technology media, BWF Version 1*

[15] JPPA-1-2009, *BWF-J audio file format* (Japanese only) http://www.jppanet.or.jp/ bwf-j/bwf-j.html

[16] *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

[17] Rec. ITU-R BR.1384-2, *Parameters for international exchange of multi-channel sound recordings with or without accompanying picture*

[18] EBU Technical Recommendation R91-2004, *Track allocations and recording levels for the exchange of multichannel audio signals*

[19] SMPTE ST 2035-2009, *Audio Channel Assignments for Digital Television Recorders (DTRs)*

[20] SMPTE ST 2035-2009 Amendment 1:2012, *Audio Channel Assignments for Digital Television Recorders (DTRs) – Amendment 1*

[21] SMPTE ST 323-2004; *Channel Assignments and Levels on Multichannel Audio Media (Film)*

[22] ISO 3166-1, *Codes for the representation of names of countries and their subdivisions - Part 1: Country codes*

[23] Rec. ITU-R BS.1771-1, *Requirements for loudness and true-peak indicating meters*

[24] Rec. ITU-R BS.1864, *Operational practices for loudness in the international exchange of digital television programmes*

[25] Rep. ITU-R BS.2103-1, *Short-term loudness metering*

[26] Rep. ITU-R BS.2054-4, *Audio levels and loudness*

[27] EBU Tech 3342, *Loudness Range: A descriptor to supplement loudness normalisation in accordance with EBU R 128*

[28] Rec. ITU-R BS.1770-4, *Algorithms to measure audio programme loudness and true-peak audio level*

[29] EBU Tech 3343-2011, *Practical Guidelines for Production and Implementation in accordance with EBU R 128*

[30] EBU Tech 3344-2011, *Practical Guidelines for Distribution of Programmes in accordance with EBU R 128*

[31] ATSC A/85:2013, *ATSC Recommended Practice: Techniques for Establishing and Maintaining Audio Loudness for Digital Television*

[32] ARIB TR-B32 V1.5, *ARIB Technical Report: Operational Guidelines for Loudness of Digital Television Programs* (Japanese edition only)

[33] ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

FORMAT DE FICHIER POUR LE TRANSFERT ET L'ÉCHANGE PROFESSIONNELS DE DONNÉES AUDIONUMÉRIQUES

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62942 a été établie par le domaine technique 6: Média de stockage, structures des données, équipements et systèmes, du comité d'études 100 de l'IEC: Systèmes et équipements audio, vidéo et services de données.

La présente version bilingue (2020-09) correspond à la version anglaise monolingue publiée en 2019-12.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

Les fichiers BWFF reposent sur le format de fichier audio WAVE[1] de Microsoft, qui est un type de fichier spécifié dans le format RIFF [1][2] de Microsoft. Les fichiers WAVE contiennent plus précisément des données audio. Le bloc de base d'un fichier RIFF est un fragment qui contient des informations spécifiques, un champ d'identification et un champ de taille. Un fichier RIFF contient un certain nombre de fragments.

Les fichiers BWFF comprennent plus précisément un fragment <Broadcast Audio Extension> afin de transporter certaines métadonnées qui sont importantes pour la diffusion et l'utilisation professionnelle. Pour que l'échange soit fiable, certaines restrictions s'appliquent au format des données audio.

Le format BWF a initialement été développé avec du texte ASCII pour tous les champs. Ensuite, à mesure que le format était développé, l'utilisation de caractères multioctets a été proposée dans le but de l'internationaliser. Bien entendu, l'utilisation de jeux de caractères multioctets dans le format existant allait entraîner des problèmes de compatibilité lors de l'analyse des métadonnées multioctets par des applications attendant du texte ASCII. La distinction entre les métadonnées lisibles par un humain et celles lisibles par une machine a été établie, et un nouveau fragment "universel" a été créé pour transporter des métadonnées internationalisées lisibles par un humain en utilisant des jeux de caractères multioctets sans problèmes d'interopérabilité. Ce nouveau fragment est décrit à l'Annexe K.

---

1 Microsoft® est une marque déposée; Windows™ est une marque de Microsoft Corp. Cette information est donnée à l'intention des utilisateurs du présent document et ne signifie nullement que l'IEC approuve ou recommande l'emploi exclusif du produit ainsi désigné. Des produits équivalents peuvent être utilisés s'il est démontré qu'ils conduisent aux mêmes résultats.

2 Les numéros entre crochets se réfèrent à la bibliographie.

FORMAT DE FICHIER POUR LE TRANSFERT ET L'ÉCHANGE PROFESSIONNELS DE DONNÉES AUDIONUMÉRIQUES

1 Domaine d'application

Le présent document spécifie un format de fichier qui permet d'échanger des données audio entre des équipements conformes. Il vise principalement les applications audio en ce qui concerne l'enregistrement, la production, la postproduction et l'archivage professionnels.

Il est obtenu à partir de l'AES31-2 [2], mais est également compatible avec des spécifications proches, notamment l'EBU Tech 3285, [3] à [10], l'ITU-R BR.1352-3-2007, [11] à [14], et le BWF-J de la Japan Post Production Association [15].

Le présent document contient la spécification du fragment d'extension audio pour diffusion et de son utilisation avec des données audio codées MIC. Des informations de base sur le format RIFF et sur la façon dont il peut être étendu à d'autres types de données audio sont données à l'Annexe E. Des détails sur le format WAVE MIC sont également donnés à l'Annexe A.

Un format étendu facultatif, BWF-E, prend en charge l'adressage 64 bits pour permettre l'utilisation de fichiers dont la taille est supérieure à 4 Go.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

ISO/IEC 10646:2017, *Technologies de l'information – Jeu universel de caractères codés (JUC)*

ISO 8601, *Eléments de données et formats d'échange – Echange d'information – Représentation de la date et de l'heure*

SMPTE ST 330-2011, *SMPTE standard for television – Unique Material Identifier (UMID)* (disponible en anglais seulement)

INTERNET ENGINEERING TASK FORCE (IETF). RFC 3629: *UTF-8, a transformation format of ISO 10646* (disponible en anglais seulement) [en ligne]. Edité par F. Yergeau. Novembre 2003 [consulté le 26/11/2019]. Disponible à l'adresse https://www.rfc-editor.org/rfc/rfc3629.txt

3 Termes et définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1
format de fichier d'échange de ressources
RIFF
représentation de fichier sur laquelle repose le format de fichier WAVE

Note 1 à l'article: L'abréviation "RIFF" est dérivée du terme anglais développé correspondant "resource interchange file format".

3.2
fragment
paquet de données qui compose les fichiers RIFF et contient les données associées

3.3
ASCII
code de représentation des caractères sur 7 bits conformément à l'ISO/IEC 646

3.4
format de fichier audio de forme d'onde
WAVE
format de fichier audio fondé sur la structure de fichiers RIFF

3.5
fichier au format d'onde de diffusion
BWFF
fichier WAVE qui contient le fragment d'extension pour diffusion ("bext") décrit dans le présent document

Note 1 à l'article: L'abréviation "BWFF" est dérivée du terme anglais développé correspondant "broadcast wave format file".

3.6
fragment d'extension pour diffusion
bext
fragment d'extension du format WAVE

Note 1 à l'article: L'abréviation "bext" est dérivée du terme anglais développé correspondant "broadcast extension chunk".

3.7
fragment d'extension audio pour diffusion universelle
ubxt
informations lisibles par un humain relatives au fragment "bext" dans les langages multioctets

Note 1 à l'article: L'abréviation "ubxt" est dérivée du terme anglais développé correspondant "universal broadcast audio extension chunk".

3.8
UMID
identificateur unique de matériel, défini dans la SMPTE ST 330

3.9
format d'onde de diffusion étendu
BWF-E
format étendu facultatif qui remplace un en-tête RIFF par un en-tête RF64 afin de prendre en charge l'adressage 64 bits et ainsi permettre l'utilisation de fichiers dont la taille est supérieure à 4 Go

Note 1 à l'article: L'abréviation "BWF-E" est dérivée du terme anglais développé correspondant "broadcast wave format, extended".

3.10
RF64
structure équivalente au type de fichier RIFF qui prend en charge l'adressage 64 bits

3.11
CHAR
entier signé sur 8 bits qui représente une valeur entière allant de −128 à +127

Note 1 à l'article: Equivalent en langage C: "caractère signé".

3.12
BYTE
entier non signé sur 8 bits qui représente une valeur entière allant de 0 à 255

Note 1 à l'article: Equivalent en langage C: "caractère non signé".

3.13
INT
entier signé sur 16 bits qui représente une valeur entière allant de −32 768 à +32 767

Note 1 à l'article: Equivalent en langage C: "entier court signé".

Note 2 à l'article: Les types de données multioctets sont au format petit-boutiste.

3.14
WORD
entier non signé sur 16 bits qui représente une valeur entière allant de 0 à +65 535

Note 1 à l'article: Equivalent en langage C: "entier court non signé".

Note 2 à l'article: Les types de données multioctets sont au format petit-boutiste.

3.15
LONG
entier signé sur 32 bits qui représente une valeur entière allant de −2 147 483 648 à +2 147 483 647

Note 1 à l'article: Equivalent en langage C: "entier long signé".

Note 2 à l'article: Les types de données multioctets sont au format petit-boutiste.

3.16
DWORD
entier non signé sur 32 bits qui représente une valeur entière allant de 0 à +4 294 967 295

Note 1 à l'article: Equivalent en langage C: "entier long non signé".

Note 2 à l'article: Les types de données multioctets sont au format petit-boutiste.

4 Fichier BWF

4.1 Fragments existants définis dans le format RIFF

La présente spécification utilise un certain nombre de fragments RIFF qui sont déjà définis (voir Annexe A). Ces fragments sont:

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

4.2 Fragments supplémentaires

D'autres fragments peuvent se trouver dans le fichier. Certains d'entre eux peuvent ne pas relever du domaine d'application du présent document. Les applications peuvent ou peuvent ne pas interpréter ou utiliser ces fragments; l'intégrité des données contenues dans ces fragments inconnus ne peut donc pas être garantie. Toutefois, il convient que les applications conformes transmettent ces fragments inconnus sans en modifier le contenu (voir également Annexe B).

4.3 Contenu d'un BWFF

Un BWFF doit contenir l'en-tête "WAVE" du format RIFF et au moins les fragments suivants:

```
<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 Fragment d'extension audio pour diffusion

Les paramètres supplémentaires nécessaires à l'échange de matériel entre les radiodiffuseurs sont ajoutés dans une extension audio spécifique pour diffusion ou dans un fragment "bext". La structure du fragment "bext" doit être définie comme suit:

```
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 Annexe G */
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 Annexe H */
INT LoudnessRange;                     /* Loudness Range of the file in LU
                                       (multiplied by 100), see Annexe H */
INT MaxTruePeakLevel;                  /* Maximum True Peak Level of the file
                                       expressed as dBTP (multiplied by 100), see
                                       Annexe H */
INT MaxMomentaryLoudness;              /* Highest value of the Momentary Loudness
                                       Level of the file in LKFS (multiplied by
                                       100), see Annexe H */
INT MaxShortTermLoudness;              /* Highest value of the Short-Term Loudness
                                       Level of the file in LKFS (multiplied by
                                       100), see Annexe H */
BYTE Reserved[180];                    /* 180 bytes, reserved for future use, set
                                       to "NULL" */
CHAR CodingHistory[];                  /* ASCII: « History coding » */
```

} BROADCAST_EXT

Les champs du fragment "bext" doivent contenir les éléments présentés dans le Tableau 1. Noter que, dans les applications où le texte ASCII est inapproprié pour les informations lisibles par un humain (par exemple lorsqu'un jeu de caractères autre que l'ISO 646 est exigé), il est nécessaire de le transporter par un autre moyen, par exemple dans un fragment de métadonnées dédiées ajouté au BWFF. Voir également Annexe K.

Il convient que tous les éléments, à l'exception de "Description", Originator", "OriginatorReference" et "CodingHistory", aient le même contenu que celui de chaque élément correspondant du fragment "ubxt" (voir Annexe K), le cas échéant. Si les données lisibles par une machine sont mises à jour dans le fragment "bext", il convient que les données lisibles par une machine correspondantes soient également mises à jour à l'identique dans le fragment "ubxt".

Tableau 1 – Définitions du contenu des champs bext

Description	Humain	Chaîne ASCII, de 256 caractères ou moins, qui contient une description de la séquence. Si des sauts de ligne sont utilisés, les lignes doivent se terminer par <CR><LF>. Si cette donnée n'est pas disponible ou si la longueur de la chaîne est inférieure à 256 caractères, le premier caractère non utilisé doit être un caractère nul (00_{16}). Pour aider les applications qui n'affichent qu'une brève description, il convient qu'un résumé de la description soit contenu dans les 64 premiers caractères. Les 192 derniers caractères peuvent être utilisés pour donner plus d'informations.
Originator	Humain	Chaîne ASCII, de 32 caractères ou moins, qui contient le nom de l'auteur du fichier audio. Si cette donnée n'est pas disponible ou si la longueur de la chaîne est inférieure à 32 caractères, le premier caractère non utilisé doit être un caractère nul (00_{16}).
OriginatorReference	Humain	Chaîne ASCII, de 32 caractères ou moins, qui contient une référence attribuée par l'organisation d'origine. Voir Annexe I. Si cette donnée n'est pas disponible ou si la longueur de la chaîne est inférieure à 32 caractères, le premier caractère non utilisé doit être un caractère nul (00_{16}).
OriginationDate	Humain	Chaîne ASCII, de 10 caractères, qui contient la date de création de la séquence audio. Format: ***aaa-mm-jj*** ***aaaa*** = 4 caractères qui représentent l'année, doit contenir une valeur comprise entre 0000 et 9999; **-** = 1 caractère; ***mm*** = 2 caractères qui représentent le mois, doit contenir une valeur comprise entre 01 et 12; **-** = 1 caractère; ***jj*** = 2 caractères qui représentent le jour du mois, doit contenir une valeur comprise entre 01 et 31. Toutes les composantes doivent être présentes. Les tirets, "**-**", doivent être utilisés comme séparateurs dans l'expression de la date, conformément à l'ISO 8601. Pour assurer la compatibilité avec d'autres mises en œuvre, il convient que l'équipement de reproduction reconnaisse également les séparateurs suivants: trait de soulignement "**_**", deux-points "**:**", espace " ", point "**.**".

OriginationTime	Humain	Chaîne ASCII, de huit caractères, qui contient l'heure de création de la séquence audio (en heures, minutes et secondes). Si cette donnée n'est pas disponible, la valeur par défaut doit être 00:00:00. Format: ***hh*:*mm*:*ss*** ***hh*** = 2 caractères qui représentent les heures, doit contenir une valeur comprise entre 00 et 23 si l'heure est donnée; **:** = 1 caractère; ***mm*** = 2 caractères qui représentent les minutes, doit contenir une valeur comprise entre 00 et 59 si l'heure est donnée; **:** = 1 caractère; ***ss*** = 2 caractères qui représentent les secondes, doit contenir une valeur comprise entre 00 et 59. Toutes les composantes doivent être présentes. Les deux-points, "**:**", doivent être utilisés comme séparateurs dans l'expression de l'heure, conformément à l'ISO 8601. Pour assurer la compatibilité avec d'autres mises en œuvre, il convient que l'équipement de reproduction reconnaisse également les caractères suivants: trait de soulignement "**_**", tiret "**-**", espace " ", point "**.**".
TimeReference	Machine	Ce champ doit contenir le nombre d'adresses d'échantillon [time code] de la séquence. Il s'agit d'une valeur non signée sur 64 bits qui contient le nombre d'échantillons depuis minuit du premier échantillon des données audio. Le nombre d'échantillons par seconde dépend de la fréquence d'échantillonnage définie dans le champ **<nSamplesPerSec>** sous **<fmt-ck>**. La valeur par défaut est zéro, ce qui correspond à minuit.
Version	Machine	Nombre binaire non signé qui indique la version du format BWF. Il doit être défini sur 0001_{16} pour la version 1, et il doit être défini sur 0002_{16} pour la version 2. Il est défini sur 0002_{16}. Voir Annexe G.
UMID	Machine	64 octets qui contiennent un UMID étendu selon la SMPTE ST 330. Si un UMID de base sur 32 octets est utilisé, les 32 derniers octets doivent être complétés avec des zéros. Si aucun UMID n'est disponible, les 64 octets doivent être complétés avec des zéros. NOTE La longueur de l'UMID est codée dans l'en-tête de l'UMID lui-même.
LoudnessValue	Machine	Valeur de la sonie intégrée du fichier en LKFS (multipliée par 100). Entier signé sur 16 bits qui correspond à l'entier de (100 × la valeur de la sonie intégrée du fichier en LKFS) ± 0,5. Voir Annexe H pour plus de détails sur la méthode de conversion.
LoudnessRange	Machine	Entier signé sur 16 bits qui représente la plage de sonies du fichier en LU. Voir Annexe H.
MaxTruePeakLevel	Machine	Entier signé sur 16 bits qui correspond à l'entier de (100 × la valeur de crête réelle maximale du fichier en dBTP) ± 0,5. Voir Annexe H pour plus de détails sur la méthode de conversion.
MaxMomentaryLoudness	Machine	Entier signé sur 16 bits qui correspond à l'entier de (100 × la valeur la plus élevée du niveau d'isosonie momentané du fichier en LKFS) ± 0,5. Voir Annexe H pour plus de détails sur la méthode de conversion.
MaxShortTermLoudness	Machine	Entier signé sur 16 bits qui correspond à l'entier de (100 × la valeur la plus élevée du niveau d'isosonie à court terme du fichier en LKFS) ± 0,5. Voir Annexe H pour plus de détails sur la méthode de conversion.
Reserved	Machine	180 octets réservés pour une extension. Ces 180 octets doivent être définis sur zéro.

CodingHistory	Humain	Bloc de taille variable de caractères ASCII qui comprend zéro, une ou plusieurs chaînes, chacune se terminant par <CR><LF>. Le premier caractère non utilisé doit être un caractère nul (00_{16}). Chaque chaîne doit contenir une description d'un processus de codage appliqué aux données audio. Il convient que chaque nouvelle application de codage ajoute une nouvelle chaîne avec les informations appropriées. Voir Annexe J.

4.5 Nom de fichier

Il convient d'enregistrer un fichier BWF sous un nom qui peut s'adapter au plus grand nombre d'ordinateurs de différents types. Voir Annexe C.

4.6 Utilisation des voies

Les fichiers audio sont habituellement de deux types: mono (une voie) ou stéréo (deux voies). Pour utiliser le type multivoie dans les fichiers au format d'onde de diffusion, voir Annexe D.

4.7 Taille de fichier

L'espace d'adressage de 32 bits d'un fichier WAVE limite sa taille maximale à 4 Go. En pratique, certains systèmes informatiques peuvent imposer une limite inférieure de 2 Go. Pour les fichiers plus volumineux, l'Annexe F décrit un format de fichier étendu, BWF-E.

Annexe A
(normative)

Format de fichier WAVE du format RIFF

A.1 Généralités

Les informations données dans la présente annexe suivent les documents de spécification du format de fichier RIFF de Microsoft. Voir [1].

A.2 Format RIFF

A.2.1 Généralités

Le format RIFF est une structure de fichiers avec codes de formatage développée pour être utilisée sur les plateformes multimédias. Le présent article décrit un "fragment" et le "format RIFF".

A.2.2 Fragments

Le bloc de base d'un fichier RIFF est nommé "fragment". En utilisant la syntaxe du langage C, un fragment peut être défini comme suit:

```
typedef    unsigned long DWORD;
typedef    unsigned char BYTE;
typedef    DWORD FOURCC;           /* Four-character code */
typedef    FOURCC CKID;            /* Four-character-code chunk identifier */
typedef    DWORD CKSIZE;           /* 32-bit unsigned size value */
typedef    struct {                /* Chunk structure */
           CKID ckID;              /* Chunk type identifier */
           CKSIZE    ckSize;       /* Chunk size field (size of ckData) */
           BYTE ckData[ckSize];    /* Chunk data */
           } CK;
```

Un "FOURCC" est représenté sous la forme d'une séquence de un à quatre caractères alphanumériques ASCII, complétée à droite par des caractères blancs (valeur de caractère ASCII 32), sans blanc à l'intérieur.

Par exemple, le code de quatre caractères (FOURCC) "FOO" est stocké comme une séquence de quatre octets: "F", "O", "O", " " dans les adresses ascendantes. Pour permettre des comparaisons rapides, un code de quatre caractères peut également être traité comme un nombre de 32 bits.

Les trois parties du fragment sont décrites dans le Tableau A.1.

Tableau A.1 – Description du fragment

Partie	Description
ckID	Code de quatre caractères qui identifie la représentation des données du fragment. Un programme qui lit un fichier RIFF peut ignorer tout fragment dont il ne reconnaît pas l'identifiant. Il ignore simplement le nombre d'octets spécifié par ckSize plus l'octet de remplissage, s'il est présent.
ckSize	Valeur non signée sur 32 bits qui identifie la taille de ckData. Cette valeur de taille ne comprend pas la taille des champs ckID et ckSize, ou l'octet de remplissage à la fin de ckData.
ckData	Données binaires de taille fixe ou variable. Le début de ckData est aligné au niveau des mots par rapport au début du fichier RIFF. Si la taille du fragment est un nombre impair d'octets, un octet de remplissage de valeur zéro est écrit après ckData. La valeur ckSize n'inclut pas l'octet de remplissage.

Un fragment peut être représenté avec la notation suivante (dans cet exemple, ckSize et l'octet de remplissage sont implicites):

```
<ckID>      ( <ckData> )
```

Un fragment RIFF peut contenir des fragments imbriqués (ou sous-fragments).

Tous les autres types de fragments stockent un seul élément de données binaires dans <ckData>.

A.2.3 Formats RIFF

Un format RIFF est un fragment avec un identifiant de fragment "RIFF". Le terme fait également référence à un format de fichier qui respecte la structure RIFF. Un "WAVE" est enregistré comme un "type de format RIFF".

En utilisant la notation qui permet de représenter un fragment, un format RIFF ressemble à ce qui suit:

```
RIFF    ( <formType>  <ck>... )
```

Les quatre premiers octets d'un format RIFF constituent un identifiant de fragment avec les valeurs "R", "I", "F", "F". Le champ ckSize est exigé, mais il est omis de la notation pour plus de simplicité.

Le premier DWORD de données du fragment "RIFF" (représenté ci-dessus par <formType>) est la valeur d'un code de quatre caractères qui identifie la représentation des données, ou le *type de format*, du fichier. Le code du type de format est suivi d'une série de sous-fragments. Les sous-fragments présents dépendent du type de format.

La définition d'un format RIFF particulier comprend généralement les éléments suivants:

– un code unique de quatre caractères, qui identifie le type de format;
– la liste des fragments exigés;
– la liste des fragments facultatifs;
– éventuellement, l'ordre exigé des fragments.

A.3 Format WAVE

A.3.1 Généralités

Le format WAVE est défini comme suit. Les programmes doivent gérer (et doivent ignorer) tous les fragments inconnus rencontrés, comme pour tous les formats RIFF (voir Annexe B). Cependant, <fmt-ck> doit toujours apparaître avant <wave-data>. Dans un fichier WAVE, ces deux fragments sont exigés.

```
<WAVE-form> ->
        RIFF ( "WAVE"
                <fmt-ck>                        /* Format chunk */
                [<fact-ck>]                     /* Fact chunk */
                [cue-ck]                        /* Cue points */
                [<playlist-ck>]                 /* playlist */
                [<assoc-data-list>]             /* Associated data list */
                <wave-data> )                   /* Wave data */
```

Les fragments WAVE sont décrits en A.3.2.

NOTE <fact-ck>, <cue-ck>, <playlist-ck> et <assoc-data-list> ne sont pas utilisés dans la présente spécification.

A.3.2 Fragment de format WAVE

Le fragment de format WAVE <fmt-ck> spécifie le format de <wave-data>.

<fmt-ck> doit être défini comme suit:

```
<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 */
         }
```

Le contenu des champs de la partie <common-fields> du fragment doit être défini comme indiqué dans le Tableau A.2.

Tableau A.2 – Fragment de format – champs communs

Champ	Description
wFormatTag	Numéro qui indique la catégorie de format WAVE du fichier. Le contenu de la partie <format-specific-fields> du fragment et l'interprétation des données de forme d'onde dépendent de cette valeur.
nchannels	Nombre de voies représentées dans les données de forme d'onde, par exemple 1 pour mono et 2 pour stéréo.
nSamplesPerSec	Fréquence d'échantillonnage, en échantillons par seconde, à laquelle il convient de reproduire chaque voie.
nAvgBytesPerSec	Nombre moyen d'octets par seconde auquel il convient de transférer les données de forme d'onde. Les logiciels de lecture peuvent estimer la taille de la mémoire tampon en utilisant cette valeur.
nBlockAlign	Alignement des blocs en octets des données de forme d'onde. Il est nécessaire que les logiciels de lecture traitent un multiple de <nBlockAlign> octets de données à la fois, la valeur de <nBlockAlign> pouvant donc être utilisée pour l'alignement de la mémoire tampon.

Les <format-specific-fields> doivent comprendre zéro, un ou plusieurs octets de paramètres. Les paramètres qui se produisent dépendent de la catégorie du format WAVE. Il convient que les logiciels de lecture admettent (et ignorent) tout paramètre <format-specific-fields> qui se produit à la fin de ce champ.

A.3.3 Catégories de format WAVE

A.3.3.1 Généralités

La catégorie de format d'un fichier WAVE doit être spécifiée par la valeur du champ <wFormatTag> du fragment de format. La représentation des données dans <wave-data> ainsi que le contenu des <format-specific-fields> du fragment de format doivent dépendre de la catégorie de format.

Les catégories de format WAVE ouvertes et non propriétaires actuellement définies sont présentées dans le Tableau A.3.

Tableau A.3 – Catégories de format WAVE

wFormatTag	Valeur	Catégorie de format
WAVE_FORMAT_PCM	0001_{16}	Format de modulation par impulsions et codage (MIC)

NOTE Bien que d'autres formats WAVE existent, seuls les formats ci-dessus sont actuellement utilisés avec le BWFF. Les détails du format WAVE MIC sont fournis à l'Article A.2. Des informations générales sur les autres formats WAVE sont données à l'Annexe F.

A.3.3.2 Format MIC

Si le champ <wFormatTag> de <fmt-ck> est défini sur WAVE_FORMAT_PCM, les données de forme d'onde doivent comprendre des échantillons représentés au format MIC. Pour les données de forme d'onde MIC, les <format-specific-fields> doivent être définis comme suit:

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

Le champ <nBitsPerSample> doit spécifier le nombre de bits de données utilisés pour représenter chaque échantillon audio de chaque voie. S'il existe plusieurs voies, la taille d'échantillon doit être la même pour chaque voie.

Il convient que le champ <nBlockAlign> soit égal à la formule suivante, arrondie au nombre entier suivant:

```
nChannels x BytesPerSample
```

La valeur de BytesPerSample doit être calculée en arrondissant nBitsPerSample à l'octet entier suivant. Lorsque le mot de l'échantillon audio est inférieur à un nombre entier d'octets, les bits de poids fort de l'échantillon audio doivent être placés dans les bits de poids fort du mot de données; les bits de données inutilisés adjacents aux bits de poids faible doivent être définis sur zéro.

Pour les données MIC, le champ <nAvgBytesPerSec> du fragment de format doit être égal à la formule suivante:

```
nSamplesPerSec x nBlockAlign
```

NOTE La spécification WAVE d'origine autorise, par exemple, des échantillons de 20 bits qui proviennent de deux voies et sont regroupés en 5 octets, en partageant un seul octet pour les bits de poids faible des deux voies. Le présent document spécifie un nombre entier d'octets par échantillon audio afin de réduire les ambiguïtés dans les mises en œuvre et d'obtenir une compatibilité d'échange maximale.

A.3.3.3 Regroupement de données pour les fichiers WAVE MIC

Dans un fichier WAVE à une seule voie, les échantillons doivent être stockés de manière consécutive.

Pour les fichiers WAVE stéréo, la voie 0 doit représenter la voie de gauche, et la voie 1 doit représenter la voie de droite. Le Tableau A.4 et le Tableau A.5 présentent des exemples de regroupement de données pour des fichiers WAVE mono et stéréo sur 16 bits.

Tableau A.4 – Regroupement de données pour MIC mono sur 16 bits

Echantillon 1		Echantillon 2	
Voie 0 octet de poids faible	Voie 0 octet de poids fort	Voie 0 octet de poids faible	Voie 0 octet de poids fort

Tableau A.5 – Regroupement de données pour MIC stéréo sur 16 bits

Echantillon 1			
Voie 0 (gauche) octet de poids faible	Voie 0 (gauche) octet de poids fort	Voie 1 (droite) octet de poids faible	Voie 1 (droite) octet de poids fort

Dans les fichiers WAVE à plusieurs voies, les échantillons doivent être entrelacés dans une séquence de voies (voir Annexe E pour des exemples).

A.3.3.4 Format de données des échantillons

Chaque échantillon est contenu dans un entier i. La taille de i est le plus petit nombre d'octets exigés pour contenir la taille d'échantillon spécifiée. L'octet de poids faible doit être stocké en premier. Les bits qui représentent l'amplitude de l'échantillon sont stockés dans les bits de poids fort de i, et les bits restants doivent être définis sur zéro.

Par exemple, si la taille d'échantillon enregistrée dans <nBitsPerSample> est de 20 bits, chaque échantillon est stocké dans un entier de trois octets. Les quatre bits de poids faible du premier octet (de poids faible) sont définis sur zéro. Le format de données et les valeurs maximales et minimales des échantillons de forme d'onde MIC de différentes tailles sont présentés dans le Tableau A.6. Un exemple correspondant de valeurs de données MIC sur 16 bits est présenté dans le Tableau A.7.

Tableau A.6 – Format de données MIC

Taille de l'échantillon	Format de données	Valeur maximale	Valeur minimale
> 8 bits	Entier signé *i*	Plus grande valeur positive de *i*	Valeur la plus négative de *i*

Tableau A.7 – Format de données MIC sur 16 bits

Format	Valeur maximale	Valeur minimale	Valeur médiane
MIC 16 bits	32 767 (7FFF16)	−32 768 (800016)	0

NOTE Les mots existants de 8 bits et moins utilisent généralement un autre schéma.

A.3.3.5 Exemples de fichiers WAVE MIC

Le Tableau A.8 présente des exemples de paramètres de fragment de format dans quatre cas.

Tableau A.8 – Exemples de fragment de format WAVE MIC

	Champ	A	B	C	D
Champs communs	**wFormatTag**	1	1	1	1
	nchannels	1	2	2	2
	nSamplesPerSec	48 000	44 100	96 000	48 000
	nAvgBytesPerSec	96 000	176 400	576 000	288 000
	nBlockAlign	2	4	6	6
Champ spécifique au format MIC	**nBitsPerSample**	16	16	24	20
A Fichier WAVE MIC avec une fréquence d'échantillonnage de 48 kHz, mono, 16 bits par échantillon					
B Fichier WAVE MIC avec une fréquence d'échantillonnage de 44,1 kHz, stéréo, 16 bits par échantillon					
C Fichier WAVE MIC avec une fréquence d'échantillonnage de 96 kHz, stéréo, 24 bits par échantillon					
D Fichier WAVE MIC avec une fréquence d'échantillonnage de 48 kHz, stéréo, 20 bits par échantillon					

A.4 Stockage des données WAVE

Le FRAGMENT <wave-data> contient les données de forme d'onde sous forme de valeurs entières de type petit-boutiste. Il doit être défini comme suit:

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

Annexe B
(normative)

Ordre des fragments

Selon les règles RIFF, les fragments peuvent être disposés dans n'importe quel ordre dans le fichier. Dans le cas spécifique d'un fichier WAVE, le <fmt-ck> doit toujours se trouver avant le <data-ck>. Les autres fragments peuvent se trouver dans n'importe quel ordre. Par exemple, pour certaines applications, il peut être pratique d'écrire des fragments de métadonnées après le fragment de données audio (au lieu de les écrire avant).

Bien que des conventions internes concernant l'ordre d'écriture des fragments puissent s'appliquer à une application particulière, il convient que cette application soit capable de lire des fichiers qui proviennent d'autres systèmes et dont les fragments sont dans un autre ordre.

NOTE 1 Le format étendu décrit à l'Annexe F exige que le fragment "ds64", ou sa marque de réservation "JUNK", soit le premier fragment du fichier en tant qu'exception spécifique.

NOTE 2 Le fragment "JUNK" n'est pas nécessaire pour obtenir un fichier BWF conforme; il devrait cependant être inclus par les applications qui souhaitent utiliser des fichiers dont la taille est supérieure à 4 Go.

Annexe C
(normative)

Conventions de noms de fichier

C.1 Généralités

L'échange généralisé de fichiers audio implique qu'ils doivent pouvoir être lus sur des ordinateurs et des systèmes d'exploitation qui peuvent être très différents du système d'origine. Si le nom d'un fichier est inapproprié, ce fichier pourrait ne pas pouvoir être reconnu par le système cible. Par exemple, certains systèmes d'exploitation informatiques limitent le nombre de caractères composant le nom des fichiers. D'autres ne sont pas en mesure de prendre en charge les caractères multioctets. Certains caractères ont une signification particulière dans certains systèmes d'exploitation, et il convient d'éviter de les utiliser.

Ces lignes directrices visent à identifier les meilleures pratiques pour les échanges internationaux généralisés.

C.2 Longueur des noms de fichier

Il convient que le nom des BWFF ne dépasse pas 31 caractères, extension de fichier comprise.

C.3 Extension des noms de fichier

Les fichiers BWF doivent utiliser la même extension de quatre caractères, ".wav", que pour les fichiers WAVE conventionnels. Cela permet de lire le contenu audio sur la plupart des ordinateurs, sans logiciel supplémentaire. Il convient, en pratique, que les systèmes acceptent également d'autres extensions (".bwf" par exemple) qui pourraient avoir été utilisées par erreur.

C.4 Jeu de caractères des noms de fichier

Pour l'échange international, il convient que le nom des fichiers ne comprenne que des caractères ASCII (ISO/IEC 646 [33]) sur 7 bits compris entre 32 et 126 (décimal), comme indiqué dans le Tableau C.1.

Tableau C.1 – Caractères admis dans le nom des fichiers

Caractère	Valeur décimale	Valeur hexadécimale
(espace)	32	2016
…	…	…
~ (tilde)	126	7E16

De plus, les caractères répertoriés dans le Tableau C.2 sont réservés aux fonctions spéciales de certains systèmes de fichiers; il convient de ne pas les utiliser dans le nom des fichiers.

Tableau C.2 – Caractères non admis dans le nom des fichiers

Caractère	Valeur décimale	Valeur hexadécimale
"	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	$7C_{16}$

En outre, il convient de ne pas utiliser les caractères répertoriés dans le Tableau C.3 comme premier ou dernier caractère du nom d'un fichier.

Tableau C.3 – Terminateurs non admis dans le nom des fichiers

Caractère	Valeur décimale	Valeur hexadécimale
(espace)	32	20_{16}
. (point)	46	$2E_{16}$

Annexe D
(informative)

Utilisation multivoie

D.1 Généralités

Les systèmes d'édition ont généralement besoin d'accéder aux fichiers mono pour pouvoir les modifier de manière flexible. Toutefois, certains enregistreurs d'emplacement fonctionnent plus efficacement lors de l'enregistrement de plusieurs voies dans un seul fichier multivoie. En conséquence, de nombreuses applications de montage sont capables de convertir des enregistrements source multivoies en un ensemble cohérent de fichiers mono.

D.2 Regroupement de données audio multivoies

Les fichiers avec plusieurs voies audio sont construits par une simple extension du format existant. Le champ "Channel" de <fmt-ck> est défini sur le nombre de voies, et les données audio sont regroupées de manière séquentielle, comme le montrent les exemples suivants.

Les données audio d'un fichier mono sur 24 bits doivent être regroupées comme le montre la Figure D.1.

Echantillon 1			Echantillon 2		
Octet de poids faible	Octet de poids intermédiaire	Octet de poids fort	Octet de poids faible	Octet de poids intermédiaire	Octet de poids fort

IEC

Figure D.1 – Regroupement des données audio MIC mono sur 24 bits

Les données audio d'un fichier stéréo sur 16 bits doivent être regroupées comme le montre la Figure D.2.

Echantillon 1				Echantillon 2			
Voie 1 (gauche)		Voie 2 (droite)		Voie 1 (gauche)		Voie 2 (droite)	
Octet de poids faible	Octet de poids fort	Octet de poids faible	Octet de poids fort	Octet de poids faible	Octet de poids fort	Octet de poids faible	Octet de poids fort

IEC

Figure D.2 – Regroupement des données audio MIC stéréo (2 voies) sur 16 bits

Les données audio d'un fichier à quatre voies sur 24 bits doivent être regroupées comme le montre la Figure D.3.

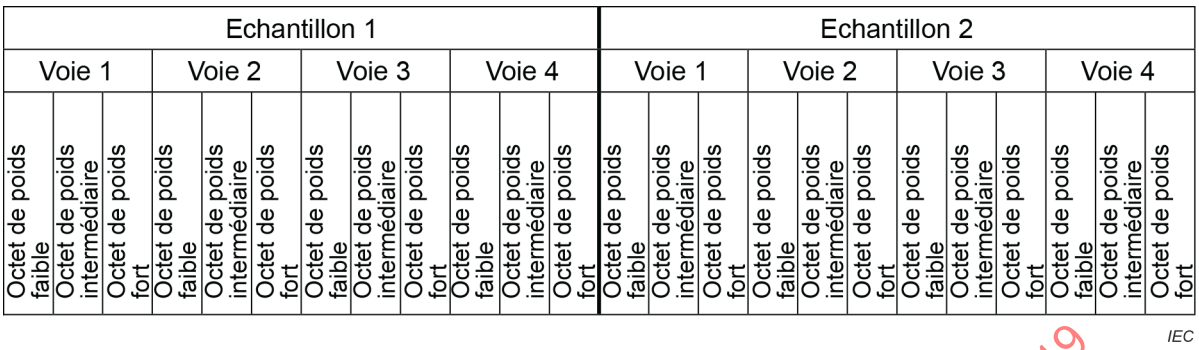
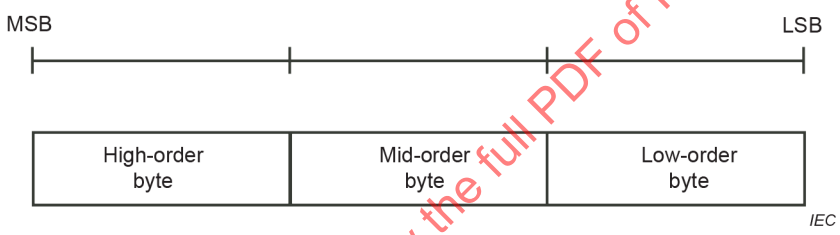


Figure D.3 – Regroupement des données audio MIC 4 voies sur 24 bits

Les octets de données d'un fichier sur 24 bits doivent être disposés comme le montre la Figure D.4.



Anglais	Français
MSB	Bit de poids fort
LSB	Bit de poids faible
High-order byte	Octet de poids fort
Mid-order byte	Octet de poids intermédiaire
Low-order byte	Octet de poids faible

Figure D.4 – Regroupement des échantillons sur 24 bits

D.3 Affectation des voies dans les fichiers multivoies

D.3.1 Généralités

L'utilisation des voies n'est pas couverte par le présent document. Il existe deux principaux domaines d'application, indiqués en D.3.2 et en D.3.3.

D.3.2 Distribution et archivage

Les enregistrements audio stéréo ou ambiophoniques destinés à l'archivage ou à la distribution d'éléments intermédiaires ou de masters au stade final peuvent utiliser les voies audio selon l'un des nombreux schémas prédéterminés. Voici des exemples: ITU-R BR.1384-2 [17], EBU R91-2004 [18], SMPTE ST 2035-2009 [19], SMPTE ST 2035-2009 Amendement 1:2012 [20] et SMPTE ST 323-2004 [21].

D.3.3 Enregistrements de production

Les enregistrements de production peuvent utiliser les voies de différentes manières. Celles-ci sont décrites dans un rapport d'enregistrement traditionnel ou dans un format approprié de métadonnées associées à l'enregistrement audio. L'ensemble des voies audio peut recevoir des enregistrements associés en ensembles mono, stéréo LR, stéréo MS ou ambiophoniques.

Annexe E
(informative)

Autres codages audio

E.1 Généralités

Tous les types WAVE autres que MIC contiennent à la fois un fragment de fait <fact-ck> et une description au format WAVE étendu dans le fragment de format <fmt-ck>.

E.2 Fichiers MPEG

Des détails sur le format WAVE MPEG sont donnés dans l'ITU-R BR.1352-3, Annexe 2 et Appendice 1 de l'Annexe 2 [14], et dans l'EBU T3285, Supplément 1 [6].

Annexe F
(normative)

Format de fichier étendu (BWF-E)

F.1 Généralités

L'espace d'adressage de 32 bits d'un fichier WAVE limite sa taille maximale à 4 Go. En pratique, certains systèmes informatiques peuvent imposer une limite inférieure de 2 Go. Cette limite n'a pas d'impact sur les fichiers mono à une fréquence d'échantillonnage de base. Elle devient plus problématique avec l'augmentation du nombre de voies dans le fichier, ou lorsque la fréquence d'échantillonnage est multipliée par deux ou quatre.

La concaténation de plusieurs fichiers BWF est possible en utilisant le fragment "link" de l'UER (Union européenne de radio-télévision, ou EBU, *European Broadcasting Union*). Cette technique est décrite dans l'EBU T3285-S4-2003 [7].

Le format de fichier étendu décrit ci-dessous vise à répondre au besoin à long terme de fichiers d'une taille supérieure à 4 Go. Il étend les capacités de taille maximale du format WAVE/RIFF en augmentant son espace d'adressage à 64 bits si nécessaire. L'effort exigé par les développeurs des logiciels est faible.

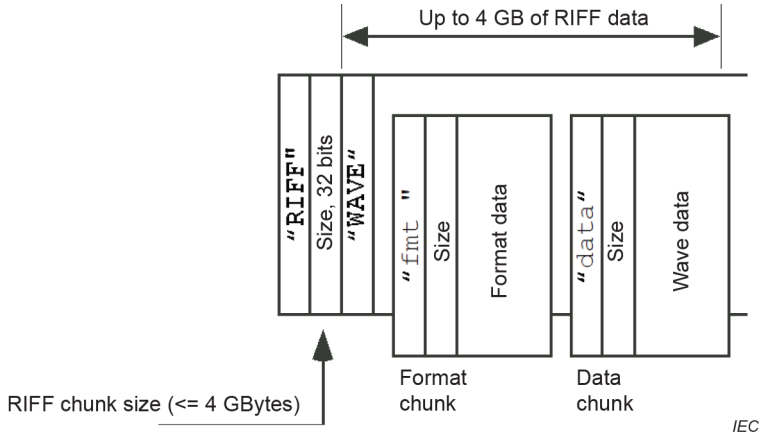
Le format de fichier BWF-E est conçu pour être une extension compatible avec le format BWF. Le format BWF-E est également conçu pour être mutuellement compatible avec le format étendu "RF64" de l'EBU T3306 [5]. Noter que l'EBU T3306 contient des spécifications complémentaires pour les affectations de masque de voie qui ne sont pas incluses dans le présent document.

Le format étendu est utilisé ici comme une extension du format de fichier WAVE/RIFF sous-jacent. La conformité au présent document exige également le fragment "bext" défini en 4.4 et peut exiger d'autres fragments de données.

F.2 Dépassement de la limite de 4 Go

F.2.1 Généralités

La limite de 4 Go est due à l'adressage 32 bits inhérent aux formats de fichier RIFF et WAVE. Avec un adressage 32 bits, un maximum de 4 294 967 296 octets = 4 Go peut être adressé (voir Figure F.1).



Anglais	Français
Up to 4 GB of RIFF data	Jusqu'à 4 Go de données RIFF
Size, 32 bits	Taille, 32 bits
Size	Taille
Format data	Données de format
Wave data	Données d'onde
RIFF chunk size (<= 4 GBytes)	Taille du fragment RIFF (<= 4 Go)
Format chunk	Fragment de format
Data chunk	Fragment de données

Figure F.1 – Format WAVE/RIFF conventionnel

Pour utiliser des fichiers plus volumineux, une plage d'adresses plus large est nécessaire. Le présent document spécifie l'adressage 64 bits. Cependant, observation évidente, mais aussi importante, le simple fait de modifier la taille de chaque champ d'un fichier WAVE pour prendre en charge l'adressage 64 bits produirait un fichier qui n'est pas compatible avec le format WAVE/RIFF normalisé.

NOTE Des contraintes supplémentaires relatives à la taille des fichiers peuvent être imposées par les mises en œuvre informatiques et les systèmes d'exploitation.

F.2.2 Format de fichier d'échange de ressources sur 64 bits (RF64)

Le présent paragraphe définit un format de fichier d'échange de ressources sur 64 bits nommé "RF64". Le format RF64 doit être identique au format RIFF, excepté pour les points ci-après.

L'identifiant "RF64" doit être utilisé à la place de "RIFF" dans les quatre premiers octets du fichier.

Un fragment "ds64" (données sur 64 bits) doit être ajouté. Il doit être le premier fragment après le fragment "RF64". Le fragment "ds64" doit contenir trois valeurs entières sur 64 bits, qui fournissent des valeurs de remplacement pour les champs 32 bits équivalents du format RIFF:

riffSize doit remplacer le champ de taille **RIFF**;

dataSize doit remplacer le champ de taille du fragment "**data**";

sampleCount doit remplacer la valeur du nombre d'échantillons dans le fragment "**fact**".

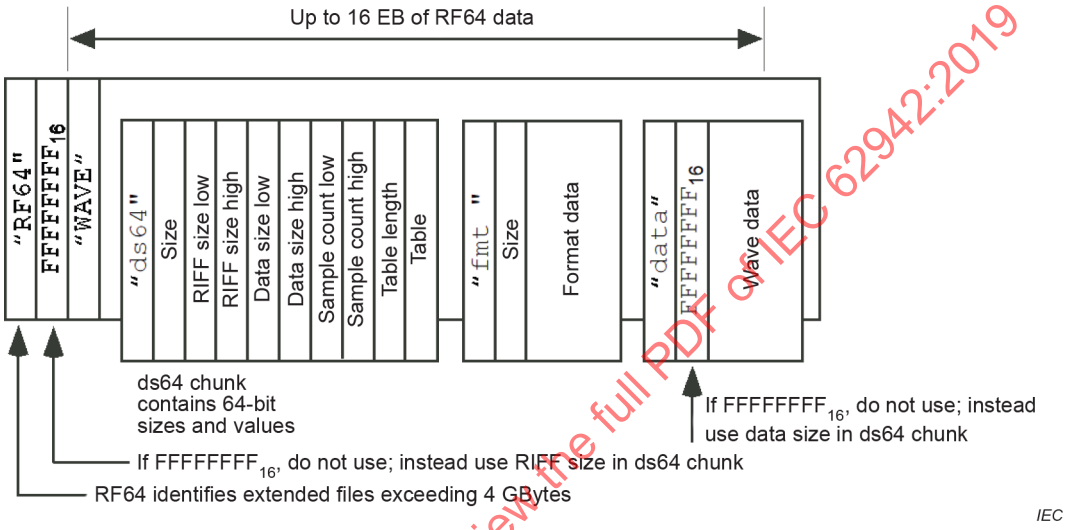
Pour les trois champs 32 bits du format WAVE/RIFF, la règle suivante doit s'appliquer:

– si la valeur sur 32 bits du champ n'est pas "$FFFFFFFF_{16}$", cette valeur sur 32 bits doit être utilisée;

– si la valeur sur 32 bits du champ est "$FFFFFFFF_{16}$", la valeur sur 64 bits du fragment "ds64" doit être utilisée à la place.

Une table étendue d'éléments ChunkSize64 vise à prendre en charge des variables 64 bits supplémentaires (voir F.4.2). Il convient de réserver de l'espace pour au moins 50 éléments.

La structure complète du format de fichier RF64 est représentée à la Figure F.2. Voir également F.4.2.



Anglais	Français
Up to 16 EB of RF64 data	Jusqu'à 16 Go de données RF64
Size	Taille
RIFF size low	Taille basse RIFF
RIFF size high	Taille haute RIFF
Data size low	Taille basse des données
Data size high	Taille haute des données
Sample count low	Nombre bas d'échantillons
Sample count high	Nombre haut d'échantillons
Table length	Longueur de table
Table	Table
Format data	Données de format
Wave data	Données d'onde
ds64 chunk contains 64-bit sizes and values	Le fragment ds64 contient des tailles et des valeurs 64 bits
If $FFFFFFFF_{16}$, do not use; instead use data size in ds64 chunk	Si $FFFFFFFF_{16}$, ne pas utiliser; utiliser à la place la taille de données du fragment ds64
If $FFFFFFFF_{16}$, do not use; instead use RIFF size in ds64 chunk	Si $FFFFFFFF_{16}$, ne pas utiliser; utiliser à la place la taille RIFF du fragment ds64
RF64 identifies extended files exceeding 4 GBytes	RF64 identifie un fichier étendu dépassant 4 Go

Figure F.2 – Format étendu WAVE/RF64