RECOMMENDATION ITU-R BR.1352-3

File format for the exchange of audio programme materials with metadata on information technology media

(Question ITU-R 58/6)

(1998-2001-2002-2007)

Scope

This Recommendation contains the specification of the broadcast audio extension chunk[1] and its use with PCM-coded, and MPEG-1 or MPEG-2 audio data. Basic information on the RIFF format and how it can be extended to other types of audio data is also included.

The ITU Radiocommunication Assembly,

considering

a) that storage media based on Information Technology, including data disks and tapes, have penetrated all areas of audio production for radio broadcasting, namely non-linear editing, on-air play-out and archives;

b) that this technology offers significant advantages in terms of operating flexibility, production flow and station automation and it is therefore attractive for the up-grading of existing studios and the design of new studio installations;

c) that the adoption of a single file format for signal interchange would greatly simplify the interoperability of individual equipment and remote studios, it would facilitate the desirable integration of editing, on-air play-out and archiving;

d) that a minimum set of broadcast related information must be included in the file to document the metadata related to the audio signal;

e) that, to ensure the compatibility between applications with different complexity, a minimum set of functions, common to all the applications able to handle the recommended file format must be agreed;

f) that Recommendation ITU-R BS.646 defines the digital audio format used in audio production for radio and television broadcasting;

g) that the need for exchanging audio materials also arises when ISO/IEC 11172-3 and ISO/IEC 13818-3 coding systems are used to compress the signal;

h) that the compatibility with currently available commercial file formats could minimize the industry efforts required to implement this format in the equipment;

j) that a standard format for the coding history information would simplify the use of the information after programme exchange;

k) that the quality of an audio signal is influenced by signal processing experienced by the signal, particularly by the use of non-linear coding and decoding during bit-rate reduction processes,

[1] A chunk is the basic building block of a file in the Microsoft ® Resource Interchange File Format (RIFF).

recommends

1 that, for the exchange of audio programmes on Information Technology media, the audio signal parameters, sampling frequency, coding resolution and pre-emphasis should be set in agreement with the relevant parts of Recommendation ITU-R BS.646;

2 that the file format specified in Annex 1 should be used for the interchange of audio programmes in linear pulse code modulation (PCM) format on Information Technology media;

3 that, when the audio signals are coded using ISO/IEC 11172-3 or ISO/IEC 13818-3 coding systems, the file format specified in Annex 1 and complemented with Annex 2 should be used for the interchange of audio programmes on Information Technology media[2];

4 that, when the file format specified in Annexes 1 and/or 2 is used to carry information on the audio material gathered and computed by a capturing workstation (Digital Audio Workstation (DAW)), the metadata should conform to the specifications detailed in Annex 3.

Annex 1

Specification of the broadcast wave format

A format for audio data files in broadcasting

1 Introduction

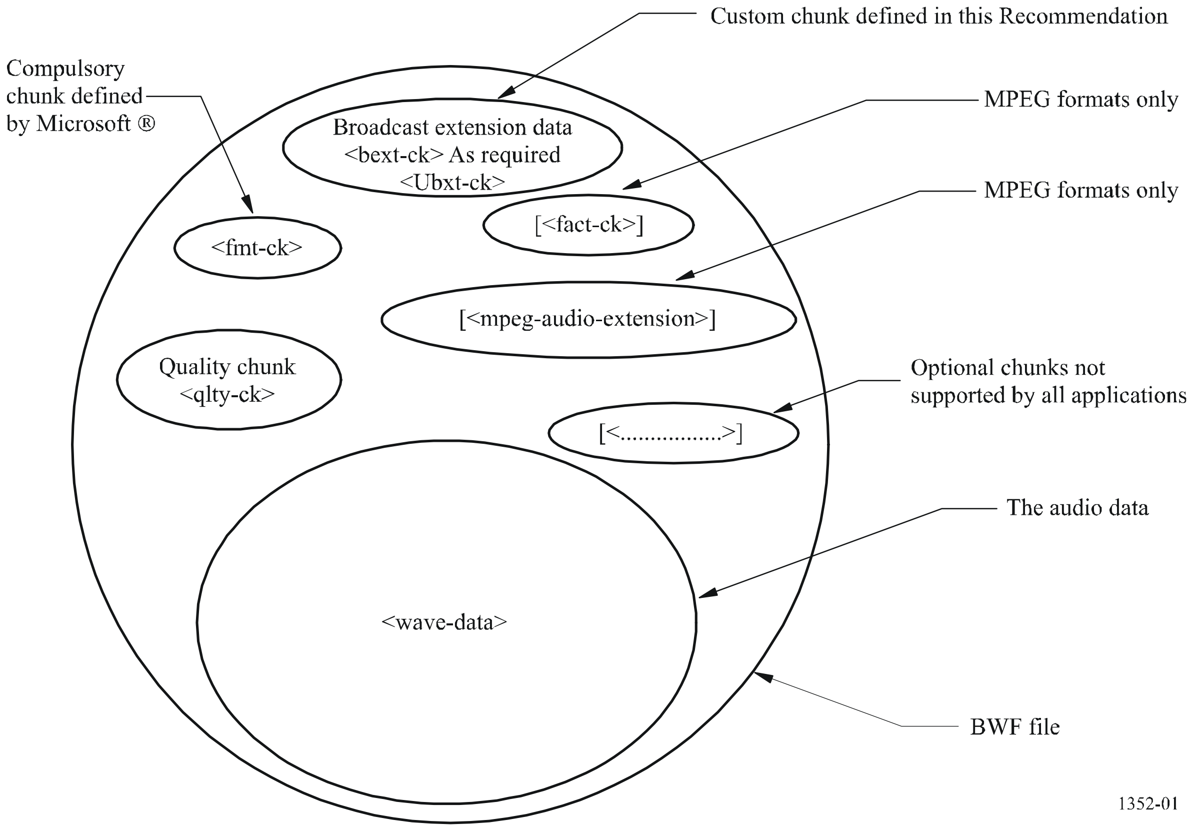
The Broadcast Wave Format (BWF) is based on the Microsoft® WAVE audio file format which is a type of file specified in the Microsoft® "Resource Interchange File Format", RIFF. WAVE files specifically contain audio data. The basic building block of the RIFF file format, called a chunk, contains a group of tightly related pieces of information. It consists of a chunk identifier, an integer value representing the length in bytes and the information carried. A RIFF file is made up of a collection of chunks.

For the BWF, some restrictions are applied to the original WAVE format. In addition the BWF file includes a <Broadcast Audio Extension> chunk. This is illustrated in Fig. 1.

[2] It is recognized that a recommendation in that sense could penalize developers using some computer platforms.

FIGURE 1

BWF file


Custom chunk defined in this Recommendation
Compulsory chunk defined by Microsoft ®
Broadcast extension data <bext-ck> As required <Ubxt-ck>
MPEG formats only
[<fact-ck>]
<fmt-ck>
MPEG formats only
[<mpeg-audio-extension>]
Quality chunk <qlty-ck>
Optional chunks not supported by all applications
[<.................>]
The audio data
<wave-data>
BWF file
1352-01

Annex contains the specification of the broadcast audio extension chunk that is used in all BWF files. In addition, information on the basic RIFF format and how it can be extended to other types of audio data is given in Appendix 1. Details of the PCM wave format are also given in Appendix 1. Detailed specifications of the extension to other types of audio data, and metadata are included in Annexes 2 and 3.

1.1 Normative provisions

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g. interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met.

The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express those mandatory provisions. The use of such words does not suggest that compliance with the Recommendation is required of any party.

2 Broadcast wave format (BWF) file

2.1 Contents of a broadcast wave format file

A broadcast wave format file shall start with the mandatory Microsoft® 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 extension>     /* ubxt is required for multi-byte language
                                          support only*/
 <fact-ck>                                /* Fact chunk is required for MPEG formats
                                          only*/
<mpeg_audio_extension>                    /* MPEG Audio Extension chunk is required
                                          for MPEG formats only*/
<wave-data> )                             /*sound data */
<quality-chunk>                           /* only required when information concerning
                                          relevant events impacting quality is needed*/
```

NOTE 1 – Additional chunks may be present in the file. Some of these may be outside the scope of this Recommendation. 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 shall pass on unknown chunks transparently.

2.2 Existing chunks defined as part of the RIFF standard

The RIFF standard is defined in documents issued by the Microsoft®[3] Corporation. This application uses a number of chunks that are already defined. These are:

fmt-ck

fact-ck

The current descriptions of these chunks are given for information in Appendix 1 to Annex 1.

2.3 Broadcast audio extension chunk[4]

Extra parameters needed for exchange of material between broadcasters are added in a specific “Broadcast Audio Extension” chunk defined as follows:

```
broadcast_audio_extension typedef struct {
      DWORD   ckID,                       /* (broadcastextension)ckID=bext. */
      DWORD   ckSize,                     /* size of extension chunk */
      BYTE    ckData[ckSize],             /* data of the chunk */
}

typedef struct broadcast_audio_extension {
CHAR Description[256],                    /* ASCII : ”Description of the sound sequence”*/
```

[3] Microsoft Resource Interchange File Format, RIFF, available (2005-12) at http://www.tactilemedia.com/info/MCI_Control_Info.html.

[4] See § 2.4 for ubxt chunk definition, to express the human-readable information of the bext chunk in a multi-byte character set.

```
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
*/
BYTE UMID_0,                      /* Binary byte 0 of SMPTE UMID */....
BYTE UMID_63,                     /* Binary byte 63 of SMPTE UMID */
CHAR Reserved[190],               /* 190 bytes, reserved for future use,
                                     set to .NULL. *          /
CHAR CodingHistory[],             /* ASCII : ”History coding“ */
} BROADCAST_EXT,
```

Field	**Description**
Description	ASCII string (maximum 256 characters) containing a free description of the sequence. To help applications which only display a short description it is recommended that a resume of the description is contained in the first 64 characters and the last 192 characters are used for details. If the length of the string is less than 256 characters the last one is followed by a null character. (0x00)
Originator	ASCII string (maximum 32 characters) containing the name of the originator/producer of the audio file. If the length of the string is less than 32 characters the field is ended by a null character. (0x00)
OriginatorReference	ASCII string (maximum 32 characters) containing a non ambiguous reference allocated by the originating organization. If the length of the string is less than 32 characters the field is ended a null character. (0x00) A standard format for the “Unique” Source Identifier (USID) information for use in the OriginatorReference field is given in Appendix 3 to Annex 1.
OriginationDate	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. Year is defined from 0000 to 9999 Month is define from 1 to 12 Day is defined from 1 to 31 The separator between the items should be a hyphen in compliance with ISO 8601. Some legacy implementations may use ‘_’ underscore, ‘:’ colon, ‘ ’ space, ‘.’ Stop, reproducing equipment should recognize these separator characters
OriginationTime	8 ASCII characters containing the time of creation of the audio sequence. The format is «‘hour,’-‘,minute,’-‘,second’» with 2 characters per item. Hour is defined from 0 to 23. Minute and second are defined from 0 to 59.

	The separator between the items should be a hyphen in compliance with ISO 8601 . Some legacy implementations may use ‘_’ underscore, ‘:’ colon, ‘ ’ space, ‘.’ Stop, reproducing equipment should recognize these separator characters.
TimeReference	This field contains the time-code of the sequence. It is a 64-bit value which contains the first sample count since midnight. The number of samples per second depends on the sample frequency that is defined in the field <nSamplesPerSec> from the <**fmt-ck**>.
Version	An unsigned binary number giving the version of the BWF. For Version 1, this is set to 0x0001.
UMID	64 bytes containing an extended UMID defined by SMPTE 330M. If a32-byte basic UMID is used, the last 32 bytes should be filled with zeros. If no UMID is available, the 64 bytes should be filled with zeros. NOTE – The length of the UMID is coded at the head of the UMID itself.
Reserved	190 bytes reserved for extension. These 190 bytes should be set to zero.
Coding History	A variable-size block of ASCII characters comprising 0 or more strings each terminated by <CR><LF>The first unused character should be a null character (0x00). Each string should contain a description of a coding process applied to the audio data. Each new coding application should add a new string with the appropriate info. A standard format for the coding history information is given in Appendix 2 to Annex 1. This information must contain the type of sound (PCM or MPEG) with its specific parameters: PCM: mode (mono, stereo), size of the sample (8, 16 bits) and sample frequency, MPEG: sampling frequency, bit rate, Layer (I or II) and the mode (mono, stereo, joint stereo or dual channel), It is recommended that the manufacturers of the coders provide an ASCII string for use in the coding history.

2.4 Universal broadcast audio extension chunk

The information contained in the Broadcast Audio Extension (bext) chunk defined in § 2.3 may additionally be carried by a dedicated chunk called “Universal Broadcast Audio Extension”, or “ubxt” chunk to express the human-readable information of the bext chunk in multi-byte languages. The basic structure of this metadata chunk is the same as that of the bext chunk. Four human-readable items, uDescription, uOriginator, uOriginatorReference and uCodingHistory, are described in UTF-8 (*8-bit UCS Transformation Format*) instead of ASCII. The first three items have 8 times the data size of the corresponding items in the bext chunk. The structure of the ubxt chunk is defined as follows:

```
typedef struct chunk_header {
        DWORD   ckID;           /* (universal broadcast extension)ckID=ubxt */
        DWORD   ckSize;         /* size of extension chunk */
        BYTE  ckData[ckSize];               /* data of the chunk */
} CHUNK_HEADER;
typedef struct universal_broadcast_audio_extension {
        BYTE  uDescription[256*8];          /* UTF-8 : “Description of the sound sequence” */
        BYTE  uOriginator[32*8];            /* UTF-8 : “Name of the originator” */
        BYTE  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 */
        WORDVersion;                        /* Version of the BWF; unsigned binary number */
        BYTE  UMID_0;                       /* Binary byte 0 of SMPTE UMID */....
        BYTE  UMID_63;                      /* Binary byte 63 of SMPTE UMID */
        CHAR Reserved[190];                 /* 190 bytes, reserved for future use, set to “NULL” */
        BYTE  uCodingHistory[];             /* UTF-8 : "Coding history“ */
} UNIV_BROADCAST_EXT;
```

Field	**Description**
uDescription	UTF-8 string, 2 048 bytes or less, containing a description of the sequence. 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 (0x00)
uOriginator	UTF-8 string, 256 bytes or less, containing the name of the originator of the audio file. If data is not available or if the length of the string is less than 256 bytes, the first unused byte shall be a null character (0x00).
uOriginatorReference	UTF-8 string, 256 bytes or less, containing a reference allocated by the originating organization. If data is not available or if the length of the string is less than 256 bytes, the first unused byte shall be a null character (0x00)
OriginationDate	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. Year is defined from 0000 to 9999 Month is define from 1 to 12 Day is defined from 1 to 31 The separator between the items should be a hyphen in compliance with ISO 8601. Some legacy implementations may use ‘_’ underscore, ‘:’ colon, ‘ ’ space, ‘.’ Stop; reproducing equipment should recognize these separator characters
OriginationTime	8 ASCII characters containing the time of creation of the audio sequence. The format is «‘hour,’-‘,minute,’-‘,second’» with 2 characters per item. Hour is defined from 0 to 23.

	Minute and second are defined from 0 to 59.
	The separator between the items should be a hyphen in compliance with ISO 8601 . Some legacy implementations may use '_' underscore, ':' colon; ' ' space; '.' Stop; reproducing equipment should recognize these separator characters.
TimeReference	This field contains the time-code of the sequence. It is a 64-bit value which contains the first sample count since midnight. The number of samples per second depends on the sample frequency that is defined in the field <nSamplesPerSec> from the <**fmt-ck**>.
Version	An unsigned binary number giving the version of the BWF. For Version 1, this is set to 0x0001.
UMID	64 bytes containing an extended UMID defined by SMPTE 330M. If a 32-byte basic UMID is used, the last 32 bytes should be filled with zeros. If no UMID is available, the 64 bytes should be filled with zeros.
	NOTE – The length of the UMID is coded at the head of the UMID itself.
Reserved	190 bytes reserved for extension. These 190 bytes should be set to zero.
uCoding History	A variable-size block of UTF-8 characters comprising 0 or more strings each terminated by <CR><LF>. The first unused byte shall be a null character (0x00).
	Each string shall contain a description of a coding process applied tothe 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 Appendix 2 to Annex 1.
	This information shall contain the type of sound (PCM or MPEG) with its specific parameters:
	PCM: mode (mono, stereo), size of the sample (8, 16 bits) and sample frequency,
	MPEG: sampling frequency, bit rate, Layer (I or II) and the mode (mono, stereo, joint stereo or dual channel),

NOTE 1 – All the items except uDescription, uOriginator, uOriginatorReference and uCodingHistory shall have the same content as that of each corresponding item of the bext chunk.§ 2.3.

NOTE 2 – When a given code value in UTF-8 is out of the subset (as defined in Chapter 12 of ISO/IEC 10646:2003) supported by a piece of processing equipment, the value shall be unchanged and ignored for processing.

Appendix 1
to Annex 1
(Informative)

RIFF WAVE (.WAV) file format

The information in this Appendix is taken from the specification documents of Microsoft® RIFF file format. It is included for information only.

1 Waveform audio file format (WAVE)

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

```
<WAVE-form> ->
        RIFF ( 'WAVE'
                <fmt-ck>                // Format chunk
                [<fact-ck>]             // Fact chunk
                [<other-ck>]            // Other optional chunks
                <wave-data> )           // Sound data
```

The WAVE chunks are described in the following sections:

1.1 WAVE format chunk

The WAVE format chunk <fmt-ck> specifies the format of the <wave-data>. The <fmt-ck> is 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 rate */
           DWORD nAvgBytesPerSec,        /* For buffer estimation*/
           WORD nBlockAlign,             /* Data block size*/
      }
```

The fields in the <common-fields> portion of the chunk are as follows:

Field	**Description**
wFormatTag	A number indicating the WAVE format category of the file. The content of the <format-specific-fields> portion of the <fmt-ck> 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 rate (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> consists of zero or more bytes of parameters. Which parameters occur depends on the WAVE format category – see the following sections for details. Playback software should be written to allow for (and ignore) any unknown <format-specific-fields> parameters that occur at the end of this field.

1.2 WAVE format categories

The format category of a WAVE file is specified by the value of the <wFormatTag> field of the 'fmt' chunk. The representation of data in <wave-data>, and the content of the <format-specific-fields> of the 'fmt' chunk, depend on the format category.

Among the currently defined open non-proprietary WAVE format categories are as follows:

wFormatTag	**Value**	**Format Category**
WAVE_FORMAT_PCM	(0x0001)	Microsoft® (PCM) format
WAVE_FORMAT_MPEG	(0x0050)	MPEG-1 Audio (audio only)

NOTE – Although other WAVE formats are registered with Microsoft®, only the above formats are used at present with the BWF. Details of the PCM WAVE format are given in the following Section 2. General information on other WAVE formats is given in § 3. Details of MPEG WAVE format are given in Annex 2. Other WAVE formats may be defined in future.

2 PCM format

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

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

The <nBitsPerSample> field specifies the number of bits of data used to represent each sample of each channel. If there are multiple channels, the sample size is the same for each channel.

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

$$\text{nchannels} \times \text{BytesPerSample}$$

The value of BytesPerSample should 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 are placed in the most significant bits of the data word, the unused data bits adjacent to the least significant bit should be set to zero

For PCM data, the <nAvgBytesPerSec> field of the 'fmt' chunk should be equal to the following formula.

$$\text{nSamplesPerSec} \times \text{nBblockAlign}$$

NOTE 1 – The original WAVE specification permits, for example 20-bit samples from two channels to be packed into 5 bytes-sharing a single byte for the least significant bits of the two channels. This Recommendation specifies a whole number of bytes per audio sample in order to reduce ambiguity in implementations and to achieve maximum interchange compatibility.

2.1 Data packing for PCM WAVE files

In a single-channel WAVE file, samples are stored consecutively. For stereo WAVE files, channel 0 represents the left-hand channel, and channel 1 represents the right-hand channel. In multiple-channel WAVE files, samples are interleaved.

The following diagrams show the data packing for 8-bit mono and stereo WAVE files:

Data packing for 8-bit mono PCM

Sample 1	Sample 2	Sample 3	Sample 4
Channel 0	Channel 0	Channel 0	Channel 0

Data packing for 8-bit stereo PCM

Sample 1		Sample 2	
Channel 0 (left)	Channel 1 (right)	Channel 0 (left)	Channel 1 (right)

The following diagrams show the data packing for 16-bit mono and stereo WAVE files:

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

Data packing for 16-bit stereo PCM

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

2.2 Data format of the samples

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

For example, if the sample size (recorded in <nBitsPerSample>) is 12 bits, then each sample is stored in a two-byte integer. The least significant four bits of the first (least significant) byte are set to zero. The data format and maximum and minimum values for PCM waveform samples of various sizes are as follows:

Sample size	Data format	Maximum value	Minimum value
One to eight bits	Unsigned integer	255 (0xFF)	0
Nine or more bits	Signed integer i	Largest positive value of i	Most negative value of i

For example, the maximum, minimum, and midpoint values for 8-bit and 16-bit PCM waveform data are as follows:

Format	Maximum value	Minimum value	Midpoint value
8-bit PCM	255 (0xFF)	0	128 (0x80)
16-bit PCM	32767(0x7FFF)	–32768(–0x8000)	0

2.3 Examples of PCM WAVE files

Example of a PCM WAVE file with 11.025 kHz sampling rate, mono, 8 bits per sample:

```
RIFF(‘WAVE’   fmt(1, 1, 11025, 11025, 1, 8)
              data( <wave-data> ) )
```

Example of a PCM WAVE file with 22.05 kHz sampling rate, stereo, 8 bits per sample:

```
RIFF(‘WAVE’   fmt(1, 2, 22050, 44100, 2, 8)
              data( <wave-data> ) )
```

Example of a PCM WAVE file with 44.1 kHz sampling rate, mono, 20 bits per sample:

```
RIFF( ‘WAVE’ INFO(INAM(“O Canada”Z) )
              fmt(1, 1, 44100, 132300, 3, 20)
              data( <wave-data> ) )
```

2.4 Storage of WAVE data

The <**wave-data**> contains the waveform data. It is defined as follows:

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

2.5 Fact chunk

The <fact-ck> fact chunk stores important information about the contents of the WAVE file. This chunk is defined as follows:

```
<fact-ck> ->    fact( <dwFileSize:DWORD> )      /*Number of samples */
```

The chunk is not required for PCM files.

The fact chunk will be expanded to include any other information required by future WAVE formats. Added fields will appear following the <dwFileSize> field. Applications can use the chunk size field to determine which fields are present.

2.6 Other optional chunks

A number of other chunks are specified for use in the WAVE format. Details of these chunks are given in the specification of the WAVE format and any updates issued later.

NOTE 1 – The WAVE format can support other optional chunks that can be included in WAVE files to carry specific information. As stated in Note 1 to § 2.1 of Annex 1, in the Broadcast Wave Format File these are considered to be private chunks and will be ignored by applications which cannot interpret them.

3 Other WAVE types

The following information has been extracted from the Microsoft® Data Standards. It outlines the necessary extensions of the basic WAVE files (used for PCM audio) to cover other types of WAVE format.

3.1 General information

All newly defined WAVE types must contain both a <fact-ck> and an extended wave format description within the <fmt-ck> format chunk. RIFF WAVE files of type WAVE_FORMAT_PCM need not have the extra chunk nor the extended wave format description.

3.2 Fact chunk

This chunk stores file dependent information about the contents of the WAVE file. It currently specifies the length of the file in samples.

WAVE format extension

The extended wave format structure added to the <**fmt-ck**> is used to define all non-PCM format wave data, and is described as follows. The general extended waveform format structure is used for all non PCM formats.

```
typedef struct waveformat_extended_tag {

        WORD    wFormatTag,            /* format type */
        WORD    nChannels,             /* number of channels (i.e. mono, stereo...) */
        DWORD   nSamplesPerSec,        /* sample rate */
        DWORD   nAvgBytesPerSec,       /* for buffer estimation */
        WORD    nBlockAlign,           /* block size of data */
        WORD    wBitsPerSample,        /* number of bits per sample of mono data */
        WORD    cbSize,                /* the count in bytes of the extra size */

} WAVEFORMATEX;
```

Field	Description
wFormatTag	Defines the type of WAVE file.
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.
nSamplesPerSec	Frequency of the sample rate of the wave file. This should be 48000 or 44100 etc. This rate is also used by the sample size entry in the fact chunk to determine the duration of the data.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvgBytesPerSec**> value.
nBlockAlign	The block alignment (in bytes) of the data in <**data-ck**>. Playback software needs to process a multiple of <**nBlockAlign**> bytes of data at a

	time, so that the value of <**nBlockAlign**> can be used for buffer alignment.
wBitsPerSample	This is the number of bits per sample per channel. Each channel is assumed to have the same sample resolution. If this field is not needed, then it should be set to zero.
cbSize	The size in bytes of the extra information in the WAVE format header not including the size of the WAVEFORMATEX structure.

NOTE – The fields following the <cbSize> field contain specific information needed for the WAVE format defined in the field <wFormatTag>. Any WAVE formats which can be used in the BWF will be specified in individual Supplements to this Recommendation.

Appendix 2 to Annex 1 (Informative)

Specification of the format for <CodingHistory> field

Introduction

The <CodingHistory> field in the <bext> 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.

Syntax

The syntax of each row should be as follows:

Parameter	Variable string <allowed option>
Coding algorithm	A=<ANALOGUE, PCM, MPEG1L1, MPEG1L2, MPEG1L3, MPEG2L1, MPEG2L2, MPEG2L3>
Sampling frequency (Hz)	F=<16000,22050,24000,32000,44100,48000>
Bit-rate (kbit/s per channel)	B=<any bit-rate allowed in MPEG 2 (ISO/IEC 13818-3)>
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 $2C_{hex}$). Examples of the contents: ID-No; codec type; A/D type>

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

Variable `B=` is only used for MPEG coding.

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

Examples of coding history fields

Example 1

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

A=MPEG1L2,F=48000,B=192,W=16,M=stereo,T=PCX9,CR/LF

Interpretation of example 1

Line 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

Line 2

The original file has been converted to an MPEG-1 Layer II BWF file using the parameters:

–	Sampling frequency:	48 kHz
–	bits per second per channel:	192 kbit/s
–	Coding resolution:	16 bits
–	Mode:	stereo
–	Coder:	PCX9 (Digigram)

Example 2 for a digitization process of analogue material

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>

A=PCM,F=48000,W=16,M=stereo,T=PCX9;DIO,<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

Line 3

The recording was stored as a BWF file with linear PCM coding using the digital input of a PCX9 interface card with:

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

Appendix 3
to Annex 1
(Informative)

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

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 the ISO 3166[5] standard [ISO, 1997].
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.

Examples of USIDs

Example 1

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

```
    UDI format: CCOOOONNNNNNNNNNNNHHMMSSRRRRRRRR
    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: CCOOOONNNNNNNNNNNNHHMMSSRRRRRRRR
    UDI Example: FIYLE0xxxxxxssssss08144887724864
```

[5] ISO 3166-1:1997 Codes for the representation of names of countries and their subdivisions – Part 1: Country codes (see: http://www.din.de/gremien/nas/nabd/iso3166ma/index.html

Appendix 4
to Annex 1
(Informative)

Definition of an optional peak envelop Level chunk <levl -ck> to the BWF

When audio files are exchanged between workstations, it can speed up the opening, display and processing of a file if data is available about the peak audio signal levels in the file. The addition of a **<*levl*>** chunk to a Broadcast Wave Format (BWF) file [1] provides a standard for storing and transferring data about the signal peaks obtained by sub-sampling the audio. This data in the chunk can be used to provide the envelope of the audio essence in the file. This will allow an audio application to display the audio files quickly, without loosing too much accuracy.

In addition, it is possible to send the peak-of-peaks, which is the *first* audio sample whose absolute value is the maximum value of the entire audio file. An audio application can use this information to normalize a file in real-time without having to scan the entire file (since this has already been done by the sender).

1 Terminology

The audio signal is divided into blocks. One **peak frame** is generated for each audio block there. There are n **peak values** for each peak frame, where n is the number of peak channels. Each peak value may consist of one (positive only) or two (one positive and one negative) **peak points.**

1.1 Generation of peak values

The audio signal is divided into blocks of samples of constant size. The default, and recommended, size of the blocks is 256 samples from each channel.

The samples of each channel are evaluated to find the peak points (maximum values). It is recommended that separate peak points are found for positive and negative samples but alternatively only the absolute value (either positive or negative) may be used. All the peak points are unsigned values.

The peak points are rounded to one of two formats, either 8 or 16 bits. In most cases the 8-bit format is sufficient. The 16-bit format should cover any cases needing higher precision.

The formatted peak points for each channel are assembled into peak frames. Each peak frame contains the positive and negative peak points (or the absolute peak point) for each channel in the same order as the audio samples.

These peak frames are carried as the data in the Peak Envelope chunk. The peak envelope chunk starts with a header that contains information that allows the peak data to be interpreted.

The **peak-of-peaks** is the *first* audio sample whose absolute value is the maximum value of the entire audio file. Rather than storing the peak-of-peaks as a sample value, the *position* of the peak-of-peaks is stored. In other words, an audio sample frame index is stored. An application then knows *where* to read the peak-of-peaks in the audio file. It would be more difficult to store a value for peak since this is dependent on the binary format of the audio samples (integers, floats, double...).

NOTES:

– The header only uses DWORDs (4 byte values) or multiples of 4 bytes to avoid problems with alignment of structures in different compilers.

– The total size of the header is 128 bytes in order to avoid cache misalignment.

2 Peak envelope chunk

The peak envelope, ***<levl>,*** chunk consists of a header followed by the data of the peak points. The overall length of the chunk will be variable, depending on the audio content, the block size and how the peak data is formatted.

```
typedef struct peak_envelope
{
        CHAR        ckID[4],             /* {'l','e','v','l'} */
        DWORD       ckSize,              /* size of chunk */
        DWORD       dwVersion,           /* version information */
        DWORD       dwFormat,;           /* format of a peak point */
                                                 1 = unsigned char
                                                 2 = unsigned short
        DWORD       dwPointsPerValue,    /* 1 = only positive peak point
                                           2 = positive AND negative peak points */
        DWORD       dwBlockSize,         /* frames per value */
        DWORD       dwPeakChannels,      /* number of channels */
        DWORD       dwNumPeakFrames,     /* number of peak frames */
        DWORD       dwPosPeakOfPeaks,    /* audio sample frame index/* or
                                         0xFFFFFFFF if unknown */
        DWORD       dwOffsetToPeaks,     /* should usually be equal to the size of this header,
but
                                         could also be higher */
        CHAR        strTimestamp[28],    /* ASCII: time stamp of the peak data */
```

2.1 Elements of the "levl" chunk

ckID	This is the 4 character array {"l", "e", "v", "l"}[6], the chunk identification.
ckSize	The size of the remainder of the chunk. (It does not include the 8 bytes used by ckID and ckSize.)
dwVersion	The version of the peak_envelope chunk. It starts with 0000.
dwFormat	The format of the peak envelope data. Two formats are allowed[7]:

dwFormat	Value	Description
LEVL_FORMAT_UINT8	1	unsigned char for each peak point
LEVL_FORMAT_UINT16	2	unsigned short integer for each peak point

dwPointsPerValue	This denotes the number of peak points per peak value. This may be either one or two.

[6] The definition DWORD ckID = "levl" would not be unique. Different C-compilers produce different orders of the characters. Therefore we define char ckID[4] = {"l", "e", "v", "l"} instead.

[7] Because any audio application that supports the **"levl"** chunk would have to implement all possible formats, only two formats are allowed.
In most cases the unsigned char (8 bit) format is sufficient. The unsigned short format (16 bit) should cover any cases needing higher precision.

dwPointsPerValue = 1

Each peak value consists of one peak point. The peak point is the maximum of the absolute values of the **dwBlockSize** audio samples in each block:

max{abs(X1),...,abs(Xn)}

NOTE – In this case the displayed waveform will always be symmetrical with respect to the horizontal axis.

dwPointsPerValue = 2

Each peak value consists of two peak points. The first peak point corresponds to the highest *positive* value of the **dwBlockSize** audio samples in the block. The second peak point corresponds to the *negative* peak of the **dwBlockSize** audio samples in the block.

It is recommended to use two peak points (**dwPointsPerValue = 2)** because unsymmetrical wave forms (e.g. a DC offset) will be correctly displayed.

dwBlockSize	This is the number of audio samples used to generate each peak frame. This number is variable. The default and recommended block size is 256.
dwPeakChannels	The number of peak channels[8].
dwNumPeakFrames	The number of peak frames. The number of peak frames is the integer obtained by rounding down the following calculation:

$$\text{dwNumPeakFrames} = \frac{(\text{numAudioFrame} + \text{dwBlockSize})}{\text{dwBlockSize}}$$

or rounding up the following calculation:

$$\text{dwNumPeakFrames} = \frac{\text{numAudioFrame}}{\text{dwBlockSize}}$$

Where numAudioFrame is the number of audio samples in each channel of the audio data.

E.g. for a peak ratio (Block size) of 256, this means:

0	audio sample -> 0 peak frame
1	audio sample -> 1 peak frame
256	audio samples -> 1 peak frame
257	audio samples -> 2 peak frames
7582	audio samples -> 30 peak frames.

dwPosPeakOfPeaks	An audio application can use this information to normalize a file without having to scan the entire file. (Since it has already been done by the sender). The benefit is a performance boost as well as the possibility to normalize a file in real-time.

The **peak-of-peaks** is *first* audio sample whose absolute value is the maximum value of the entire audio file.

Rather than storing the peak-of-peaks as a sample value, the *position* of the peak of the peaks is stored. In other words, an audio sample frame index is stored. An application then knows *where* to read the peak of the peaks in the audio file. It would be more difficult to store a value for peak since this is dependent on the binary format of the audio samples (integers, floats, double).

[8] Usually the number of peak channels equals the number of audio channels. If this number is one, the same waveform will be displayed for each audio channel.

If the value is `0xFFFFFFFF`, then that means that the peak of the peaks is unknown.

dwOffsetToPeaks Offset of the peak data from the start of the header. Usually this equals to the size of the header, but it could be higher. This can be used to ensure that the peak data begins on a `DWORD` boundary.

strTimeStamp A string containing the time stamp of the creation of the peak data. It is formatted as follows:[9]

"YYYY:MM:DD:hh:mm:ss:uuu"

where:

YYYY:	year
MM:	month
DD:	day
hh:	hours
mm:	minutes
ss:	seconds
uuu:	milliseconds

Example: "2000:08:24:13:55:40:967"

2.2 Format of a peak point

A peak value is composed of one or two peak points, flagged by **dwPointsPerValue.** The flag **dwFormat** indicates the format of the numbers representing the peak points in each peak frame.

		dwPointsPerValue	
		= 1	**= 2**
dwFormat		The number corresponds to the absolute peak	The first number corresponds to the positive peak The second number corresponds to the negative peak (Note that the "negative" peak is stored as a "positive" number)
= 1	**levl_format_uint8**	*unsigned char (0...255)*	*unsigned char (0...255)* *unsigned char (0...255)*
= 2	**levl_format_uint16**	*unsigned short* (0...65535)	*unsigned short* (0...65535) *unsigned short* (0...65535)

2.3 Multichannel peak files

For multichannel audio files, the single peak values from each channel are interleaved. A set of interleaved peak values is called a peak frame. The order of the peak values inside a peak frame corresponds to the placement of the sample points inside the RIFF audio data frame.

[9] This format has the advantage that there is no limitation in time and it is easy to read. (Other formats use a DWORD denoting the seconds since 1970, which reaches its limit after about 125 years.)

2.4 Synchronization with the audio file

The peak file must be rebuilt if either of these two conditions is met:

The time stamp is older than the time stamp of the audio file.

The number of peak frames does not correspond to the number of sample frames in the audio file.

2.5 Byte order

Because the Broadcast Wave Format file (BWF), is an extension to the RIFF format, all numbers are stored as little-endian.

Appendix 5 to Annex 1 (Informative)

Definition of an optional Link chunk <link-ck> to the BWF

Introduction

The Broadcast Wave Format (BWF) File allows a maximum file size of 4 Gbytes although in practice many RIFF/Wave applications will only support a maximum file size of 2 Gigabytes. For audio data in excess of these limits it is necessary to split the audio information into more than one BWF file. The <link> chunk provides link-up data for a seamless audio output spread over several files.

1 Terminology

File-set	The set of linked files belonging to one continuous audio signal.
Filename	The names given to each file in the file-set.
File list	A list of the Filenames in the file-set.
“Actual” attribute	An attribute flagging the filename in the file list as being the current (or “actual”) file. All other filenames in the file list are flagged as “other”.
File identifier	An optional identifier which should be the same for all files of a file-set.
‘Private’ element	An additional element in the chunk to store proprietary information in the file list.
<link> chunk	A chunk contained in all the files of a file-set. It contains a header followed by a file list and optionally a file identifier and “private” element. The data in the chunk is stored in XML 1.0 format[10], a widespread format for data exchange.

[10] Extensible Markup Language (XML) 1.0 W3C Recommendation 10-February-1998 http://www.w3.org/TR/1998/REC-xml-19980210.

2 Link chunk structure

2.1 Overview

The <link> chunk consists of a header followed by the link-up information stored in XML (eXtensible Markup Language) format. The overall length of the chunk will be variable.

```
typedef struct link
{CHAR   CkID[4],          /* {'l','i','n','k'} */
          DWORD     CkSize,            /* size of chunk */
          CHAR       XmlData[ ],        /* link-up information in XML */
 }
 Link_chunk,
```

Field **Description** **ckID** This is the 4 character array {'l', 'i', 'n', 'k'}[11] for chunk identification.

CkSize This is the size of the data section of the chunk (not including the 8 bytes used by ckID and ckSize.)

XmlData This buffer contains the link-up information in XML (ASCII characters).

2.2 XML data structure in <xmlData> variable data field

The data structure is hierarchical. Data are stored in text strings. For the exact syntax specification a DTD (data transfer document) is added.

```
<LINK>
               <FILE type="…">
                        <FILENUMBER>...</FILENUMBER>
                        <FILENAME>...</FILENAME>
               </FILE>
               .......
               Possible further FILE elements
               .......
               <ID>...</ID>        optional
               <PRIVATE>       optional
               ..... implementation dependent
               </PRIVATE>
</LINK>
```

LINK This is the root element of the XML data. LINK contains one or more FILE elements with the file description. It may also contain identifier ID and/or a PRIVATE element.

ID The identifier ID is common for all files of a given file-set. It is stored as a text string of characters permitted by the #PCDATA definition of the XML 1.0 specification, which includes all visible ASCII characters, spaces etc.

PRIVATE The PRIVATE element may contain implementation-dependent information consisting of any XML data (such as further elements or #PCDATA).

FILE The FILE element contains the FILENUMBER element and the FILENAME element. The type attribute should be 'actual' in the case that the file in the list describes the file to which the chunk belongs. All other files should have the

[11] The definition DWORD ckID = "link" would not be unique. Different C-compilers produce different orders of the characters. Therefore we define char ckID[4] = {'l', 'i', 'n', 'k'} instead.

	type attribute 'other'. The filename of the file should be the same as it appears in the file list.
FILENUMBER	Files should be numbered sequentially according to their chronological order in the file-set.Integer numbers (ASCII characters) beginning with number 1 should be used.
FILENAME	Text string stored in the same format as the ID.

2.3 DTD for XML structure of the <link> chunk

The DTD (document type definition) is described in the XML 1.0 specification as a definition of the syntax of an XML structure. The format and the attributes of the different elements of the <link> chunk are described below, including sub-elements and their multiplicity.

Element LINK should contain one or more sub-elements FILE ('+' indicates one or more), it may contain a subelement ID and a sub-element PRIVATE ('?' indicates one or none).

Each element FILE should contain one sub-element FILENUMBER and one sub-element FILENAME. A type attribute should be specified, which may be either "actual" or "other".

Sub-elements FILENUMBER, FILENAME and ID must contain text strings (called #PCDATA in XML).

Sub-element PRIVATE may contain any of the defined elements. If PRIVATE needs to contain elements other than the defined ones, the DTD must be modified accordingly.

```
<!ELEMENT LINK          (FILE+, ID?, PRIVATE?)>
<!ELEMENT FILE          (FILENUMBER, FILENAME)>
<!ATTLIST FILE          type ("actual" | "other") #REQUIRED>
<!ELEMENT FILE          NUMBER (#PCDATA)>
<!ELEMENT FILE          NAME (#PCDATA)>
<!ELEMENT ID            (#PCDATA)>
<!ELEMENT PRIVATE       ANY>
```

3 Renaming of linked files

If one or more filenames is changed, the corresponding FILENAME entries in each of the <link> chunks belonging to the whole file-set should be changed.

The continuous sound signal in this example has been split into a file-set of three BWF files called "Sinatra_1.wav", "Sinatra_2.wav" and "Sinatra_3.wav". The XML structures of the <link> chunks of the three files are identical except for the type attribute.

3.1 <link> chunk of "Sinatra_1.wav"

```
<LINK>
        <FILE type="actual">
                <FILENUMBER>1</FILENUMBER>
                <FILENAME>Sinatra_1.wav</FILENAME>
        </FILE>
        <FILE type="other">
                <FILENUMBER>2</FILENUMBER>
                <FILENAME>Sinatra_2.wav</FILENAME>
        </FILE>
        <FILE type="other">
                <FILENUMBER>3</FILENUMBER>
                <FILENAME>Sinatra_3.wav</FILENAME>
        </FILE>
        <ID>73365869</ID>
</LINK>
```

3.2 <link> chunk of "Sinatra_2.wav"

```
<LINK>
        <FILE type="other">
                <FILENUMBER>1</FILENUMBER>
                <FILENAME>Sinatra_1.wav</FILENAME>
        </FILE>
        <FILE type="actual">
                <FILENUMBER>2</FILENUMBER>
                <FILENAME>Sinatra_2.wav</FILENAME>
        </FILE>
        <FILE type="other">
                <FILENUMBER>3</FILENUMBER>
                <FILENAME>Sinatra_3.wav</FILENAME>
        </FILE>
        <ID>73365869</ID>
</LINK>
```

3.3 <link> chunk of "Sinatra_3.wav"

```
<LINK>
        <FILE type="other">
                <FILENUMBER>1</FILENUMBER>
                <FILENAME>Sinatra_1.wav</FILENAME>
        </FILE>
        <FILE type="other">
                <FILENUMBER>2</FILENUMBER>
                <FILENAME>Sinatra_2.wav</FILENAME>
        </FILE>
        <FILE type="actual">
                <FILENUMBER>3</FILENUMBER>
                <FILENAME>Sinatra_3.wav</FILENAME>
        </FILE>
        <ID>73365869</ID>
<LINK>
```

Appendix 6 to Annex 1 (Normative)

Filename conventions

1 General

The general interchange of audio files mean that they must be playable on computer and operating-system types that may be quite different from the originating system. An inappropriate filename could mean that the file cannot be recognized 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.

2 File-name length

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

3 File-name extension

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

4 File-name character set

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

Character	Decimal value	Hexadecimal value
(Space)	32	0x20
…	…	…
~ (tilda)	126	0x7E

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

Character	Decimal value	Hexadecimal value
“	34	0x22
*	42	0x2A
/	47	0x2F
:	58	0x3A
<	60	0x3C
>	62	0x3E
?	63	0x3F
\	92	0x5C
|	124	0x7C

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

Character	Decimal value	Hexadecimal value
(Space)	32	0x20
(period)	46	0x2E

Annex 2

Specification of the broadcast wave format with MPEG-1 audio

A format for audio data files in broadcasting

1 Introduction

This Annex contains the specification for the use of the BWF to carry MPEG audio only signals. For MPEG audio, it is necessary to add the following information to the basic chunks specified in the main part of this Recommendation:

– an extension to the format chunk;

– a fact chunk;

– an MPEG_extension chunk.

The extension to the format chunk and the fact chunk are both specified as part of the WAVE format and the relevant information is given in Appendix 1 to Annex 2.

The specification of the MPEG_extension chunk is given in § 2 of Annex 2.

The main part of this Recommendation contains the specification of the broadcast audio extension chunk that is used in all BWF. Information on the basic RIFF format is given in Appendix 1 to Annex 2.

2 MPEG audio

Microsoft® have specified how MPEG audio data can be organized in WAVE files. An extension to the format chunk and a fact chunk carry further information needed to specify MPEG coding options. The general principles are given in Appendix 1 to Annex 1 and the details are given in Appendix 1 to Annex 2. For the MPEG Layer II, it has been found that extra information needs to be carried about the coding of the signal. This is carried in the <**MPEG Audio Extension**> chunk, developed by the MPEG Layer 2 Audio Interest group. This chunk is specified below.

2.1 MPEG audio extension chunk

The MPEG audio extension chunk is defined as follows:

```
typedef struct {
    DWORD   ckID,           /*  (mpeg_extension)ckID='mext' */
    DWORD   ckSize,         /*  size of extension chunk:
                                cksize =000C*/
    BYTE  ckData[ckSize],   /*  data of the chunk */
}

typedef struct mpeg_audio_extension {
WORD SoundInformtion,       /* more information about sound */
WORD FrameSize,             /* nominal size of a frame */
WORD AncillaryDataLength,   /* Ancillary data length */
WORD AncillaryDataDef,      /* Type of ancillary data */
CHAR Reserved [4],             "NULL"*/
} MPEG_EXT ;
```

Champ	**Description**
SoundInformation	16 bits giving additional information about the sound file: For MPEG Layer II (or Layer I): Bit 0: ‘1’ Homogeneous sound data ‘0’ Non homogeneous sound data Bits 1 and 2 are used for additional information for homogeneous sound files: Bit 1: ‘0’ Padding bit is used in the file so may alternate between ‘0’ or ‘1’ ‘1’ Padding bit is set to ‘0’ in the whole file Bit 2: ‘1’ The file contains a sequence of frames with padding bit set to ‘0’ and sample frequency equal to 22.05 or 44.1 kHz

NOTE 1 – Such a file does not comply with the MPEG standard (clause 2.4.2.3, definition of padding_bit), but can be regarded as a special case of variable bit rate. There is no need for an MPEG decoder to decode such a bitstream, as most decoders will perform this function. The bit rate will be slightly lower than that indicated in the header.

	Bit 3: ‘1’ Free format is used ‘0’ No free format audio frame.
FrameSize	16 bit number of bytes of a nominal frame. This field has a meaning only for homogeneous files, otherwise it is set to ‘0’. If the padding bit is not used, i.e. it remains constant in all frames of the sound file, the field <FrameSize> contains the same value as the field <nBlockAlign> in the format chunk. If the padding bit is used and variable lengths occur in the sound data, <FrameSize> contains the size of a frame with the padding bit set to ‘0’. The length of a frame with the padding bit set to ‘1’ is one byte more (four bytes for Layer I), i.e. <FrameSize+1>. The fact that <nBlockAlign> is set to ‘1’ means variable frame lengths (FrameSize or FrameSize+1) with variable padding bit.
AncillaryDataLength	16-bit number giving the minimal number of known bytes for ancillary data in the full sound file. The value is relative from the end of the audio frame.
AncillaryDataDef	This 16-bit value specifies the content of the ancillary data with: Bit 0 set to ‘1’: Energy of the left channel present in ancillary data Bit 1 set to ‘1’: A private byte, is free for internal use in ancillary data Bit 2 set to ‘1’: Energy of the right channel present in ancillary data Bit 3 set to ‘0’: Reserved for future use for ADR data Bit 4 set to ‘0’: Reserved for future use for DAB data Bit 5 set to ‘0’: Reserved for future use for J 52 data Bit 6 to 15 set to ‘0’: Reserved for future use

NOTES:

– The items present in the ancillary data follow the same order as the bit numbers in AncillaryDataDef. The first item is stored at the end of the ancillary data, the second item is stored just before the first, etc., moving from back to front.

– For a mono file, bit 2 is always set to ‘0’ and bit 0 concerns the energy of the mono frame.

– For a stereo file, if bit 2 equals ‘0’ and bit 0 equals ‘1’ the energy concerns the maximum of left and right energy.

– The energy is stored in 2 bytes and corresponds to the absolute value of the maximum sample used to code the frame. This is a 15-bit value in Big Endian format.

Reserved	4 bytes reserved for future use. These 4 bytes must be set to null. In any future use, the null value will be used for the default value to maintain compatibility.

Appendix 1 to Annex 2 (Informative)

RIFF WAVE (.WAV) file format

This Appendix gives the specification of the extra information necessary for a WAVE file containing MPEG Audio.

The information in this Appendix is taken from the specification documents of Microsoft® RIFF file format. It is included for information only.

1 MPEG-1 audio (audio-only)

1.1 Fact chunk

This chunk is required for all WAVE formats other than WAVE_FORMAT_PCM. It stores file dependent information about the contents of the WAVE data. It currently specifies the time length of the data in samples.

NOTE 1 – See also Appendix 1 to Annex 1, § 2.5.

1.2 WAVE format header

```
#define WAVE_FORMAT_MPEG  (0x0050)

typedef struct mpeg1waveformat_tag {
    WAVEFORMATEX          wfx;
    WORD     fwHeadLayer;
    DWORD    dwHeadBitrate;
    WORD     fwHeadMode;
    WORD     fwHeadModeExt;
    WORD     wHeadEmphasis;
    WORD     fwHeadFlags;
    DWORD    dwPTSLow;
    DWORD    dwPTSHigh;
} MPEG1WAVEFORMAT;
```

Field	**Description**
wFormatTag	This must be set to WAVE_FORMAT_MPEG. *[0x0050]*
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.
nSamplesPerSec	Sampling frequency (Hz) of the wave file: 32 000, 44 100, or 48 000 etc. Note, however, that if the sampling frequency of the data is variable, then this field should be set to zero. It is strongly recommended that a fixed sampling frequency be used for desktop applications.
nAvgBytesPerSec	Average data rate; this might not be a legal MPEG bit rate if variable bit rate coding under Layer III is used.
nBlockAlign	The block alignment (in bytes) of the data in <**data-ck**>. For audio streams that have a fixed audio frame length, the block alignment is equal to the length of the frame. For streams in which the frame length varies, <**nBlockAlign**> should be set to 1.
	With a sampling frequency of 32 or 48 kHz, the size of an MPEG audio frame is a function of the bit rate. If an audio stream uses a constant bit rate, the size of the audio frames does not vary. Therefore, the following formulas apply:
	Layer I: nBlockAlign = 4*(int)(12*BitRate/SamplingFreq)
	Layers II and III: nBlockAlign = (int)(144*BitRate/SamplingFreq)
	Example 1: For Layer I, with a sampling frequency of 32 000 Hz and a bit rate of 256 kbit/s, nBlockAlign = 384 bytes.
	If an audio stream contains frames with different bit rates, then the length of the frames varies within the stream. Variable frame lengths also occur when using a sampling frequency of 44.1 kHz: in order to maintain the data rate at the nominal value, the size of an MPEG audio frame is periodically increased by one "slot" (4 bytes in Layer I, 1 byte in Layers II and III) as compared to the formulas given above. In these two cases, the concept of block alignment is invalid. The value of <**nBlockAlign**> must therefore be set to 1, so that MPEG-aware applications can tell whether the data is block-aligned or not.

NOTE – It is possible to construct an audio stream that has constant-length audio frames at 44.1 kHz by setting the padding_bit in each audio frame header to the same value (either 0 or 1). Note, however, that the bit rate of the resulting stream will not correspond exactly to the nominal value in the frame header, and therefore some decoders may not be capable of decoding the stream correctly. In the interests of standardization and compatibility, this approach is discouraged.

WBitsPerSample	Not used; set to zero.
CbSize	The size in bytes of the extended information after the WAVEFORMATEX structure. For the standard WAVE_FORMAT_MPEG format, this is 22 (0x0016). If extra fields are added, this value will increase.
fwHeadLayer	The MPEG audio layer, as defined by the following flags:
	ACM_MPEG_LAYER1 – Layer I.
	ACM_MPEG_LAYER2 – Layer II.
	ACM_MPEG_LAYER3 – Layer III

	Some legal MPEG streams may contain frames of different layers. In this case, the above flags should be ORed together so that a driver may determine which layers are present in the stream.
dwHeadBitrate	The bit rate of the data, in bits per second. This value must be a standard bit rate according to the MPEG specification; not all bit rates are valid for all modes and layers. See Tables 1 and 2. Note that this field records the actual bit rate, not MPEG frame header code. If the bit rate is variable, or if it is a non-standard bit rate, then this field should be set to zero. It is recommended that variable bit rate coding be avoided where possible.
fwHeadMode	Stream mode, as defined by the following flags: ACM_MPEG_STEREO – stereo. ACM_MPEG_JOINTSTEREO – joint-stereo. ACM_MPEG_DUALCHANNEL – dual-channel (for example, a bilingual stream). ACM_MPEG_SINGLECHANNEL – single channel. Some legal MPEG streams may contain frames of different modes. In this case, the above flags should be ORed together so that a driver may tell which modes are present in the stream. This situation is particularly likely with joint-stereo encoding, as encoders may find it useful to switch dynamically between stereo and joint-stereo according to the characteristics of the signal. In this case, both the ACM_MPEG_STEREO and the ACM_MPEG_JOINTSTEREO flags should be set.
fwHeadModeExt	Contains extra parameters for joint-stereo coding; not used for other modes. See Table 3. Some legal MPEG streams may contain frames of different mode extensions. In this case, the values in Table 3 may be ORed together. Note that fwHeadModeExt is only used for joint-stereo coding; for other modes (single channel, dual channel, or stereo), it should be set to zero. In general, encoders will dynamically switch between the various possible mode_extension values according to the characteristics of the signal. Therefore, for normal joint-stereo encoding, this field should be set to 0x000f. However, if it is desirable to limit the encoder to a particular type of joint-stereo coding, this field may be used to specify the allowable types.
wHeadEmphasis	Describes the de-emphasis required by the decoder; this implies the emphasis performed on the stream prior to encoding. See Table 4.
fwHeadFlags	Sets the corresponding flags in the audio frame header: ACM_MPEG_PRIVATEBIT – set the private bit. ACM_MPEG_COPYRIGHT – set the copyright bit. ACM_MPEG_ORIGINALHOME – sets the original/home bit. ACM_MPEG_PROTECTIONBIT – sets the protection bit, and inserts a 16-bit error protection code into each frame. ACM_MPEG_ID_MPEG1 – sets the ID bit to 1, defining the stream as an MPEG-1 audio stream. *This flag must always be set*

explicitly to maintain compatibility with future MPEG audio extensions (i.e. MPEG-2).

An encoder will use the value of these flags to set the corresponding bits in the header of each MPEG audio frame. When describing an encoded data stream, these flags represent a logical OR of the flags set in each frame header. That is, if the copyright bit is set in one or more frame headers in the stream, then the ACM_MPEG_COPYRIGHT flag will be set. Therefore, the value of these flags is not necessarily valid for every audio frame.

dwPTSLow	This field (together with the following field) consists of the presentation time stamp (PTS) of the first frame of the audio stream, as taken from the MPEG system layer. dwPTSLow contains the 32 LSBs of the 33-bit PTS. The PTS may be used to aid in the re-integration of an audio stream with an associated video stream. If the audio stream is not associated with a system layer, then this field should be set to zero
dwPTSHigh	This field (together with the previous field) consists of the presentation time stamp (PTS) of the first frame of the audio stream, as taken from the MPEG system layer. The LSB of dwPTSHigh contains the MSB of the 33-bit PTS. The PTS may be used to aid in the re-integration of an audio stream with an associated video stream. If the audio stream is not associated with a system layer, then this field should be set to zero.

NOTE – The previous two fields can be treated as a single 64-bit integer; optionally, the dwPTSHigh field can be tested as a flag to determine whether the MSB is set or cleared.

TABLE 1

Allowable bit rates (bit/s)

MPEG frame header code	Layer I	Layer II	Layer III
‘0000’	free format	free format	free format
‘0001’	32000	32000	32000
‘0010’	64000	48000	40000
‘0011’	96000	56000	48000
‘0100’	128000	64000	56000
‘0101’	160000	80000	64000
‘0110’	192000	96000	80000
‘0111’	224000	112000	96000
‘1000’	256000	128000	112000
‘1001’	288000	160000	128000
‘1010’	320000	192000	160000
‘1011’	352000	224000	192000
‘1100’	384000	256000	224000
‘1101’	416000	320000	256000
‘1110’	448000	384000	320000
‘1111’	forbidden	forbidden	forbidden

TABLE 2

Allowable mode-bit rate combinations for Layer II

Bit rate (bit/s)	**Allowable modes**
32000	single channel
48000	single channel
56000	single channel
64000	all modes
80000	single channel
96000	all modes
112000	all modes
128000	all modes
160000	all modes
192000	all modes
224000	stereo, intensity stereo, dual channel
256000	stereo, intensity stereo, dual channel
320000	stereo, intensity stereo, dual channel
384000	stereo, intensity stereo, dual channel

TABLE 3

Mode extension

fwHeadModeExt	**MPEG frame header code**	**Layers I and II**	**Layers III**
0x0001	‘00’	sub-bands 4-31 in intensity stereo	no intensity or MS-stereo coding
0x0002	‘01’	sub-bands 8-31 in intensity stereo	intensity stereo
0x0004	‘10’	sub-bands 12-31 in intensity stereo	MS-stereo
0x0008	‘11’	sub-bands 16-31 in intensity stereo	both intensity and MS-stereo coding

TABLE 4

Emphasis field

wHeadEmphasis	**MPEG frame header code**	**De-emphasis required**
1	‘00’	no emphasis
2	‘01’	50/15 µs emphasis
3	‘10’	Reserved
4	‘11’	ITU-T Recommendation J.17

1.3 Flags used in data fields

fwHeadLayer

The following flags are defined for the <fwHeadLayer> field. For encoding, one of these flags should be set so that the encoder knows what layer to use. For decoding, the driver can check these flags to determine whether it is capable of decoding the stream. Note that a legal MPEG stream may use different layers in different frames within a single stream. Therefore, more than one of these flags may be set.

```
    #define ACM_MPEG_LAYER1         (0x0001)
    #define ACM_MPEG_LAYER2         (0x0002)
    #define ACM_MPEG_LAYER3         (0x0004)
```

fwHeadMode

The following flags are defined for the <fwHeadMode> field. For encoding, one of these flags should be set so that the encoder knows what mode to use; for joint-stereo encoding, typically the ACM_MPEG_STEREO and ACM_MPEG_JOINTSTEREO flags will both be set so that the encoder can use joint-stereo coding only when it is more efficient than stereo. For decoding, the driver can check these flags to determine whether it is capable of decoding the stream. Note that a legal MPEG stream may use different layers in different frames within a single stream. Therefore, more than one of these flags may be set.

```
    #define ACM_MPEG_STEREO         (0x0001)
    #define ACM_MPEG_JOINTSTEREO    (0x0002)
    #define ACM_MPEG_DUALCHANNEL    (0x0004)
    #define ACM_MPEG_SINGLECHANNEL  (0x0008)
```

fwHeadModeExt

Table 3 defines flags for the <fwHeadModeExt> field. This field is only used for joint-stereo coding; for other encoding modes, this field should be set to zero. For joint-stereo encoding, these flags indicate the types of joint-stereo encoding which an encoder is permitted to use. Normally, an encoder will dynamically select the mode extension which is most appropriate for the input signal; therefore, an application would typically set this field to 0x000f so that the encoder may select between all possibilities; however, it is possible to limit the encoder by clearing some of the flags. For an encoded stream, this field indicates the values of the MPEG *mode_extension* field which are present in the stream.

fwHeadFlags

The following flags are defined for the <fwHeadFlags> field. These flags should be set before encoding so that the appropriate bits are set in the MPEG frame header. When describing an encoded MPEG audio stream, these flags represent a logical OR of the corresponding bits in the header of each audio frame. That is, if the bit is set in any of the frames, it is set in the <fwHeadFlags> field. If an application wraps a RIFF WAVE header around a pre-encoded MPEG audio bit stream, it is responsible for parsing the bit stream and setting the flags in this field.

```
    #define ACM_MPEG_PRIVATEBIT     (0x0001)
    #define ACM_MPEG_COPYRIGHT      (0x0002)
    #define ACM_MPEG_ORIGINALHOME   (0x0004)
    #define ACM_MPEG_PROTECTIONBIT  (0x0008)
    #define ACM_MPEG_ID_MPEG1       (0x0010)
```

1.4 Audio data in MPEG files

The <**data chunk**> consists of an MPEG-1 audio sequence as defined by the ISO 11172 specification, Part 3 (audio). This sequence consists of a bit stream, which is stored in the data chunk as an array of bytes. Within a byte, the MSB is the first bit of the stream, and the LSB is the last bit. The data is *not* byte-reversed. For example, the following data consists of the first 16 bits (from left to right) of a typical audio frame header:

```
Syncword        ID      Layer     ProtectionBit   …
111111111111    1       10        1               …
```

This data would be stored in bytes in the following order:

```
Byte0 Byte1 ...
FF    FD    ...
```

1.4.1 MPEG audio frames

An MPEG audio sequence consists of a series of audio frames, each of which begins with a frame header. Most of the fields within this frame header correspond to fields in the MPEG1WAVEFORMAT structure defined above. For encoding, these fields can be set in the MPEG1WAVEFORMAT structure, and the driver can use this information to set the appropriate bits in the frame header when it encodes. For decoding, a driver can check these fields to determine whether it is capable of decoding the stream.

1.4.2 Encoding

A driver that encodes an MPEG audio stream should read the header fields in the MPEG1WAVEFORMAT structure and set the corresponding bits in the MPEG frame header. If there is any other information that a driver requires, it must get this information either from a configuration dialogue box, or through a driver callback function. For more information, see the Ancillary Data section, below.

If a pre-encoded MPEG audio stream is wrapped with a RIFF header, it is a function of the application to separate the bit stream into its component parts and set the fields in the MPEG1WAVEFORMAT structure. If the sampling frequency or the bit rate index is not constant throughout the data stream, the driver should set the corresponding MPEG1WAVEFORMAT fields (<nSamplesPerSec> and <dwHeadBitrate>) to zero, as described above. If the stream contains frames of more than one layer, it should set the flags in <fwHeadLayer> for all layers which are present in the stream. Since fields such as <fwHeadFlags> can vary from frame to frame, caution must be used in setting and testing these flags; in general, an application should not rely on them to be valid for every frame. When setting these flags, adhere to the following guidelines:

– ACM_MPEG_COPYRIGHT should be set if any of the frames in the stream have the copyright bit set.

– ACM_MPEG_PROTECTIONBIT should be set if any of the frames in the stream have the protection bit set.

– ACM_MPEG_ORIGINALHOME should be set if any of the frames in the stream have the original/home bit set. This bit may be cleared if a copy of the stream is made.

– ACM_MPEG_PRIVATEBIT should be set if any of the frames in the stream have the private bit set.

– ACM_MPEG_ID_MPEG1 should be set if any of the frames in the stream have the ID bit set. For MPEG-1 streams, the ID bit should always be set; however, future extensions of MPEG (such as the MPEG-2 multi-channel format) may have the ID bit cleared.

If the MPEG audio stream was taken from a system-layer MPEG stream, or if the stream is intended to be integrated into the system layer, then the PTS fields may be used. The PTS is a field in the MPEG system layer that is used for synchronization of the various fields. The MPEG PTS field is 33 bits, and therefore the RIFF WAVE format header stores the value in two fields: <dwPTSLow> contains the 32 LSBs of the PTS, and <dwPTSHigh> contains the MSB. These two fields may be taken together as a 64-bit integer; optionally, the <dwPTSHigh> field may be tested as a flag to determine whether the MSB is set or cleared. When extracting an audio stream from a system layer, a driver should set the PTS fields to the PTS of the first frame of the audio data. This may later be used to re-integrate the stream into the system layer. *The PTS fields should not be used for any other purpose*. If the audio stream is not associated with the MPEG system layer, then the PTS fields should be set to zero.

1.4.3 Decoding

A driver may test the fields in the MPEG1WAVEFORMAT structure to determine whether it is capable of decoding the stream. However, the driver must be aware that some fields, such as the <fwHeadFlags> field, may not be consistent for every frame in the bit stream. A driver should never use the fields of the MPEG1WAVEFORMAT structure to perform the actual decoding. The decoding parameters should be taken entirely from the MPEG data stream.

A driver may check the <nSamplesPerSec> field to determine whether it supports the sampling frequency specified. If the MPEG stream contains data with a variable sampling rate, then the <nSamplesPerSec> field will be set to zero. If the driver cannot handle this type of data stream, then it should not attempt to decode the data, but should fail immediately.

1.5 Ancillary data

The audio data in an MPEG audio frame may not fill the entire frame. Any remaining data is called *ancillary data*. This data may have any format desired, and may be used to pass additional information of any kind. If a driver wishes to support the ancillary data, it must have a facility for passing the data to and from the calling application. The driver may use a callback function for this purpose. Basically, the driver may call a specified callback function whenever it has ancillary data to pass to the application (i.e. on decode) or whenever it requires more ancillary data (on encode).

Drivers should be aware that not all applications will want to process the ancillary data. Therefore, a driver should only provide this service when explicitly requested by the application. The driver may define a custom message that enables and disables the callback facility. Separate messages could be defined for the encoding and decoding operations for more flexibility.

Note that this method may not be appropriate for all drivers or all applications; it is included only as an illustration of how ancillary data may be supported.

NOTE 1 – More information on the ancillary data is contained in the <**MPEG_Audio_Extension chunk**> which should be used for MPEG files conforming to the Broadcast Wave format. See Section 2 of the main body of Annex 2.

References

ISO/IEC 11173-3: MPEG 1.

ISO/IEC 13818-3: MPEG 2.

NOTE – Microsoft® documents are available at the following Internet address: http://www.microsoft.com.

Annex 3

Specification of the BWF

A format for audio data files in broadcasting

METADATA SPECIFICATIONS

1 Introduction

This Annex contains the specification for the use of the BWF to carry information on the audio material gathered and computed by a DAW (see Fig. 2). The BWF file is used as a platform-independent container for the sound signal and all the relevant metadata. The receiving archive server is able to extract the required information from the file and use it as necessary; for example, enter it into the database etc. (see Fig. 3).

FIGURE 2

Data gathering by a workstation into the BWF file

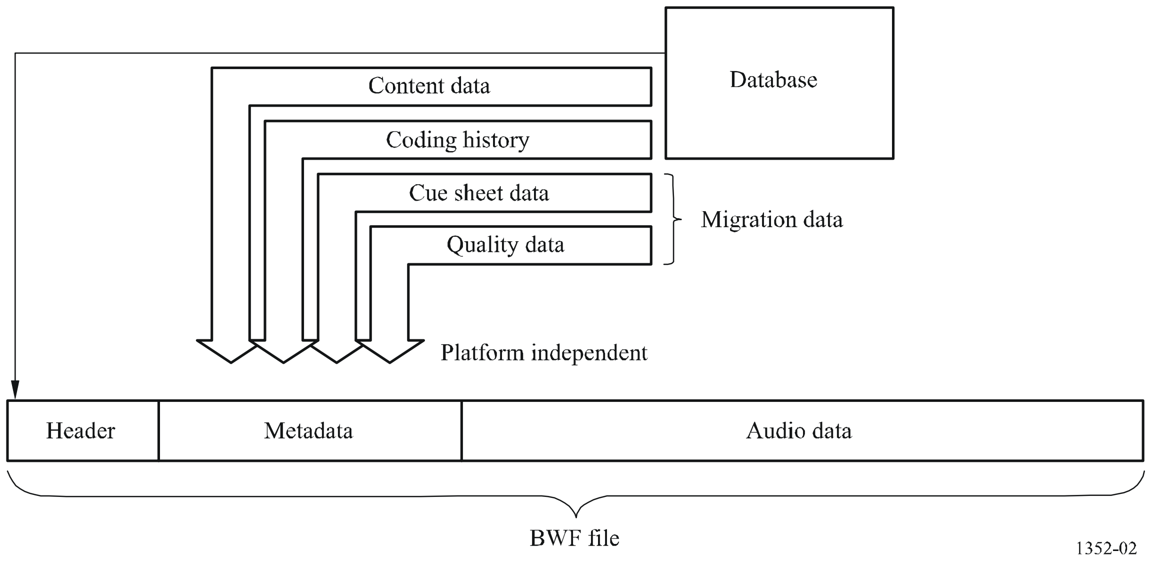
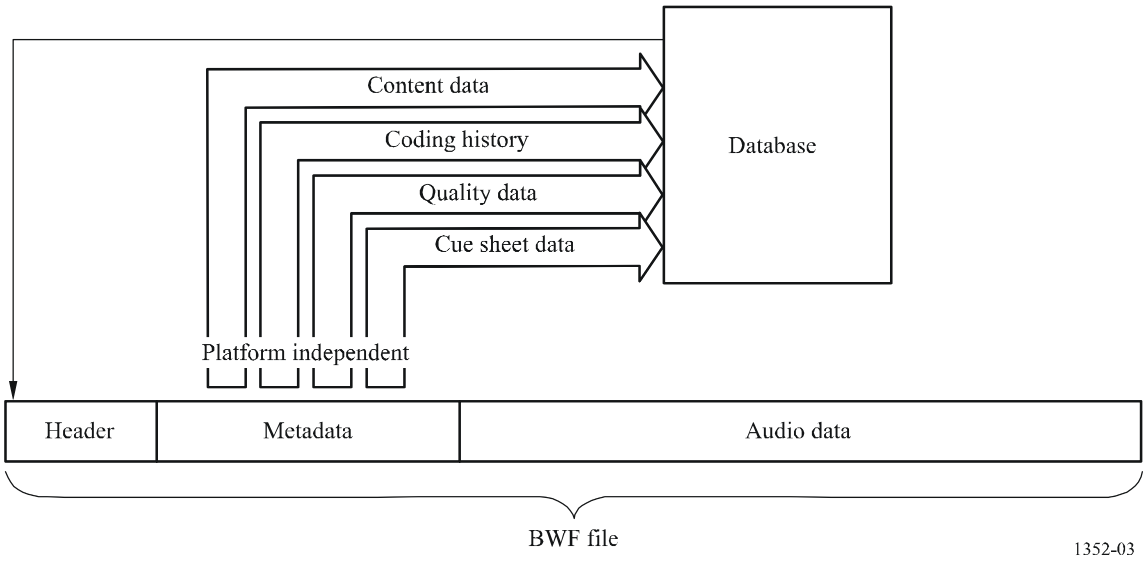


FIGURE 3

Receiving archive server extracting data from the BWF file



This Annex specifies a new chunk to carry the information not already present in a basic BWF file and also specifies how existing chunks in the BWF should be used.

Care should be taken when BWF files containing quality reports are edited. If an editing system combines more than one BWF file, the edit decision list (EDL) should point to appropriate parts of the coding history and quality chunks of each BWF source file. Furthermore, if a new file is rendered from parts of other files, a new coding history and quality chunk should be produced for the new file.

2 Capturing report

To safeguard original analogue or digital single carriers held archives, it is important to re-record the original sound signal at full quality into the BWF files. A capturing report contains information on the whole processing chain from the analogue to digital domain, or for transfers from within the digital domain (e.g. from CD or DAT).

The capturing report is laid down, together with data from the analysis of the audio signal, as part of the metadata of the BWF file.

The capturing report consists of three parts:

– CodingHistory field in the <bext> chunk of the BWF file. This contains details of the whole transmission chain, e.g. from the type of magnetic tape, compact disc or DAT cassette through to BWF file (history of the sound signal).

– The Quality Report in the <qlty> chunk. This contains information describing all relevant events affecting the quality of the recorded sound signal in the wave data chunk. Each event, whether recognized by the operator or the computer, is listed with details of the type of event, exact time stamps, priority and event status. Overall quality parameters, etc. are also reported.

– The Cue Sheet in the <qlty> chunk is a list of events marked with exact time stamps and further description of the sound signal, e.g. the beginning of an aria or the starting point of an important speech. Thus archivists are able to complete the metadata of the database with computer aided tools.

2.1 Syntax of the capturing report

– The capturing report consists of strings of ASCII (ISO 646) [ISO/IEC, 1991] characters arranged in rows of up to 256 characters.

– Each row should be terminated by <CR/LF> (ASCII 0Dh, 0Ah).

– A row may contain one or more variable strings separated by commas (ASCII 2Bh).

– Variable strings are in ASCII characters and should contain no commas.

– Semicolons (ASCII 3Bh) should be used as separators within variable strings.

3 CodingHistory field in the <bext> chunk

The strings used in the coding history field are specified in Appendix 2 to Annex 1. This information is repeated below for convenience.

A=<ANALOGUE, ………..>	Information about the analogue sound signal path
A=<PCM, ………………...>	Information about the digital sound signal path
F=<48000, 441000, etc.>	Sampling frequency [Hz]
W=<16, 18, 20, 22, 24, etc.>	Word length [bits]
M=<mono, stereo, 2-channel>	Mode
T=<free ASCII-text string>	Text for comments

4 Quality Chunk

The Quality Chunk is defined in the italic text in § 4.1:

4.1 Elements of the Quality Chunk

FileSecurityReport:	This field contains the FileSecurityCode of QualityChunk. It is a 32-bit value which contains the checksum [0 ....231].
FileSecurityWave:	This field contains the FileSecurityCode of BWF Wave data. It is a 32-bit value which contains the checksum [0 ....231].

```
Quality-chunk typedef struct {
          DWORD      ckID;                /* (quality-chunk) cklD='qlty' */
          DWORD      ckSize;              /* size of quality chunk */
          BYTE       ckData[ckSize];      /* data of the chunk */
}

typedef struct quality_chunk {
DWORD FileSecurityReport;                 /* FileSecurityCode of quality report */
DWORD FileSecurityWave;                       /* FileSecurityCode of BWF wave data */
CHAR BasicData[ ];                            /* ASCII: « Basic data » */
CHAR StartModulation[]  ;                 /* ASCII: « Start modulation data » */
CHAR QualityEvent[ ];                         /* ASCII: « Quality event data » */
CHAR EndModulation[];                     /* ASCII: « End modulation data » */
CHAR QualityParameter[ ]                      /* ASCII: « Quality parameter data » */
CHAR OperatorComment[ ];                  /* ASCII: « Comments of operator » */
CHAR CueSheet[ ];                         /* ASCII: « Cue sheet data » */
} quality-chunk
```

BasicData:	Basic data of capturing.
B=	ASCII string containing basic data about the sound material.

Archive No. (AN): Archive number (maximum 32 characters).

Title (TT): Title/Take of the sound data (maximum 256 characters).

Duration (TD): 10 ASCII characters containing the time duration of the sound sequence.

Format: « hh:mm:ss:d »

Hours	hh:	0…23
Minutes	mm:	0…59
Seconds	ss:	0…59
1/10s	d:	0…9

Date (DD): 10 ASCII characters containing the date of digitization.

Format: « yyyy:mm:dd »

Year	yyyy:	0000...9999
Month	mm:	0...12
Day	dd:	0…31

Operator (OP): ASCII string (maximum 64 characters) containing the name of the person carrying out the digitizing operation.

Copying station (CS): ASCII string (maximum 64 characters) containing the type and serial No. of the workstation used to create the file.

StartModulation: Start of modulation (SM) of the original recording.

SM= 10 ASCII characters containing the starting time of the sound signal from the start of the file.

Format: « hh:mm:ss:d »

Hours	hh:	0…23
Minutes	mm:	0…59
Seconds	ss:	0…59
1/10 s	d:	0…9

Sample count (SC): Sample address code of the SM point from the start of the file (hexadecimal start of modulation).

Format: « ########H »

0H….. FFFFFFFFH (0….. 4.295×10^9)

Comment (T): ASCII string containing comments.

OualityEvent Information describing each quality event in the sound signal. One QualityEvent string is used for each event.

Q= ASCII string (maximum 256 characters) containing quality events.

Event number (M): Numbered mark originated manually by operator.

Format: « M### » ###: 001...999

Event number (A): Numbered mark originated automatically by system.

Format: « A### » ###: 001…999

Priority (PRI): Priority of the quality event

Format: « # » #: 1 (LO)…… 5 (HI)

Time stamp (TS): 10 ASCII characters containing the time stamp of the quality event from the start of the file.

Format: « hh:mm:ss:d »

Hours	hh:	0…23
Minutes	mm:	0…59
Seconds	ss:	0…59
1/10 s	d:	0…9

Event type (E):	ASCII string (maximum 16 characters) describing the type of event, e.g. “Click”, “AnalogOver”, “Transparency” or QualityParameter (defined below) exceeding limits, e.g. “QP:Azimuth:L-20.9smp”.
Status (S):	ASCII string (maximum 16 characters) containing the processing status of the event, e.g. “unclear”, “checked”, “restored”, “deleted”.
Comment (T):	ASCII string containing comments.
Sample count (SC):	Sample address code of the TS point from the start of the file (hexadecimal ASCII). Format: « ########H » 0H…… FFFFFFFFH (0…… 4.295 $\times$ 10^9)
QualityParameter	Quality parameters (QP) describing the sound signal.
P=	ASCII string (maximum 256 characters) containing quality parameters.

Parameters (QP):	MaxPeak:	–xx.x dBFSL;–yy.y dBFSR	[–99.9…–00.0]
	MeanLevel:	–xx.x dBFSL;–yy.y dBFSR	[–99.9…–00.0]
	Correlation:	±x.x	[–1.0….…+1.0]
	Dynamic:	xx.x dBL; yy.y dBR (Dynamic range)	[00.0….… 99.9]
	ClippedSamples:	xxxx smpL; yyyy smpR	[0…..……9999]
	SNR:	xx.x dBL; yy.y dBR (Signal-to-noise-ratio)	[00.0….….99.9]
	Bandwidth:	xxxxx HzL; yyyyy HzR	[0……...20000]
	Azimuth:	L±xx.x smp	[–99.9.…+99.9]
	Balance:	L±x.x dB	[–9.9…….+9.9]
	DC-Offset:	x.x %L; y.y %R	[0.0………..9.9]
	Speech:	xx.x%	[0.0………99.9]
	Stereo:	xx.x%	[0.0………99.9]
		(L = left channel, R = right channel)	

Quality factor (QF):	Summary quality factor of the sound file [1…… 5 (best), 0 = undefined]
Inspector (IN):	ASCII string (maximum 64 characters) containing the name of the person inspecting the sound file.

File status (FS):	ASCII character string describing the status “Ready for transmission?”.
	[Y(es) / N(o) / U: File is ready/not ready/FS is undefined].
OperatorComment	Operator comments.
T=	ASCII string (maximum 256 characters) containing comments.
EndModulation	End of modulation.
EM=	10 ASCII characters containing the end of modulation time of the sound signal.
	Format: « hh:mm:ss:d »

Hours	hh:	0…23
Minutes	mm:	0…59
Seconds	ss:	0…59
1/10 s	d:	0…9

Sample count (SC):	Sample address code of the EM point (hexadecimal ASCII).
	Format: « ########H »
	0H……FFFFFFFFH (0……4.295×10^9)
Comment (T):	ASCII string containing comments.
CueSheet	Cue sheet data
C=	ASCII string (maximum 256 characters) containing cue points.
Cue number (N):	Number of cue point automatically originated by the system.
	Format: «N###» ###: 001...999
Time stamp (TS):	10 ASCII characters containing the time stamp of the cue point.
	Format: « hh:mm:ss:d »

Hours	hh:	0…23
Minutes	mm:	0…59
Seconds	ss:	0…59
1/10 s	d:	0…9

Text (T):	ASCII string containing describing comments of the cue point
	e.g. “Beginning of an aria”.
Sample count (SC):	Sample address code of the TS point (hexadecimal ASCII)
	Format: « ########H »
	0H…… FFFFFFFFH (0…4.295×10^9)

5 Examples of capturing reports

5.1 Digitization process of analogue material

(basic information contained in CodingHistory field of the <bext> chunk)

Line

01 A=ANALOGUE, M=stereo, T=Studer A816; SN1007; 38; No./telcom; Agfa PER528 <CR/LF>

02 A=PCM, F=48000, W=18, M=stereo, T=NVision NV 1000; A/D<CR/LF>

03 A=PCM, F=48000, W=16, M=stereo, T=nodither; DIO<CR/LF>

(QualityReport in the quality chunk)

Line No.

01 <FileSecurityReport>

02 <FileSecurityWave>

03 B=CS=QUADRIGA2.0; SN10012, OP=name of operator<CR/LF>

04 B=AN=archive number, TT=title of sound<CR/LF>

05 B=DD= yyyy:mm:dd, TD=hh:mm:ss:d<CR/LF>

06 SM=00:00:04:5, T=tape noise changing to ambience, SC=34BC0H<CR/LF>

07 Q=A001, PRI=2, TS=00:01:04:0, E=Click, S=unclear, SC=2EE000H<CR/LF>

08 Q=A002, PRI=3, TS=00:12:10:3, E=DropOut, S=checked, SC=216E340H<CR/LF>

09 Q=A003, PRI=4, TS=00:14:23:0, E=Transparency, S=checked, SC=2781480H<CR/LF>

10 Q=M004, PRI=1, TS=00:18:23:1, E=PrintThrough, S=checked, SC=327EF40H<CR/LF>

11 Q=A005, PRIG, TS=00:20:01:6, E=Click0n, S=unclear, T=needs restoration, SC=3701400H<CR/LF>

12 Q=A006, PRI=5, TS=00:21:20:3, E=QP:Azimuth:L=–20.9smp, S=unclear, SC=3A9B840H<CR/LF>

13 Q=A007, PRI=3, TS=00:21:44:7, E=AnalogOver, S=checked, SC=3BB9740H<CR/LF>

14 Q=A008, TS=00:22:11:7, E=C1ickOff, SC=3BB9740H<CR/LF>

15 Q=A009, PRI=1, TS=00:28:04:0, E=DropOut, S=deleted, SC=4D16600H<CR/LF>

16 EM=00:39:01:5, T=fade-out of applause, SC=6B2F740H<CR/LF>

17 P=QP:MaxPeak:–2. 1dBFSL;–2.8dBFSR<CR/LF>

18 P=QP:MeanLevel:–11.5dBFSL; 8.3dBFSR<CR/LF>

19 P=QP:Correlation:+0.8<CR/LF>

20 P=QP:Dynamic:51.4dBL;49.6dBR<CR/LF>

21 PAP:ClippedSamples:OsmpL;OsmpR<CR/LF>

22 P=QP:SNR:32.3dBL;35.1dBR<CR/LF>

23 P=QP:Bandwidth:8687HzL;7943HzR<CR/LF>

24 P=QP:Azimuth:L–6.2smp<CR/LF>

25 P=QP:Balance L:+2.1dB<CR/LF>

26 P=QP:DC-Offset:0.0%L;0.0%R<CR/LF>

27 P=QP:Speech:64.2%<CR/LF>

28 P=QP:Stereo:89.3%<CR/LF>

29 P=QF=2<CR/LF>

30 P=IN=name of inspector<CR/LF>

31 P=FS=N<CR/LF>

(CueSheet in the quality chunk)

Line No.

32 C=N001, TS=00:17:02:5, T=beginning of speech, SC=2ECE6C0 H<CR/LF>

33 C=N002, TS=00:33:19:2, T=start of aria, SC=5B84200H<CR/LF>

Interpretation of Example 1

(basic information in the CodingHistory)

Line 1: The analogue magnetic tape type Agfa PER528 is played back on a tape recorder Stude A816 with serial No. 1007 using a telcom expander:

Tape speed:	38 cm/s
Mode:	stereo

Line 2: For the digitization an A/D converter type NVision NV 1000 is used with:

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

Line 3: The original file is recorded as a linear BWF file with PCM coding using the digital input of the re-recording station without dithering:

Sampling frequency:	48 kHz
Coding resolution:	16 bits per sample
Mode:	stereo

(QualityReport in the quality chunk)

Line 1 to 2: File security codes of quality chunk and wave data.

Line 3 to 5: Re-recording station QUADRIGA2.0 with serial No. 10012 is used by the operator

(OP). The tape has the archive number (AN) and the title (TT) and was digitized on date.

(DD). The duration of the sound signal in the BWF file is (TD).

Line 6: Start of modulation (SM) at time stamp (TS) and sample count (SC) with comment (T).

Line 7 to 15: Events (E) recognized by operator (M) and/or system control (A) with priority (PRI) and at time stamp (TS). The event status (S) and comments (T) give further information. The sample count (SC) gives the precise time stamp.

Line 16: End of modulation (EM) at time stamp and sample count (SC) with comment (T).

Line 17 to 28: Quality parameters (QP) of the complete sound signal in the wave data chunk.

Line 29 to 31: Summary quality factor (QF) given by the automatic system control and the name of the inspector (IN), and the decision (FS) whether the quality of the sound file is “ready for transmission”.

(CueSheet in the quality chunk)

Line 32 to 33: Cue points mark the beginning of a speech and the starting point of an aria.

5.2 Capturing process of a compact disc

(basic information in CodingHistory field of the <bext> chunk)

Line No.

01 A=PCM, F=44100, W=16, M=stereo, T=SonyCDP-D500; SN2172; Mitsui CD-R74<CR/LF>

02 A=PCM, F=48000, W=24, M=stereo, T=DCS972; D/D<CR/LF>

03 A=PCM, F=48000, W=24, M=stereo, T=nodither;DIO<CR/LF>

(QualityReport in the quality chunk)

Line No.

01 <FileSecurityReport>

02 <FileSecurityWave>

etc: similar to the example in § 5.1 above.

(CueSheet in the quality chunk)

similar to the example in § 5.1 above.

Interpretation of Example 2

(basic information in the CodingHistory)

Line 1: A CD recordable type Mitsui CD-R74 is played back on a CD player Sony CDP-D500 with serial No. 2172:

Sampling frequency:	44.1 kHz
Coding resolution:	16 bits per sample
Mode:	stereo

Line 2: A sample rate converter type DCS972 is used with:

Sampling frequency:	48 kHz (from 44.1 kHz)
Coding resolution:	24 bits per sample
Mode:	stereo

Line 3: The original file is recorded as a linear BWF file with PCM coding using the digital input of the re-recording station without dithering:

Sampling frequency:	48 kHz
Coding resolution:	24 bits per sample
Mode:	stereo

(QualityReport in the quality chunk)

Line 1 to 2: File security codes of quality chunk and wave data.

The other data are used according to the CD capturing process similar to Example 1 in § 5.1 above.

(CueSheet in the quality chunk)

The cue sheet data are used according to the compact disc capturing process similar to Example 1 in § 5.1 above.

5.3 Capturing process of a DAT cassette

(basic information in CodingHistory field of the <bext> chunk)

Line No.

01 A=PCM, F=48000, W=16, M=stereo, T=SonyPCM-8500; SN1037; TDKDA-R120 <CR/LF>

02 A=PCM, F=48000, W=16, M=stereo, T=no dither; DIO<CR/LF>

(QualityReport in the quality chunk)

Line No.

01 <FileSecurityReport>

02 <FileSecurityWave>

etc: similar to the example in § 5.1 above.

(CueSheet in the quality chunk)

similar to the example in § 5.1 above.

Interpretation of Example 3

(basic information in the CodingHistory)

Line 1: A DAT cassette type TDK DA-8120 is played back on a DAT recorder Sony PCM-8500 with serial No. 1037:

Sampling frequency:	48 kHz
Coding resolution:	16 bits per sample
Mode:	stereo

Line 2: The original file is recorded as a linear BWF file with PCM coding using the digital input of the re-recording station without dithering:

Sampling frequency:	48 kHz
Coding resolution:	16 bits per sample
Mode:	stereo

(QualityReport in the quality chunk)

Line 1 to 2: File security codes of quality chunk and wave data.

The other data are used according to the DAT cassette capturing process similar to Example 1 in § 5.1 above.

(CueSheet in the quality chunk)

The cue sheet data are used according to the DAT cassette capturing process similar to Example 1 in § 5.1 above.