BWF

— a format for audio data files in broadcasting

Version 1

July 2001

EBU Broadcast
Wave Format

European Broadcasting Union
Case postale 45
Ancienne Route 17A
CH-1218 Grand-Saconnex
Geneva, Switzerland
techreview@ebu.ch

EBU·UER

Summary

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

As well as the audio data, a BWF file contains the minimum informationm – or metadata – that is considered necessary for all broadcast applications. The Broadcast Wave Format is based on the Microsoft WAVE audio file format. The EBU has added a "Broadcast Audio Extension" chunk to the basic WAVE format.

BWF Version 0

The specification of the Broadcast Wave Format for PCM audio data (now referred to as Version 0) was published in 1997 in EBU document Tech 3285. It is now replaced by Version 1 [see below].

BWF Version 1

Version 1 differs from Version 0 only in that 64 of the 254 reserved bytes in Version 0 are used to contain an SMPTE UMID [1] and the field is changed.

1. SMPTE 330M-2000: Television - Unique Material Identifier (UMID).

EBU Broadcast Wave Format

July 2001

Contents

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 RIFF files is a "chunk" that contains specific information, an identification field and a size field. A RIFF file consists of a number 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*.

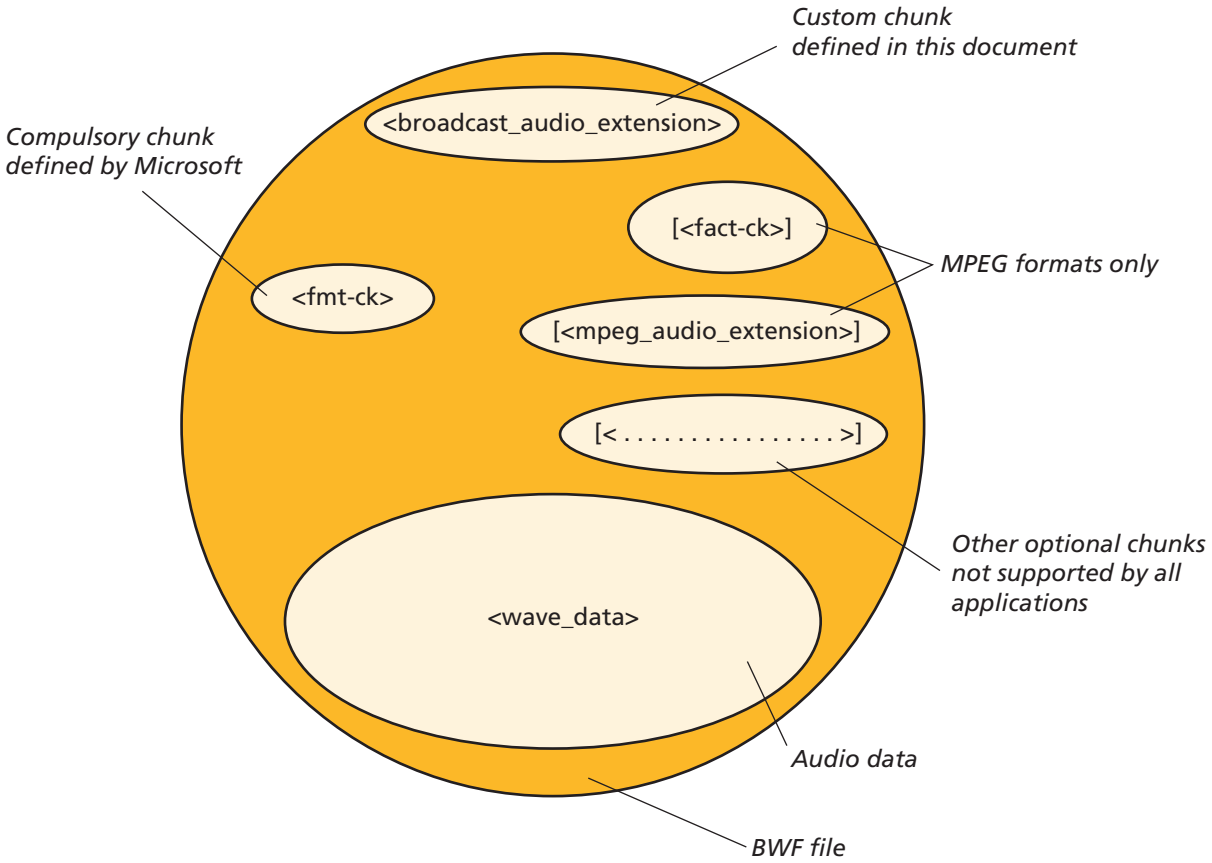

Figure 1
Broadcast Wave File format.

This document contains the specification of Version 1 the broadcast audio extension chunk, which is used in all BWF files. In addition, basic information on the RIFF format and how it can be extended to other types of audio data is given in *Appendix A*. Details of the PCM Wave format are also given in *Appendix A*. Detailed specifications of the extension to specific other types of audio data will be published in Supplements to this document.

BWF Version 0

Version 0 of the BWF was published in 1997. It has now been replaced by Version 1.

BWF Version 1

Version 1 of the BWF file differs from Version 0 only in that 64 of the 254 reserved bytes in Version 0 are used to contain an SMPTE UMID (Unique Material Identifier) and the <Version> field has been changed accordingly.

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

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

Chapter 1
Broadcast Wave Format file

1.1. 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'
        <broadcast_audio_extension>     //information on the audio sequence
        <fmt-ck>                        //Format of the audio signal: PCM/MPEG
        [<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
```

Note: *Any additional types of chunks that are present in the file have to be considered as private. Applications are not required to interpret or make use of these chunks. Thus the integrity of the data contained in any chunks not listed above is not guaranteed. However, BWF applications should pass on these chunks whenever possible.*

1.2. Existing chunks defined as part of the RIFF standard

The RIFF standard is defined in documents issued by the Microsoft Corporation [1]. This application uses a number of chunks which are already defined. These are:

```
fmt-ck
fact-ck
```

The current descriptions of these chunks are given for information in *Appendix A*.

1.3. Broadcast Audio Extension chunk

Extra parameters needed for the 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» */
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 */
BYTE Reserved[190] ;                    /* 190 bytes, reserved for future use, set to "NULL" */
CHAR CodingHistory[];                   /* ASCII : « History coding » */
} BROADCAST_EXT
```

1.3.1. Terminology

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 résumé of the description is contained in the first 64 characters, and the last 192 characters are use for details.

If the length of the string is less than 256 characters, the last one is followed by a null character. (00)

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.

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 by a null character.

Note: *The EBU has defined a format for the OriginatorReference field. See EBU Recommendation R99-1999 [2].*

OriginationDate — Ten ASCII characters containing the date of creation of the audio sequence. The format is "yyyy-mm-dd" (year-month-day).

Year is defined from 0000 to 9999

Month is define from 1 to 12

Day is defined from 1 to 28,29,30 or 31

The separator between the items can be anything but it is recommended that one of the following characters is used:

'-' hyphen '_' underscore ':' colon ' ' space '.' stop

OriginationTime

Eight ASCII characters containing the time of creation of the audio sequence. The format is “hh-mm-ss” (hours-minutes-seconds).

Hour is defined from 0 to 23.

Minute and second are defined from 0 to 59.

The separator between the items can be anything but it is recommended that one of the following characters is used:

‘-‘ hyphen ‘_’ underscore ‘:’ colon ‘ ’ space ‘.’ stop

TimeReference

This field contains the timecode 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 which is defined in the field <nSamplesPerSec> from the <**format chunk**>.

Version

An unsigned binary number giving the version of the BWF, particularly the contents of the Reserved field. For Version 1, this is set to 0001h.

UMID

64 bytes containing a UMID (Unique Material Identifier) to the SPMTE 330M standard [3]. If only a 32 byte “basic UMID” is used, the last 32 bytes should be set to zero. (The length of the UMID is given internally.)

Note: *The EBU intends to publish guidance on the use of the UMID in audio files.*

Reserved

190 bytes reserved for extensions. If the Version field is set to 0001h, these 190 bytes must be set to a NULL (zero) value.

CodingHistory

Non-restricted ASCII characters, containing a collection of strings terminated by CR/LF. Each string contains a description of a coding process applied to the audio data. Each new coding application is required to add a new string with the appropriate information.

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 : sample 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.

Note: *The EBU has defined a format for CodingHistory which will simplify the interpretation of the information provided in this field.*
See EBU Recommendation R98-1999 [4].

1.4. Other information specific to applications

The EBU will define other chunks to carry or point to data which are specific to certain applications, e.g. for edited audio or for archival.

See the EBU website for details [5].

Bibliography

[1] MSDN online library: http://www.msdn.microsoft.com/library/default.asp

[2] EBU R 99-1999: **"Unique" Source Identifier (USID) for use in the OriginatorReference field of the Broadcast Wave Format**.

[3] SMPTE 330M-2000: **Television - Unique Material Identifier (UMID)**.

[4] EBU R 98-1999: **Format for CodingHistory field in Broadcast Wave Format files, BWF**.

[5] EBU website: http://www.ebu.ch/pmc_bwf.html.

[6] Microsoft Software Developers Kit, Multimedia Standards Update, rev 3.0, 15 April 1994.

Appendix A
RIFF WAVE (.WAV) file format

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

For full information on the RIFF file format, consult the latest version of the Microsoft Software Developers Kit [6].

In this appendix, EBU explanatory notes are given in a blue-coloured text box.

A.1. Waveform Audio File Format (WAVE)

The WAVE format 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
        [<fact-ck>]                 // Fact chunk
        [<other-ck>]                // Other optional chunks
        <wave-data>                 // Wave data
```

The WAVE chunks are described in the following sections.

A.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' chunk, and the interpretation of the waveform data, depend on this value.
nchannels	The number of channels represented in the waveform data, such as 1 for mono or 2 for stereo.
nSamplesPerSec	The sampling rate (in samples per second) at which each channel should be played.
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.

A.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	(0 x 0001)	Microsoft Pulse Code Modulation (PCM) format
WAVE_FORMAT_MPEG	(0 x 0050)	MPEG-1 Audio (audio only)

EBU Note 1: *Although other WAVE formats are registered with Microsoft, only the above formats are at present used with the BWF. Details of the PCM WAVE format are given in Section A.2 of this appendix. General information on other WAVE formats is given in Section A.3.*

Details of the MPEG WAVE format are given in Supplement 1 to this document. Other WAVE formats may be defined in future Supplements.

A.2. Pulse Code Modulation (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 pulse code modulation (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.

For PCM data, the <nAvgBytesPerSec> field of the "fmt" chunk should be equal to the following formula rounded up to the next whole number:

$$\frac{\text{nChannel x nSamplesPerSecond x nBitsPer Sample}}{8}$$

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

$$\frac{\text{nChannel x nBitsPer Sample}}{8}$$

EBU Note 2: *The above formulae do not always give the correct answer. Strictly speaking, the number of bytes per sample (nBitsPerSample/8) should be rounded first. Then this integer should be multiplied by <nChannels> (which is always an integer) to give <nBlockAlign>. This in turn should be multiplied by <nSamplesPerSecond> to give <nAvgBytesPerSec>.*

A.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 channel, and channel 1 represents the right channel. The speaker position mapping for more than two channels is currently undefined. In multiple-channel WAVE files, samples are interleaved.

The following diagrams show the data packing for 8-bit and 16-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)

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

A.2.2. Data format of the samples

Each sample is contained in an integer ***i***. The size of ***i*** is the smallest number of bytes required to contain the specified sample size. The least significant byte 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 is 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 (0 x FF)	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 (0 x FF)	0	128 (0 x 80)
16-bit PCM	32767 (0 x 7FFF)	-32768 (-0 x 8000)	0

A.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> ) )
```

A.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> )
```

A.2.5. Fact chunk

The <fact-ck> 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
```

This 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.

A.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

EBU Note 3: *The WAVE format can support other optional chunks which can be included in WAVE files to carry specific information. As stated in the note to Section 1.1 (see Chapter 1), these are considered to be private chunks in the Broadcast Wave Format and will be ignored by applications which cannot interpret them.*

A.3. Other WAVE types

EBU Note 4: *The following information has been extracted from the Microsoft Software Developers Kit [6]. It outlines the necessary extensions of the basic WAVE files (used for PCM audio) to cover other types of WAVE format.*

A.3.1. General information

All newly-defined WAVE types must contain both a <**fact chunk>** 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.

A.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.

See *Section A.2.5.* in this appendix.

A.3.3. 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;
```

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

Field	**Notes**
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 length in time of the data.
nAvgBytesPerSec	Average data rate. Playback software can estimate the buffer size using the <**nAvg-BytesPerSec**> 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 data. 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.

EBU Note 5: *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 document, published by the EBU.*