RECOMMENDATION ITU-R BR.1352

FILE FORMAT FOR THE EXCHANGE OF AUDIO PROGRAMME MATERIALS ON INFORMATION TECHNOLOGY MEDIA

(Question ITU-R 215/10)

(1998)

The ITU Radiocommunication Assembly,

considering

a) that storage media based on Information Technology, including data disks and tapes, are expected to penetrate 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 equipments 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 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 various multichannel formats are the subject of Recommendation ITU-R BS.775 and that they are expected to be widely used in the near future;

h) 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;

j) that several world broadcasters have already agreed on the adoption of a common file format for programme exchange;

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

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[1] of audio programmes in linear 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].

[1] The adoption of the recommended file format, not only for interchange but also as the file format for the recorded medium, would be a preferable solution for the users as no overhead both in terms of time and temporary storage space would be required for conversion when transferring the audio signal between equipments. However, it is recognized that a recommendation in that sense could penalize developers using some computer platforms.

[2] Other annexes can be defined in future, to extend the file format to carry audio signals coded with other systems of interest to the broadcasters.

ANNEX 1

Specification of the broadcast wave format
A format for audio data files in broadcasting

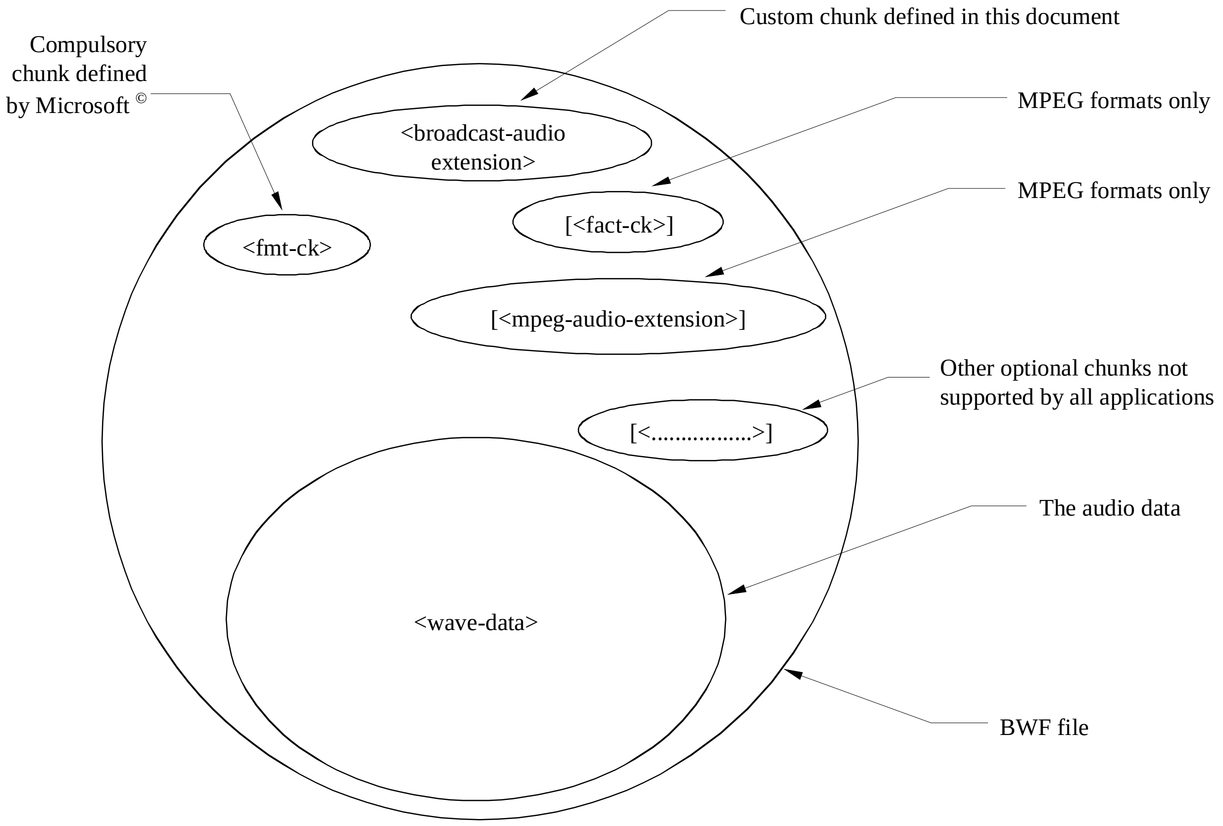
1 Introduction

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 Figure 1 below.

FIGURE 1

Broadcast wave format file


Custom chunk defined in this document
Compulsory chunk defined by Microsoft ©
MPEG formats only
MPEG formats only
<broadcast-audio extension>
[<fact-ck>]
<fmt-ck>
[<mpeg-audio-extension>]
Other optional chunks not supported by all applications
[<.................>]
The audio data
<wave-data>
BWF file

Temp 10-11/33-01

This document contains the specification of the broadcast audio extension chunk which 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 the appendix. Details of the PCM wave format are also given in the appendix. Detailed specifications of the extension to other types of audio data will be published in other annexes to this Recommendation.

2 Broadcast wave format 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’
        <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.

2.2 Existing chunks defined as part of the RIFF standard

The RIFF standard is defined in documents issued by the Microsoft Corporation. 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 1 to Annex 1.

2.3 Broadcast audio extension chunk

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»*/
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 */
CHAR Reserved[254] ;            /* 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.
Originator	If the length of the string is less than 256 characters the last one is followed by a null character. (00)
OriginatorReference	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.
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 can be anything but it is recommended that one of the following characters be used:
	‘-‘ hyphen ‘_’ underscore ‘:’ colon ‘ ’ space ‘.’ stop
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 can be anything but it is recommended that one of the following characters be used:
	‘-‘ hyphen ‘_’ underscore ‘:’ colon ‘ ’ space ‘.’ stop
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 which is defined in the field <nSamplesPerSec> from the <**format chunk**>.
Version	An unsigned binary number giving the version of the BWF. Initially this is set to zero.
Reserved	254 bytes reserved for extension. These 254 bytes must be set to a NULL value. In the future the null value will be used as a default to maintain compatibility.
CodingHistory	Non-restricted ASCII characters containing a collection of strings terminated by CR/LF. Each string contains a description of the coding process applied. 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: 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.

NOTE – Studies are under way to propose a format for coding history which will simplify the interpretation of the information provided in this field.

2.4 Other information specific to applications

Studies are under way to define other chunks to carry or point to data which are specific to certain applications, e.g. for edited audio or for archival.

APPENDIX (TO ANNEX 1)

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.

For full information, consult the latest version of the Microsoft® Software Developers Kit Multimedia Standards Update, (Rev. 3.0, 15 April 1994 or later).

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
                    [<fact-ck>]                           // Fact chunk
                    [<other-ck>]                          // Other optional chunks
                    <wave-data> )                         // Wave 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' 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.

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 Pulse Code Modulation (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 Section 3. Details of MPEG WAVE format are given in Annex 2. Other WAVE formats may be defined in future.

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:

$$\frac{\text{nChannels x nBitsPerSecond x nBitsPerSample}}{8}$$

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

$$\frac{\text{nChannels x nBitsPerSample}}{8}$$

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. 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 a 8-bit mono and stereo WAVE files:

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

Data packing for 8-bit mono PCM

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

Data packing for 8-bit stereo PCM

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

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 mono 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

Data packing for 16-bit stereo PCM

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 – 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 2.1 of Annex 1, in the Broadcast Wave Format 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: Update April 15 1994. 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 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.

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 defined 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	**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 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.

REFERENCES

Microsoft® Resource Interchange File Format, RIFF.

Microsoft® Software Developers Kit Multimedia Standards Update, Rev. 3.0, 15 April 1994.

ANNEX 2

Specification of the broadcast wave format A format for audio data files in broadcasting

SPECIFICATIONS FOR USE WITH MPEG AUDIO

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 document:

- 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 Section 2 of Annex 2.

The main part of this document contains the specification of the broadcast audio extension chunk which 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 2, 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 SoundInformation;       /* 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];           /* Reserved for future use; set to null */
} MPEG_EXT ;
```

Field	**Description**
SoundInformation	16 bits giving additional information about the sound file:

For MPEG Layer II (or Layer I):

Bit 0:	‘1’	Homogeneous sound data
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 – 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)

RIFF WAVE (.WAV) file format

This section 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.

For full information, consult the latest version of the Microsoft© Software Developers Kit Multimedia Standards Update, (Rev. 3.0, 15 April 1994 or later).

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 – See also Appendix 1 to Annex 1 Section 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. *[0x00 50]*
nChannels	Number of channels in the wave, 1 for mono, 2 for stereo.
nSamplesPerSec	Sampling frequency (in 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 which 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 kbits/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 which 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 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 (0x00 16). 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 (bits/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 (bits/sec)	**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’	CCITT 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 layer [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 which 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 which 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 presentation time stamp (PTS) fields may be used. The PTS is a field in the MPEG system layer which 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 which 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 - 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

Microsoft® Resource Interchange File Format, RIFF.

Microsoft® Software Developers Kit Multimedia Standards Update, Rev. 3.0, 15 April 1994.

ISO/IEC 11173-3: MPEG 1.

ISO/IEC 13818-3: MPEG 2.

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