AES standard for network and file transfer of audio — Audio-file transfer and exchange — Radio traffic audio delivery extension to the broadcast-wave-file format

Published by
Audio Engineering Society, Inc.

Abstract
This document provides a convention for communicating basic radio traffic and continuity data via a dedicated chunk embedded in broadcast wave file compliant WAVE files.

Contents

Foreword

[This foreword is not a part of *AES standard for network and file transfer of audio — Audio-file transfer and exchange — Radio traffic audio delivery extension to the broadcast-wave-file format,* AES46-2002.]

This document was written by a task group, headed by D. Pierce and G. Steadman, of the SC-06-01 Working Group on Audio-File Transfer and Exchange of the SC-06 Subcommittee on Network and File Transfer of Audio, under project AES-X87. The members of the task group were G. Novacek, Pierce, Steadman, G. Uzelac, and J. Zigler.

Mark Yonge, chair
Brooks Harris, vice-chair
SC-06-01
2002-02-21

NOTE: In AES standards documents, sentences containing the verb "shall" are requirements for compliance with the standard. Sentences containing the verb "should" are strong suggestions (recommendations). Sentences giving permission use the verb "may." Sentences expressing a possibility use the verb "can."

AES standard for network and file transfer of audio — Audio-file transfer and exchange — Radio traffic audio delivery extension to the broadcast-wave-file format

0 Introduction

0.1 Rationale

The radio broadcast industry utilizes a variety of production, on-air and other equipment in daily operation. No single vendor dominates the industry. Users have long complained about the inability to transport audio and traffic-continuity data between systems in a uniform and easy fashion. This complaint is because of the lack of any uniform agreement about an exchange standard for communicating this information among systems. Often, different on-air delivery systems use proprietary audio-file formats and incompatible access methods to manage audio storage and playback, yet the scheduling, continuity or traffic information they use to label audio files share many common attributes. Furthermore, audio data itself is represented in various often-proprietary formats. To simplify the communication among different systems such as audio production and on-air delivery systems, a common representation for both continuity or traffic information and audio data is desirable.

The resource interchange file format (RIFF) `WAVE` format has emerged as a dominant audio representation. It supports a wide variety of audio formats such as linear pulse-code modulation (PCM), Moving Pictures Experts Group (MPEG) formats, different sampling frequencies and sample sizes, multiple tracks, and so on. The RIFF conventions allow the arbitrary addition of other data without impacting the ability of diverse RIFF-compliant applications to read and interpret needed data. Thus, adding an extension to a `WAVE` file allows inclusion of needed continuity or traffic data to a widely accepted representation.

The RIFF specification requires all readers to be able to read all compliant RIFF files. When such an application encounters data that it is not prepared to handle, it can simply ignore the data and move on. Some RIFF consumer applications are intolerant of new and unknown chunks. For this reason alone, these applications are not RIFF-compliant; but they may be front-ended by so-called chunk-stripper utilities, the product of which is then RIFF-compliant.

The radio traffic data (commonly called CART) format described in this document utilizes a widely used audio-file format (`WAVE` and broadcast wave file). It incorporates broadcast-specific cartridge-labeling information into a specialized chunk within the file itself. As a result, the burden of linking multiple systems is reduced to the producer applications writing a single file and the consumer applications reading it. The destination application can thereby extract information and insert it into the native database application as needed.

0.2 Conventions

0.2.1 Decimal points

According to IEC directives, the comma is used in all text to indicate the decimal point. However, in specified coding, including the examples shown, the full stop is used as in IEC programming language standards.

0.2.2 Data representation

All coding and data representations are printed in an equally spaced font.

0.2.3 Non-printing ASCII characters

Non-printing characters are delimited by angle brackets, as in `<CR>` for carriage return.

0.2.4 Reserved bits
Unless otherwise indicated, bit assignments shown as reserved are reserved for future standardization by the AES, only by means of amendment or revision of this document.

1 Scope

This document provides a means for communicating basic radio traffic and continuity data via a dedicated chunk embedded in broadcast-wave-compliant WAVE files. The new RIFF chunk supports most common data used in radio traffic and continuity systems, while the WAVE format itself supports most sampling frequencies, sample widths, and audio formats.

This document does not specify representation of this or other data within a specific application's space, only in the public interchange between disparate systems. Any such private representation may be covered by other standards or by a particular vendor's best judgement.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this document. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the indicated standards.

ISO/IEC 646:1991, *Information technology — ISO-7-bit coded character set for information exchange.* Geneva CH: International Organization for Standardization.

RIFF file structure. See the resource locator on the databases page of www.aes.org/standards.

3 Definitions and abbreviations

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

3.2
chunk
data package within RIFF files containing related data

3.3
EBU broadcast wave file
BWF
WAVE file containing the EBU bext chunk and extensions as described in European Broadcast Union EBU Tech. Doc. 3285

NOTE See clause 8.

3.4
bext
broadcast wave file or BWF extension chunk

3.5
cart
CART
extension chunk to WAVE file containing CART format data as described in this document

3.6
ASCII
7-bit coded character set, according to ISO 646

3.7
NUL
character code specified as 0/0 in ISO 646 wherein all bits of the code are set to zero

3.8
CR
character code specified as 0/13 in ISO 646 for carriage return

3.8
LF
character code specified as 0/10 in ISO 646 for line feed

3.10
MPEG
mpeg
compressed audio data as specified within ISO 11172-3:1993, originally formulated by the Motion Pictures Experts Group as MPEG I audio

3.11
WAVE file
Audio waveform file format using the RIFF file representation

NOTE See clause 8.

4 Coding conventions

4.1 Coding examples

Coding examples and data layouts described herein use the syntax and conventions of the C-programming language. These examples and layouts may be used for illustrative purposes only and neither constrain or recommend a particular implementation style or method.

The mnemonics that describe the data types used in these examples shall be as shown in Table 1.

Table 1 — Example data types

Atomic type	Meaning	C type
CHAR	8-bit signed integer, representing integer values from –128 to +127	signed character
BYTE	8-bit unsigned integer, representing integer values from 0 to +255	unsigned character
INT	16-bit signed integer in little-endian format (most-significant octet last), representing integer values from –32768 to +32767	signed short integer
WORD	16-bit unsigned integer in little-endian format, representing integer values from 0 to +65535	unsigned short integer
LONG	32-bit signed integer in little-endian format, representing integer values from –2147483648 to +2147483647	signed long integer
DWORD	32-bit unsigned integer in little-endian format, representing integer values from 0 to +42949672951	unsigned long integer

The tag type used in RIFF files may be further defined as

```
typedef DWORD FOURCC; // Four-character code
```

A macro operator, `cvtFOURCC` can be defined whose purpose is to convert a character string or four characters into a `FOURCC` representation in a system-independent, portable fashion, for example:

```
tag4cc = cvtFOURCC("ABCD");
tag4cc = cvtFOURCC('A', 'B', 'C', 'D');
```

When the string `//` appears in structure layout or other illustrative example, it can designate the start of an explanatory comment. Neither `//` nor any text following are part of the actual layout or data.

4.2 Octet ordering

The octet ordering used for the storage of multi-octet numeric data (`INT`, `WORD`, `DWORD`, `LONG`, and so on) in RIFF files shall be the least significant octet first.

16-bit values (with bits numbered 00 though 15) shall be stored in files as in Table 2.

Table 2 — 16-bit values

	1 1 1 1 1 1 0 0 5 4 3 2 1 0 9 8
Octet 1	
Octet 0	0 0 0 0 0 0 0 0 7 6 5 4 3 2 1 0

32-bit values (with bits numbered 00 through 31 from least significant to most significant) shall be stored in files as in Table 3.

Table 3 — 32-bit values

	3 3 2 2 2 2 2 2 1 0 9 8 7 6 5 4
Octet 3	
Octet 2	2 2 2 2 1 1 1 1 3 2 1 0 9 8 7 6
Octet 1	1 1 1 1 1 1 0 0 5 4 3 2 1 0 9 8
Octet 0	0 0 0 0 0 0 0 0 7 6 5 4 3 2 1 0

5 Character set

The `cart` extension chunk shall use the ASCII character set for all text strings.

The first character of the identifier `FOURCC` shall be an upper- or lower-case alphabetic character, followed by one to three upper- or lower-case alphabetic or numeric characters. If the identifier is less than four characters long, the remaining characters shall be `<NUL>` characters.

Upper-case `FOURCC` reserved identifiers, when used for chunk identifiers, is by convention reserved for specific registered RIFF identifiers (see annex C). Other chunk identifiers shall use lower-case alphabetic and numeric characters.

Internal data within chunks using `FOURCC` identifiers may use upper- or lower-case alphabetical and numeric characters

6 `cart` extension chunk

6.1 Chunk ordering

The first chunk in a cart `WAVE` file shall be the format chunk.

If required, the fact chunk should be second. The last chunk should be the data chunk. Other chunks in the file may be in any order.

NOTE This chunk order can provide optimum speed of access to the radio-traffic data with a wide range of computer filing systems. Any RIFF compliant chunk sequence can be encountered in practical interchange.

The example is an MPEG-encoded WAVE file with both BWF and cart chunks, in addition to the fact chunk and mpeg chunk.

EXAMPLE

```
<WAVE-form> ->
RIFF('WAVE'
<fmt-ck>  // required for all WAVE files
<fact-ck> // required for non-PCM data
<bext-ck> // EBU BWF chunk
<mpeg-ck> // EBU MPEG-extension data chunk
<cart-ck> // cart information
<data-ck> // audio data, required for all WAVE files
```

6.2 Contents of `cart` extension chunk

6.2.1 Contents

The `cart` extension chunk shall have the contents shown in Table 5. The example uses C-programming notation for illustration.

EXAMPLE

```
typedef struct cartchunk_tag
{
     DWORD ckID;                // FOURCC chunk ID: cart
     DWORD ckSize;              // chunk data length in octets
     BYTE ckData[ckSize];       // data, as cart_EXTENSION type
}
typedef struct cart_extension_tag
{
   CHAR Version[4];             // Version of the data structure
   CHAR Title[64];              // ASCII title of cart audio sequence
   CHAR Artist[64];             // ASCII artist or creator name
   CHAR CutID[64];              // ASCII cut number identification
   CHAR ClientID[64];           // ASCII client identification
   CHAR Category[64];           // ASCII Category ID, PSA, NEWS, etc
   CHAR Classification[64];     // ASCII Classification or auxiliary key
   CHAR OutCue[64];             // ASCII out cue text
   CHAR StartDate[10];          // ASCII YYYY-MM-DD
   CHAR StartTime[8];           // ASCII hh:mm:ss
   CHAR EndDate[10];            // ASCII YYYY-MM-DD
   CHAR EndTime[8];             // ASCII hh:mm:ss
   CHAR ProducerAppID[64];      // Name of vendor or application
   CHAR ProducerAppVersion[64]; // Version of producer application
   CHAR UserDef[64];            // User defined text
   DWORD dwLevelReference       // Sample value for 0 dB reference
   CART_TIMER PostTimer[8];     // 8 time markers after head
   CHAR Reserved[276];          // Reserved for future expansion
   CHAR URL[1024];              // Uniform resource locator
   CHAR TagText[];              // Free form text for scripts or tags
} CART_EXTENSION;

typedef struct cart_timer_tag   // Post timer storage unit
{
   FOURCC dwUsage;              // FOURCC timer usage ID
   DWORD dwValue;               // timer value in samples from head
} CART_TIMER;
```

Table 5 — `cart` extension chunk contents

Field	Description
`Version`	4-character ASCII numeric string giving the version of the `cart` data structure, particularly the contents and usage of the reserved area. The first two numbers shall give the major release level (with leading 0) from 00 to 99 and the last two shall give the revision level (with leading 0) in the range of 00 to 99. The version number of the `cart` data structure as described in this document shall be version 1.01, and thus is represented by the string 0101.
`Title`	ASCII string, 64-characters or less, representing the title of the cut. The title should be a descriptive summary of the audio contents of the file, and may be used as an entry into a table of contents, and so on. Applications that do not support a 64-character title may truncate the field as needed.
`Artist`	ASCII string, 64-characters or less, holding the artist or creator name for the audio cut.
`CutNum`	ASCII string, 64-characters or less, representing the cut number, or unique cut key. The string shall be left justified. Some consumer systems can have restricted cut number lengths or allowable character set. These applications should provide some means of synthesizing a usable cut identifier if it has such restrictions.
`ClientID`	ASCII string, 64-characters or less, holding a client or customer identification or name.
`Category`	ASCII string, 64-characters or less, holding a category name. The category name may be application dependent. Applications should use common category names. See annex A for a list of recommended category names.
`Classification`	ASCII string, 64-characters or less, holding a classification key. This key may be used for general classification, selection or sorting based on language, locale or other similar applications.
`OutCue`	ASCII string, 64-characters or less, holding the optional out cue phrase to be displayed when the cut is being played. This shall be a user readable cue string.
`StartDate`	ASCII date string, 10 characters, of the form YYYY-MM-DD, such as 1998-12-25, holding the start date. Year (YYYY) shall be defined as 0000 to 9999. Month (MM) shall be defined as 01 to 12. Day (DD) shall be defined as 01 to 28, 29, 30 or 31 as applicable. The separator between date fields shall be a hyphen, (-). Note: This format complies with ISO 8601 and is compatible with other dates in BWF files. To signify an immediate start date, applications shall use 1900-01-01.
`StartTime`	ASCII time string, 8 characters, of the form hh:mm:ss, such as 12:31:45, representing the 24-hour time-of-day for the start time on the assigned StartDate. Hour (hh) shall be defined as 00 to 23. Minutes (mm) and seconds (ss) shall be defined as 00 to 59. The separator between time fields shall be a colon, (:). If blank, applications shall assume a start time of 00:00:00.

EndDate	As in start date, but shall indicate the date after which the sequence will no longer be active. If the sequence is to run forever, the date shall be 9999-12-31. There shall be no default for this field.
EndTime	This code shall indicate the time of day on the appointed end date after which the sequence becomes inactive. If blank, applications shall assume an end time of 23:59:59.
ProducerAppID	An ASCII string, 64 characters or less, containing the vendor name, product name or both of the program or application that produced the WAVE file with this cart chunk.
ProducerAppVersion	An ASCII string, 64 characters or less, containing the version of the program or applications that produced the WAVE file containing the cart chunk. Because this string is informational only, the application may represent the version in any convenient format.
UserDef	An ASCII string, 64 characters or less, whose use and contents may be defined by the user of the system.
dwLevelReference	A 32-bit signed (2's complement) integer word that shall hold the sample value of the 0-dB reference level for the originating system. This reference can facilitate scaling and metering consistency across disparate systems. As an example, a 16-bit linear PCM system that has its meters calibrated as 0 corresponding to maximum signed digital value shall have the value set to 32768 (8000_{16}). The peak value shall be the absolute value of the largest sample value possible before saturation. In the example given, that of a 16-bit linear system using 2's complement notation, the range of allowable values is -32768 to 32767, thus the maximum peak value is 32768 in the example given.
PostTimer	Eight CART_TIMER structures representing time marks. The time units shall be in sample periods at the sampling frequency of the associated audio data and shall be referenced to the first sample of the audio data. The timer range shall be 2^{32} or 4,294,967,295 sample periods. These periods allow timer ranges at a sampling frequency of 48 kHz, for example, to extend beyond 24 h (24:51:18). These timers may be used to activate events in the cart system. Each timer entry shall consist of a FOURCC timer usage identifier (dwUsage) and a 32-bit unsigned integer DWORD timer in sample periods (dwValue) as described above. Applications should use FOURCC usage identifiers as described in annex A.3 If a timer is not used, or is not set, its usage identifier should be set to all <NUL> characters (00000000_{16}) and its timer value set to 0 (00000000_{16}).
Reserved	This area, 276 octets, shall be reserved.
URL	An ASCII string, 1024-characters or less, representing a universal resource locator (URL) referencing or referenced by the audio program. The URL field contents should conform to the URL syntax as shown in annex C.
TagText	Non-restricted ASCII characters containing a collection of strings each terminated by <CR><LF>. This text may be system- or user- defined descriptive text for the sound, such as live tag, script information, descriptive text special instructions, and so on.

6.2.2 Default values and empty data

Text fields that do not require strings shall have a zero-length string. The first octet of the field shall be a <NUL> character, and all subsequent data in the field shall be ignored.

Binary value fields not requiring a specific value shall be set to a value of 0.

The action taken on encountering a blank or empty data may be implementation and site-installation dependent.

6.2.3 Short strings

In cases where the string contents, in octets, is shorter than the specified field size, the string shall be terminated by a <NUL> octet following the last significant character. This terminator and the remainder of the field shall be ignored. Strings shall be left justified.

6.3 Other relevant information

All the other information regarding WAVE audio characteristics can be found in the mandatory fmt chunk. This includes sampling frequency, number of tracks, sample width and sample format. For other than PCM format, the fact chunk and the EBU mpeg chunk can contain further information. Refer to the EBU Tech Document 3285 for information on these data (see annex B).

6.4 Broadcast wave usage

The broadcast wave bext chunk may be used in conjunction with the cart data described in this document. Its data does not conflict with nor replicate data here.

Refer to EBU Tech 3285 for more specifics on the recommended official usage of these fields.

7 Private and application-specific information

Private and application specific data not contained in the cart chunk data described here is outside the scope of this document.

8 Assignment of coding

The coding of resource locators, identifiers, and formats shall conform to recognized industry practice as determined by the AESSC according to its rules. This determination shall be shown as of the printing date of this standard in annex C which shall be published and kept current on the AESSC Web site databases page.

Annex A

*(*informative*)*

Recommended parameter names

A.1 Category names

Categories and aliases should be according to table A.1. The actual categories and aliases may be site dependent, implementation dependent, or both.

Table A.1 – Recommended category names and aliases

Category names	Aliases
All	ALL
Beds	BED, BEDS
Sound bits	BIT, BITS
Commercials	COM, COMM
Contests	CON, CONT
Daily play lists	DAY
Emergency broadcast	EB
Sound effects	EFX
Fillers	FIL, FILL
Station ID	ID
Intros	INT, INTR
Jingles	JIN, JING
Liners	LIN, LINE
Logos	LOG, LOGO
Magic call	MAG, MAGI
Music	MUS, MUSC
Network delay	NET, NETW
News	NEW, NEWS
Promos	PRO, PROM
Public service announcements	PSA
Segues	SEG
Shows	SHW, SHOW
Sound effects	SND
Spots	SPO, SPOT
Sports	SPR, SPRT
Stagers	STG, STAG
Announcer stack	STK, STAK
Sweeps	SWP, SWEP
Test tones	TST, TEST
Temporary	TMP, TEMP

A.2 Mark timer identification

Timer types, along with their FOURCC identification should be according to table A.2. The interpretation and behavior of systems on encountering timer information may be site dependent, implementation dependent, or both.

Table A.2 – Basic timer types

Timer ID	Description	Start-End	Enumerated	Multiples
	Unused	No	No	Yes
SEG	Segue timer	Yes	Yes	Yes
AUD	Audio boundary	Yes	No	No
INT	Introduction	Yes	Yes	Yes
OUT	Epilog	Yes	Yes	Yes
SEC	Secondary	Yes	Yes	Yes
TER	Tertiary	Yes	Yes	Yes
MRK	Generic marker	No	Yes	Yes
EOD	End-of-data	No	No	Yes

Timers may be qualified in one of three ways:

a) as start or end timers, by appending a lower case ASCII letter `s` for a start timer or a lower case ASCII letter `e` for an end timer; for example, the timer identification `AUDs` designates the start of audio following silence, while `AUDe` designates the end of the audio segment;

b) as enumerated timers, by appending an ASCII numeric character; for example, `SEC1` may be designated secondary timer number 1, `SEC2` may be secondary number 2, and so on;

c) as multiple timers, by having multiple instances of the same timer ID; one may have, for example, multiple instances of `MRK`.

Each application may prioritize the order of the timers.

Annex B

(informative)

Informative references

Microsoft Software Developers Kit Multimedia Standards Update, rev 3.0 15 April 1994, Microsoft Corporation

Microsoft Multimedia Programmer's Reference 1991-1992, Microsoft Corporation

EBU Tech Document 3285 – Supplement 1: *Specification of the Broadcast Wave Format, Supplement 1 – MPEG audio*. Geneva CH: European Broadcasting Union.

ISO 11172-3 - *Part 3: Audio, Information technology - coding of moving pictures & associated audio - for digital storage*. Geneva CH: International Standards Organization

Annex C

*(*normative*)*

Currently approved resource locator, identifier, and format references

See the resource locator on the databases page of www.aes.org/standards/ for updated information.

IETF RFC1738 *Uniform Resource Locators (URL).* Internet Engineering Task Force.

EBU Tech 3285, *Specification of the Broadcast Wave Format.* Geneva CH: European Broadcasting Union.