Specification
for
CART/Audio Delivery Extension
to the
EBU Broadcast WAVE Format
(proposed)

Dick Pierce, Principle Software Developer, Audicy
Professional Audio Development
Hanover, MA USA

Geoff Steadman, Product Manager, Audicy
Orban, Inc., Harman Industries
San Leandro, CA USA

Contributing Authors

Greg Uzelac
Broadcast Electronics, Inc.
Quincy, IL USA

Jeff Zigler
Prophet Systems, Inc.
Ogalala, NE USA

Eugene Novacek
ENCO Systems, Inc.
Farmington Hills, MI USA

Revision 0.9
December 01, 1999

0 SUMMARY OF CHANGES

01 December, Dick Pierce
- Bring syntax more into compliance with AES documentation requirements
- Explicitly specify unassigned timer ID

22 September, Dick Pierce, Greg Uzelac, Geoff Steadman
- Rework version ID
- Minor typo corrections

16 September, Dick Pierce, Greg Uzelac
- Incorporate feedback from maillist discussions
- Addition of ClientID, ProducerAppName and ProducerAppVersion fields
- Lessen restrictions on and increase size of CutNum fields
- Documentation clarification
- Define standard timer namings
- Add suggested list of category names

22 July 1999, Geoff Steadman, Dick Pierce
- Expand justification arguments
- Clarify structure usage
- Minor format changes

29 June 1999, Dick Pierce
- Clarify timer usage and cut number for maximum portability

19 June 1999, Dick Pierce
- As a result of discussions with several other vendors,
- Rework the timers representation,
- Move the version field to the first position
- Add classification/auxiliary key field
- Add appendices on data types and character sets

1 March 1999, Dick Pierce
- Incorporated category change in actual doc

14 December 1998, Dick Pierce
- Extend size of base structure increasing reserved area
- Add discussion regarding extension and layout choices.
- Ad version ID field
- Increase size of category field

24 November 1998, Dick Pierce
- Structure member name changes to conform to EBU practices, grammar cleanup

09 October 1998, Dick Pierce
- Added 0 dB reference level field

29 September 1998, Dick Pierce
- Clarified usage of start and end date, especially defaults. Remove use of term “kill date.” All timers are now head-referenced. Default timers

24 September 1998, Dick Pierce
- Expanded text fields for 64 characters for most text strings, cut number to 8 digits.

09 September 1998, Dick Pierce
- Clarified relationship between “cart” chunk and “bext” chunk: “cart” chunk is separate, optional chunk, not an extension of the “bext” chunk.

01 September 1998, Dick Pierce
- Original version

1 INTRODUCTION

This white paper proposes a new RIFF WAVE data type specifically for use as an *interchange* medium by audio production and on-air delivery systems to exchange audio data in the form of WAVE files, along with basic scheduling, traffic[1] or continuity information. What amounts to a "digital label" embedded in a common audio file type, and respected as a standard, will provide what amounts to a translation protocol, enabling better integration between on air delivery and production systems, while minimizing the need for custom software development for vendors seeking to have their systems talk to each other. In no way is this should this proposal be construed as forcing any vendor to alter their own internal file representations; our goal here is to define a common language for different systems and applications to speak when they need communicate with each other.

The relation of this new object, the 'cart' chunk, to other components of a Broadcast Wave File, is illustrated schematically in Figure 1 below.

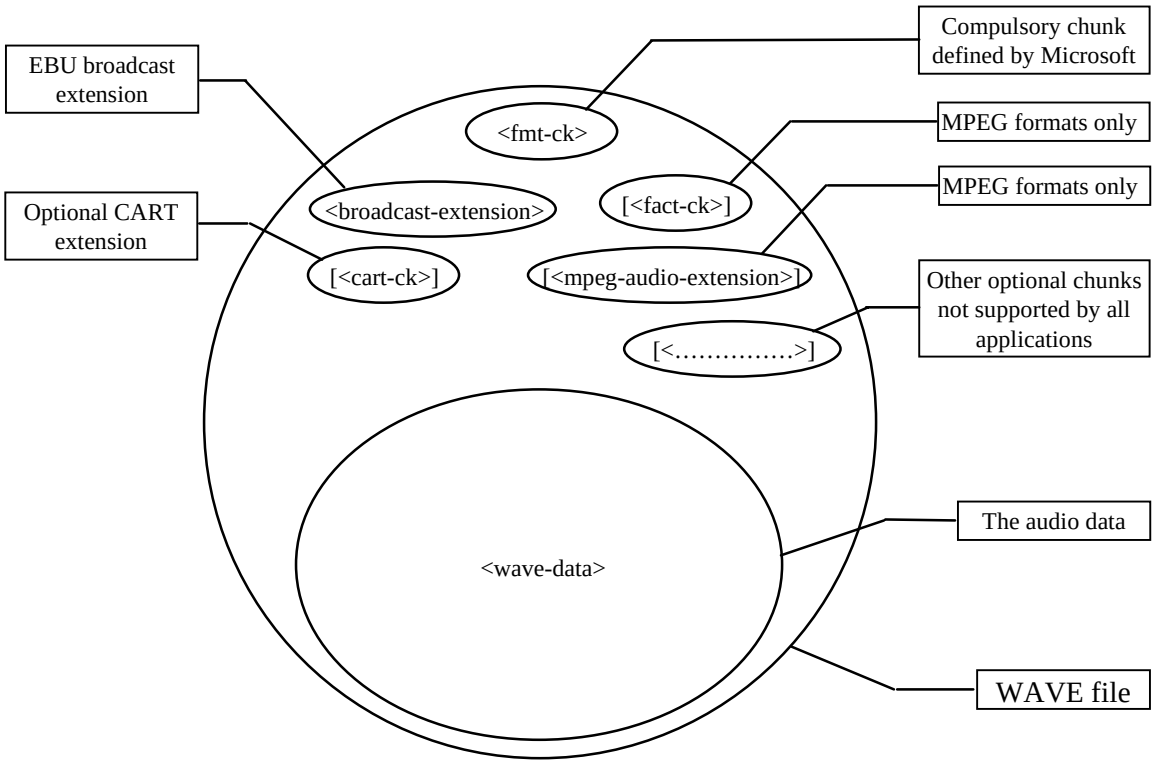

Figure 1: Broadcast WAVE format with CART extension

This proposed extension, as mentioned, is based on the EBU Broadcast Extension WAVE file format (BEXT), the MPEG extension, and supporting documentation[2].

1.1 Justification for 'cart' extension to WAVE file

The radio broadcast industry utilizes a wide variety of production, on-air and other equipment in daily operation. No single vendor dominates the industry. Users have long complained about the inability

[1] As a point of clarification "traffic" in the context of this document is intended to mean radio station traffic management, as in play scheduling and the like, and not "road traffic" information.

[2] The EBU Broadcast Extension and MPEG extension data are defined in EBU Technical Document 3285, "Specification of the Broadcast Wave Format," July 1997, EBU Publications, Grand-Saconnex (Geneva) Switzerland.

to transport audio and traffic/continuity data between systems in a uniform and easy fashion. This is due to the lack of any uniform agreed-upon exchange standard for communicating this information between systems. Different on-air delivery systems often 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. Further, audio data itself is represented in various, often proprietary formats. To simplify the communication between different systems such as audio production and on-air delivery systems, a common representation for both continuity/traffic information and audio data is desirable.

The RIFF WAVE format has emerged as a dominant audio representation, and supports a wide variety of audio formats (linear PCM, MPEG, different samples rates and sample sizes, multiple tracks, etc.). The RIFF conventions allow the arbitrary addition of other data without impacting the ability of diverse RIFF-compliant[3] applications from reading and interpreting needed data. Thus, adding an extension to a WAVE file allows inclusion of needed continuity/traffic data to a widely accepted standard representation.

By utilizing a standard audio file format (WAVE and EBU/BEXT) and incorporating the common cart 'labeling' information into a specialized chunk within the file itself, the burden of linking multiple systems is reduced to producer applications writing a single file, and the consumer applications reading it. The destination application can extract information and insert it into the native database application as needed. Communication between a production/delivery system could be reduced to a simple, purely passive link that allows the production application to write the properly formatted WAVE file in a standard "drop box" location, where the delivery system, periodically polling the drop-box for new additions, finds the file, opens it, and uses it's own native access methods for adding this information to its database. Providing this type of "translation layer" offers vendors a means of integration with other systems, without impacting their own native file formats and access methods.

The result is that both production/editing systems and on-air delivery systems can communicate readily without the need for implementation-specific intelligence or design. However, the use of this new data is not necessarily limited to these applications and the examples illustrated here.

As to the audio contents, the recommendation has been made elsewhere of the importance on standardizing on a common exchange format, and WAVE, especially in the form of the EBU Broadcast Extension standard. We endorse this recommendation, though this does not necessarily require the use of EBU/BEXT files as each system's native format.

---

[3] 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. There, indeed, exist some RIFF WAVE consumer applications that are intolerant of new and unknown chunks. For this reason alone, these applications are not RIFF-compliant. They may be front-ended by so-called "chunk stripper" utilities, the combination of which are then RIFF-compliant.

2 BROADCAST WAVE WITH CART EXTENSION

2.1 Contents of Proposed BEXT with CART extension

We are proposing using the standard EBU/BEXT wave format file with the addition of a new optional chunk type. Such a file would have the following minimal chunks:

```
<WAVE-form>  ->
   RIFF('WAVE'
      <fmt-ck>                    // mandatory for all WAVE files
      [<fact-ck>]                 // required for MPEG format files
      <bext-ck>                   // EBU Broadcast Extension information
      [<mpeg-ck>]                 // required for EBU MPEG format files
      [<cart-ck>]                 // optional cart information
      <data-ck>                   // audio data, required for all WAVE files
```

Note: *Any additional chunk types that are present in the file are considered private. Applications are not required to interpret or make use of these chunks.*

The 'cart' chunk is described in the following C-style data definitions:

```
typedef struct cartchunk_tag
{
   DWORD ckID;                    /* FOURCC chunk ID: "cart"            */
   DWORD ckSize;                  /* chunk data length in bytes         */
   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/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/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[4]       */
   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;
```

Field	Description
Version	A 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 give the major release level (with leading 0) from 00 to 99 and the last two the revision level (with leading 0) in the range of 00 to 99. The first release, first revision, for example, would be “0100”.
Title	Up to 64-characters for the title of the cut. This differs in use from the EBU BEXT <description> field. The title is normally viewable on the cart or delivery applications and can be used as an entry into a table of contents or a key in an

[4] The reserved area size is chosen so that the entire defined base cart chunk size, neglecting tag text, is 1024 bytes. This is so that the tag text can start in a known position. Since the tag text area is free-form, it’s length is variable. Consumer applications can know the exact length by subtracting the size of the cart structure as described from the chunk length. It’s position in the file is simply the position of the cart chunk area plus the size of the cart structure as defined.

indexing or search system[5]. If the title occupies less than 64 characters, the last valid character is followed by a NULL byte '\0'. Some applications may not support a 64 character title and may truncate it as needed.

Artist — A 64-character long ASCII string holding the artist or creator name. This is different than the <originator> field in the EBU BEXT chunk in that it can be used to describe the original artist of a song, for example, while the <originator> field would be more appropriate as the producer of the specific audio file. If the string is less than 64 characters, it is terminated with a NULL byte.

CutNum — A 64-character ASCII string representing the cut number, or unique cut key. The string should be left justified and terminated, if less than 64 digits long, by a NULL byte. Some consumer systems may have restricted cut number lengths or allowable character set. These applications should provide some means of synthesizing a usable cut ID if it has such restrictions.

ClientID — A 64-character long ASCII string holding a client or customer identification or name. If the string is less than 64 characters, it is terminated with a NULL byte.

Category — A 64-character ASCII string holding a category name. The category name is somewhat application dependent. It's advisable, though, to use common or standard category names, such as "PSA" or "NEWS" and so on A more complete list of suggested category names is listed in the following appendix. If the string is less than 64 characters, it is terminated with a NULL byte.

Classification — A 64-character ASCII string holding a classification key. This key can be used for general classification, selection or sorting based on language, locale or other similar applications. If the string is less than 64 characters, it is terminated with a NULL byte.

OutCue — A 64-character ASCII string holding the optional outcue phrase to be displayed when the cut is being played. This is a user readable cue string. If less than 64 characters in length, it's terminated with a NULL byte.

StartDate — An ASCII date string of the form yyyy/mm/dd[6], such as 1998/12/25, holding the start date. Any valid date can be used. To signify an immediate start date, use "1900/01/01."

Year is defined as 0000 to 9999.

Month is defined as 1 to 12 (or 01 to 12).

Day is defined as 1 (or 01) to 28, 29 30 or 31, as applicable.

Separator between fields is normally "/", but can be any of "-", "_", ":", " " (space) or ".".

StartTime — An ASCII time string 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>. If blank, assume 00:00:00.

---

[5] It is reasonable to use the BEXT Description field for more detail about the cart.

[6] The format for the date and time strings is intended to mirror that of the EBU BWF usage. The justification for this is that the same interpreter for the EBU date and time strings can be used here.

Hour is defined as 0 (or 00) to 23.

Minutes and seconds are defined as 0 (or 00) to 59.

Separator between fields is normally ":", but can be any of "-", "_", " " (space) or ".".

EndDate — As above in start date, but indicating the end date[7]. This the date on after which the sequence will no longer be active. If the sequence is to run forever, use "9999/12/31". There is no default for this field.

EndTime — As above in start time, indicating the time of day on the appointed end date. If blank, assumed 23:59:59.

ProducerAppID — A 64-character ASCII string containing the vendor name and/or product name of the program or application that produced the WAVE file with this 'cart' chunk. If less than 64 characters long, it is terminated with a NULL byte.

ProducerAppVersion — A 64-character ASCII string containing the version of the program or applications that produced the WAVE file containing the 'cart' chunk. If less than 64 characters long, it is terminated with a NULL byte.

UserDef — A 64-character ASCII string whose use and contents are defined by the user of the system. If the user text is less than 64 characters long it is terminated with a NULL byte.

dwLevelReference — A 32 bit signed (two's complement) integer word holding the sample value of the 0 dB reference level for the originating system. This is to 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 will have the value set to 32768 (0x00008000)[8]. A similar system with the peak value set to +6 dB will have this value set to 16384 (0x00004000), since the 0 dB meter reference level is 6 dB below, or ½ have the value of, saturation.

PostTimer — Eight **CART_TIMER** structures representing time marks. The time units are in sample periods at the audio data's current sample rate[9]. These timers are used to activate events in the cart system.

Each timer entry consists of a timer dwUsage tag or ID and a 32 bit unsigned integer timer dwValue as described above. The usage can be viewed as a 4-character tag (as a FOURCC) representation of the purpose of the time.

An enumeration of "standard" timer tags is a matter for further discussion and standardization[10]. See the attached appendix for more on timer identification.

---

7 Often referred as the "kill" date. The term "end" date is to be preferred because of the confusion with the concept of "killing" the entry altogether, purging it from the system, as opposed to simply deactivating it.

8 The peak value can be viewed as the absolute value of the largest sample value possible before saturation. In the example given, that of a 16 bit linear system using two's complement notation, the range of allowable values is -32768 to 32767, thus the maximum peak value is 32768 in the example given.

9 The timer range is 2^{32} or 4,294,967,295 sample periods. This allows timer ranges at a sample rate of 48 kHz, for example, to extend beyond 24 hours (24:51:18).

	If a timer is not used (unset), its ID should be set to CART_TIMER_UNUSED (a null string, or 0x00000000) and its value set to CART_TIMER_UNSET (0xFFFFFFFF). The order of the timers can be taken as a priority of timers, where conflicts may exist, the lower-order timers have precedence over higher order entries.
Reserved	Area reserved for future expansion of the standard.
TagText	Non restricted ASCII characters containing a collection of strings each terminated by CR/LF. This text can be system- or even user- defined descriptive text for the sound, such as live tag, script information, special instructions, etc.[11].

2.2 Other Relevant Information

All the other information regarding WAVE audio characteristics can be found in the mandatory "fmt" chunk. This includes sample rate, number of tracks, sample width and sample format. For other than PCM format, the "fact" chunk and the EBU "mpeg" chunk will contain further information. Refer to the relevant documentation for information on these data. Information regarding the exact format of the audio data representation is also to be found in the reference documentation.

2.2.1 'bext' chunk usage suggestions

The BEXT chunk already has fields that could be of use in a cart application without violating the intentions of the BEXT standard. Some of them are described here with some suggestions for usage in a cart/delivery system context. Refer to EBU 3285 [3] for more specifics on the "official" usage of these fields.

Field	**Description**
Description	A 256 character ASCII field that can hold a free for description. Several cart/delivery systems utilize a long description field, and this could be used for such a purpose.
Originator	A 32 character ASCII field holding the originator or creator of the sound.
OriginatorReference	A 32 character ASCII field holding, as 3285 describes, "a non-ambiguous reference allocated by the originating organization."
OriginationDate	10 character ASCII for date of creation
OriginationTime	8 character ASCII for time of creation
TimeReference	64-bit time code of sound sequence
Version	Unsigned int version number.
Reserved	254 byte reserved area
CodingHistory	Non-restricted ASCII characters containing a history the sound's coding process.

3 APPLICATION-SPECIFIC INFORMATION

It is certain that some radio scheduling applications can make use of or require data not included above. The RIFF chunk file format allows addition of arbitrary new chunks in which such information could be placed for private exchange between applications that can understand the information. Such

---

[10] The possibility of private, proprietary timer tags was advanced. However, again it was strongly argued against in the interest of maximizing portability amongst all possible applications. Instead, work will continue on drawing up a standard list of timers having the widest applicability.

[11] An alternative use for this area is discussed later in this document.

chunks, however, will fall under the category of "private" chunks and can be ignored by applications conforming to the standard definitions. As such, they are beyond the scope of this document.

Some correspondents have raised objections to the fixed-field layout of the cart chunk, preferring instead a free-format keyword-oriented approach. The EBU BWF chunk format is similarly a fixed-field layout, and the proposed cart extension was designed with the BWF layout in mind. The free-format approach, while providing a degree of flexibility and expandability, suffers from a corresponding increase in complexity since, for one, a more intelligent parsing system is required for reading data. The convention runs the risk of violating the very notion of a standard by allowing essentially indeterminate content to be added to the chunk. One must remember that the objective here is to provide a commonly agreed upon means of communicating known and well-characterized data between diverse applications. The goal is to allow as many such diverse applications to connect as possible. It's also to be noted that none of the other relevant chunks (fmt, fact, bext, mext, etc.) employ such a free-format layout.

The proposed standard addresses these issues in two ways. First, as in the EBU BWF standard, an area has been set aside for future standardization. This area would continue expansion of fixed size fields in a manner as currently done. Secondly, the area at the end of the chunk (currently designated as the TagText member) is available for arbitrary expansion. If desired, this area could be used for such keyword-oriented data.

If the latter path is chosen, it is the recommendation of the authors that a tagged format, essentially the same as the basic RIFF chunk format, be used in this area. This retains the advantage of keyword-orientation while also retaining the full backwards and forwards compatibility of the basic tagged methods. Diverse applications can easily navigate this area without the requirement of understanding all possible members. It is further recommended that such an approach be subject to the standardization process.

And, ultimately, there is a third approach, as mentioned above: further chunks can be defined as seen necessary.

XML has been proposed as one alternative solution to the problem. While it is agreed that XML can provide a high degree of flexibility and extensibility, there are several problems with this approach:

- All proposed XML-based schemes require that two files be used: one for the traffic/continuity meta-data, and the second for the audio itself. Many respondents have already objected strongly to this implementation, preferring, instead, to keep the all of the information in a single file. The single file approach ensures the traffic data and audio data will remain together. These respondents agree that separating the meta-data and audio into two separate files unacceptably increases the risk of the loosing one or the other.
- In a two-file method, there can be a hidden operating system dependency in the link between the two files. It is possible, for example, to have the link point to a file whose name is incompatible with the requirements of the target system.
- In no way is the present approach incompatible with XML-based approach. Storing the data within the confines of the audio file via the 'cart' chunk approach does not preclude using an XML representation, and vice versa.

To restate the point of this proposal, the goal is to provide a common means of communicating specific information for a reasonably well defined and constrained set of applications. The proposed format is an *interchange format*, and in no way dictates or limits a given application's internal representations of this or other data.

4 APPENDICES

4.1 Atomic Data Types

The atomic data types used in this document are those as specified by Microsoft in the RIFF description [2]. The following is excerpted from that source:

Label	Meaning	C type
CHAR	8-bit signed integer	signed char
BYTE	8-bit unsigned quantity	unsigned char
INT	16-bit signed integer in Intel format	signed short int
WORD	16-bit unsigned integer in Intel format	unsigned short int
LONG	32-bit signed integer in Intel format	signed long int
DWORD	32-bit unsigned integer in Intel format	unsigned long int

The tag type used in RIFF files is further defined in that document thus:

```
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’);
```

4.2 Byte Ordering

The byte ordering used for the storage of multi-byte numeric data (INT, WORD, DWORD, LONG, etc.) in RIFF files is the “Intel” format (least significant byte first).

There is an associated RIFF format designed for “Motorola” format representation (most significant byte first), the “RIFFX” convention. The use of that representation or its suitability is beyond the scope of this proposal at the current time.

4.3 Character Set

The character set used in readable strings is described above as “ASCII.” The use of this term is exactly the same as the usage in the EBU Broadcast Wave Extension standard [3]. That document is unclear as to the exact set of characters specified. The set of characters belong to the set **isascii()** is unambiguous in this case and will be used pending a more formal definition that includes such possibilities as extended Roman sets and so forth. One suggested convention is the use of the Windows ANSI character set.

Multi-byte, Unicode and other representations are explicitly excluded from the defined fields. However, by agreement, producer and consumer application may employ these alternate representations in private data areas. Such private areas, of course, preclude portability.

4.4 Default Values and Empty Data

Unless otherwise specified, values are optional. For ASCII fields, this would constitute a zero length string signified by the first byte being a NULL character. For binary or numeric data not specified above, this would be a value of 0.

The action to be taken on encountering a blank or empty data is implementation and site/installation dependent.

4.5 Category Names

To a great extent, category names are not strictly constrained to the point where a standardized list can be enumerated. Often, the assignment of such names is specific to each site, affiliation or network. Further, there is a native language factor at work as well, and that needs to be accounted for. However, this is another area where there is a high degree of commonality, such that a list of standardized names can be suggested.

In addition to these names, the list below also suggests appropriate aliases for those system that may make use of such.

Category	Aliases
All	ALL
Beds	BED, BEDS
Sound bits	BIT, BITS
Commercials	COM, COMM
Contests	CON, CONT
Daily playlists	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

4.6 Mark Timer Identification

Below is a list of basic timer types, and their proposed (FOURCC) names. The basic types may be qualified by append “s” (lower-case) to signify a start, “e” (lower case) to signify an end, or a digit to signify a sequence.

Timer ID	Description	Start/End	Enumerated	Multiples
	Unused	No	No	Yes
SEG	Segue timer	Yes	Yes	Yes
AUD	Audio boundary	Yes	No	No
INT	Intro	Yes	Yes	Yes
OUT	Outtro	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 so designated in the timers can be specified in one of three ways:

- as start/end timers, by appending a lower case “s” for a start timer or a lower case “e” for an end timer. For example, the ID “AUDs” designates the start of audio following silence, while “AUDe” designates the end of the audio segment, followed by, say, silence.
- as enumerated timers, by appending an ASCII numeric character. “SEC1”, for instances, designated secondary timer number 1, “SEC2” is secondary number 2, and so on.
- as multiple timers, simply by having multiple instances of the same timer ID. One could have, for example, multiple instances of “MRK ”.

Each application should prioritize the order in which the timers are written so that the important timers, like EOD, will always be present and the less important ones, like “MRK”, can be dropped if the limit of 8 is reached.

An empty timer ID, consisting of a string of nulls (equal to a FOURCC value of 0x00000000), signals that particular timer is not in use. In this case, the corresponding timer value shall be set to CART_TIMER_UNSET (0xFFFFFFFF).

Multiple timers of the same name can be supported (some readers will only be able to use the first *n* occurrences). This convention leaves more space for the name itself and also ensures that contiguous numbered timers are stored. For example, if two tertiary timers were to be written, the following code fragment could be used.

```
dwPostTimer[0].usage = cvtFOURCC('T', 'E', 'R', 's');
dwPostTimer[0].value = 322000;
dwPostTimer[1].usage = cvtFOURCC('T', 'E', 'R', 'e');
dwPostTimer[1].value = 322999;

dwPostTimer[2].usage = cvtFOURCC('T', 'E', 'R', 's');
dwPostTimer[2].value = 453000;
dwPostTimer[3].usage = cvtFOURCC('T', 'E', 'R', 'e');
dwPostTimer[3].value = 453999;
```

4.7 Standards Process

This proposal was presented to the AES standards committee SC-06-01 (File Interchange) at the AES convention in New York in September 1999 as project AES-X87.

5 REFERENCES

[1] Microsoft Software Developers Kit Multimedia Standards Update, rev 3.0 15 April 1994
[2] Microsoft Multimedia Programmer's Reference
[3] "Specification of the Broadcast Wave Format," EBU Tech. Doc. 3285, EBU Publications, Ancienne Route 17A, CH-1218 Grand Saconnex (Geneva) Switzerland, July 1997[12]
[4] "Specification of the Broadcast Wave Format, Supplement 1: MPEG Audio" EBU Tech. Doc. 3285A, EBU Publications, Ancienne Route 17A, CH-1218 Grand Saconnex (Geneva) Switzerland, July 1997

[12] EBU publications can be obtained either by mailing the EBU Publications address or from the EBU publications web site at http://www.ebu.ch/publications.html