EBU

OPERATING EUROVISION AND EURORADIO

TECH 3285-s7

SPECIFICATION OF THE BROADCAST WAVE FORMAT - A FORMAT FOR AUDIO DATA FILES IN BROADCASTING

Supplement 7: <chna> Chunk

Geneva
May 2018

This and other pages in the document are deliberately left blank to facilitate two-sided printing.

Contents

Specification of the Broadcast Wave Format - a format for audio data files

Supplement 7: <chna> Chunk

EBU Committee	*First Issued*	*Revised*	*Re-issued*
TC	2015	2018[1]	

Keywords: Audio file format, BWF, RF64, BF64, Multichannel Broadcast Wave File, <chna> chunk.

1. Introduction

The primary purpose of the <chna> chunk described in this document is to provide the references from each track in a BWF [1] or BW64 [2] file to the IDs in the Audio Definition Model (ADM) metadata defined in ITU-R BS.2076 [3].

Apart from the primary purpose of linking each track in the file with its associated ADM metadata, the <chna> chunk also allows faster access to ADM IDs without having to gain access the XML metadata (if the IDs are within a range of values defined by the Common Definitions for the ADM in ITU-R BS.2094 [5]). As the <chna> chunk can be fixed in size, and is placed before the <data> and <axml> chunks, it is easier to access, generate or modify its contents on the fly.

2. Terminology

ADM Audio Definition Model – a metadata model for describing the format and content of audio files (ITU-R BS.2076 [3]).

<chna> chunk A chunk containing a set of IDs for each track in the file. These IDs will either refer to ADM elements, or be referred to from an ADM element.

Track A track is a where a sequence of samples (or coded data) is stored in the <data> chunk of the BWF file. In multi-track files the samples are interleaved.

3. CHNA chunk

3.1 Definition

The <chna> chunk consists of a header followed by the number of tracks and number of track UIDs used. This is followed by an array of ID structures that each contains IDs corresponding to ADM element IDs. The ADM IDs within the chunk can either refer to ADM metadata carried in the <axml> chunk, or in an external common definition file. If the last four hexadecimal digits of the IDs have a

[1] References updated. [3] contains a definition of the <chna> chunk that this supplement is completely aligned with.

value of 0x0FFF and below then they are defined as common definitions in *Recommendation ITU-R BS.2094-0 – Common Definitions for the Audio Definition Model* (for example channel definitions for 'FrontLeft' and 'FrontRight'). Any IDs with values of 0x1 000 and above are defined as custom definitions and so will be contained in the <axml> chunk within the file.

The size of the chunk depends upon the number of track UIDs to be defined. The number of ID structures must be equal to or greater than the number of track UIDs used. By allowing the number of ID structures to exceed the number of UIDs, it can facilitate updating and adding new IDs to the chunk without having to change the size of the chunk.

For example, it may not be clear how many UIDs will be generated at the beginning, so if the number of ID structures in the chunk is set to 64 (as this is considered by the implementer to be more than enough for their task); the software then generates 55 UIDs (an example number of initial UIDs) which fill up the first 55 ID structures, so the remaining 9 ID structures are set to zero values.

The **audioID** structure contains an index to the track used in the <data> chunk (which contains the audio samples), starting with the value of 1 for the first track. It contains a UID for the track, which the ADM metadata will contain.

The audio elements of a track may be differently in the course of a file; in this case, there will be a different UID for each definition. Therefore it is possible to have multiple UIDs for each track. The other two values in the structure are references to the IDs of the ADM's **audioTrackFormat** and **audioPackFormat** elements.

typedef struct chna			
{			
	CHAR	**ckID[4];**	**// {'c','h','n','a'}**
	DWORD	**ckSize;**	**// size of chunk**
	WORD	**numTracks;**	**// number of tracks used**
	WORD	**numUIDs;**	**// number of track UIDs used**
	audioID	**ID[N];**	**// IDs for each track (where N >= numUIDs)**
}			
chna_chunk; **typedef struct audioID**			
{			
	WORD	**trackIndex;**	**// index of track in file**
	CHAR	**UID[12];**	**// audioTrackUID value**
	CHAR	**trackRef[14];**	**// audioTrackFormatID reference**
	CHAR	**packRef[11];**	**// audioPackFormatID reference**
	CHAR	**pad;**	**// padding byte to ensure even number of bytes**
}			

3.2 Elements of the 'chna' chunk

ckID This is the 4 character array {'c','h','n','a'}[2] for chunk identification.

ckSize This is the size of the data section of the chunk. (It does not include the 8 bytes used by **ckID** and **ckSize**.)

numTracks The number of tracks used in the file. Even if a track contains more than one set of IDs, it is still just one track.

numUIDs The number of UIDs used in the file. As it is possible to give a single track multiple UIDs (covering different time periods), this could be a greater value than **numTracks**. This value should match the number of defined IDs in **ID**.

ID The structure containing the set of audio reference IDs for the track. This array contains N IDs, where N ≥ numUIDs. When **numUIDs** is less than N the contents of the unused track IDs are set to zero. When reading the chunk the value of N can be derived from **ckSize**, as **ckSize** = 4 + (N * 40), so N = (**ckSize -** 4) / 40.

trackIndex The index of the track in the file, starting at 1. This corresponds directly to the order of the tracks interlaced in the <data> chunk.

UID The **audioTrackUID** value of the track. The character array has the format ATU_xxxxxxxx where x is a hexadecimal digit.

trackRef The **audioTrackFormatID** reference of the track. The character array has the format AT_xxxxxxxx_xx where x is a hexadecimal digit.

packRef The **audioPackFormatID** reference of the track. The character array has the format AP_xxxxxxxx where x is a hexadecimal digit. When **audioPackFormatID** is not required (when **audioStreamFormat** is referring to an **audioPackFormat** rather than an **audioChannelFormat**) this field should be filled with null values.

pad A single byte to ensure the **audioID** structure has an even number of bytes.

When an ID is not being used the **trackIndex** should be given the value of zero and the other fields should be given null strings that are the same length as the usual ID string used. So the null string for **packRef** would consist of 11 null characters (ASCII value zero) and **trackRef** would consist of 14 null characters.

4. Examples

To help illustrate the operation of the <chna> chunk some simple examples are given here.

The pseudo-code in each example uses string-like notation for the IDs (e.g. "AT_00010001_01"), whereas in practice an array of characters should be used to ensure correct ordering of the characters (so it would actually be done this way:
{'A','T','_','0','0','0','1','0','0','0','1','_','0','1'}).

[2] **Remark:** The definition DWORD ckID = "chna" would not be unique. Different architectures produce different orders of the characters. Therefore we define char ckID[4] = {'c','h','n','a'} instead.

4.1 Simple stereo file

The majority of audio files in existence are still 2-channel stereo files, with the first track containing the left channel, and the second track containing the right channel. The ADM has a definition of a left channel with an ID of AT_00010001_01, and the right channel with an ID of AT_00010002_01. The stereo pack definition has the ID of AP_00010002.

```
ckID = {'c','h','n','a'};
ckSize = 84;
numTracks = 2;
numUIDs = 2;
ID[0] = { trackIndex = 1; UID = "ATU_00000001"; trackRef = "AT_00010001_01"; packRef = "AP_00010001"; pad = '\0`; };
ID[1] = { trackIndex = 2; UID = "ATU_00000002"; trackRef = "AT_00010002_01"; packRef = "AP_00010001"; pad = '\0`; };
```

The number of ID structures is 2, so there are no unused ID structures in this example.

4.2 Coded 5.1 with stereo versions

Non-PCM audio could be stored in the file, an example would be Dolby E carrying a coded version of 5.1 audio in two tracks. This example shows how 6 tracks contain Dolby E 5.1 with two stereo pairs (maybe a stereo mix of the 5.1 and an alternative language version).

```
ckID = {'c','h','n','a'};
ckSize = 244;
numTracks = 6;
numUIDs = 6;
ID[0] = { trackIndex = 1; UID = "ATU_00000001"; trackRef = "AT_00020001_01"; packRef = ['\0']*11;     pad = '\0`; };
ID[1] = { trackIndex = 2; UID = "ATU_00000002"; trackRef = "AT_00020001_02"; packRef = ['\0']*11;     pad = '\0`; };
ID[2] = { trackIndex = 3; UID = "ATU_00000003"; trackRef = "AT_00010001_01"; packRef = "AP_00010001"; pad = '\0`; };
ID[3] = { trackIndex = 4; UID = "ATU_00000004"; trackRef = "AT_00010002_01"; packRef = "AP_00010001"; pad = '\0`; };
ID[4] = { trackIndex = 5; UID = "ATU_00000005"; trackRef = "AT_00010001_01"; packRef = "AP_00010001"; pad = '\0`; };
ID[5] = { trackIndex = 6; UID = "ATU_00000006"; trackRef = "AT_00010002_01"; packRef = "AP_00010001"; pad = '\0`; };
```

The first two tracks (1 & 2) contain the Dolby E data, so the **trackRefs** contains the IDs AT_00020001_01 and AT_00020001_02. The way these **audioTrackFormat** IDs are designed means that these two IDs both reference an **audioStreamFormat** ID of AS_00040001. This stream (with the 0004 set of digits) is known to be Dolby E data and contain 5.1 coded channels. No **packRef** ID is required as the pack is known to be 5.1 from the stream reference.

The next pair of tracks (3 & 4) is the first stereo pair, and the 5 & 6 pair of tracks is the second stereo pair. Both pairs have the same **trackRef** and **packRef** IDs as they have the same format.

4.3 Simple object-based example

Audio objects may only cover a sub-section of time in the audio file. To save space, non-overlapping objects may share the same track. This is where multiple UIDs in the same track would occur. This example also uses more ID structures (32 in this case) than numUIDs to show how unused ID structures are set to zero.

```
ckID = {'c','h','n','a'};
ckSize = 1284;
numTracks = 2;
numUIDs = 4;
ID[0]={ trackIndex=1; UID="ATU_00000001"; trackRef="AT_00031001_01"; packRef="AP_00031001"; pad='\0`; };
ID[1]={ trackIndex=1; UID="ATU_00000002"; trackRef="AT_00031003_01"; packRef="AP_00031002"; pad='\0`; };
```

```
ID[2]={ trackIndex=1; UID="ATU_00000003"; trackRef="AT_00031004_01"; packRef="AP_00031003"; pad='\0`; };
ID[3]={ trackIndex=2; UID="ATU_00000004"; trackRef="AT_00031002_01"; packRef="AP_00031001"; pad='\0`; };
ID[4]={ trackIndex=0; UID['\0']*12;      trackRef['\0']*14;        packRef['\0']*11;     pad='\0`; };
:
ID[31]={ trackIndex=0; UID['\0']*12;     trackRef['\0']*14;        packRef['\0']*11;     pad='\0`; };
```

The first track contains 3 UIDs, so will contain 3 different objects (with the track IDs of AT_00031001_01, AT_00031003_01 and AT_00031004_01) at different time locations within the file. The second track contains one UID, so contains one object. This object has the same pack ID (AP_00031001) as the first object in track 1. This suggests the first object contains two channels carried in both track 1 and track 2. The ADM metadata carried in the <axml> would be used to clarify the allocation of channels and tracks.

5. Handling files without a <chna> Chunk

Any files that are generated without the <chna> chunk (including all files that pre-date this document) should be handled pragmatically, particularly if a <chna> chunk is to be automatically generated for the file. Assuming that no other knowledge of the track allocation in a WAV/BWF file exists, then a default set of IDs for the <chna> can be assigned when generating the chunk.

The number of tracks in a file will be known. The assumption made will be that the file contains PCM audio in a channel-based format. The order of the tracks will simply be the track IDs taken in numerical order from 00010001_01 (the first four digits represent the type of track, and so they don't change; nor does the two digit suffix). As we don't know the pack to which the channels are assigned to, this is left null. These ID digits are hexadecimal numbers.

So for a 6 track file the automatically generated chunk will be:

```
ckID = {'c','h','n','a'};
ckSize = 244;
numTracks = 6;
numUIDs = 6;
ID[0] = { trackIndex = 1; UID = "ATU_00000001"; trackRef = "AT_00010001_01"; packRef = ['\0']*11; pad = '\0`; };
ID[1] = { trackIndex = 2; UID = "ATU_00000002"; trackRef = "AT_00010002_01"; packRef = ['\0']*11; pad = '\0`; };
ID[2] = { trackIndex = 3; UID = "ATU_00000003"; trackRef = "AT_00010003_01"; packRef = ['\0']*11; pad = '\0`; };
ID[3] = { trackIndex = 4; UID = "ATU_00000004"; trackRef = "AT_00010004_01"; packRef = ['\0']*11; pad = '\0`; };
ID[4] = { trackIndex = 5; UID = "ATU_00000005"; trackRef = "AT_00010005_01"; packRef = ['\0']*11; pad = '\0`; };
ID[5] = { trackIndex = 6; UID = "ATU_00000006"; trackRef = "AT_00010006_01"; packRef = ['\0']*11; pad = '\0`; };
```

If a WAV/BWF file that is being read doesn't contain a <chna> chunk, then we use the sequential IDs for the tracks as shown above. Therefore track 1 will have the definition for AT_00010001_01 (FrontLeft, PCM).

This default order of IDs allows backwards compatibility with the WAVEFORMATEXTENSIBLE[3] channel definitions. If the standard definitions for AT_00010001_01 to AT_0001012_01 (there are 18 channels defined in WAVEFORMATEXTENSIBLE) match those of the WAVEFORMATEXTENSIBLE specification, then this compatibility can be ensured.

While no pack information is used, it could be possible for the user/tools to make assumptions on the pack used from the number of the tracks in the file. For example, if it is a 6 track file, the audio pack ID for the 5.1 format could be used (e.g. AP_00010003).

[3] https://msdn.microsoft.com/en-us/library/windows/desktop/dd757714%28v=vs.85%29.aspx

6. Bibliography

[1] EBU Tech 3285: Specification of the Broadcast Wave Format – A format for audio files in broadcasting.

[2] ITU-R BS.2088-0 (2015): Long Form File Format for the International Exchange of Audio Programme Materials With Metadata

[3] ITU-R BS.2076-1 (2017): Audio Definition Model

[4] EBU Tech 3293: EBU Core Metadata Set