AudioMoth Triggered T.WAV File Format

theteam@openacousticdevices.info

4th January 2021

Version 1.4.0 of the AudioMoth firmware introduced a file format for triggered recordings that allows silent periods between periods of audio content to be encoded within a standard WAV file. This allows triggered recordings to be made without suffering from *lost periods* between recordings where the device is closing and re-opening files and cannot also be listening for acoustic events. The first use of this file format was the amplitude threshold recording functionality included in the same version of firmware. This document describes the file format and how the triggered recordings can be expanded to their original length whilst maintaining the original timing of events within the recording.

1 Motivation

Many acoustic recorders allow for recordings to be made in response to a trigger event. These events may be external, such as from a motion sensor, or more commonly internal, using features of the sound that the recorder is itself detecting. For example, a recording may be initiated whenever the amplitude exceeds a fixed threshold. This feature saves storage space and also battery consumption as writing to permanent storage is typically an energy intensive process.

When triggered events are written to permanent storage in separate files a number of problems become apparent. First, opening and closing files on media such as SD cards typically takes several seconds, and thus, to avoid lost periods between recordings significant buffering is required. At high sample rates it is often not possible to provide sufficient buffering and acoustic events may be lost. Second, unless timestamps on the files are very accurate, the precise timing of events that are recorded in separate files is lost. Finally, the multiple files that may be generated are somewhat inconvenient to handle and process uniformly.

To address these issues AudioMoth uses a custom T.WAV file format to handle triggered recordings. Rather than splitting triggered events into separate files at record time, a single T.WAV file is generated on the device. This file contains each triggered recording along with silent meta-data that encodes the length of non-triggered periods between recordings. When viewed or played with existing WAV tools, only the triggered recordings are visible and audible. However, the silent unrecorded intervals can be re-inserted in post-processing to restore accurate sample-level inter-event timings. The single file can then be split into multiple shorter files as required for any particular processing task.

2 File Creation

When writing T.WAV files, AudioMoth devices process audio in segments of L bytes[1] and determine if each individual segment should be written to the SD card or not. In the case of the amplitude threshold recording this is determined by whether or not the maximum amplitude of the audio within the segment exceeds the specified threshold.

If the segment is to be written to the SD card, it is written immediately as a block of L bytes representing $L/2$ samples; each encoded as a 16-bit signed integer. If the current segment, and any subsequent segments, are not to be written to the SD card, AudioMoth counts how many bytes have been skipped, and then when it next writes a segment to the SD card, it precedes this segment with an additional 512-byte block that encodes the number of skipped bytes.

In this way, the 512-byte encoded block can represent a silent period of any length; significantly reducing the length of the file written to the SD card. The 512-byte encoded block is read by conventional WAV tools as standard 16-bit signed integer audio samples. By ensuring that the encoded block contains only 16-bit signed integers that map to -1, 0 or 1, the block is effectively made silent when viewed or played with standard WAV tools.

Figure 1 which shows an example T.WAV file with two segments of length L bytes and two 512-byte encoded blocks. Note that the first and last segments are special cases. The first segment is overwritten by the WAV header at the start of the file, while the last segment may be less than L bytes in length (or may be omitted entirely) in order to accommodate the recording length requested by the user.

Since the first and last segments cannot be included within silent unrecorded intervals, a different approach is taken when they are do not satisfy the condition to be written to the SD card. In this case, segments of equal

[1] L is currently 32kB.

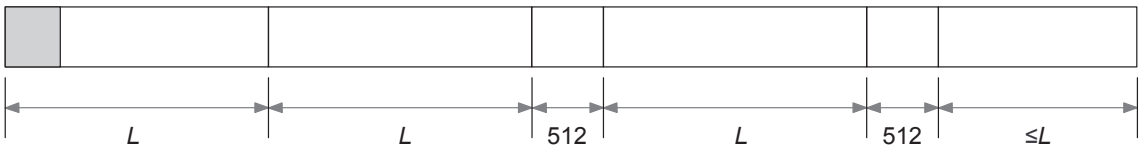

Figure 1: *Example T.WAV file. The shaded area represents the file header.*

0 - 3	"RIFF"	RIFF chunk ID
4 - 7	Unsigned 32-bit integer	RIFF chunk size $(N - 8)$
8 - 11	"WAVE"	WAVE ID
12 - 15	"fmt "	FORMAT chunk ID
16 - 19	Unsigned 32-bit integer	FORMAT chunk size (16)
20 - 21	Unsigned 16-bit integer	PCM format (1)
22 - 23	Unsigned 16-bit integer	Number of channels (1)
24 - 27	Unsigned 32-bit integer	Samples per second (n)
28 - 31	Unsigned 32-bit integer	Bytes per second $(2 \times n)$
32 - 33	Unsigned 16-bit integer	Bytes per capture (1)
34 - 35	Unsigned 16-bit integer	Bits per sample (16)
36 - 39	"LIST"	LIST chunk ID
40 - 43	Unsigned 32-bit integer	LIST chunk size (436)
44 - 47	"INFO"	INFO ID
48 - 51	"ICMT"	ICMT chunk ID
52 - 55	Unsigned 32-bit integer	ICMT chunk size (384)
56 - 439	384-byte string	Comment string
440 - 443	"IART"	IART chunk ID
444 - 447	Unsigned 32-bit integer	IART chunk size (32)
448 - 479	32-byte string	Artist string
480 - 483	"data"	DATA chunk ID
484 - 487	Unsigned 32-bit integer	DATA chunk size $(N - 488)$

Table 1: *Header format for an AudioMoth WAV file which is N bytes in length and is recorded at n samples per second.*

0 - 1	Signed 16-bit integer	-1 or 1 - Bit 0
:	:	:
62 - 63	Signed 16-bit integer	-1 or 1 - Bit 31
64 - 65	Signed 16-bit integer	0
:	:	:
510 - 511	Signed 16-bit integer	0

Table 2: *Format of the 512-byte encoded block.*

length are written where all 16-bit signed integers are set to zero.

3 File Format

The length of the audio segments, L, is 32kB in the current firmware implementation. This is currently determined by the internal use of SRAM within the AudioMoth and it may take other values as long as it satisfies the constraint that it is a multiple of 512 bytes and is longer than the WAV header.

Table 1 shows the format of the WAV header applied to the T.WAV file. The current header is 488 bytes long. Most of the header structure is fixed by the RIFF and WAV formats themselves. However, there are two text fields representing the header comment and the artist fields which may be subject to changes in length in future versions of the firmware. Thus, code manipulating these files should read the location and length of these fields from the header itself.

Table 2 shows the format of the 512-byte encoded block. The first 32 16-bit signed integers in this block represent the bits of a 32-bit value (encoded least significant bit first with -1 and 1 mapping to 0 and 1) that represents the length, in multiples of 512 bytes, of the silent non-recorded period. The remaining 224 16-bit signed integer values are all always set to zero.

4 File Expansion

Expansion of a T.WAV file can simply be performed by reading the original file in chunks of 512 bytes starting at the beginning of the file. Each 512-byte chunk can be compared against the format of the 512-byte encoded block. If it encodes a valid encoded block, it is replaced by the appropriate number of zero filled 512-byte chunks, otherwise the original chunk is written to the output. It is then necessary to update the WAV header to reflect the size of the new expanded file. This requires an update to the size of the RIFF chunk and the DATA chunk (see Table 1) either by calculating the additional number of bytes written and summing this to the original values, or by working backwards from the total size of the final file.

A reference implementation in C is provided (see `expand.c` in this repository). The code can be compiled using `gcc` in macOS, Linux, or Windows Subsystem for Linux.

```
> gcc -o expand expand.c
```

Any T.WAV file can then be expanded by providing the filename as an argument.

```
> ./expand 20201103_112430T.WAV
```

The code performs a direct expansion of a T.WAV into the equivalent WAV file and updates the header appropriately to reflect the size of the expanded file; parsing the WAV header so that no header specific details (such as the length of the comment field) are hard-coded

Expansion of T.WAV files is also supported within the AudioMoth Configuration App. This tool provides additional options to allow the resulting expanded file to be split into smaller files as required for any particular processing toolchain.