Acoustic Data Support at LISHA's IoT Platform

SmartData containing raw sound samples, preserving all features, can be sent to the platform, using the Digital Unit of type 2, subtype 35 (as described here). The processing applied to the data is based on specific microphone's specifications.
The WAV files that are sent to the platform using 32 bits per sample encoding, are compressed/inflated on-the-fly at the platform. The WAV file's header is exposed on the table below, according to this reference.

WAV Header

Field	Header Value	Meaning
Chunk Id	5249 4646	'RIFF'
Chunk Size	26e2 0400	320038
Format	5741 5645	'WAVE'
SubChunk1 Id	666d 7420	'fmt '
SubChunk1 Size	1200 0000	18
Audio Format	0100	1 = PCM
Number of Channels	0100	1
Sample Rate	204e 0000	20000
Byte Rate	8038 0100	80000
Block Align	0400	4
Bits Per Sample	2000	32
-	0000	Bytes 38 and 39 are not mapped
SubChunk2 Id	6461 7461	'data'
SubChunk2 Size	00e2 0400	320000

To save storage capacity, once the microphone's characteristics do not demand more than 16 bits to encode sensed data, the WAV data received by the platform are encoded with 16 bits per sample, and then compressed using the **Free Lossless Audio Codec FLAC** . When data is retrieved from the platform, the data is read from the database, and restored to the original format of 32 bits per sample WAV. The **ffmpeg** tool is used to compress / restore the data.

SmartData and Time Series

The sound data (the **value** field of the object) sent to the platform using SmartData in JSON format has to be encoded in **Base64** format. The data returned by a **get()** command will also be encoded this way.

An exemplo of JSON that should be sent to insert a WAV data into the platform, using **put.php**

```
'smartdata' : [ { 'version' : '1.1', 'unit' : 0x02230000, 'value' : <base64 encoded data>, 'error' : 0, 'confidence' : 0, "x" : 123123123, "y" : 123112233, "z" : 83728372, 't' : 1509391500000000, 'dev' : 1 } ]
```

In order to retrieve the WAV data, use the **get.php**, providing the same unit, coordinates and device:

```
'series' : { 'version' : '1.1', 'unit' : 0x02230000, 'x' : 123123123, 'y' : 123112233, 'z' : 83728372, 'r' : 0, 't0' : 1509390000000000, 't1' : 1509392000000000, 'workflow' : 0, 'dev' : 1 }
```

Remarks and Evaluation

The **pyflac** library and the **FFMpeg** library (a wrapper for the ffmpeg tool) were also evaluated. The first also demand a call to an external script, and the latter has some broken dependencies (from older PHP versions), and needs files as input. Therefore, no advantages from calling directly the external **ffmpeg** tool are evident, and no pre-processing (rescaling) is needed before applying compression, as ffmpeg make all necessary processing.

The evaluation was made with 8 sample files. After compressing and restoring each file, the maximum difference between the original data and the restored data was measured. The Peak Signal to Noise Ratio (PSNR) was also calculated for each restored file, compared to the original data. As shown in the table below, the **FFMpeg** and **pyflac** shown very similar values for the PSNR results. Therefore, the **FFMpeg** is employed in this implementation of the platform compression/decompression tool. All files have 320046 bytes size, and the compression shrink them to an average size of 136150 bytes (42,54% of the original size), achieving a good compression ratio.

File name	PyFlac Max Error	PyFlac PSNR	FFMpeg Max Error	FFMpeg PSNR	Compressed size
Sample1.wav	65535	110,33	65535	110,33	134720 (42,09%)
Sample2.wav	65535	110,42	65535	110,42	133845 (41,82%)
Sample3.wav	65535	110,23	65535	110,23	146540 (45,79%)
Sample4.wav	65535	110,18	65535	110,18	134416 (41,99%)
Sample5.wav	65535	110,28	65535	110,28	135859 (42,45%)
Sample6.wav	65535	110,09	65535	110,10	136130 (42,53%)