PRESTO – Preservation Technologies for European Broadcast Archives

IST-1999-20013

AUDIO KEY LINKS: LOSSLESS COMPRESSION

Authors: Daniele AIROLA, Salvatore CANGIALOSI and Giorgio Dimino (RAI)

1. Summary

2. Introduction

3. Key link description and requirements

The main tasks of this activity are the selection of a suitable coding strategy between those proposed in the literature and the performance of tests to verify if the compression ratio achieved is such to justify the inclusion of such a tool in the reference transcription chain. In case of positive results, then, an operative scenario will be devised and experimented.

The main requirements for a lossless compression tool to be included in an audio preservation system are:

- The decompressed signal must be bit-to-bit equal to the original
- It must be possible to implement the compression method in real time
- The compression algorithm must be fully specified and possibly standardized by a proper International Body (this activity extends beyond the PRESTO scope, and can only be promoted here)
- The decompression tool should preferably be implemented in software and run faster than real time on a conventional PC
- The decompression tool should be platform independent
- In no case the compressed signal should result in a file larger than the original
- Partial file access should be possible
- A proper encapsulation in the BWF container must be defined

4. Algorithm selection

4.1. Available codecs

Doing an internet search for lossless audio coding, a good deal of software tools can be found. Some of them require a licence, others are freely usable and in few cases full documentation and source code is available on line.
Here is a list of codecs with the corresponding URLs:

- Flac - An open source codec. Available for Windows and Unix, with Winamp and XMMS plugins. The source code is available at http://flac.sourceforge.net.
- FROG - A closed source, Windows-only codec, with a Winamp plugin. Available a http://ghido.shelter.ro.
- Kexis - An open-source source codec. Still in the alpha stage. No player or library support yet. See http://sourceforge.net/projects/kexis.
- LPAC - A closed source codec. Available for Windows and Unix, with a Winamp plugin. http://www-ft.ee.tu-berlin.de/~liebchen/lpac.html.
- Monkey's Audio - A closed source, Windows-only codec, with a Winamp plugin. Available at http://www.monkeysaudio.com.
- Pegasus-SPS - A closed source, Windows-only codec. http://www.jpg.com/products/sound.html
- RKAU - A closed source, Windows-only codec. http://rksoft/virtualave.net.
- Shorten - A.J. Robinson's well-known codec; source is available. http://www.softsound.com/shorten.html.
- WaveZIP - A closed source, Windows-only archiver. Uses the MUSICompress[tm] engine which supposedly has a patent. See http://members.aol.com/_ht_a/sndspace/index.html
- WavPack - A closed source, Windows-only archiver. Available at http://www.wavpack.com.

The above list is certainly not exhaustive, but exploring the provided URLs, a good idea of the variety of tools available and their varying state of development can be obtained. Some packages are stable commercial software, others are university research projects, others are even individual developments.

4.2. Feature comparisons

The codecs listed in the previous section have been compared against the list of PRESTO requirements, in order to verify if any of them can be suitable to professional archiving applications. The table below summarises the results.

	Specification or source code available	24 bit samples support	Standard	Supported platform	Partial file access	User metadata provision	Real time (or faster) coding and decoding
Flac	Yes	Yes	No	Any	Yes	Yes	Yes
Monkey's Audio	No	Yes	No	Windows	Yes	No	Yes

RKAU	No	Yes	No	Windows	Yes	No	Yes
Shorten	Yes	No	No	Any	No	No	Yes
WavPack	No	Yes	No	Windows	Yes	No	Yes
Kexis	Yes	??	No	Any	No	No	Yes
LPAC	No	Yes	No	Windows, Linux, Solaris	Yes	No	Yes
FROG	No	??	No	Windows	Yes	??	Yes
Pegasus-SPS	No	??	No	Windows	No	??	Yes

It can be noted that all the codecs can be run under Microsoft Windows, while only a few support Linux and other Unix systems. None of them has been recommended by the official Standards Bodies, as still no major professional application has demanded this kind of tools. Only the tools that have reached a stable state of development and that are able to handle 24 bit samples and providing partial file access have been considered for further tests.

4.3. Performance tests

Five of the above listed codecs have been chosen for performance tests. The tests included both coding efficiency and speed. The efficiency has been assessed by encoding a selection of music materials of various genres digitised at 24 bit per sample from 33 RPM vinyl sources and then computing the ratio between the storage occupation of the coded file with respect to that of the original one. The coding speed has been estimated by timing the encoding process on a reference platform, that is a PC equipped with a PIII processor clocked at 1 GHz under Windows 2000 operating system.
All the codecs can be configured according to several options. Verified that for all the codecs no significant efficiency improvement can be obtained by switching on the so called "high quality features", while these generally caused a dramatic increase in computing time, the settings recommended for high coding speed have been adopted. These settings often coincide with the defaults.
The tests have been run according to two different procedures:

1. Coding of 24 bit per sample materials
2. Coding of a 16 bit per sample version of the material obtained by removing the 8 least significant bits from the 24 bit source

For comparison, a well known general purpose lossless codec (pkzip) has been included in the test.

Here is the list of codecs tested:

- Flac
- LPAC
- WavPack
- RKAU
- Monkey's Audio
- PKZIP

The corpus of test materials is described in the following table:

Name	From	Duration (s)
Barry_intertraccia	BARRY WHITE- "LET THE MUSIC PLAY"	176

Caro_pianoforte	AA.VV. - "CARO PIANOFORTE"	1183
Chopin_notturni	CHOPIN - "NOTTURNI"	1477
Dire	DIRE STRAITS - "LOVE OVER GOLD"	396
I_dont_know_where_love_has_gone	BARRY WHITE - "LET THE MUSIC PLAY"	300
I_soblue_and_you_are_too	BARRY WHITE - "LET THE MUSIC PLAY"	438
Pape_satan	FABIO FABOR - "PAPE SATAN"	1185
Patchwork	THE PHANTOMS - "PATCHWORK"	1268

The test platform was configured as follows:
CPU: Pentium III @ 1 GHz
RAM: 256 MB DDR
Disk: 40 Gbyte IBM 7200 RPM - EIDE
VGA: MATROX G400
Operating System: Microsoft Windows 2000

The test results are contained in the following tables and summarised in the graphs of Figure 1 and Figure 2.

Coder FLAC									
		24 bit				**16 bit**			
Sequence title	**Duration (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**
Barry_intertraccia	176	50.697.360	30.185.163	0,60	11	33.798.288	13.297.194	0,39	9
Caro_pianoforte	1183	340.722.212	166.403.375	0,49	80	227.148.324	61.032.380	0,27	60
Chopin_notturni	1477	425.657.034	209.383.556	0,49	98	283.771.594	76.643.389	0,27	76
Dire	396	114.250.944	63.914.832	0,56	27	76.167.358	25.369.462	0,33	21
I_dont_know_where_love_has_gone	300	86.655.168	63.151.294	0,73	21	57.770.174	29.988.563	0,52	16
I_soblue_and_you_are_too	438	126.369.954	73.650.933	0,58	30	84.246.690	32.172.983	0,38	23
Pape_satan	1185	341.312.126	206.005.859	0,60	81	227.541.630	88.758.730	0,39	63
Patchwork	1268	365.298.434	228.378.934	0,63	87	243.532.546	105.306.213	0,43	67
TOTAL	**6423**	**1.850.963.232**	**1.041.073.946**	**0,56**	**435**	**1.233.976.604**	**432.568.914**	**0,35**	**335**

Coder LPAC									
		24 bit				**16 bit**			
Sequence title	**Duration (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**
Barry_intertraccia	176	50.697.360	28.698.004	0,57	14	33.798.288	12.763.860	0,38	10
Caro_pianoforte	1183	340.722.212	161.163.129	0,47	90	227.148.324	53.852.702	0,24	100
Chopin_notturni	1477	425.657.034	205.667.402	0,48	108	283.771.594	72.329.838	0,25	75
Dire	396	114.250.944	60.067.354	0,53	30	76.167.358	24.212.898	0,32	22
I_dont_know_where_love_has_gone	300	86.655.168	57.874.612	0,67	26	57.770.174	29.306.516	0,51	19
I_soblue_and_you_are_too	438	126.369.954	71.095.359	0,56	35	84.246.690	30.633.791	0,36	25
Pape_satan	1185	341.312.126	195.250.157	0,57	99	227.541.630	84.460.670	0,37	75
Patchwork	1268	365.298.434	218.712.155	0,60	103	243.532.546	100.553.927	0,41	79
TOTAL	**6423**	**1.850.963.232**	**998.528.172**	**0,54**	**505**	**1.233.976.604**	**408.114.202**	**0,33**	**405**

Coder Monkey's Audio									
		24 bit				16 bit			
Sequence title	**Duration (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**
Barry_intertraccia	176	50.697.360	28.611.160	0,56	7	33.798.288	12.732.808	0,38	6
Caro_pianoforte	1183	340.722.212	162.285.380	0,48	43	227.148.324	55.558.068	0,24	29
Chopin_notturni	1477	425.657.034	206.604.770	0,49	56	283.771.594	73.407.642	0,26	37
Dire	396	114.250.944	60.370.778	0,53	15	76.167.358	24.632.818	0,32	11
I_dont_know_where_love_has_gone	300	86.655.168	57.422.174	0,66	13	57.770.174	28.988.042	0,50	9
I_soblue_and_you_are_too	438	126.369.954	70.818.506	0,56	17	84.246.690	30.488.206	0,36	13
Pape_satan	1185	341.312.126	198.841.030	0,58	48	227.541.630	88.504.402	0,39	33
Patchwork	1268	365.298.434	218.466.602	0,60	49	243.532.546	100.580.710	0,41	36
TOTAL	**6423**	**1.850.963.232**	**1.003.420.400**	**0,54**	**248**	**1.233.976.604**	**414.892.696**	**0,34**	**174**

Coder Wavepack									
		24 bit				16 bit			
Sequence title	**Duration (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**
Barry_intertraccia	176	50.697.360	29.881.699	0,59	11	33.798.288	14.150.760	0,42	8
Caro_pianoforte	1183	340.722.212	171.202.094	0,50	66	227.148.324	71.028.933	0,31	42
Chopin_notturni	1477	425.657.034	213.941.607	0,50	83	283.771.594	90.806.634	0,32	52
Dire	396	114.250.944	62.306.408	0,55	21	76.167.358	28.517.534	0,37	14
I_dont_know_where_love_has_gone	300	86.655.168	59.160.491	0,68	20	57.770.174	30.545.833	0,53	12
I_soblue_and_you_are_too	438	126.369.954	75.249.836	0,60	25	84.246.690	34.787.095	0,41	16
Pape_satan	1185	341.312.126	204.274.969	0,60	72	227.541.630	96.119.242	0,42	44
Patchwork	1268	365.298.434	231.655.528	0,63	80	243.532.546	112.308.155	0,46	33
TOTAL	**6423**	**1.850.963.232**	**1.047.672.632**	**0,57**	**378**	**1.233.976.604**	**478.264.186**	**0,39**	**221**

Coder RKAU									
		24 bit				16 bit			
Sequence title	**Duration (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**
Barry_intertraccia	176	50.697.360	28.699.178	0,57	40	33.798.288	12.527.887	0,37	37
Caro_pianoforte	1183	340.722.212	162.004.915	0,48	222	227.148.324	52.682.604	0,23	201
Chopin_notturni	1477	425.657.034	205.345.945	0,48	282	283.771.594	69.526.932	0,25	278
Dire	396	114.250.944	59.920.972	0,52	88	76.167.358	23.438.016	0,31	72
I_dont_know_where_love_has_gone	300	86.655.168	58.079.502	0,67	82	57.770.174	28.970.656	0,50	63
I_soblue_and_you_are_too	438	126.369.954	70.946.736	0,56	93	84.246.690	29.853.373	0,35	91
Pape_satan	1185	341.312.126	193.409.297	0,57	330	227.541.630	81.341.730	0,36	315
Patchwork	1268	365.298.434	220.170.383	0,60	291	243.532.546	99.690.870	0,41	273
TOTAL	**6423**	**1.850.963.232**	**998.576.928**	**0,54**	**1428**	**1.233.976.604**	**398.032.068**	**0,32**	**1330**

Coder PKZIP									
		24 bit				16 bit			
Sequence title	**Duration (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**	**Original (byte)**	**Coded (byte)**	**Ratio**	**Coding time (s)**
Barry_intertraccia	176	50.697.360	44.943.084	0,89	13	33.798.288	26.748.680	0,79	9

Caro_pianoforte	1183	340.722.212	302.640.952	0,89	97	227.148.324	174.762.959	0,77	73
Chopin_notturni	1477	425.657.034	367.823.076	0,86	127	283.771.594	209.248.208	0,74	80
Dire	396	114.250.944	99.390.097	0,87	33	76.167.358	56.409.039	0,74	21
I_dont_know_where_love_has_gone	300	86.655.168	81.691.287	0,94	24	57.770.174	50.687.048	0,88	16
I_soblue_and_you_are_too	438	126.369.954	114.496.084	0,91	36	84.246.690	68.601.199	0,81	23
Pape_satan	1185	341.312.126	308.024.138	0,90	98	227.541.630	183.333.711	0,81	62
Patchwork	1268	365.298.434	335.131.584	0,92	114	243.532.546	203.354.816	0,84	67
TOTAL	**6423**	**1.850.963.232**	**1.654.140.302**	**0,89**	**542**	**1.233.976.604**	**973.145.660**	**0,79**	**351**

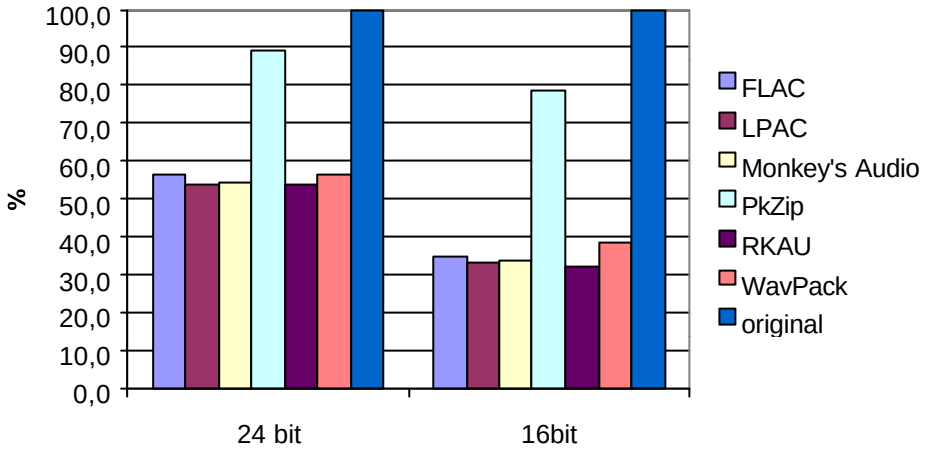

Figure 1. Comparison of the average compression ratios on the test sequences.

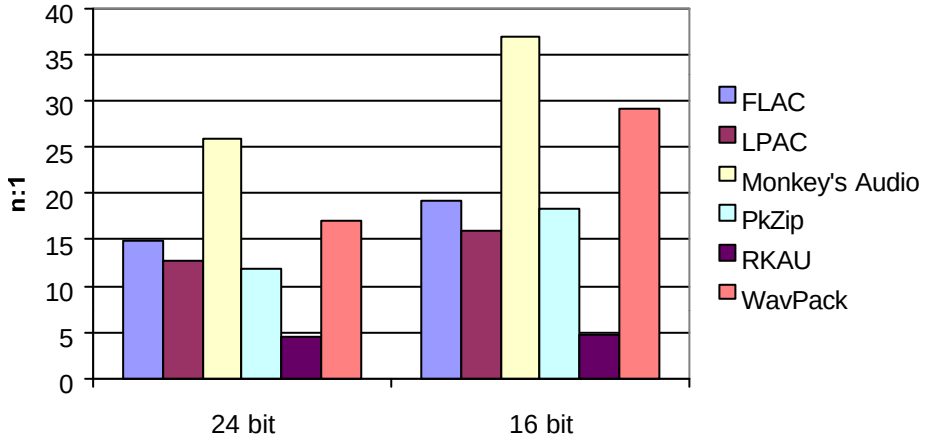

Figure 2. Comparison of encoding speed. The ordinate expresses how many times the encoding is faster than real time.

From the graphs above, the following considerations can be drawn:

- The encoding efficiency of the 5 audio codecs is comparable. The differences are within 3% at 24 bits and 7% at 16 bits.
- Audio specific algorithms perform significantly better than general purpose schemes (PKZIP)

- The average coding ratio of the audio codecs is close to 0.5 for 24 bit materials and close to 0.33 for 16 bit materials. This implies that it would be possible to half the storage requirements for 24 bit materials and to divide it by three for 16 bit materials, provided that the test sequences are representative of the average of the archive content. These figures will be verified during the field trials.
- The performance difference between 16 and 24 bit materials needs further explanation. Looking at the tables, it can be noted that the difference in size of the coded materials is almost the same as the difference of the originals. This means that there is very little correlation to be exploited in 8 least significant bits of the audio samples, that therefore contain mainly noise. This is understandable as the dynamic range of the source materials, coming from legacy analog media, is limited to 70 dB or less.
- All the codecs can encode both the tested sample sizes at much faster than real time. The performance spans from 4 to 26 times at 24 bit and from 5 to 37 times at 16 bit. This great variation between tools can depend both on software optimisation and on algorithm complexity. Decoding time has not been logged but is generally shorter than encoding time.

The results of the test suggest that lossless coding can be effectively applied to both 16 and 24 bit sound material, bringing a significant reduction in storage space. However, the figures obtained with this limited test need to be confirmed by more extensive field trials. All the tested algorithms show equivalent coding efficiency. Conversely, the coding speed of the various software can vary significantly. Among them, however, only FLAC is distributed freely under General Purpose Licence (GPL) in source form and has a built in provision for user metadata, useful for storing the information contained in the BWF header of the source material. Therefore, given that the inclusion of lossless coding in the PRESTO test bed is to be considered as a proof of concept of the approach, rather than a final recommendation of a specific tool for building real systems, FLAC has been selected.

4.4. FLAC description

An extensive description of FLAC can be found on the FLAC web site (http://flac.sourceforge.net). Here, the main features are summarised for the reader convenience.

FLAC stands for Free Lossless Audio Codec. The FLAC project consists of:

- the stream format
- **`libFLAC`**, which implements reference encoders and decoders
- **`flac`**, a command-line wrapper around libFLAC to encode and decode .flac files
- input plugins for various music players (Winamp, XMMS, and more in the works)

It can be compiled on many platforms: most Unixes (Linux, *BSD, Solaris, OS X), Windows, BeOS, and OS/2.

Architecture

Similar to many audio coders, a FLAC encoder has the following stages:

- **Blocking**. The input is broken up into many contiguous blocks. With FLAC, the blocks may vary in size. The optimal size of the block is usually affected by many factors,

including the sample rate, spectral characteristics over time, etc. Though FLAC allows the block size to vary within a stream, the reference encoder uses a fixed block size.

- **Interchannel Decorrelation**. In the case of stereo streams, the encoder will create mid and side signals based on the average and difference (respectively) of the left and right channels. The encoder will then pass the best form of the signal to the next stage.
- **Prediction**. The block is passed through a prediction stage where the encoder tries to find a mathematical description (usually an approximate one) of the signal. This description is typically much smaller than the raw signal itself. Since the methods of prediction are known to both the encoder and decoder, only the parameters of the predictor need be included in the compressed stream. FLAC currently uses four different classes of predictors (described in the prediction section), but the format has reserved space for additional methods. FLAC allows the class of predictor to change from block to block, or even within the channels of a block.
- **Residual coding**. If the predictor does not describe the signal exactly, the difference between the original signal and the predicted signal (called the error or residual signal) must be coded losslessy. If the predictor is effective, the residual signal will require fewer bits per sample than the original signal. FLAC currently uses only one method for encoding the residual, but the format has reserved space for additional methods. FLAC allows the residual coding method to change from block to block, or even within the channels of a block.

In addition, FLAC specifies a metadata system, which allows arbitrary information about the stream to be included at the beginning of the stream.

The basic structure of a FLAC stream is:

- The four byte string "fLaC"
- The STREAMINFO metadata block
- Zero or more other metadata blocks
- One or more audio frames

The first four bytes are to identify the FLAC stream. The metadata that follows contains all the information about the stream except for the audio data itself. After the metadata comes the encoded audio data.

METADATA

FLAC defines several types of metadata blocks. Metadata blocks can be any length and new ones can be defined. A decoder is allowed to skip any metadata types it does not understand. Only one is mandatory: the STREAMINFO block. This block has information like the sample rate, number of channels, etc., and data that can help the decoder manage its buffers, like the minimum and maximum data rate and minimum and maximum block size. Also included in the STREAMINFO block is the MD5 signature of the *unencoded* audio data. This is useful for checking an entire stream for transmission errors.

Other blocks allow for padding, seek tables, and application-specific data.

AUDIO DATA

After the metadata comes the encoded audio data. Audio data and metadata are not interleaved. Like most audio codecs, FLAC splits the unencoded audio data into blocks, and encodes each block separately. The encoded block is packed into a frame and appended to the stream.

FRAMING

An audio frame is preceded by a frame header and trailed by a frame footer. The header starts with a sync code, and contains the minimum information necessary for a decoder to play the stream, like sample rate, bits per sample, etc. It also contains the block or sample number and

an 8-bit CRC of the frame header. The sync code, frame header CRC, and block/sample number allow resynchronization and seeking even in the absence of seek points. The frame footer contains a 16-bit CRC of the entire encoded frame for error detection. If the reference decoder detects a CRC error it will generate a silent block.

The command-line file encoder/decoder is named **flac**. The input to the encoder and the output to the decoder must either be RIFF WAVE format, or raw interleaved sample data. **flac** only supports linear PCM samples. Another restriction is that the input must be 8, 16, or 24 bits per sample.

flac will be invoked one of four ways, depending on whether you are encoding, decoding, testing, or analyzing:
Encoding: `flac [-s] [--skip #] [-V] [<format-options>] [<encoding options>] [inputfile[...]]`
Decoding: `flac -d [-s] [--skip #] [<format-options>] [inputfile [...]]`
Testing: `flac -t [-s] [inputfile [...]]`
Analyzing: `flac -a [-s] [--skip #] [<analysis-options>] [inputfile [...]]`

In any case, if no input file is specified, *stdin* is assumed. If only one input file is specified, it may be "-" for *stdin*. When *stdin* is used as input, **flac** will write to *stdout*. Otherwise **flac** will perform the desired operation on each input file to similarly named output files (meaning for encoding, the extension will be replaced with ".flac", or appended with ".flac" if the input file has no extension, and for decoding, the extension will be ".wav" for WAVE output and ".raw" for raw output). The original file is not deleted unless *--delete-input-file* is specified in the options.
The encoding options affect the compression ratio and encoding speed. The format options are used to tell **flac** the arrangement of samples if the input file (or output file when decoding) is a raw file. If it is a RIFF WAVE file the format options are not needed since they are read from the WAVE header.
In test mode, **flac** acts just like in decode mode, except no output file is written. Both decode and test modes detect errors in the stream, but they also detect when the MD5 signature of the decoded audio does not match the stored MD5 signature, even when the bitstream is valid.
Refer to the on-line manual on the FLAC web site for the description of all the available options.

5. Software adaptation

As shown in Figure 3, user applications interact with the digital archive through BWF files. The lossless coder must transparently compress and decompress the sound files without interfering with the users operations.
To include FLAC in the PRESTO test bed, some adaptation needs to be done, in order to fulfil the requirements on BWF compatibility and partial access to files.

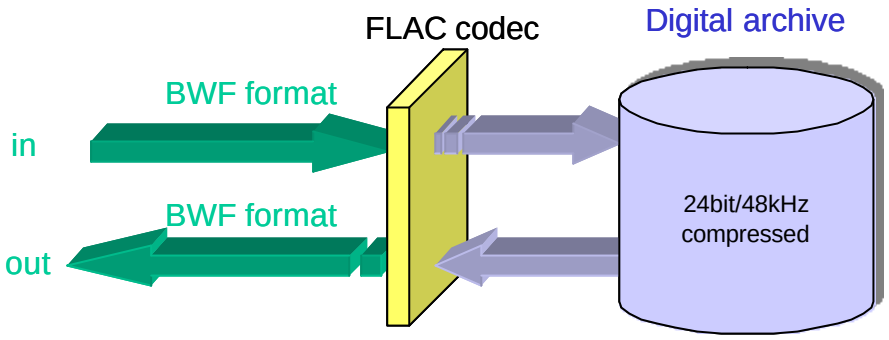

Figure 3. Lossless coding is a layer between the user applications and the storage.

5.1 From BWF format to compressed format

The most important task for the implementation of FLAC tools within the PRESTO process was the adaptation between the BWF (Broadcast Wave Format) file produced by the digitisation process and the FLAC input format requirement. The adaptation regards both the audio chunk and the other data chunks that BWF carries itself.
The BWF is a file format defined by the EBU, derived from the Microsoft RIFF-WAV format, as the standard format for the exchange of audio materials as file between the EBU members. RIFF-WAV is structured as a container encapsulating both the audio samples and several sets of associated metadata, some of which mandatory, other optional, organised in *chunks*, that are records composed of a chunk identifier, a length field and a variable length payload. The chunk identifier defines the type of data contained and therefore its formatting. The length allows to skip a chunk without knowing its internal structure. The BWF format uses the basic format and data chunks defined by the RIFF-WAV format, that contain respectively the format of the audio samples (sampling frequency, number of channels, etc ...) and the audio itself, and defines other chunks specific to broadcast applications. Figure 4 shows the structure of a basic BWF file.

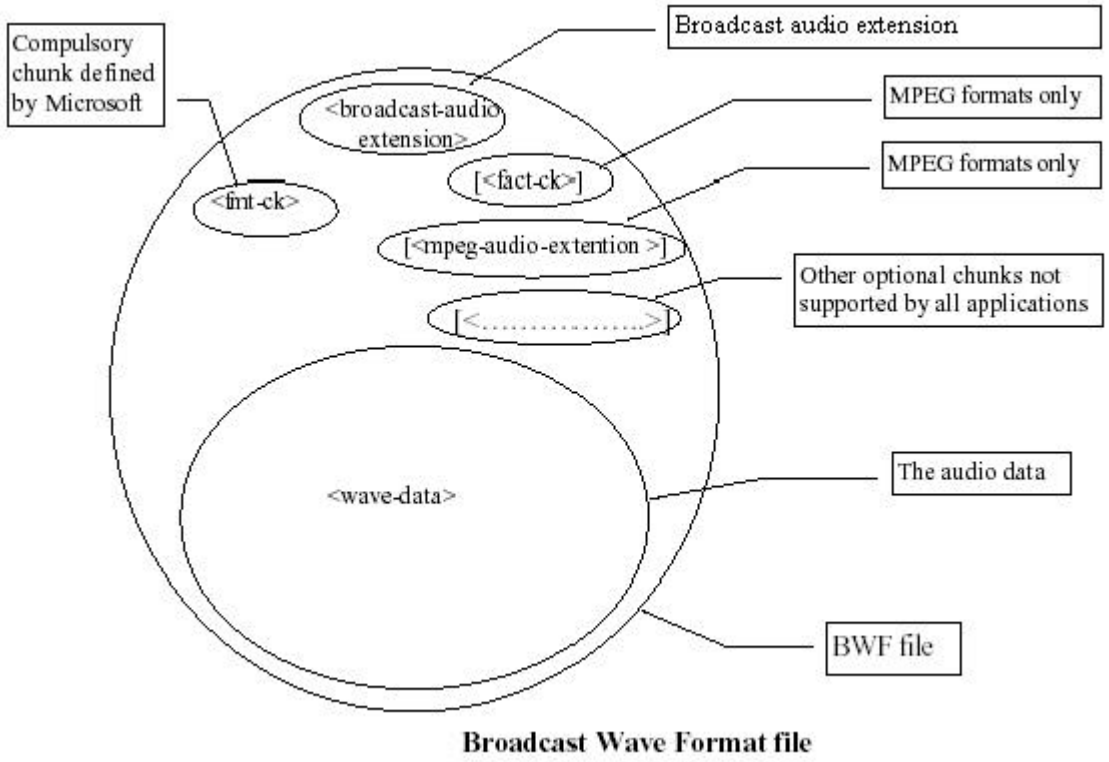

Figure 4. Basic structure of a BWF file.

When processing with a lossless compression tool (in our case flac) the BWF audio files produced by the transcription process, care must be taken that no information be lost, not only of audio nature, but also including the set of metadata encapsulated in the file. Examining Figure 4, we can derive that the relevant metadata chunks are the format chunk (frmt-ck) and the broadcast audio extension chunk. The former contains vital information for the playing of the audio and is therefore to be expected that any other audio file format has provision for storing them, while the latter contains user specific information that require proper handling. Two basic approaches could be followed:

1. Encapsulate the compressed audio processed with FLAC into the data chunck of BWF
2. Keep the compressed audio into the FLAC container and save the broadcast audio extension content into a user metadata area provided by the FLAC format

The first approach is certainly more consistent with the EBU policy than the other, but it is of no practical use today, as the only audio formats recognised by BWF players and processors are linear PCM and optionally MPEG. Therefore, this solution would lead to files unusable by any existing application. The second approach has the benefit of allowing the use of the existing FLAC decoder for the audio related operations, like decode, play and seek, while requires the development of a custom tool for the insertion and extraction of the broadcast extension metadata from the FLAC file format. This second approach has been adopted, as it has been considered functionally equivalent to the first with respect to the PRESTO test bed implementation but offers the advantage of a faster development.

The process of BWF files encoding to FLAC has been structured as follows:

- BWF chunk decomposition
- Audio data compression
- Broadcast extension chunk encapsulation

5.1.1 BWF chunk decomposition

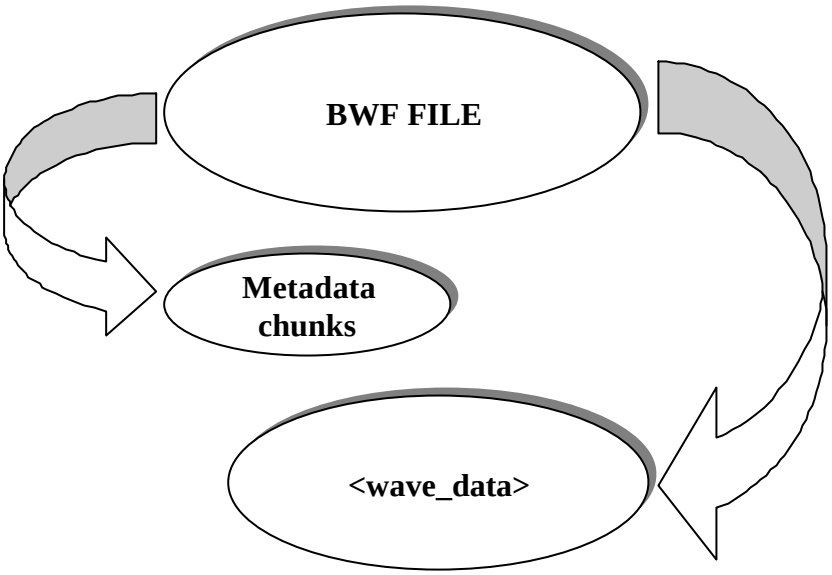

Figure 5. Decomposition of the BWF format.

The BWF file relevant chunks are the audio chunk (<wave-data>) containing the essence (audio samples), the format chunk and the broadcast extension chunk. The FLAC encoder recognises the BWF format as a RIFF-WAV and therefore neglects the broadcast extension chunk, the default behaviour in the presence of unknown chunks, that must therefore be extracted from the BWF file and temporarily stored on a separate file.

5.1.2 Audio data compression

Unfortunately, the current release of FLAC is buggy when parsing RIFF-WAV file containing audio digitised at 24 bit per sample (actually the Microsoft specification is unclear if 24 bit is an officially supported format or not). Therefore, we have to remove the RIFF-WAV encapsulation and encode the audio in raw format, putting the format metadata on the command line.
During this phase the <wave_data> chunk is compressed using the flac encoder with the command:

```
flac -fl -fc 2 -fp 24 -fs 48000 –S 1000x -P $size -o $bwf.flac $bwf.raw
```

Where:

- `$bwf.flac` is the output file;
- `$bwf.raw` is the input file;
- `$size` is the size necessary for the BWF metadata chunks encapsulation (this value is computed for every processed file). The -P directive instructs the encoder to reserve the given size of bytes for subsequent encapsulation of user data.

5.1.3 BWF metadata chunks encapsulation

The FLAC file format architecture supports up to 128 kinds of metadata blocks. Currently the following are defined:

- STREAMINFO: This block has information about the whole stream, like sample rate, number of channels, total number of samples, etc. It must be present as the first metadata block in the stream. Other metadata blocks may follow, and ones that the decoder doesn't understand, it will skip.
- APPLICATION: This block is for use by third-party applications. The only mandatory field is a 32-bit identifier. This ID is granted upon request to an application by the FLAC maintainers. The remainder of the block is defined by the registered application.
- PADDING: This block allows for an arbitrary amount of padding. The contents of a PADDING block have no meaning. This block is useful when it is known that an APPLICATION block will be added after encoding; the user can instruct the encoder to reserve a PADDING block of the proper size so that the application may directly write over it later instead of having to insert the APPLICATION block (which would normally require rewriting the entire file).
- SEEKTABLE: This is an optional block for storing seek points. It is possible to seek to any given sample in a FLAC stream without a seek table, but the delay can be unpredictable since the bit rate may vary widely within a stream. By adding seek points to a stream, this delay can be significantly reduced. Each seek point takes 18 bytes, so 1% resolution within a stream adds less than 2k. There can be only one seek table in a stream, but the table can have any number of seek points. There is also a special 'placeholder' seek point which will be ignored by decoders but which can be used to reserve space for future seek point insertion.

During the compression phase, an empty space has been allocated into a padding block, fitting exactly the broadcast extension chunk dimension. After the compression the padding block is overwritten with the BWF chunk.
This approach allows the uncompressed storage of any metadata chunk and is therefore not limited to the broadcast extension. The extraction can be done in an extremely fast way (this operation doesn’t require decompression) using the ***metaflac*** tool included in the FLAC distribution.

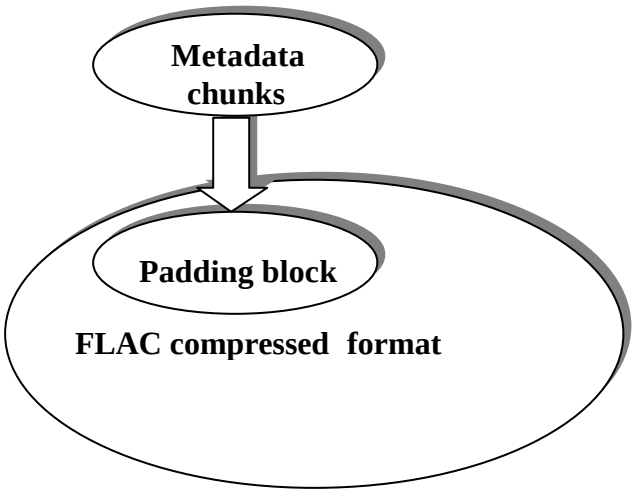

Figure 6. Metadata encapsulation into the FLAC format.

5.2 Partial File Access

An intrinsic characteristic of lossless compression technology is that of generating a stream with a variable bit rate, as compression efficiency depends on the instantaneous characteristic of the input signal. In other words, there is no simple relation between the time duration of an audio clip and the corresponding storage space. This characteristic doesn't allow an easy positioning inside the compressed stream without a partial stream decompression. Furthermore, as the codecs remove the temporal correlation of the audio signal by means of time prediction, it is not possible to start the decoding from an arbitrary point in the stream, but only at block boundaries, where the time prediction chain is broken (see section 4.4). The FLAC compressor supplies natively a method that allows a discrete (non continuos) positioning into the compressed file. This feature is obtained by creating, during the compression phase, a seek table (see paragraph 5.1.3) containing a user-defined number of seek points, that are associations between play time position and file displacement in bytes (see Figure 7), evenly distributed along the whole stream. The built-in number of points used by FLAC is 100 (giving a 1% seek precision into the stream at the cost of 18 byte for each seek point). It is also possible to insert manually user-defined seek points giving the time position of the seekable sample.

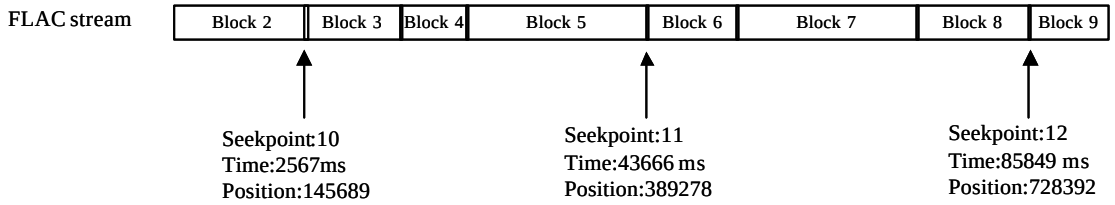

Figure 7. FLAC seek points example.

Therefore, using the seek table, the extraction of a precise portion of signal from the compressed file is possible, reducing both the extraction time (alias CPU time) and the disk space necessary (complete stream decompression could require a larger amount of space). A full exploitation of this feature requires however the use of low level API, accessible via the library **libflac**, as the distributed **flac** decoder provides only a partial implementation, sufficient anyway for the evaluation purposes of the PRESTO test bed.