Journal of Engineering and Applied Sciences 12 (20): 5301-5306, 2017
ISSN: 1816-949X

Design and Implementation of an Audio Parser and Player

T. Anu Radha, K. Aruna Manjusha, R. Karthik, Mahendra Vucha and A.L. Siridhara
Department of Electronics and Communication Engineering,
MLR Institute of Technology, Hyderabad, India

Abstract: Audio parsers are applications that scan through a particular file and extract information from them. Parsers are mainly used in design applications to test encoders and also in design of codecs. Players are needed to play the files. Very few parsers are available that can parse as well as play audio files. Also, most players available do not display parsed information. So, a well-designed audio parser and player is needed. Our project is to design a gui application that can parse and play audio files. The codecs supported are MP3 and AAC. In AAC, the ADIF and ADTS formats are supported. ID3v2 tags too are parsed and displayed. An audio player in integrated to play the files from the application itself. Also, features are added to the player to enhance the functionality. Play, pause, stop and seek features are supported. The application can be easily extended to other codecs.

Key words: Audio parser, codec, VC++, MPEG, ID3, seek features

INTRODUCTION

Parsers are programs that read (scan) through a file and extracts information from it. Audio parsers are needed to extract information such as the format used to code the file, the bit rate, the sampling frequency, etc. In design applications such as testing a codec, it is necessary to test the performance of the codec against corrupt frames. An audio parser helps to identify such corrupt frames. In designing new encoders, the encoded file can be parsed to check whether it is in the actual format needed. Parsers also are used as a quick reference for finding information from audio files. Audio players are needed to play audio files.

As of today there are many audio playing and parsing applications available. One of the well-known audio parser applications is Microsoft® Corporation's "Windows Media ASF View 9 series" which parses file of type .mp3, .asf, .wma and .wmv. However, this application like other parsing applications does not have a built in player to play the parsed files. On the audio players side the most popular ones are Winamp, Windows Media® Player, Foobar, Real® Player, etc. However, these players are suited for end user applications and not for design applications. The reason is that most of these players do not display the parsed audio information. For design applications, we have audio tools such as Goldwave or cool edit pro, etc. But these give audio information and modification but not frame information (Schildt, 2001; Balaguruswamy, 2001; British Standards Institutions, 1997). So, we require a well-built audio parser and player to do all the above mentioned tasks.

The MPEG Layer III codec or the MP3 codec is one of the most popular digital audio encoding formats (Raissi, 2002). It is the most well-known lossy compression format. It was invented and standardized in 1991 by a team of engineers working in the framework of the ISO (International Organization for Standardization)/IEC (International Electrotechnical Commission) MPEG (Motion Pictures Expert Group) audio committee under the chairmanship of Professor Hans Musmann (University of Hannover-Germany).

The MP3 encoder compresses an original PCM audio file by a factor of 12 without any noticeable quality loss. The MP3 format enables variable bit rate encoding. It also allows different sampling frequency and various channel modes.

The MP3 format uses at its heart, a hybrid transformation to transform a time domain signal into a frequency domain signal:

- 32-band polyphase quadrature filter
- 36 or 12 tap MDCT; size can be selected independent for sub-band 0...1 and 2...31
- Aliasing reduction postprocessing

It also employs several techniques such as:

- Huffman coding
- MDCT/modified discrete cosine transform
- Non linear quantization with scaling
- Intensity stereo, part of joint stereo
- M/S matrixing, part of joint stereo
- Lattice quantization for high frequencies

Corresponding Author: R. Karthik, Department of Electronics and Communication Engineering, MLR Institute of Technology, Hyderabad, India

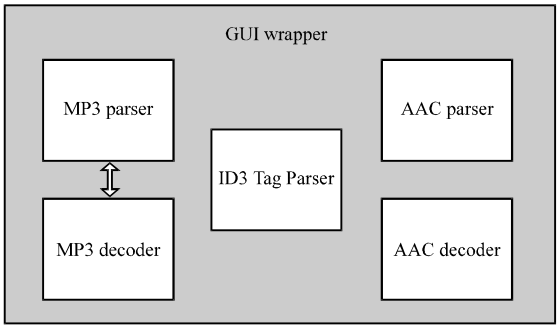

Fig. 1: Overall functional diagram

Block diagram: The overall functional diagram of the parser and player is as shown in Fig. 1. The GUI wrapper along with the glue logic integrates the parsers and decoder as a single working entity (Fig. 1).

Approach to problem solving: In development of this application, we use Object Oriented Programming (OOP) approach. Object oriented programming has the following striking features which make it a good choice for software development.

Data abstraction and encapsulation: The wrapping up of data and functions into a single unit is known as encapsulation. This feature makes each module as a separate entity which cannot be corrupted from outside. Thus each module can be developed and tested individually and when integrated with rest of the modules, it is assured to work correctly. This principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.

Inheritance: Inheritance is the process by which objects of one class acquire the properties of another class. It supports the concept of hierarchical classification. This feature helps in easily extending the features of an already designed module to incorporate minor changes or extensions. Thus everything need not be designed from scratch again. This helps in eliminating redundant code and extends the use of existing classes.

Partitioning: It is easy to partition the reseach in a project based on objects. Software complexity can be easily managed. Hence, development of the application can be done part by part and then integrated without posing any problems.

Choice of programming language: The programming language that we have chosen is C++. C++ has a number of features which make it an appropriate choice. C++ is a versatile language for handling very large programs. It is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real-life application systems. C++ supports object oriented programming which suits our need. Since, the syntax and semantics of C++ program is very similar to C, it makes learning this language quite easier. Moreover, C++ provides constructs to access and modify bit-fields. C++ also provides features that help in integrating audio decoders and players (exe files) to the existing application to extend its capability. Thus, C++ proves to be an optimal choice.

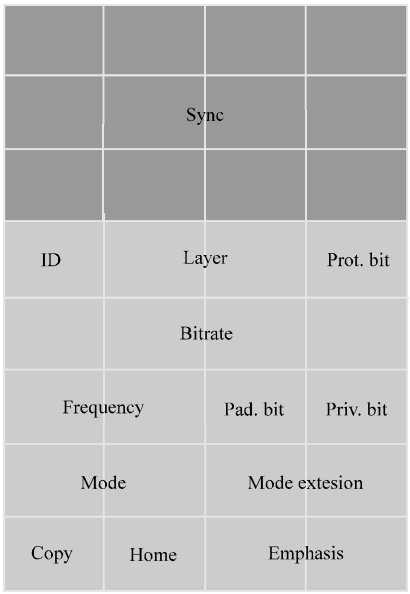

Fig. 2: MP3 header layout

MATERIALS AND METHODS

Design of MP3 parser and ID3 tag parser

MP3 frame layout: An MP3 file is made of smaller fragments called frames. Each frame stores 1152 audio samples and lasts for 26 msec. This means that the frame rate will be around 38 fps. In addition a frame is subdivided into two granules each containing 576 samples. Since, the bitrate determines the size of each sample, increasing the bitrate will also increase the size of the frame. The size is also depending on the sampling frequency according to Eq. 1:

$$\frac{144\times\text{bitrate}}{\text{Sampling-frequency}}+\text{padding[bytes]} \quad (1)$$

The frame layout is as shown; header, CRC, side information, main data, ancillary data.

Table 1: MP3 frame bitrate

Bits	MPEG-1 layer I	MPEG-1 layer II	MPEG-1 layer III	MPEG-2 layer I	MPEG-2 layer II	MPEG-2 layer III
0000	-	-	-	-	-	-
0001	32	32	32	32	32	8
0010	64	48	40	64	48	16
0011	96	56	48	96	56	24
0100	128	64	56	128	64	32
0101	160	80	64	160	80	64
0110	192	96	80	192	96	80
0111	224	112	96	224	112	56
1000	256	128	112	256	128	64
1001	288	160	128	288	160	128
1010	320	192	160	320	192	160
1011	352	224	192	352	224	112
1100	384	256	224	384	256	128
1101	416	320	256	416	320	256
1110	448	384	320	448	384	320
1111	448	384	320	448	384	320

Frame header: The frame header is 32 bits long and contains a synchronization word together with a description of the frame. The synchronization word found in the beginning of each frame enables MP3 receivers to lock onto the signal at any point in the stream. The layout of the frame header is as shown in Fig. 2. It consists of the following fields.

Sync (12 bits): This is the synchronization word described. All 12 bits must be set, i.e., "1111 1111 1111".

ID (1 bit): Specifies the MPEG version. A set bit means that the frame is encoded with the MPEG-1 standard if not MPEG-2 is used.

Layer (2 bits): This field tells what layer is being used. Here 00 is reserved, 11 is used for layer 1, 10 for layer 2 and 01 for layer 3.

Protection bit (1 bit): If the protection bit is set, the CRC field will be used.

Bitrate (4 bits): These four bits tells the decoder in what bitrate the frame is encoded. This value will be the same for all frames if the stream is encoded using CBR. Table 1 show the bitrate corresponding to each layer.

Frequency (2 bits): 2 bits that give the sampling frequency. Table 2 show the frequency values used for different versions. Further fields are as follows.

Padding bit (1 bit): An encoded stream with bit rate 128 kbit/s and sampling frequency of 44100 Hz will create frames of size 417 bytes. To exactly fit the bitrate some of these frames will have to be 418 bytes. These frames set the padding bit.

Private bit (1 bit): One bit for application-specific triggers.

Mode extension (2 bits): These 2 bits are only usable in joint stereo mode and they specify which methods to use. The joint stereo mode can be changed from one frame to another or even switched on or off. Table 2 shows how to interpret the bits.

Table 2: MP3-sampling frequency index

Bits	MPEG 1	MPEG 2	MPEG 2.5
00	44100 Hz	22050 Hz	11025 Hz
01	48000 Hz	24000 Hz	12000 Hz
10	32000 Hz	16000 Hz	8000 Hz
11	Reserv.	Reserv.	Reserv.

Copyright bit (1 bit): If this bit is set it means that it is illegal to copy the contents.

Home (original bit) (1 bit): The original bit indicates if it is set that the frame is located on its original media.

Emphasis (2 bits): The emphasis indication is used to tell the decoder that the file must be de-emphasized, i.e., the decoder must "re-equalize" the sound after a dolby-like noise suppression. It is rarely used.

Design of the MP3 parser: The parsing of an MP3 file consists of the following steps.

Sync word detection: Two consecutive characters are read from the MP3 file from the beginning of the file. Using bit masks "ff" (in hex) on the first character and "f0" (in hex) on the second character, sync word detection for 12 continuous 1's is carried out. Occurrence of 121's indicates the beginning of an MP3 frame. The position of the first frame in the file is recorded. It is used in parsing ID3 tags (if any, present) explained later (Nillson, 2000).

Parsing the header: The size of the MP3 frame header including the sync-word is 4 bytes (4 characters). Since, first 2 bytes have already been read during sync word detection, the next 2 bytes are read and assigned to 2 characters. Using bit-masks different fields of the MP3 frame header are extracted and are stored in a structure "mp3_frame_hdr_content". This information

Table 3: MP3 mode extension

Mode extentions	Values
Stereo	00
Joint stereo	01
Dual channel	10
Single channel	11

also contains the sampling frequency index and bitrate index. Table 1-3 are stored in structures named "bitrate_calc" and "mp3_freq_calc". Using these structures as lookup tables the bitrate and the sampling frequency used is found out.

RESULTS AND DISCUSSION

Extracting the parameters: Having obtained information like sampling frequency, bitrate from the file. The size of the MP3 frame is found using Eq. 1. We repeat the equation here for convenience:

$$\frac{144\times\text{bitrate}}{\text{Sampling-frequency}}+\text{padding[bytes]} \quad (1)$$

Additional information like the duration of the MP3 frame is also found out. It can be found out using the Eq. 2:

$$\text{Duration} = (1152/\text{sampling\_frequency})\ (\text{sec}) \quad (2)$$

Writing parsed information: The parsed header information is written to an intermediate file "frame_output.txt". Another file "inter.txt" records the beginning position of each frame in the "file frame_output.txt" and frame length of each frame. This file acts as a pointer to "frame_output.txt". It is used in displaying parsed information.

Writing data: Using the MP3 frame size computed (which includes the header) mp3 data is read byte by byte and written on to the file "frame_output.txt". After this step, we go back to step 1 and again sync word detection is carried out and the next four steps are repeated till end of file is reached.

ID3v2 tag format: The ID3 tag is laid out as:

- Header (10 bytes)
- Extended header (variable length, optional)
- Frames (variable length)
- Padding (variable length, optional)
- Footer (10 bytes, optional)

Tag header: The first part of the ID3v2 tag is the 10 byte tag header. The layout of the ID3 header is as:

- ID3 (3)
- VER (2)
- Flags (1)
- Size of the tag (4)

There are four parts in the header and no of bytes occupied by each part is mentioned in the braces. The ID3 field is the first part which occupies three bytes holding the characters "ID3". This is followed by two bytes VER field that indicate the version number. The version is followed by the ID3v2 flags field of the 8 bits allocated only 4 bits are used. So, this has the form abcd0000.

Flag a (7th bit), unsynchronisation: This indicates whether or not unsynchronisation (explained later) is applied on all frames a set bit indicates usage.

Flag b (6th bit), extended header: This indicates whether or not the header is followed by an extended header. A set bit indicates the presence of an extended header.

Flag c (5th bit), experimental indicator: This flag will always be set when the tag is in an experimental stage.

Flag d (4th bit), footer present: This flag indicates that a footer is present at the very end of the tag. A set bit indicates the presence of a footer.

The next four bytes tell the size of the ID3 tag including the extended header, the frames and padding. It is stored as a 32 bit synch safe integer. Sync safe integers are integers that keep its highest bit (bit 7) zeroed, making seven bits out of eight available. Thus, a 32 bit sync safe integer can store 28 bits of information. For example, 255 $(11111111)_{(2)}$ encoded as a 16 bit sync safe integer is 383 $(00000001\ 01111111)_{(2)}$.

Padding: It is optional to include padding after the final frame (at the end of the ID3 tag), making the size of all the frames together smaller than the size given in the tag header. A possible purpose of this padding is to allow for adding a few additional frames or enlarge existing frames within the tag without having to rewrite the entire file. The value of the padding bytes must be 0×00. A tag must not have any padding between the frames or between the tag header and the frames. Also, the footer and padding are mutually exclusive.

Frames: The frames are laid out as follows:

- Frame id: 0x XX XX XX XX
- Frame size: $4\times(0XXXXXXX)_{(2)}$
- Flags 0x XX XX
- Data 0x . . . . .

The first four bytes together form the frame id which is made out of the characters capital A-Z and 0-9. Identifiers beginning with "X", "Y" and "Z" are for experimental frames and free for everyone to use. The frame ID is followed by a size descriptor containing the size of the data in the final frame. The size is excluding the frame header which is nothing but "total frame size" 10 bytes and stored as a 32 bit sync safe integer. A tag must contain at least one frame. A frame must be at least 1 byte big, excluding the header. The frame status flags are in the form $(0abc0000\ 0h00kmnp)_{(2)}$. The status flags are explained.

Tag alter preservation: This flag tells the tag parser what to do with this frame if it is unknown and the tag is altered in any way. This applies to all kinds of alterations including adding more padding and reordering the frames:

- 0 frame should be preserved
- 1 frame should be discarded

File alter preservation: This flag tells the tag parser what to do with this frame if it is unknown and the file, excluding the tag is altered. This does not apply when the audio is completely replaced with other audio data:

- 0 frame should be preserved
- 1 frame should be discarded

Read only: This flag if set, tells the parser that the contents of this frame are intended to be read only. Changing the contents might break something, e.g., a signature. If the contents are changed without knowledge of why the frame was flagged read only and without taking the proper means to compensate, e.g., recalculating the signature, the bit must be cleared.

Grouping identity: This flag indicates whether or not this frame belongs in a group with other frames. If set a group identifier byte is added to the frame. Every frame with the same group identifier belongs to the same group:

- 0 frame does not contain group information
- 1 frame contains group information

Compression: This flag indicates whether or not the frame is compressed. A "data length indicator" byte must be included in the frame:

- 0 frame is not compressed
- 1 frame is compressed using zlib (zlib) deflate method

If set, this requires the "data length indicator" bit to be set as well.

Encryption: This flag indicates whether or not the frame is encrypted. If set, one byte indicating with which method it was encrypted will be added to the frame. The description of the ENCR frame for more information about encryption method registration. Encryption should be done after compression. Whether or not setting this flag requires the presence of a "data length indicator" depends on the specific algorithm used:

- 0 frame is not encrypted
- 1 frame is encrypted

Unsynchronisation: This flag indicates whether or not unsynchronisation was applied to this frame. Section 6 for details on unsynchronisation. If this flag is set all data from the end of this header to the end of this frame has been unsynchronised:

- 0 frame has not been unsynchronised
- 1 frame has been unsynchronised

Data length indicator: This flag indicates that a data length indicator has been added to the frame:

- 0 there is no data length indicator
- 1 a data length indicator has been added to the frame

The flags field is followed by the field data. The field data may itself contain various fields depending on the type. For, e.g., consider the terms of use frame (Frame Id: user). In this the frame header is followed by a byte which is the text encoding frame.

The application has been coded and tested for various files. The parsed MP3 files have been verified using the Windows ASF viewer application. Using the Free AAC encoder (FAAC-main project of which FAAD is a part) (Sourceforge, 2013) audio samples were encoded with specific bit rates and sampling frequencies in both adif and adts formats. These files were tested on the player and were found to be correct. The features of the player have also been tested. The player can pause or play an MP3 or aac file. It can also seek through the file and play from any specified point. The timing display of the file is also tested for various files and is found to be correct.

CONCLUSION

The application has been coded and tested for various files. The features of the player have also been tested. The player can pause or play an MP3 or aac file. It can also seek through the file and play from any specified point. The timing display of the file is also tested for various files and is found to be correct.

RECOMMENDATIONS

The future research that can be carried out are additions of new parsers; other codec parsers can be added to the code. Using the existing frame index box, header and data text box we can other parsers to frame based codecs such as WMA, OGG-Vorbis, FLAC, etc. Integration of other players; other audio players can be integrated with the application. Additions of features; more features can be added to the code such as fast forwarding (playing at double or multiple rate), volume control, etc., can be added using already parsed data to play the file instead of integrating another player we can design a player that makes use of the data parsed by this application to decode and play the file.

ACKNOWLEDGEMENT

Researchers would like to thank Mr Sanjay Bhat for his support in carrying out the project.

REFERENCES

Balaguruswamy, E., 2001. Object Oriented Programming with C++. 2nd Edn., McGraw-Hill Education, New York, USA., ISBN:9780070402119, Pages: 533.

British Standards Institution, 1997. Information Technology Generic Coding of Moving Pictures and Associated Audio Information: Advanced Audio Coding (AAC). British Standards Institution, London, UK., ISBN:9780580280849, Pages: 147.

Nillson, M., 2000. ID3 tag version 2.4.0-structures, native frames. D3v2, Linkoping,Sweden. http://id3.org/

Raissi, R., 2002. The theory behind mp3. Mp3-Tech.Org, USA. http://www.mp3-tech.orgprogrammerdocs/mp3_theory.pdf.

Schildt, H., 2001. Java 2: The Complete Reference. 4th Edn., McGraw-Hill Education, New York, USA., ISBN:9780072130843, Pages: 1077.

Sourceforge, 2013. Freeware advanced audio coder. Sourceforge, Fairfax, Virginia. https://sourceforge.net/projects/faac/.