cellar
Internet-Draft
Intended status: Informational
Expires: 2 May 2022

M. Richardson

A. Weaver

29 October 2021

Free Lossless Audio Codec
draft-ietf-cellar-flac-02

Abstract

This document defines FLAC, which stands for Free Lossless Audio Codec, a free, open source codec for lossless audio compression and decompression.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 May 2022.

Copyright Notice

Table of Contents

1. Introduction

This is a detailed description of the FLAC format. There is also a companion document that describes FLAC-to-Ogg mapping (https://xiph.org/flac/ogg_mapping.html).

For a user-oriented overview, see About the FLAC Format (https://xiph.org/flac/documentation_format_overview.html).

2. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Acknowledgments

FLAC owes much to the many people who have advanced the audio compression field so freely. For instance: - A. J. Robinson (http://svr-www.eng.cam.ac.uk/~ajr/) for his work on Shorten (http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html); his paper is a good starting point on some of the basic methods used by FLAC. FLAC trivially extends and improves the fixed predictors, LPC coefficient quantization, and Exponential-Golomb coding used in Shorten. - S. W. Golomb (https://web.archive.org/web/20040215005354/http://csi.usc.edu/faculty/golomb.html) and Robert F. Rice; their universal codes are used by FLAC's entropy coder. - N. Levinson and J. Durbin; the reference encoder uses an algorithm developed and refined by them for

determining the LPC coefficients from the autocorrelation coefficients. - And of course, Claude Shannon (http://en.wikipedia.org/wiki/Claude_Shannon)

4. Scope

FLAC stands for Free Lossless Audio Codec: it is designed to reduce the amount of computer storage space needed to store digital audio signals without needing to remove information in doing so (i.e. lossless). FLAC is free in the sense that its specification is open, its reference implementation is open-source and it is not encumbered by any known patent.

FLAC is able to achieve lossless compression because samples in audio signals tend to be highly correlated with their close neighbors. In contrast with general purpose compressors, which often use dictionaries, do run-length coding or exploit long-term repetition, FLAC removes redundancy solely in the very short term, looking back at most 32 samples.

The FLAC format is suited for pulse-code modulated (PCM) audio with 1 to 8 channels, sample rates from 1 to 1048576 Hertz and bit depths between 4 and 32 bits. Most tools for reading and writing the FLAC format have been optimized for CD-audio, which is PCM audio with 2 channels, a sample rate of 44.1 kHz and a bit depth of 16 bits.

Compared to other lossless (audio) coding formats, FLAC is a format with low complexity and can be coded to and from with little computing resources. Decoding of FLAC has seen many independent implementations on many different platforms, and both encoding and decoding can be implemented without needing floating-point arithmetic.

The coding methods provided by the FLAC format works best on PCM audio signals of which the samples have a signed representation and are centered around zero. Audio signals in which samples have an unsigned representation must be transformed to a signed representation as described in this document in order to achieve reasonable compression. The FLAC format is not suited to compress audio that is not PCM. Pulse-density modulated audio, e.g. DSD, cannot be compressed by FLAC.

5. Architecture

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

* Blocking (see section on Blocking (#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 (see section on Interchannel Decorrelation (#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 (see section on Prediction (#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, 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 (See section on Residual Coding (#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 losslessly. 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.

6. Definitions

* *Block*: A (short) section of linear pulse-code modulated audio, with one or more channels.

* *Subblock*: All samples within a corresponding block for 1 channel. One or more subblocks form a block, and all subblocks in a certain block contain the same number of samples.

* *Frame*: A frame header plus one or more subframes. It encodes the contents of a corresponding block.

* *Subframe*: An encoded subblock. All subframes within a frame code for the same number of samples. A subframe MAY correspond to a subblock, else it corresponds to either the addition or subtraction of two subblocks, see section on interchannel decorrelation (#interchannel-decorrelation).

* *Blocksize*: The total number of samples contained in a block or coded in a frame, divided by the number of channels. In other words, the number of samples in any subblock of a block, or any subframe of a frame. This is also called *interchannel samples*.

* *Bit depth* or *bits per sample*: the number of bits used to contain each sample. This MUST be the same for all subblocks in a block but MAY be different for different subframes in a frame because of interchannel decorrelation (#interchannel-decorrelation).

* *Predictor*: a model used to predict samples in an audio signal based on past samples. FLAC uses such predictors to remove redundancy in a signal in order to be able to compress it.

* *Linear predictor*: a predictor using linear prediction (https://en.wikipedia.org/wiki/Linear_prediction). This is also called *linear predictive coding (LPC)*. With a linear predictor each prediction is a linear combination of past samples, hence the name. A linear predictor has a causal discrete-time finite impulse response (https://en.wikipedia.org/wiki/Finite_impulse_response).

* *Fixed predictor*: a linear predictor in which the model parameters are the same across all FLAC files, and thus not need to be stored.

* *Predictor order*: the number of past samples that a predictor uses. For example, a 4th order predictor uses the 4 samples directly preceding a certain sample to predict it. In FLAC, samples used in a predictor are always consecutive, and are always the samples directly before the sample that is being predicted

* *Residual*: The audio signal that remains after a predictor has been subtracted from a subblock. If the predictor has been able to remove redundancy from the signal, the samples of the remaining signal (the *residual samples*) will have, on average, a smaller numerical value than the original signal.

* *Rice code*: A variable-length code (https://en.wikipedia.org/wiki/Variable-length_code) which compresses data by making use of the observation that, after using an effective predictor, most residual samples are closer to zero than the original samples, while still allowing for a small part of the samples to be much larger.

7. Blocking

The size used for blocking the audio data has a direct effect on the compression ratio. If the block size is too small, the resulting large number of frames mean that excess bits will be wasted on frame headers. If the block size is too large, the characteristics of the signal MAY vary so much that the encoder will be unable to find a good predictor. In order to simplify encoder/decoder design, FLAC imposes a minimum block size of 16 samples, and a maximum block size of 65535 samples. This range covers the optimal size for all of the audio data FLAC supports.

Currently the reference encoder uses a fixed block size, optimized on the sample rate of the input. Future versions MAY vary the block size depending on the characteristics of the signal.

Blocked data is passed to the predictor stage one subblock (channel) at a time. Each subblock is independently coded into a subframe, and the subframes are concatenated into a frame. Because each channel is coded separately, one channel of a stereo frame MAY be encoded as a constant subframe, and the other an LPC subframe.

8. Interchannel Decorrelation

In many audio files, channels are correlated. The FLAC format can exploit this correlation in stereo files by not directly coding subblocks into subframes, but instead coding an average of all samples in both subblocks (a mid channel) or the difference between all samples in both subblocks (a side channel). The following combinations are possible:

* *Independent*. All channels are coded independently. All non-stereo files MUST be encoded this way.

* *Mid-side*. A left and right subblock are converted to mid and side subframes. To calculate a sample for a mid subframe, the corresponding left and right samples are summed and the result is shifted right by 1 bit. To calculate a sample for a side subframe, the corresponding right sample is subtracted from the corresponding left sample. On decoding, the mid channel has to be shifted left by 1 bit. Also, if the side channel is uneven, 1 has to be added to the mid channel after the left shift. To reconstruct the left channel, the corresponding samples in the mid and side subframes are added and the result shifted right by 1 bit, while for the right channel the side channel has to be subtracted from the mid channel and the result shifted right by 1 bit.

* *Left-side*. The left subblock is coded and the left and right subblock are used to code a side subframe. The side subframe is constructed in the same way as for mid-side. To decode, the right subblock is restored by subtracting the samples in the side subframe from the corresponding samples the left subframe.

* *Right-side*. The right subblock is coded and the left and right subblock are used to code a side subframe. Note that the actual coded subframe order is side-right. The side subframe is constructed in the same way as for mid-side. To decode, the left subblock is restored by adding the samples in the side subframe to the corresponding samples in the left subframe.

The side channel needs one extra bit of bit depth as the subtraction can produce sample values twice as large as the maximum possible in any given bit depth. The mid channel in mid-side stereo does not need one extra bit, as it is shifted left one bit. The left shift of the mid channel does not lead to non-lossless behavior, because an uneven sample in the mid subframe must always be accompanied by a corresponding uneven sample in the side subframe, which means the lost least significant bit can be restored by taking it from the sample in the side subframe.

9. Prediction

FLAC uses four methods for modeling the input signal:

1. *Verbatim*. This is essentially a zero-order predictor of the signal. The predicted signal is zero, meaning the residual is the signal itself, and the compression is zero. This is the baseline against which the other predictors are measured. If you feed random data to the encoder, the verbatim predictor will probably be used for every subblock. Since the raw signal is not actually passed through the residual coding stage (it is added to the stream 'verbatim'), the encoding results will not be the same as a zero-order linear predictor.

2. *Constant*. This predictor is used whenever the subblock is pure DC ("digital silence"), i.e. a constant value throughout. The signal is run-length encoded and added to the stream.

3. *Fixed linear predictor*. FLAC uses a class of computationally-efficient fixed linear predictors (for a good description, see audiopak (http://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf) and shorten (http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html)). FLAC adds a fourth-order predictor to the zero-to-third-order predictors used by Shorten. Since the predictors are fixed, the predictor order is the only parameter that needs to be stored in the compressed stream. The error signal is then passed to the residual coder.

4. *FIR Linear prediction*. For more accurate modeling (at a cost of slower encoding), FLAC supports up to 32nd order FIR linear prediction (again, for information on linear prediction, see audiopak (http://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf) and shorten (http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html)). The reference encoder uses the Levinson-Durbin method for calculating the LPC coefficients from the autocorrelation coefficients, and the coefficients are quantized before computing the residual. Whereas encoders such as Shorten used a fixed quantization for the entire input, FLAC allows the quantized coefficient precision to vary from subframe to subframe. The FLAC reference encoder estimates the optimal precision to use based on the block size and dynamic range of the original signal.

10. Residual Coding

FLAC uses Exponential-Golomb (a variant of Rice) coding as its residual encoder. You can learn more about exp-golomb coding (https://en.wikipedia.org/wiki/Exponential-Golomb_coding) on Wikipedia.

FLAC currently defines two similar methods for the coding of the error signal from the prediction stage. The error signal is coded using Exponential-Golomb codes in one of two ways:

1. the encoder estimates a single exp-golomb parameter based on the variance of the residual and exp-golomb codes the entire residual using this parameter;

2. the residual is partitioned into several equal-length regions of contiguous samples, and each region is coded with its own exp-golomb parameter based on the region's mean.

(Note that the first method is a special case of the second method with one partition, except the exp-golomb parameter is based on the residual variance instead of the mean.)

The FLAC format has reserved space for other coding methods. Some possibilities for volunteers would be to explore better context-modeling of the exp-golomb parameter, or Huffman coding. See LOCO-I (http://www.hpl.hp.com/techreports/98/HPL-98-193.html) and pucrunch (http://web.archive.org/web/20140827133312/http://www.cs.tut.fi/~albert/Dev/pucrunch/packing.html) for descriptions of several universal codes.

11. Format

This section specifies the FLAC bitstream format.

11.1. Principles

FLAC has no format version information, but it does contain reserved space in several places. Future versions of the format MAY use this reserved space safely without breaking the format of older streams. Older decoders MAY choose to abort decoding or skip data encoded with newer methods. Apart from reserved patterns, in places the format specifies invalid patterns, meaning that the patterns MAY never appear in any valid bitstream, in any prior, present, or future versions of the format. These invalid patterns are usually used to make the synchronization mechanism more robust.

All numbers used in a FLAC bitstream MUST be integers; there are no floating-point representations. All numbers MUST be big-endian coded, except the length field used in Vorbis comments, which MUST be little-endian coded. All numbers MUST be unsigned except linear predictor coefficients, the linear prediction shift and numbers which directly represent samples, which MUST be signed. None of these restrictions apply to application metadata blocks.

All samples encoded to and decoded from the FLAC format MUST be in a signed representation.

There are several ways to convert unsigned sample representations to signed sample representations, but the coding methods provided by the FLAC format work best on audio signals of which the numerical values of the samples are centered around zero, i.e. have no DC offset. In most unsigned audio formats, signals are centered around halfway the range of the unsigned integer type used. If that is the case, all sample representations SHOULD be converted by first copying the number to a signed integer with sufficient range and then subtracting half of the range of the unsigned integer type, which should result in a signal with samples centered around 0.

11.2. Overview

Before the formal description of the stream, an overview might be helpful.

* A FLAC bitstream consists of the "fLaC" (i.e. 0x664C6143) marker at the beginning of the stream, followed by a mandatory metadata block (called the STREAMINFO block), any number of other metadata blocks, then the audio frames.

* FLAC 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.

  - 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 metadata will be edited after encoding; the user can instruct the encoder to reserve a PADDING block of sufficient size so that when metadata is added, it will simply overwrite the padding (which is relatively quick) instead of having to insert it into the right place in the existing file (which would normally require rewriting the entire file).

- 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 is of the block is defined by the registered application. Visit the registration page (https://xiph.org/flac/id.html) if you would like to register an ID for your application with FLAC.

- 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 bitrate 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 SEEKTABLE in a stream, but the table can have any number of seek points. There is also a special 'placeholder' seekpoint which will be ignored by decoders but which can be used to reserve space for future seek point insertion.

- VORBIS_COMMENT: This block is for storing a list of human-readable name/value pairs. Values are encoded using UTF-8. It is an implementation of the Vorbis comment specification (http://xiph.org/vorbis/doc/v-comment.html) (without the framing bit). This is the only officially supported tagging mechanism in FLAC. There MUST be only zero or one VORBIS_COMMENT blocks in a stream. In some external documentation, Vorbis comments are called FLAC tags to lessen confusion.

- CUESHEET: This block is for storing various information that can be used in a cue sheet. It supports track and index points, compatible with Red Book CD digital audio discs, as well as other CD-DA metadata such as media catalog number and track ISRCs. The CUESHEET block is especially useful for backing up CD-DA discs, but it can be used as a general purpose cueing mechanism for playback.

- PICTURE: This block is for storing pictures associated with the file, most commonly cover art from CDs. There MAY be more than one PICTURE block in a file. The picture format is similar to the APIC frame in ID3v2 (http://www.id3.org/id3v2.4.0-frames). The PICTURE block has a type, MIME type, and UTF-8 description like ID3v2, and supports external linking via URL (though this is discouraged). The differences are that there is no uniqueness constraint on the description field, and the MIME type is mandatory. The FLAC PICTURE block also includes the resolution, color depth, and palette size so that the client can search for a suitable picture without having to scan them all.

* The audio data is composed of one or more audio frames. Each frame consists of a frame header, which contains a sync code, information about the frame like the block size, sample rate, number of channels, et cetera, and an 8-bit CRC. The frame header also contains either the sample number of the first sample in the frame (for variable-blocksize streams), or the frame number (for fixed-blocksize streams). This allows for fast, sample-accurate seeking to be performed. Following the frame header are encoded subframes, one for each channel, and finally, the frame is zero-padded to a byte boundary. Each subframe has its own header that specifies how the subframe is encoded.

* Since a decoder MAY start decoding in the middle of a stream, there MUST be a method to determine the start of a frame. A 14-bit sync code begins each frame. The sync code will not appear anywhere else in the frame header. However, since it MAY appear in the subframes, the decoder has two other ways of ensuring a correct sync. The first is to check that the rest of the frame header contains no invalid data. Even this is not foolproof since valid header patterns can still occur within the subframes. The decoder's final check is to generate an 8-bit CRC of the frame header and compare this to the CRC stored at the end of the frame header.

* Again, since a decoder MAY start decoding at an arbitrary frame in the stream, each frame header MUST contain some basic information about the stream because the decoder MAY not have access to the STREAMINFO metadata block at the start of the stream. This information includes sample rate, bits per sample, number of channels, etc. Since the frame header is pure overhead, it has a direct effect on the compression ratio. To keep the frame header as small as possible, FLAC uses lookup tables for the most commonly used values for frame parameters. For instance, the sample rate part of the frame header is specified using 4 bits. Eight of the bit patterns correspond to the commonly used sample

rates of 8, 16, 22.05, 24, 32, 44.1, 48 or 96 kHz. However, odd sample rates can be specified by using one of the 'hint' bit patterns, directing the decoder to find the exact sample rate at the end of the frame header. The same method is used for specifying the block size and bits per sample. In this way, the frame header size stays small for all of the most common forms of audio data.

* Individual subframes (one for each channel) are coded separately within a frame, and appear serially in the stream. In other words, the encoded audio data is NOT channel-interleaved. This reduces decoder complexity at the cost of requiring larger decode buffers. Each subframe has its own header specifying the attributes of the subframe, like prediction method and order, residual coding parameters, etc. The header is followed by the encoded audio data for that channel.

11.3. Subset

FLAC specifies a subset of itself as the Subset format. The purpose of this is to ensure that any streams encoded according to the Subset are truly "streamable", meaning that a decoder that cannot seek within the stream can still pick up in the middle of the stream and start decoding. It also makes hardware decoder implementations more practical by limiting the encoding parameters such that decoder buffer sizes and other resource requirements can be easily determined. *flac* generates Subset streams by default unless the "--lax" command-line option is used. The Subset makes the following limitations on what MAY be used in the stream:

* The blocksize bits in the FRAME_HEADER (see FRAME_HEADER section (#frameheader)) MUST be 0b0001-0b1110. The blocksize MUST be <= 16384; if the sample rate is <= 48000 Hz, the blocksize MUST be <= 4608 = 2^9 * 3^2.

* The sample rate bits in the FRAME_HEADER MUST be 0b0001-0b1110.

* The bits-per-sample bits in the FRAME_HEADER MUST be 0b001-0b111.

* If the sample rate is <= 48000 Hz, the filter order in LPC subframes (see SUBFRAME_LPC section (#subframelpc)) MUST be less than or equal to 12, i.e. the subframe type bits in the SUBFRAME_HEADER (see SUBFRAME_HEADER section (#subframeheader)) SHOULD NOT be 0b101100-0b111111.

* The Rice partition order (see Coded residual section (#coded-residual)) MUST be less than or equal to 8.

11.4. Conventions

The following tables constitute a formal description of the FLAC format. Values expressed as u(n) represent unsigned big-endian integer using n bits. n may be expressed as an equation using * (multiplication), / (division), + (addition), or - (subtraction). An inclusive range of the number of bits expressed may be represented with an ellipsis, such as u(m...n). The name of a value followed by an asterisk * indicates zero or more occurrences of the value. The name of a value followed by a plus sign + indicates one or more occurrences of the value.

11.5. STREAM

Data	Description
u(32)	"fLaC", the FLAC stream marker in ASCII, meaning byte 0 of the stream is 0x66, followed by 0x4C 0x61 0x43
METADATA_BLOCK_STREAMINFO	This is the mandatory STREAMINFO metadata block that has the basic properties of the stream.
METADATA_BLOCK*	Zero or more metadata blocks
FRAME+	One or more audio frames

Table 1

11.6. METADATA_BLOCK

Data	Description
METADATA_BLOCK_HEADER	A block header that specifies the type and size of the metadata block data.
METADATA_BLOCK_DATA	

Table 2

11.7. METADATA_BLOCK_HEADER

Data	Description
u(1)	Last-metadata-block flag: '1' if this block is the last metadata block before the audio blocks, '0' otherwise.
u(7)	BLOCK_TYPE
u(24)	Length (in bytes) of metadata to follow (does not include the size of the METADATA_BLOCK_HEADER)

Table 3

11.8. BLOCK_TYPE

Value	Description
0	STREAMINFO
1	PADDING
2	APPLICATION
3	SEEKTABLE
4	VORBIS_COMMENT
5	CUESHEET
6	PICTURE
7 - 126	reserved
127	invalid, to avoid confusion with a frame sync code

Table 4

11.9. METADATA_BLOCK_DATA

Data	Description
METADATA_BLOCK_STREAMINFO || METADATA_BLOCK_PADDING || METADATA_BLOCK_APPLICATION || METADATA_BLOCK_SEEKTABLE || METADATA_BLOCK_VORBIS_COMMENT || METADATA_BLOCK_CUESHEET || METADATA_BLOCK_PICTURE	The block data MUST match the block type in the block header.

Table 5

11.10. METADATA_BLOCK_STREAMINFO

Data	Description
u(16)	The minimum block size (in samples) used in the stream.
u(16)	The maximum block size (in samples) used in the stream. (Minimum blocksize == maximum blocksize) implies a fixed-blocksize stream.
u(24)	The minimum frame size (in bytes) used in the stream. A value of 0 signifies that the value is not known.
u(24)	The maximum frame size (in bytes) used in the stream. A value of 0 signifies that the value is not known.
u(20)	Sample rate in Hz. Though 20 bits are available, the maximum sample rate is limited by the structure of frame headers to 655350 Hz. Also, a value of 0 is invalid.
u(3)	(number of channels)-1. FLAC supports from 1 to 8 channels
u(5)	(bits per sample)-1. FLAC supports from 4 to 32 bits per sample. Currently the reference encoder and decoders only support up to 24 bits per sample.

u(36)	Total samples in stream. 'Samples' means inter-channel sample, i.e. one second of 44.1 kHz audio will have 44100 samples regardless of the number of channels. A value of zero here means the number of total samples is unknown.
u(128)	MD5 signature of the unencoded audio data. This allows the decoder to determine if an error exists in the audio data even when the error does not result in an invalid bitstream.

Table 6

FLAC specifies a minimum block size of 16 and a maximum block size of 65535, meaning the bit patterns corresponding to the numbers 0-15 in the minimum blocksize and maximum blocksize fields are invalid.

The MD5 signature is made by performing an MD5 transformation on the samples of all channels interleaved, represented in signed, little-endian form. This interleaving is on a per-sample basis, so for a stereo file this means first the first sample of the first channel, then the first sample of the second channel, then the second sample of the first channel etc. Before performing the MD5 transformation, all samples must be byte-aligned. So, in case the bit depth is not a whole number of bytes, additional zero bits are inserted at the most-significant position until each sample representation is a whole number of bytes.

11.11. METADATA_BLOCK_PADDING

Data	Description
u(n)	n '0' bits (n MUST be a multiple of 8)

Table 7

11.12. METADATA_BLOCK_APPLICATION

Data	Description
u(32)	Registered application ID. (Visit the registration page (https://xiph.org/flac/id.html) to register an ID with FLAC.)

u(n)	Application data (n MUST be a multiple of 8)

Table 8

11.13. METADATA_BLOCK_SEEKTABLE

Data	Description
SEEKPOINT+	One or more seek points.

Table 9

NOTE - The number of seek points is implied by the metadata header 'length' field, i.e. equal to length / 18.

11.14. SEEKPOINT

Data	Description
u(64)	Sample number of first sample in the target frame, or 0xFFFFFFFFFFFFFFFF for a placeholder point.
u(64)	Offset (in bytes) from the first byte of the first frame header to the first byte of the target frame's header.
u(16)	Number of samples in the target frame.

Table 10

NOTES

* For placeholder points, the second and third field values are undefined.

* Seek points within a table MUST be sorted in ascending order by sample number.

* Seek points within a table MUST be unique by sample number, with the exception of placeholder points.

* The previous two notes imply that there MAY be any number of placeholder points, but they MUST all occur at the end of the table.

11.15. METADATA_BLOCK_VORBIS_COMMENT

Data	Description
u(n)	Also known as FLAC tags, the contents of a vorbis comment packet as specified here (http://www.xiph.org/vorbis/doc/v-comment.html) (without the framing bit). Note that the vorbis comment spec allows for on the order of 2^64 bytes of data where as the FLAC metadata block is limited to 2^24 bytes. Given the stated purpose of vorbis comments, i.e. human-readable textual information, this limit is unlikely to be restrictive. Also note that the 32-bit field lengths are little-endian coded according to the vorbis spec, as opposed to the usual big-endian coding of fixed-length integers in the rest of FLAC.

Table 11

11.16. METADATA_BLOCK_CUESHEET

Data	Description
u(128*8)	Media catalog number, in ASCII printable characters 0x20-0x7E. In general, the media catalog number SHOULD be 0 to 128 bytes long; any unused characters SHOULD be right-padded with NUL characters. For CD-DA, this is a thirteen digit number, followed by 115 NUL bytes.
u(64)	The number of lead-in samples. This field has meaning only for CD-DA cuesheets; for other uses it SHOULD be 0. For CD-DA, the lead-in is the TRACK 00 area where the table of contents is stored; more precisely, it is the number of samples from the first sample of the media to the first sample of the first index point of the first track. According to the Red Book, the lead-in MUST be silence and CD grabbing software does not usually store it; additionally, the lead-in MUST be at least two

	seconds but MAY be longer. For these reasons the lead-in length is stored here so that the absolute position of the first track can be computed. Note that the lead-in stored here is the number of samples up to the first index point of the first track, not necessarily to INDEX 01 of the first track; even the first track MAY have INDEX 00 data.
u(1)	1 if the CUESHEET corresponds to a Compact Disc, else 0.
u(7+258*8)	Reserved. All bits MUST be set to zero.
u(8)	The number of tracks. Must be at least 1 (because of the requisite lead-out track). For CD-DA, this number MUST be no more than 100 (99 regular tracks and one lead-out track).
CUESHEET_TRACK+	One or more tracks. A CUESHEET block is REQUIRED to have a lead-out track; it is always the last track in the CUESHEET. For CD-DA, the lead-out track number MUST be 170 as specified by the Red Book, otherwise it MUST be 255.

Table 12

11.17. CUESHEET_TRACK

Data	Description
u(64)	Track offset in samples, relative to the beginning of the FLAC audio stream. It is the offset to the first index point of the track. (Note how this differs from CD-DA, where the track's offset in the TOC is that of the track's INDEX 01 even if there is an INDEX 00.) For CD-DA, the offset MUST be evenly divisible by 588 samples (588 samples = 44100 samples/s * 1/75 s).
u(8)	Track number. A track number of 0 is not allowed to avoid conflicting with the CD-DA spec, which reserves this for the lead-in. For CD-DA the number MUST be 1-99, or 170 for the lead-out; for

	non-CD-DA, the track number MUST for 255 for the lead-out. It is not REQUIRED but encouraged to start with track 1 and increase sequentially. Track numbers MUST be unique within a CUESHEET.
u(12*8)	Track ISRC. This is a 12-digit alphanumeric code; see here (http://isrc.ifpi.org/) and here (http://www.disctronics.co.uk/technology/cdaudio/cdaud_isrc.htm). A value of 12 ASCII NUL characters MAY be used to denote absence of an ISRC.
u(1)	The track type: 0 for audio, 1 for non-audio. This corresponds to the CD-DA Q-channel control bit 3.
u(1)	The pre-emphasis flag: 0 for no pre-emphasis, 1 for pre-emphasis. This corresponds to the CD-DA Q-channel control bit 5; see here (http://www.chipchapin.com/CDMedia/cdda9.php3).
u(6+13*8)	Reserved. All bits MUST be set to zero.
u(8)	The number of track index points. There MUST be at least one index in every track in a CUESHEET except for the lead-out track, which MUST have zero. For CD-DA, this number SHOULD NOT be more than 100.
CUESHEET_TRACK_INDEX+	For all tracks except the lead-out track, one or more track index points.

Table 13

11.18. CUESHEET_TRACK_INDEX

Data	Description
u(64)	Offset in samples, relative to the track offset, of the index point. For CD-DA, the offset MUST be evenly divisible by 588 samples (588 samples = 44100 samples/s * 1/75 s). Note that the offset is from the beginning of the track, not the beginning of the audio data.
u(8)	The index point number. For CD-DA, an index number of 0 corresponds to the track pre-gap. The first index in

		a track MUST have a number of 0 or 1, and subsequently, index numbers MUST increase by 1. Index numbers MUST be unique within a track.
u(3*8)	Reserved. All bits MUST be set to zero.	

Table 14

11.19. METADATA_BLOCK_PICTURE

Data	Description
u(32)	The PICTURE_TYPE according to the ID3v2 APIC frame.
u(32)	The length of the MIME type string in bytes.
u(n*8)	The MIME type string, in printable ASCII characters 0x20-0x7E. The MIME type MAY also be --> to signify that the data part is a URL of the picture instead of the picture data itself.
u(32)	The length of the description string in bytes.
u(n*8)	The description of the picture, in UTF-8.
u(32)	The width of the picture in pixels.
u(32)	The height of the picture in pixels.
u(32)	The color depth of the picture in bits-per-pixel.
u(32)	For indexed-color pictures (e.g. GIF), the number of colors used, or 0 for non-indexed pictures.
u(32)	The length of the picture data in bytes.
u(n*8)	The binary picture data.

Table 15

11.20. PICTURE_TYPE

Value	Description
0	Other
1	32x32 pixels 'file icon' (PNG only)
2	Other file icon
3	Cover (front)
4	Cover (back)
5	Leaflet page
6	Media (e.g. label side of CD)
7	Lead artist/lead performer/soloist
8	Artist/performer
9	Conductor
10	Band/Orchestra
11	Composer
12	Lyricist/text writer
13	Recording Location
14	During recording
15	During performance
16	Movie/video screen capture
17	A bright colored fish
18	Illustration
19	Band/artist logotype
20	Publisher/Studio logotype

Table 16

Other values are reserved and SHOULD NOT be used. There MAY only be one each of picture type 1 and 2 in a file.

11.21. FRAME

Data	Description
FRAME_HEADER	
SUBFRAME+	One SUBFRAME per channel.
u(?)	Zero-padding to byte alignment.
FRAME_FOOTER	

Table 17

11.22. FRAME_HEADER

Data	Description
u(14)	Sync code '0b11111111111110'
u(1)	FRAME HEADER RESERVED
u(1)	BLOCKING STRATEGY
u(4)	INTERCHANNEL SAMPLE BLOCK SIZE
u(4)	SAMPLE RATE
u(4)	CHANNEL ASSIGNMENT
u(3)	SAMPLE SIZE
u(1)	FRAME HEADER RESERVED2
u(?)	CODED NUMBER
u(?)	BLOCK SIZE INT
u(?)	SAMPLE RATE INT

u(8)	FRAME CRC

Table 18

11.22.1. FRAME HEADER RESERVED

Value	Description
0	mandatory value
1	reserved for future use

Table 19

FRAME HEADER RESERVED MUST remain reserved for 0 in order for a FLAC frame's initial 15 bits to be distinguishable from the start of an MPEG audio frame (see also (http://lists.xiph.org/pipermail/flac-dev/2008-December/002607.html)).

11.22.2. BLOCKING STRATEGY

Value	Description
0	fixed-blocksize stream; frame header encodes the frame number
1	variable-blocksize stream; frame header encodes the sample number

Table 20

The BLOCKING STRATEGY bit MUST be the same throughout the entire stream.

The BLOCKING STRATEGY bit determines how to calculate the sample number of the first sample in the frame. If the bit is 0 (fixed-blocksize), the frame header encodes the frame number as above, and the frame's starting sample number will be the frame number times the blocksize. If it is 1 (variable-blocksize), the frame header encodes the frame's starting sample number itself. (In the case of a fixed-blocksize stream, only the last block MAY be shorter than the stream blocksize; its starting sample number will be calculated as the frame number times the previous frame's blocksize, or zero if it is the first frame).

11.22.3. INTERCHANNEL SAMPLE BLOCK SIZE

Value	Description
0b0000	reserved
0b0001	192 samples
0b0010 - 0b0101	576 * (2^(n-2)) samples, i.e. 576, 1152, 2304 or 4608
0b0110	get 8 bit (blocksize-1) from end of header
0b0111	get 16 bit (blocksize-1) from end of header
0b1000 - 0b1111	256 * (2^(n-8)) samples, i.e. 256, 512, 1024, 2048, 4096, 8192, 16384 or 32768

Table 21

11.22.4. SAMPLE RATE

Value	Description
0b0000	get from STREAMINFO metadata block
0b0001	88.2 kHz
0b0010	176.4 kHz
0b0011	192 kHz

0b0100	8 kHz
0b0101	16 kHz
0b0110	22.05 kHz
0b0111	24 kHz
0b1000	32 kHz
0b1001	44.1 kHz
0b1010	48 kHz
0b1011	96 kHz
0b1100	get 8 bit sample rate (in kHz) from end of header
0b1101	get 16 bit sample rate (in Hz) from end of header
0b1110	get 16 bit sample rate (in daHz) from end of header
0b1111	invalid, to prevent sync-fooling string of 1s

Table 22

11.22.5. CHANNEL ASSIGNMENT

Values 0b0000-0b0111 represent the (number of independent channels)-1 coded independently, channel order follows SMPTE/ITU-R recommendations. Values 0b1000-0b1010 represent 2 channel (stereo) audio where the signal has been mapped to a different representation, see section on Interchannel Decorrelation (#interchannel-decorrelation).

Value	Description
0b0000	1 channel: mono
0b0001	2 channels: left, right
0b0010	3 channels: left, right, center
0b0011	4 channels: front left, front right, back left, back right
0b0100	5 channels: front left, front right, front center, back/surround left, back/surround right
0b0101	6 channels: front left, front right, front center, LFE, back/surround left, back/surround right
0b0110	7 channels: front left, front right, front center, LFE, back center, side left, side right
0b0111	8 channels: front left, front right, front center, LFE, back left, back right, side left, side right
0b1000	left/side stereo: channel 0 is the left channel, channel 1 is the side(difference) channel
0b1001	right/side stereo: channel 0 is the side(difference) channel, channel 1 is the right channel
0b1010	mid/side stereo: channel 0 is the mid(average) channel, channel 1 is the side(difference) channel
0b1011 - 0b1111	reserved

Table 23

Please note that the actual coded subframe order for right/side stereo is side-right.

11.22.6. SAMPLE SIZE

Value	Description
0b000	get from STREAMINFO metadata block
0b001	8 bits per sample
0b010	12 bits per sample
0b011	reserved
0b100	16 bits per sample
0b101	20 bits per sample
0b110	24 bits per sample
0b111	reserved

Table 24

For subframes that encode a difference channel, the sample size is one bit larger than the sample size of the frame, in order to be able to encode the difference between extreme values.

11.22.7. FRAME HEADER RESERVED2

Value	Description
0	mandatory value
1	reserved for future use

Table 25

11.22.8. CODED NUMBER

Frame/Sample numbers are encoded using the UTF-8 format, from BEFORE it was limited to 4 bytes by RFC3629, this variant supports the original 7 byte maximum.

Note to implementors: All Unicode compliant UTF-8 decoders and encoders are limited to 4 bytes, it's best to just write your own one off solution.

```
if(variable blocksize)
  `u(8...56)`: "UTF-8" coded sample number (decoded number is 36 bits)
else
  `u(8...48)`: "UTF-8" coded frame number (decoded number is 31 bits)
```

11.22.9. BLOCK SIZE INT

```
if(`INTERCHANNEL SAMPLE BLOCK SIZE` == 0b0110)
  8 bit (blocksize-1)
else if(`INTERCHANNEL SAMPLE BLOCK SIZE` == 0b0111)
  16 bit (blocksize-1)
```

11.22.10. SAMPLE RATE INT

```
if(`SAMPLE RATE` == 0b1100)
  8 bit sample rate (in kHz)
else if(`SAMPLE RATE` == 0b1101)
  16 bit sample rate (in Hz)
else if(`SAMPLE RATE` == 0b1110)
  16 bit sample rate (in daHz)
```

11.22.11. FRAME CRC

CRC-8 (polynomial = x^8 + x^2 + x^1 + x^0, initialized with 0) of everything before the CRC, including the sync code

11.23. FRAME_FOOTER

Data	Description
u(16)	CRC-16 (polynomial = x^16 + x^15 + x^2 + x^0, initialized with 0) of everything before the CRC, back to and including the frame header sync code

Table 26

11.24. SUBFRAME

Data	Description
SUBFRAME_HEADER	
SUBFRAME_CONSTANT || SUBFRAME_FIXED || SUBFRAME_LPC || SUBFRAME_VERBATIM	The SUBFRAME_HEADER specifies which one.

Table 27

11.25. SUBFRAME_HEADER

Data	Description
u(1)	Zero bit padding, to prevent sync-fooling string of 1s
u(6)	SUBFRAME TYPE (see section on SUBFRAME TYPE (#subframe-type))
u(1+k)	WASTED BITS PER SAMPLE FLAG (see section on WASTED BITS PER SAMPLE FLAG (#wasted-bits-per-sample-flag))

Table 28

11.25.1. SUBFRAME TYPE

Value	Description
0b000000	SUBFRAME_CONSTANT
0b000001	SUBFRAME_VERBATIM
0b00001x	reserved
0b0001xx	reserved
0b001xxx	if(xxx <= 4) SUBFRAME_FIXED, xxx=order; else reserved
0b01xxxx	reserved
0b1xxxxx	SUBFRAME_LPC, xxxxx=order-1

Table 29

11.25.2. WASTED BITS PER SAMPLE FLAG

Certain file formats, like AIFF, can store audio samples with a bit depth that is not an integer number of bytes by padding them with least significant zero bits to a bit depth that is an integer number of bytes. For example, shifting a 14-bit sample right by 2 pads it to a 16-bit sample, which then has two zero least-significant bits. In this specification, these least-significant zero bits are referred to as wasted bits-per-sample or simply wasted bits. They are wasted in a sense that they contain no information, but are stored anyway.

The wasted bits-per-sample flag in a subframe header is set to 1 if a certain number of least-significant bits of all samples in the current subframe are zero. If this is the case, the number of wasted bits-per-sample (k) minus 1 follows the flag in an unary encoding. For example, if k is 3, 0b001 follows. If k = 0, the wasted bits-per-sample flag is 0 and no unary coded k follows.

In case k is not equal to 0, samples are coded ignoring k least-significant bits. For example, if the preceding frame header specified a sample size of 16 bits per sample and k is 3, samples in the subframe are coded as 13 bits per sample. A decoder MUST add k least-significant zero bits by shifting left (padding) after decoding a subframe sample. In case the frame has left/side, right/side or mid/side stereo, padding MUST happen to a sample before it is used to reconstruct a left or right sample.

Besides audio files that have a certain number of wasted bits for the whole file, there exist audio files in which the number of wasted bits varies. There are DVD-Audio discs in which blocks of samples have had their least-significant bits selectively zeroed, as to slightly improve the compression of their otherwise lossless Meridian Lossless Packing codec. There are also audio processors like lossyWAV that enable users to improve compression of their files by a lossless audio codec in a non-lossless way. Because of this the number of wasted bits k MAY change between frames and MAY differ between subframes.

11.26. SUBFRAME_CONSTANT

Data	Description
u(n)	Unencoded constant value of the subblock, n = frame's bits-per-sample.

Table 30

11.27. SUBFRAME_FIXED

Data	Description
u(n)	Unencoded warm-up samples (n = frame's bits-per-sample * predictor order).
RESIDUAL	Encoded residual

Table 31

11.28. SUBFRAME_LPC

Data	Description
u(n)	Unencoded warm-up samples (n = frame's bits-per-sample * lpc order).
u(4)	(quantized linear predictor coefficients' precision in bits)-1 (NOTE: 0b1111 is invalid).
u(5)	Quantized linear predictor coefficient shift needed in bits (NOTE: this number is signed two's-complement).
u(n)	Unencoded predictor coefficients (n = qlp coeff precision * lpc order) (NOTE: the coefficients are signed two's-complement).
RESIDUAL	Encoded residual

Table 32

11.29. SUBFRAME_VERBATIM

Data	Description
u(n*i)	Unencoded subblock, where n is frame's bits-per-sample and i is frame's blocksize.

Table 33

11.30. RESIDUAL

Data	Description
u(2)	RESIDUAL_CODING_METHOD
RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB ||| RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB2	

Table 34

11.30.1. RESIDUAL_CODING_METHOD

Value	Description
0b00	partitioned Exp-Golomb coding with 4-bit Exp-Golomb parameter; RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB follows
0b01	partitioned Exp-Golomb coding with 5-bit Exp-Golomb parameter; RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB2 follows
0b10 - 0b11	reserved

Table 35

11.30.2. RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB

Data	Description
u(4)	Partition order.
EXP_GOLOMB_PARTITION+	There will be 2^order partitions.

Table 36

11.30.2.1. EXP_GOLOMB_PARTITION

Data	Description
u(4(+5))	EXP-GOLOMB PARTITION ENCODING PARAMETER (see section on EXP-GOLOMB PARTITION ENCODING PARAMETER (#exp-golomb-partition-encoding-parameter))
u(?)	ENCODED RESIDUAL (see section on ENCODED RESIDUAL (#encoded-residual))

Table 37

11.30.2.2. EXP GOLOMB PARTITION ENCODING PARAMETER

Value	Description
0b0000 - 0b1110	Exp-golomb parameter.
0b1111	Escape code, meaning the partition is in unencoded binary form using n bits per sample; n follows as a 5-bit number.

Table 38

11.30.3. RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB2

Data	Description
u(4)	Partition order.
EXP-GOLOMB2_PARTITION+	There will be 2^order partitions.

Table 39

11.30.3.1. EXP_GOLOMB2_PARTITION

Data	Description
u(5(+5))	EXP-GOLOMB2 PARTITION ENCODING PARAMETER (see section on EXP-GOLOMB2 PARTITION ENCODING PARAMETER (#expgolomb2-partition-encoding-parameter))
u(?)	ENCODED RESIDUAL (see section on ENCODED RESIDUAL (#encoded-residual))

Table 40

11.30.3.2. EXP-GOLOMB2 PARTITION ENCODING PARAMETER

Value	Description
0b00000 - 0b11110	Exp-golomb parameter.
0b11111	Escape code, meaning the partition is in unencoded binary form using n bits per sample; n follows as a 5-bit number.

Table 41

11.30.4. ENCODED RESIDUAL

The number of samples (n) in the partition is determined as follows:

* if the partition order is zero, n = frame's blocksize - predictor order

* else if this is not the first partition of the subframe, n = (frame's blocksize / (2^partition order))

* else n = (frame's blocksize / (2^partition order)) - predictor order

12. Security Considerations

Like any other codec (such as [RFC6716]), FLAC should not be used with insecure ciphers or cipher modes that are vulnerable to known plaintext attacks. Some of the header bits as well as the padding are easily predictable.

Implementations of the FLAC codec need to take appropriate security considerations into account. Those related to denial of service are outlined in Section 2.1 of [RFC4732]. It is extremely important for the decoder to be robust against malicious payloads. Malicious payloads MUST NOT cause the decoder to overrun its allocated memory or to take an excessive amount of resources to decode. An overrun in allocated memory could lead to arbitrary code execution by an attacker. The same applies to the encoder, even though problems in encoders are typically rarer. Malicious audio streams MUST NOT cause the encoder to misbehave because this would allow an attacker to attack transcoding gateways. An example is allocating more memory than available especially with blocksizes of more than 10000 or with big metadata blocks, or not allocating enough memory before copying data, which lead to execution of malicious code, crashes, freezes or reboots on some known implementations. See the FLAC decoder testbench (https://wiki.hydrogenaud.io/index.php?title=FLAC_decoder_testbench) for a non-exhaustive list of FLAC files with extreme configurations which lead to crashes or reboots on some known implementations.

None of the content carried in FLAC is intended to be executable.

13. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/info/rfc4732>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

14. Informative References

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <https://www.rfc-editor.org/info/rfc6716>.

Authors' Addresses

Michael Richardson

Email: mcr@sandelman.ca

Andrew Weaver

Email: theandrewjw@gmail.com