Workgroup:	cellar
Internet-Draft:	draft-ietf-cellar-flac-03
Published:	23 April 2022
Intended Status:	Standards Track
Expires:	25 October 2022

Authors:	M.Q.C. van Beurden A. Weaver

Free Lossless Audio Codec

Abstract

This document defines the Free Lossless Audio Codec (FLAC) format. FLAC 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. 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.

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 25 October 2022.

Copyright Notice

Table of Contents

1. Introduction

This document defines the FLAC format. FLAC files and streams can code 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 coding to and decoding from 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.

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 coding methods provided by the FLAC format work 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.

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.

Values expressed as `u(n)` represent unsigned big-endian integer using `n` bits. Values expressed as `s(n)` represent signed big-endian integer using `n` bits, signed two's complement. `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.

3. Acknowledgments

FLAC owes much to the many people who have advanced the audio compression field so freely. For instance:

- A. J. Robinson for his work on Shorten; his paper ([robinson-tr156]) 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 Rice coding used in Shorten.
- S. W. Golomb 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

4. Definitions

- **Lossless compression**: reducing the amount of computer storage space needed to store data without needing to remove or irreversibly alter any of this data in doing so. In other words, decompressing losslessly compressed information returns exactly the original data.
- **Lossy compression**: like lossless compression, but instead removing, irreversibly altering or only approximating information for the purpose of further reducing the amount of computer storage space needed. In other words, decompressing lossy compressed information returns an approximation of the original data.
- **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.

- **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.
- **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. 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.
- **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 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.

5. Conceptual overview

Similar to many audio coders, a FLAC file is encoded following the steps below. On decoding a FLAC file, these steps are undone in reverse order, i.e. from bottom to top.

- `Blocking` (see section on Blocking). The input is split 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. However, as finding the optimal block size arrangement is a rather complex problem, the FLAC format allows for a constant block size throughout a stream as well.
- `Interchannel Decorrelation` (see section on Interchannel Decorrelation). In the case of stereo streams, the FLAC format allows for transforming the left-right signal into a mid-side signal to remove redundancy, if there is any. Besides coding as left-right and mid-side, it is also possible to code left-side and side-right, whichever ordering results in the highest compression. Choosing between any of these transformation is done independently for each block.
- `Prediction` (see section on Prediction). To remove redundancy in a signal, a predictor is stored for each subblock or its transformation as formed in the previous step. A predictor consists of a simple mathematical description that can be used, as the name implies, to predict a certain sample from the samples that preceded it. As this prediction is rarely exact, the error of this prediction is passed to the next stage. The predictor of each subblock is completely independent from other subblocks. 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. In case no usable predictor can be found for a certain subblock, the signal is stored instead of compressed and the next stage is skipped.

- `Residual Coding` (See section on Residual Coding). As 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 uses Rice coding, a subset of Golomb coding, with either 4-bit or 5-bit parameters to code the residual signal.

In addition, FLAC specifies a metadata system (see section on File-level metadata), which allows arbitrary information about the stream to be included at the beginning of the stream.

5.1. 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.

While the block size MAY vary in a FLAC file, it is often difficult to find the optimal arrangement of block sizes for maximum compression. Because of this the FLAC format explicitly stores whether a file has a constant or a variable blocksize throughout the stream, and stores a block number instead of a sample number to slighly improve compression in case a stream has a constant block size.

Blocked data is passed to the predictor stage one subblock 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, subframes MAY use different predictors, even within a frame.

5.2. 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 right 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 right one bit. The right 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.

5.3. Prediction

The FLAC format has four methods for modeling the input signal:

1. **Verbatim**. Samples are stored directly, without any modelling. This method is used for inputs with little correlation like white noise. Since the raw signal is not actually passed through the residual coding stage (it is added to the stream 'verbatim'), the method is different from using a zero-order fixed predictor.
2. **Constant**. A single sample value is stored. This method is used whenever a signal is pure DC ("digital silence"), i.e. a constant value throughout.
3. **Fixed predictor**. Samples are predicted with one of five fixed (i.e. predefined) predictors, the error of this prediction is processed by the residual coder. These fixed predictors are well suited for predicting simple waveforms. Since the predictors are fixed, no predictor coefficients are stored. From a mathematical point of view, the predictors work by extrapolating the signal from the previous samples. The number of previous samples used is equal to the predictor order. For more information see the section on the fixed predictor subframe
4. **Linear predictor**. Samples are predicted using past samples and a set of predictor coefficients, the error of this prediction is processed by the residual coder. Compared to a fixed predictor, using a generic linear predictor adds overhead as predictor coefficients need to be stored. Therefore, this method of prediction is best suited for predicting more complex waveforms, where the added overhead is offset by space savings in the residual coding stage resulting from more accurate prediction. A linear predictor in FLAC has two parameters besides the predictor coefficients and the predictor order: the number of bits with which each coefficient is stored (the coefficient precision) and a prediction right shift. A prediction is formed by taking the sum of multiplying each predictor coefficient with the corresponding past sample, and dividing that sum by applying the specified right shift. For more information see the section on the linear predictor subframe

For more information on fixed and linear predictors, see [HPL-1999-144] and [robinson-tr156].

5.4. Residual Coding

In case a subframe uses a predictor to approximate the audio signal, a residual needs to be stored to 'correct' the approximation to the exact value. When an effective predictor is used, the average numerical value of the residual samples is smaller than that of the samples before prediction. While having smaller values on average, it is possible a few 'outlier' residual samples are much larger than any of the original samples. Sometimes these outliers even exceed the range the bit depth of the original audio offers.

To be able to efficiently code such a stream of relatively small numbers with an occasional outlier, Rice coding (a subset of Golomb coding) is used. Depending on how small the numbers are that have to be coded, a Rice parameter is chosen. The numerical value of each residual sample is split in two parts by dividing it with `2^(Rice parameter)`, creating a quotient and a remainder. The quotient is stored in unary form, the remainder in binary form. If indeed most residual samples are close to zero and the Rice parameter is chosen right, this form of coding, a so-called variable-length code, needs less bits to store than storing the residual in unencoded form.

As Rice codes can only handle unsigned numbers, signed numbers are zigzag encoded to a so-called folded residual. For more information see section coded residual for a more thorough explanation.

Quite often the optimal Rice parameter varies over the course of a subframe. To accommodate this, the residual can be split up into partitions, where each partition has its own Rice parameter. To keep overhead and complexity low, the number of partitions used in a subframe is limited to powers of two.

The FLAC format uses two forms of Rice coding, which only differ in the number of bits used for encoding the Rice parameter, either 4 or 5 bits.

6. Format 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.

7. Format lay-out

Before the formal description of the stream, an overview of the lay-out of FLAC file 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 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 (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. 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.

8. Format 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 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 depth bits in the frame header MUST be 0b001-0b111.
- If the sample rate is <= 48000 Hz, the filter order in linear subframes (see section linear predictor subframe) MUST be less than or equal to 12, i.e. the subframe type bits in the subframe header (see subframe header section) SHOULD NOT be 0b101100-0b111111.
- The Rice partition order (see coded residual section) MUST be less than or equal to 8.

9. File-level metadata

At the start of a FLAC file or stream, following the fLaC ASCII file signature, one or more metadata blocks MUST be present before any audio frames appear. The first metadata block MUST be a streaminfo block.

9.1. Metadata block header

Each metadata block starts with a 4 byte header. The first bit in this header flags whether a metadata block is the last one, it is a 0 when other metadata blocks follow, otherwise it is a 1. The 7 remaining bits of the first header byte contain the type of the metadata block as an unsigned number between 0 and 126 according to the following table. A value of 127 (i.e. 0b1111111) is invalid. The three bytes that follow code for the size of the metadata block in bytes excluding the 4 header bytes as an unsigned number coded big-endian.

Value	Metadata block type
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 1

9.2. Streaminfo

The streaminfo metadata block contains technical information about the FLAC stream relevant for decoding. Decoder behavior in case of incorrect or incomplete information is left unspecified (i.e. up to the decoder implementation). A decoder MAY choose to stop further decoding in case the information supplied by the streaminfo metadata block turns out to be incorrect or invalid. A decoder accepting information from the streaminfo block (most significantly the maximum frame size, maximum block size, number of audio channels, number of bits per sample and total number of samples) without doing further checks during decoding of audio frames could be vulnerable to buffer overflows. See also the section on security considerations.

Data	Description
`u(16)`	The minimum block size (in samples) used in the stream, excluding the last block.
`u(16)`	The maximum block size (in samples) used in the 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. A value of `0` signifies that the value is not known.

Table 2

The minimum block size is excluding the last block of a FLAC file, which may be smaller. If the minimum block size is equal to the maximum block size, the file contains a fixed block size stream. Note that the actual maximum block size might be smaller than the maximum block size listed in the streaminfo block, and the actual smallest block size excluding the last block might be larger than the minimum block size listed in the streaminfo block. This is because the encoder has to write these fields before receiving any input audio data, and cannot know beforehand what block sizes it will use, only between what bounds these will be chosen.

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 block size and maximum block size 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.

9.3. Padding

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

Table 3

9.4. Application

Data	Description
`u(32)`	Registered application ID. (Visit the registration page to register an ID with FLAC.)
`u(n)`	Application data (n MUST be a multiple of 8)

Table 4

9.5. Seektable

Data	Description
`SEEKPOINT+`	One or more seek points.

Table 5

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

9.5.1. 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 6

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.

9.6. Vorbis comment

A vorbis comment metadata block contains human-readable information coded in UTF-8. The name vorbis comment points to the fact that the vorbis codec stores such metadata in almost the same way. A vorbis comment metadata block consists of a vendor string optionally followed by a number of fields, which are pairs of field names and field contents. Many users refer to these fields as FLAC tags or simply as tags. A FLAC file MUST NOT contain more than one vorbis comment metadata block.

In a vorbis comment metadata block, the metadata block header is directly followed by 4 bytes containing the length in bytes of the vendor string as an unsigned number coded little-endian. The vendor string follows UTF-8 coded, and is not terminated in any way.

Following the vendor string are 4 bytes containing the number of fields that are in the vorbis comment block, stored as an unsigned number, coded little-endian. If this number is non-zero, it is followed by the fields themselves, each field stored with a 4 byte length. First, the 4 byte field length in bytes is stored as an unsigned number, coded little-endian. The field itself is, like the vendor string, UTF-8 coded, not terminated in any way.

Each field consists of a field name and a field content, separated by an = character. The field name MUST only consist of UTF-8 code points U+0020 through U+0074, excluding U+003D, which is the = character. In other words, the field name can contain all printable ASCII characters except the equals sign. The evaluation of the field names MUST be case insensitive, so U+0041 through 0+005A (A-Z) MUST be considered equivalent to U+0061 through U+007A (a-z) respectively. The field contents can contain any UTF-8 character.

Note that the vorbis comment as used in vorbis allows for on the order of 2^64 bytes of data whereas 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 coded little-endian, as opposed to the usual big-endian coding of fixed-length integers in the rest of the FLAC format.

9.6.1. Standard field names

Except the one defined in the section channel mask, no standard field names are defined. In general, most software recognizes the following field names

- Title: name of the current work
- Artist: name of the artist generally responsible for the current work. For orchestral works this is usually the composer, otherwise is it often the performer
- Album: name of the collection the current work belongs to

For a more comprehensive list of possible field names, the list of tags used in the MusicBrainz project is recommended.

9.6.2. Channel mask

Besides fields containing information about the work itself, one field is defined for technical reasons, of which the field name is WAVEFORMATEXTENSIBLE_CHANNEL_MASK. This field contains information on which channels the file contains. Use of this field is RECOMMENDED in case these differ from the channels defined in the section channels bits.

The channel mask consists of flag bits indicating which channels are present, stored in a hexadecimal representation preceded by 0x. The flags only signal which channels are present, not in which order, so in case a file has to be encoded in which channels are ordered differently, they have to be reordered. Please note that a file in which the channel order is defined through the WAVEFORMATEXTENSIBLE_CHANNEL_MASK is not streamable, i.e. non-subset, as the field is not found in each frame header. The mask bits can be found in the following table

Bit number	Channel description
0	Front left
1	Front right
2	Front center
3	Low-frequency effects (LFE)
4	Back left
5	Back right
6	Front left of center
7	Front right of center
8	Back center
9	Side left
10	Side right
11	Top center
12	Top front left
13	Top front center
14	Top front right
15	Top rear left
16	Top rear center
17	Top rear right

Table 7

Following are 3 examples:

- if a file has a single channel, being a LFE channel, the vorbis comment field is WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x8
- if a file has 4 channels, being front left, front right, top front left and top front right, the vorbis comment field is WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x5003
- if an input has 4 channels, being back center, top front center, front center and top rear center in that order, they have to be reordered to front center, back center, top front center and top rear center. The vorbis comment field added is WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x12004.

WAVEFORMATEXTENSIBLE_CHANNEL_MASK fields MAY be padded with zeros, for example, 0x0008 for a single LFE channel. Parsing of WAVEFORMATEXTENSIBLE_CHANNEL_MASK fields MUST be case-insensitive for both the field name and the field contents.

9.7. Cuesheet

To either store the track and index point structure of a CD-DA along with its audio or to provide a mechanism to store locations of interest within a FLAC file, a cuesheet metadata block can be used. Certain aspects of this metadata block follow directly from the CD-DA specification, called Red Book, which is standardized as [IEC.60908.1999]. For more information on the function and history of these aspects, please refer to [IEC.60908.1999].

The structure of a cuesheet metadata block is enumerated in the following table.

Data	Description
`u(128*8)`	Media catalog number, in ASCII printable characters 0x20-0x7E.
`u(64)`	Number of lead-in samples.
`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)`	Number of tracks in this cuesheet.
Cuesheet tracks	A number of structures as specified in the section cuesheet track equal to the number of tracks specified previously.

Table 8

If the media catalog number is less than 128 bytes long, it SHOULD be right-padded with NUL characters. For CD-DA, this is a thirteen digit number, followed by 115 NUL bytes.

The number of lead-in samples 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 [IEC.60908.1999], 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.

The number of tracks MUST be at least 1, as a cuesheet block MUST have a lead-out track. For CD-DA, this number MUST be no more than 100 (99 regular tracks and one lead-out track). The lead-out track is always the last track in the cuesheet. For CD-DA, the lead-out track number MUST be 170 as specified by [IEC.60908.1999], otherwise it MUST be 255.

9.7.1. Cuesheet track

Data	Description
`u(64)`	Track offset of first index point in samples, relative to the beginning of the FLAC audio stream.
`u(8)`	Track number.
`u(12*8)`	Track 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.
`u(6+13*8)`	Reserved. All bits MUST be set to zero.
`u(8)`	The number of track index points.
Cuesheet track index points	For all tracks except the lead-out track, a number of structures as specified in the section cuesheet track index point equal to the number of index points specified previously.

Table 9

Note that the track offset differs from the one in 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 track offset MUST be evenly divisible by 588 samples (588 samples = 44100 samples/s * 1/75 s).

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 RECOMMENDED to start with track 1 and increase sequentially. Track numbers MUST be unique within a cuesheet.

The track ISRC (International Standard Recording Code) is a 12-digit alphanumeric code; see [ISRC-handbook]. A value of 12 ASCII NUL characters MAY be used to denote absence of an ISRC.

There MUST be at least one index point in every track in a cuesheet except for the lead-out track, which MUST have zero. For CD-DA, the number of index points SHOULD NOT be more than 100.

9.7.1.1. Cuesheet track index point

Data	Description
`u(64)`	Offset in samples, relative to the track offset, of the index point.

Data	Description
u(8)	The track index point number.
u(3*8)	Reserved. All bits MUST be set to zero.

Table 10

For CD-DA, the track index point 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.

For CD-DA, an track index point number of 0 corresponds to the track pre-gap. The first index point in a track MUST have a number of 0 or 1, and subsequently, index point numbers MUST increase by 1. Index point numbers MUST be unique within a track.

9.8. Picture

The picture metadata block contains image data of a picture in some way belonging to the audio contained in the FLAC file. Its format is derived from the APIC frame in the ID3v2 specification. However, contrary to the APIC frame in ID3v2, the MIME type and description are prepended with a 4-byte length field instead of being null delimited strings. A FLAC file MAY contain one or more picture metadata blocks.

Note that while the length fields for MIME type, description and picture data are 4 bytes in length and could in theory code for a size up to 4 GiB, the total metadata block size cannot exceed what can be described by the metadata block header, i.e. 16 MiB.

Data	Description
u(32)	The picture type according to next table
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.

Data	Description
`u(n*8)`	The binary picture data.

Table 11

The following table contains all defined picture types. Values other than those listed in the table are reserved and SHOULD NOT be used. There MAY only be one each of picture type 1 and 2 in a file. In general practice, many decoders display the contents of a picture metadata block with picture type 3 (front cover) during playback, if present.

Value	Picture type
0	Other
1	PNG file icon of 32x32 pixels
2	General file icon
3	Front cover
4	Back cover
5	Liner notes page
6	Media label (e.g. CD, Vinyl or Cassette label)
7	Lead artist, lead performer or soloist
8	Artist or performer
9	Conductor
10	Band or orchestra
11	Composer
12	Lyricist or text writer
13	Recording location
14	During recording
15	During performance
16	Movie or video screen capture
17	A bright colored fish
18	Illustration
19	Band or artist logotype
20	Publisher or studio logotype

Table 12

10. Frame structure

Directly after the last metadata block, one or more frames follow. Each frame consists of a frame header, one or more subframes, padding zero bits to achieve byte-alignment and a frame footer. The number of subframes in each frame is equal to the number of audio channels.

10.1. Frame header

Each frame starts with the 15-bit frame sync code 0b111111111111100. Following the sync code is the blocking strategy bit, which MUST NOT change during the audio stream. The blocking strategy bit is 0 for a fixed blocksize stream or 1 for variable blocksize stream. If the blocking strategy is known, a decoder can search for a 16-bit sync code, either 0xF8 for a fixed blocksize stream or 0xF9 for a variable blocksize stream. To ease the search for the sync code and further reduction of false positives, all frames MUST start on a byte boundary.

10.1.1. Blocksize bits

Following the frame sync code and blocksize strategy bit are 4 bits referred to as the blocksize bits. Their value relates to the blocksize according to the following table, where v is the value of the 4 bits as an unsigned number. Uncommon blocksizes are stored after the coded number.

Value	Blocksize
0b0000	reserved
0b0001	192
0b0010 - 0b0101	144 * (2^v), i.e. 576, 1152, 2304 or 4608
0b0110	uncommon blocksize minus 1 stored as an 8-bit number
0b0111	uncommon blocksize minus 1 stored as a 16-bit number
0b1000 - 0b1111	2^v, i.e. 256, 512, 1024, 2048, 4096, 8192, 16384 or 32768

Table 13

10.1.2. Sample rate bits

The next 4 bits, referred to as the sample rate bits, contain the sample rate according to the following table

Value	Sample rate
0b0000	sample rate only stored in 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

Value	Sample rate
0b1000	32 kHz
0b1001	44.1 kHz
0b1010	48 kHz
0b1011	96 kHz
0b1100	uncommon sample rate in kHz stored as an 8-bit number
0b1101	uncommon sample rate in Hz stored as a 16-bit number
0b1110	uncommon sample rate in Hz divided by 10, stored as a 16-bit number
0b1111	invalid

Table 14

10.1.3. Channels bits

The next 4 bits (the first 4 bits of the fourth byte of each frame), referred to as the channel bits, code for both the number of channels as well as any stereo decorrelation used according to the following table, where v is the value of the 4 bits as an unsigned number. See also the section on interchannel decorrelation.

Value	Channels
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	2 channels, stored as left/side stereo
0b1001	2 channels, stored as right/side stereo
0b1010	2 channels, stored as mid/side stereo

Value	Channels
0b1011 - 0b1111	reserved

Table 15

10.1.4. Bit depth bits

The next 3 bits code for the bit depth of the samples in the subframe according to the following table.

Value	Bit depth
0b000	bit depth only stored in 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 16

The next bit is reserved and MUST be zero.

10.1.5. Coded number

Following the reserved bit (starting at the fifth byte of the frame) is either a sample or a frame number, which will be referred to as the coded number. When dealing with variable blocksize streams, the sample number of the first sample in the frame is encoded. When the file contains a fixed blocksize stream, the frame number is encoded. The coded number is stored in a variable length code like UTF-8, but extended to a maximum of 36 bits unencoded, 7 byte encoded. When a frame number is encoded, the value MUST NOT be larger than what fits a value 31 bits unencoded or 6 byte encoded. Please note that most general purpose UTF-8 encoders and decoders will not be able to handle these extended codes.

10.1.6. Uncommon blocksize

If the blocksize bits defined earlier in this section were 0b0110 or 0b0111 (uncommon blocksize minus 1 stored), this follows the coded number as either an 8-bit or a 16-bit unsigned number coded big-endian.

10.1.7. Uncommon sample rate

Following either the coded number or an uncommon blocksize is the sample rate, if the sample rate bits were 0b1100, 0b1101 or 0b1110 (uncommon sample rate stored), as either an 8-bit or a 16-bit unsigned number coded big-endian.

10.1.8. Frame header CRC

Finally, after either the frame/sample number, the blocksize or the sample rate, is a 8-bit CRC. This CRC is initialized with 0 and has the polynomial x^8 + x^2 + x^1 + x^0. This CRC covers the whole frame header before the CRC, including the sync code.

10.2. Subframes

Following the frame header are a number subframes equal to the number of audio channels.

10.2.1. Subframe header

Each subframe starts with a header. The first bit of the header is always 0, followed by 6 bits describing which subframe type is used according to the following table, where v is the value of the 6 bits as an unsigned number.

Value	Subframe type
0b000000	Constant subframe
0b000001	Verbatim subframe
0b000010 - 0b000111	reserved
0b001000 - 0b001100	Subframe with a fixed predictor v-8, i.e. 0, 1, 2, 3 or 4
0b001101 - 0b011111	reserved
0b100000 - 0b111111	Subframe with a linear predictor v-31, i.e. 1 through 32 (inclusive)

Table 17

Following the subframe type bits is a bit that flags whether the subframe has any wasted bits. If it is 0, the subframe doesn't have any wasted bits and the subframe header is complete. If it is 1, the subframe does have wasted bits and the number of wasted bits follows unary coded.

10.2.2. Wasted bits per sample

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.

10.2.3. Constant subframe

In a constant subframe only a single sample is stored. This sample is stored as a integer number coded big-endian, signed two's complement. The number of bits used to store this sample depends on the bit depth of the current subframe. The bit depth of a subframe is equal to the bit depth as coded in the frame header, minus the number of wasted bits coded in the subframe header. In case a subframe is a side subframe (see the section on interchannel decorrelation, the bit depth of that subframe is increased by 1 bit.

10.2.4. Verbatim subframe

A verbatim subframe stores all samples unencoded in sequential order. See section on Constant subframe on how a sample is stored unencoded. The number of samples that need to be stored in a subframe is given by the blocksize in the frame header.

10.2.5. Fixed predictor subframe

Five different fixed predictors are defined in the following table, one for each prediction order 0 through 4. In the table is also a derivation, which explains the rationale for choosing these fixed predictors.

Order	Prediction	Derivation
0	0	N/A
1	s(n-1)	N/A
2	2 * s(n-1) - s(n-2)	s(n-1) + s'(n-1)
3	3 * s(n-1) - 3 * s(n-2) + s(n-3)	s(n-1) + s'(n-1) + s''(n-1)
4	4 * s(n-1) - 6 * s(n-2) + 4 * s(n-3) - s(n-4)	s(n-1) + s'(n-1) + s''(n-1) + s'''(n-1)

Table 18

Where

- n is the number of the sample being predicted
- s(n) is the sample being predicted
- s(n-1) is the sample before the one being predicted
- s'(n-1) is the difference between the previous sample and the sample before that, i.e. s(n-1) - s(n-2). This is the closest available first-order discrete derivative
- s''(n-1) is s'(n-1) - s'(n-2) or the closest available second-order discrete derivative

- s'''(n-1) is s''(n-1) - s''(n-2) or the closest available third-order discrete derivative

To encode a signal with a fixed predictor, each sample has the corresponding prediction subtracted and sent to the residual coder. To decode a signal with a fixed predictor, first the residual has to be decoded, after which for each sample the prediction can be added. This means that decoding MUST be a sequential process within a subframe, as for each sample, enough fully decoded previous samples are needed to calculate the prediction.

For fixed predictor order 0, the prediction is always 0, thus each residual sample is equal to its corresponding input or decoded sample. The difference between a fixed predictor with order 0 and a verbatim subframe, is that a verbatim subframe stores all samples unencoded, while a fixed predictor with order 0 has all its samples processed by the residual coder.

The first order fixed predictor is comparable to how DPCM encoding works, as the resulting residual sample is the difference between the corresponding sample and the sample before it. The higher fixed predictors can be understood as polynomials fitted to the previous samples.

As the fixed predictors are specified, they do not have to be stored. The fixed predictor order specifies which predictor is used. To be able to predict samples, warm-up samples are stored, as the predictor needs previous samples in its prediction. The number of warm-up samples is equal to the predictor order. See section on Constant subframe on how samples are stored unencoded. Directly following the warm-up samples is the coded residual.

10.2.6. Linear predictor subframe

Whereas fixed predictors are well suited for simple signals, using a (non-fixed) linear predictor on more complex signals can improve compression by making the residual samples even smaller. There is a certain trade-off however, as storing the predictor coefficients takes up space as well.

In the FLAC format, a predictor is defined by up to 32 predictor coefficients and a shift. To form a prediction, each coefficient is multiplied with its corresponding past sample, the results are added and this addition is then shifted. To encode a signal with a linear predictor, each sample has the corresponding prediction subtracted and sent to the residual coder. To decode a signal with a linear predictor, first the residual has to be decoded, after which for each sample the prediction can be added. This means that decoding MUST be a sequential process within a subframe, as for each sample, enough fully decoded previous samples are needed to calculate the prediction.

The table below defines how a linear predictor subframe appears in the bitstream

Data	Description
`s(n)`	Unencoded warm-up samples (n = frame's bits-per-sample * lpc order).
`u(4)`	(Predictor coefficient precision in bits)-1 (NOTE: 0b1111 is invalid).
`s(5)`	Prediction right shift needed in bits.
`s(n)`	Unencoded predictor coefficients (n = predictor coefficient precision * lpc order).
`Coded residual`	Encoded residual

Table 19

See section on Constant subframe on how the warm-up samples are stored unencoded. The unencoded predictor coefficients are stored the same way as the warm-up samples, but the number of bits needed for each coefficient is defined by the predictor coefficient precision. While the prediction right shift is signed two's complement, this number MUST be positive.

Please note that the order in which the predictor coefficients appear in the bitstream corresponds to which **past** sample they belong. In other words, the order of the predictor coefficients is opposite to the chronological order of the samples. So, the first predictor coefficient has to be multiplied with the sample directly before the sample that is being predicted, the second predictor coefficient has to be multiplied with the sample before that etc.

10.2.7. Coded residual

The first two bits in a coded residual indicate which coding method is used. See the table below

Value	Description
0b00	partitioned Rice code with 4-bit parameters
0b01	partitioned Rice code with 5-bit parameters
0b10 - 0b11	reserved

Table 20

Both defined coding methods work the same way, but differ in the number of bits used for rice parameters. The 4 bits that directly follow the coding method bits form the partition order, which is an unsigned number. The rest of the coded residual consists of 2^(partition order) partitions. For example, if the 4 bits are 0b1000, the partition order is 8 and the residual is split up into 2^8 = 256 partitions.

Each partition contains a certain amount of residual samples. The number of residual samples in the first partition is equal to (blocksize >> partition order) - predictor order, i.e. the blocksize divided by the number of partitions minus the predictor order. In all other partitions the number of residual samples is equal to (blocksize >> partition order).

The partition order MUST be so that the blocksize is evenly divisible by the number of partitions. This means for example that for all odd blocksizes, only partition order 0 is allowed. The partition order also MUST be so that the (blocksize >> partition order) is larger than the predictor order. This means for example that with a blocksize of 4096 and a predictor order of 4, partition order cannot be larger than 9.

In case the coded residual of a subframe is one with a 4-bit Rice parameter (see table at the start of this section), the first 4 bits of each partition are either a rice parameter or an escape code. These 4 bits indicate an escape code if they are 0b1111, otherwise they contain the rice parameter as an unsigned number. In case the coded residual of the current subframe is one with a 5-bit Rice parameter, the first 5 bits indicate an escape code if they are 0b11111, otherwise they contain the rice parameter as an unsigned number as well.

In case an escape code was used, the partition does not contain a variable-length rice coded residual, but a fixed-length unencoded residual. Directly following the escape code are 5 bits containing the number of bits with which each residual sample is stored, as an unsigned number. The residual samples themselves are stored signed two's complement.

In case a rice parameter was provided, the partition contains a rice coded residual. The residual samples, which are signed numbers, are represented by unsigned numbers in the rice code. For positive numbers, the representation is the number doubled, for negative numbers, the representation is the number multiplied by -2 and has 1 subtracted. This representation of signed numbers is also known as zigzag encoding and the zigzag encoded residual is called the folded residual. The folded residual samples are then each divided by the rice parameter. The result of each division rounded down (the quotient) is stored unary, the remainder is stored binary.

Decoding the coded residual thus involves selecting the right coding method, finding the number of partitions, reading unary and binary parts of each codeword one-by-one and keeping track of when a new partition starts and thus when a new rice parameter needs to be read.

10.3. Frame footer

Following the last subframe is the frame footer. If the last subframe is not byte aligned (i.e. the bits required to store all subframes put together are not divisible by 8), zero bits are added until byte alignment is reached. Following this is a 16-bit CRC, initialized with 0, with polynomial x^16 + x^15 + x^2 + x^0. This CRC covers the whole frame excluding the 16-bit CRC, including the sync code.

11. Implementation status

This section records the status of known implementations of the FLAC format, and is based on a proposal described in [RFC7942]. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

A reference encoder and decoder implementation of the FLAC format exists, known as libFLAC, maintained by Xiph.Org. It can be found at https://xiph.org/flac/ Note that while all libFLAC components are licensed under 3-clause BSD, the flac and metaflac command line tools often supplied together with libFLAC are licensed under GPL.

Another completely independent implementation of both encoder and decoder of the FLAC format is available in libavcodec, maintained by FFmpeg, licensed under LGPL 2.1 or later. It can be found at https://ffmpeg.org/

A list of other implementations and an overview of which parts of the format they implement can be found here: https://github.com/ietf-wg-cellar/flac-specification/wiki/Implementations

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 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

[HPL-1999-144] Hans, M. and RW. Schafer, "Lossless Compression of Digital Audio", DOI 10.1109/79.939834, November 1999, <https://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf>.

[IEC.60908.1999] International Electrotechnical Commission, "Audio recording - Compact disc digital audio system", IEC International standard 60908 second edition, 1999.

[ISRC-handbook] "International Standard Recording Code (ISRC) Handbook, 4th edition", 2021, <https://www.ifpi.org/isrc_handbook/>.

[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>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[robinson-tr156] Robinson, T., "SHORTEN: Simple lossless and near-lossless waveform compression", December 1994, <https://mi.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html>.

Appendix A. Numerical considerations

In order to maintain lossless behavior, all arithmetic used in encoding and decoding sample values MUST be done with integer data types to eliminate the possibility of introducing rounding errors associated with floating-point arithmetic. Use of floating-point representations in analysis (e.g. finding a good predictor or rice parameter) is not a concern, as long as the process of using the found predictor and rice parameter to encode audio samples is implemented with only integer math.

Furthermore, the possibility of integer overflow MUST be eliminated by using data types large enough to never overflow. Choosing a 64-bit signed data type for all arithmetic involving sample values would make sure the possibility for overflow is eliminated, but usually smaller data types are chosen for increased performance, especially in embedded devices. This section will provide guidelines for choosing the right data type in each step of encoding and decoding FLAC files.

A.1. Determining necessary data type size

To find the smallest data type size that is guaranteed not to overflow for a certain sequence of arithmetic operations, the combination of values producing the largest possible result should be considered.

If for example two 16-bit signed integers are added, the largest possible result forms if both values are the largest number that can be represented with a 16-bit signed integer. To store the result, an signed integer data type with at least 17 bits is needed. Similarly, when adding 4 of these values, 18 bits are needed, when adding 8, 19 bits are needed etc. In general, the number of bits necessary when adding numbers together is increased by the log base 2 of the number of values rounded up to the nearest integer. So, when adding 18 unknown values stored in 8 bit signed integers, we need a signed integer data type of at least 13 bits to store the result, as the log base 2 of 18 rounded up is 5.

In case of multiplication, the number of bits needed for the result is the size of the first variable plus the size of the second variable, but counting only one sign bit if working with signed data types. If for example a 16-bit signed integer is multiplied by a 16-bit signed integer, the result needs at least 31 bits to store without overflowing.

A.2. Stereo decorrelation

When stereo decorrelation is used, the side channel will have one extra bit of bit depth, see section on Interchannel Decorrelation.

This means that while 16-bit signed integers have sufficient range to store samples from a fully decoded FLAC frame with a bit depth of 16 bit, the decoding of a side subframe in such a file will need a data type with at least 17 bit to store decoded subframe samples before undoing stereo decorrelation.

Most FLAC decoders store decoded (subframe) samples as 32-bit values, which is sufficient for files with bit depths up to (and including) 31 bit.

A.3. Prediction

A prediction (which is used to calculate the residual on encoding or added to the residual to calculate the sample value on decoding) is formed by multiplying and summing preceding sample values. In order to eliminate the possibility of integer overflow, the combination of preceding sample values and predictor coefficients producing the largest possible value should be considered.

To determine the size of the data type needed to calculate either a residual sample (on encoding) or an audio sample value (on decoding) in a fixed predictor subframe, the maximal possible value for these is calculated as described in the previous subsection in the following table. For example: if a frame codes for 16-bit audio and has some form of stereo decorrelation, the subframe coding for the side channel would need 16+1+3 bits in case a third order fixed predictor is used.

Order	Calculation of residual	Sample values summed	Extra bits
0	s(n)	1	0
1	s(n) - s(n-1)	2	1
2	s(n) - 2 * s(n-1) + s(n-2)	4	2
3	s(n) - 3 * s(n-1) + 3 * s(n-2) - s(n-3)	8	3
4	s(n) - 4 * s(n-1) + 6 * s(n-2) - 4 * s(n-3) + s(n-4)	16	4

Table 21

Where

- n is the number of the sample being predicted
- s(n) is the sample being predicted
- s(n-1) is the sample before the one being predicted, s(n-2) is the sample before that etc.

For subframes with a linear predictor, calculation is a little more complicated. Each prediction is a sum of several multiplications. Each of these multiply a sample value with a predictor coefficient. The extra bits needed can be calculated by adding the predictor coefficient

precision (in bits) to the bit depth of the audio samples. As both are signed numbers and only one 'sign bit' is necessary, 1 bit can be subtracted. To account for the summing of these multiplications, the log base 2 of the predictor order rounded up is added.

For example, if the sample bitdepth of the source is 24, the current subframe encodes a side channel (see the section on interchannel decorrelation), the predictor order is 12 and the predictor coefficient precision is 15 bits, the minimum required size of the used signed integer data type is at least (24 + 1) + (15 - 1) + ceil(log2(12)) = 43 bits. As another example, with a side-channel subframe bit depth of 16, a predictor order of 8 and a predictor coefficient precision of 12 bits, the minimum required size of the used signed integer data type is (16 + 1) + (12 - 1) + ceil(log2(8)) = 31 bits.

After the prediction has been shifted right, the number of bits needed is reduced by the amount of right shift and increased by one bit for the subtraction from the current sample on encoding. On decoding, the data type size needed to store the result of the addition of the residual and the prediction should fit the subframe bit depth, assuming all calculations were done correctly.

Taking the last example where 31 bits were needed for the prediction, the required data type size for the residual samples in case of a right shift of 10 bits would be 31 - 10 + 1 = 22 bits.

A.4. Rice coding

When folding (i.e. zig-zag encoding) the residual sample values, no extra bits are needed when the absolute value of each residual sample is first stored in an unsigned data type of the size of the last step, then doubled and then has one subtracted depending on whether the residual sample was positive or negative. Many implementations however choose to require one extra bit of data type size so zig-zag encoding can happen in one step and without a cast instead of the procedure described in the previous sentence.

Appendix B. Examples

This informational appendix contains short example FLAC files and short parts of FLAC files which are decoded step by step. These examples provide a more engaging way to understand the FLAC format than the formal specification. The text explaining these examples assumes the reader has at least cursory read the specification and that the reader refers to the specification for explanation of the terminology used. These examples mostly focus on the lay-out of several metadata blocks and subframe types and the implications of certain aspects (for example wasted bits and stereo decorrelation) on this lay-out.

The examples feature (parts of) files generated by various FLAC encoders. These are presented in hexadecimal or binary format, followed by tables and text referring to various features by their starting bit positions in these representations. Each starting position (shortened to 'start' in the tables) is a hexadecimal byte position and a start bit within that byte, separated by a plus sign. Counts for these start at zero. For example, a feature starting at the 3rd bit of the 17th byte is referred to as starting at 0x10+2.

All data in this appendix has been thoroughly verified. However, as this appendix is informational, in case any information here conflicts with statements in the formal specification, the latter takes precedence.

B.1. Decoding example 1

This very short example FLAC file codes for PCM audio that has two channels, each containing 1 sample. The focus of this example is on the essential parts of a FLAC file.

B.1.1. Example file 1 in hexadecimal representation

```
00000000: 664c 6143 8000 0022 1000 1000  fLaC..."....
0000000c: 0000 0f00 000f 0ac4 42f0 0000  ........B...
00000018: 0001 3e84 b418 07dc 6903 0758  ..>.....i..X
00000024: 6a3d ad1a 2e0f fff8 6918 0000  j=......i...
00000030: bf03 58fd 0312 8baa 9a         ..X......
```

B.1.2. Example file 1 in binary representation

```
00000000: 01100110 01001100 01100001 01000011  fLaC
00000004: 10000000 00000000 00000000 00100010  ..."
00000008: 00010000 00000000 00010000 00000000  ....
0000000c: 00000000 00000000 00001111 00000000  ....
00000010: 00000000 00001111 00001010 11000100  ....
00000014: 01000010 11110000 00000000 00000000  B...
00000018: 00000000 00000001 00111110 10000100  ..>.
0000001c: 10110100 00011000 00000111 11011100  ....
00000020: 01101001 00000011 00000111 01011000  i..X
00000024: 01101010 00111101 10101101 00011010  j=..
00000028: 00101110 00001111 11111111 11111000  ....
0000002c: 01101001 00011000 00000000 00000000  i...
00000030: 10111111 00000011 01011000 11111101  ..X.
00000034: 00000011 00010010 10001011 10101010  ....
00000038: 10011010
```

B.1.3. Signature and streaminfo

The first 4 bytes of the file contain the fLaC file signature. Directly following it is a metadata block. The signature and the first metadata block header are broken down in the following table

Start	Length	Contents	Description
0x00+0	4 byte	0x664C6143	fLaC
0x04+0	1 bit	0b1	Last metadata block
0x04+1	7 bit	0b0000000	Streaminfo metadata block
0x05+0	3 byte	0x000022	Length 34 byte

Table 22

As the header indicates that this is the last metadata block, the position of the first audio frame can now be calculated as the position of the first byte after the metadata block header + the length of the block, i.e. 8+34 = 42 or 0x2a. As can be seen 0x2a indeed contains the frame sync code for fixed blocksize streams, 0xfff8.

The streaminfo metadata block contents are broken down in the following table

Start	Length	Contents	Description
0x08+0	2 byte	0x1000	Min. blocksize 4096
0x0a+0	2 byte	0x1000	Max. blocksize 4096
0x0c+0	3 byte	0x00000f	Min. frame size 15 byte
0x0f+0	3 byte	0x00000f	Max. frame size 15 byte
0x12+0	20 bit	0x0ac4, 0b0100	Sample rate 44100 Hertz
0x14+4	3 bit	0b001	2 channels
0x14+7	5 bit	0b01111	Sample bit depth 16
0x15+4	36 bit	0b0000, 0x00000001	Total no. of samples 1
0x1a	16 byte	(...)	MD5 signature

Table 23

The minimum and maximum blocksize are both 4096. This was apparently the blocksize the encoder planned to use, but as only 1 interchannel sample was provided, no frames with 4096 samples are actually present in this file.

Note that anywhere a number of samples is mentioned (blocksize, total number of samples, sample rate), interchannel samples are meant.

The MD5 sum (starting at 0x1a) is 0x3e84 b418 07dc 6903 0758 6a3d ad1a 2e0f. This will be validated after decoding the samples.

B.1.4. Audio frames

The frame header starts at position 0x2a and is broken down in the following table.

Start	Length	Contents	Description
0x2a+0	15 bit	0xff, 0b1111100	frame sync
0x2b+7	1 bit	0b0	blocksize strategy
0x2c+0	4 bit	0b0110	8-bit blocksize further down
0x2c+4	4 bit	0b1001	sample rate 44.1kHz
0x2d+0	4 bit	0b0001	stereo, no decorrelation
0x2d+4	3 bit	0b100	bit depth 16 bit
0x2d+7	1 bit	0b0	mandatory 0 bit
0x2e+0	1 byte	0x00	frame number 0

Start	Length	Contents	Description
0x2f+0	1 byte	0x00	blocksize 1
0x30+0	1 byte	0xbf	frame header CRC

Table 24

As the stream is a fixed blocksize stream, the number at 0x2e contains a frame number. As the value is smaller than 128, only 1 byte is used for the encoding.

At byte 0x31 the subframe header of the first subframe starts, it is broken down in the following table.

Start	Length	Contents	Description
0x31+0	1 bit	0b0	mandatory 0 bit
0x31+1	6 bit	0b000001	verbatim subframe
0x31+7	1 bit	0b1	wasted bits present
0x32+0	2 bit	0b01	2 wasted bits
0x32+2	14 bit	0b011000, 0xfd	14-bit unencoded sample

Table 25

As the wasted bits flag is 1 in this subframe, an unary coded number follows. Starting at 0x32, we see 0b01, which unary codes for 1, meaning we have 2 wasted bits in this subframe.

As this is a verbatim subframe, the subframe only contains unencoded sample values. With a blocksize of 1, it contains only a single sample. The bit depth of the audio is 16 bit, but as the subframe header signals 2 wasted bits, only 14 bits are stored. As no stereo decorrelation is used, a bit depth increase for the side channel is not applicable. So, the next 14 bit (starting at position 0x32+2) contain the unencoded sample coded big-endian, signed two's complement. The value reads 0b011000 11111101, or 6397. This value needs to be shifted left by 2 bits, to account for the wasted bits. The value is then 0b011000 11111101 00, or 25588.

The second subframe starts at 0x34, it is broken down in the following table.

Start	Length	Contents	Description
0x34+0	1 bit	0b0	mandatory 0 bit
0x34+1	6 bit	0b000001	verbatim subframe
0x34+7	1 bit	0b1	wasted bits present
0x35+0	4 bit	0b0001	4 wasted bits
0x35+4	12 bit	0b0010, 0x8b	12-bit unencoded sample

Table 26

Here the wasted bits flag is also one, but the unary coded number that follows it is 4 bit long, indicating 4 wasted bits. This means the sample is stored in 12 bits. The sample value is 0b0010 10001011, or 651. This value now has to be shifted left by 4 bits, i.e. 0b0010 10001011 0000 or 10416.

At this point, we would do stereo decorrelation if that was applicable.

As the last subframe ends byte-aligned, no padding bits were inserted. The next 2 bytes, starting at 0x38, contain the frame CRC. As this is the only frame in the file, the file ends with the CRC.

To validate the MD5, we line up the samples interleaved, byte-aligned, little endian, signed two's complement. The first sample, the value of which was 25588 translates to 0xf463, the second sample had a value of 10416 which translates to 0xb028. When MD5 summing 0xf463b028, we get the MD5 sum found in the header, so decoding was lossless.

B.2. Decoding example 2

This FLAC file is larger than the first example, but still contains very little audio. The focus of this example is on decoding a subframe with a fixed predictor and a coded residual, but it also contains a very short seektable, vorbis comment and padding metadata block.

B.2.1. Example file 2 in hexadecimal representation

```
00000000: 664c 6143 0000 0022 0010 0010  fLaC..."....
0000000c: 0000 1700 0044 0ac4 42f0 0000  .....D..B...
00000018: 0013 d5b0 5649 75e9 8b8d 8b93  ....VIu.....
00000024: 0422 757b 8103 0300 0012 0000  ."u{........
00000030: 0000 0000 0000 0000 0000 0000  ............
0000003c: 0000 0010 0400 003a 2000 0000  .......: ...
00000048: 7265 6665 7265 6e63 6520 6c69  reference li
00000054: 6246 4c41 4320 312e 332e 3320  bFLAC 1.3.3
00000060: 3230 3139 3038 3034 0100 0000  20190804....
0000006c: 0e00 0000 5449 544c 453d d7a9  ....TITLE=..
00000078: d79c d795 d79d 8100 0006 0000  ............
00000084: 0000 0000 fff8 6998 000f 9912  ......i.....
00000090: 0867 0162 3d14 4299 8f5d f70d  .g.b=.B..]..
0000009c: 6fe0 0c17 caeb 2100 0ee7 a77a  o.....!....z
000000a8: 24a1 590c 1217 b603 097b 784f  $.Y......{xO
000000b4: aa9a 33d2 85e0 70ad 5b1b 4851  ..3...p.[.HQ
000000c0: b401 0d99 d2cd 1a68 f1e6 b810  .......h....
000000cc: fff8 6918 0102 a402 c382 c40b  ..i.........
000000d8: c14a 03ee 48dd 03b6 7c13 30    .J..H...|.0
```

B.2.2. Example file 2 in binary representation (only audio frames)

```
00000088: 11111111 11111000 01101001 10011000  ..i.
0000008c: 00000000 00001111 10011001 00010010  ....
00000090: 00001000 01100111 00000001 01100010  .g.b
00000094: 00111101 00010100 01000010 10011001  =.B.
00000098: 10001111 01011101 11110111 00001101  .]..
0000009c: 01101111 11100000 00001100 00010111  o...
000000a0: 11001010 11101011 00100001 00000000  ..!.
000000a4: 00001110 11100111 10100111 01111010  ...z
000000a8: 00100100 10100001 01011001 00001100  $.Y.
000000ac: 00010010 00010111 10110110 00000011  ....
000000b0: 00001001 01111011 01111000 01001111  .{xO
000000b4: 10101010 10011010 00110011 11010010  ..3.
000000b8: 10000101 11100000 01110000 10101101  ..p.
000000bc: 01011011 00011011 01001000 01010001  [.HQ
000000c0: 10110100 00000001 00001101 10011001  ....
000000c4: 11010010 11001101 00011010 01101000  ...h
000000c8: 11110001 11100110 10111000 00010000  ....
000000cc: 11111111 11111000 01101001 00011000  ..i.
000000d0: 00000001 00000010 10100100 00000010  ....
000000d4: 11000011 10000010 11000100 00001011  ....
000000d8: 11000001 01001010 00000011 11101110  .J..
000000dc: 01001000 11011101 00000011 10110110  H...
000000e0: 01111100 00010011 00110000           |.0
```

B.2.3. Signature and streaminfo

Most of the streaminfo block is the same as in example 1, so only parts that are different are listed in the following table

Start	Length	Contents	Description
0x04+0	1 bit	0b0	Not the last metadata block
0x08+0	2 byte	0x0010	Min. blocksize 16
0x0a+0	2 byte	0x0010	Max. blocksize 16
0x0c+0	3 byte	0x000017	Min. frame size 23 byte
0x0f+0	3 byte	0x000044	Max. frame size 68 byte
0x15+4	36 bit	0b0000, 0x00000013	Total no. of samples 19
0x1a	16 byte	(...)	MD5 signature

Table 27

This time, the minimum and maximum blocksizes are reflected in the file: there is one block of 16 samples, but the last block (which has 3 samples) is excluded from this number. The MD5 signature is 0xd5b0 5649 75e9 8b8d 8b93 0422 757b 8103, this will be verified at the end of this example.

B.2.4. Seektable

The seektable metadata block only holds one entry. It is not really useful here, as it points to the first frame, but it is enough for this example. The seektable metadata block is broken down in the following table.

Start	Length	Contents	Description
0x2a+0	1 bit	0b0	Not the last metadata block
0x2a+1	7 bit	0b0000011	Seektable metadata block
0x2b+0	3 byte	0x000012	Length 18 byte
0x2e+0	8 byte	0x0000000000000000	Seekpoint to sample 0
0x36+0	8 byte	0x0000000000000000	Seekpoint to offset 0
0x3e+0	2 byte	0x0010	Seekpoint to blocksize 16

Table 28

B.2.5. Vorbis comment

The vorbis comment metadata block contains the vendor string and a single comment. It is broken down in the following table.

Start	Length	Contents	Description
0x40+0	1 bit	0b0	Not the last metadata block
0x40+1	7 bit	0b0000100	Vorbis comment metadata block
0x41+0	3 byte	0x00003a	Length 58 byte
0x44+0	4 byte	0x20000000	Vendor string length 32 byte
0x48+0	32 byte	(...)	Vendor string
0x68+0	4 byte	0x01000000	Number of fields 1
0x6c+0	4 byte	0x0e000000	Field length 14 byte
0x70+0	14 byte	(...)	Field contents

Table 29

The vendor string is reference libFLAC 1.3.3 20190804, the field contents of the only field is TITLE=(U+05E9 U+05DC U+05D5 U+05DD) שלום. The vorbis comment field is 14 bytes but only 10 characters in size, because it contains four 2-byte characters.

B.2.6. Padding

The last metadata block is a (very short) padding block.

Start	Length	Contents	Description
0x7e+0	1 bit	0b1	Last metadata block
0x7e+1	7 bit	0b0000001	Padding metadata block
0x7f+0	3 byte	0x000006	Length 6 byte
0x82+0	6 byte	0x000000000000	Padding bytes

Table 30

B.2.7. First audio frame

The frame header starts at position 0x88 and is broken down in the following table.

Start	Length	Contents	Description
0x88+0	15 bit	0xff, 0b1111100	frame sync
0x89+7	1 bit	0b0	blocksize strategy
0x8a+0	4 bit	0b0110	8-bit blocksize further down
0x8a+4	4 bit	0b1001	sample rate 44.1kHz
0x8b+0	4 bit	0b1001	right-side stereo
0x8b+4	3 bit	0b100	bit depth 16 bit
0x8b+7	1 bit	0b0	mandatory 0 bit
0x8c+0	1 byte	0x00	frame number 0
0x8d+0	1 byte	0x0f	blocksize 16
0x8e+0	1 byte	0x99	frame header CRC

Table 31

The first subframe starts at byte 0x8f, it is broken down in the following table excluding the coded residual. As this subframe codes for a side channel, the bit depth is increased by 1 bit from 16 bit to 17 bit. This is most clearly present in the unencoded warm-up sample.

Start	Length	Contents	Description
0x8f+0	1 bit	0b0	mandatory 0 bit
0x8f+1	6 bit	0b001001	fixed subframe, 1st order
0x8f+7	1 bit	0b0	no wasted bits present
0x90+0	17 bit	0x0867, 0b0	unencoded warm-up sample

Table 32

The coded residual is broken down in the following table. All quotients are unary coded, all remainders are unencoded with a number of bits specified by the rice parameter.

Start	Length	Contents	Description
0x92+1	2 bit	0b00	Rice code with 4-bit parameter
0x92+3	4 bit	0b0000	Partition order 0
0x92+7	4 bit	0b1011	Rice parameter 11
0x93+3	4 bit	0b0001	Quotient 3
0x93+7	11 bit	0b00011110100	Remainder 244
0x95+2	2 bit	0b01	Quotient 1
0x95+4	11 bit	0b01000100001	Remainder 545
0x96+7	2 bit	0b01	Quotient 1
0x97+1	11 bit	0b00110011000	Remainder 408
0x98+4	1 bit	0b1	Quotient 0
0x98+5	11 bit	0b11101011101	Remainder 1885
0x9a+0	1 bit	0b1	Quotient 0
0x9a+1	11 bit	0b11101110000	Remainder 1904
0x9b+4	1 bit	0b1	Quotient 0
0x9b+5	11 bit	0b10101101111	Remainder 1391
0x9d+0	1 bit	0b1	Quotient 0
0x9d+1	11 bit	0b11000000000	Remainder 1536
0x9e+4	1 bit	0b1	Quotient 0
0x9e+5	11 bit	0b10000010111	Remainder 1047
0xa0+0	1 bit	0b1	Quotient 0
0xa0+1	11 bit	0b10010101110	Remainder 1198
0xa1+4	1 bit	0b1	Quotient 0
0xa1+5	11 bit	0b01100100001	Remainder 801
0xa3+0	13 bit	0b0000000000001	Quotient 12
0xa4+5	11 bit	0b11011100111	Remainder 1767

Start	Length	Contents	Description
0xa6+0	1 bit	0b1	Quotient 0
0xa6+1	11 bit	0b01001110111	Remainder 631
0xa7+4	1 bit	0b1	Quotient 0
0xa7+5	11 bit	0b01000100100	Remainder 548
0xa9+0	1 bit	0b1	Quotient 0
0xa9+1	11 bit	0b01000010101	Remainder 533
0xaa+4	1 bit	0b1	Quotient 0
0xaa+5	11 bit	0b00100001100	Remainder 268

Table 33

At this point, the decoder should know it is done decoding the coded residual, as it received 16 samples: 1 warm-up sample and 15 residual samples. Each residual sample can be calculated from the quotient and remainder, and undoing the zig-zag encoding. For example, the value of the first zig-zag encoded residual sample is 3 * 2^11 + 244 = 6388. As this is an even number, the zig-zag encoding is undone by dividing by 2, the residual sample value is 3194. This is done for all residual samples in the next table

Quotient	Remainder	Zig-zag encoded	Residual sample value
3	244	6388	3194
1	545	2593	-1297
1	408	2456	1228
0	1885	1885	-943
0	1904	1904	952
0	1391	1391	-696
0	1536	1536	768
0	1047	1047	-524
0	1198	1198	599
0	801	801	-401
12	1767	26343	-13172
0	631	631	-316
0	548	548	274

Quotient	Remainder	Zig-zag encoded	Residual sample value
0	533	533	-267
0	268	268	134

Table 34

It can be calculated that using a Rice code is in this case more efficient than storing values unencoded. The rice code (excluding the partition order and parameter) is 199 bits in length. The largest residual value (-13172) would need 15 bits to be stored unencoded, so storing all 15 samples with 15 bits results in a sequence with a length of 225 bits.

The next step is using the predictor and the residuals to restore the sample values. As this subframe uses a fixed predictor with order 1, this means adding the residual value to the value of the previous sample.

Residual	Sample value
(warm-up)	4302
3194	7496
-1297	6199
1228	7427
-943	6484
952	7436
-696	6740
768	7508
-524	6984
599	7583
-401	7182
-13172	-5990
-316	-6306
274	-6032
-267	-6299
134	-6165

Table 35

With this, decoding of the first subframe is complete. Decoding of the second subframe is very similar, as it also uses a fixed predictor of order 1, so this is left as an exercise for the reader, results are in the next table. The next step is stereo decorrelation, which is done in the following table. As the stereo decorrelation is right-side, in which the actual ordering of the subframes is side-right, the samples in the right channel come directly from the second subframe, while the samples in the left channel are found by adding the values of both subframes for each sample.

Subframe 1	Subframe 2	Left	Right
4302	6070	10372	6070
7496	10545	18041	10545
6199	8743	14942	8743
7427	10449	17876	10449
6484	9143	15627	9143
7436	10463	17899	10463
6740	9502	16242	9502
7508	10569	18077	10569
6984	9840	16824	9840
7583	10680	18263	10680
7182	10113	17295	10113
-5990	-8428	-14418	-8428
-6306	-8895	-15201	-8895
-6032	-8476	-14508	-8476
-6299	-8896	-15195	-8896
-6165	-8653	-14818	-8653

Table 36

As the second subframe ends byte-aligned, no padding bits follow it. Finally, the last 2 bytes in the frame is the frame CRC.

B.2.8. Second audio frame

The second audio frame is very similar to the frame decoded in the first example, but this time not 1 but 3 samples are present.

The frame header starts at position 0xcc and is broken down in the following table.

Start	Length	Contents	Description
0xcc+0	15 bit	0xff, 0b1111100	frame sync
0xcd+7	1 bit	0b0	blocksize strategy
0xce+0	4 bit	0b0110	8-bit blocksize further down
0xce+4	4 bit	0b1001	sample rate 44.1kHz
0xcf+0	4 bit	0b0001	stereo, no decorrelation
0xcf+4	3 bit	0b100	bit depth 16 bit
0xcf+7	1 bit	0b0	mandatory 0 bit
0xd0+0	1 byte	0x01	frame number 1
0xd1+0	1 byte	0x02	blocksize 3
0xd2+0	1 byte	0xa4	frame header CRC

Table 37

The first subframe starts at 0xd3+0 and is broken down in the following table.

Start	Length	Contents	Description
0xd3+0	1 bit	0b0	mandatory 0 bit
0xd3+1	6 bit	0b000001	verbatim subframe
0xd3+7	1 bit	0b0	no wasted bits present
0xd4+0	16 bit	0xc382	16-bit unencoded sample
0xd6+0	16 bit	0xc40b	16-bit unencoded sample
0xd8+0	16 bit	0xc14a	16-bit unencoded sample

Table 38

The second subframe starts at 0xda+0 and is broken down in the following table

Start	Length	Contents	Description
0xda+0	1 bit	0b0	mandatory 0 bit
0xda+1	6 bit	0b000001	verbatim subframe
0xda+7	1 bit	0b1	wasted bits present
0xdb+0	1 bit	0b1	1 wasted bit

Start	Length	Contents	Description
0xdb+1	15 bit	0b110111001001000	15-bit unencoded sample
0xdd+0	15 bit	0b110111010000001	15-bit unencoded sample
0xde+7	15 bit	0b110110110011111	15-bit unencoded sample

Table 39

As this subframe has wasted bits, the 15-bit unencoded samples need to be shifted left by 1 bit. For example, sample 1 is stored as -4536 and becomes -9072 after shifting left 1 bit.

As the last subframe does not end on byte alignment, 2 padding bits are added before the 2 byte frame CRC follows at 0xe1+0.

B.2.9. MD5 checksum verification

All samples in the file have been decoded, we can now verify the MD5 sum. All sample values must be interleaved and stored signed, coded little-endian. The result of this follows in groups of 12 samples (i.e. 6 interchannel samples)

```
0x8428 B617 7946 3129 5E3A 2722 D445 D128 0B3D B723 EB45 DF28
0x723f 1E25 9D46 4929 B841 7026 5747 B829 8F43 8127 AEC7 14DF
0x9FC4 41DD 54C7 E4DE A5C4 40DD 1EC6 33DE 82C3 90DC 0BC4 02DD
0x4AC1 3EDB
```

The MD5sum of this is indeed the same as the one found in the streaminfo metadata block.

B.3. Decoding example 3

This example is once again a very short FLAC file. The focus of this example is on decoding a subframe with a linear predictor and a coded residual with more than one partition.

B.3.1. Example file 3 in hexadecimal representation

```
00000000: 664c 6143 8000 0022 1000 1000  fLaC..."....
0000000c: 0000 1f00 001f 07d0 0070 0000  .........p..
00000018: 0018 f8f9 e396 f5cb cfc6 dc80  ............
00000024: 7f99 7790 6b32 fff8 6802 0017  ..w.k2..h...
00000030: e944 004f 6f31 3d10 47d2 27cb  .D.Oo1=.G.'.
0000003c: 6d09 0831 452b dc28 2222 8057  m..1E+.("".W
00000048: a3                             .
```

B.3.2. Example file 3 in binary representation (only audio frame)

```
0000002a: 11111111 11111000 01101000 00000010  ..h.
0000002e: 00000000 00010111 11101001 01000100  ...D
00000032: 00000000 01001111 01101111 00110001  .Oo1
00000036: 00111101 00010000 01000111 11010010  =.G.
0000003a: 00100111 11001011 01101101 00001001  '.m.
0000003e: 00001000 00110001 01000101 00101011  .1E+
00000042: 11011100 00101000 00100010 00100010  .("" 
00000046: 10000000 01010111 10100011           .W.
```

B.3.3. Signature and streaminfo

Most of the streaminfo block is the same as in example 1, so only parts that are different are listed in the following table

Start	Length	Contents	Description
0x0c+0	3 byte	0x00001f	Min. frame size 31 byte
0x0f+0	3 byte	0x00001f	Max. frame size 31 byte
0x12+0	20 bit	0x07d0, 0x0000	Sample rate 32000 Hertz
0x14+4	3 bit	0b000	1 channel
0x14+7	5 bit	0b00111	Sample bit depth 8 bit
0x15+4	36 bit	0b0000, 0x00000018	Total no. of samples 24
0x1a	16 byte	(...)	MD5 signature

Table 40

B.3.4. Audio frame

The frame header starts at position 0x2a and is broken down in the following table.

Start	Length	Contents	Description
0x2a+0	15 bit	0xff, 0b1111100	Frame sync
0x2b+7	1 bit	0b0	Blocksize strategy
0x2c+0	4 bit	0b0110	8-bit blocksize further down
0x2c+4	4 bit	0b1000	Sample rate 32kHz
0x2d+0	4 bit	0b0000	Mono audio (1 channel)
0x2d+4	3 bit	0b001	Bit depth 8 bit
0x2d+7	1 bit	0b0	Mandatory 0 bit

Start	Length	Contents	Description
0x2e+0	1 byte	0x00	Frame number 0
0x2f+0	1 byte	0x17	Blocksize 24
0x30+0	1 byte	0xe9	Frame header CRC

Table 41

The first and only subframe starts at byte 0x31, it is broken down in the following table, without the coded residual.

Start	Length	Contents	Description
0x31+0	1 bit	0b0	Mandatory 0 bit
0x31+1	6 bit	0b100010	Linear prediction subframe, 3rd order
0x31+7	1 bit	0b0	No wasted bits present
0x32+0	8 bit	0x00	Unencoded warm-up sample 0
0x33+0	8 bit	0x4f	Unencoded warm-up sample 79
0x34+0	8 bit	0x6f	Unencoded warm-up sample 111
0x35+0	4 bit	0b0011	Coefficient precision 4 bit
0x35+4	5 bit	0b00010	Prediction right shift 2
0x36+1	4 bit	0b0111	Predictor coefficient 7
0x36+5	4 bit	0b1010	Predictor coefficient -6
0x37+1	4 bit	0b0010	Predictor coefficient 2

Table 42

The data stream continues with the coded residual, which is broken down in the following table. Residual partition 3 and 4 are left as an exercise for the reader.

Start	Length	Contents	Description
0x37+5	2 bit	0b00	Rice-coded residual, 4-bit parameter
0x37+7	4 bit	0b0010	Partition order 2
0x38+3	4 bit	0b0011	Rice parameter 3
0x38+7	1 bit	0b1	Quotient 0
0x39+0	3 bit	0b110	Remainder 6

Start	Length	Contents	Description
0x39+3	1 bit	0b1	Quotient 0
0x39+4	3 bit	0b001	Remainder 1
0x39+7	4 bit	0b0001	Quotient 3
0x3a+3	3 bit	0b001	Remainder 1
0x3a+6	4 bit	0b1111	No rice parameter, escape code
0x3b+2	5 bit	0b00101	Partition encoded with 5 bits
0x3b+7	5 bit	0b10110	Residual -10
0x3c+4	5 bit	0b11010	Residual -6
0x3d+1	5 bit	0b00010	Residual 2
0x3d+6	5 bit	0b01000	Residual 8
0x3e+3	5 bit	0b01000	Residual 8
0x3f+0	5 bit	0b00110	Residual 6
0x3f+5	4 bit	0b0010	Rice parameter 2
0x40+1	22 bit	(...)	Residual partition 3
0x42+7	4 bit	0b0001	Rice parameter 1
0x43+3	23 bit	(...)	Residual partition 4

Table 43

The frame ends with 6 padding bits and a 2 byte frame CRC

To decode this subframe, 21 predictions have to be calculated and added to their corresponding residuals. This is a sequential process: as each prediction uses previous samples, it is not possible to start this decoding halfway a subframe or decode a subframe with parallel threads.

Residual	Predictor w/o shift	Predictor	Sample value
(warm-up)	N/A	N/A	0
(warm-up)	N/A	N/A	79
(warm-up)	N/A	N/A	111
3	303	75	78
-1	38	9	8

Residual	Predictor w/o shift	Predictor	Sample value
-13	-190	-48	-61
-10	-319	-80	-90
-6	-248	-62	-68
2	-58	-15	-13
8	137	34	42
8	236	59	67
6	191	47	53
0	53	13	13
-3	-93	-24	-27
-5	-161	-41	-46
-4	-134	-34	-38
-1	-44	-11	-12
1	52	13	14
1	94	23	24
4	60	15	19
2	17	4	6
2	-24	-6	-4
2	-26	-7	-5
0	1	0	0

Table 44

Lining all these samples up, we get the following input for the MD5 summing process.

```
0x004F 6F4E 08C3 A6BC F32A 4335 0DE5 D2DA F40E 1813 06FC FB00
```

Which indeed results in the MD5 signature found in the streaminfo metadata block.

Authors' Addresses

Martijn van Beurden
Netherlands
Email: mvanb1@gmail.com

Andrew Weaver
Email: theandrewjw@gmail.com