Workgroup: cellar
Internet-Draft: draft-ietf-cellar-flac-14
Published: 14 January 2024
Intended Status: Standards Track
Expires: 17 July 2024
Authors: M.Q.C. van Beurden A. Weaver

Free Lossless Audio Codec

Abstract

This document defines the Free Lossless Audio Codec (FLAC) format and its streamable subset. FLAC is designed to reduce the amount of computer storage space needed to store digital audio signals without losing information in doing so (i.e., lossless). FLAC is free in the sense that its specification is open and its reference implementation is open-source. 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 17 July 2024.

Copyright Notice

Table of Contents

1. Introduction

This document defines the FLAC format and its streamable subset. FLAC files and streams can code for pulse-code modulated (PCM) audio with 1 to 8 channels, sample rates from 1 up to 1048575 hertz and bit depths from 4 up to 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 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 for compressing audio that is not PCM.

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. Where necessary n is expressed as an equation using * (multiplication), / (division), + (addition), or - (subtraction). An inclusive range of the number of bits expressed is represented with an ellipsis, such as u(m...n).

While the FLAC format can store digital audio as well as other digital signals, this document uses terminology specific to digital audio. The use of more generic terminology was deemed less clear, so a reader interested in non-audio use of the FLAC format is expected to make the translation from audio-specific terms to more generic terminology.

3. 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 one 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, one or more subframes, and a frame footer. It encodes the contents of a corresponding block.

* **Subframe**: An encoded subblock. All subframes within a frame code for the same number of samples. When interchannel decorrelation is used, a subframe can correspond to either the (per-sample) average of two subblocks or the (per-sample) difference between two subblocks, instead of to a subblock directly, see Section 4.2.

* **Interchannel samples**: A sample count that applies to all channels. For example, one second of 44.1 kHz audio has 44100 interchannel samples, meaning each channel has that number of samples.

* **Block size**: The number of interchannel samples contained in a block or coded in a frame.

* **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. (See Section 4.2 for details on 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 (see [LinearPrediction]). 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 (see [FIR]).

* **Muxing**: short for multiplexing, combining several streams or files into a single stream or file. In the context of this document, muxing more specifically refers to embedding a FLAC stream in a container as described in Section 10.

* **Fixed predictor**: a linear predictor in which the model parameters are the same across all FLAC files, and thus do 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 (see [VarLengthCode]) that 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.

4. Conceptual overview

Similar to many other 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 4.1). The input is split up into many contiguous blocks.

* **Interchannel Decorrelation** (see Section 4.2). In the case of stereo streams, the FLAC format allows for transforming the left-right signal into a mid-side signal, a left-side signal or a side-right signal to remove redundancy between channels. Choosing between any of these transformations is done independently for each block.

* **Prediction** (see Section 4.3). 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 on 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 to be included in the compressed stream. If no usable predictor can be found for a certain subblock, the signal is stored uncompressed and the next stage is skipped.

* **Residual Coding** (see Section 4.4). 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) is 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 8), which allows arbitrary information about the stream to be included at the beginning of the stream.

4.1. Blocking

The block size used for audio data has a direct effect on the compression ratio. If the block size is too small, the resulting large number of frames means that a disproportionate amount of bytes will be spent 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, except for the last block, and a maximum block size of 65535 samples. The last block is allowed to be smaller than 16 samples to be able to match the length of the encoded audio without using padding.

While the block size does not have to be constant 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 block size throughout the stream, and stores a block number instead of a sample number to slightly improve compression if a stream has a constant block size.

4.2. 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, all mid channel samples have to be shifted left by 1 bit. Also, if a side channel sample is odd, 1 has to be added to the corresponding mid channel sample after it has been shifted left by one bit. 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 subblocks 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 in the the left subframe.

* **Side-right**. The left and right subblocks are used to code a side subframe and the right subblock is coded. 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 lossy behavior, because an odd sample in the mid subframe must always be accompanied by a corresponding odd 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.

4.3. Prediction

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

1. **Verbatim**. Samples are stored directly, without any modeling. 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'), this 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, and 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 Section 9.2.5.

4. **Linear predictor**. Samples are predicted using past samples and a set of predictor coefficients, and 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 Section 9.2.6.

A FLAC encoder is free to select any of the above methods to model the input. However, to ensure lossless coding, the following exceptions apply:

- When the samples that need to be stored do not all have the same value (i.e., the signal is not constant), a constant subframe cannot be used.
- When an encoder is unable to find a fixed or linear predictor for which all residual samples are representable in 32-bit signed integers as stated in Section 9.2.7, a verbatim subframe is used.

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

4.4. Residual Coding

If a subframe uses a predictor to approximate the audio signal, a residual is 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 that 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 into two parts by dividing it by 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 a suitable Rice parameter is chosen, this form of coding, with a so-called variable-length code, uses fewer bits than the residual in unencoded form.

As Rice codes can only handle unsigned numbers, signed numbers are zigzag encoded to a so-called folded residual. See Section 9.2.7 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.

5. 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 when encountering data encoded using methods they do not recognize. Apart from reserved patterns, the format specifies forbidden patterns in certain places, meaning that the patterns MUST NOT appear in any bitstream. They are listed in the following table.

Description	Reference
Metadata block type 127	Section 8.1
Minimum and maximum block sizes smaller than 16 in streaminfo metadata block	Section 8.2
Sample rate bits 0b1111	Section 9.1.2
Uncommon blocksize 65536	Section 9.1.6
Predictor coefficient precision bits 0b1111	Section 9.2.6
Negative predictor right shift	Section 9.2.6

Table 1

All numbers used in a FLAC bitstream are integers, there are no floating-point representations. All numbers are big-endian coded, except the field lengths used in Vorbis comments (see Section 8.6), which are little-endian coded. This exception for Vorbis comments is to keep as much commonality as possible with Vorbis comments as used by the Vorbis codec (see [Vorbis]). All numbers are unsigned except linear predictor coefficients, the linear prediction shift (see Section 9.2.6), and numbers that directly represent samples, which are signed. None of these restrictions apply to application metadata blocks or to Vorbis comment field contents.

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, converting sample representations by first copying the number to a signed integer with sufficient range and then subtracting half of the range of the unsigned integer type, results in a signal with samples centered around 0.

Unary coding in a FLAC bitstream is done with zero bits terminated with a one bit, e.g., the number 5 is coded unary as 0b000001. This prevents the frame sync code from appearing in unary coded numbers.

When a FLAC file contains data that is forbidden or otherwise not valid, decoder behavior is left unspecified. A decoder MAY choose to stop decoding upon encountering such data. Examples of such data are

- One or more decoded sample values exceed the range offered by the bit depth as coded for that frame. E.g., in a frame with a bit depth of 8 bits, any samples not in the inclusive range from -128 to 127 are not valid.
- The number of wasted bits (see Section 9.2.2) used by a subframe is such that the bit depth of that subframe (see Section 9.2.3 for a description of subframe bit depth) equals zero or is negative.
- A frame header CRC (see Section 9.1.8) or frame footer CRC (see Section 9.3) does not validate.
- One of the forbidden bit patterns described in Table 1 above is used.

6. Format layout overview

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, and then the audio frames.

FLAC supports 127 kinds of metadata blocks; currently, 7 kinds are defined in Section 8.

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 and number of channels), and an 8-bit CRC. The frame header also contains either the sample number of the first sample in the frame (for variable block size streams), or the frame number (for fixed block size streams). This allows for fast, sample-accurate seeking to be performed. Following the frame header are encoded subframes, one for each channel. The frame is then zero-padded to a byte boundary and finished with a frame footer containing a checksum for the frame. Each subframe has its own header that specifies how the subframe is encoded.

In order to allow a decoder to start decoding at any place in the stream, each frame starts with a byte-aligned 15-bit sync code. However, since it is not guaranteed that the sync code does not appear elsewhere in the frame, the decoder can check that it synced correctly by parsing the rest of the frame header and validating the frame header CRC.

Furthermore, to allow a decoder to start decoding at any place in the stream even without having received a streaminfo metadata block,

each frame header contains some basic information about the stream. This information includes sample rate, bits per sample, number of channels, etc. Since the frame header is 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 properties. When a certain property has a value that is not covered by the lookup table, the decoder is directed to find the value of that property (for example, the sample rate) at the end of the frame header or in the streaminfo metadata block. If a frame header refers to the streaminfo metadata block, the file is not 'streamable', see Section 7 for details. By using lookup tables, the file is streamable and the frame header size small for 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. Each subframe header is followed by the encoded audio data for that channel.

7. Streamable subset

The FLAC format specifies a subset of itself as the FLAC streamable subset. The purpose of this is to ensure that any streams encoded according to this 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 in such a way that decoder buffer sizes and other resource requirements can be easily determined. The streamable subset makes the following limitations on what MAY be used in the stream:

- The sample rate bits (see Section 9.1.2) in the frame header MUST be 0b0001-0b1110, i.e., the frame header MUST NOT refer to the streaminfo metadata block to describe the sample rate.
- The bit depth bits (see Section 9.1.4) in the frame header MUST be 0b001-0b111, i.e., the frame header MUST NOT refer to the streaminfo metadata block to describe the bit depth.
- The stream MUST NOT contain blocks with more than 16384 interchannel samples, i.e., the maximum block size must not be larger than 16384.
- Audio with a sample rate less than or equal to 48000 Hz MUST NOT be contained in blocks with more than 4608 interchannel samples, i.e., the maximum block size used for this audio must not be larger than 4608.

*Linear prediction subframes (see Section 9.2.6) containing audio with a sample rate less than or equal to 48000 Hz MUST have a predictor order less than or equal to 12, i.e., the subframe type bits in the subframe header (see Section 9.2.1) MUST NOT be 0b101100-0b111111.
*The Rice partition order (see Section 9.2.7) MUST be less than or equal to 8.
*The channel ordering MUST be equal to one defined in Section 9.1.3, i.e., the FLAC file MUST NOT need a WAVEFORMATEXTENSIBLE_CHANNEL_MASK tag to describe the channel ordering. See Section 8.6.2 for details.

8. 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.

8.1. 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 forbidden. 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	forbidden, to avoid confusion with a frame sync code

Table 2

8.2. Streaminfo

The streaminfo metadata 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. There MUST be no more than one streaminfo metadata block per FLAC stream.

If the streaminfo metadata block contains incorrect or incomplete information, decoder behavior is left unspecified (i.e., up to the decoder implementation). A decoder MAY choose to stop further decoding when the information supplied by the streaminfo metadata block turns out to be incorrect or contains forbidden values. 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 Section 12.

The following table describes the streaminfo metadata block, excluding the metadata block header.

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.
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.
u(36)	Total number of interchannel samples in the stream. A value of zero here means the number of total samples is unknown.
u(128)	MD5 checksum of the unencoded audio data. This allows the decoder to determine if an error exists in the audio data even when, despite the error, the bitstream itself is valid. A value of 0 signifies that the value is not known.

Table 3

The minimum block size and the maximum block size MUST be in the 16-65535 range. The minimum block size MUST be equal to or less than the maximum block size.

Any frame but the last one MUST have a block size equal to or greater than the minimum block size and MUST have a block size equal to or lesser than the maximum block size. The last frame MUST have a block size equal to or lesser than the maximum block size, it does not have to comply to the minimum block size because the block size of that frame must be able to accommodate the length of the audio data the stream contains.

If the minimum block size is equal to the maximum block size, the file contains a fixed block size stream, as the minimum block size excludes the last block. Note that in the case of a stream with a variable block size, the actual maximum block size MAY be smaller than the maximum block size listed in the streaminfo block, and the actual smallest block size excluding the last block MAY 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.

The sample rate MUST NOT be 0 when the FLAC file contains audio. A sample rate of 0 MAY be used when non-audio is represented. This is useful if data is encoded that is not along a time axis, or when the sample rate of the data lies outside the range that FLAC can represent in the streaminfo metadata block. If a sample rate of 0 is used it is recommended to store the meaning of the encoded content in a Vorbis comment field (see Section 8.6) or an application metadata block (see Section 8.4). This document does not define such metadata.

The MD5 checksum is computed by applying the MD5 message-digest algorithm in [RFC1321]. The message to this algorithm consists of all 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 computing the checksum, all samples must be byte-aligned. If the bit depth is not a whole number of bytes, the value of each sample is sign extended to the next whole number of bytes.

So, in the case of a 2-channel stream with 6-bit samples, bits will be lined up as follows.

```
SSAAAAAASSBBBBBBSSCCCCCC
^   ^   ^   ^   ^   ^
|   |   |   |   |  Bits of 2nd sample of 1st channel
|   |   |   |  Sign extension bits of 2nd sample of 2nd channel
|   |   |  Bits of 1st sample of 2nd channel
|   |  Sign extension bits of 1st sample of 2nd channel
|  Bits of 1st sample of 1st channel
Sign extention bits of 1st sample of 1st channel
```

As another example, in the case of a 1-channel with 12-bit samples, bits are lined up as follows, showing the little-endian byte order

```
AAAAAAAASSSSAAAABBBBBBBBSSSSBBBB
   ^     ^   ^   ^       ^   ^
   |     |   |   |       |  Most-significant 4 bits of 2nd sample
   |     |   |   | Sign extension bits of 2nd sample
   |     |   |  Least-significant 8 bits of 2nd sample
   |     |  Most-significant 4 bits of 1st sample
   |    Sign extension bits of 1st sample
  Least-significant 8 bits of 1st sample
```

8.3. Padding

The padding metadata block allows for an arbitrary amount of padding. 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 existing file (which would normally require rewriting the entire file). There MAY be one or more padding metadata blocks per FLAC stream.

Data	Description
u(n)	n '0' bits (n MUST be a multiple of 8, i.e., a whole number of bytes, and MAY be zero). n is 8 times the size described in the metadata block header.

Table 4

8.4. Application

The application metadata block is for use by third-party applications. The only mandatory field is a 32-bit identifier. An ID registry is being maintained at https://xiph.org/flac/id.html.

Data	Description
u(32)	Registered application ID.
u(n)	Application data (n MUST be a multiple of 8, i.e., a whole number of bytes) n is 8 times the size described in the metadata block header, minus the 32 bits already used for the application ID.

Table 5

Application IDs are registered with the IANA, see Section 13.2.

8.5. Seektable

The seektable metadata block can be used to store 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. There MUST NOT be more than one seektable metadata block in a stream, but the table can have any number of seek points.

Each seek point takes 18 bytes, so a seek table with 1% resolution within a stream adds less than 2 kilobyte of data. The number of seek points is implied by the size described in the metadata block header, i.e., equal to size / 18. There is also a special 'placeholder' seekpoint that will be ignored by decoders but can be used to reserve space for future seek point insertion.

Data	Description
Seekpoints	Zero or more seek points as defined in Section 8.5.1.

Table 6

A seektable is generally not usable for seeking in a FLAC file embedded in a container (see Section 10), as such containers usually interleave FLAC data with other data and the offsets used in seekpoints are those of an unmuxed FLAC stream. Also, containers often provide their own seeking methods. It is, however, possible to store the seektable in the container along with other metadata when muxing a FLAC file, so this stored seektable can be restored when demuxing the FLAC stream into a standalone FLAC file.

8.5.1. Seekpoint

Data	Description
u(64)	Sample number of the 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 7

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.
- The sample offsets are those of an unmuxed FLAC stream. The offsets MUST NOT be updated on muxing to reflect the new offsets of FLAC frames in a container.

8.6. 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, see [Vorbis]. 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 of which is 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+007E, 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, the FLAC metadata block 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.

8.6.1. Standard field names

Only one standard field name is defined: the channel mask field, in Section 8.6.2. No other field names are defined because the applicability of any field name is strongly tied to the content it is associated with. For example, field names useful for describing files that contain a single work of music would be unusable when labeling archived broadcasts, recordings of any kind, or a

collection of music works. Even when describing a single work of music, different conventions exist depending on the kind of music: orchestral music differs from music by solo artists or bands.

Despite the fact that no field names are formally defined, there is a general trend among devices and software capable of FLAC playback that are meant to play music. Most of those recognize at least 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, it is often the performer.
*Album: name of the collection the current work belongs to.

For a more comprehensive list of possible field names suited for describing a single work of music in various genres, the list of tags used in the MusicBrainz project, see [MusicBrainz], is suggested.

8.6.2. 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 is used to communicate that the channels in a file differ from the default channels defined in Section 9.1.3. For example, by default, a FLAC file containing two channels is interpreted to contain a left and right channel, but with this field, it is possible to describe different channel contents.

The channel mask consists of flag bits indicating which channels are present. The flags only signal which channels are present, not in which order, so if a file has to be encoded in which channels are ordered differently, they have to be reordered. This mask is stored with a hexadecimal representation, preceded by 0x, see the examples below. Please note that a file in which the channel order is defined through the WAVEFORMATEXTENSIBLE_CHANNEL_MASK is not streamable (see Section 7), 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

Bit number	Channel description
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 8

Following are three 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 four 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 four 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=0x12104.

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.

A WAVEFORMATEXTENSIBLE_CHANNEL_MASK field of 0x0 can be used to indicate that none of the audio channels of a file correlate with speaker positions. This is the case when audio needs to be decoded into speaker positions (e.g., Ambisonics B-format audio) or when a multitrack recording is contained.

It is possible for a WAVEFORMATEXTENSIBLE_CHANNEL_MASK field to code for fewer channels than are present in the audio. If that is the case, the remaining channels SHOULD NOT be rendered by a playback application unfamiliar with their purpose. For example, the Ambisonics UHJ format is compatible with stereo playback: its first two channels can be played back on stereo equipment, but all four channels together can be decoded into surround sound. For that example, the Vorbis comment field WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x3 would be set, indicating the first two channels are front left and front right, and other channels do not correlate with speaker positions directly.

If audio channels not assigned to any speaker are contained and decoding to speaker positions is possible, it is recommended to provide metadata on how this decoding should take place in another Vorbis comment field or an application metadata block. This document does not define such metadata.

8.7. Cuesheet

To either store the track and index point structure of a Compact Disc Digital Audio (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]. The description below is complete and further reference to [IEC.60908.1999] is not needed to implement this metadata block.

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 CD-DA, 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 Section 8.7.1 equal to the number of tracks specified previously.

Table 9

If the media catalog number is less than 128 bytes long, it is right-padded with 0x00 bytes. For CD-DA, this is a thirteen digit number, followed by 115 0x00 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.

8.7.1. Cuesheet track

Data	Description
u(64)	Track offset of the 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 Section 8.7.1.1 equal to the number of index points specified previously.

Table 10

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, because the CD-DA specification 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 be 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 0x00 characters MAY be used to denote the 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 MUST NOT be more than 100.

8.7.1.1. Cuesheet track index point

Data	Description
u(64)	Offset in samples, relative to the track offset, of the index point.
u(8)	The track index point number.
u(3*8)	Reserved. All bits MUST be set to zero.

Table 11

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, a 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.

8.8. 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, see [ID3v2]. However, contrary to the APIC frame in ID3v2, the media type and description are prepended with a 4-byte length field instead of being 0x00 delimited strings. A FLAC file MAY contain one or more picture metadata blocks.

Note that while the length fields for media 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.

Instead of picture data, the picture metadata block can also contain an URI as described in [RFC3986].

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

Data	Description
u(32)	The picture type according to next table
u(32)	The length of the media type string in bytes.
u(n*8)	The media type string as specified by [RFC2046], or the text string --> to signify that the data part is a URI of the picture instead of the picture data itself. This field must be in printable ASCII characters 0x20-0x7E.
u(32)	The length of the description string in bytes.
u(n*8)	The description of the picture, in UTF-8.

Data	Description
u(32)	The width of the picture in pixels.
u(32)	The height of the picture in pixels.
u(32)	The color depth of the picture in bits per pixel.
u(32)	For indexed-color pictures (e.g., GIF), the number of colors used, or 0 for non-indexed pictures.
u(32)	The length of the picture data in bytes.
u(n*8)	The binary picture data.

Table 12

The height, width, color depth, and 'number of colors' fields are for informational purposes only. Applications MUST NOT use them in decoding the picture or deciding how to display it, but MAY use them to decide whether to process a block or not (e.g., when selecting between different picture blocks) and MAY show them to the user. If a picture has no concept for any of these fields (e.g., vector images may not have a height or width in pixels) or the content of any field is unknown, the affected fields MUST be set to zero.

The following table contains all the defined picture types. Values other than those listed in the table are reserved. There MAY only be one each of picture types 1 and 2 in a file. In general practice, many FLAC playback devices and software 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, see [RFC2083]
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

Value	Picture type
20	Publisher or studio logotype

Table 13

The origin and use of value 17, "A bright colored fish", is unclear. This was copied to maintain compatibility with ID3v2. Applications are discouraged from offering this value to users when embedding a picture.

If not a picture but a URI is contained in this block, the following points apply:

* The URI can be either in absolute or relative form. If an URI is in relative form, it is related to the URI of the FLAC content processed.
* Applications MUST obtain explicit user approval to retrieve images via remote protocols and to retrieve local images not located in the same directory as the FLAC file being processed.
* Applications supporting linked images MUST handle unavailability of URIs gracefully. They MAY report unavailability to the user.
* Applications MAY reject processing URIs for any reason, in particular for security or privacy reasons.

9. 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.

Each frame header stores the audio sample rate, number of bits per sample, and number of channels independently of the streaminfo metadata block and other frame headers. This was done to permit multicasting of FLAC files, but it also allows these properties to change mid-stream. Because not all environments in which FLAC decoders are used are able to cope with changes to these properties during playback, a decoder MAY choose to stop decoding on such a change. A decoder that does not check for such a change could be vulnerable to buffer overflows. See also Section 12.

Note that storing audio with changing audio properties in FLAC results in various practical problems. For example, these changes of audio properties must happen on a frame boundary, or the process will not be lossless. When a variable block size is chosen to accommodate this, note that blocks smaller than 16 samples are not allowed and it is therefore not possible to store an audio stream in which these properties change within 16 samples of the last change or the start of the file. Also, since the streaminfo metadata block

can only accommodate a single set of properties, it is only valid for part of such an audio stream. Instead, it is RECOMMENDED to store an audio stream with changing properties in FLAC encapsulated in a container capable of handling such changes, as these do not suffer from the mentioned limitations. See Section 10 for details.

9.1. Frame header

Each frame MUST start on a byte boundary and 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 block size stream or 1 for a variable block size stream. If the blocking strategy is known, a decoder can include this bit when searching for the start of a frame to reduce the possibility of encountering a false positive, as the first two bytes of a frame are either 0xFFF8 for a fixed block size stream or 0xFFF9 for a variable block size stream.

9.1.1. Block size bits

Following the frame sync code and blocking strategy bit are 4 bits (the first 4 bits of the third byte of each frame) referred to as the block size bits. Their value relates to the block size according to the following table, where v is the value of the 4 bits as an unsigned number. If the block size bits code for an uncommon block size, this is stored after the coded number, see Section 9.1.6.

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

Table 14

9.1.2. Sample rate bits

The next 4 bits (the last 4 bits of the third byte of each frame), referred to as the sample rate bits, contain the sample rate of the audio according to the following table. If the sample rate bits code for an uncommon sample rate, this is stored after the uncommon block size or after the coded number if no uncommon block size was used. See Section 9.1.7.

Value	Sample rate
0b0000	sample rate only stored in the streaminfo metadata block

Value	Sample rate
0b0001	88.2 kHz
0b0010	176.4 kHz
0b0011	192 kHz
0b0100	8 kHz
0b0101	16 kHz
0b0110	22.05 kHz
0b0111	24 kHz
0b1000	32 kHz
0b1001	44.1 kHz
0b1010	48 kHz
0b1011	96 kHz
0b1100	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	forbidden

Table 15

9.1.3. Channels bits

The next 4 bits (the first 4 bits of the fourth byte of each frame), referred to as the channels bits, contain both the number of channels of the audio as well as any stereo decorrelation used according to the following table.

If a channel layout different than the ones listed in the following table is used, this can be signaled with a WAVEFORMATEXTENSIBLE_CHANNEL_MASK tag in a Vorbis comment metadata block, see Section 8.6.2 for details. Note that even when such a different channel layout is specified with a WAVEFORMATEXTENSIBLE_CHANNEL_MASK and the channel ordering in the following table is overridden, the channels bits still contain the actual number of channels coded in the frame. For details on the way left/side, right/side, and mid/side stereo are coded, see Section 4.2.

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	

Value	Channels
	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, left, right, stored as left/side stereo
0b1001	2 channels, left, right, stored as right/side stereo
0b1010	2 channels, left, right, stored as mid/side stereo
0b1011 - 0b1111	reserved

Table 16

9.1.4. Bit depth bits

The next 3 bits (bits 5, 6 and 7 of each fourth byte of each frame) contain the bit depth of the audio according to the following table.

Value	Bit depth
0b000	bit depth only stored in the 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	32 bits per sample

Table 17

The next bit is reserved and MUST be zero.

9.1.5. 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 block size streams, the sample number of the first sample in the frame is encoded. When the file contains a fixed block size stream, the frame number is encoded. See Section 9.1 on the blocking strategy bit which signals whether a stream is a fixed block size stream or a variable block size stream. Also see Appendix B.1.

The coded number is stored in a variable length code like UTF-8 as defined in [RFC3629], but extended to a maximum of 36 bits unencoded, 7 bytes encoded.

When a frame number is encoded, the value MUST NOT be larger than what fits a value of 31 bits unencoded or 6 bytes encoded. Please note that as most general purpose UTF-8 encoders and decoders follow [RFC3629], they will not be able to handle these extended codes.

Furthermore, while UTF-8 is specifically used to encode characters, FLAC uses it to encode numbers instead. To encode or decode a coded number, follow the procedures of Section 3 of [RFC3629], but instead of using a character number, use a frame or sample number, and instead of the table in Section 3 of [RFC3629], use the extended table below.

Number range (hexadecimal)	**Octet sequence (binary)**
0000 0000 0000 - 0000 0000 007F	0xxxxxxx
0000 0000 0080 - 0000 0000 07FF	110xxxxx 10xxxxxx
0000 0000 0800 - 0000 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0000 0001 0000 - 0000 001F FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0000 0020 0000 - 0000 03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0000 0400 0000 - 0000 7FFF FFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0000 8000 0000 - 000F FFFF FFFF	11111110 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Table 18

If the coded number is a frame number, it MUST be equal to the number of frames preceding the current frame. If the coded number is a sample number, it MUST be equal to the number of samples preceding the current frame. In a stream where these requirements are not met, seeking is not (reliably) possible.

For example, a frame that belongs to a variable block size stream and has exactly 51 billion samples preceding it, has its coded number constructed as follows.

```
Octets 1-5
0b11111110 0b10101111 0b10011111 0b10110101 0b10100011
               ^^^^^^     ^^^^^^     ^^^^^^     ^^^^^^
                 |          |          |      Bits 18-13
                 |          |      Bits 24-19
                 |      Bits 30-25
             Bits 36-31

Octets 6-7
0b10111000 0b10000000
    ^^^^^^     ^^^^^^
      |       Bits 6-1
  Bits 12-7
```

A decoder that relies on the coded number during seeking could be vulnerable to buffer overflows or getting stuck in an infinite loop if it seeks in a stream where the coded numbers are not strictly increasing or otherwise not valid. See also Section 12.

9.1.6. Uncommon block size

If the block size bits defined earlier in this section were 0b0110 or 0b0111 (uncommon block size minus 1 stored), this follows the coded number as either an 8-bit or a 16-bit unsigned number coded big-endian. A value of 65535 (corresponding to a block size of 65536) is forbidden and MUST NOT be used, because such a block size cannot be represented in the streaminfo metadata block. A value from 0 up to (and including) 14, which corresponds to a block size from 1 to 15, is only valid for the last frame in a stream and MUST NOT be used for any other frame. See also Section 8.2.

9.1.7. Uncommon sample rate

Following the uncommon block size (or the coded number if no uncommon block size is stored) 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.

The sample rate MUST NOT be 0 when the subframe contains audio. A sample rate of 0 MAY be used when non-audio is represented. See Section 8.2 for details.

9.1.8. Frame header CRC

Finally, after either the frame/sample number, an uncommon block size, or an uncommon sample rate, depending on whether the latter two are stored, is an 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.

9.2. Subframes

Following the frame header are a number of subframes equal to the number of audio channels. Note that as subframes contain a bitstream that does not necessarily has to be a whole number of bytes, only the first subframe always starts at a byte boundary.

9.2.1. Subframe header

Each subframe starts with a header. The first bit of the header MUST be 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 of order v-8, i.e., 0, 1, 2, 3 or 4
0b001101 - 0b011111	reserved
0b100000 - 0b111111	Subframe with a linear predictor of order v-31, i.e., 1 through 32 (inclusive)

Table 19

Following the subframe type bits is a bit that flags whether the subframe uses any wasted bits (see Section 9.2.2). If it is 0, the subframe doesn't use any wasted bits and the subframe header is complete. If it is 1, the subframe does use wasted bits and the number of used wasted bits follows unary coded.

9.2.2. Wasted bits per sample

Most uncompressed audio file formats can only store audio samples with a bit depth that is an integer number of bytes. Samples of which the bit depth is not an integer number of bytes are usually stored in such formats 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 the sense that they contain no information, but are stored anyway.

The FLAC format can optionally take advantage of these wasted bits by signaling their presence and coding the subframe without them. To do this, the wasted bits per sample flag in a subframe header is set to 0 and 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 this document, if a subframe header signals a certain number of wasted bits, it is said it 'uses' these wasted bits.

If a subframe uses wasted bits (i.e., k is not equal to 0), samples are coded ignoring k least-significant bits. For example, if a frame not employing stereo decorrelation specifies a sample size of 16 bits per sample in the frame header and k of a subframe is 3, samples in the subframe are coded as 13 bits per sample. For more details, see Section 9.2.3 on how the bit depth of a subframe is calculated. A decoder MUST add k least-significant zero bits by

shifting left (padding) after decoding a subframe sample. If the frame has left/side, right/side, or mid/side stereo, a decoder MUST perform padding on the subframes before restoring the channels to left and right. The number of wasted bits per sample MUST be such that the resulting number of bits per sample (of which the calculation is explained in Section 9.2.3) is larger than zero.

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 to slightly improve the compression of their otherwise lossless Meridian Lossless Packing codec, see [MLP]. There are also audio processors like lossyWAV, see [lossyWAV], which zero a number of least-sigificant bits for a block of samples, increasing the compression in a non-lossless way. Because of this, the number of wasted bits k MAY change between frames and MAY differ between subframes. If the number of wasted bits changes halfway through a subframe (e.g., the first part has 2 wasted bits and the second part has 4 wasted bits) the subframe uses the lowest number of wasted bits, as otherwise non-zero bits would be discarded and the process would not be lossless.

9.2.3. Constant subframe

In a constant subframe, only a single sample is stored. This sample is stored as an 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 (see Section 9.1.4), minus the number of used wasted bits coded in the subframe header (see Section 9.2.2). If a subframe is a side subframe (see Section 4.2), the bit depth of that subframe is increased by 1 bit.

9.2.4. Verbatim subframe

A verbatim subframe stores all samples unencoded in sequential order. See Section 9.2.3 on how a sample is stored unencoded. The number of samples that need to be stored in a subframe is given by the block size in the frame header.

9.2.5. 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	a(n-1)	N/A
2	2 * a(n-1) - a(n-2)	a(n-1) + a'(n-1)
3	3 * a(n-1) - 3 * a(n-2) + a(n-3)	a(n-1) + a'(n-1) + a''(n-1)
4	4 * a(n-1) - 6 * a(n-2) + 4 * a(n-3) - a(n-4)	a(n-1) + a'(n-1) + a''(n-1) + a'''(n-1)

Table 20

Where

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

As a predictor makes use of samples preceding the sample that is predicted, it can only be used when enough samples are known. As each subframe in FLAC is coded completely independently, the first few samples in each subframe cannot be predicted. Therefore, a number of so-called warm-up samples equal to the predictor order is stored. These are stored unencoded, bypassing the predictor and residual coding stages. See Section 9.2.3 on how samples are stored unencoded. The table below defines how a fixed predictor subframe appears in the bitstream.

Data	Description
s(n)	Unencoded warm-up samples (n = subframe's bits per sample * predictor order).
Coded residual	Coded residual as defined in Section 9.2.7

Table 21

As the fixed predictors are specified, they do not have to be stored. The fixed predictor order, which is stored in the subframe header, specifies which predictor is used.

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, the residual is decoded, and then the prediction can be added for each sample. This means that decoding is necessarily 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 order fixed predictors can be understood as polynomials fitted to the previous samples.

9.2.6. 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 by its corresponding past sample, the results are summed, and this sum 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, the residual is decoded, and then the prediction can be added for each sample. This means that decoding MUST be a sequential process within a subframe, as for each sample, enough decoded 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 = subframe's bits per sample * lpc order).
u(4)	(Predictor coefficient precision in bits)-1 (NOTE: 0b1111 is forbidden).
s(5)	Prediction right shift needed in bits.
s(n)	Predictor coefficients (n = predictor coefficient precision * lpc order).
Coded residual	Coded residual as defined in Section 9.2.7

Table 22

See Section 9.2.3 on how the warm-up samples are stored unencoded. The predictor coefficients are stored as an integer number coded big-endian, signed two's complement, where 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 NOT be negative, see Appendix B.4 for an explanation why this is.

Please note that the order in which the predictor coefficients appear in the bitstream corresponds to which **past** sample they belong to. 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.

9.2.7. 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 23

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 number of residual samples. The number of residual samples in the first partition is equal to (block size >> partition order) - predictor order, i.e., the block size divided by the number of partitions minus the predictor order. In all other partitions, the number of residual samples is equal to (block size >> partition order).

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

Each partition starts with a parameter. If the coded residual of a subframe is one with 4-bit Rice parameters (see the 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. If the coded residual of the current subframe is one with 5-bit Rice parameters, the first 5 bits of each partition indicate an escape code if they are 0b11111, otherwise, they contain the Rice parameter as an unsigned number as well.

9.2.7.1. Escaped partition

If 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. For example, when a partition is escaped and each residual sample is stored with 3 bits, the number -1 is represented as 0b111.

Note that it is possible that the number of bits with which each sample is stored is 0, which means all residual samples in that partition have a value of 0 and that no bits are used to store the samples. In that case, the partition contains nothing except the escape code and 0b00000.

9.2.7.2. Rice code

If a Rice parameter was provided for a certain partition, that 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. The zigzag encoded residual is called the folded residual.

Each folded residual sample is then split into two parts, a most-significant part and a least-significant part. The Rice parameter at the start of each partition determines where that split lies: it is the number of bits in the least-significant part. Each residual sample is then stored by coding the most-significant part as unary, followed by the least-significant part as binary.

For example, take a partition with Rice parameter 3 containing a folded residual sample with 38 as its value, which is 0b100110 in binary. The most-significant part is 0b100 (4) and is stored unary

as 0b00001. The least-significant part is 0b110 (6) and is stored as is. The Rice code word is thus 0b00001110. The Rice code words for all residual samples in a partition are stored consecutively.

To decode a Rice code word, zero bits must be counted until encountering a one bit, after which a number of bits given by the Rice parameter must be read. The count of zero bits is shifted left by the Rice parameter (i.e., multiplied by 2 raised to the power Rice parameter) and bitwise ORed with (i.e., added to) the read value. This is the folded residual value. An even folded residual value is shifted right 1 bit (i.e., divided by two) to get the (unfolded) residual value. An odd folded residual value is shifted right 1 bit and then has all bits flipped (1 added to and divided by -2) to get the (unfolded) residual value, subject to negative numbers being signed two's complement on the decoding machine.

Appendix D shows decoding of a complete coded residual.

9.2.7.3. Residual sample value limit

All residual sample values MUST be representable in the range offered by a 32-bit integer, signed one's complement. Equivalently, all residual sample values MUST fall in the range offered by a 32-bit integer signed two's complement excluding the most negative possible value of that range. This means residual sample values MUST NOT have an absolute value equal to, or larger than, 2 to the power 31. A FLAC encoder MUST make sure of this. If a FLAC encoder is, for a certain subframe, unable to find a suitable predictor for which all residual samples fall within said range, it MUST default to writing a verbatim subframe. Appendix A explains in which circumstances residual samples are already implicitly representable in said range and thus an additional check is not needed.

The reason for this limit is to ensure that decoders can use 32-bit integers when processing residuals, simplifying decoding. The reason the most negative value of a 32-bit int signed two's complement is specifically excluded is to prevent decoders from having to implement specific handling of that value, as it cannot be negated within a 32-bit signed int, and most library routines calculating an absolute value have undefined behavior on processing that value.

9.3. Frame footer

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

x^0. This CRC covers the whole frame excluding the 16-bit CRC, including the sync code.

10. Container mappings

The FLAC format can be used without any container, as it already provides for the most basic features normally associated with a container. However, the functionality this basic container provides is rather limited, and for more advanced features, like combining FLAC audio with video, it needs to be encapsulated by a more capable container. This presents a problem: because of these container features, the FLAC format mixes data that belongs to the encoded data (like block size and sample rate) with data that belongs to the container (like checksum and timecode). The choice was made to encapsulate FLAC frames as they are, which means some data will be duplicated and potentially deviating between the FLAC frames and the encapsulating container.

As FLAC frames are completely independent of each other, container format features handling dependencies do not need to be used. For example, all FLAC frames embedded in Matroska are marked as keyframes when they are stored in a SimpleBlock, and tracks in an MP4 file containing only FLAC frames do not need a sync sample box.

10.1. Ogg mapping

The Ogg container format is defined in [RFC3533]. The first packet of a logical bitstream carrying FLAC data is structured according to the following table.

Data	Description
5 bytes	Bytes 0x7F 0x46 0x4C 0x41 0x43 (as also defined by [RFC5334])
2 bytes	Version number of the FLAC-in-Ogg mapping. These bytes are 0x01 0x00, meaning version 1.0 of the mapping.
2 bytes	Number of header packets (excluding the first header packet) as an unsigned number coded big-endian.
4 bytes	The fLaC signature
4 bytes	A metadata block header for the streaminfo block
34 bytes	A streaminfo metadata block

Table 24

The number of header packets MAY be 0, which means the number of packets that follow is unknown. This first packet MUST NOT share a Ogg page with any other packets. This means the first page of a logical stream of FLAC-in-Ogg is always 79 bytes.

Following the first packet are one or more header packets, each of which contains a single metadata block. The first of these packets SHOULD be a Vorbis comment metadata block, for historic reasons. This is contrary to unencapsulated FLAC streams, where the order of metadata blocks is not important except for the streaminfo block and where a Vorbis comment metadata block is optional.

Following the header packets are audio packets. Each audio packet contains a single FLAC frame. The first audio packet MUST start on a new Ogg page, i.e., the last metadata block MUST finish its page before any audio packets are encapsulated.

The granule position of all pages containing header packets MUST be 0. For pages containing audio packets, the granule position is the number of the last sample contained in the last completed packet in the frame. The sample numbering considers interchannel samples. If a page contains no packet end (e.g., when it only contains the start of a large packet, which continues on the next page), then the granule position is set to the maximum value possible, i.e., 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF.

The granule position of the first audio data page with a completed packet MAY be larger than the number of samples contained in packets that complete on that page. In other words, the apparent sample number of the first sample in the stream following from the granule position and the audio data MAY be larger than 0. This allows, for example, a server to cast a live stream to several clients that joined at different moments, without rewriting the granule position for each client.

If an audio stream is encoded where audio properties (sample rate, number of channels, or bit depth) change at some point in the stream, this should be dealt with by finishing encoding of the current Ogg stream and starting a new Ogg stream, concatenated to the previous one. This is called chaining in Ogg. See the Ogg specification [RFC3533] for details.

10.2. Matroska mapping

The Matroska container format is defined in [I-D.ietf-cellar-matroska]. The codec ID (EBML path \Segment\Tracks\TrackEntry\CodecID) assigned to signal tracks carrying FLAC data is A_FLAC in ASCII. All FLAC data before the first audio frame (i.e., the fLaC ASCII signature and all metadata blocks) is stored as CodecPrivate data (EBML path \Segment\Tracks\TrackEntry\CodecPrivate).

Each FLAC frame (including all of its subframes) is treated as a single frame in the Matroska context.

If an audio stream is encoded where audio properties (sample rate, number of channels, or bit depth) change at some point in the stream, this should be dealt with by finishing the current Matroska segment and starting a new one with the new properties.

10.3. ISO Base Media File Format (MP4) mapping

The full encapsulation definition of FLAC audio in MP4 files was deemed too extensive to include in this document. A definition document can be found at [FLAC-in-MP4-specification].

11. Implementation status

Note to RFC Editor - please remove this entire section before publication, as well as the reference to RFC 7942.

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 at [FLAC-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. Section 2.1 of [RFC4732] provides

general information on DoS attacks on end-systems and describes some mitigation strategies. Areas of concern specific to FLAC follow.

It is extremely important for the decoder to be robust against malformed payloads. Payloads that do not conform to this specification **MUST NOT** cause the decoder to overrun its allocated memory or 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 with encoders are typically rarer. Malformed audio streams **MUST NOT** cause the encoder to misbehave because this would allow an attacker to attack transcoding gateways.

As with all compression algorithms, both encoding and decoding can produce an output much larger than the input. For decoding, the most extreme possible case of this is a frame with eight constant subframes of block size 65535 and coding for 32-bit PCM. This frame is only 49 bytes in size, but codes for more than 2 megabytes of uncompressed PCM data. For encoding, it is possible to have an even larger size increase, although such behavior is generally considered faulty. This happens if the encoder chooses a rice parameter that does not fit with the residual that has to be encoded. In such a case, very long unary coded symbols can appear, in the most extreme case, more than 4 gigabytes per sample. Decoder and encoder implementors are advised to take precautions to prevent excessive resource utilization in such cases.

Where metadata is handled, implementors are advised to either thoroughly test the handling of extreme cases or impose reasonable limits beyond the limits of this specification document. For example, a single Vorbis comment metadata block can contain millions of valid fields. It is unlikely such a limit is ever reached except in a potentially malicious file. Likewise, the media type and description of a picture metadata block can be millions of characters long, despite there being no reasonable use of such contents. One possible use case for very long character strings is in lyrics, which can be stored in Vorbis comment metadata block fields.

Various kinds of metadata blocks contain length fields or field counts. While reading a block following these lengths or counts, a decoder MUST make sure higher-level lengths or counts (most importantly, the length field of the metadata block itself) are not exceeded. As some of these length fields code string lengths, memory for which must be allocated, parsers MUST first verify that a block is valid before allocating memory based on its contents, except when explicitly instructed to salvage data from a malformed file.

Metadata blocks can also contain references, e.g., the picture metadata block can contain a URI. When following an URI, the security considerations of [RFC3986] apply. Applications MUST obtain explicit user approval to retrieve resources via remote protocols. Following external URIs introduces a tracking risk from on-path observers and the operator of the service hosting the URI. Likewise, the choice of scheme, if it isn't protected like https, could also introduce integrity attacks by an on-path observer. A malicious operator of the service hosting the URI can return arbitrary content that the parser will read. Also, such retrievals can be used in a DDoS attack when the URI points to a potential victim. Therefore, applications need to ask user approval for each retrieval individually, take extra precautions when parsing retrieved data, and cache retrieved resources. Applications MUST obtain explicit user approval to retrieve local resources not located in the same directory as the FLAC file being processed. Since relative URIs are permitted, applications MUST guard against directory traversal attacks and guard against a violation of a same-origin policy if such a policy is being enforced.

Seeking in a FLAC stream that is not in a container relies on the coded number in frame headers and optionally a seektable metadata block. Parsers MUST employ thorough checks on whether a found coded number or seekpoint is at all possible, e.g., whether it is within bounds and not directly contradicting any other coded number or seekpoint that the seeking process relies on. Without these checks, seeking might get stuck in an infinite loop when numbers in frames are non-consecutive or otherwise not valid, which could be used in denial of service attacks.

Implementors are advised to employ fuzz testing combined with different sanitizers on FLAC decoders to find security problems. Ignoring the results of CRC checks improves the efficiency of decoder fuzz testing.

See [FLAC-decoder-testbench] for a non-exhaustive list of FLAC files with extreme configurations that lead to crashes or reboots on some known implementations. Besides providing a starting point for security testing, this set of files can also be used to test conformance with this specification.

FLAC files may contain executable code, although the FLAC format is not designed for it and it is uncommon. One use case where FLAC is occasionally used to store executable code is when compressing images of mixed mode CDs, which contain both audio and non-audio data, of which the non-audio portion can contain executable code. In that case, the executable code is stored as if it were audio and is potentially obscured. Of course, it is also possible to store executable code as metadata, for example as a vorbis comment with

help of a binary-to-text encoding or directly in an application metadata block. Applications MUST NOT execute code contained in FLAC files or present parts of FLAC files as executable code to the user, except when an application has that explicit purpose, e.g., applications reading FLAC files as disc images and presenting it as virtual disc drive.

13. IANA Considerations

This document registers one new media type, "audio/flac", as defined in the following section, and creates a new IANA registry.

13.1. Media type registration

The following information serves as the registration form for the "audio/flac" media type. This media type is applicable for FLAC audio that is not packaged in a container as described in Section 10. FLAC audio packaged in such a container will take on the media type of that container, for example, audio/ogg when packaged in an Ogg container, or video/mp4 when packaged in an MP4 container alongside a video track.

Type name: audio

Subtype name: flac

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: as per THISRFC

Security considerations: see the security considerations in Section 12 of THISRFC

Interoperability considerations: see the descriptions of past format changes in Appendix B of THISRFC

Published specification: THISRFC

Applications that use this media type: ffmpeg, apache, firefox

Fragment identifier considerations: none

Additional information:

  Deprecated alias names for this type: audio/x-flac

  Magic number(s): fLaC

  File extension(s): flac

  Macintosh file type code(s): none

  Uniform Type Identifier: org.xiph.flac conforms to public.audio

  Windows Clipboard Format Name: audio/flac

Person & email address to contact for further information:
IETF CELLAR WG cellar@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: IETF CELLAR WG

Change controller: Internet Engineering Task Force (mailto:iesg@ietf.org)

Provisional registration? (standards tree only): NO

13.2. Application ID Registry

This document creates a new IANA registry called the "FLAC Application Metadata Block ID" registry. The values correspond to the 32-bit identifier described in Section 8.4.

To register a new Application ID in this registry, one needs an Application ID, a description, optionally a reference to a document describing the Application ID and a Change Controller (IETF or email of registrant). The Application IDs are to be allocated according to the "First Come First Served" policy [RFC8126], so that there is no impediment to registering any Application IDs the FLAC community encounters, especially if they were used in audio files but were not registered when the audio files were encoded. An Application ID can be any 32-bit value, but is often composed of 4 ASCII characters, to be human-readable.

The FLAC Application Metadata Block ID registry is assigned the following initial values, taken from the registration page at xiph.org (see [ID-registration-page]), which is no longer being maintained as it is replaced by this registry.

Application ID	ASCII rendition (if available)	Description	Specification	Change controller
0x41544348	ATCH	FlacFile	[FlacFile]	IETF
0x42534F4C	BSOL	beSolo		IETF
0x42554753	BUGS	Bugs Player		IETF
0x43756573	Cues	GoldWave cue points		IETF
0x46696361	Fica	CUE Splitter		IETF
0x46746F6C	Ftol	flac-tools		IETF
0x4D4F5442	MOTB	MOTB MetaCzar		IETF
0x4D505345	MPSE	MP3 Stream Editor		IETF
0x4D754D4C	MuML	MusicML: Music Metadata Language		IETF
0x52494646	RIFF	Sound Devices RIFF chunk storage		IETF
0x5346464C	SFFL	Sound Font FLAC		IETF

Application ID	ASCII rendition (if available)	Description	Specification	Change controller
0x534F4E59	SONY	Sony Creative Software		IETF
0x5351455A	SQEZ	flacsqueeze		IETF
0x54745776	TtWv	TwistedWave		IETF
0x55495453	UITS	UITS Embedding tools		IETF
0x61696666	aiff	FLAC AIFF chunk storage	[Foreign-metadata]	IETF
0x696D6167	imag	flac-image		IETF
0x7065656D	peem	Parseable Embedded Extensible Metadata		IETF
0x71667374	qfst	QFLAC Studio		IETF
0x72696666	riff	FLAC RIFF chunk storage	[Foreign-metadata]	IETF
0x74756E65	tune	TagTuner		IETF
0x773634C0	w64	FLAC Wave64 chunk storage	[Foreign-metadata]	IETF
0x78626174	xbat	XBAT		IETF
0x786D6364	xmcd	xmcd		IETF

Table 25

14. 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 (see [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, see [Rice].
*N. Levinson and J. Durbin; the FLAC reference encoder (see Section 11) uses an algorithm developed and refined by them for

determining the LPC coefficients from the autocorrelation coefficients, see [Durbin].

*And of course, Claude Shannon, see [Shannon].

The FLAC format, the FLAC reference implementation, and this document were originally developed by Josh Coalson. While many others have contributed since, this original effort is deeply appreciated.

15. References

15.1. Normative References

[I-D.ietf-cellar-matroska] Lhomme, S., Bunkus, M., and D. Rice, "Matroska Media Container Format Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-matroska-21, 22 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-matroska-21>.

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

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2083] Boutell, T., "PNG (Portable Network Graphics) Specification Version 1.0", RFC 2083, DOI 10.17487/RFC2083, March 1997, <https://www.rfc-editor.org/info/rfc2083>.

[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>.

[RFC3533] Pfeiffer, S., "The Ogg Encapsulation Format Version 0", RFC 3533, DOI 10.17487/RFC3533, May 2003, <https://www.rfc-editor.org/info/rfc3533>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986]
Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

15.2. Informative References

[Durbin] Durbin, J., "The Fitting of Time-Series Models", DOI 10.2307/1401322, December 1959, <https://www.jstor.org/stable/1401322>.

[FIR] "Finite impulse response - Wikipedia", <https://en.wikipedia.org/wiki/Finite_impulse_response>.

[FLAC-decoder-testbench] "FLAC decoder testbench", commit aa7b0c6, August 2023, <https://github.com/ietf-wg-cellar/flac-test-files>.

[FLAC-in-MP4-specification]
Montgomery, C., "Encapsulation of FLAC in ISO Base Media File Format", commit 78d85dd, July 2022, <https://github.com/xiph/flac/blob/master/doc/isoflac.txt>.

[FLAC-specification-github] "FLAC specification github repository", <https://github.com/ietf-wg-cellar/flac-specification>.

[FLAC-wiki-implementations] "FLAC specification wiki: Implementations", <https://github.com/ietf-wg-cellar/flac-specification/wiki/Implementations>.

[FLAC-wiki-interoperability] "FLAC specification wiki: Interoperability considerations", <https://github.com/ietf-wg-cellar/flac-specification/wiki/Interoperability-considerations>.

[FlacFile] "FlacFile", October 2007, <https://web.archive.org/web/20071023070305/http://firestuff.org:80/flacfile/>.

[Foreign-metadata] "Specification of foreign metadata storage in FLAC", November 2023, <https://github.com/xiph/flac/blob/master/doc/foreign_metadata_storage.md>.

[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>.

[ID-registration-page] "FLAC - ID Registry", <https://xiph.org/flac/id.html>.

[ID3v2] Nilsson, M., "id3v2.4.0-frames.txt", November 2000, <https://web.archive.org/web/20220903174949/https://id3.org/id3v2.4.0-frames>.

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

[LinearPrediction] "Linear prediction - Wikipedia", <https://en.wikipedia.org/wiki/Linear_prediction>.

[MLP] Gerzon, MA., Craven, PG., Stuart, JR., Law, MJ., and RJ. Wilson, "The MLP Lossless Compression System", September 1999, <https://www.aes.org/e-lib/online/browse.cfm?elib=8082>.

[MusicBrainz] MusicBrainz, "Tags & Variables - MusicBrainz Picard v2.10 documentation", <https://picard-docs.musicbrainz.org/en/variables/variables.html>.

[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>.

[RFC5334] Goncalves, I., Pfeiffer, S., and C. Montgomery, "Ogg Media Types", RFC 5334, DOI 10.17487/RFC5334, September 2008, <https://www.rfc-editor.org/info/rfc5334>.

[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>.

[Rice] Rice, RF. and JR. Plaunt, "Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data", DOI 10.1109/TCOM.1971.1090789, December 1971, <https://ieeexplore.ieee.org/document/1090789>.

[Shannon] Shannon, CE., "Communication in the Presence of Noise", DOI 10.1109/JRPROC.1949.232969, January 1949, <https://ieeexplore.ieee.org/document/1697831>.

[VarLengthCode] "Variable-length code - Wikipedia", <https://en.wikipedia.org/wiki/Variable-length_code>.

[Vorbis] Xiph.Org, "Ogg Vorbis I format specification: comment field and header specification", <https://xiph.org/vorbis/doc/v-comment.html>.

[lossyWAV] "lossyWAV - Hydrogenaudio Knowledgebase", <https://wiki.hydrogenaud.io/index.php?title=LossyWAV>.

[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 can be eliminated by using large enough data types. 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 appendix provides guidelines for choosing the appropriate data type for each step of encoding and decoding FLAC files.

In this appendix, signed data types are signed two's complement.

A.1. Determining the 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, a 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.

When multiplying two numbers, the number of bits needed for the result is the size of the first number plus the size of the second number. If, for example, a 16-bit signed integer is multiplied by another 16-bit signed integer, the result needs at least 32 bits to be stored without overflowing. To show this in practice, the largest signed value that can be stored in 4 bits is -8. (-8)*(-8) is 64, which needs at least 8 bits (signed) to store.

A.2. Stereo decorrelation

When stereo decorrelation is used, the side channel will have one extra bit of bit depth, see Section 4.2.

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 bits, the decoding of a side subframe in such a file will need a data type with at least 17 bits 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 bits.

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 Appendix A.1 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 if a third order fixed predictor is used.

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

Table 26

Where

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

For subframes with a linear predictor, the calculation is a little more complicated. Each prediction is the 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. 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 bit depth of the source is 24, the current subframe encodes a side channel (see Section 4.2), 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 + ceil(log2(12)) = 44 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 + ceil(log2(8)) = 32 bits.

A.4. Residual

As stated in Section 9.2.7, an encoder must make sure residual samples are representable by a 32-bit integer, signed two's complement, excluding the most negative value. Continuing as in the previous section, it is possible to calculate when residual samples already implicitly fit and when an additional check is needed. This implicit fit is achieved when residuals would fit a theoretical 31-bit signed int, as that satisfies both of the mentioned criteria. When this implicit fit is not achieved, all residual values must be calculated and checked individually.

For the residual of a fixed predictor, the maximum residual sample size was already calculated in the previous section. However, for a linear predictor, the prediction is shifted right by a certain amount. The number of bits needed for the residual is the number of bits calculated in the previous section, reduced by the prediction right shift, and increased by one bit to account for the subtraction of the prediction from the current sample on encoding.

Taking the last example of the previous section, where 32 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 32 - 10 + 1 = 23 bits, which means it is not necessary to perform the aforementioned check.

As another example, when encoding 32-bit PCM with fixed predictors, all predictor orders must be checked. While the 0-order fixed predictor is guaranteed to have residual samples that fit a 32-bit signed int, it might produce a residual sample value that is the most negative representable value of that 32-bit signed int.

Note that on decoding, while the residual sample values are limited to the aforementioned range, the predictions are not. This means that while the decoding of the residual samples can happen fully in 32-bit signed integers, decoders must be sure to execute the addition of each residual sample to its accompanying prediction with a wide enough signed integer data type like on encoding.

A.5. 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. Past format changes

This informational appendix documents the changes made to the FLAC format over the years. This information might be of use when encountering FLAC files that were made with software following the format as it was before the changes documented in this appendix.

The FLAC format was first specified in December 2000 and the bitstream format was considered frozen with the release of FLAC (the reference encoder/decoder) 1.0 in July 2001. Only changes made since this first stable release are considered in this appendix. Changes

made to the FLAC streamable subset definition (see Section 7) are not considered.

B.1. Addition of blocking strategy bit

Perhaps the largest backwards incompatible change to the specification was published in July 2007. Before this change, variable block size streams were not explicitly marked as such by a flag bit in the frame header. A decoder had two ways to detect a variable block size stream, either by comparing the minimum and maximum block size in the STREAMINFO metadata block (which are equal for a fixed block size stream), or, if a decoder did not receive a STREAMINFO metadata block, by detecting a change of block size during a stream, which could in theory not happen at all. As the meaning of the coded number in the frame header depends on whether or not a stream is variable block size, this presented a problem: the meaning of the coded number could not be reliably determined. To fix this problem, one of the reserved bits was changed to be used as a blocking strategy bit. See also Section 9.1.

Along with the addition of a new flag, the meaning of the block size bits (see Section 9.1.1) was subtly changed. Initially, block size bits patterns 0b0001-0b0101 and 0b1000-0b1111 could only be used for fixed block size streams, while 0b0110 and 0b0111 could be used for both fixed block size and variable block size streams. With the change, these restrictions were lifted, and patterns 0b0001-0b1111 are now used for both variable block size and fixed block size streams.

B.2. Restriction of encoded residual samples

Another change to the specification was deemed necessary during standardization by the CELLAR working group of the IETF. As specified in Section 9.2.7 a limit is imposed on residual samples. This limit was not specified prior to the IETF standardization effort. However, as far as was known to the working group, no FLAC encoder at that time produced FLAC files containing residual samples exceeding this limit. This is mostly because it is very unlikely to encounter residual samples exceeding this limit when encoding 24-bit PCM, and encoding of PCM with higher bit depths was not yet implemented in any known encoder. In fact, these FLAC encoders would produce corrupt files upon being triggered to produce such residual samples and it is unlikely any non-experimental encoder would ever do so, even when presented with crafted material. Therefore, it was not expected that existing implementations would be rendered non-compliant by this change.

B.3. Addition of 5-bit Rice parameters

One significant addition to the format was the residual coding method using 5-bit Rice parameters. Prior to publication of this addition in July 2007, there was only one residual coding method specified, a partitioned Rice code with 4-bit Rice parameters. The range offered by this coding method proved too small when encoding 24-bit PCM, therefore, a second residual coding method was specified, identical to the first but with 5-bit Rice parameters.

B.4. Restriction of LPC shift to non-negative values

As stated in Section 9.2.6, the predictor right shift is a number signed two's complement, which MUST NOT be negative. This is because right shifting a number by a negative amount is undefined behavior in the C programming language standard. The intended behavior was that a positive number would be a right shift and a negative number would be a left shift. The FLAC reference encoder was changed in 2007 to not generate LPC subframes with a negative predictor right shift, as it turned out that the use of such subframes would only very rarely provide any benefit, and the decoders that were already widely in use at that point were not able to handle such subframes.

Appendix C. Interoperability considerations

As documented in Appendix B, there have been some changes and additions to the FLAC format. Additionally, implementation of certain features of the FLAC format took many years, meaning early decoder implementations could not be tested against files with these features. Finally, many lower-quality FLAC decoders only implement just enough features required for playback of the most common FLAC files.

This appendix provides some considerations for encoder implementations aiming to create highly compatible files. As this topic is one that might change after this document is finished, consult [FLAC-wiki-interoperability] for more up-to-date information.

C.1. Features outside of the streamable subset

As described in Section 7, FLAC specifies a subset of its capabilities as the FLAC streamable subset. Certain decoders may choose to only decode FLAC files conforming to the limitations imposed by the streamable subset. Therefore, maximum compatibility with decoders is achieved when the limitations of the FLAC streamable subset are followed when creating FLAC files.

C.2. Variable block size

Because it is often difficult to find the optimal arrangement of block sizes for maximum compression, most encoders choose to create files with a fixed block size. Because of this, many decoder implementations receive minimal use when handling variable block size streams, and this can reveal bugs or reveal that implementations do not decode them at all. Furthermore, as explained in Appendix B.1, there have been some changes to the way variable block size streams were encoded. Because of this, maximum compatibility with decoders is achieved when FLAC files are created using fixed block size streams.

C.3. 5-bit Rice parameter

As the addition of the 5-bit Rice parameter, as described in Appendix B.3, occurred quite a few years after the FLAC format was first introduced, some early decoders might not be able to decode files containing such Rice parameters. The introduction of this was specifically aimed at improving compression of 24-bit PCM audio, and compression of 16-bit PCM audio only rarely benefits from using 5-bit Rice parameters. Therefore, maximum compatibility with decoders is achieved when FLAC files containing audio with a bit depth of 16 bits or lower are created without any use of 5-bit Rice parameters.

C.4. Rice escape code

Escaped Rice partitions are seldom used, as it turned out their use provides only a very small compression improvement. As many encoders therefore do not use these by default or are not capable of producing them at all, it is likely that many decoder implementations are not able to decode them correctly. Therefore, maximum compatibility with decoders is achieved when FLAC files are created without any use of escaped Rice partitions.

C.5. Uncommon block size

For unknown reasons, some decoders have chosen to support only common block sizes for all but the last block of a stream. Therefore, maximum compatibility with decoders is achieved when creating FLAC files using common block sizes, as listed in Section 9.1.1, for all but the last block of a stream.

C.6. Uncommon bit depth

Most audio is stored in bit depths that are a whole number of bytes, e.g., 8, 16 or 24 bit. There is however audio with different bit depths. A few examples:

*DVD-Audio has the possibility to store 20 bit PCM audio.

*DAT and DV can store 12 bit PCM audio.
*NICAM-728 samples at 14 bit, which is companded to 10 bit.
*8-bit μ-law can be losslessly converted to 14 bit (Linear) PCM.
*8-bit A-law can be losslessly converted to 13 bit (Linear) PCM.

The FLAC format can contain these bit depths directly, but because they are uncommon, some decoders are not able to process the resulting files correctly. It is possible to store these formats in a FLAC file with a more common bit depth without sacrificing compression by padding each sample with zero bits to a bit depth that is a whole byte. The FLAC format can efficiently compress these wasted bits. See Section 9.2.2 for details.

Therefore, maximum compatibility with decoders is achieved when FLAC files are created by padding samples of such audio with zero bits to the bit depth that is the next whole number of bytes.

In cases where the original signal is already padded, this operation cannot be reversed losslessly without knowing the original bit depth. To leave no ambiguity, the original bit depth needs to be stored, for example, in a vorbis comment field, by storing the header of the original file, or in a description of the file. The choice of a suitable method is left to the implementer.

Besides audio with a 'non-whole byte' bit depth, some decoder implementations have chosen to only accept FLAC files coding for PCM audio with a bit depth of 16 bit. Many implementations support bit depths up to 24 bit but no higher. Consult [FLAC-wiki-interoperability] for more up-to-date information.

C.7. Multi-channel audio and uncommon sample rates

Many FLAC audio players are unable to render multi-channel audio or audio with an uncommon sample rate. While this is not a concern specific to the FLAC format, it is of note when requiring maximum compatibility with decoders. Unlike the previously mentioned interoperability considerations, this is one where compatibility cannot be improved without sacrificing the lossless nature of the FLAC format.

From a non-exhaustive inquiry, it seems that a non-negligible amount of players, especially hardware players, do not support audio with 3 or more channels or sample rates other than those considered common, see Section 9.1.2.

For those players that do support and are able to render multi-channel audio, many do not parse and use the WAVEFORMATEXTENSIBLE_CHANNEL_MASK tag (see Section 8.6.2). This too is an interoperability consideration where compatibility cannot be improved without sacrificing the lossless nature of the FLAC format.

C.8. Changing audio properties mid-stream

Each FLAC frame header stores the audio sample rate, number of bits per sample, and number of channels independently of the streaminfo metadata block and other frame headers. This was done to permit multicasting of FLAC files, but it also allows these properties to change mid-stream. However, many FLAC decoders do not handle such changes, as few other formats are capable of holding such streams and changing playback properties during playback is often not possible without interrupting playback. Also, as explained in Section 9, using this feature of FLAC results in various practical problems.

However, even when storing an audio stream with changing properties in FLAC encapsulated in a container capable of handling such changes, as recommended in Section 9, many decoders are not able to decode such a stream correctly. Therefore, maximum compatibility with decoders is achieved when FLAC files are created with a single set of audio properties, in which the properties coded in the streaminfo metadata block (see Section 8.2) and the properties coded in all frame headers (see Section 9.1) are the same. This can be achieved by splitting up an input stream with changing audio properties at the points where these properties change into separate streams or files.

Appendix D. Examples

This informational appendix contains short example FLAC files that 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 cursorily read the specification and that the reader refers to the specification for explanation of the terminology used. These examples mostly focus on the layout of several metadata blocks and subframe types and the implications of certain aspects (for example, wasted bits and stereo decorrelation) on this layout.

The examples feature 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. The files that are explored in these examples can be found at [FLAC-specification-github].

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

D.1. Decoding example 1

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

D.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......
```

D.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
```

D.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 bytes	0x664C6143	fLaC
0x04+0	1 bit	0b1	Last metadata block
0x04+1	7 bits	0b0000000	Streaminfo metadata block

Start	Length	Contents	Description
0x05+0	3 bytes	0x000022	Length 34 byte

Table 27

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 block size streams, 0xfff8.

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

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

Table 28

The minimum and maximum block size are both 4096. This was apparently the block size 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 (block size, total number of samples, sample rate), interchannel samples are meant.

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

D.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 bits	0xff, 0b1111100	frame sync
0x2b+7	1 bit	0b0	blocking strategy
0x2c+0	4 bits	0b0110	8-bit block size further down
0x2c+4	4 bits	0b1001	sample rate 44.1 kHz
0x2d+0	4 bits	0b0001	stereo, no decorrelation

Start	Length	Contents	Description
0x2d+4	3 bits	0b100	bit depth 16 bit
0x2d+7	1 bit	0b0	mandatory 0 bit
0x2e+0	1 byte	0x00	frame number 0
0x2f+0	1 byte	0x00	block size 1
0x30+0	1 byte	0xbf	frame header CRC

Table 29

As the stream is a fixed block size 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 first subframe starts, which is broken down in the following table.

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

Table 30

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 this subframe uses 2 wasted bits.

As this is a verbatim subframe, the subframe only contains unencoded sample values. With a block size of 1, it contains only a single sample. The bit depth of the audio is 16 bits, but as the subframe header signals the use of 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 bits (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, and is broken down in the following table.

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

Table 31

Here the wasted bits flag is also one, but the unary coded number that follows it is 4 bit long, indicating the use of 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 undo stereo decorrelation if that was applicable.

As the last subframe ends byte-aligned, no padding bits follow it. 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 checksum, we line up the samples interleaved, byte-aligned, little endian, signed two's complement. The first sample, with value 25588, translates to 0xf463, the second sample, with value 10416, translates to 0xb028. When computing the MD5 checksum with 0xf463b028 as input, we get the MD5 checksum found in the header, so decoding was lossless.

D.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, a Vorbis comment metadata block, and a padding metadata block.

D.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
```

D.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
```

D.2.3. Streaminfo metadata block

Most of the streaminfo block, including its header, 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 bytes	0x0010	Min. block size 16
0x0a+0	2 bytes	0x0010	Max. block size 16
0x0c+0	3 bytes	0x000017	Min. frame size 23 byte
0x0f+0	3 bytes	0x000044	Max. frame size 68 byte
0x15+4	36 bits	0b0000, 0x00000013	Total no. of samples 19
0x1a	16 bytes	(...)	MD5 checksum

Table 32

This time, the minimum and maximum block sizes are reflected in the file: there is one block of 16 samples, the last block (which has 3 samples) is not considered for the minimum block size. The MD5 checksum is 0xd5b0 5649 75e9 8b8d 8b93 0422 757b 8103, this will be verified at the end of this example.

D.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 bits	0b0000011	Seektable metadata block
0x2b+0	3 bytes	0x000012	Length 18 byte
0x2e+0	8 bytes	0x0000000000000000	Seekpoint to sample 0
0x36+0	8 bytes	0x0000000000000000	Seekpoint to offset 0
0x3e+0	2 bytes	0x0010	Seekpoint to block size 16

Table 33

D.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 bits	0b0000100	Vorbis comment metadata block
0x41+0	3 bytes	0x00003a	Length 58 byte
0x44+0	4 bytes	0x20000000	Vendor string length 32 byte
0x48+0	32 bytes	(...)	Vendor string
0x68+0	4 bytes	0x01000000	Number of fields 1
0x6c+0	4 bytes	0x0e000000	Field length 14 byte
0x70+0	14 bytes	(...)	Field contents

Table 34

The vendor string is reference libFLAC 1.3.3 20190804, and the field contents of the only field is TITLE=שלום. The Vorbis comment field is 14 bytes but only 10 characters in size, because it contains four 2-byte characters.

D.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 bits	0b0000001	Padding metadata block
0x7f+0	3 bytes	0x000006	Length 6 byte
0x82+0	6 bytes	0x000000000000	Padding bytes

Table 35

D.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 bits	0xff, 0b1111100	frame sync
0x89+7	1 bit	0b0	blocking strategy
0x8a+0	4 bits	0b0110	8-bit block size further down
0x8a+4	4 bits	0b1001	sample rate 44.1 kHz
0x8b+0	4 bits	0b1001	side-right stereo
0x8b+4	3 bits	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	block size 16
0x8e+0	1 byte	0x99	frame header CRC

Table 36

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 bits	0b001001	fixed subframe, 1st order
0x8f+7	1 bit	0b0	no wasted bits used
0x90+0	17 bits	0x0867, 0b0	unencoded warm-up sample

Table 37

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

Start	Length	Contents	Description
0x92+1	2 bits	0b00	Rice code with 4-bit parameter
0x92+3	4 bits	0b0000	Partition order 0
0x92+7	4 bits	0b1011	Rice parameter 11
0x93+3	4 bits	0b0001	Quotient 3
0x93+7	11 bits	0b00011110100	Remainder 244
0x95+2	2 bits	0b01	Quotient 1
0x95+4	11 bits	0b01000100001	Remainder 545
0x96+7	2 bits	0b01	Quotient 1
0x97+1	11 bits	0b00110011000	Remainder 408
0x98+4	1 bit	0b1	Quotient 0
0x98+5	11 bits	0b11101011101	Remainder 1885
0x9a+0	1 bit	0b1	Quotient 0

Start	Length	Contents	Description
0x9a+1	11 bits	0b11101110000	Remainder 1904
0x9b+4	1 bit	0b1	Quotient 0
0x9b+5	11 bits	0b10101101111	Remainder 1391
0x9d+0	1 bit	0b1	Quotient 0
0x9d+1	11 bits	0b11000000000	Remainder 1536
0x9e+4	1 bit	0b1	Quotient 0
0x9e+5	11 bits	0b10000010111	Remainder 1047
0xa0+0	1 bit	0b1	Quotient 0
0xa0+1	11 bits	0b10010101110	Remainder 1198
0xa1+4	1 bit	0b1	Quotient 0
0xa1+5	11 bits	0b01100100001	Remainder 801
0xa3+0	13 bits	0b0000000000001	Quotient 12
0xa4+5	11 bits	0b11011100111	Remainder 1767
0xa6+0	1 bit	0b1	Quotient 0
0xa6+1	11 bits	0b01001110111	Remainder 631
0xa7+4	1 bit	0b1	Quotient 0
0xa7+5	11 bits	0b01000100100	Remainder 548
0xa9+0	1 bit	0b1	Quotient 0
0xa9+1	11 bits	0b01000010101	Remainder 533
0xaa+4	1 bit	0b1	Quotient 0
0xaa+5	11 bits	0b00100001100	Remainder 268

Table 38

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 39

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 40

With this, the decoding of the first subframe is complete. The 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, the results are in the next table. The next step is undoing stereo decorrelation, which is done in the following table. As the stereo decorrelation 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

Subframe 1	Subframe 2	Left	Right
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 41

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

D.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 bits	0xff, 0b1111100	frame sync
0xcd+7	1 bit	0b0	blocking strategy
0xce+0	4 bits	0b0110	8-bit block size further down
0xce+4	4 bits	0b1001	sample rate 44.1 kHz
0xcf+0	4 bits	0b0001	stereo, no decorrelation
0xcf+4	3 bits	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	block size 3
0xd2+0	1 byte	0xa4	frame header CRC

Table 42

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 bits	0b000001	verbatim subframe

Start	Length	Contents	Description
0xd3+7	1 bit	0b0	no wasted bits used
0xd4+0	16 bits	0xc382	16-bit unencoded sample
0xd6+0	16 bits	0xc40b	16-bit unencoded sample
0xd8+0	16 bits	0xc14a	16-bit unencoded sample

Table 43

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 bits	0b000001	verbatim subframe
0xda+7	1 bit	0b1	wasted bits used
0xdb+0	1 bit	0b1	1 wasted bit used
0xdb+1	15 bits	0b110111001001000	15-bit unencoded sample
0xdd+0	15 bits	0b110111010000001	15-bit unencoded sample
0xde+7	15 bits	0b110110110011111	15-bit unencoded sample

Table 44

As this subframe uses 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.

D.2.9. MD5 checksum verification

All samples in the file have been decoded, we can now verify the MD5 checksum. 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) per line.

```
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 MD5 checksum of this is indeed the same as the one found in the streaminfo metadata block.

D.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.

D.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                             .
```

D.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.
```

D.3.3. Streaminfo metadata block

Most of the streaminfo metadata block, including its header, 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 bytes	0x00001f	Min. frame size 31 byte
0x0f+0	3 bytes	0x00001f	Max. frame size 31 byte
0x12+0	20 bits	0x07d0, 0x0000	Sample rate 32000 hertz
0x14+4	3 bits	0b000	1 channel
0x14+7	5 bits	0b00111	Sample bit depth 8 bit
0x15+4	36 bits	0b0000, 0x00000018	Total no. of samples 24
0x1a	16 bytes	(...)	MD5 checksum

Table 45

D.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 bits	0xff, 0b1111100	Frame sync
0x2b+7	1 bit	0b0	blocking strategy
0x2c+0	4 bits	0b0110	8-bit block size further down
0x2c+4	4 bits	0b1000	Sample rate 32 kHz

Start	Length	Contents	Description
0x2d+0	4 bits	0b0000	Mono audio (1 channel)
0x2d+4	3 bits	0b001	Bit depth 8 bit
0x2d+7	1 bit	0b0	Mandatory 0 bit
0x2e+0	1 byte	0x00	Frame number 0
0x2f+0	1 byte	0x17	Block size 24
0x30+0	1 byte	0xe9	Frame header CRC

Table 46

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

Table 47

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

Start	Length	Contents	Description
0x37+5	2 bits	0b00	Rice-coded residual, 4-bit parameter
0x37+7	4 bits	0b0010	Partition order 2
0x38+3	4 bits	0b0011	Rice parameter 3
0x38+7	1 bit	0b1	Quotient 0
0x39+0	3 bits	0b110	Remainder 6
0x39+3	1 bit	0b1	Quotient 0
0x39+4	3 bits	0b001	Remainder 1
0x39+7	4 bits	0b0001	Quotient 3
0x3a+3	3 bits	0b001	Remainder 1
0x3a+6	4 bits	0b1111	No Rice parameter, escape code
0x3b+2	5 bits	0b00101	Partition encoded with 5 bits
0x3b+7	5 bits	0b10110	Residual -10
0x3c+4	5 bits	0b11010	Residual -6
0x3d+1	5 bits	0b00010	Residual 2
0x3d+6	5 bits	0b01000	Residual 8
0x3e+3	5 bits	0b01000	Residual 8

Start	Length	Contents	Description
0x3f+0	5 bits	0b00110	Residual 6
0x3f+5	4 bits	0b0010	Rice parameter 2
0x40+1	22 bits	(...)	Residual partition 3
0x42+7	4 bits	0b0001	Rice parameter 1
0x43+3	23 bits	(...)	Residual partition 4

Table 48

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.

The following table breaks down the calculation for each sample. For example, the predictor without shift value of row 4 is found by applying the predictor with the three warm-up samples: 7*111 - 6*79 + 2*0 = 303. This value is then shifted right by 2 bits: 303 >> 2 = 75. Then, the decoded residual sample is added: 75 + 3 = 78.

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
-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 49

By lining all these samples up, we get the following input for the MD5 checksum calculation process.

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

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

Authors' Addresses

Martijn van Beurden
Netherlands

Email: mvanb1@gmail.com

Andrew Weaver

Email: theandrewjw@gmail.com