Internet Engineering Task Force (IETF)
Request for Comments: 7845
Updates: 5334
Category: Standards Track
ISSN: 2070-1721

T. Terriberry
Mozilla Corporation
R. Lee
Voicetronix
R. Giles
Mozilla Corporation

April 2016

Ogg Encapsulation for the Opus Audio Codec

Abstract

This document defines the Ogg encapsulation for the Opus interactive speech and audio codec. This allows data encoded in the Opus format to be stored in an Ogg logical bitstream.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The IETF Opus codec is a low-latency audio codec optimized for both voice and general-purpose audio. See [RFC6716] for technical details. This document defines the encapsulation of Opus in a continuous, logical Ogg bitstream [RFC3533]. Ogg encapsulation provides Opus with a long-term storage format supporting all of the essential features, including metadata, fast and accurate seeking, corruption detection, recapture after errors, low overhead, and the ability to multiplex Opus with other codecs (including video) with minimal buffering. It also provides a live streamable format capable of delivery over a reliable stream-oriented transport, without requiring all the data (or even the total length of the data) up-front, in a form that is identical to the on-disk storage format.

Ogg bitstreams are made up of a series of "pages", each of which contains data from one or more "packets". Pages are the fundamental unit of multiplexing in an Ogg stream. Each page is associated with a particular logical stream and contains a capture pattern and checksum, flags to mark the beginning and end of the logical stream, and a "granule position" that represents an absolute position in the stream, to aid seeking. A single page can contain up to 65,025 octets of packet data from up to 255 different packets. Packets can be split arbitrarily across pages and continued from one page to the next (allowing packets much larger than would fit on a single page). Each page contains "lacing values" that indicate how the data is partitioned into packets, allowing a demultiplexer (demuxer) to recover the packet boundaries without examining the encoded data. A packet is said to "complete" on a page when the page contains the final lacing value corresponding to that packet.

This encapsulation defines the contents of the packet data, including the necessary headers, the organization of those packets into a logical stream, and the interpretation of the codec-specific granule position field. It does not attempt to describe or specify the existing Ogg container format. Readers unfamiliar with the basic concepts mentioned above are encouraged to review the details in [RFC3533].

2. Terminology

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 [RFC2119].

3. Packet Organization

An Ogg Opus stream is organized as follows (see Figure 1 for an example).

```
        Page 0         Pages 1 ... n        Pages (n+1) ...
     +------------+ +---+ +---+ ... +---+ +-----------+ +---------+ +--
     |            | |   | |   |     |   | |           | |         | |
     |+----------+| |+-----------------+| |+-------------------+ +-----
     |||ID Header|| ||  Comment Header || ||Audio Data Packet 1| | ...
     |+----------+| |+-----------------+| |+-------------------+ +-----
     |            | |   | |   |     |   | |           | |         | |
     +------------+ +---+ +---+ ... +---+ +-----------+ +---------+ +--
     ^      ^                           ^
     |      |                           |
     |      |                           Mandatory Page Break
     |      |
     |      ID header is contained on a single page
     |
     ’Beginning Of Stream’
```

Figure 1: Example Packet Organization for a Logical Ogg Opus Stream

There are two mandatory header packets. The first packet in the logical Ogg bitstream MUST contain the identification (ID) header, which uniquely identifies a stream as Opus audio. The format of this header is defined in Section 5.1. It is placed alone (without any other packet data) on the first page of the logical Ogg bitstream and completes on that page. This page has its ’beginning of stream’ flag set.

The second packet in the logical Ogg bitstream MUST contain the comment header, which contains user-supplied metadata. The format of this header is defined in Section 5.2. It MAY span multiple pages, beginning on the second page of the logical stream. However many pages it spans, the comment header packet MUST finish the page on which it completes.

All subsequent pages are audio data pages, and the Ogg packets they contain are audio data packets. Each audio data packet contains one Opus packet for each of N different streams, where N is typically one for mono or stereo, but MAY be greater than one for multichannel audio. The value N is specified in the ID header (see Section 5.1.1), and is fixed over the entire length of the logical Ogg bitstream.

The first (N - 1) Opus packets, if any, are packed one after another into the Ogg packet, using the self-delimiting framing from Appendix B of [RFC6716]. The remaining Opus packet is packed at the end of the Ogg packet using the regular, undelimited framing from Section 3 of [RFC6716]. All of the Opus packets in a single Ogg packet MUST be constrained to have the same duration. An implementation of this specification SHOULD treat any Opus packet whose duration is different from that of the first Opus packet in an Ogg packet as if it were a malformed Opus packet with an invalid Table Of Contents (TOC) sequence.

The TOC sequence at the beginning of each Opus packet indicates the coding mode, audio bandwidth, channel count, duration (frame size), and number of frames per packet, as described in Section 3.1 of [RFC6716]. The coding mode is one of SILK, Hybrid, or Constrained Energy Lapped Transform (CELT). The combination of coding mode, audio bandwidth, and frame size is referred to as the configuration of an Opus packet.

Packets are placed into Ogg pages in order until the end of stream. Audio data packets might span page boundaries. The first audio data page could have the ’continued packet’ flag set (indicating the first audio data packet is continued from a previous page) if, for example, it was a live stream joined mid-broadcast, with the headers pasted on the front. If a page has the ’continued packet’ flag set and one of the following conditions is also true:

- the previous page with packet data does not end in a continued packet (does not end with a lacing value of 255) OR

- the page sequence numbers are not consecutive,

then a demuxer MUST NOT attempt to decode the data for the first packet on the page unless the demuxer has some special knowledge that would allow it to interpret this data despite the missing pieces. An implementation MUST treat a zero-octet audio data packet as if it were a malformed Opus packet as described in Section 3.4 of [RFC6716].

A logical stream ends with a page with the ’end of stream’ flag set, but implementations need to be prepared to deal with truncated streams that do not have a page marked ’end of stream’. There is no reason for the final packet on the last page to be a continued packet, i.e., for the final lacing value to be 255. However, demuxers might encounter such streams, possibly as the result of a transfer that did not complete or of corruption. If a packet continues onto a subsequent page (i.e., when the page ends with a lacing value of 255) and one of the following conditions is also true:

- the next page with packet data does not have the 'continued packet' flag set, OR
- there is no next page with packet data, OR
- the page sequence numbers are not consecutive,

then a demuxer MUST NOT attempt to decode the data from that packet unless the demuxer has some special knowledge that would allow it to interpret this data despite the missing pieces. There MUST NOT be any more pages in an Opus logical bitstream after a page marked 'end of stream'.

4. Granule Position

The granule position MUST be zero for the ID header page and the page where the comment header completes. That is, the first page in the logical stream and the last header page before the first audio data page both have a granule position of zero.

The granule position of an audio data page encodes the total number of PCM samples in the stream up to and including the last fully decodable sample from the last packet completed on that page. The granule position of the first audio data page will usually be larger than zero, as described in Section 4.5.

A page that is entirely spanned by a single packet (that completes on a subsequent page) has no granule position, and the granule position field is set to the special value '-1' in two's complement.

The granule position of an audio data page is in units of PCM audio samples at a fixed rate of 48 kHz (per channel; a stereo stream's granule position does not increment at twice the speed of a mono stream). It is possible to run an Opus decoder at other sampling rates, but all Opus packets encode samples at a sampling rate that evenly divides 48 kHz. Therefore, the value in the granule position field always counts samples assuming a 48 kHz decoding rate, and the rest of this specification makes the same assumption.

The duration of an Opus packet as defined in [RFC6716] can be any multiple of 2.5 ms, up to a maximum of 120 ms. This duration is encoded in the TOC sequence at the beginning of each packet. The number of samples returned by a decoder corresponds to this duration exactly, even for the first few packets. For example, a 20 ms packet fed to a decoder running at 48 kHz will always return 960 samples. A demuxer can parse the TOC sequence at the beginning of each Ogg packet to work backwards or forwards from a packet with a known granule position (i.e., the last packet completed on some page) in order to assign granule positions to every packet, or even every individual sample. The one exception is the last page in the stream, as described below.

All other pages with completed packets after the first MUST have a granule position equal to the number of samples contained in packets that complete on that page plus the granule position of the most recent page with completed packets. This guarantees that a demuxer can assign individual packets the same granule position when working forwards as when working backwards. For this to work, there cannot be any gaps.

4.1. Repairing Gaps in Real-Time Streams

In order to support capturing a real-time stream that has lost or not transmitted packets, a multiplexer (muxer) SHOULD emit packets that explicitly request the use of Packet Loss Concealment (PLC) in place of the missing packets. Implementations that fail to do so still MUST NOT increment the granule position for a page by anything other than the number of samples contained in packets that actually complete on that page.

Only gaps that are a multiple of 2.5 ms are repairable, as these are the only durations that can be created by packet loss or discontinuous transmission. Muxers need not handle other gap sizes. Creating the necessary packets involves synthesizing a TOC byte (defined in Section 3.1 of [RFC6716]) -- and whatever additional internal framing is needed -- to indicate the packet duration for each stream. The actual length of each missing Opus frame inside the packet is zero bytes, as defined in Section 3.2.1 of [RFC6716].

Zero-byte frames MAY be packed into packets using any of codes 0, 1, 2, or 3. When successive frames have the same configuration, the higher code packings reduce overhead. Likewise, if the TOC configuration matches, the muxer MAY further combine the empty frames with previous or subsequent nonzero-length frames (using code 2 or variable bitrate (VBR) code 3).

[RFC6716] does not impose any requirements on the PLC, but this section outlines choices that are expected to have a positive influence on most PLC implementations, including the reference implementation. Synthesized TOC sequences SHOULD maintain the same mode, audio bandwidth, channel count, and frame size as the previous packet (if any). This is the simplest and usually the most well-tested case for the PLC to handle and it covers all losses that do not include a configuration switch, as defined in Section 4.5 of [RFC6716].

When a previous packet is available, keeping the audio bandwidth and channel count the same allows the PLC to provide maximum continuity in the concealment data it generates. However, if the size of the gap is not a multiple of the most recent frame size, then the frame size will have to change for at least some frames. Such changes SHOULD be delayed as long as possible to simplify things for PLC implementations.

As an example, a 95 ms gap could be encoded as nineteen 5 ms frames in two bytes with a single constant bitrate (CBR) code 3 packet. If the previous frame size was 20 ms, using four 20 ms frames followed by three 5 ms frames requires 4 bytes (plus an extra byte of Ogg lacing overhead), but allows the PLC to use its well-tested steady state behavior for as long as possible. The total bitrate of the latter approach, including Ogg overhead, is about 0.4 kbps, so the impact on file size is minimal.

Changing modes is discouraged, since this causes some decoder implementations to reset their PLC state. However, SILK and Hybrid mode frames cannot fill gaps that are not a multiple of 10 ms. If switching to CELT mode is needed to match the gap size, a muxer SHOULD do so at the end of the gap to allow the PLC to function for as long as possible.

In the example above, if the previous frame was a 20 ms SILK mode frame, the better solution is to synthesize a packet describing four 20 ms SILK frames, followed by a packet with a single 10 ms SILK frame, and finally a packet with a 5 ms CELT frame, to fill the 95 ms gap. This also requires four bytes to describe the synthesized packet data (two bytes for a CBR code 3 and one byte each for two code 0 packets) but three bytes of Ogg lacing overhead are needed to mark the packet boundaries. At 0.6 kbps, this is still a minimal bitrate impact over a naive, low-quality solution.

Since medium-band audio is an option only in the SILK mode, wideband frames SHOULD be generated if switching from that configuration to CELT mode, to ensure that any PLC implementation that does try to migrate state between the modes will be able to preserve all of the available audio bandwidth.

4.2. Pre-skip

There is some amount of latency introduced during the decoding process, to allow for overlap in the CELT mode, stereo mixing in the SILK mode, and resampling. The encoder might have introduced additional latency through its own resampling and analysis (though the exact amount is not specified). Therefore, the first few samples produced by the decoder do not correspond to real input audio, but are instead composed of padding inserted by the encoder to compensate for this latency. These samples need to be stored and decoded, as Opus is an asymptotically convergent predictive codec, meaning the decoded contents of each frame depend on the recent history of decoder inputs. However, a player will want to skip these samples after decoding them.

A ’pre-skip’ field in the ID header (see Section 5.1) signals the number of samples that SHOULD be skipped (decoded but discarded) at the beginning of the stream, though some specific applications might have a reason for looking at that data. This amount need not be a multiple of 2.5 ms, MAY be smaller than a single packet, or MAY span the contents of several packets. These samples are not valid audio.

For example, if the first Opus frame uses the CELT mode, it will always produce 120 samples of windowed overlap-add data. However, the overlap data is initially all zeros (since there is no prior frame), meaning this cannot, in general, accurately represent the original audio. The SILK mode requires additional delay to account for its analysis and resampling latency. The encoder delays the original audio to avoid this problem.

The ’pre-skip’ field MAY also be used to perform sample-accurate cropping of already encoded streams. In this case, a value of at least 3840 samples (80 ms) provides sufficient history to the decoder that it will have converged before the stream’s output begins.

4.3. PCM Sample Position

The PCM sample position is determined from the granule position using the following formula:

```
       ’PCM sample position’ = ’granule position’ - ’pre-skip’
```

For example, if the granule position of the first audio data page is 59,971, and the pre-skip is 11,971, then the PCM sample position of the last decoded sample from that page is 48,000.

This can be converted into a playback time using the following formula:

$$\text{'playback time'} = \frac{\text{'PCM sample position'}}{48000.0}$$

The initial PCM sample position before any samples are played is normally ’0’. In this case, the PCM sample position of the first audio sample to be played starts at ’1’, because it marks the time on the clock _after_ that sample has been played, and a stream that is exactly one second long has a final PCM sample position of ’48000’, as in the example here.

Vorbis streams use a granule position smaller than the number of audio samples contained in the first audio data page to indicate that some of those samples are trimmed from the output (see [VORBIS-TRIM]). However, to do so, Vorbis requires that the first audio data page contains exactly two packets, in order to allow the decoder to perform PCM position adjustments before needing to return any PCM data. Opus uses the pre-skip mechanism for this purpose instead, since the encoder might introduce more than a single packet’s worth of latency, and since very large packets in streams with a very large number of channels might not fit on a single page.

4.4. End Trimming

The page with the ’end of stream’ flag set MAY have a granule position that indicates the page contains less audio data than would normally be returned by decoding up through the final packet. This is used to end the stream somewhere other than an even frame boundary. The granule position of the most recent audio data page with completed packets is used to make this determination, or ’0’ is used if there were no previous audio data pages with a completed packet. The difference between these granule positions indicates how many samples to keep after decoding the packets that completed on the final page. The remaining samples are discarded. The number of discarded samples SHOULD be no larger than the number decoded from the last packet.

4.5. Restrictions on the Initial Granule Position

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. However, it MUST NOT be smaller, unless that page has the ’end of stream’ flag set. Allowing a granule position larger than the number of samples allows the beginning of a stream to be cropped or a live stream to be joined without rewriting the granule position of all the remaining pages. This means that the PCM sample position just before the first sample to be played MAY be larger than ’0’. Synchronization when multiplexing with other logical streams still uses the PCM sample position relative to ’0’ to compute sample times. This does not affect the behavior of pre-skip: exactly ’pre-skip’ samples SHOULD be skipped from the beginning of the decoded output, even if the initial PCM sample position is greater than zero.

On the other hand, a granule position that is smaller than the number of decoded samples prevents a demuxer from working backwards to assign each packet or each individual sample a valid granule position, since granule positions are non-negative. An implementation MUST treat any stream as invalid if the granule position is smaller than the number of samples contained in packets that complete on the first audio data page with a completed packet, unless that page has the ’end of stream’ flag set. It MAY defer this action until it decodes the last packet completed on that page.

If that page has the ’end of stream’ flag set, a demuxer MUST treat any stream as invalid if its granule position is smaller than the ’pre-skip’ amount. This would indicate that there are more samples to be skipped from the initial decoded output than exist in the stream. If the granule position is smaller than the number of decoded samples produced by the packets that complete on that page, then a demuxer MUST use an initial granule position of ’0’, and can work forwards from ’0’ to timestamp individual packets. If the granule position is larger than the number of decoded samples available, then the demuxer MUST still work backwards as described above, even if the ’end of stream’ flag is set, to determine the initial granule position, and thus the initial PCM sample position. Both of these will be greater than ’0’ in this case.

4.6. Seeking and Pre-roll

Seeking in Ogg files is best performed using a bisection search for a page whose granule position corresponds to a PCM position at or before the seek target. With appropriately weighted bisection, accurate seeking can be performed in just one or two bisections on average, even in multi-gigabyte files. See [SEEKING] for an example of general implementation guidance.

When seeking within an Ogg Opus stream, an implementation SHOULD start decoding (and discarding the output) at least 3840 samples (80 ms) prior to the seek target in order to ensure that the output audio is correct by the time it reaches the seek target. This "pre-roll" is separate from, and unrelated to, the pre-skip used at the beginning of the stream. If the point 80 ms prior to the seek target comes before the initial PCM sample position, an implementation SHOULD start decoding from the beginning of the stream, applying pre-skip as normal, regardless of whether the pre-skip is larger or smaller than 80 ms, and then continue to discard samples to reach the seek target (if any).

5. Header Packets

An Ogg Opus logical stream contains exactly two mandatory header packets: an identification header and a comment header.

5.1. Identification Header

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |      ’O’      |      ’p’      |      ’u’      |      ’s’      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |      ’H’      |      ’e’      |      ’a’      |      ’d’      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Version = 1  | Channel Count |           Pre-skip            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                     Input Sample Rate (Hz)                    |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Output Gain (Q7.8 in dB)    | Mapping Family|               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               :
 |                                                               |
 :               Optional Channel Mapping Table...               :
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: ID Header Packet

The fields in the identification (ID) header have the following meaning:

1. Magic Signature:

   This is an 8-octet (64-bit) field that allows codec identification and is human readable. It contains, in order, the magic numbers:

   0x4F ’O’

   0x70 ’p’

   0x75 ’u’

0x73 ’s’

0x48 ’H’

0x65 ’e’

0x61 ’a’

0x64 ’d’

Starting with "Op" helps distinguish it from audio data packets, as this is an invalid TOC sequence.

2. Version (8 bits, unsigned):

   The version number MUST always be ’1’ for this version of the encapsulation specification. Implementations SHOULD treat streams where the upper four bits of the version number match that of a recognized specification as backwards compatible with that specification. That is, the version number can be split into "major" and "minor" version sub-fields, with changes to the minor sub-field (in the lower four bits) signaling compatible changes. For example, an implementation of this specification SHOULD accept any stream with a version number of ’15’ or less, and SHOULD assume any stream with a version number ’16’ or greater is incompatible. The initial version ’1’ was chosen to keep implementations from relying on this octet as a null terminator for the "OpusHead" string.

3. Output Channel Count ’C’ (8 bits, unsigned):

   This is the number of output channels. This might be different than the number of encoded channels, which can change on a packet-by-packet basis. This value MUST NOT be zero. The maximum allowable value depends on the channel mapping family, and might be as large as 255. See Section 5.1.1 for details.

4. Pre-skip (16 bits, unsigned, little endian):

   This is the number of samples (at 48 kHz) to discard from the decoder output when starting playback, and also the number to subtract from a page’s granule position to calculate its PCM sample position. When cropping the beginning of existing Ogg Opus streams, a pre-skip of at least 3,840 samples (80 ms) is RECOMMENDED to ensure complete convergence in the decoder.

5. Input Sample Rate (32 bits, unsigned, little endian):

   This is the sample rate of the original input (before encoding), in Hz. This field is _not_ the sample rate to use for playback of the encoded data.

   Opus can switch between internal audio bandwidths of 4, 6, 8, 12, and 20 kHz. Each packet in the stream can have a different audio bandwidth. Regardless of the audio bandwidth, the reference decoder supports decoding any stream at a sample rate of 8, 12, 16, 24, or 48 kHz. The original sample rate of the audio passed to the encoder is not preserved by the lossy compression.

   An Ogg Opus player SHOULD select the playback sample rate according to the following procedure:

   1. If the hardware supports 48 kHz playback, decode at 48 kHz.

   2. Otherwise, if the hardware’s highest available sample rate is a supported rate, decode at this sample rate.

   3. Otherwise, if the hardware’s highest available sample rate is less than 48 kHz, decode at the next higher Opus supported rate above the highest available hardware rate and resample.

   4. Otherwise, decode at 48 kHz and resample.

   However, the ’input sample rate’ field allows the muxer to pass the sample rate of the original input stream as metadata. This is useful when the user requires the output sample rate to match the input sample rate. For example, when not playing the output, an implementation writing PCM format samples to disk might choose to resample the audio back to the original input sample rate to reduce surprise to the user, who might reasonably expect to get back a file with the same sample rate.

   A value of zero indicates "unspecified". Muxers SHOULD write the actual input sample rate or zero, but implementations that do something with this field SHOULD take care to behave sanely if given crazy values (e.g., do not actually upsample the output to 10 MHz if requested). Implementations SHOULD support input sample rates between 8 kHz and 192 kHz (inclusive). Rates outside this range MAY be ignored by falling back to the default rate of 48 kHz instead.

6. Output Gain (16 bits, signed, little endian):

   This is a gain to be applied when decoding. It is 20*log10 of the factor by which to scale the decoder output to achieve the desired playback volume, stored in a 16-bit, signed, two’s complement fixed-point value with 8 fractional bits (i.e., Q7.8 [Q-NOTATION]).

   To apply the gain, an implementation could use the following:

   ```
   sample *= pow(10, output_gain/(20.0*256))
   ```

   where ’output_gain’ is the raw 16-bit value from the header.

   Players and media frameworks SHOULD apply it by default. If a player chooses to apply any volume adjustment or gain modification, such as the R128_TRACK_GAIN (see Section 5.2), the adjustment MUST be applied in addition to this output gain in order to achieve playback at the normalized volume.

   A muxer SHOULD set this field to zero, and instead apply any gain prior to encoding, when this is possible and does not conflict with the user’s wishes. A nonzero output gain indicates the gain was adjusted after encoding, or that a user wished to adjust the gain for playback while preserving the ability to recover the original signal amplitude.

   Although the output gain has enormous range (+/- 128 dB, enough to amplify inaudible sounds to the threshold of physical pain), most applications can only reasonably use a small portion of this range around zero. The large range serves in part to ensure that gain can always be losslessly transferred between OpusHead and R128 gain tags (see below) without saturating.

7. Channel Mapping Family (8 bits, unsigned):

   This octet indicates the order and semantic meaning of the output channels.

   Each currently specified value of this octet indicates a mapping family, which defines a set of allowed channel counts, and the ordered set of channel names for each allowed channel count. The details are described in Section 5.1.1.

8. Channel Mapping Table:

   This table defines the mapping from encoded streams to output channels. Its contents are specified in Section 5.1.1.

All fields in the ID headers are REQUIRED, except for ’channel mapping table’, which MUST be omitted when the channel mapping family is 0, but is REQUIRED otherwise. Implementations SHOULD treat a stream as invalid if it contains an ID header that does not have enough data for these fields, even if it contain a valid ’magic signature’. Future versions of this specification, even backwards-compatible versions, might include additional fields in the ID header. If an ID header has a compatible major version, but a larger minor version, an implementation MUST NOT treat it as invalid for containing additional data not specified here, provided it still completes on the first page.

5.1.1. Channel Mapping

An Ogg Opus stream allows mapping one number of Opus streams (N) to a possibly larger number of decoded channels (M + N) to yet another number of output channels (C), which might be larger or smaller than the number of decoded channels. The order and meaning of these channels are defined by a channel mapping, which consists of the ’channel mapping family’ octet and, for channel mapping families other than family 0, a ’channel mapping table’, as illustrated in Figure 3.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
                                                 +-+-+-+-+-+-+-+-+
                                                 | Stream Count  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 | Coupled Count |              Channel Mapping...               :
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Channel Mapping Table

The fields in the channel mapping table have the following meaning:

1. Stream Count ’N’ (8 bits, unsigned):

   This is the total number of streams encoded in each Ogg packet. This value is necessary to correctly parse the packed Opus packets inside an Ogg packet, as described in Section 3. This value MUST NOT be zero, as without at least one Opus packet with a valid TOC sequence, a demuxer cannot recover the duration of an Ogg packet.

   For channel mapping family 0, this value defaults to 1, and is not coded.

2. Coupled Stream Count ’M’ (8 bits, unsigned):

   This is the number of streams whose decoders are to be configured to produce two channels (stereo). This MUST be no larger than the total number of streams, N.

   Each packet in an Opus stream has an internal channel count of 1 or 2, which can change from packet to packet. This is selected by the encoder depending on the bitrate and the audio being encoded. The original channel count of the audio passed to the encoder is not necessarily preserved by the lossy compression.

   Regardless of the internal channel count, any Opus stream can be decoded as mono (a single channel) or stereo (two channels) by appropriate initialization of the decoder. The ’coupled stream count’ field indicates that the decoders for the first M Opus streams are to be initialized for stereo (two-channel) output, and the remaining (N - M) decoders are to be initialized for mono (a single channel) only. The total number of decoded channels, (M + N), MUST be no larger than 255, as there is no way to index more channels than that in the channel mapping.

   For channel mapping family 0, this value defaults to (C - 1) (i.e., 0 for mono and 1 for stereo), and is not coded.

3. Channel Mapping (8*C bits):

   This contains one octet per output channel, indicating which decoded channel is to be used for each one. Let ’index’ be the value of this octet for a particular output channel. This value MUST either be smaller than (M + N) or be the special value 255. If ’index’ is less than 2*M, the output MUST be taken from decoding stream (’index’/2) as stereo and selecting the left channel if ’index’ is even, and the right channel if ’index’ is odd. If ’index’ is 2*M or larger, but less than 255, the output MUST be taken from decoding stream (’index’ - M) as mono. If ’index’ is 255, the corresponding output channel MUST contain pure silence.

   The number of output channels, C, is not constrained to match the number of decoded channels (M + N). A single index value MAY appear multiple times, i.e., the same decoded channel might be mapped to multiple output channels. Some decoded channels might not be assigned to any output channel, as well.

For channel mapping family 0, the first index defaults to 0, and if C == 2, the second index defaults to 1. Neither index is coded.

After producing the output channels, the channel mapping family determines the semantic meaning of each one. There are three defined mapping families in this specification.

5.1.1.1. Channel Mapping Family 0

Allowed numbers of channels: 1 or 2. RTP mapping. This is the same channel interpretation as [RFC7587].

- 1 channel: monophonic (mono).
- 2 channels: stereo (left, right).

Special mapping: This channel mapping family also indicates that the content consists of a single Opus stream that is stereo if and only if C == 2, with stream index 0 mapped to output channel 0 (mono, or left channel) and stream index 1 mapped to output channel 1 (right channel) if stereo. When the ’channel mapping family’ octet has this value, the channel mapping table MUST be omitted from the ID header packet.

5.1.1.2. Channel Mapping Family 1

Allowed numbers of channels: 1...8. Vorbis channel order (see below).

Each channel is assigned to a speaker location in a conventional surround arrangement. Specific locations depend on the number of channels, and are given below in order of the corresponding channel indices.

- 1 channel: monophonic (mono).
- 2 channels: stereo (left, right).
- 3 channels: linear surround (left, center, right).
- 4 channels: quadraphonic (front left, front right, rear left, rear right).
- 5 channels: 5.0 surround (front left, front center, front right, rear left, rear right).

- 6 channels: 5.1 surround (front left, front center, front right, rear left, rear right, LFE).

- 7 channels: 6.1 surround (front left, front center, front right, side left, side right, rear center, LFE).

- 8 channels: 7.1 surround (front left, front center, front right, side left, side right, rear left, rear right, LFE).

This set of surround options and speaker location orderings is the same as those used by the Vorbis codec [VORBIS-MAPPING]. The ordering is different from the one used by the WAVE [WAVE-MULTICHANNEL] and Free Lossless Audio Codec (FLAC) [FLAC] formats, so correct ordering requires permutation of the output channels when decoding to or encoding from those formats. "LFE" here refers to a Low Frequency Effects channel, often mapped to a subwoofer with no particular spatial position. Implementations SHOULD identify "side" or "rear" speaker locations with "surround" and "back" as appropriate when interfacing with audio formats or systems that prefer that terminology.

5.1.1.3. Channel Mapping Family 255

Allowed numbers of channels: 1...255. No defined channel meaning.

Channels are unidentified. General-purpose players SHOULD NOT attempt to play these streams. Offline implementations MAY deinterleave the output into separate PCM files, one per channel. Implementations SHOULD NOT produce output for channels mapped to stream index 255 (pure silence) unless they have no other way to indicate the index of non-silent channels.

5.1.1.4. Undefined Channel Mappings

The remaining channel mapping families (2...254) are reserved. A demuxer implementation encountering a reserved ’channel mapping family’ value SHOULD act as though the value is 255.

5.1.1.5. Downmixing

An Ogg Opus player MUST support any valid channel mapping with a channel mapping family of 0 or 1, even if the number of channels does not match the physically connected audio hardware. Players SHOULD perform channel mixing to increase or reduce the number of channels as needed.

Implementations MAY use the matrices in Figures 4 through 9 to implement downmixing from multichannel files using channel mapping family 1 (Section 5.1.1.2), which are known to give acceptable results for stereo. Matrices for 3 and 4 channels are normalized so each coefficient row sums to 1 to avoid clipping. For 5 or more channels, they are normalized to 2 as a compromise between clipping and dynamic range reduction.

In these matrices the front-left and front-right channels are generally passed through directly. When a surround channel is split between both the left and right stereo channels, coefficients are chosen so their squares sum to 1, which helps preserve the perceived intensity. Rear channels are mixed more diffusely or attenuated to maintain focus on the front channels.

```
L output = ( 0.585786 * left + 0.414214 * center                    )
R output = (                   0.414214 * center + 0.585786 * right )
```

Exact coefficient values are 1 and 1/sqrt(2), multiplied by 1/(1 + 1/sqrt(2)) for normalization.

Figure 4: Stereo Downmix Matrix for the Linear Surround Channel Mapping

```
/          \   /                                     \ / FL \
| L output |   | 0.422650 0.000000 0.366025 0.211325 | | FR |
| R output | = | 0.000000 0.422650 0.211325 0.366025 | | RL |
\          /   \                                     / \ RR /
```

Exact coefficient values are 1, sqrt(3)/2 and 1/2, multiplied by 1/(1 + sqrt(3)/2 + 1/2) for normalization.

Figure 5: Stereo Downmix Matrix for the Quadraphonic Channel Mapping

```
                                                            / FL \
/   \   /                                              \ | FC |
| L |   | 0.650802 0.460186 0.000000 0.563611 0.325401 | | FR |
| R | = | 0.000000 0.460186 0.650802 0.325401 0.563611 | | RL |
\   /   \                                              / | RR |
                                                            \    /
```

Exact coefficient values are 1, 1/sqrt(2), sqrt(3)/2 and 1/2, multiplied by 2/(1 + 1/sqrt(2) + sqrt(3)/2 + 1/2) for normalization.

Figure 6: Stereo Downmix Matrix for the 5.0 Surround Mapping

$$\begin{pmatrix} L \\ R \end{pmatrix} = \begin{pmatrix} 0.529067 & 0.374107 & 0.000000 & 0.458186 & 0.264534 & 0.374107 \\ 0.000000 & 0.374107 & 0.529067 & 0.264534 & 0.458186 & 0.374107 \end{pmatrix} \begin{pmatrix} FL \\ FC \\ FR \\ RL \\ RR \\ LFE \end{pmatrix}$$

Exact coefficient values are 1, 1/sqrt(2), sqrt(3)/2 and 1/2, multiplied by 2/(1 + 1/sqrt(2) + sqrt(3)/2 + 1/2 + 1/sqrt(2)) for normalization.

Figure 7: Stereo Downmix Matrix for the 5.1 Surround Mapping

$$\begin{pmatrix} 0.455310 & 0.321953 & 0.000000 & 0.394310 & 0.227655 & 0.278819 & 0.321953 \\ 0.000000 & 0.321953 & 0.455310 & 0.227655 & 0.394310 & 0.278819 & 0.321953 \end{pmatrix}$$

Exact coefficient values are 1, 1/sqrt(2), sqrt(3)/2, 1/2 and sqrt(3)/2/sqrt(2), multiplied by 2/(1 + 1/sqrt(2) + sqrt(3)/2 + 1/2 + sqrt(3)/2/sqrt(2) + 1/sqrt(2)) for normalization. The coefficients are in the same order as in Section 5.1.1.2 and the matrices above.

Figure 8: Stereo Downmix Matrix for the 6.1 Surround Mapping

$$\begin{pmatrix} .388631 & .274804 & .000000 & .336565 & .194316 & .336565 & .194316 & .274804 \\ .000000 & .274804 & .388631 & .194316 & .336565 & .194316 & .336565 & .274804 \end{pmatrix}$$

Exact coefficient values are 1, 1/sqrt(2), sqrt(3)/2 and 1/2, multiplied by 2/(2 + 2/sqrt(2) + sqrt(3)) for normalization. The coefficients are in the same order as in Section 5.1.1.2 and the matrices above.

Figure 9: Stereo Downmix Matrix for the 7.1 Surround Mapping

5.2. Comment Header

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |      ’O’      |      ’p’      |      ’u’      |      ’s’      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |      ’T’      |      ’a’      |      ’g’      |      ’s’      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                     Vendor String Length                      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               |
     :                        Vendor String...                       :
     |                                                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                   User Comment List Length                    |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                 User Comment #0 String Length                 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               |
     :                   User Comment #0 String...                   :
     |                                                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                 User Comment #1 String Length                 |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     :                                                               :
```

Figure 10: Comment Header Packet

The comment header consists of a 64-bit ’magic signature’ field, followed by data in the same format as the [VORBIS-COMMENT] header used in Ogg Vorbis, except (like Ogg Theora and Speex) the final ’framing bit’ specified in the Vorbis specification is not present.

1. Magic Signature:

   This is an 8-octet (64-bit) field that allows codec identification and is human readable. It contains, in order, the magic numbers:

   0x4F ’O’

   0x70 ’p’

   0x75 ’u’

   0x73 ’s’

   0x54 ’T’

   0x61 ’a’

   0x67 ’g’

   0x73 ’s’

   Starting with "Op" helps distinguish it from audio data packets, as this is an invalid TOC sequence.

2. Vendor String Length (32 bits, unsigned, little endian):

   This field gives the length of the following vendor string, in octets. It MUST NOT indicate that the vendor string is longer than the rest of the packet.

3. Vendor String (variable length, UTF-8 vector):

   This is a simple human-readable tag for vendor information, encoded as a UTF-8 string [RFC3629]. No terminating null octet is necessary.

   This tag is intended to identify the codec encoder and encapsulation implementations, for tracing differences in technical behavior. User-facing applications can use the ’ENCODER’ user comment tag to identify themselves.

4. User Comment List Length (32 bits, unsigned, little endian):

   This field indicates the number of user-supplied comments. It MAY indicate there are zero user-supplied comments, in which case there are no additional fields in the packet. It MUST NOT indicate that there are so many comments that the comment string lengths would require more data than is available in the rest of the packet.

5. User Comment #i String Length (32 bits, unsigned, little endian):

   This field gives the length of the following user comment string, in octets. There is one for each user comment indicated by the 'user comment list length' field. It MUST NOT indicate that the string is longer than the rest of the packet.

6. User Comment #i String (variable length, UTF-8 vector):

   This field contains a single user comment encoded as a UTF-8 string [RFC3629]. There is one for each user comment indicated by the 'user comment list length' field.

The 'vendor string length' and 'user comment list length' fields are REQUIRED, and implementations SHOULD treat a stream as invalid if it contains a comment header that does not have enough data for these fields, or that does not contain enough data for the corresponding vendor string or user comments they describe. Making this check before allocating the associated memory to contain the data helps prevent a possible Denial-of-Service (DoS) attack from small comment headers that claim to contain strings longer than the entire packet or more user comments than could possibly fit in the packet.

Immediately following the user comment list, the comment header MAY contain zero-padding or other binary data that is not specified here. If the least-significant bit of the first byte of this data is 1, then editors SHOULD preserve the contents of this data when updating the tags, but if this bit is 0, all such data MAY be treated as padding, and truncated or discarded as desired. This allows informal experimentation with the format of this binary data until it can be specified later.

The comment header can be arbitrarily large and might be spread over a large number of Ogg pages. Implementations MUST avoid attempting to allocate excessive amounts of memory when presented with a very large comment header. To accomplish this, implementations MAY treat a stream as invalid if it has a comment header larger than

125,829,120 octets (120 MB), and MAY ignore individual comments that are not fully contained within the first 61,440 octets of the comment header.

5.2.1. Tag Definitions

The user comment strings follow the NAME=value format described by [VORBIS-COMMENT] with the same recommended tag names: ARTIST, TITLE, DATE, ALBUM, and so on.

Two new comment tags are introduced here:

First, an optional gain for track normalization:

R128_TRACK_GAIN=-573

representing the volume shift needed to normalize the track’s volume during isolated playback, in random shuffle, and so on. The gain is a Q7.8 fixed-point number in dB, as in the ID header’s ’output gain’ field. This tag is similar to the REPLAYGAIN_TRACK_GAIN tag in Vorbis [REPLAY-GAIN], except that the normal volume reference is the [EBU-R128] standard.

Second, an optional gain for album normalization:

R128_ALBUM_GAIN=111

representing the volume shift needed to normalize the overall volume when played as part of a particular collection of tracks. The gain is also a Q7.8 fixed-point number in dB, as in the ID header’s ’output gain’ field. The values ’-573’ and ’111’ given here are just examples.

An Ogg Opus stream MUST NOT have more than one of each of these tags, and, if present, their values MUST be an integer from -32768 to 32767, inclusive, represented in ASCII as a base 10 number with no whitespace. A leading ’+’ or ’-’ character is valid. Leading zeros are also permitted, but the value MUST be represented by no more than 6 characters. Other non-digit characters MUST NOT be present.

If present, R128_TRACK_GAIN and R128_ALBUM_GAIN MUST correctly represent the R128 normalization gain relative to the ’output gain’ field specified in the ID header. If a player chooses to make use of the R128_TRACK_GAIN tag or the R128_ALBUM_GAIN tag, it MUST apply those gains _in addition_ to the ’output gain’ value. If a tool modifies the ID header’s ’output gain’ field, it MUST also update or remove the R128_TRACK_GAIN and R128_ALBUM_GAIN comment tags if present. A muxer SHOULD place the gain it wants other tools to use by default into the 'output gain' field, and not the comment tag.

To avoid confusion with multiple normalization schemes, an Opus comment header SHOULD NOT contain any of the REPLAYGAIN_TRACK_GAIN, REPLAYGAIN_TRACK_PEAK, REPLAYGAIN_ALBUM_GAIN, or REPLAYGAIN_ALBUM_PEAK tags, unless they are only to be used in some context where there is guaranteed to be no such confusion. [EBU-R128] normalization is preferred to the earlier REPLAYGAIN schemes because of its clear definition and adoption by industry. Peak normalizations are difficult to calculate reliably for lossy codecs because of variation in excursion heights due to decoder differences. In the authors' investigations, they were not applied consistently or broadly enough to merit inclusion here.

6. Packet Size Limits

Technically, valid Opus packets can be arbitrarily large due to the padding format, although the amount of non-padding data they can contain is bounded. These packets might be spread over a similarly enormous number of Ogg pages. When encoding, implementations SHOULD limit the use of padding in audio data packets to no more than is necessary to make a VBR stream CBR, unless they have no reasonable way to determine what is necessary. Demuxers SHOULD treat audio data packets as invalid (treat them as if they were malformed Opus packets with an invalid TOC sequence) if they are larger than 61,440 octets per Opus stream, unless they have a specific reason for allowing extra padding. Such packets necessarily contain more padding than needed to make a stream CBR. Demuxers MUST avoid attempting to allocate excessive amounts of memory when presented with a very large packet. Demuxers MAY treat audio data packets as invalid or partially process them if they are larger than 61,440 octets in an Ogg Opus stream with channel mapping families 0 or 1. Demuxers MAY treat audio data packets as invalid or partially process them in any Ogg Opus stream if the packet is larger than 61,440 octets and also larger than 7,680 octets per Opus stream. The presence of an extremely large packet in the stream could indicate a memory exhaustion attack or stream corruption.

In an Ogg Opus stream, the largest possible valid packet that does not use padding has a size of (61,298*N - 2) octets. With 255 streams, this is 15,630,988 octets and can span up to 61,298 Ogg pages, all but one of which will have a granule position of -1. This is, of course, a very extreme packet, consisting of 255 streams, each containing 120 ms of audio encoded as 2.5 ms frames, each frame using the maximum possible number of octets (1275) and stored in the least efficient manner allowed (a VBR code 3 Opus packet). Even in such a packet, most of the data will be zeros as 2.5 ms frames cannot actually use all 1275 octets.

The largest packet consisting of entirely useful data is (15,326*N - 2) octets. This corresponds to 120 ms of audio encoded as 10 ms frames in either SILK or Hybrid mode, but at a data rate of over 1 Mbps, which makes little sense for the quality achieved.

A more reasonable limit is (7,664*N - 2) octets. This corresponds to 120 ms of audio encoded as 20 ms stereo CELT mode frames, with a total bitrate just under 511 kbps (not counting the Ogg encapsulation overhead). For channel mapping family 1, N = 8 provides a reasonable upper bound, as it allows for each of the 8 possible output channels to be decoded from a separate stereo Opus stream. This gives a size of 61,310 octets, which is rounded up to a multiple of 1,024 octets to yield the audio data packet size of 61,440 octets that any implementation is expected to be able to process successfully.

7. Encoder Guidelines

When encoding Opus streams, Ogg muxers SHOULD take into account the algorithmic delay of the Opus encoder.

In encoders derived from the reference implementation [RFC6716], the number of samples can be queried with

```
 opus_encoder_ctl(encoder_state, OPUS_GET_LOOKAHEAD(&delay_samples));
```

To achieve good quality in the very first samples of a stream, implementations MAY use linear predictive coding (LPC) extrapolation to generate at least 120 extra samples at the beginning to avoid the Opus encoder having to encode a discontinuous signal. For more information on linear prediction, see [LINEAR-PREDICTION]. For an input file containing ’length’ samples, the implementation SHOULD set the ’pre-skip’ header value to (delay_samples + extra_samples), encode at least (length + delay_samples + extra_samples) samples, and set the granule position of the last page to (length + delay_samples + extra_samples). This ensures that the encoded file has the same duration as the original, with no time offset. The best way to pad the end of the stream is to also use LPC extrapolation, but zero-padding is also acceptable.

7.1. LPC Extrapolation

The first step in LPC extrapolation is to compute linear prediction coefficients [LPC-SAMPLE]. When extending the end of the signal, order-N (typically with N ranging from 8 to 40) LPC analysis is performed on a window near the end of the signal. The last N samples are used as memory to an infinite impulse response (IIR) filter.

The filter is then applied on a zero input to extrapolate the end of the signal. Let 'a(k)' be the kth LPC coefficient and 'x(n)' be the nth sample of the signal. Each new sample past the end of the signal is computed as

$$x(n) = \sum_{k = 1}^{N} a(k)*x(n - k)$$

The process is repeated independently for each channel. It is possible to extend the beginning of the signal by applying the same process backward in time. When extending the beginning of the signal, it is best to apply a "fade in" to the extrapolated signal, e.g., by multiplying it by a half-Hanning window [HANNING].

7.2. Continuous Chaining

In some applications, such as Internet radio, it is desirable to cut a long stream into smaller chains, e.g., so the comment header can be updated. This can be done simply by separating the input streams into segments and encoding each segment independently. The drawback of this approach is that it creates a small discontinuity at the boundary due to the lossy nature of Opus. A muxer MAY avoid this discontinuity by using the following procedure:

1. Encode the last frame of the first segment as an independent frame by turning off all forms of inter-frame prediction. De-emphasis is allowed.

2. Set the granule position of the last page to a point near the end of the last frame.

3. Begin the second segment with a copy of the last frame of the first segment.

4. Set the 'pre-skip' value of the second stream in such a way as to properly join the two streams.

5. Continue the encoding process normally from there, without any reset to the encoder.

In encoders derived from the reference implementation, inter-frame prediction can be turned off by calling

```
opus_encoder_ctl(encoder_state, OPUS_SET_PREDICTION_DISABLED(1));
```

For best results, this implementation requires that prediction be explicitly enabled again before resuming normal encoding, even after a reset.

8. Security Considerations

Implementations of the Opus codec need to take appropriate security considerations into account, as outlined in [RFC4732]. This is just as much a problem for the container as it is for the codec itself. Malicious payloads and/or input streams can be used to attack codec implementations. Implementations MUST NOT overrun their allocated memory nor consume excessive resources when decoding payloads or processing input streams. Although problems in encoding applications are typically rarer, this still applies to a muxer, as vulnerabilities would allow an attacker to attack transcoding gateways.

Header parsing code contains the most likely area for potential overruns. It is important for implementations to ensure their buffers contain enough data for all of the required fields before attempting to read it (for example, for all of the channel map data in the ID header). Implementations would do well to validate the indices of the channel map, also, to ensure they meet all of the restrictions outlined in Section 5.1.1, in order to avoid attempting to read data from channels that do not exist.

To avoid excessive resource usage, we advise implementations to be especially wary of streams that might cause them to process far more data than was actually transmitted. For example, a relatively small comment header may contain values for the string lengths or user comment list length that imply that it is many gigabytes in size. Even computing the size of the required buffer could overflow a 32-bit integer, and actually attempting to allocate such a buffer before verifying it would be a reasonable size is a bad idea. After reading the user comment list length, implementations might wish to verify that the header contains at least the minimum amount of data for that many comments (4 additional octets per comment, to indicate each has a length of zero) before proceeding any further, again taking care to avoid overflow in these calculations. If allocating an array of pointers to point at these strings, the size of the pointers may be larger than 4 octets, potentially requiring a separate overflow check.

Another bug in this class we have observed more than once involves the handling of invalid data at the end of a stream. Often, implementations will seek to the end of a stream to locate the last timestamp in order to compute its total duration. If they do not find a valid capture pattern and Ogg page from the desired logical stream, they will back up and try again. If care is not taken to avoid re-scanning data that was already scanned, this search can quickly devolve into something with a complexity that is quadratic in the amount of invalid data.

In general, when seeking, implementations will wish to be cautious about the effects of invalid granule position values and ensure all algorithms will continue to make progress and eventually terminate, even if these are missing or out of order.

Like most other container formats, Ogg Opus streams SHOULD NOT be used with insecure ciphers or cipher modes that are vulnerable to known-plaintext attacks. Elements such as the Ogg page capture pattern and the ’magic signature’ fields in the ID header and the comment header all have easily predictable values, in addition to various elements of the codec data itself.

9. Content Type

An "Ogg Opus file" consists of one or more sequentially multiplexed segments, each containing exactly one Ogg Opus stream. The RECOMMENDED mime-type for Ogg Opus files is "audio/ogg".

If more specificity is desired, one MAY indicate the presence of Opus streams using the codecs parameter defined in [RFC6381] and [RFC5334], e.g.,

```
audio/ogg; codecs=opus
```

for an Ogg Opus file.

The RECOMMENDED filename extension for Ogg Opus files is ’.opus’.

When Opus is concurrently multiplexed with other streams in an Ogg container, one SHOULD use one of the "audio/ogg", "video/ogg", or "application/ogg" mime-types, as defined in [RFC5334]. Such streams are not strictly "Ogg Opus files" as described above, since they contain more than a single Opus stream per sequentially multiplexed segment, e.g., video or multiple audio tracks. In such cases, the '.opus' filename extension is NOT RECOMMENDED.

In either case, this document updates [RFC5334] to add "opus" as a codecs parameter value with char[8]: 'OpusHead' as Codec Identifier.

10. IANA Considerations

Per this document, IANA has updated the "Media Types" registry by adding .opus as a file extension for "audio/ogg" and adding itself as a reference alongside [RFC5334] for "audio/ogg", "video/ogg", and "application/ogg" Media Types.

This document defines a new registry "Opus Channel Mapping Families" to indicate how the semantic meanings of the channels in a multichannel Opus stream are described. IANA has created a new namespace of "Opus Channel Mapping Families". This registry is listed on the IANA Matrix. Modifications to this registry follow the "Specification Required" registration policy as defined in [RFC5226]. Each registry entry consists of a Channel Mapping Family Number, which is specified in decimal in the range 0 to 255, inclusive, and a Reference (or list of references). Each Reference must point to sufficient documentation to describe what information is coded in the Opus identification header for this channel mapping family, how a demuxer determines the stream count ('N') and coupled stream count ('M') from this information, and how it determines the proper interpretation of each of the decoded channels.

This document defines three initial assignments for this registry.

Value	Reference
0	RFC 7845, Section 5.1.1.1
1	RFC 7845, Section 5.1.1.2
255	RFC 7845, Section 5.1.1.3

The designated expert will determine if the Reference points to a specification that meets the requirements for permanence and ready availability laid out in [RFC5226] and whether it specifies the information described above with sufficient clarity to allow interoperable implementations.

11. References

11.1. Normative References

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

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[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>.

[RFC6381] Gellens, R., Singer, D., and P. Frojdh, "The 'Codecs' and 'Profiles' Parameters for "Bucket" Media Types", RFC 6381, DOI 10.17487/RFC6381, August 2011, <https://www.rfc-editor.org/info/rfc6381>.

[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>.

[EBU-R128] EBU Technical Committee, "Loudness Recommendation EBU R128", August 2011, <https://tech.ebu.ch/loudness>.

[VORBIS-COMMENT] Montgomery, C., "Ogg Vorbis I Format Specification: Comment Field and Header Specification", July 2002, <https://www.xiph.org/vorbis/doc/v-comment.html>.

11.2. Informative References

[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>.

[RFC7587] Spittka, J., Vos, K., and JM. Valin, "RTP Payload Format for the Opus Speech and Audio Codec", RFC 7587, DOI 10.17487/RFC7587, June 2015, <https://www.rfc-editor.org/info/rfc7587>.

[FLAC] Coalson, J., "FLAC - Free Lossless Audio Codec Format Description", January 2008, <https://xiph.org/flac/format.html>.

[HANNING] Wikipedia, "Hann window", February 2016, <https://en.wikipedia.org/w/index.php?title=Window_function&oldid=703074467#Hann_.28Hanning.29_window>.

[LINEAR-PREDICTION] Wikipedia, "Linear Predictive Coding", October 2015, <https://en.wikipedia.org/w/index.php?title=Linear_predictive_coding&oldid=687498962>.

[LPC-SAMPLE] Degener, J. and C. Bormann, "Autocorrelation LPC coeff generation algorithm (Vorbis source code)", November 1994, <https://svn.xiph.org/trunk/vorbis/lib/lpc.c>.

[Q-NOTATION] Wikipedia, "Q (number format)", December 2015, <https://en.wikipedia.org/w/index.php?title=Q_%28number_format%29&oldid=697252615>.

[REPLAY-GAIN] Parker, C. and M. Leese, "VorbisComment: Replay Gain", June 2009, <https://wiki.xiph.org/VorbisComment#Replay_Gain>.

[SEEKING] Pfeiffer, S., Parker, C., and G. Maxwell, "Granulepos Encoding and How Seeking Really Works", May 2012, <https://wiki.xiph.org/Seeking>.

[VORBIS-MAPPING]
Montgomery, C., "The Vorbis I Specification, Section 4.3.9 Output Channel Order", January 2010, <https://www.xiph.org/vorbis/doc/Vorbis_I_spec.html#x1-810004.3.9>.

[VORBIS-TRIM]
Montgomery, C., "The Vorbis I Specification, Appendix A: Embedding Vorbis into an Ogg stream", November 2008, <https://xiph.org/vorbis/doc/Vorbis_I_spec.html#x1-132000A.2>.

[WAVE-MULTICHANNEL]
Microsoft Corporation, "Multiple Channel Audio Data and WAVE Files", March 2007, <https://msdn.microsoft.com/en-us/windows/hardware/gg463006.aspx>.

Acknowledgments

Thanks to Ben Campbell, Joel M. Halpern, Mark Harris, Greg Maxwell, Christopher "Monty" Montgomery, Jean-Marc Valin, Stephan Wenger, and Mo Zanaty for their valuable contributions to this document. Additional thanks to Andrew D’Addesio, Greg Maxwell, and Vincent Penquerc’h for their feedback based on early implementations.

Authors’ Addresses

Timothy B. Terriberry
Mozilla Corporation
331 E. Evelyn Ave.
Mountain View, CA 94041
United States

Phone: +1 650 903-0800
Email: tterribe@xiph.org

Ron Lee
Voicetronix
246 Pulteney Street, Level 1
Adelaide, SA 5000
Australia

Phone: +61 8 8232 9112
Email: ron@debian.org

Ralph Giles
Mozilla Corporation
163 West Hastings Street
Vancouver, BC V6B 1H5
Canada

Phone: +1 778 785 1540
Email: giles@xiph.org