mlcodec
Internet-Draft
Updates: 6716 (if approved)
Intended status: Standards Track
Expires: 23 April 2026

JM. Valin
Google

20 October 2025

Scalable Quality Extension for the Opus Codec (Opus HD)
draft-ietf-mlcodec-opus-scalable-quality-extension-01

Abstract

This document updates RFC6716 to add support for a scalable quality layer.

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 23 April 2026.

Table of Contents

1. Introduction

This document updates RFC6716 to add support for a scalable quality extension layer. Implementations conforming to this document will be referred to as Opus HD.

1.1. Requirements Language

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.

2. Scalable Quality Extension

The Opus codec was designed to operate at sampling frequencies up to 48 kHz, with an audio bandwidth up to 20 kHz. The CELT mode that is used for high bitrate coding uses vector quantization with a mostly implicit bit allocation system that is dictated by the bitstream definition. Opus can allocate up to 8 bits per MDCT bin in some of the bands.

While Opus capabilities listed above are sufficient to achieve perceptually transparent audio coding, there is a use for codecs that scale beyond those specs. That includes the current market for 24-bit/96 kHz codecs, but also any application where the intended recipient is not (only) a human being, e.g. ultra-sonic applications.

This document proposes a scalable quality extension layer that both increases the resolution of existing Opus quantizers below 20 kHz, and defines a way of coding audio above 20 kHz, with a sampling rate of 96 kHz. The extension is designed to be forward and backward compatible with [RFC6716]. All extra bits use the Opus extension mechanism defined in [opus-extension] and a 96 kHz decoder is designed to decode a regular 48 kHz RFC 6716 stream and vice versa.

The code implementing this draft is available on the main branch of the Opus repository at https://gitlab.xiph.org/xiph/opus/ and requires building with --enable-qext (or defining ENABLE_QEXT).

2.1. Extended resolution

To reduce the coding error, we need to increase the resolution for 3 different quantizers: the fine energy quantizer (scalar), the band pyramid vector quantizer (PVQ), and the band splitting angle quantizer. We also introduce a new cubic quantizer that scales to higher bit depths than PVQ. To preserve compatibility, all of the bits extending the Opus resolution are stored in the extension payload.

2.1.1. Fine energy quantizer

For each band we can increase the resolution of the fine energy quantizer by adding extra bits. The extra bits are added in the same way as the regular fine energy quantizer adds resolution on top of the coarse energy quantizer.

2.1.2. PVQ

From a size-K PVQ codebook in N dimensions we can create an extended codebook of size u*K, where u is always odd and selected as 2^b-1, where b is the extra depth. Let y_i be the (integer) value for dimension i of the size-K codebook and z_i be the corresponding value for the size-u*K codebook. We define a refinement r_i = z_i - u*y_i where |r_i| < u. In the N=2 special case, |r_i| < (u+1)/2. Only the refinement r_i needs to be coded since the regular Opus bitstream already includes y_i. The last residual value r_{N-1} does not need to be coded since it’s value can be inferred from the other values and the knowledge that the sum of the absolute values is u*K. The only exception is when y_{N-1}=0, in which case, a single sign bit is

coded, but the magnitude is still inferred.

Even though |r_i| < u, smaller values or r_i are more likely, so we benefit from entropy coding r_i. We assume that the likelihood of for |r_i| < (u+1)/2 is 7/8 and use that probability for decoding a "large" flag. If large=0, we decode b bits and and subtract u/2 to get r_i. If large=1, we decode a sign bit, followed by an integer with b-1 bits to which we add u/2+1 and apply the sign.

2.1.3. Angle quantizer

When using mid-side stereo or when splitting a band, we code an angle representing the atan of two sub-vectors' magnitude ratio. The standard Opus encoder can code angles with up to 8 bits. In a similar way to how we define the PVQ refinement, we pick u = 2^b-1 where u is the number of (equidistant) extra quantization levels to be added between each of the original levels. We code a unit symbol between 0 and u-1, where 0 is almost mid-point to the previous (lower) quantization level, u-1 is almost mid-point to the next (higher) level, and (u+1)/2 perfectly lines up with with the originally selected quantization of the standard Opus layer.

2.1.4. Cubic quantizer

The existing Opus PVQ only scales up to 32-bit codebooks. For cases where there is no PVQ in the base Opus layer, we define a new cubic quantizer. Whereas the PVQ codebook is defined as a reflected simplex warped onto the unit sphere, the cubic quantizer warps an N-dimensional cubic shell to the same unit sphere. Cubic codewords specify which face of the cube the vector lies on by coding the dimension and sign of the largest component (using 1+log2(N) bits). The face of an N-dimensional hyper-cube shell is a full N-1-dimensional cube and can be coded with N-1 scalar values from 0 to Q-1 ((N-1)*log2(Q) bits). We use even Q (Q=2^b) for non-transient bands (B==1) and odd Q (Q=2^b-1) for transient bands (B>1).

2.2. Extended frequency range

To extend the audio bandwidth, we need to define more frequency bands. Because psychoacoustics is no longer involved past 20 kHz, all new bands are defined to have a width of 2 kHz. Therefore, when encoding 48-kHz content we add 2 extra bands and when encoding 96-kHz content, we add 14 extra bands. A flag is encoded to specify whether 2 or 14 bands are added. The decoder uses that flag to know how many bands to decode, regardless of whether decoding at 48 or 96 kHz.

2.3. Bit allocation

The allocation of the extra bit depth b is explicitly signaled for each band at a time, using a resolution of 1/4 bit depth between 0 and a band-dependent cap C, where C=12 for bands up to 20 kHz, and C=14 for the added bands. For band b_i, we use entropy coding to give a higher probability to three different cases: b_i=0, b_i=C, and b_i=b_{i-1}. In the case where b_{i-1} is either 0 or C, we merge two of the probabilities. The ICDF for the general case is {120, 112, 70, 0}, where the first symbol means b_i=0, the second means b_i=C, the third means b_i=b_{i-1}, and the last symbol means that b_i is equal to 1 plus a unit value coded from 0 to C-1. For b_{i-1} = 0, we use the ICDF {64, 50, 0} and for b_{i-1}=C, we use {110, 60, 0}, where the last symbol always means that a unit is coded. We start with b_{-1} = 0.

Given b_i, the number of extra energy bits is given by (b_i+3)/4. The number of 1/8 bits (BITRES) allocated for PVQ refinement and/or cubic codebook bits is given by ((W-1)*C * b_i * 8 + 2)/4, where W is the number of bins in the band and C is the number of channels.

2.4. Time-domain processing at 96 kHz

CELT includes two time-domain filter pairs that require updating for 96 kHz: the preemphasis/deempahsis filters, as well as the pitch prefilter/postfilter. The CELT deemphasis filter is currently defined as D(z)=1/(1 - a1*z^-1) for a 48 kHz signal, where a1=27853/32768. To obtain approximately the same response in the 0-20 kHz range using a sampling rate of 96 kHz, we instead use D(z)=g*(1 - b1*z^-1)/(1 - a1*z^-1), where g=5415/8192, b1=7209/32768, a1=30245/32768.

For the pitch pre-filter/post-filter, we use zero-insertion upsampling of the 48 kHz filters, which results in the same frequency response below 24 kHz and a "folded" image above 24 kHz. For example, if for a pitch period T (in 48 kHz units) the postfilter was P(z)=1/(1 - a0*z^-T+1 - a1*z^-T - a2*z^-T-1), then for the same pitch, the 96 kHz filter becomes P(z)=1/(1 - a0*z^-2T+2 - a1*z^-2T - a2*z^-2T-2).

3. Format

The extension payload is entropy-coded in the following order

Symbol(s)	PDF/Description
96 kHz flag	{1, 1}/2
Intensity stereo	uint
Dual stereo	{1, 1}/2
Intra coarse energy	{7, 1}/2
Coarse energy (high bands)	
Bit allocation	Section 2.3
Fine energy (low bands)	Section 2.1.1
PVQ refinement	Section 2.1.2, Section 2.1.3
Fine energy (high bands)	Section 2.3
PVQ and cubic codebook (high bands)	Section 2.1.4

Table 1

4. Conformance

This section defines some tests for evaluating Opus HD conformance. The evaluations are based on test vectors, along with a custom-made comparison tool named qext_compare and derived from the original opus_compare tool from RFC 6716.

4.1. Decoder

For a decoder to conform to this specification, its output MUST be within the specified bounds for all testvectors when compared using qext_compare. Two sets of testvectors are provided. The first, qext_vector01.bit through qext_vector06.bit are high-quality 1024 kb/s bitstreams for which the decoder target files are qext_vector01.f32 through qext_vector06.f32.

The testvectors can be downloaded at https://media.xiph.org/opus/ietf/opushd_testvectors.tar.gz.

Using the reference decoder, a testvector can be decoded as:

```
% opus_demo -d 96000 2 -f32 qext_vector01.f32 qext_test01.f32
```

Then the output can be compared to the reference with specific thresholds:

```
% qext_compare -s -f32 -thresholds 0.05 0.1 0.1 \
                      qext_vector01dec.f32 qext_test01.f32
```

which will output "Comparison PASSED" if the tested decoder is close enough to the target output.

The second set of testvectors are meant to test corner cases. The bitstream files are qext_vector01fuzz.bit through qext_vector06fuzz.bit, the corresponding target files qext_vector01fuzz.f32 through qext_vector06fuzz.f32. Those are decoded in the same way as the first set of testvectors, but the comparison thresholds are looser:

```
% qext_compare -s -f32 -thresholds 0.1 0.5 1.0  \
                      qext_vector01decfuzz.f32 qext_test01fuzz.f32
```

For the tested decoder to be deemed compliant with this specification, all testvectors from both sets MUST pass.

4.2. Encoder

It is RECOMMENDED, but not mandatory for an encoder to comply with the following criteria. Encoder testing involves encoding uncoded testvectors, decoding them with the reference decoder, and comparing to the original uncoded files. The test is meant to evaluate encoding at bitrates around 1 Mb/s. For example, encoding at 1024 kb/s can be done with (may be different for the encoder being tested):

```
% opus_demo -e audio 96000 2 1024000 -f32 -cbr -qext \
                           qext_vector01.f32 qext_test01.bit
```

The resulting bitstream then needs to be decoded with the reference implementation with:

```
% opus_demo -d 96000 2 -f32 qext_test01.bit qext_test01enc.f32
```

The decoded output (from the tested encoder) can then be compared against the reference original uncoded PCM:

```
% qext_compare -s -f32 -skip <N> \
                    -thresholds 0.1 0.5 <RMS threshold> \
                    qext_vector01.f32 qext_test01enc.f32
```

where <RMS threshold> for testvectors 1 through 6 are 5, 320, 20, 20, 40, and 5, respectively. The <N> value for the skip compensates for the encoder delay. For the "audio" encoding mode, N=624 samples. For "restricted-lowdelay", N=240 samples.

Because of the inherent limitations of objective quality evaluation metrics -- including the qext_compare tool -- it is also RECOMMENDED to perform a subjective evaluation of an encoder.

5. IANA Considerations

[Note: Until the IANA performs the actions described below, implementers should use 124 instead of 33 as the extension number.]

This document assigns ID 33 to the "Opus Extension IDs" registry created in [opus-extension] to implement the proposed scalable quality extension.

6. Security Considerations

This document does not add security considerations beyond those already documented in [RFC6716].

7. References

7.1. 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>.

[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>.

[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>.

[opus-extension] Terriberry, T.B. and J.-M. Valin, "Extension Formatting for the Opus Codec (draft-ietf-mlcodec-opus-extension)", October 2023.

Author’s Address

Jean-Marc Valin
Google
Canada
Email: jeanmarcv@google.com