Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 23 September 2026

ssw. Whited, Ed.

22 March 2026

Ogg Skeleton
draft-swhited-ogg-skeleton-00

Abstract

Ogg Skeleton defines a logical bitstream that provides structuring information for multitrack Ogg files. It provides clues for synchronization and content negotiation including language selection. It also provides keypoint indices for optimal seeking over high-latency connections or in time-critical scenarios.

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 September 2026.

Table of Contents

1. Introduction

Ogg [RFC3533] is a generic container format, enabling interleaving of several tracks of frame-wise encoded content in a time-multiplexed manner. As an example, an Ogg physical bitstream could encapsulate several tracks of video encoded in [Theora] and multiple tracks of audio encoded in Opus [RFC6716] or FLAC [RFC9639] at the same time. A player that decodes such a bitstream could then play one video channel as the main video playback, alpha-blend another one on top of it (e.g. a caption track), play a main Opus audio track together with several FLAC audio tracks simultaneously (e.g. as sound effects), and provide a choice of Opus channels providing commentary in different languages. Such a file is generally possible to create with Ogg, it is however not possible to generically parse such a file, seek on it, understand what codecs are contained in such a file, and dynamically handle and play back such content.

Ogg does not know anything about the content it carries and leaves it to the media mapping of each codec to declare and describe itself. There is no meta information available at the Ogg level about the content tracks encapsulated within an Ogg physical bitstream. This is particularly a problem if you don’t have all the decoder libraries available and just want to parse an Ogg file to find out what type of data it encapsulates, or want to seek to a temporal offset without having to decode the data (such as on a Web server that serves media).

Ogg Skeleton is designed to overcome these problems. Ogg Skeleton is a logical bitstream within an Ogg stream that contains information about the other encapsulated logical bitstreams. For each logical bitstream it provides information such as its media type, and explains the way that Ogg pages are mapped to time.

Seeking in an Ogg file is typically implemented as a bisection search for the seek target timestamp. However when seeking over a high latency connection, such as the internet, such searches can be slow. Some bitstreams, such as Theora video streams, have keyframes. In order to seek to a given temporal offset in a Theora stream, you must first perform a bisection search to find the target Theora frame, determine its keyframe, and then perform another bisection search to locate that keyframe and decode forwards to the temoporal offset. This can be very slow. Ogg Skeleton provides an index of keyframes, or an index of periodic samples on streams without the concept of a keyframe, enabling seeking over high-latency connections optimally with "one hop".

Ogg Skeleton is also designed to allow the creation of substreams from Ogg physical bitstreams that retain the original timing information. For example, when cutting out the segment between the 7th and the 59th second of an Ogg file, it is ideal to continue to start this cut out file with a playback time of 7 seconds and not of 0. This is of particular interest if you’re streaming this file from a web server after a query for a temporal subpart such as in http://example.com/video.ogv?t=7-59 .

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. The skeleton logical bitstream

2.1. Ident Header

The skeleton logical bitstream starts with an ident header containing information for the complete Ogg physical bitstream. The ident header has the following format:

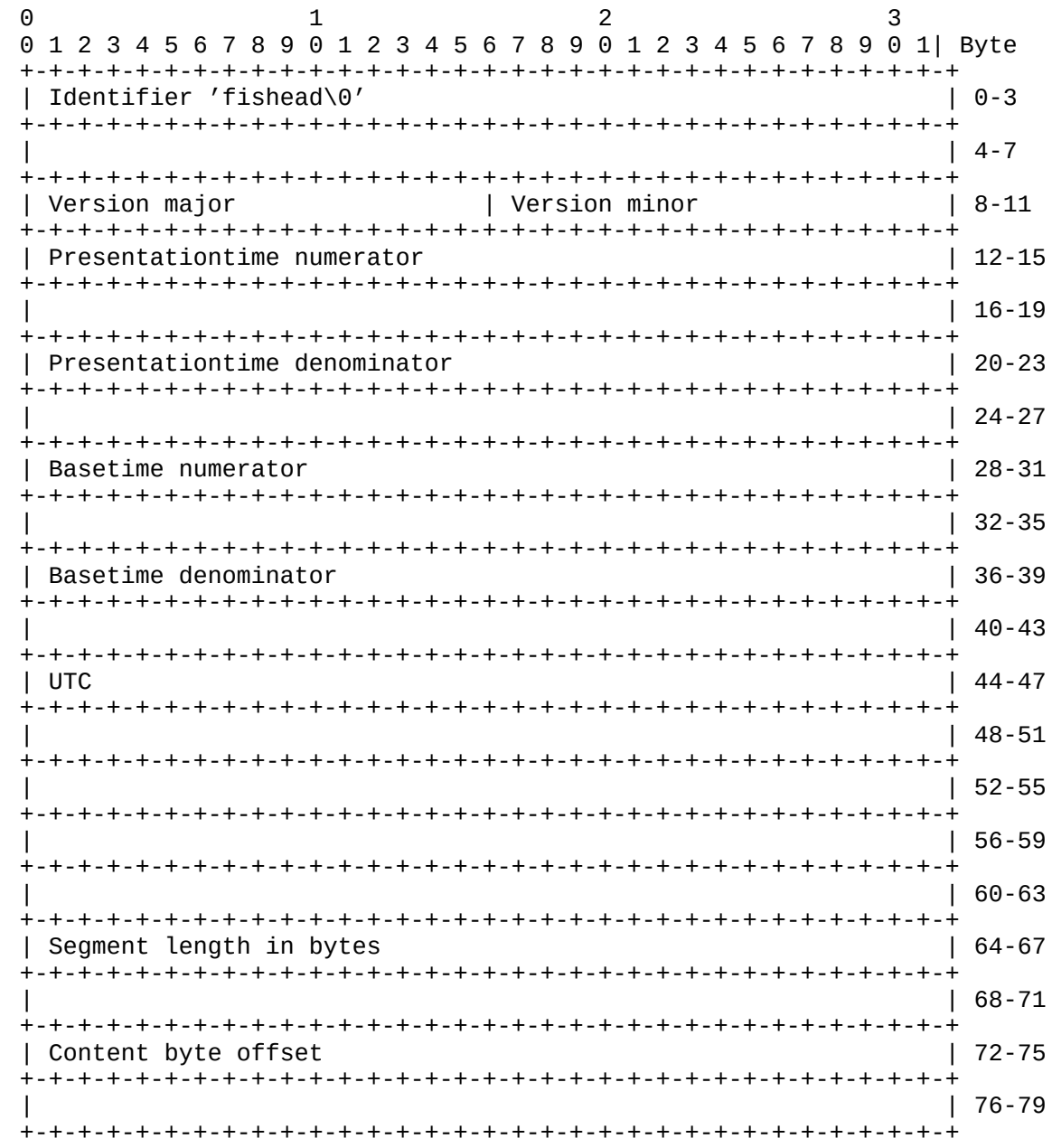

Figure 1: Ident header packet layout

Fields with more than one byte length MUST be encoded Least Significant Byte (LSB) first byte order.

The fields in the skeleton ident header have the following meaning:

Identifier An 8 byte field that identifies this bitstream as Ogg Skeleton. It contains the magic numbers:

* 0x66 'f'
* 0x69 'i'
* 0x73 's'
* 0x68 'h'
* 0x65 'e'
* 0x61 'a'
* 0x64 'd'
* 0x00 '\0'

Version major 2 byte unsigned integer signifying the major version number of the skeleton bitstream. This document specifies the major version 4.

Version minor 2 byte unsigned integer signifying the minor version number of the skeleton bitstream. This document specifies the minor version 0.

Presentationtime numerator & denominator 8 byte unsigned integer each. Together they represent the time at which to start presenting the Ogg physical bitstream given as a rational number. The denominator represents the temporal resolution at which the presentationtime is given. E.g. 5/1000 results in a presentationtime of 0.005 sec. This enables a very high temporal resolution without having to store floating point numbers. In a newly created physical bitstream presentationtime and basetime are the same. When remultiplexing a subpart of the stream, this number MUST be adapted to the requested start time offset of the newly created stream. Presentationtime MUST be larger than or equal to zero.

Basetime numerator & denominator 8 byte signed integer each. Together they represent the basetime of the Ogg physical bitstream given as a rational number like the presentationtime. This number is fixed once the physical bitstream is created and provides a mapping to time for the beginning of the physical bitstream when it starts with a granule position of 0.

UTC A 20 byte string containing a UTC time in the form of YYYYMMDDTHHMMSS.sssZ [ISO.8601.1988]. It associates a calendar date and a wall-clock time with the basetime. If unused the UTC field MUST be a sequence of 20 NUL bytes, making this ident packet and thus the BoS page of the skeleton bitstream constant length.

The possible temporal resolution of the presentation and basetime is on the order of 2^-64. For example, the time formats in use for media that are described in this document range from 1/24 to 1/60 for the different [SMPTE] formats. This resolution is enough for any one of these. It is also expected to accommodate any future needs of time resolution for any other time format and time-continuously sampled data.

A denominator of 0 in either presentationtime or basetime indicates that the respective time is 0 regardless of the value of the numerator.

2.2. Secondary Header

The skeleton secondary headers are a sequence of packets that each contain information about one of the other logical bitstreams contained within the Ogg stream. A skeleton secondary header packet has the following format:

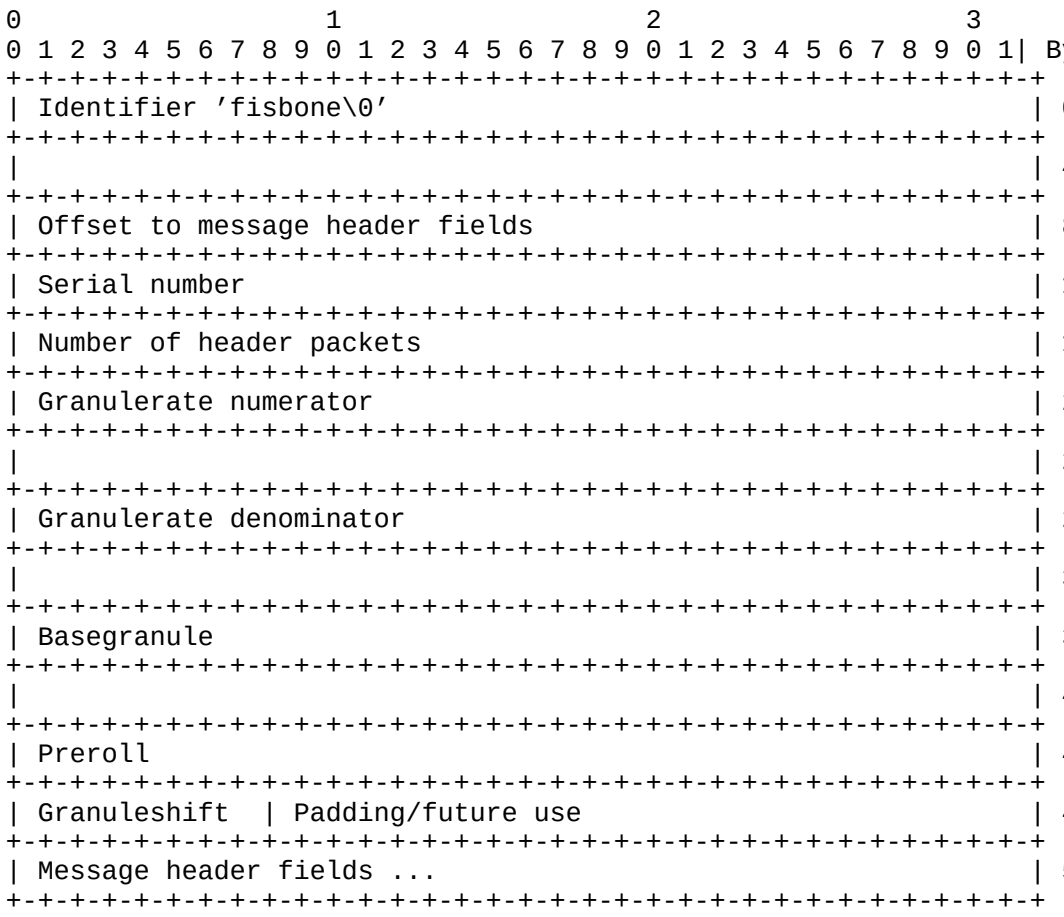

Figure 2: Secondary header packet layout

Fields that are longer than one byte MUST be encoded with LSB first byte order.

The fields in a skeleton secondary header packet have the following meaning:

Identifier An 8 byte field that identifies this packet as a skeleton secondary header for identifying other logical bitstreams. It contains the magic numbers:

* 0x66 'f'

* 0x69 'i'

* 0x73 ’s’
* 0x62 ’b’
* 0x6f ’o’
* 0x6e ’n’
* 0x65 ’e’
* 0x00 ’\0’

Offset to message header fields 4 byte unsigned integer that contains the number of bytes used in this packet before the message header fields. For the version of the skeleton bitstream described in this document this number is fixed to 44. This field accommodates future changes to the skeleton bitstream allowing implementations to parse message header fields even if additional fields have been added in a future revision of Skeleton.

Serial number 4 byte signed integer containing the bitstream serial number of the logical bitstream described by this secondary header packet.

Number of header packets 4 byte unsigned integer that contains the number of header packets of the referenced logical bitstream consisting of the BoS page and the secondary header pages that are included before the Skeleton EoS page.

Granulerate numerator & denominator 8 byte signed integer each. They represent the temporal resolution of the logical bitstream in Hz given as a rational number in the same way as the basetime field.

Basegranule 8 byte signed integer that represents the granule number with which this logical bitstream starts, which is originally 0, but will be a positive offset when only a subpart of the stream is requested.

Preroll 4 byte unsigned integer that contains the number of packets to pre-roll in order to decode a current packet correctly. This is for example the case with Ogg Vorbis, which requires a pre-roll of 2 packets.

Granuleshift A 1 byte unsigned integer describing whether to partition the granule_position ([RFC3533], Section 6) into two for that logical bitstream, and how many of the lower bits to use for the partitioning. The upper bits signify a time-continuous

granule position for an independently decodable and presentable data granule. The lower bits are generally used to specify the relative offset of dependent packets, such as predicted frames of a video. Hence these can be addressed, though not decoded without tracing back to the last fully decodable data granule. This is the case with Ogg Theora; the general procedure is given in Section 4.2.

Padding/future use 3 bytes of padding data that MUST be set to zero but may be used in future versions of Skeleton.

Message header fields Header fields following the generic Internet Message Format defined in [RFC2822]. Each header field consists of a name followed by a colon (0x3a ":") and the field value. Field names are case-insensitive. The field value MAY be preceded by any amount of white space, though a single space (SP, ASCII value 32) is RECOMMENDED. Multi-line header fields as described in Section 2.2.3 of [RFC2822] MUST be supported.

2.2.1. Message Header Fields

The following message headers are REQUIRED:

Content-type Mime type of the content encoded in this stream, e.g. audio/opus, video/theora, etc.

Role Describes the function of this track. Common examples are "video/main", "audio/main", "text/caption". It is RECOMMENDED to stick to the existing role names defined in Part Role of [SkeletonHeaders].

Name A unique free text string which can be used to directly address or identify the track and which may be shown in user interfaces that list or allow for selection of tracks.

For more message headers, see [SkeletonHeaders].

As per [RFC2277], message header fields are considered protocol data, i.e. they are not expected to have human readable text, and they MUST be entirely encoded in UTF-8. In addition, the mandatory header fields MUST be encoded in ASCII.

2.3. Keypoint Index Packets

Before the Skeleton EoS page in the segment header pages come the keypoint index packets. Index packets are OPTIONAL in a valid Skeleton bitstream. If index packets are included, there SHOULD be at least one keypoint packet for each content logical bitstream in the Ogg stream.

In order to save space, the offsets and timestamps in keypoint packets are stored as deltas, and then variable byte-encoded. The offset and timestamp deltas store the difference between the keypoint's offset and timestamp from the previous keypoint's offset and timestamp. To calculate the page offset of a keypoint, calculate the sum of the offset deltas of up to and including the keypoint in question. The variable byte encoded integers are encoded using 7 bits per byte to store the integer's bits, and the high bit is set in the last byte used to encode the integer. The bits and bytes are in LSB first byte order. For example, the integer 7843, or 0001 1110 1010 0011 in binary, would be stored as two bytes: 0xBD 0x23, or 1011 1101 0010 0011 in binary.

Each index packet contains the following fields:

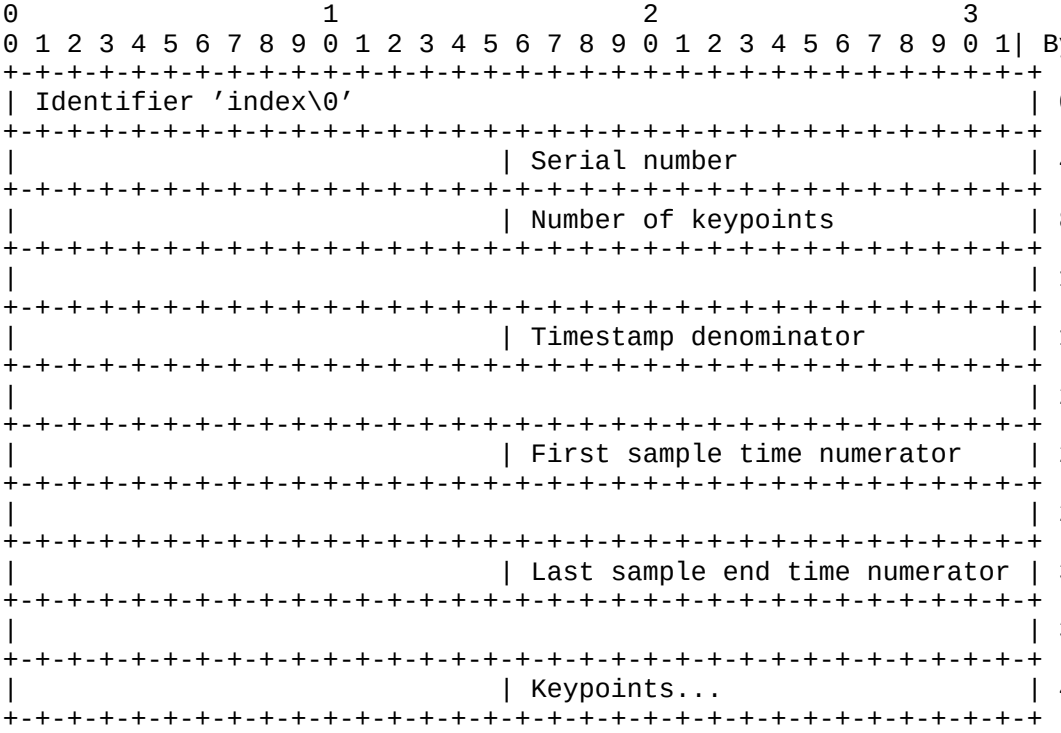

Figure 3: Keypoint index packet layout

The fields in an index packet have the following meaning:

Identifier An 6 byte field that identifies this page as an index packet. It contains the magic numbers:

* 0x69 'i'
* 0x6e 'n'
* 0x64 'd'
* 0x65 'e'
* 0x78 'x'
* 0x00 '\0'

Serial number 4 byte signed integer containing the bitstream serial number of the Ogg logical bitstream described by this index packet.

Number of keypoints 8 byte unsigned integer containing the number of keypoints included at the end of this index packet. This value MAY be zero.

Timestamp denominator 8 byte signed integer representing the presentation time denominator for this stream. All timestamps, including keypoint timestamps and first and last sample timestamps are fractions of seconds over this denominator. The timestamp denominator MUST NOT be 0.

First sample time numerator 8 byte signed integer representing the numerator for the presentation time of the first sample in the track.

Last sample end time numerator 8 byte signed integer representing the numerator for the presentation end time of the last sample in the track.

Keypoints 'Number of keypoints' key points, starting with the first keypoint at offset 42.

2.3.1. Keypoints

Each keypoint comprises the following fields:

1. The keypoint’s page’s byte offset delta, as a variable byte encoded integer. This is the number of bytes that this keypoint is after the preceeding keypoint’s offset, or from the start of the segment if this is the first keypoint. The keypoint’s page start is therefore the sum of the byte-offset-deltas of all the keypoints which come before it.

2. The presentation time numerator delta of the first keypoint which starts on the page at the keypoint’s offset, as a variable byte encoded integer. This is the difference from the previous keypoint’s timestamp numerator. The keypoint’s timestamp numerator is therefore the sum of all the timestamp numerator deltas up to and including this keypoint’s. Divide the timestamp numerator sum by the timestamp denominator to determine the presentation time of the keyframe in seconds

Keypoint’s MUST be stored in increasing order by offset (and thus by presentation time).

The byte offsets stored in keypoints are relative to the start of the Ogg bitstream segment. For a physical Ogg bitstream made up of two chained Oggs, the offsets in the second Ogg segment’s bitstream’s index are relative to the beginning of the second Ogg in the chain, not the first. If a physical Ogg bitstream is made up of chained Oggs, the presence of an index in one segment does not imply that there will be an index in any other segment.

The first-sample-time and last-sample-time are rational numbers, in seconds. If the denominator is 0 for the first-sample-time or the last-sample-time, then that value was unable to be determined at indexing time, and is unknown.

The exact number of keypoints used to construct the index is up to the indexer, but it is RECOMMENDED to include at most one keypoint per every 64KB of data, or every 1000ms, whichever is least frequent. For codecs which use key frames, indexers SHOULD create one keypoint pointing to each key frame.

3. Ogg Media Mapping

Because Ogg may be used to encapsulate any any type of time-continuous data stream it does not place requirements on the order of data streams in the file. An Ogg physical bitstream with a Skeleton track, however, MUST have the following page order:

* Skeleton Beginning-of-Stream (BoS) page.

* BoS pages of the other logical bitstreams.

* Secondary header pages of all logical bitstreams (including Skeleton secondary headers).

* Skeleton keypoint index packets (if present)

* Skeleton End-of-Stream (EoS) page.

* Data and EoS pages of logical bitstreams, excluding skeleton, multiplexed in a time-synchronous fashion.

In addition, Skeleton has the following further restrictions:

* An Ogg segment MUST NOT contain more than one Skeleton logical bitstream.

* The skeleton BoS page MUST only contain the Skeleton Ident (fishead) header as a single packet.

* The secondary header pages of a Skeleton logical bitstream consist only of fisbone header packets.

* The Skeleton stream MUST NOT contain any content pages or data packets other than keypoint index packets.

* The skeleton EoS page MUST contain a single packet of length zero.

* The skeleton EoS page MUST appear before any data pages for any other logical bitstream in the Ogg bitstream.

* The skeleton EoS page MUST end the skeleton logical bitstream. If an Ogg stream parser reaches the skeleton EoS page, it knows that it has received all the BoS and secondary header pages and can start setting up its decoding or parsing environment.

4. Time Handling

With time-continuous data inside Ogg, one needs to handle data at four different levels:

* at the byte level: upon seeking,

* at the packet level: upon encapsulating,

* at the granule level: upon recomposing, and

* at the time level: upon displaying and addressing.

This section explains how they all fit together.

4.1. Conceptual Overview

Ogg bitstreams represent a single timeline with multiple content tracks. All of these tracks relate to the same timeline which starts at a certain time point and ends when the last bitstream ends.

An example bitstream can be seen in the following figure. It consists of an Ogg bitstream that contains 4 media bitstreams. The picture is a conceptual representation of the time intervals covered by the different logical bitstreams and the Ogg pages used to encapsulate the data. In the flat representation these are multiplexed such that the data packets of each of these bitstreams occur at the correct time.

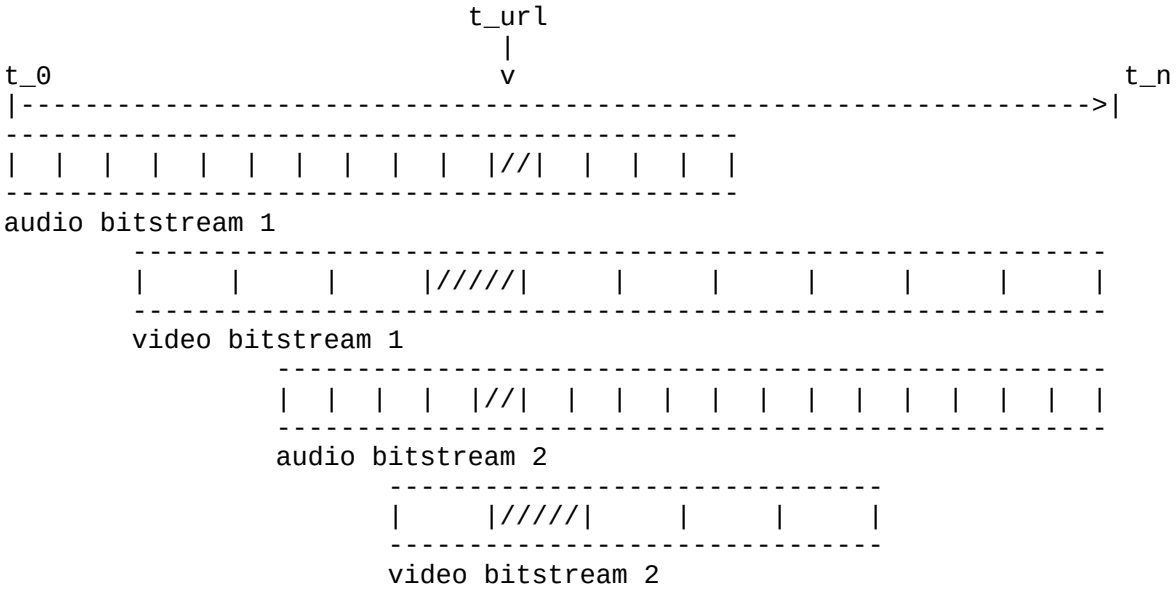

Figure 4: Ogg bitstream layout example

The time point at which an Ogg bitstream starts (t_0 in Figure 4) is called the "basetime" and represents the time in seconds associated with the granule position of 0 on all logical bitstreams. Typically, a newly created Ogg file starts all its logical bitstreams at granule position 0, and a typical extract of an Ogg bitstream, such as the one starting at t_url in Figure 4, starts each of its logical bitstreams at different granule positions. These granule positions are stored in the "basegranule" field of the skeleton secondary header packets.

The "basetime" of an Ogg bitstream may be 0, but it can also be any positive time. For example, in professional video production, the first frame of video of a program normally refers to a SMPTE basetime [SMPTE] of 01:00:00:00, not 00:00:00:00. Associating such a practice to a digital video resource requires a way to store that basetime with the resource and interpreting it correctly when addressing offsets such as t_url. Skeleton provides such a mapping through the basetime field in the skeleton ident header.

Also associated with the basetime is an [ISO.8601.1988] calendar date and wall-clock time (a "UTC base") which represent a real-world time giving some meaningful calendar date association to the content such as the creation time or the first presentation time. The UTC base is specified in the "UTC" field of the Skeleton ident header.

4.2. Mapping a granule position to a time position

Each one of the encapsulated data bitstreams have their own temporal resolution at which they provide data to cover the given timeline. This temporal resolution is usually given through the sampling rate of the particular bitstream. For example, a raw audio bitstream at CD quality is sampled with a sampling rate of 44100 Hz. A video bitstream may be sampled with a frame rate of 25 frames per second.

This temporal resolution is called the "granulerate". A granule is a data element that is based on a regular data rate specific to the content type, such as the frame rate for video or the sampling rate for audio. It even exists for bitstreams that are not sampled at a regular rate—then it is the highest resolution of any of the used sampling rates. The granulerate is specified in the skeleton secondary header packets for each logical bitstream.

Each one of the bitstreams insert data into the Ogg bitstream through packets which have an associated temporal duration based on the encoder packaging. Packets are packaged into Ogg pages, which have a granule position associated with them ([RFC3533], Section 6). Not taking the special case of a granuleshift into account, the granule position specifies the number of granules that have been encapsulated since the implicit start of the original bitstream until and including the given Ogg page.

The granule position together with the granulerate and granuleshift information of the skeleton secondary header packets for the particular logical bitstream are used for the calculation of the time position for which a data packet of the logical bitstream completes data. A granule position of -1 indicates a special case and MUST NOT be used for calculation of a mapping to time.

In principle, the granule position of an Ogg page divided by the granulerate of this page's logical bitstream provides the time position that is reached in that bitstream after decoding all data packets finished on this page. However, the granule position field in an Ogg page allows for a more finely-grained description of the temporal position. The following image explains the composition of the granule position field in an Ogg page:

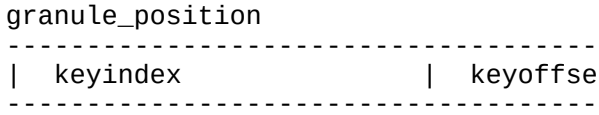

Figure 5: Granule position field layout

The granuleshift field of the skeleton secondary header packets describes how many of the granule_position's 64 bits are being used for the keyoffset. The keyoffset part of the granule_position is commonly used when the logical bitstream consists of packets that can only be fully decoded when referring back to a previous packet. For example, video streams often consist of inter and intra coded frames, where the intra frames are fully decodable and the inter frames are intermediate frames that require backtracking to the last inter frame for accurate decoding. Another example is a logical bitstream that is mapped as instantaneous information (i.e. their granuleposition represents the start time and the end time of the packet data), but actually has a duration associated to it, which is provided through a subsequent packet. The keyindex part of the granule_position is then used to provide the temporal position of the reference packet and the keyoffset part provides a counter for the data in between.

The calculation of the temporal position of an Ogg page using Skeleton is thus specified through the following formula:

```
t_page = basetime + ((keyindex + keyoffset) / granulerate)
```

Figure 6: Ogg page to temporal position formula

The basetime provides the time offset used at the beginning of the logical bitstream for the first data packet and thus MUST be added for a correct calculation of the temporal position.

As an example regard an audio bitstream that has a granulerate of 44100 (i.e. 44100 samples per 1 sec), a granuleshift of 0, and starts at 4 sec. When reaching a granule_position of 88200, this maps to a time position of 6 seconds:

```
t_page = 4 + ((88200 + 0) / 44100) = 6
```

Figure 7

This signifies that the bitstream has reached the second sec of the audio bitstream after the end of decoding this page’s packets, but maps to 6 seconds because of the basetime.

As another example consider a video bitstream that has a granulerate of 25 (i.e. 25 frames per 1 second), a granuleshift of 3 (because it encodes - say - 7 partial frames between each fully encoded frame), and starts at 0 sec. When reaching a granule_position of 997, i.e. a keyindex of 62 and a keyshift of 5, this maps to a fully decodable time position of 2.68 seconds:

t_page = 0 + ((62 + 5) / 25) = 2.68 sec

Figure 8

The granulerate of a time-instantaneous bitstream can be chosen arbitrarily by the bitstream multiplexer. Per default, a granulerate of 1000 is used, which is the resolution of npt. The resolution of all the time schemes is given as:

npt 1000 (milliseconds)

smpte-24 24 (24 fps)

smpte-24-drop 24/1.001 = 23.976 (approx. as per SMPTE)

smpte-25 25

smpte-30 30

smpte-30-drop 30/1.001 = 29.970 (approx. as per SMPTE)

smpte-50 50

smpte-60 60

smpte-60-drop 60/1.001 = 59.940 (approx. as per SMPTE)

The granule position of the page finishing data of a time-instantaneous bitstream packet MUST signify the start time of that packet. For example, a CMML bitstream with a granulerate of 1000, a basetime of 0, and a clip that lasts from npt=12.020 till npt=15.0 will get a granule_position of 12020. In contrast, the granule_position of the page finishing data of e.g. an audio bitstream with granulerate 44100, basetime 0 and containing data from npt=12.020 to npt=15.0 will be 661500.

A note about field overflows: an overflow of the granule position field can destroy the temporal integrity of the Ogg physical bitstream. In this case, a multiplexer MUST end the Ogg physical bitstream and restart a new one resetting the counter to 0 and adjusting the basetime appropriately. This is also called sequential multiplexing in Ogg. The same measure MUST be taken in case of an overflow of the page_sequence_number on one of the logical bitstreams.

4.3. Seeking into the bitstream

Seeking to a time offset inside an Ogg logical bitstream is a fundamental activity frequently performed on media data. Time inside an Ogg with a Skeleton track is specified as a temporal offset from the "beginning" of the stream, making use of the basetime field. Time offsets can also be specified as calendar dates and times. The UTC base is then used as a basis for offsetting.

The basetime allows to correctly map a temporal offset point such as a temporal URI to a byte position in the stream. In the above figure take t_uri=npt:14.0 as the temporal offset addressed on a stream with t_0=npt:5.0 as the basetime— this requires a stream offsetting of only 9 sec to the appropriate granule position in each of the bitstreams, in the figure marked through patterned pages.

The seeking action is performed on the interleaved bitstream, in which the data packets occur in a temporally consecutive order based on the time at which their data ends. These times are represented in the granule positions of the Ogg pages, which are only allowed to monotonically increase within one logical bitstream. This implies that when having found an Ogg page with a granule position that maps to a given seek time (i.e. covers the time or ends at it), the seek has found the right location. This applies over all logical bitstreams. In the above example, this means that the byte position of the first occurring page of the patterned pages has been found.

There is a complication to the seeking: some logical bitstreams have backwards dependencies in their data packets and these have to be taken into account for seeking. For example, a logical bitstream may require several of its previous packets to allow a correct and complete decoding of the actual packet that occurs at the seektime. This is the case for Theora which requires to go back to the previous keyframe when decoding from a time offset. It is also the case for Vorbis which requires the previous 2 packets for accurate setup of the frequency transform - Speex needs approximately 2 packets for similar reasons. Even instantaneous bitstreams may require to go back to a previous packet to recover the last state information.

Therefore, once seeking has located the correct byte position that refers to the given temporal offset, it MUST seek back. For logical bitstreams that have a non-zero "granuleshift" in the skeleton, it MUST seek back to the Ogg page that has a "keyindex" granule position. For logical bitstreams that have a non-zero "preroll" in the skeleton, it MUST seek back that many packets. The earliest byte position that satisfies all these requirements is the correct seek position.

A player that presents from an offset MUST take into account that the bitstream may contain some packets that are only there to allow accurate decoding of the seek time. When the backwards dependencies were resolved for a specific logical bitstream, several non-relevant Ogg pages of may also have ended up in the intermediate. These have to be skipped by a player. The time that a player MUST start presenting from is given in the "presentationtime" in the skeleton ident header.

4.4. Remultiplexing an Ogg Bitstream

Ogg with a Skeleton track allows for the creation of mashups of a file without actual decoding and re-encoding. A mashup in the sense used here is when a subpart of a Ogg physical bitstream is required, such as a temporal sub-interval from the whole file. Skeleton allows the creation of the mashup bitstream through recomposition and remultiplexing. There are several aims for performing the remultiplexing with as little effort and therefore as little delay as possible:

* no decoding of the logical bitstreams is performed.

* no changes to the pages, in particular to the granule positions are made.

* changes occur only to the control section.

The fields of the skeleton track allow achievement of all these aims. Remultiplexing is essentially achieved by seeking to the position as described above and then including from each logical bitstream only the relevant Ogg pages into the new stream. Changes to fields in the bitstream are restricted to the control section:

* the "presentationtime" MUST be adjusted to the requested start time

* the "startgranule" for each logical bitstream MUST be adjusted to the granule position at which each logical bitstream starts. This is not the first granule position of the Ogg pages included into the bitstream, but rather the last one that did not get included, as it represents the start time of the bitstream.

Everything else, and in particular the Ogg pages, stay the same. This is important to allow caching of such files as is required for Web proxies.

5. IANA Considerations

// TODO: define an Ogg Skeleton Header Registry and register the // stuff from the [SkeletonHeaders] wiki page? This memo includes no request to IANA.

6. Security Considerations

Ogg format bitstreams contain several multiplexed binary and non-binary data bitstream. There is no generic encryption or signing mechanism provided for the complete bitstream or anyone of its parts. As the format of the encapsulated media bitstreams is not prescribed and is identified through the "Content-type" Message header field in that bitstream’s skeleton secondary header packet, it is possible to encrypt or sign that media bitstream and then mark it accordingly with a MIME type that signifies the encryption. It is up to the applications that use this bitstream to provide an appropriate codec to handle such bitstreams.

As Ogg format bitstreams generally contain arbitrary bitstreams, it is possible to include executable content in them. This can be an issue with applications that decode these bitstreams, especially when they are used in a network scenario. Such applications MUST ensure correct handling of manipulated bitstreams, of buffer overflow and the like.

7. Normative References

[RFC2822] Resnick, P., Ed., "Internet Message Format", RFC 2822, DOI 10.17487/RFC2822, April 2001, <https://www.rfc-editor.org/info/rfc2822>.

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

[ISO.8601.1988]
International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times", ISO Standard 8601, June 1988.

8. Informative 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>.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <https://www.rfc-editor.org/info/rfc2277>.

[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>.

[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>.

[RFC9639] van Beurden, M.Q.C. and A. Weaver, "Free Lossless Audio Codec (FLAC)", RFC 9639, DOI 10.17487/RFC9639, December 2024, <https://www.rfc-editor.org/info/rfc9639>.

[SkeletonHeaders]
Xiph.Org, "Skeleton Headers", March 2026, <https://wiki.xiph.org/SkeletonHeaders>.

[SMPTE] SMPTE, "SMPTE STANDARD for Television, Audio and Film - Time and Control Code", ANSI 12M-1999, September 1999.

[Theora] Xiph.Org, "Theora Specification", 3 June 2017, <https://xiph.org/theora/doc/Theora.pdf>.

Acknowledgements

Thanks to Silvia Pfeiffer, and Conrad D. Parker for their work on an earlier attempt to specify the Skeleton bitstream which was consulted (and stolen from) heavily while writing this document. Thanks also to Christopher Montgomery and Andre Pang who are thanked in the earlier draft mentioned previously and (probably) deserve to be thanked again. Also to the contributors to the Ogg Skeleton description on the Xiph.Org wiki.

Author’s Address

Sam Whited (editor)
Email: sam@samwhited.com
URI: https://blog.samwhited.com