                   The Ogg Encapsulation Format Version 0

Status of this Memo

   This memo provides information for the Internet community. It does
   not specify an Internet standard of any kind. Distribution of this
   memo is unlimited.

Copyright Notice

Abstract

   This document describes the Ogg bitstream format version 0, which is
   a general, freely-available encapsulation format for media streams.
   It is able to encapsulate any kind and number of video and audio
   encoding formats as well as other data streams in a single bitstream.

Terminology

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in BCP 14, RFC 2119 [2].

Table of Contents

1. Introduction

   The Ogg bitstream format has been developed as a part of a larger
   project aimed at creating a set of components for the coding and
   decoding of multimedia content (codecs) which are to be freely
   available and freely re-implementable, both in software and in
   hardware for the computing community at large, including the Internet
   community. It is the intention of the Ogg developers represented by
   Xiph.Org that it be usable without intellectual property concerns.

   This document describes the Ogg bitstream format and how to use it to
   encapsulate one or several media bitstreams created by one or several
   encoders. The Ogg transport bitstream is designed to provide
   framing, error protection and seeking structure for higher-level
   codec streams that consist of raw, unencapsulated data packets, such
   as the Vorbis audio codec or the upcoming Tarkin and Theora video
   codecs. It is capable of interleaving different binary media and
   other time-continuous data streams that are prepared by an encoder as
   a sequence of data packets. Ogg provides enough information to
   properly separate data back into such encoder created data packets at
   the original packet boundaries without relying on decoding to find
   packet boundaries.

   Please note that the MIME type application/ogg has been registered
   with the IANA [1].

2. Definitions

   For describing the Ogg encapsulation process, a set of terms will be
   used whose meaning needs to be well understood. Therefore, some of
   the most fundamental terms are defined now before we start with the
   description of the requirements for a generic media stream
   encapsulation format, the process of encapsulation, and the concrete
   format of the Ogg bitstream. See the Appendix for a more complete
   glossary.

   The result of an Ogg encapsulation is called the "Physical (Ogg)
   Bitstream". It encapsulates one or several encoder-created
   bitstreams, which are called "Logical Bitstreams". A logical
   bitstream, provided to the Ogg encapsulation process, has a
   structure, i.e., it is split up into a sequence of so-called
   "Packets". The packets are created by the encoder of that logical
   bitstream and represent meaningful entities for that encoder only
   (e.g., an uncompressed stream may use video frames as packets). They
   do not contain boundary information - strung together they appear to
   be streams of random bytes with no landmarks.

Please note that the term "packet" is not used in this document to signify entities for transport over a network.

3. Requirements for a generic encapsulation format

The design idea behind Ogg was to provide a generic, linear media transport format to enable both file-based storage and stream-based transmission of one or several interleaved media streams independent of the encoding format of the media data. Such an encapsulation format needs to provide:

o framing for logical bitstreams.

o interleaving of different logical bitstreams.

o detection of corruption.

o recapture after a parsing error.

o position landmarks for direct random access of arbitrary positions in the bitstream.

o streaming capability (i.e., no seeking is needed to build a 100% complete bitstream).

o small overhead (i.e., use no more than approximately 1-2% of bitstream bandwidth for packet boundary marking, high-level framing, sync and seeking).

o simplicity to enable fast parsing.

o simple concatenation mechanism of several physical bitstreams.

All of these design considerations have been taken into consideration for Ogg. Ogg supports framing and interleaving of logical bitstreams, seeking landmarks, detection of corruption, and stream resynchronisation after a parsing error with no more than approximately 1-2% overhead. It is a generic framework to perform encapsulation of time-continuous bitstreams. It does not know any specifics about the codec data that it encapsulates and is thus independent of any media codec.

4. The Ogg bitstream format

A physical Ogg bitstream consists of multiple logical bitstreams interleaved in so-called "Pages". Whole pages are taken in order from multiple logical bitstreams multiplexed at the page level. The logical bitstreams are identified by a unique serial number in the

header of each page of the physical bitstream. This unique serial
number is created randomly and does not have any connection to the
content or encoder of the logical bitstream it represents. Pages of
all logical bitstreams are concurrently interleaved, but they need
not be in a regular order - they are only required to be consecutive
within the logical bitstream. Ogg demultiplexing reconstructs the
original logical bitstreams from the physical bitstream by taking the
pages in order from the physical bitstream and redirecting them into
the appropriate logical decoding entity.

Each Ogg page contains only one type of data as it belongs to one
logical bitstream only. Pages are of variable size and have a page
header containing encapsulation and error recovery information. Each
logical bitstream in a physical Ogg bitstream starts with a special
start page (bos=beginning of stream) and ends with a special page
(eos=end of stream).

The bos page contains information to uniquely identify the codec type
and MAY contain information to set up the decoding process. The bos
page SHOULD also contain information about the encoded media - for
example, for audio, it should contain the sample rate and number of
channels. By convention, the first bytes of the bos page contain
magic data that uniquely identifies the required codec. It is the
responsibility of anyone fielding a new codec to make sure it is
possible to reliably distinguish his/her codec from all other codecs
in use. There is no fixed way to detect the end of the codec-
identifying marker. The format of the bos page is dependent on the
codec and therefore MUST be given in the encapsulation specification
of that logical bitstream type. Ogg also allows but does not require
secondary header packets after the bos page for logical bitstreams
and these must also precede any data packets in any logical
bitstream. These subsequent header packets are framed into an
integral number of pages, which will not contain any data packets.
So, a physical bitstream begins with the bos pages of all logical
bitstreams containing one initial header packet per page, followed by
the subsidiary header packets of all streams, followed by pages
containing data packets.

The encapsulation specification for one or more logical bitstreams is
called a "media mapping". An example for a media mapping is "Ogg
Vorbis", which uses the Ogg framework to encapsulate Vorbis-encoded
audio data for stream-based storage (such as files) and transport
(such as TCP streams or pipes). Ogg Vorbis provides the name and
revision of the Vorbis codec, the audio rate and the audio quality on
the Ogg Vorbis bos page. It also uses two additional header pages
per logical bitstream. The Ogg Vorbis bos page starts with the byte
0x01, followed by "vorbis" (a total of 7 bytes of identifier).

Ogg knows two types of multiplexing: concurrent multiplexing (so-
called "Grouping") and sequential multiplexing (so-called
"Chaining"). Grouping defines how to interleave several logical
bitstreams page-wise in the same physical bitstream. Grouping is for
example needed for interleaving a video stream with several
synchronised audio tracks using different codecs in different logical
bitstreams. Chaining on the other hand, is defined to provide a
simple mechanism to concatenate physical Ogg bitstreams, as is often
needed for streaming applications.

In grouping, all bos pages of all logical bitstreams MUST appear
together at the beginning of the Ogg bitstream. The media mapping
specifies the order of the initial pages. For example, the grouping
of a specific Ogg video and Ogg audio bitstream may specify that the
physical bitstream MUST begin with the bos page of the logical video
bitstream, followed by the bos page of the audio bitstream. Unlike
bos pages, eos pages for the logical bitstreams need not all occur
contiguously. Eos pages may be 'nil' pages, that is, pages
containing no content but simply a page header with position
information and the eos flag set in the page header. Each grouped
logical bitstream MUST have a unique serial number within the scope
of the physical bitstream.

In chaining, complete logical bitstreams are concatenated. The
bitstreams do not overlap, i.e., the eos page of a given logical
bitstream is immediately followed by the bos page of the next. Each
chained logical bitstream MUST have a unique serial number within the
scope of the physical bitstream.

It is possible to consecutively chain groups of concurrently
multiplexed bitstreams. The groups, when unchained, MUST stand on
their own as a valid concurrently multiplexed bitstream. The
following diagram shows a schematic example of such a physical
bitstream that obeys all the rules of both grouped and chained
multiplexed bitstreams.

```
          physical bitstream with pages of
       different logical bitstreams grouped and chained
    -------------------------------------------------------------
    |*A*|*B*|*C*|A|A|C|B|A|B|#A#|C|...|B|C|#B#|#C#|*D*|D|...|#D#|
    -------------------------------------------------------------
     bos bos bos               eos         eos eos bos      eos
```
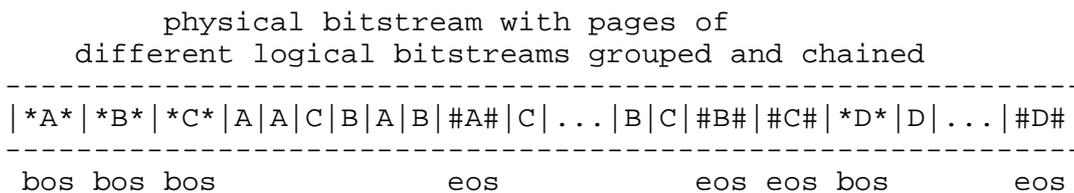
In this example, there are two chained physical bitstreams, the first
of which is a grouped stream of three logical bitstreams A, B, and C.
The second physical bitstream is chained after the end of the grouped
bitstream, which ends after the last eos page of all its grouped
logical bitstreams. As can be seen, grouped bitstreams begin

together - all of the bos pages MUST appear before any data pages.
It can also be seen that pages of concurrently multiplexed bitstreams
need not conform to a regular order. And it can be seen that a
grouped bitstream can end long before the other bitstreams in the
group end.

Ogg does not know any specifics about the codec data except that each
logical bitstream belongs to a different codec, the data from the
codec comes in order and has position markers (so-called "Granule
positions"). Ogg does not have a concept of 'time': it only knows
about sequentially increasing, unitless position markers. An
application can only get temporal information through higher layers
which have access to the codec APIs to assign and convert granule
positions or time.

A specific definition of a media mapping using Ogg may put further
constraints on its specific use of the Ogg bitstream format. For
example, a specific media mapping may require that all the eos pages
for all grouped bitstreams need to appear in direct sequence. An
example for a media mapping is the specification of "Ogg Vorbis".
Another example is the upcoming "Ogg Theora" specification which
encapsulates Theora-encoded video data and usually comes multiplexed
with a Vorbis stream for an Ogg containing synchronised audio and
video. As Ogg does not specify temporal relationships between the
encapsulated concurrently multiplexed bitstreams, the temporal
synchronisation between the audio and video stream will be specified
in this media mapping. To enable streaming, pages from various
logical bitstreams will typically be interleaved in chronological
order.

5. The encapsulation process

   The process of multiplexing different logical bitstreams happens at
   the level of pages as described above. The bitstreams provided by
   encoders are however handed over to Ogg as so-called "Packets" with
   packet boundaries dependent on the encoding format. The process of
   encapsulating packets into pages will be described now.

   From Ogg's perspective, packets can be of any arbitrary size. A
   specific media mapping will define how to group or break up packets
   from a specific media encoder. As Ogg pages have a maximum size of
   about 64 kBytes, sometimes a packet has to be distributed over
   several pages. To simplify that process, Ogg divides each packet
   into 255 byte long chunks plus a final shorter chunk. These chunks
   are called "Ogg Segments". They are only a logical construct and do
   not have a header for themselves.

A group of contiguous segments is wrapped into a variable length page
preceded by a header. A segment table in the page header tells about
the "Lacing values" (sizes) of the segments included in the page. A
flag in the page header tells whether a page contains a packet
continued from a previous page. Note that a lacing value of 255
implies that a second lacing value follows in the packet, and a value
of less than 255 marks the end of the packet after that many
additional bytes. A packet of 255 bytes (or a multiple of 255 bytes)
is terminated by a lacing value of 0. Note also that a 'nil' (zero
length) packet is not an error; it consists of nothing more than a
lacing value of zero in the header.

The encoding is optimized for speed and the expected case of the
majority of packets being between 50 and 200 bytes large. This is a
design justification rather than a recommendation. This encoding
both avoids imposing a maximum packet size as well as imposing
minimum overhead on small packets. In contrast, e.g., simply using
two bytes at the head of every packet and having a max packet size of
32 kBytes would always penalize small packets (< 255 bytes, the
typical case) with twice the segmentation overhead. Using the lacing
values as suggested, small packets see the minimum possible byte-
aligned overhead (1 byte) and large packets (>512 bytes) see a fairly
constant ~0.5% overhead on encoding space.

The following diagram shows a schematic example of a media mapping
using Ogg and grouped logical bitstreams:

```
        logical bitstream with packet boundaries
   ------------------------------------------------------------------
> |     packet_1              | packet_2          | packet_3 |  <
   ------------------------------------------------------------------

                  |segmentation (logically only)
                  v

     packet_1 (5 segments)         packet_2 (4 segs)    p_3 (2 segs)
     ----------------------------- -------------------- ------------
..  |seg_1|seg_2|seg_3|seg_4|s_5 | |seg_1|seg_2|seg_3|| |seg_1|s_2 | ..
     ----------------------------- -------------------- ------------

                  | page encapsulation
                  v

  page_1 (packet_1 data)    page_2 (pket_1 data)   page_3 (packet_2 data)
-----------------------    ---------------        -----------------------
|H|------------------- |   |H|----------- |       |H|------------------- |
|D||seg_1|seg_2|seg_3| |   |D|seg_4|s_5 | |       |D||seg_1|seg_2|seg_3| | ...
|R|------------------- |   |R|----------- |       |R|------------------- |
-----------------------    ---------------        -----------------------

                      |
pages of              |
other      --------|   |
logical       -------
bitstreams    | MUX |
               -------
                  |
                  v

                 page_1  page_2          page_3
            ------  ------ -------  ----- -------
...  ||   | ||    | ||    | ||   | ||    |  ...
            ------  ------ -------  ----- -------
                 physical Ogg bitstream
```

In this example we take a snapshot of the encapsulation process of
one logical bitstream. We can see part of that bitstream's
subdivision into packets as provided by the codec. The Ogg
encapsulation process chops up the packets into segments. The
packets in this example are rather large such that packet_1 is split
into 5 segments - 4 segments with 255 bytes and a final smaller one.
Packet_2 is split into 4 segments - 3 segments with 255 bytes and a

final very small one - and packet_3 is split into two segments. The
encapsulation process then creates pages, which are quite small in
this example. Page_1 consists of the first three segments of
packet_1, page_2 contains the remaining 2 segments from packet_1, and
page_3 contains the first three pages of packet_2. Finally, this
logical bitstream is multiplexed into a physical Ogg bitstream with
pages of other logical bitstreams.

6. The Ogg page format

   A physical Ogg bitstream consists of a sequence of concatenated
   pages. Pages are of variable size, usually 4-8 kB, maximum 65307
   bytes. A page header contains all the information needed to
   demultiplex the logical bitstreams out of the physical bitstream and
   to perform basic error recovery and landmarks for seeking. Each page
   is a self-contained entity such that the page decode mechanism can
   recognize, verify, and handle single pages at a time without
   requiring the overall bitstream.

   The Ogg page header has the following format:

```
 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| Byte
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| capture_pattern: Magic number for page start "OggS"          | 0-3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| version       | header_type   | granule_position             | 4-7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                              | 8-11
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               | bitstream_serial_number      | 12-15
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               | page_sequence_number         | 16-19
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               | CRC_checksum                 | 20-23
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |page_segments  | segment_table | 24-27
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| ...                                                          | 28-
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The LSb (least significant bit) comes first in the Bytes. Fields
   with more than one byte length are encoded LSB (least significant
   byte) first.

The fields in the page header have the following meaning:

1. capture_pattern: a 4 Byte field that signifies the beginning of a
   page. It contains the magic numbers:

        0x4f 'O'

        0x67 'g'

        0x67 'g'

        0x53 'S'

   It helps a decoder to find the page boundaries and regain
   synchronisation after parsing a corrupted stream. Once the
   capture pattern is found, the decoder verifies page sync and
   integrity by computing and comparing the checksum.

2. stream_structure_version: 1 Byte signifying the version number of
   the Ogg file format used in this stream (this document specifies
   version 0).

3. header_type_flag: the bits in this 1 Byte field identify the
   specific type of this page.

   * bit 0x01

      set: page contains data of a packet continued from the previous
         page

      unset: page contains a fresh packet

   * bit 0x02

      set: this is the first page of a logical bitstream (bos)

      unset: this page is not a first page

   * bit 0x04

      set: this is the last page of a logical bitstream (eos)

      unset: this page is not a last page

4. granule_position: an 8 Byte field containing position information.
   For example, for an audio stream, it MAY contain the total number
   of PCM samples encoded after including all frames finished on this
   page. For a video stream it MAY contain the total number of video

frames encoded after this page. This is a hint for the decoder
and gives it some timing and position information. Its meaning is
dependent on the codec for that logical bitstream and specified in
a specific media mapping. A special value of -1 (in two's
complement) indicates that no packets finish on this page.

5. bitstream_serial_number: a 4 Byte field containing the unique
   serial number by which the logical bitstream is identified.

6. page_sequence_number: a 4 Byte field containing the sequence
   number of the page so the decoder can identify page loss. This
   sequence number is increasing on each logical bitstream
   separately.

7. CRC_checksum: a 4 Byte field containing a 32 bit CRC checksum of
   the page (including header with zero CRC field and page content).
   The generator polynomial is 0x04c11db7.

8. number_page_segments: 1 Byte giving the number of segment entries
   encoded in the segment table.

9. segment_table: number_page_segments Bytes containing the lacing
   values of all segments in this page. Each Byte contains one
   lacing value.

The total header size in bytes is given by:
header_size = number_page_segments + 27 [Byte]

The total page size in Bytes is given by:
page_size = header_size + sum(lacing_values: 1..number_page_segments)
[Byte]

7. Security Considerations

The Ogg encapsulation format is a container format and only
encapsulates content (such as Vorbis-encoded audio). It does not
provide for any generic encryption or signing of itself or its
contained content bitstreams. However, it encapsulates any kind of
content bitstream as long as there is a codec for it, and is thus
able to contain encrypted and signed content data. It is also
possible to add an external security mechanism that encrypts or signs
an Ogg physical bitstream and thus provides content confidentiality
and authenticity.

As Ogg encapsulates binary data, it is possible to include executable
content in an Ogg bitstream. This can be an issue with applications
that are implemented using the Ogg format, especially when Ogg is
used for streaming or file transfer in a networking scenario. As

such, Ogg does not pose a threat there. However, an application
decoding Ogg and its encapsulated content bitstreams has to ensure
correct handling of manipulated bitstreams, of buffer overflows and
the like.

8. References

   [1] Walleij, L., "The application/ogg Media Type", RFC 3534, May
       2003.

   [2] Bradner, S., "Key words for use in RFCs to Indicate Requirement
       Levels", BCP 14, RFC 2119, March 1997.

Appendix A. Glossary of terms and abbreviations

    bos page: The initial page (beginning of stream) of a logical
        bitstream which contains information to identify the codec type
        and other decoding-relevant information.

    chaining (or sequential multiplexing): Concatenation of two or more
        complete physical Ogg bitstreams.

    eos page: The final page (end of stream) of a logical bitstream.

    granule position: An increasing position number for a specific
        logical bitstream stored in the page header. Its meaning is
        dependent on the codec for that logical bitstream and specified in
        a specific media mapping.

    grouping (or concurrent multiplexing): Interleaving of pages of
        several logical bitstreams into one complete physical Ogg
        bitstream under the restriction that all bos pages of all grouped
        logical bitstreams MUST appear before any data pages.

    lacing value: An entry in the segment table of a page header
        representing the size of the related segment.

    logical bitstream: A sequence of bits being the result of an encoded
        media stream.

    media mapping: A specific use of the Ogg encapsulation format
        together with a specific (set of) codec(s).

    (Ogg) packet: A subpart of a logical bitstream that is created by the
        encoder for that bitstream and represents a meaningful entity for
        the encoder, but only a sequence of bits to the Ogg encapsulation.

    (Ogg) page: A physical bitstream consists of a sequence of Ogg pages
        containing data of one logical bitstream only. It usually
        contains a group of contiguous segments of one packet only, but
        sometimes packets are too large and need to be split over several
        pages.

    physical (Ogg) bitstream: The sequence of bits resulting from an Ogg
        encapsulation of one or several logical bitstreams. It consists
        of a sequence of pages from the logical bitstreams with the
        restriction that the pages of one logical bitstream MUST come in
        their correct temporal order.

(Ogg) segment: The Ogg encapsulation process splits each packet into
   chunks of 255 bytes plus a last fractional chunk of less than 255
   bytes. These chunks are called segments.

Appendix B. Acknowledgements

   The author gratefully acknowledges the work that Christopher
   Montgomery and the Xiph.Org foundation have done in defining the Ogg
   multimedia project and as part of it the open file format described
   in this document. The author hopes that providing this document to
   the Internet community will help in promoting the Ogg multimedia
   project at http://www.xiph.org/. Many thanks also for the many
   technical and typo corrections that C. Montgomery and the Ogg
   community provided as feedback to this RFC.

Author's Address

   Silvia Pfeiffer
   CSIRO, Australia
   Locked Bag 17
   North Ryde, NSW 2113
   Australia

   Phone: +61 2 9325 3141
   EMail: Silvia.Pfeiffer@csiro.au
   URI: http://www.cmis.csiro.au/Silvia.Pfeiffer/

Full Copyright Statement

Acknowledgement