CELLAR Group
Internet-Draft
Intended status: Standards Track
Expires: 30 July 2024

S. Lhomme

M. Bunkus

D. Rice

27 January 2024

Matroska Media Container Chapter Codecs Specifications
draft-ietf-cellar-chapter-codecs-04

Abstract

This document defines common Matroska Chapter Codecs, the basic Matroska Script and the DVD inspired DVD menu [DVD-Video].

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 30 July 2024.

Table of Contents

1. Introduction

TODO

2. Status of this document

This document is a work-in-progress specification defining the Matroska file format as part of the IETF Cellar working group (https://datatracker.ietf.org/wg/cellar/charter/). It uses basic elements and concept already defined in the Matroska specifications defined by this workgroup [Matroska].

3. Security Considerations

Tag values can be either strings or binary blobs. This document inherits security considerations from the EBML [RFC8794] and Matroska [Matroska] documents.

4. IANA Considerations

To be determined.

5. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

6. Matroska Chapter Codecs

Chapter codecs are a way to add more complex playback features than the usual linear playback.

Some ChapProcess elements hold commands to execute when entering/ leaving a chapter.

When chapter codecs are used the EditionFlagOrdered of the edition they belong to MUST be set.

6.1. Segment Linking

Chapter Codecs can reference another Segment and jump to that Segment.

The Chapter Codecs MAY store the Segment information in their own format, possibly not using the SegmentUUID format. The ChapterTranslate element and its child elements SHOULD be used to link the internal chapter codec representation, the chapter codec number and the actual Segment it represents.

For example, if a chapter codec of type "1" in SegmentA needs to link to SegmentB, it can store that information as "SegB" in its internal data.

The translation ChapterTranslate in SegmentB would use the following elements: * ChapterTranslate\ChapterTranslateCodec = 1 * ChapterTranslate\ChapterTranslateID = "SegB"

The Matroska Player MUST use the SegmentFamily to find all Segments that need translation between the chapter codec values and the actual segment it targets.

7. Matroska Chapter Codecs and Nested Chapters

When Nested Chapters contain chapters codecs -- via the ChapProcess Element -- the enter/leave commands -- ChapProcessTime Element -- MUST be executed in a specific order, if the Matroska Player supports the chapter codecs included in the chapters.

When starting playback, the Matroska Player MUST start at the ChapterTimeStart of the first chapter of the ordered chapter. The enter commands of that chapter MUST be executed. If that chapter contains Nested Chapters, the enter commands of the Nested Chapter with the same ChapterTimeStart MUST be executed. If that chapter contains Nested Chapters, the enter commands of the Nested Chapter with the same ChapterTimeStart MUST be executed, and so on until there is no Nested Chapter with the same ChapterTimeStart.

When switching from a chapter to another:

* the leave commands (ChapProcessTime=2) of the chapter MUST be executed, then the leave commands of its parent chapter, etc. until the common Parent Chapter or Edition element. The leave command of that Parent Chapter or Edition element MUST NOT be executed.
* the enter commands (ChapProcessTime=1) of the Nested Chapter of the common Parent Chapter or Edition element, to reach the chapter we switch to, MUST be executed, then the enter commands of its Nested Chapter to reach the chapter we switch to MUST be executed, until that chapter is the chapter we switch to. The enter commands of that chapter MUST be executed as well.

When the last Chapter finished playing -- i.e. its ChapterTimeEnd has been reached -- the Matroska Player MUST execute its leaved commands, then the leave commands of it’s Parent Chapter, until the parent of the chapter is the Edition.

7.1. Matroska Script (0)

This is the case when ChapProcessCodecID = 0. This is a script language build for Matroska purposes. The inspiration comes from ActionScript, javascript and other similar scripting languages. The commands are stored as text commands, in UTF-8. The syntax is C like, with commands spanned on many lines, each terminating with a ";". You can also include comments at the end of lines with "//" or comment many lines using "/* */". The scripts are stored in ChapProcessData. For the moment ChapProcessPrivate is not used.

The one and only command existing for the moment is GotoAndPlay(ChapterUID);. As the same suggests, it means that, when this command is encountered, the Matroska Player SHOULD jump to the Chapter specified by the UID and play it.

7.2. DVD menu (1)

This is the case when ChapProcessCodecID = 1. Each level of a chapter corresponds to a logical level in the DVD system [DVD-Video] that is stored in the first octet of the ChapProcessPrivate. This DVD hierarchy is as follows:

ChapProcessPrivate	DVD Name	Hierarchy	Commands Possible	Comment
0x30	SS	DVD domain	-	First Play, Video Manager, Video Title
0x2A	LU	Language Unit	-	Contains only PGCs
0x28	TT	Title	-	Contains only PGCs
0x20	PGC	Program Group Chain (PGC)	*	
0x18	PG	Program 1 / Program 2 / Program 3	-	
0x10	PTT	Part Of Title 1 / Part Of Title 2	-	Equivalent to the chapters on the sleeve.
0x08	CN	Cell 1 / Cell 2 / Cell 3 / Cell 4 / Cell 5 / Cell 6	-	

Table 1

You can also recover wether a Segment is a Video Manager (VMG), Video Title Set (VTS) or Video Title Set Menu (VTSM) from the ChapterTranslateID element found in the Segment Info. This field uses 2 octets as follows:

1. Domain Type: 0 for VMG, the domain number for VTS and VTSM
2. Domain Value: 0 for VMG and VTSM, 1 for the VTS source.

For instance, the menu part from VTS_01_0.VOB would be coded [1,0] and the content part from VTS_02_3.VOB would be [2,1]. The VMG is always [0,0]

The following octets of ChapProcessPrivate are as follows:

Octet 1	DVD Name	Following Octets
0x30	SS	Domain name code (1: 0x00= First play, 0xC0= VMG, 0x40= VTSM, 0x80= VTS) + VTS(M) number (2)
0x2A	LU	Language code (2) + Language extension (1)
0x28	TT	global Title number (2) + corresponding TTN of the VTS (1)
0x20	PGC	PGC number (2) + Playback Type (1) + Disabled User Operations (4)
0x18	PG	Program number (2)
0x10	PTT	PTT-chapter number (1)
0x08	CN	Cell number [VOB ID(2)][Cell ID(1)][Angle Num(1)]

Table 2

If the level specified in ChapProcessPrivate is a PGC (0x20), there is an octet called the Playback Type, specifying the kind of PGC defined:

* 0x00: entry only/basic PGC
* 0x82: Title+Entry Menu (only found in the Video Manager domain)
* 0x83: Root Menu (only found in the VTSM domain)
* 0x84: Subpicture Menu (only found in the VTSM domain)
* 0x85: Audio Menu (only found in the VTSM domain)

* 0x86: Angle Menu (only found in the VTSM domain)
* 0x87: Chapter Menu (only found in the VTSM domain)

The next 4 following octets correspond to the User Operation flags in the standard PGC. When a bit is set, the command SHOULD be disabled.

ChapProcessData contains the pre/post/cell commands in binary format as there are stored on a DVD. There is just an octet preceding these data to specify the number of commands in the element. As follows: [# of commands(1)][command 1 (8)][command 2 (8)][command 3 (8)].

More information on the DVD commands and format on DVD from the [DVD-Info] project.

8. Normative References

[Matroska] Lhomme, S., Bunkus, M., and D. Rice, "Media Container Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-matroska-10, 1 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-matroska-10>.

[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>.

[RFC8794] Lhomme, S., Rice, D., and M. Bunkus, "Extensible Binary Meta Language", RFC 8794, DOI 10.17487/RFC8794, July 2020, <https://www.rfc-editor.org/info/rfc8794>.

9. Informative References

[DVD-Info] "DVD-Video Information", <http://dvd.sourceforge.net/dvdinfo/>.

[DVD-Video] DVD Forum, "DVD-Books: Part 3 DVD-Video Book", 1 November 1995, <http://www.dvdforum.org/>.

Authors' Addresses

Steve Lhomme
Email: slhomme@matroska.org

Moritz Bunkus
Email: moritz@bunkus.org

Dave Rice
Email: dave@dericed.com

cellar
Internet-Draft
Intended status: Standards Track
Expires: 5 November 2024

S. Lhomme

M. Bunkus

D. Rice

4 May 2024

Matroska Media Container Codec Specifications
draft-ietf-cellar-codec-13

Abstract

This document defines the Matroska codec mappings, including the codec ID, layout of data in a Block Element and in an optional CodecPrivate Element.

Table of Contents

1. Introduction

Matroska is a multimedia container format. It stores interleaved and timestamped audio/video/subtitle data using various codecs. To interpret the codec data, a mapping between the way the data is stored in Matroska and how it is understood by such a codec is necessary.

This document intends to define this mapping for many commonly used codecs in Matroska.

2. Status of This Document

This document is a work-in-progress specification defining the Matroska file format as part of the IETF Cellar working group (https://datatracker.ietf.org/wg/cellar/charter/). It uses basic elements and concept already defined in the Matroska specifications defined by this workgroup [Matroska].

3. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Codec Mappings

A Codec Mapping is a set of attributes to identify, name, and contextualize the format and characteristics of encoded data that can be contained within Matroska Clusters.

Each TrackEntry used within Matroska MUST reference a defined Codec Mapping using the Codec ID to identify and describe the format of the encoded data in its associated Clusters. This Codec ID is a unique registered identifier that represents the encoding stored within the

Track. Certain encodings MAY also require some form of codec initialization to provide its decoder with context and technical metadata.

The intention behind this list is not to list all existing audio and video codecs, but rather to list those codecs that are currently supported in Matroska and therefore need a well defined Codec ID so that all developers supporting Matroska will use the same Codec ID. If you feel we missed support for a very important codec, please tell us on our development mailing list (cellar at ietf.org).

4.1. Defining Matroska Codec Support

Support for a codec is defined in Matroska with the following values.

4.1.1. Codec ID

Each codec supported for storage in Matroska MUST have a unique Codec ID. Each Codec ID MUST be prefixed with the string from the following table according to the associated type of the codec. All characters of a Codec ID Prefix MUST be capital letters (A-Z) except for the last character of a Codec ID Prefix which MUST be an underscore ("_").

Codec Type	Codec ID Prefix
Video	"V_"
Audio	"A_"
Subtitle	"S_"
Button	"B_"

Table 1

Each Codec ID MUST include a Major Codec ID immediately following the Codec ID Prefix. A Major Codec ID MAY be followed by an OPTIONAL Codec ID Suffix to communicate a refinement of the Major Codec ID. If a Codec ID Suffix is used, then the Codec ID MUST include a forward slash ("/") as a separator between the Major Codec ID and the Codec ID Suffix. The Major Codec ID MUST be composed of only capital letters (A-Z) and numbers (0-9). The Codec ID Suffix MUST be composed of only capital letters (A-Z), numbers (0-9), underscore ("_"), and forward slash ("/").

The following table provides examples of valid Codec IDs and their components:

Codec ID Prefix	Major Codec ID	Separator	Codec ID Suffix	Codec ID
A_	AAC	/	MPEG2/LC/ SBR	A_AAC/MPEG2/LC/ SBR
V_	MPEG4	/	ISO/ASP	V_MPEG4/ISO/ASP
V_	MPEG1			V_MPEG1

Table 2

4.1.2. Codec Name

Each encoding supported for storage in Matroska MUST have a Codec Name. The Codec Name provides a readable label for the encoding.

4.1.3. Description

An optional description for the encoding. This value is only intended for human consumption.

4.1.4. Initialization

Each encoding supported for storage in Matroska MUST have a defined Initialization. The Initialization MUST describe the storage of data necessary to initialize the decoder, which MUST be stored within the CodecPrivate Element. When the Initialization is updated within a track, then that updated Initialization data MUST be written into the CodecState Element of the first Cluster to require it. If the encoding does not require any form of Initialization, then none MUST be used to define the Initialization and the CodecPrivate Element SHOULD NOT be written and MUST be ignored. Data that is defined Initialization to be stored in the CodecPrivate Element is known as Private Data.

4.1.5. Codec BlockAdditions

Additional data that contextualizes or supplements a Block can be stored within the BlockAdditional Element of a BlockMore Element. This BlockAdditional data MAY be passed to the associated decoder along with the content of the Block Element. Each BlockAdditional is coupled with a BlockAddID that identifies the kind of data it contains. The following table defines the meanings of BlockAddID values.

BlockAddID Value	Definition
0	Invalid.
1	Indicates that the context of the BlockAdditional data is defined by the corresponding Codec Mapping.
2 or greater	BlockAddID values of 2 and greater are mapped to the BlockAddIDValue of the BlockAdditionMapping of the associated Track.

Table 3

The values of BlockAddID that are 2 of greater have no semantic meaning, but simply associate the BlockMore Element with a BlockAdditionMapping of the associated Track. See Section 6 on Block Additional Mappings for more information.

The following XML depicts the nested Elements of a BlockGroup Element with an example of BlockAdditions:

```
<BlockGroup>
  <Block>{Binary data of a VP9 video frame in YUV}</Block>
  <BlockAdditions>
    <BlockMore>
      <BlockAddID>1</BlockAddID>
      <BlockAdditional>
        {alpha channel encoding to supplement the VP9 frame}
      </BlockAdditional>
    </BlockMore>
  </BlockAdditions>
</BlockGroup>
```

4.1.6. Citation

Documentation of the associated normative and informative references for the codec is RECOMMENDED.

4.1.7. Deprecation Date

A timestamp, expressed in [RFC3339] that notes when support for the Codec Mapping within Matroska was deprecated. If a Codec Mapping is defined with a Deprecation Date, then it is RECOMMENDED that Matroska Writers SHOULD NOT use the Codec Mapping after the Deprecation Date.

4.1.8. Superseded By

A Codec Mapping MAY only be defined with a Superseded By value, if it has an expressed Deprecation Date. If used, the Superseded By value MUST store the Codec ID of another Codec Mapping that has superseded the Codec Mapping.

4.2. Recommendations for the Creation of New Codec Mappings

Creators of new Codec Mappings to be used in the context of Matroska:

* SHOULD assume that all Codec Mappings they create might become standardized, public, commonly deployed, or usable across multiple implementations.

* SHOULD employ meaningful values for Codec ID and Codec Name that they have reason to believe are currently unused.

* SHOULD NOT prefix their Codec ID with "X_" or similar constructs.

These recommendations are based on Section 3 of [RFC6648].

4.3. Video Codec Mappings

4.3.1. V_MS/VFW/FOURCC

Codec ID: V_MS/VFW/FOURCC

Codec Name: Microsoft (TM) Video Codec Manager (VCM)

Description: The private data contains the VCM structure BITMAPINFOHEADER including the extra private bytes, as defined by Microsoft (https://msdn.microsoft.com/en-us/library/windows/desktop/dd318229(v=vs.85).aspx). The data are stored in little-endian format (like on IA32 machines). Where is the Huffman table stored in HuffYUV, not AVISTREAMINFO ??? And the FourCC, not in AVISTREAMINFO.fccHandler ???

Initialization: Private Data contains the VCM structure BITMAPINFOHEADER including the extra private bytes, as defined by Microsoft in https://msdn.microsoft.com/en-us/library/windows/desktop/dd183376(v=vs.85).aspx (https://msdn.microsoft.com/en-us/library/windows/desktop/dd183376(v=vs.85).aspx).

Citation: https://msdn.microsoft.com/en-us/library/windows/desktop/dd183376(v=vs.85).aspx (https://msdn.microsoft.com/en-us/library/windows/desktop/dd183376(v=vs.85).aspx)

4.3.2. V_UNCOMPRESSED

Codec ID: V_UNCOMPRESSED

Codec Name: Video, raw uncompressed video frames

Description: All details about the used color specs and bit depth are to be put/read from the TrackEntry\Video\UncompressedFourCC elements.

Initialization: none

4.3.3. V_MPEG4/ISO/SP

Codec ID: V_MPEG4/ISO/SP

Codec Name: MPEG4 ISO simple profile (DivX4)

Description: Stream was created via improved codec API (UCI) or even transmuxed from AVI (no b-frames in Simple Profile), frame order is coding order.

Initialization: none

4.3.4. V_MPEG4/ISO/ASP

Codec ID: V_MPEG4/ISO/ASP

Codec Name: MPEG4 ISO advanced simple profile (DivX5, XviD, FFMPEG)

Description: Stream was created via improved codec API (UCI) or transmuxed from MP4, not simply transmuxed from AVI. Note there are differences how b-frames are handled in these original streams, when being compared to a VfW created stream, as here there are no dummy frames inserted, the frame order is exactly the same as the coding order, same as in MP4 streams.

Initialization: none

4.3.5. V_MPEG4/ISO/AP

Codec ID: V_MPEG4/ISO/AP

Codec Name: MPEG4 ISO advanced profile

Description: Stream was created via improved codec API (UCI) or transmuxed from MP4, not simply transmuxed from AVI. Note there are differences how b-frames are handled in these original streams, when being compared to a VfW created stream, as here there are no dummy frames inserted, the frame order is exactly the same as the coding order, same as in MP4 streams.

Initialization: none

4.3.6. V_MPEG4/MS/V3

Codec ID: V_MPEG4/MS/V3

Codec Name: Microsoft (TM) MPEG4 V3

Description: Microsoft (TM) MPEG4 V3 and derivates, means DivX3, Angelpotion, SMR, etc.; stream was created using VfW codec or transmuxed from AVI; note that V1/V2 are covered in VfW compatibility mode.

Initialization: none

4.3.7. V_MPEG1

Codec ID: V_MPEG1

Codec Name: MPEG 1

Description: The Matroska video stream will contain a demuxed Elementary Stream (ES), where block boundaries are still to be defined. It’s RECOMMENDED to use MPEG2MKV.exe for creating those files, and to compare the results with self-made implementations

Initialization: none

4.3.8. V_MPEG2

Codec ID: V_MPEG2

Codec Name: MPEG 2

Description: The Matroska video stream will contain a demuxed Elementary Stream (ES), where block boundaries are still to be defined. It's RECOMMENDED to use MPEG2MKV.exe for creating those files, and to compare the results with self-made implementations

Initialization: none

4.3.9. V_MPEG4/ISO/AVC

Codec ID: V_MPEG4/ISO/AVC

Codec Name: AVC/H.264

Description: Individual pictures (which could be a frame, a field, or 2 fields having the same timestamp) of AVC/H.264 stored as described in [ISO.14496-15].

Initialization: The Private Data contains a AVCDecoderConfigurationRecord structure, as defined in [ISO.14496-15]. For legacy reasons, because Block Addition Mappings are preferred; see Section 4.7, the AVCDecoderConfigurationRecord structure MAY be followed by an extension block beginning with a 4-byte extension block size field in big-endian byte order which is the size of the extension block minus 4 (excluding the size of the extension block size field) and a 4-byte field corresponding to a BlockAddIDType of "mvcC" followed by a content corresponding to the content of BlockAddIDExtraData for mvcC; see Section 4.7.8.

4.3.10. V_MPEGH/ISO/HEVC

Codec ID: V_MPEGH/ISO/HEVC

Codec Name: HEVC/H.265

Description: Individual pictures (which could be a frame, a field, or 2 fields having the same timestamp) of HEVC/H.265 stored as described in [ISO.14496-15].

Initialization: The Private Data contains a HEVCDecoderConfigurationRecord structure, as defined in [ISO.14496-15].

4.3.11. V_MPEGI/ISO/VVC

Codec ID: V_MPEGI/ISO/VVC

Codec Name: VVC/H.266

Description: Individual pictures (which could be a frame, a field, or 2 fields having the same timestamp) of VVC/H.266 stored as described in [ISO.14496-15].

Initialization: The Private Data contains a VVCDecoderConfigurationRecord structure, as defined in [ISO.14496-15].

4.3.12. V_AVS2

Codec ID: V_AVS2

Codec Name: AVS2-P2/IEEE.1857.4

Description: Individual pictures of AVS2-P2 stored as described in the second part of [IEEE.1857-4].

Initialization: none.

4.3.13. V_AVS3

Codec ID: V_AVS3

Codec Name: AVS3-P2/IEEE.1857.10

Description: Individual pictures of AVS3-P2 stored as described in the second part of [IEEE.1857-10].

Initialization: none.

4.3.14. V_REAL/RV10

Codec ID: V_REAL/RV10

Codec Name: RealVideo 1.0 aka RealVideo 5

Description: Individual slices from the Real container are combined into a single frame.

Initialization: The Private Data contains a real_video_props_t structure in big-endian byte order as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.3.15. V_REAL/RV20

Codec ID: V_REAL/RV20

Codec Name: RealVideo G2 and RealVideo G2+SVT

Description: Individual slices from the Real container are combined into a single frame.

Initialization: The Private Data contains a real_video_props_t structure in big-endian byte order as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.3.16. V_REAL/RV30

Codec ID: V_REAL/RV30

Codec Name: RealVideo 8

Description: Individual slices from the Real container are combined into a single frame.

Initialization: The Private Data contains a real_video_props_t structure in big-endian byte order as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.3.17. V_REAL/RV40

Codec ID: V_REAL/RV40

Codec Name: rv40 : RealVideo 9

Description: Individual slices from the Real container are combined into a single frame.

Initialization: The Private Data contains a real_video_props_t structure in big-endian byte order as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.3.18. V_QUICKTIME

Codec ID: V_QUICKTIME

Codec Name: Video taken from QuickTime(TM) files

Description: Several codecs as stored in QuickTime (e.g., Sorenson or Cinepak).

Initialization: The Private Data contains all additional data that is stored in the 'stsd' (sample description) atom in the QuickTime file *after* the mandatory video descriptor structure (starting with the size and FourCC fields). For an explanation of the QuickTime file format read QuickTime File Format Specification (https://developer.apple.com/library/mac/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html).

4.3.19. V_THEORA

Codec ID: V_THEORA

Codec Name: Theora

Initialization: The Private Data contains the first three Theora packets in order. The lengths of the packets precedes them. The actual layout is:

* Byte 1: number of distinct packets #p minus one inside the CodecPrivate block. This MUST be "2" for current (as of 2016-07-08) Theora headers.
* Bytes 2..n: lengths of the first #p packets, coded in Xiph-style lacing. The length of the last packet is the length of the CodecPrivate block minus the lengths coded in these bytes minus one.
* Bytes n+1..: The Theora identification header, followed by the commend header followed by the codec setup header. Those are described in the Theora specs (http://www.theora.org/doc/Theora.pdf).

4.3.20. V_PRORES

Codec ID: V_PRORES

Codec Name: Apple ProRes

Initialization: The Private Data contains the FourCC as found in MP4 movies:

* ap4x: ProRes 4444 XQ
* ap4h: ProRes 4444
* apch: ProRes 422 High Quality
* apcn: ProRes 422 Standard Definition
* apcs: ProRes 422 LT
* apco: ProRes 422 Proxy
* aprh: ProRes RAW High Quality
* aprn: ProRes RAW Standard Definition

this page for more technical details on ProRes (http://wiki.multimedia.cx/index.php?title=Apple_ProRes#Frame_layout)

4.3.21. V_VP8

Codec ID: V_VP8

Codec Name: VP8 Codec format

Description: VP8 is an open and royalty free video compression format developed by Google and created by On2 Technologies as a successor to VP7. [RFC6386]

Codec BlockAdditions: A single-channel encoding of an alpha channel MAY be stored in BlockAdditions. The BlockAddId of the BlockMore containing these data MUST be 1.

Initialization: none

4.3.22. V_VP9

Codec ID: V_VP9

Codec Name: VP9 Codec format

Description: VP9 is an open and royalty free video compression format developed by Google as a successor to VP8. Draft VP9 Bitstream and Decoding Process Specification (https://www.webmproject.org/vp9/)

Codec BlockAdditions: A single-channel encoding of an alpha channel MAY be stored in BlockAdditions. The BlockAddId of the BlockMore containing these data MUST be 1.

Initialization: none

4.3.23. V_FFV1

Codec ID: V_FFV1

Codec Name: FF Video Codec 1

Description: FFV1 is a lossless intra-frame video encoding format designed to efficiently compress video data in a variety of pixel formats. Compared to uncompressed video, FFV1 offers storage compression, frame fixity, and self-description, which makes FFV1 useful as a preservation or intermediate video format. Draft FFV1 Specification (https://datatracker.ietf.org/doc/draft-ietf-cellar-ffv1/)

Initialization: For FFV1 versions 0 or 1, Private Data SHOULD NOT be written. For FFV1 version 3 or greater, the Private Data MUST contain the FFV1 Configuration Record structure, as defined in https://tools.ietf.org/html/draft-ietf-cellar-ffv1-04#section-4.2 (https://tools.ietf.org/html/draft-ietf-cellar-ffv1-04#section-4.2), and no other data.

4.4. Audio Codec Mappings

4.4.1. A_MPEG/L3

Codec ID: A_MPEG/L3

Codec Name: MPEG Audio 1, 2, 2.5 Layer III

Description: The data contain everything needed for playback in the MPEG Audio header of each frame. Corresponding ACM wFormatTag : 0x0055

Initialization: none

4.4.2. A_MPEG/L2

Codec ID: A_MPEG/L2

Codec Name: MPEG Audio 1, 2 Layer II

Description: The data contain everything needed for playback in the MPEG Audio header of each frame. Corresponding ACM wFormatTag : 0x0050

Initialization: none

4.4.3. A_MPEG/L1

Codec ID: A_MPEG/L1

Codec Name: MPEG Audio 1, 2 Layer I

Description: The data contain everything needed for playback in the MPEG Audio header of each frame. Corresponding ACM wFormatTag : 0x0050

Initialization: none

4.4.4. A_PCM/INT/BIG

Codec ID: A_PCM/INT/BIG

Codec Name: PCM Integer Big Endian

Description: The audio bit depth MUST be read and set from the BitDepth Element. Audio samples MUST be considered as signed values, except if the audio bit depth is 8 which MUST be interpreted as unsigned values. Corresponding ACM wFormatTag : ???

Initialization: none

4.4.5. A_PCM/INT/LIT

Codec ID: A_PCM/INT/LIT

Codec Name: PCM Integer Little Endian

Description: The audio bit depth MUST be read and set from the BitDepth Element. Audio samples MUST be considered as signed values, except if the audio bit depth is 8 which MUST be interpreted as unsigned values. Corresponding ACM wFormatTag : 0x0001

Initialization: none

4.4.6. A_PCM/FLOAT/IEEE

Codec ID: A_PCM/FLOAT/IEEE

Codec Name: Floating-Point, IEEE compatible

Description: The audio bit depth MUST be read and set from the BitDepth Element (32 bit in most cases). The floats are stored as defined in [IEEE.754] and in little-endian order. Corresponding ACM wFormatTag : 0x0003

Initialization: none

4.4.7. A_MPC

Codec ID: A_MPC

Codec Name: MPC (musepack) SV8

Description: The main developer for musepack has requested that we wait until the SV8 framing has been fully defined for musepack before defining how to store it in Matroska.

4.4.8. A_AC3

Codec ID: A_AC3

Codec Name: (Dolby) AC3

Description: BSID <= 8 !! The private data is void ??? Corresponding ACM wFormatTag : 0x2000 ; channel number have to be read from the corresponding audio element

4.4.9. A_AC3/BSID9

Codec ID: A_AC3/BSID9

Codec Name: (Dolby) AC3

Description: The ac3 frame header has, similar to the mpeg-audio header a version field. Normal ac3 is defined as bitstream id 8 (5 Bits, numbers are 0-15). Everything below 8 is still compatible with all decoders that handle 8 correctly. Everything higher are additions that break decoder compatibility. For the samplerates 24kHz (00); 22,05kHz (01) and 16kHz (10) the BSID is 9 For the samplerates 12kHz (00); 11,025kHz (01) and 8kHz (10) the BSID is 10

Initialization: none

4.4.10. A_AC3/BSID10

Codec ID: A_AC3/BSID10

Codec Name: (Dolby) AC3

Description: The ac3 frame header has, similar to the mpeg-audio header a version field. Normal ac3 is defined as bitstream id 8 (5 Bits, numbers are 0-15). Everything below 8 is still compatible with all decoders that handle 8 correctly. Everything higher are additions that break decoder compatibility. For the samplerates 24kHz (00); 22,05kHz (01) and 16kHz (10) the BSID is 9 For the samplerates 12kHz (00); 11,025kHz (01) and 8kHz (10) the BSID is 10

Initialization: none

4.4.11. A_ALAC

Codec ID: A_ALAC

Codec Name: ALAC (Apple Lossless Audio Codec)

Initialization: The Private Data contains ALAC's magic cookie (both the codec specific configuration as well as the optional channel layout information). Its format is described in ALAC's official source code (http://alac.macosforge.org/trac/browser/trunk/ALACMagicCookieDescription.txt).

4.4.12. A_DTS

Codec ID: A_DTS

Codec Name: Digital Theatre System

Description: Supports DTS, DTS-ES, DTS-96/26, DTS-HD High Resolution Audio and DTS-HD Master Audio. The private data is void. Corresponding ACM wFormatTag : 0x2001

Initialization: none

4.4.13. A_DTS/EXPRESS

Codec ID: A_DTS/EXPRESS

Codec Name: Digital Theatre System Express

Description: DTS Express (a.k.a. LBR) audio streams. The private data is void. Corresponding ACM wFormatTag : 0x2001

Initialization: none

4.4.14. A_DTS/LOSSLESS

Codec ID: A_DTS/LOSSLESS

Codec Name: Digital Theatre System Lossless

Description: DTS Lossless audio that does not have a core substream. The private data is void. Corresponding ACM wFormatTag : 0x2001

Initialization: none

4.4.15. A_VORBIS

Codec ID: A_VORBIS

Codec Name: Vorbis

Initialization: The Private Data contains the first three Vorbis packet in order. The lengths of the packets precedes them. The actual layout is: - Byte 1: number of distinct packets #p minus one inside the CodecPrivate block. This MUST be "2" for current (as of 2016-07-08) Vorbis headers. - Bytes 2..n: lengths of the first #p packets, coded in Xiph-style lacing. The length of the last packet is the length of the CodecPrivate block minus the lengths coded in these bytes minus one. - Bytes n+1..: The Vorbis identification header (https://xiph.org/vorbis/doc/Vorbis_I_spec.html), followed by the Vorbis comment header (https://xiph.org/vorbis/doc/v-comment.html) followed by the codec setup header (https://xiph.org/vorbis/doc/Vorbis_I_spec.html).

4.4.16. A_FLAC

Codec ID: A_FLAC

Codec Name: FLAC (Free Lossless Audio Codec) (http://flac.sourceforge.net/)

Initialization: The Private Data contains all the header/metadata packets before the first data packet. These include the first header packet containing only the word fLaC as well as all metadata packets.

4.4.17. A_REAL/14_4

Codec ID: A_REAL/14_4

Codec Name: Real Audio 1

Initialization: The Private Data contains either the "real_audio_v4_props_t" or the "real_audio_v5_props_t" structure (differentiated by their "version" field; big-endian byte order) as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.4.18. A_REAL/28_8

Codec ID: A_REAL/28_8

Codec Name: Real Audio 2

Initialization: The Private Data contains either the "real_audio_v4_props_t" or the "real_audio_v5_props_t" structure (differentiated by their "version" field; big-endian byte order) as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.4.19. A_REAL/COOK

Codec ID: A_REAL/COOK

Codec Name: Real Audio Cook Codec (codename: Gecko)

Initialization: The Private Data contains either the "real_audio_v4_props_t" or the "real_audio_v5_props_t" structure (differentiated by their "version" field; big-endian byte order) as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.4.20. A_REAL/SIPR

Codec ID: A_REAL/SIPR

Codec Name: Sipro Voice Codec

Initialization: The Private Data contains either the "real_audio_v4_props_t" or the "real_audio_v5_props_t" structure (differentiated by their "version" field; big-endian byte order) as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.4.21. A_REAL/RALF

Codec ID: A_REAL/RALF

Codec Name: Real Audio Lossless Format

Initialization: The Private Data contains either the "real_audio_v4_props_t" or the "real_audio_v5_props_t" structure (differentiated by their "version" field; big-endian byte order) as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.4.22. A_REAL/ATRC

Codec ID: A_REAL/ATRC

Codec Name: Sony Atrac3 Codec

Initialization: The Private Data contains either the "real_audio_v4_props_t" or the "real_audio_v5_props_t" structure (differentiated by their "version" field; big-endian byte order) as found in librmff (https://github.com/mbunkus/mkvtoolnix/blob/master/lib/librmff/librmff.h).

4.4.23. A_MS/ACM

Codec ID: A_MS/ACM

Codec Name: Microsoft(TM) Audio Codec Manager (ACM)

Description: The data are stored in little-endian format (like on IA32 machines).

Initialization: The Private Data contains the [WAVEFORMATEX] structure including the extra format information bytes. The structure is stored without packing or padding bytes. A WORD corresponds to a signed 2 octets integer, DWORD corresponds to a signed 4 octets integer. The extra format information are appended after the WAVEFORMATEX octets.

4.4.24. A_AAC/MPEG2/MAIN

Codec ID: A_AAC/MPEG2/MAIN

Codec Name: MPEG2 Main Profile

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.25. A_AAC/MPEG2/LC

Codec ID: A_AAC/MPEG2/LC

Codec Name: Low Complexity

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.26. A_AAC/MPEG2/LC/SBR

Codec ID: A_AAC/MPEG2/LC/SBR

Codec Name: Low Complexity with Spectral Band Replication

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.27. A_AAC/MPEG2/SSR

Codec ID: A_AAC/MPEG2/SSR

Codec Name: Scalable Sampling Rate

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.28. A_AAC/MPEG4/MAIN

Codec ID: A_AAC/MPEG4/MAIN

Codec Name: MPEG4 Main Profile

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.29. A_AAC/MPEG4/LC

Codec ID: A_AAC/MPEG4/LC

Codec Name: Low Complexity

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.30. A_AAC/MPEG4/LC/SBR

Codec ID: A_AAC/MPEG4/LC/SBR

Codec Name: Low Complexity with Spectral Band Replication

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.31. A_AAC/MPEG4/SSR

Codec ID: A_AAC/MPEG4/SSR

Codec Name: Scalable Sampling Rate

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.32. A_AAC/MPEG4/LTP

Codec ID: A_AAC/MPEG4/LTP

Codec Name: Long Term Prediction

Description: Channel number and sample rate have to be read from the corresponding audio element. Audio stream is stripped from ADTS headers and normal Matroska frame based muxing scheme is applied. AAC audio always uses wFormatTag 0xFF.

Initialization: none

4.4.33. A_QUICKTIME

Codec ID: A_QUICKTIME

Codec Name: Audio taken from QuickTime(TM) files

Description: Several codecs as stored in QuickTime (e.g., QDesign Music v1 or v2).

Initialization: The Private Data contains all additional data that is stored in the 'stsd' (sample description) atom in the QuickTime file *after* the mandatory sound descriptor structure (starting with the size and FourCC fields). For an explanation of the QuickTime file format read QuickTime File Format Specification (https://developer.apple.com/library/mac/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html).

4.4.34. A_QUICKTIME/QDMC

Codec ID: A_QUICKTIME/QDMC

Codec Name: QDesign Music

Description:

Initialization: The Private Data contains all additional data that is stored in the 'stsd' (sample description) atom in the QuickTime file *after* the mandatory sound descriptor structure (starting with the

size and FourCC fields). For an explanation of the QuickTime file format read QuickTime File Format Specification (https://developer.apple.com/library/mac/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html).

Superseded By: A_QUICKTIME

4.4.35. A_QUICKTIME/QDM2

Codec ID: A_QUICKTIME/QDM2

Codec Name: QDesign Music v2

Description:

Initialization: The Private Data contains all additional data that is stored in the 'stsd' (sample description) atom in the QuickTime file *after* the mandatory sound descriptor structure (starting with the size and FourCC fields). For an explanation of the QuickTime file format read QuickTime File Format Specification (https://developer.apple.com/library/mac/documentation/QuickTime/QTFF/QTFFPreface/qtffPreface.html).

Superseded By: A_QUICKTIME

4.4.36. A_TTA1

Codec ID: A_TTA1

Codec Name: The True Audio (http://tausoft.org/) lossless audio compressor

Description: TTA format description (http://tausoft.org/wiki/True_Audio_Codec_Format) Each frame is kept intact, including the CRC32. The header and seektable are dropped. SamplingFrequency, Channels and BitDepth are used in the TrackEntry. wFormatTag = 0x77A1

Initialization: none

4.4.37. A_WAVPACK4

Codec ID: A_WAVPACK4

Codec Name: WavPack (http://www.wavpack.com/) lossless audio compressor

Description: The Wavpack packets consist of a stripped header followed by the frame data. For multi-track (> 2 tracks) a frame consists of many packets. For more details, check the WavPack muxing description (wavpack.html).

Codec BlockAdditions: For hybrid A_WAVPACK4 encodings (that include a lossy encoding with a supplemental correction to produce a lossless encoding), the correction part is stored in BlockAdditional. The BlockAddId of the BlockMore containing these data MUST be 1.

Initialization: none

4.4.38. A_ATRAC/AT1

Codec ID: A_ATRAC/AT1

Codec Name: Sony ATRAC1 Codec

Description: The original ATRAC codec by Sony, mainly used in MiniDisc platforms. The core technical details on ATRAC1 can be found in [AtracAES]. An example encoder/decoder can be found at [atracdenc].

Initialization: None

4.5. Subtitle Codec Mappings

4.5.1. S_TEXT/UTF8

Codec ID: S_TEXT/UTF8

Codec Name: UTF-8 Plain Text

Description: Basic text subtitles. For more information see Section 5 on Subtitles.

4.5.2. S_TEXT/SSA

Codec ID: S_TEXT/SSA

Codec Name: Subtitles Format

Description: The [Script Info] and [V4 Styles] sections are stored in the codecprivate. Each event is stored in its own Block. For more information see Section 5.3 on SSA/ASS.

4.5.3. S_TEXT/ASS

Codec ID: S_TEXT/ASS

Codec Name: Advanced Subtitles Format

Description: The [Script Info] and [V4 Styles] sections are stored in the codecprivate. Each event is stored in its own Block. For more information see Section 5.3 on SSA/ASS.

4.5.4. S_TEXT/WEBVTT

Codec ID: S_TEXT/WEBVTT

Codec Name: Web Video Text Tracks Format (WebVTT)

Description: Advanced text subtitles. For more information see Section 5.4 on WebVTT.

4.5.5. S_IMAGE/BMP

Codec ID: S_IMAGE/BMP

Codec Name: Bitmap

Description: Basic image based subtitle format; The subtitles are stored as images, like in the DVD [DVD-Video]. The timestamp in the block header of Matroska indicates the start display time, the duration is set with the Duration element. The full data for the subtitle bitmap is stored in the Block’s data section.

4.5.6. S_DVBSUB

Codec ID: S_DVBSUB

Codec Name: Digital Video Broadcasting (DVB) subtitles

Description: This is the graphical subtitle format used in the Digital Video Broadcasting standard. For more information see Section 5.7 on Digital Video Broadcasting (DVB).

4.5.7. S_VOBSUB

Codec ID: S_VOBSUB

Codec Name: VobSub subtitles

Description: The same subtitle format used on DVDs [DVD-Video]. Supported is only format version 7 and newer. VobSubs consist of two files, the .idx containing information, and the .sub, containing the actual data. The .idx file is stripped of all empty lines, of all comments and of lines beginning with alt: or langidx:. The line beginning with id: SHOULD be transformed into the appropriate Matroska track language element and is discarded. All remaining lines but the ones containing timestamps and file positions are put into the CodecPrivate element.

For each line containing the timestamp and file position data is read from the appropriate position in the .sub file. This data consists of a MPEG program stream which in turn contains SPU packets. The MPEG program stream data is discarded, and each SPU packet is put into one Matroska frame.

4.5.8. S_HDMV/PGS

Codec ID: S_HDMV/PGS

Codec Name: HDMV presentation graphics subtitles (PGS)

Description: This is the graphical subtitle format used on Blu-rays. For more information, see Section 5.6 on HDMV text presentation.

4.5.9. S_HDMV/TEXTST

Codec ID: S_HDMV/TEXTST

Codec Name: HDMV text subtitles

Description: This is the textual subtitle format used on Blu-rays. For more information, see Section 5.5 on HDMV graphics presentation.

4.5.10. S_KATE

Codec ID: S_KATE

Codec Name: Karaoke And Text Encapsulation

Description: A subtitle format developed for ogg. The mapping for Matroska is described on the Xiph wiki (http://wiki.xiph.org/index.php/OggKate#Matroska_mapping). As for Theora and Vorbis, Kate headers are stored in the private data as xiph-laced packets.

4.5.11. S_ARIBSUB

Codec ID: S_ARIBSUB

Codec Name: ARIB STD-B24 subtitles

Description: This is the textual subtitle format used in the ISDB/ ARIB broadcasting standard. For more information see Section 5.8 on ARIB (ISDB) subtitles.

4.6. Button Codec Mappings

4.6.1. B_VOBBTN

Codec ID: B_VOBBTN

Codec Name: VobBtn Buttons

Description: Based on MPEG/VOB PCI packets (http://dvd.sourceforge.net/dvdinfo/pci_pkt.html). The file contains a header consisting of the string "butonDVD" followed by the width and height in pixels (16-bit integer each) and 4 reserved bytes. The rest is full PCI packets (http://dvd.sourceforge.net/dvdinfo/ pci_pkt.html).

4.7. Block Addition Mappings

Registered BlockAddIDType are:

4.7.1. Use BlockAddIDValue

Block type identifier: 0

Block type name: Use BlockAddIDValue

Description: This value indicates that the actual type is stored in BlockAddIDValue instead. This value is expected to be used when it is important to have a strong compatibility with players or derived formats not supporting BlockAdditionMapping but using BlockAdditions with an unknown BlockAddIDValue, and SHOULD NOT be used if it is possible to use another value.

4.7.2. Opaque data

Block type identifier: 1

Block type name: Opaque data

Description: the BlockAdditional data is interpreted as opaque additional data passed to the codec with the Block data. BlockAddIDValue MUST be 1.

4.7.3. ITU T.35 metadata

Block type identifier: 4

Block type name: ITU T.35 metadata

Description: the BlockAdditional data is interpreted as ITU T.35 metadata, as defined by ITU-T T.35 terminal codes. BlockAddIDValue MUST be 4.

4.7.4. avcE

Block type identifier: 0x61766345

Block type name: Dolby Vision enhancement-layer AVC configuration

Description: the BlockAddIDExtraData data is interpreted as the Dolby Vision enhancement-layer AVC configuration box as described in [DolbyVisionWithinIso]. This extension MUST NOT be used if Codec ID is not V_MPEG4/ISO/AVC.

4.7.5. dvcC

Block type identifier: 0x64766343

Block type name: Dolby Vision configuration

Description: the BlockAddIDExtraData data is interpreted as DOVIDecoderConfigurationRecord structure, as defined in [DolbyVisionWithinIso], for Dolby Vision profiles less than and equal to 7.

4.7.6. dvvC

Block type identifier: 0x64767643

Block type name: Dolby Vision configuration

Description: the BlockAddIDExtraData data is interpreted as DOVIDecoderConfigurationRecord structure, as defined in [DolbyVisionWithinIso], for Dolby Vision profiles greater than 7.

4.7.7. hvcE

Block type identifier: 0x68766345

Block type name: Dolby Vision enhancement-layer HEVC configuration

Description: the BlockAddIDExtraData data is interpreted as the Dolby Vision enhancement-layer HEVC configuration as described in [DolbyVisionWithinIso]. This extension MUST NOT be used if Codec ID is not V_MPEGH/ISO/HEVC.

4.7.8. mvcC

Block type identifier: 0x6D766343

Block type name: MVC configuration

Description: the BlockAddIDExtraData data is interpreted as MVCDecoderConfigurationRecord structure, as defined in [ISO.14496-15]. This extension MUST NOT be used if Codec ID is not V_MPEG4/ISO/AVC.

5. Subtitles

Because Matroska is a general container format, we try to avoid specifying the formats to store in it. This type of work is really outside of the scope of a container-only format. However, because the use of subtitles in A/V containers has been so limited (with the exception of DVD) we are taking the time to specify how to store some of the more common subtitle formats in Matroska. This is being done to help facilitate their growth. Otherwise, incompatibilities could prevent the standardization and use of subtitle storage.

This page is not meant to be a complete listing of all subtitle formats that will be used in Matroska, it is only meant to be a guide for the more common, current formats. It is possible that we will add future formats to this page as they are created, but it is not likely as any other new subtitle format designer would likely have their own specifications. Any specification listed here SHOULD be strictly adhered to or it SHOULD NOT use the corresponding Codec ID.

Here is a list of pointers for storing subtitles in Matroska:

* Any Matroska file containing only subtitles SHOULD use the extension ".mks".
* As a general rule of thumb for all codecs, information that is global to an entire stream SHOULD be stored in the CodecPrivate element.

* Start and stop timestamps that are used in a timestamps original storage format SHOULD be removed when being placed in Matroska as they could interfere if the file is edited afterwards. Instead, the Blocks timestamp and Duration SHOULD be used to say when the timestamp is displayed.
* Because a "subtitle" stream is actually just an overlay stream, anything with a transparency layer could be use, including video.

5.1. Images Subtitles

The first image format that is a goal to import into Matroska is the VobSub subtitle format. This subtitle type is generated by exporting the subtitles from a DVD [DVD-Video].

The requirement for muxing VobSub into Matroska is v7 subtitles (see first line of the .IDX file). If the version is smaller, you must remux them using the SubResync utility from VobSub 2.23 (or MPC) into v7 format. Generally any newly created subs will be in v7 format.

The .IFO file will not be used at all.

If there is more than one subtitle stream in the VobSub set, each stream will need to be separated into separate tracks for storage in Matroska. E.g. the VobSub file contains streams for both English and German subtitles. Then the resulting Matroska file SHOULD contain two tracks. That way the language information can be dropped and mapped to Matroska's language tags.

The .IDX file is reformatted (see below) and placed in the CodecPrivate.

Each .BMP will be stored in its own Block. The Timestamp with be stored in the Blocks Timestamp and the duration will be stored in the Default Duration.

Here is an example .IDX file:

```
# VobSub index file, v7 (do not modify this line!)
#
# To repair desynchronization, you can insert gaps this way:
# (it usually happens after vob id changes)
#
# delay: [sign]hh:mm:ss:ms
#
# Where:
# [sign]: +, - (optional)
# hh: hours (0 <= hh)
# mm/ss: minutes/seconds (0 <= mm/ss <= 59)
```

```
    # ms: milliseconds (0 <= ms <= 999)
    #
    # Note: You can't position a sub before the previous with a negative
    # value.
    #
    # You can also modify timestamps or delete a few subs you don't
    # like. Just make sure they stay in increasing order.

    # Settings

    # Original frame size
    size: 720x480

    # Origin, relative to the upper-left corner, can be overloaded by
    # alignment
    org: 0, 0

    # Image scaling (hor,ver), origin is at the upper-left corner or at
    # the alignment coord (x, y)
    scale: 100%, 100%

    # Alpha blending
    alpha: 100%

    # Smoothing for very blocky images (use OLD for no filtering)
    smooth: OFF

    # In millisecs
    fadein/out: 50, 50

    # Force subtitle placement relative to (org.x, org.y)
    align: OFF at LEFT TOP

    # For correcting non-progressive desync. (in millisecs or
    # hh:mm:ss:ms)
    # Note: Not effective in DirectVobSub, use "delay: ... " instead.
    time offset: 0

    # ON: displays only forced subtitles, OFF: shows everything
    forced subs: OFF

    # The original palette of the DVD
    palette: 000000, 7e7e7e, fbff8b, cb86f1, 7f74b8, e23f06, 0a48ea, \
    b3d65a, 6b92f1, 87f087, c02081, f8d0f4, e3c411, 382201, e8840b, \
    fdfdfd

    # Custom colors (transp idxs and the four colors)
    custom colors: OFF, tridx: 0000, colors: 000000, 000000, 000000, \
```

```
000000

# Language index in use
langidx: 0

# English
id: en, index: 0
# Uncomment next line to activate alternative name in DirectVobSub /
# Windows Media Player 6.x
# alt: English
# Vob/Cell ID: 1, 1 (PTS: 0)
timestamp: 00:00:01:101, filepos: 000000000
timestamp: 00:00:08:708, filepos: 000001000
```

First, lines beginning with "#" are removed. These are comments to make text file editing easier, and as this is not a text file, they aren't needed.

Next remove the "langidx" and "id" lines. These are used to differentiate the subtitle streams and define the language. As the streams will be stored separately anyway, there is no need to differentiate them here. Also, the language setting will be stored in the Matroska tags, so there is no need to store it here.

Finally, the "timestamp" will be used to set the Block's timestamp. Once it is set there, there is no need for it to be stored here. Also, as it may interfere if the file is edited, it SHOULD NOT be stored here.

Once all of these items are removed, the data to store in the CodecPrivate SHOULD look like this:

```
size: 720x480
org: 0, 0
scale: 100%, 100%
alpha: 100%
smooth: OFF
fadein/out: 50, 50
align: OFF at LEFT TOP
time offset: 0
forced subs: OFF
palette: 000000, 7e7e7e, fbff8b, cb86f1, 7f74b8, e23f06, 0a48ea, \
b3d65a, 6b92f1, 87f087, c02081, f8d0f4, e3c411, 382201, e8840b, \
fdfdfd
custom colors: OFF, tridx: 0000, colors: 000000, 000000, 000000, \
000000
```

There SHOULD also be two Blocks containing one image each with the timestamps "00:00:01:101" and "00:00:08:708".

5.2. SRT Subtitles

SRT is perhaps the most basic of all subtitle formats.

It consists of four parts, all in text:

1. A number indicating which subtitle it is in the sequence. 2. The time that the subtitle appears on the screen, and then disappears. 3. The subtitle itself. 4. A blank line indicating the start of a new subtitle.

When placing SRT in Matroska, part 3 is converted to UTF-8 (S_TEXT/UTF8) and placed in the data portion of the Block. Part 2 is used to set the timestamp of the Block, and BlockDuration element. Nothing else is used.

Here is an example SRT file:

```
1
00:02:17,440 --> 00:02:20,375
Senator, we’re making
our final approach into Coruscant.

2
00:02:20,476 --> 00:02:22,501
Very good, Lieutenant.
```

In this example, the text "Senator, we’re making our final approach into Coruscant." would be converted into UTF-8 and placed in the Block. The timestamp of the block would be set to "00:02:17,440". And the BlockDuration element would be set to "00:00:02,935".

The same is repeated for the next subtitle.

Because there are no general settings for SRT, the CodecPrivate is left blank.

5.3. SSA/ASS Subtitles

SSA stands for Sub Station Alpha. It’s the file format used by the popular subtitle editor, SubStation Alpha (http://wiki.multimedia.cx/index.php?title=SubStation_Alpha). This format is widely used by fansubbers.

It allows you to do some advanced display features, like positioning, karaoke, style managements...

For detailed information on SSA/ASS, see the SSA specs (http://moodub.free.fr/video/ass-specs.doc). It includes an SSA specs description and the advanced features added by ASS format (standing for Advanced SSA). Because SSA and ASS are so similar, they are treated the same here.

Like SRT, this format is text based with a particular syntax.

A file consists of 4 or 5 parts, declared ala INI file (but it's not an INI !)

The first, "[Script Info]" contains some information about the subtitle file, such as it's title, who created it, type of script and a very important one: "PlayResY". Be careful of this value, everything in your script (font size, positioning) is scaled by it. Sub Station Alpha uses your desktops Y resolution to write this value, so if a friend with a large monitor and a high screen resolution gives you an edited script, you can mess everything up by saving the script in SSA with your low-cost monitor.

The second, "[V4 Styles]", is a list of style definitions. A style describe how will look a text on the screen. It defines font, font size, primary/.../outile colour, position, alignment, etc.

For example, this:

```
Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, \
TertiaryColour, BackColour, Bold, Italic, BorderStyle, Outline, \
Shadow, Alignment, MarginL, MarginR, MarginV, AlphaLevel, Encoding
Style: Wolf main,Wolf_Rain,56,15724527,15724527,15724527,4144959,0,\
0,1,1,2,2,5,5,30,0,0
```

The third, "[Events]", is the list of text you want to display at the right timing. You can specify some attribute here. Like the style to use for this event (MUSTbe defined in the list), the position of the text (Left, Right, Vertical Margin), an effect. Name is mostly used by translator to know who said this sentence. Timing is in h:mm:ss.cc (centisec).

```
Format: Marked, Start, End, Style, Name, MarginL, MarginR, MarginV, \
Effect, Text
Dialogue: Marked=0,0:02:40.65,0:02:41.79,Wolf main,Cher,0000,0000,\
0000,,Et les enregistrements de ses ondes delta ?
Dialogue: Marked=0,0:02:42.42,0:02:44.15,Wolf main,autre,0000,0000,\
0000,,Toujours rien.
```

"[Pictures]" or "[Fonts]" part can be found in some SSA file, they contains UUE-encoded pictures/font but those features are only used by Sub Station Alpha -- i.e., no filter (Vobsub/Avery Lee Subtiler filter) use them.

Now, how are they stored in Matroska?

* All text is converted to UTF-8
* All the headers are stored in CodecPrivate (Script Info and the Styles list)
* Start & End field are used to set TimeStamp and the BlockDuration element. the data stored is:
* Events are stored in the Block in this order: ReadOrder, Layer, Style, Name, MarginL, MarginR, MarginV, Effect, Text (Layer comes from ASS specs ... it’s empty for SSA.) "ReadOrder field is needed for the decoder to be able to reorder the streamed samples as they were placed originally in the file."

Here is an example of an SSA file.

```
[Script Info]
; This is a Sub Station Alpha v4 script.
; For Sub Station Alpha info and downloads,
; go to \
; [http://www.eswat.demon.co.uk/](http://www.eswat.demon.co.uk/)
; or email \
; [kotus@eswat.demon.co.uk](mailto:kotus@eswat.demon.co.uk)
Title: Wolf's rain 2
Original Script: Anime-spirit Ishin-francais
Original Translation: Coolman
Original Editing: Spikewolfwood
Original Timing: Lord_alucard
Original Script Checking: Spikewolfwood
ScriptType: v4.00
Collisions: Normal
PlayResY: 1024
PlayDepth: 0
Wav: 0, 128697,D:\Alex\Anime\- Fansub -\- TAFF -\WR_-_02_Wav.wav
Wav: 0, 120692,H:\team truc\WR_-_02.wav
Wav: 0, 116504,E:\sub\wolf's_rain\WOLF'S RAIN 02.wav
LastWav: 3
Timer: 100,0000

[V4 Styles]
Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, \
TertiaryColour, BackColour, Bold, Italic, BorderStyle, Outline, \
Shadow, Alignment, MarginL, MarginR, MarginV, AlphaLevel, Encoding
Style: Default,Arial,20,65535,65535,65535,-2147483640,-1,0,1,3,0,2,\
30,30,30,0,0
Style: Titre_episode,Akbar,140,15724527,65535,65535,986895,-1,0,1,1,\
0,3,30,30,30,0,0
Style: Wolf main,Wolf_Rain,56,15724527,15724527,15724527,4144959,0,\
0,1,1,2,2,5,5,30,0,0

[Events]
Format: Marked, Start, End, Style, Name, MarginL, MarginR, MarginV, \
Effect, Text
Dialogue: Marked=0,0:02:40.65,0:02:41.79,Wolf main,Cher,0000,0000,\
0000,,Et les enregistrements de ses ondes delta ?
Dialogue: Marked=0,0:02:42.42,0:02:44.15,Wolf main,autre,0000,0000,\
0000,,Toujours rien.
```

Here is what would be placed into the CodecPrivate element.

```
[Script Info]
; This is a Sub Station Alpha v4 script.
; For Sub Station Alpha info and downloads,
; go to \
; [http://www.eswat.demon.co.uk/](http://www.eswat.demon.co.uk/)
; or email \
; [kotus@eswat.demon.co.uk](mailto:kotus@eswat.demon.co.uk)
Title: Wolf's rain 2
Original Script: Anime-spirit Ishin-francais
Original Translation: Coolman
Original Editing: Spikewolfwood
Original Timing: Lord_alucard
Original Script Checking: Spikewolfwood
ScriptType: v4.00
Collisions: Normal
PlayResY: 1024
PlayDepth: 0
Wav: 0, 128697,D:\Alex\Anime\- Fansub -\- TAFF -\WR_-_02_Wav.wav
Wav: 0, 120692,H:\team truc\WR_-_02.wav
Wav: 0, 116504,E:\sub\wolf's_rain\WOLF'S RAIN 02.wav
LastWav: 3
Timer: 100,0000

[V4 Styles]
Format: Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, \
TertiaryColour, BackColour, Bold, Italic, BorderStyle, Outline, \
Shadow, Alignment, MarginL, MarginR, MarginV, AlphaLevel, Encoding
Style: Default,Arial,20,65535,65535,65535,-2147483640,-1,0,1,3,0,2,\
30,30,30,0,0
Style: Titre_episode,Akbar,140,15724527,65535,65535,986895,-1,0,1,1,\
0,3,30,30,30,0,0
Style: Wolf main,Wolf_Rain,56,15724527,15724527,15724527,4144959,0,\
0,1,1,2,2,5,5,30,0,0

And here are the two blocks that would be generated.

Block's timestamp: 00:02:40.650 BlockDuration: 00:00:01.140

1,,Wolf main,Cher,0000,0000,0000,,Et les enregistrements de ses \
ondes delta ?

Block's timestamp: 00:02:42.420 BlockDuration: 00:00:01.730

2,,Wolf main,autre,0000,0000,0000,,Toujours rien.
```

5.4. WebVTT

The "Web Video Text Tracks Format" (short: WebVTT) is developed by the World Wide Web Consortium (W3C) (https://www.w3.org/). Its specifications are freely available (https://w3c.github.io/webvtt/).

The guiding principles for the storage of WebVTT in Matroska are:

* Consistency: store data in a similar way to other subtitle codecs
* Simplicity: making decoding and remuxing as easy as possible for existing infrastructures
* Completeness: keeping as much data as possible from the original WebVTT file

5.4.1. Storage of WebVTT in Matroska

5.4.1.1. CodecID: codec identification

The CodecID to use is S_TEXT/WEBVTT.

5.4.1.2. CodecPrivate: storage of global WebVTT blocks

This element contains all global blocks before the first subtitle entry. This starts at the "WEBVTT" file identification marker but excludes the optional byte order mark.

5.4.1.3. Storage of non-global WebVTT blocks

Non-global WebVTT blocks (e.g., "NOTE") before a WebVTT Cue Text are stored in Matroska's BlockAddition element together with the Matroska Block containing the WebVTT Cue Text these blocks precede (see below for the actual format).

5.4.1.4. Storage of Cues in Matroska blocks

Each WebVTT Cue Text is stored directly in the Matroska Block.

A muxer MUST change all WebVTT Cue Timestamps present within the Cue Text to be relative to the Matroska Block's timestamp.

The Cue's start timestamp is used as the Matroska Block's timestamp.

The difference between the Cue's end timestamp and its start timestamp is used as the Matroska Block's duration.

5.4.1.5. BlockAdditions: storing non-global WebVTT blocks, Cue Settings Lists and Cue identifiers

Each Matroska Block may be accompanied by one BlockAdditions element. Its format is as follows:

1. The first line contains the WebVTT Cue Text's optional Cue Settings List followed by one line feed character (U+0x000a). The Cue Settings List may be empty, in which case the line consists of the line feed character only.
2. The second line contains the WebVTT Cue Text's optional Cue Identifier followed by one line feed character (U+0x000a). The line may be empty indicating that there was no Cue Identifier in the source file, in which case the line consists of the line feed character only.
3. The third and all following lines contain all WebVTT Comment Blocks that precede the current WebVTT Cue Block. These may be absent.

If there is no Matroska BlockAddition element stored together with the Matroska Block, then all three components (Cue Settings List, Cue Identifier, Cue Comments) MUST be assumed to be absent.

5.4.2. Examples of transformation

Here's an example how a WebVTT is transformed.

5.4.2.1. Example WebVTT file

Let's take the following example file:

```
WEBVTT with text after the signature

STYLE
::cue {
  background-image: linear-gradient(to bottom, dimgray, lightgray);
  color: papayawhip;
}
/* Style blocks cannot use blank lines nor "dash dash greater \
than" */

NOTE comment blocks can be used between style blocks.

STYLE
::cue(b) {
  color: peachpuff;
}
```

```
REGION
id:bill
width:40%
lines:3
regionanchor:0%,100%
viewportanchor:10%,90%
scroll:up

NOTE
Notes always span a whole block and can cover multiple
lines. Like this one.
An empty line ends the block.

hello
00:00:00.000 --> 00:00:10.000
Example entry 1: Hello <b>world</b>.

NOTE style blocks cannot appear after the first cue.

00:00:25.000 --> 00:00:35.000
Example entry 2: Another entry.
This one has multiple lines.

00:01:03.000 --> 00:01:06.500 position:90% align:right size:35%
Example entry 3: That stuff to the right of the timestamps are cue \
settings.

00:03:10.000 --> 00:03:20.000
Example entry 4: Entries can even include timestamps.
For example:<00:03:15.000>This becomes visible five seconds
after the first part.
```

5.4.2.2. Example of CodecPrivate

The resulting CodecPrivate element will look like this:

```
WEBVTT with text after the signature

STYLE
::cue {
  background-image: linear-gradient(to bottom, dimgray, lightgray);
  color: papayawhip;
}
/* Style blocks cannot use blank lines nor "dash dash greater \
than" */

NOTE comment blocks can be used between style blocks.

STYLE
::cue(b) {
  color: peachpuff;
}

REGION
id:bill
width:40%
lines:3
regionanchor:0%,100%
viewportanchor:10%,90%
scroll:up

NOTE
Notes always span a whole block and can cover multiple
lines. Like this one.
An empty line ends the block.
```

5.4.2.3. Storage of Cue 1

Example Cue 1: timestamp 00:00:00.000, duration 00:00:10.000, Block's content:

```
Example entry 1: Hello <b>world</b>.
```

BlockAddition's content starts with one empty line as there's no Cue Settings List:

```
hello
```

5.4.2.4. Storage of Cue 2

Example Cue 2: timestamp 00:00:25.000, duration 00:00:10.000, Block's content:

Example entry 2: Another entry.
This one has multiple lines.

BlockAddition’s content starts with two empty lines as there’s neither a Cue Settings List nor a Cue Identifier:

NOTE style blocks cannot appear after the first cue.

5.4.2.5. Storage of Cue 3

Example Cue 3: timestamp 00:01:03.000, duration 00:00:03.500, Block’s content:

Example entry 3: That stuff to the right of the timestamps are cue \ settings.

BlockAddition’s content ends with an empty line as there’s no Cue Identifier and there were no WebVTT Comment blocks:

position:90% align:right size:35%

5.4.2.6. Storage of Cue 4

Example Cue 4: timestamp 00:03:10.000, duration 00:00:10.000, Block’s content:

Example entry 4: Entries can even include timestamps. For example:00:00:05.000 (00:00:05.000)This becomes visible five seconds after the first part.

This Block does not need a BlockAddition as the Cue did not contain an Identifier, nor a Settings List, and it wasn’t preceded by Comment blocks.

5.4.3. Storage of WebVTT in Matroska vs. WebM

Note: the storage of WebVTT in Matroska is not the same as the design document for storage of WebVTT in WebM. There are several reasons for this including but not limited to: the WebM document is old (from February 2012) and was based on an earlier draft of WebVTT and ignores several parts that were added to WebVTT later; WebM does still not support subtitles at all (http://www.webmproject.org/docs/container/); the proposal suggests splitting the information across multiple tracks making demuxer’s and remuxer’s life very difficult.

5.5. HDMV presentation graphics subtitles

The specifications for the HDMV presentation graphics subtitle format (short: HDMV PGS) can be found in the document "Blu-ray Disc Read-Only Format; Part 3 Audio Visual Basic Specifications" in section 9.14 "HDMV graphics streams".

5.5.1. Storage of HDMV presentation graphics subtitles

The CodecID to use is S_HDMV/PGS. A CodecPrivate element is not used.

5.5.1.1. Storage of HDMV PGS Segments in Matroska Blocks

Each HDMV PGS Segment (short: Segment) will be stored in a Matroska Block. A Segment is the data structure described in section 9.14.2.1 "Segment coding structure and parameters" of the Blu-ray specifications.

Each Segment contains a presentation timestamp. This timestamp will be used as the timestamp for the Matroska Block.

A Segment is normally shown until a subsequent Segment is encountered. Therefore, the Matroska Block MAY have no Duration. In that case, a player MUST display a Segment within a Matroska Block until the next Segment is encountered.

A muxer MAY use a Duration, e.g., by calculating the distance between two subsequent Segments. If a Matroska Block has a Duration, a player MUST display that Segment only for the duration of the Block’s Duration.

5.6. HDMV text subtitles

The specifications for the HDMV text subtitle format (short: HDMV TextST) can be found in the document "Blu-ray Disc Read-Only Format; Part 3 Audio Visual Basic Specifications" in section 9.15 "HDMV text subtitle streams".

5.6.1. Storage of HDMV text subtitles

The CodecID to use is S_HDMV/TEXTST.

A CodecPrivate Element is required. It MUST contain the stream’s Dialog Style Segment as described in section 9.15.4.2 "Dialog Style Segment" of the Blu-ray specifications.

5.6.1.1. Storage of HDMV TextST Dialog Presentation Segments in Matroska Blocks

Each HDMV Dialog Presentation Segment (short: Segment) will be stored in a Matroska Block. A Segment is the data structure described in section 9.15.4.3 "Dialog presentation segment" of the Blu-ray specifications.

Each Segment contains a start and an end presentation timestamp (short: start PTS & end PTS). The start PTS will be used as the timestamp for the Matroska Block. The Matroska Block MUST have a Duration, and that Duration is the difference between the end PTS and the start PTS.

A player MUST use the Matroska Block's timestamp and Duration instead of the Segment's start and end PTS for determining when and how long to show the Segment.

5.6.1.2. Character set

When TextST subtitles are stored inside Matroska, the only allowed character set is UTF-8.

Each HDMV text subtitle stream in a Blu-ray can use one of a handful of character sets. This information is not stored in the MPEG2 Transport Stream itself but in the accompanying Clip Information file.

Therefore, a muxer MUST parse the accompanying Clip Information file. If the information indicates a character set other than UTF-8, it MUST re-encode all text Dialog Presentation Segments from the indicated character set to UTF-8 prior to storing them in Matroska.

5.7. Digital Video Broadcasting (DVB) subtitles

The specifications for the Digital Video Broadcasting subtitle bitstream format (short: DVB subtitles) can be found in the document "ETSI EN 300 743 - Digital Video Broadcasting (DVB); Subtitling systems". The storage of DVB subtitles in MPEG transport streams is specified in the document "ETSI EN 300 468 - Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

5.7.1. Storage of DVB subtitles

5.7.1.1. CodecID

The CodecID to use is S_DVBSUB.

5.7.1.2. CodecPrivate

The CodecPrivate element is five bytes long and has the following structure:

* 2 bytes: composition page ID (bit string, left bit first)
* 2 bytes: ancillary page ID (bit string, left bit first)
* 1 byte: subtitling type (bit string, left bit first)

The semantics of these bytes are the same as the ones described in section 6.2.41 "Subtitling descriptor" of ETSI EN 300 468.

5.7.1.3. Storage of DVB subtitles in Matroska Blocks

Each Matroska Block consists of one or more DVB Subtitle Segments as described in segment 7.2 "Syntax and semantics of the subtitling segment" of ETSI EN 300 743.

Each Matroska Block SHOULD have a Duration indicating how long the DVB Subtitle Segments in that Block SHOULD be displayed.

5.8. ARIB (ISDB) subtitles

The specifications for the ARIB B-24 subtitle bitstream format (short: ARIB subtitles) and its storage in MPEG transport streams can be found in the documents [ARIB.STD-B24], [ARIB.STD-B10], and [ARIB.TR-B14].

5.8.1. Storage of ARIB subtitles

5.8.1.1. CodecID

The CodecID to use is S_ARIBSUB.

5.8.1.2. CodecPrivate

The CodecPrivate element is three bytes long and has the following structure:

* 1 byte: component tag (bit string, left bit first)
* 2 bytes: data component ID (bit string, left bit first)

The semantics of the component tag are the same as those described in [ARIB.STD-B10], part 2, Annex J. The semantics of the data component ID are the same as those described in [ARIB.TR-B14], fascicle 2, Vol. 3, Section 2, 4.2.8.1.

5.8.1.3. Storage of ARIB subtitles in Matroska Blocks

Each Matroska Block consists of a single synchronized PES data structure as described in chapter 5 "Independent PES transmission protocol" of [ARIB.STD-B24], volume 3, with a Synchronized_PES_data_byte block containing one or more ISDB Caption Data Groups as described in chapter 9 "Transmission of caption and superimpose" of [ARIB.STD-B24], volume 1, part 3. All of the Caption Statement Data Groups in a given Matroska Track MUST use the same language index.

A Data Group is normally shown until a subsequent Group provides instructions to clear it. Therefore, the Matroska Block SHOULD NOT have a Duration. A player SHOULD display a Data Group within a Matroska Block until its internal duration elapses, or until a subsequent Data Group removes it.

6. Block Additional Mapping

Extra data or metadata can be added to each Block using BlockAdditional data. Each BlockAdditional contains a BlockAddID that identifies the kind of data it contains. When the BlockAddID is set to "1" the contents of the BlockAdditional Element are define by the Codec Mappings defines; see Section 4.1.5. When the BlockAddID is set a value greater than "1", then the contents of the BlockAdditional Element are defined by the BlockAdditionalMapping Element, within the associated Track Element, where the BlockAddID Element of BlockAdditional Element equals the BlockAddIDValue of the associated Track’s BlockAdditionalMapping Element. That BlockAdditionalMapping Element identifies a particular Block Additional Mapping by the BlockAddIDType.

The following XML depicts a use of a Block Additional Mapping to associate a timecode value with a Block:

```
<Segment>
  <!--Mandatory elements ommitted for readability-->
  <Tracks>
    <TrackEntry>
      <TrackNumber>1</TrackNumber>
      <TrackUID>568001708</TrackUID>
      <TrackType>1</TrackType>
      <BlockAdditionalMapping>
        <BlockAddIDValue>2</BlockAddIDValue><!--arbitrary value
          used in BlockAddID-->
        <BlockAddIDName>timecode</BlockAddIDName>
        <BlockAddIDType>12</BlockAddIDType>
      </BlockAdditionalMapping>
      <CodecID>V_FFV1</CodecID>
      <Video>
        <PixelWidth>1920</PixelWidth>
        <PixelHeight>1080</PixelHeight>
      </Video>
    </TrackEntry>
  </Tracks>
  <Cluster>
    <Timestamp>3000</Timestamp>
    <BlockGroup>
      <Block>{binary video frame}</Block>
      <BlockAdditions>
        <BlockMore>
          <BlockAddID>2</BlockAddID><!--arbitrary value from
            BlockAdditionalMapping-->
          <BlockAdditional>01:00:00:00</BlockAdditional>
        </BlockMore>
      </BlockAdditions>
    </BlockGroup>
  </Cluster>
</Segment>
```

Block Additional Mappings detail how additional data MAY be stored in the BlockMore Element with a BlockAdditionMapping Element, within the Track Element, which identifies the BlockAdditional content. Block Additional Mappings define the BlockAddIDType value reserved to identify that type of data as well as providing an optional label stored within the BlockAddIDName Element. When the Block Additional Mapping is dependent on additional contextual information, then the Mapping SHOULD describe how such additional contextual information is stored within the BlockAddIDExtraData Element.

The following Block Additional Mappings are defined.

6.1. Summary of Assigned BlockAddIDType Values

For convenience, the following table shows the assigned BlockAddIDType values along with the BlockAddIDName and Citation.

BlockAddIDType	BlockAddIDName	Citation
121	SMPTE ST 12-1 timecode	Section 6.2

Table 4

6.2. SMPTE ST 12-1 Timecode

6.2.1. Timecode Description

SMPTE ST 12-1 timecode values can be stored in the BlockMore Element to associate the content of a Matroska Block with a particular timecode value. If the Block uses Lacing, the timecode value is associated with the first frame of the Lace.

The Block Additional Mapping contains a full binary representation of a 64 bit SMPTE timecode value stored in big-endian format and expressed exactly as defined in Section 8 and 9 of SMPTE 12M [ST12]. For convenience, here are the bit assignments for a SMPTE ST 12-1 binary representation as described in Section 6.2 of [RFC5484]:

Bit Positions	Label
0--3	Units of frames
4--7	First binary group
8--9	Tens of frames
10	Drop frame flag
11	Color frame flag
12--15	Second binary group
16--19	Units of seconds
20--23	Third binary group
24--26	Tens of seconds

27	Polarity correction
28--31	Fourth binary group
32--35	Units of minutes
36--39	Fifth binary group
40--42	Tens of minutes
43	Binary group flag BGF0
44--47	Sixth binary group
48--51	Units of hours
52--55	Seventh binary group
56--57	Tens of hours
58	Binary group flag BGF1
59	Binary group flag BGF2
60--63	Eighth binary group

Table 5

For example, a timecode value of "07:32:54;18" can be expressed as a 64 bit SMPTE 12M value as:

10000000 01100000 01100000 01010000
00100000 00110000 01110000 00000000

6.2.2. BlockAddIDType

The BlockAddIDType value reserved for timecode is "121".

6.2.3. BlockAddIDName

The BlockAddIDName value reserved for timecode is "SMPTE ST 12-1 timecode".

6.2.4. BlockAddIDExtraData

BlockAddIDExtraData is unused within this block additional mapping.

7. Security Considerations

This document inherits security considerations from the EBML [RFC8794] and Matroska [Matroska] documents.

8. IANA Considerations

To be determined.

9. References

9.1. Normative References

[ARIB.STD-B10]
ARIB, "Service Information for Digital Broadcasting System", 5 December 2019, <https://www.arib.or.jp/english/std_tr/broadcasting/desc/std-b10.html>.

[ARIB.STD-B24]
ARIB, "Data Coding and Transmission Specification for Digital Broadcasting", 6 October 2022, <https://www.arib.or.jp/english/std_tr/broadcasting/desc/std-b24.html>.

[ARIB.TR-B14]
ARIB, "Operational Guidelines for Digital Terrestrial Television Broadcasting", 6 October 2022, <https://www.arib.or.jp/english/std_tr/broadcasting/desc/tr-b14.html>.

[DolbyVisionWithinIso]
Dolby, "Dolby Vision Streams Within the ISO Base MediaFile Format", 7 February 2020, <https://www.dolby.com/us/en/technologies/dolby-vision/dolby-vision-bitstreams-within-the-iso-base-media-file-format-v2.1.2.pdf>.

[IEEE.1857-10]
IEEE, "IEEE Standard for Third Generation Video Coding", 9 November 2021, <https://standards.ieee.org/standard/1857_10-2021.html>.

[IEEE.1857-4]
IEEE, "IEEE Standard for Second-Generation IEEE 1857 Video Coding", 23 October 2018, <https://standards.ieee.org/standard/1857_4-2018.html>.

[IEEE.754] IEEE, "IEEE Standard for Binary Floating-Point Arithmetic", 13 June 2019, <https://standards.ieee.org/standard/754-2019.html>.

[ISO.14496-15]
International Organization for Standardization, "Information technology Coding of audio-visual objects Part 15: Carriage of network abstraction layer (NAL) unit structured video in ISO base media file format", ISO Standard 14496, 2014.

[Matroska] Lhomme, S., Bunkus, M., and D. Rice, "Media Container Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-matroska-10, 1 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-matroska-10>.

[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>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC6386] Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., and Y. Xu, "VP8 Data Format and Decoding Guide", RFC 6386, DOI 10.17487/RFC6386, November 2011, <https://www.rfc-editor.org/info/rfc6386>.

[RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, DOI 10.17487/RFC6648, June 2012, <https://www.rfc-editor.org/info/rfc6648>.

[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>.

[RFC8794] Lhomme, S., Rice, D., and M. Bunkus, "Extensible Binary Meta Language", RFC 8794, DOI 10.17487/RFC8794, July 2020, <https://www.rfc-editor.org/info/rfc8794>.

[ST12] SMPTE, "Time and Control Code", ST ST 12-1:2014, DOI 10.5594/SMPTE.ST12-1.2014, 20 February 2014, <http://ieeexplore.ieee.org/document/7291029/>.

[WAVEFORMATEX] Microsoft Corporation, "WAVEFORMATEX structure", 4 April 2021, <https://docs.microsoft.com/en-us/windows/win32/api/mmeapi/ns-mmeapi-waveformatex>.

9.2. Informative References

[AtracAES] Sony Corporate Research Laboratories, "ATRAC: Adaptive Transform Acoustic Coding for MiniDisc", 1 October 1992, <https://www.minidisc.wiki/technology/atrac/aes>.

[atracdenc] Cherednik, D., "atracdenc - ATRAC1 and ATRAC3 Decoder/ Encoder", 12 October 2022, <https://github.com/dcherednik/atracdenc>.

[DVD-Video] DVD Forum, "DVD-Books: Part 3 DVD-Video Book", November 1995, <http://www.dvdforum.org/>.

[RFC5484] Singer, D., "Associating Time-Codes with RTP Streams", RFC 5484, DOI 10.17487/RFC5484, March 2009, <https://www.rfc-editor.org/info/rfc5484>.

Authors' Addresses

Steve Lhomme
Email: slhomme@matroska.org

Moritz Bunkus
Email: moritz@bunkus.org

Dave Rice
Email: dave@dericed.com

CELLAR Group
Internet-Draft
Intended status: Standards Track
Expires: 30 July 2024

S. Lhomme

M. Bunkus

D. Rice

27 January 2024

Matroska Media Container Control Track Specifications
draft-ietf-cellar-control-04

Abstract

This document defines the Control Track usage found in the Matroska container.

Table of Contents

1. Introduction

2. Status of this document

This document is a work-in-progress specification defining the Matroska file format as part of the IETF Cellar working group (https://datatracker.ietf.org/wg/cellar/charter/). It uses basic elements and concept already defined in the Matroska specifications defined by this workgroup [Matroska].

3. Security Considerations

This document inherits security considerations from the EBML [RFC8794] and Matroska [Matroska] documents.

4. IANA Considerations

To be determined.

5. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

6. Edition Flags

6.1. EditionFlagHidden

When the EditionFlagHidden flag is set to false it means the Edition is visible and selectable in a Matroska Player. All ChapterAtoms Elements MUST be interpreted with their own ChapterFlagHidden flags.

ChapterFlagHidden	False	True	visible
Chapter 1	X		yes
Chapter 2		X	no

Table 1: ChapterAtom visibility to the user

When the EditionFlagHidden flag is set to true the Edition is hidden and SHOULD NOT be selectable in a Matroska Player. If all Editions EditionFlagHidden flags are set to true, there is no visible Edition. In this case all ChapterAtoms Elements MUST also be interpreted as if their ChapterFlagHidden flag is also set to true, regardless with their own ChapterFlagHidden flags.

ChapterFlagHidden	False	True	visible
Chapter 1	X		no
Chapter 2		X	no

Table 2: ChapterAtom visibility in hidden editions

6.2. EditionFlagDefault

It is RECOMMENDED that no more than one Edition have an EditionFlagDefault Flag set to true. The first Edition with both the EditionFlagDefault Flag set to true and the EditionFlagHidden Flag set to false is the Default Edition. When all EditionFlagDefault Flags are set to false, then the first Edition with the EditionFlagHidden Flag set to false is the Default Edition. The Default Edition is the edition that should be used for playback by default.

6.3. Default Edition

The Default Edition is the Edition that a Matroska Player SHOULD use for playback by default.

The first Edition with both the EditionFlagDefault flag set to true and the EditionFlagHidden flag set to false is the Default Edition. When all EditionFlagDefault flags are set to false and all EditionFlagHidden flag set to true, then the first Edition is the Default Edition. When all EditionFlagHidden flags are set to true, then the first Edition with the EditionFlagDefault flag set to true is the Default Edition. When all EditionFlagDefault flags are set to false, then the first Edition with the EditionFlagHidden flag set to false is the Default Edition. When there is no Edition with a EditionFlagDefault flag are set to true and a EditionFlagHidden flags are set to false, then the first Edition with the EditionFlagHidden flag set to false is the Default Edition.

In other words, in case the Default Edition is not obvious, the first Edition with a EditionFlagHidden flag set to false SHOULD be preferred.

Edition	FlagHidden	FlagDefault	Default Edition
Edition 1	true	true	
Edition 2	true	true	
Edition 3	false	true	X

Table 3: Default edition, some visible, all default

Edition	FlagHidden	FlagDefault	Default Edition
Edition 1	true	false	X
Edition 2	true	false	
Edition 3	true	false	

Table 4: Default edition, all hidden, no default

Edition	FlagHidden	FlagDefault	Default Edition
Edition 1	true	false	
Edition 2	true	true	X
Edition 3	true	false	

Table 5: Default edition, all hidden, with default

Edition	FlagHidden	FlagDefault	Default Edition
Edition 1	true	false	
Edition 2	false	false	X
Edition 3	false	false	

Table 6: Default edition, some visible, no default

Edition	FlagHidden	FlagDefault	Default Edition
Edition 1	true	false	
Edition 2	true	true	
Edition 3	false	false	X

Table 7: Default edition, some visible, some default

7. Chapter Flags

If a Control Track toggles the parent's ChapterFlagHidden flag to false, then only the parent ChapterAtom and its second child ChapterAtom MUST be interpreted as if ChapterFlagHidden is set to false. The first child ChapterAtom, which has the ChapterFlagHidden flag set to true, retains its value until its value is toggled to false by a Control Track.

The ChapterFlagEnabled value can be toggled by control tracks.

7.1. ChapterFlagEnabled

If the ChapterFlagEnabled flag is set to false a Matroska Player MUST NOT use this Chapter and all his Nested Chapters. For Simple Chapters, a Matroska Player MAY display this enabled Chapter with a marker in the timeline. For Ordered Chapters a Matroska Player MUST use the duration of this enabled Chapter.

Chapter + Nested Chapter	ChapterFlagEnabled	used
Chapter 1	true	yes
+Nested Chapter 1.1	true	yes
+Nested Chapter 1.2	false	no
++Nested Chapter 1.2.1	true	no
++Nested Chapter 1.2.2	false	no
Chapter 2	false	no
+Nested Chapter 2.1	true	no
+Nested Chapter 2.2	true	no

Table 8

8. Matroska Schema

Extra elements used to handle Control Tracks and advanced selection features:

8.1. Segment

8.1.1. Chapters

8.1.1.1. EditionEntry

8.1.1.1.1. EditionFlagHidden Element

id / type / default: 0x45BD / uinteger / 0
range: 0-1
path: \Segment\Chapters\EditionEntry\EditionFlagHidden
minOccurs / maxOccurs: 1 / 1
definition: Set to 1 if an edition is hidden. Hidden editions SHOULD NOT be available to the user interface (but still to Control Tracks; see Section 7 on Chapter flags).

8.1.1.1.1.1. ChapterFlagEnabled Element

id / type / default: 0x4598 / uinteger / 1
range: 0-1
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterFlagEnabled
minOccurs / maxOccurs: 1 / 1
definition: Set to 1 if the chapter is enabled. It can be enabled/disabled by a Control Track. When disabled, the movie SHOULD skip all the content between the TimeStart and TimeEnd of this chapter; see Section 7 on Chapter flags.

8.1.1.1.1.2. ChapterTrack Element

id / type: 0x8F / master
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterTrack
maxOccurs: 1
definition: List of tracks on which the chapter applies. If this Element is not present, all tracks apply

8.1.1.1.1.3. ChapterTrackUID Element

id / type: 0x89 / uinteger
range: not 0
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterTrack\ChapterTrackUID
minOccurs: 1
definition: UID of the Track to apply this chapter to. In the absence of a control track, choosing this chapter will select the listed Tracks and deselect unlisted tracks. Absence of this Element indicates that the Chapter SHOULD be applied to any currently used Tracks.

9. Menu Specifications

This document is a _draft of the Menu system_ that will be the default one in Matroska. As it will just be composed of a Control Track, it will be seen as a "codec" and could be replaced later by something else if needed.

A menu is like what you see on DVDs [DVD-Video], when you have some screens to select the audio format, subtitles or scene selection.

9.1. Requirements

What we'll try to have is a system that can do almost everything done on a DVD, or more, or better, or drop the unused features if necessary.

As the name suggests, a Control Track is a track that can control the playback of the file and/or all the playback features. To make it as simple as possible for Matroska Players, the Control Track will just give orders to the Matroska Player and get the actions associated with the highlights/hotspots.

9.1.1. Highlights/Hotspots

A highlight is basically a rectangle/key associated with an action UID. When that rectangle/key is activated, the Matroska Player send the UID of the action to the Control Track handler (codec). The fact that it can also be a key means that even for audio only files, a keyboard shortcut or button panel could be used for menus. But in that case, the hotspot will have to be associated with a name to display.

This highlight is sent from the Control Track to the Matroska Player. Then the Matroska Player has to handle that highlight until it's deactivated; see Section 9.1.2.

The highlight contains a UID of the action, a displayable name (UTF-8), an associated key (list of keys to be defined, probably up/down/left/right/select), a screen position/range and an image to display. The image will be displayed either when the user place the mouse over the rectangle (or any other shape), or when an option of the screen is selected (not activated). There could be a second image used when the option is activated. And there could be a third image that can serve as background. This way you could have a still image (like in some DVDs [DVD-Video]) for the menu and behind that image blank video (small bitrate).

When a highlight is activated by the user, the Matroska Player has to send the UID of the action to the Control Track. Then the Control Track codec will handle the action and possibly give new orders to the Matroska Player.

The format used for storing images SHOULD be extensible. For the moment we'll use PNG and BMP, both with alpha channel.

9.1.2. Playback features

All the following features will be sent from the Control Track to the Matroska Player :

* Jump to chapter (UID, prev, next, number)
* Disable all tracks of a kind (audio, video, subtitle)
* Enable track UID (the kind doesn't matter)
* Define/Disable a highlight
* Enable/Disable jumping
* Enable/Disable track selection of a kind
* Select Edition ID (see chapters)
* Pause playback
* Stop playback
* Enable/Disable a Chapter UID
* Hide/Unhide a Chapter UID

All the actions will be written in a normal Matroska track, with a timestamp. A "Menu Frame" SHOULD be able to contain more that one action/highlight for a given timestamp. (to be determined, EBML format structure)

9.1.3. Player requirements

Some Matroska Players might not support the control track. That mean they will play the active/looped parts as part of the data. So I suggest putting the active/looped parts of a movie at the end of a movie. When a Menu-aware Matroska Player encounter the default Control Track of a Matroska file, the first order SHOULD be to jump at the start of the active/looped part of the movie.

9.2. Working Graph

```
Matroska Source file -> Control Track <-> Player.
                     -> other tracks   -> rendered
```

10. Normative References

[Matroska] Lhomme, S., Bunkus, M., and D. Rice, "Media Container Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-matroska-10, 1 May 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-matroska-10>.

[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>.

[RFC8794] Lhomme, S., Rice, D., and M. Bunkus, "Extensible Binary Meta Language", RFC 8794, DOI 10.17487/RFC8794, July 2020, <https://www.rfc-editor.org/info/rfc8794>.

11. Informative References

[DVD-Video] DVD Forum, "DVD-Books: Part 3 DVD-Video Book", 1 November 1995, <http://www.dvdforum.org/>.

Authors' Addresses

Steve Lhomme
Email: slhomme@matroska.org

Moritz Bunkus
Email: moritz@bunkus.org

Dave Rice
Email: dave@dericed.com

cellar
Internet-Draft
Intended status: Standards Track
Expires: 20 July 2024

M. Niedermayer

D. Rice

J. Martinez

17 January 2024

FFV1 Video Coding Format Version 4
draft-ietf-cellar-ffv1-v4-22

Abstract

This document defines FFV1, a lossless, intra-frame video encoding format. FFV1 is designed to efficiently compress video data in a variety of pixel formats. Compared to uncompressed video, FFV1 offers storage compression, frame fixity, and self-description, which makes FFV1 useful as a preservation or intermediate video format.

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 20 July 2024.

Copyright Notice

Table of Contents

1. Introduction

This document describes FFV1, a lossless video encoding format. The design of FFV1 considers the storage of image characteristics, data fixity, and the optimized use of encoding time and storage requirements. FFV1 is designed to support a wide range of lossless video applications such as long-term audiovisual preservation, scientific imaging, screen recording, and other video encoding scenarios that seek to avoid the generational loss of lossy video encodings.

This document defines a version 4 of FFV1. Prior versions of FFV1 are defined within [I-D.ietf-cellar-ffv1].

This document assumes familiarity with mathematical and coding concepts such as Range encoding [Range-Encoding] and YCbCr color spaces [YCbCr].

This specification describes the valid bitstream and how to decode it. Nonconformant bitstreams and the nonconformant handling of bitstreams are outside this specification. A decoder can perform any action that it deems appropriate for an invalid bitstream: reject the bitstream, attempt to perform error concealment, or re-download or use a redundant copy of the invalid part.

2. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Definitions

FFV1: The chosen name of this video encoding format, which is the short version of "FF Video 1". The letters "FF" come from "FFmpeg", which is the name of the reference decoder whose first letters originally meant "Fast Forward".

Container: A format that encapsulates Frames (see Section 4.4) and (when required) a Configuration Record into a bitstream.

Sample: The smallest addressable representation of a color component or a luma component in a Frame. Examples of Sample are Luma (Y), Blue-difference Chroma (Cb), Red-difference Chroma (Cr), Transparency, Red, Green, and Blue.

Symbol: A value stored in the bitstream, which is defined and decoded through one of the methods described in Table 4.

Line: A discrete component of a static image composed of Samples that represent a specific quantification of Samples of that image.

Plane: A discrete component of a static image composed of Lines that represent a specific quantification of Lines of that image.

Pixel: The smallest addressable representation of a color in a Frame. It is composed of one or more Samples.

MSB: Most Significant Bit, the bit that can cause the largest change in magnitude of the symbol.

VLC: Variable Length Code, a code that maps source symbols to a variable number of bits.

RGB: A reference to the method of storing the value of a pixel by using three numeric values that represent Red, Green, and Blue.

YCbCr: A reference to the method of storing the value of a pixel by using three numeric values that represent the luma of the pixel (Y) and the chroma of the pixel (Cb and Cr). The term YCbCr is used for historical reasons and currently references any color space relying on one luma Sample and two chroma Samples, e.g., YCbCr (luma, blue-difference chroma, red-difference chroma), YCgCo, or ICtCp (intensity, blue-yellow, red-green).

TBA: To Be Announced. Used in reference to the development of future iterations of the FFV1 specification.

2.2. Conventions

2.2.1. Pseudocode

The FFV1 bitstream is described in this document using pseudocode. Note that the pseudocode is used to illustrate the structure of FFV1 and is not intended to specify any particular implementation. The pseudocode used is based upon the C programming language [ISO.9899.2018] and uses its if/else, while, and for keywords as well as functions defined within this document.

In some instances, pseudocode is presented in a two-column format such as shown in Figure 1. In this form, the type column provides a symbol as defined in Table 4 that defines the storage of the data referenced in that same line of pseudocode.

```
pseudocode                                                      | type
----------------------------------------------------------------|-----
ExamplePseudoCode( ) {                                          |
    value                                                       | ur
}                                                               |
```

Figure 1: A depiction of type-labeled pseudocode used within this document.

2.2.2. Arithmetic Operators

Note: the operators and the order of precedence are the same as used in the C programming language [ISO.9899.2018], with the exception of >> (removal of implementation-defined behavior) and ^ (power instead of XOR) operators, which are redefined within this section.

a + b means a plus b.

a - b means a minus b.

-a means negation of a.

a * b means a multiplied by b.

a / b means a divided by b.

a ^ b means a raised to the b-th power.

a & b means bitwise "and" of a and b.

a | b means bitwise "or" of a and b.

a >> b means arithmetic right shift of the two's complement integer representation of a by b binary digits. This is equivalent to dividing a by 2, b times, with rounding toward negative infinity.

a << b means arithmetic left shift of the two's complement integer representation of a by b binary digits.

2.2.3. Assignment Operators

a = b means a is assigned b.

a++ is equivalent to a is assigned a + 1.

a-- is equivalent to a is assigned a - 1.

a += b is equivalent to a is assigned a + b.

a -= b is equivalent to a is assigned a - b.

a *= b is equivalent to a is assigned a * b.

2.2.4. Comparison Operators

a > b is true when a is greater than b.

a >= b is true when a is greater than or equal to b.

a < b is true when a is less than b.

a <= b is true when a is less than or equal b.

a == b is true when a is equal to b.

a != b is true when a is not equal to b.

a && b is true when both a is true and b is true.

a || b is true when either a is true or b is true.

!a is true when a is not true.

a ? b : c if a is true, then b, otherwise c.

2.2.5. Mathematical Functions

floor(a) means the largest integer less than or equal to a.

ceil(a) means the smallest integer greater than or equal to a.

sign(a) extracts the sign of a number, i.e., if a < 0 then -1, else if a > 0 then 1, else 0.

abs(a) means the absolute value of a, i.e., abs(a) = sign(a) * a.

log2(a) means the base-two logarithm of a.

min(a,b) means the smaller of two values a and b.

max(a,b) means the larger of two values a and b.

median(a,b,c) means the numerical middle value in a data set of a, b, and c, i.e., a+b+c-min(a,b,c)-max(a,b,c).

a ==> b means a implies b.

a <==> b means a ==> b , b ==> a.

a_b means the b-th value of a sequence of a.

a_(b,c) means the 'b,c'-th value of a sequence of a.

2.2.6. Order of Operation Precedence

When order of precedence is not indicated explicitly by use of parentheses, operations are evaluated in the following order (from top to bottom, operations of same precedence being evaluated from left to right). This order of operations is based on the order of operations used in Standard C.

```
a++, a--
!a, -a
a ^ b
a * b, a / b
a + b, a - b
a << b, a >> b
a < b, a <= b, a > b, a >= b
a == b, a != b
a & b
a | b
a && b
a || b
a ? b : c
a = b, a += b, a -= b, a *= b
```

2.2.7. Range

a...b means any value from a to b, inclusive.

2.2.8. NumBytes

NumBytes is a nonnegative integer that expresses the size in 8-bit octets of a particular FFV1 Configuration Record or Frame. FFV1 relies on its container to store the NumBytes values; see Section 4.3.3.

2.2.9. Bitstream Functions

2.2.9.1. remaining_bits_in_bitstream

remaining_bits_in_bitstream(NumBytes) means the count of remaining bits after the pointer in that Configuration Record or Frame. It is computed from the NumBytes value multiplied by 8 minus the count of bits of that Configuration Record or Frame already read by the bitstream parser.

2.2.9.2. remaining_symbols_in_syntax

remaining_symbols_in_syntax( ) is true as long as the range coder has not consumed all the given input bytes.

2.2.9.3. byte_aligned

byte_aligned( ) is true if remaining_bits_in_bitstream(NumBytes) is a multiple of 8, otherwise false.

2.2.9.4. get_bits

get_bits(i) is the action to read the next i bits in the bitstream, from most significant bit to least significant bit, and to return the corresponding value. The pointer is increased by i.

3. Sample Coding

For each Slice (as described in Section 4.5) of a Frame, the Planes, Lines, and Samples are coded in an order determined by the color space (see Section 3.7). Each Sample is predicted by the median predictor as described in Section 3.3 from other Samples within the same Plane, and the difference is stored using the method described in Section 3.8.

3.1. Border

A border is assumed for each coded Slice for the purpose of the median predictor and context according to the following rules:

* One column of Samples to the left of the coded Slice is assumed as identical to the Samples of the leftmost column of the coded Slice shifted down by one row. The value of the topmost Sample of the column of Samples to the left of the coded Slice is assumed to be 0.
* One column of Samples to the right of the coded Slice is assumed as identical to the Samples of the rightmost column of the coded Slice.

* An additional column of Samples to the left of the coded Slice and two rows of Samples above the coded Slice are assumed to be 0.

Figure 2 depicts a Slice of nine Samples a,b,c,d,e,f,g,h,i in a three-by-three arrangement along with its assumed border.

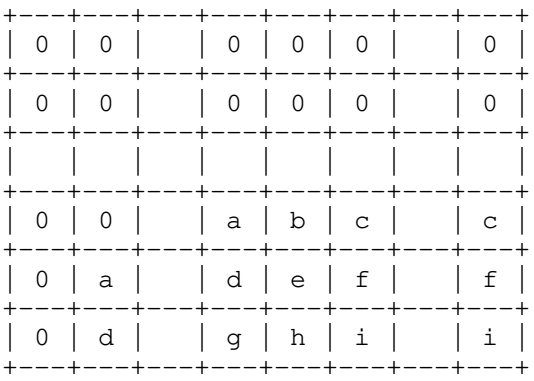


Figure 2: A depiction of FFV1's assumed border for a set of example Samples.

3.2. Samples

Relative to any Sample X, six other relatively positioned Samples from the coded Samples and presumed border are identified according to the labels used in Figure 3. The labels for these relatively positioned Samples are used within the median predictor and context.

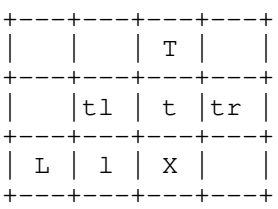


Figure 3: A depiction of how relatively positioned Samples are referenced within this document.

The labels for these relative Samples are made of the first letters of the words Top, Left, and Right.

3.3. Median Predictor

The prediction for any Sample value at position X may be computed based upon the relative neighboring values of l, t, and tl via this equation:

median(l, t, l + t - tl)

Note that this prediction template is also used in [ISO.14495-1.1999] and [HuffYUV].

3.3.1. Exception

If colorspace_type == 0 && bits_per_raw_sample == 16 && (coder_type == 1 || coder_type == 2) (see Section 4.2.5, Section 4.2.7, and Section 4.2.3), the following median predictor MUST be used:

median(left16s, top16s, left16s + top16s - diag16s)

where:

```
left16s = l  >= 32768 ? ( l  - 65536 ) : l
top16s  = t  >= 32768 ? ( t  - 65536 ) : t
diag16s = tl >= 32768 ? ( tl - 65536 ) : tl
```

Background: a two's complement 16-bit signed integer was used for storing Sample values in all known implementations of FFV1 bitstream (see Appendix C). So in some circumstances, the most significant bit was wrongly interpreted (used as a sign bit instead of the 16th bit of an unsigned integer). Note that when the issue was discovered, the only impacted configuration of all known implementations was the 16-bit YCbCr with no pixel transformation and with the range coder coder type, as the other potentially impacted configurations (e.g., the 15/16-bit JPEG 2000 Reversible Color Transform (RCT) [ISO.15444-1.2019] with range coder or the 16-bit content with the Golomb Rice coder type) were not implemented. Meanwhile, the 16-bit JPEG 2000 RCT with range coder was deployed without this issue in one implementation and validated by one conformance checker. It is expected (to be confirmed) that this exception for the median predictor will be removed in the next version of the FFV1 bitstream.

3.4. Quantization Table Sets

Quantization Tables are used on Sample Differences (see Section 3.8), so Quantized Sample Differences are stored in the bitstream.

The FFV1 bitstream contains one or more Quantization Table Sets. Each Quantization Table Set contains exactly five Quantization Tables with each Quantization Table corresponding to one of the five Quantized Sample Differences. For each Quantization Table, both the number of quantization steps and their distribution are stored in the FFV1 bitstream; each Quantization Table has exactly 256 entries, and the eight least significant bits of the Quantized Sample Difference are used as an index:

```
Q_(j)[k] = quant_tables[i][j][k&255]
```

Figure 4: Description of the mapping from sample differences to the corresponding Quantized Sample Differences.

In this formula, i is the Quantization Table Set index, j is the Quantized Table index, and k is the Quantized Sample Difference (see Section 4.1.1).

3.5. Context

Relative to any Sample X, the Quantized Sample Differences L-l, l-tl, tl-t, T-t, and t-tr are used as context:

```
context = Q_(0)[l - tl] +
          Q_(1)[tl - t] +
          Q_(2)[t - tr] +
          Q_(3)[L - l]  +
          Q_(4)[T - t]
```

Figure 5: Description of the computing of the Context.

If context >= 0 then context is used, and the difference between the Sample and its predicted value is encoded as is; else -context is used, and the difference between the Sample and its predicted value is encoded with a flipped sign.

3.6. Quantization Table Set Indexes

For each Plane of each Slice, a Quantization Table Set is selected from an index:

* For Y Plane, quant_table_set_index[0] index is used.
* For Cb and Cr Planes, quant_table_set_index[1] index is used.
* For extra Plane, quant_table_set_index[(version <= 3 || chroma_planes) ? 2 : 1] index is used.

Background: in the first implementations of the FFV1 bitstream, the index for Cb and Cr Planes was stored even if it was not used (chroma_planes set to 0), this index is kept for version <= 3 in order to keep compatibility with FFV1 bitstreams in the wild.

3.7. Color Spaces

FFV1 supports several color spaces. The count of allowed coded Planes and the meaning of the extra Plane are determined by the selected color space.

The FFV1 bitstream interleaves data in an order determined by the color space. In YCbCr for each Plane, each Line is coded from top to bottom, and for each Line, each Sample is coded from left to right. In JPEG 2000 RCT for each Line from top to bottom, each Plane is coded, and for each Plane, each Sample is encoded from left to right.

3.7.1. YCbCr

This color space allows one to four Planes.

The Cb and Cr Planes are optional, but if they are used, then they MUST be used together. Omitting the Cb and Cr Planes codes the frames in gray scale without color data.

An optional transparency Plane can be used to code transparency data.

An FFV1 Frame using YCbCr MUST use one of the following arrangements:

* Y
* Y, Transparency
* Y, Cb, Cr
* Y, Cb, Cr, Transparency

The Y Plane MUST be coded first. If the Cb and Cr Planes are used, then they MUST be coded after the Y Plane. If a transparency Plane is used, then it MUST be coded last.

3.7.2. RGB

This color space allows three or four Planes.

An optional transparency Plane can be used to code transparency data.

JPEG 2000 RCT is a Reversible Color Transform that codes RGB (Red, Green, Blue) Planes losslessly in a modified YCbCr color space [ISO.15444-1.2019]. Reversible pixel transformations between YCbCr and RGB use the following formulae:

```
Cb = b - g
Cr = r - g
Y = g + (Cb + Cr) >> 2
```

Figure 6: Description of the transformation of pixels from RGB color space to coded, modified YCbCr color space.

```
g = Y - (Cb + Cr) >> 2
r = Cr + g
b = Cb + g
```

Figure 7: Description of the transformation of pixels from coded, modified YCbCr color space to RGB color space.

Cb and Cr are positively offset by 1 << bits_per_raw_sample after the conversion from RGB to the modified YCbCr, and they are negatively offset by the same value before the conversion from the modified YCbCr to RGB in order to have only nonnegative values after the conversion.

When FFV1 uses the JPEG 2000 RCT, the horizontal Lines are interleaved to improve caching efficiency since it is most likely that the JPEG 2000 RCT will immediately be converted to RGB during decoding. The interleaved coding order is also Y, then Cb, then Cr, and then, if used, transparency.

As an example, a Frame that is two pixels wide and two pixels high could comprise the following structure:

```
+-----------------------+-----------------------+
| Pixel(1,1)            | Pixel(2,1)            |
| Y(1,1) Cb(1,1) Cr(1,1) | Y(2,1) Cb(2,1) Cr(2,1) |
+-----------------------+-----------------------+
| Pixel(1,2)            | Pixel(2,2)            |
| Y(1,2) Cb(1,2) Cr(1,2) | Y(2,2) Cb(2,2) Cr(2,2) |
+-----------------------+-----------------------+
```

In JPEG 2000 RCT, the coding order is left to right and then top to bottom, with values interleaved by Lines and stored in this order:

```
Y(1,1) Y(2,1) Cb(1,1) Cb(2,1) Cr(1,1) Cr(2,1) Y(1,2) Y(2,2) Cb(1,2)
Cb(2,2) Cr(1,2) Cr(2,2)
```

3.7.2.1. RGB Exception

If bits_per_raw_sample is between 9 and 15 inclusive and extra_plane is 0, the following formulae for reversible conversions between YCbCr and RGB MUST be used instead of the ones above:

```
Cb = g - b
Cr = r - b
Y = b + (Cb + Cr) >> 2
```

Figure 8: Description of the transformation of pixels from RGB color space to coded, modified YCbCr color space (in case of exception).

```
b = Y - (Cb + Cr) >> 2
r = Cr + b
g = Cb + b
```

Figure 9: Description of the transformation of pixels from coded, modified YCbCr color space to RGB color space (in case of exception).

Background: At the time of this writing, in all known implementations of the FFV1 bitstream, when bits_per_raw_sample was between 9 and 15 inclusive and extra_plane was 0, Green Blue Red (GBR) Planes were used as Blue Green Red (BGR) Planes during both encoding and decoding. Meanwhile, 16-bit JPEG 2000 RCT was implemented without this issue in one implementation and validated by one conformance checker. Methods to address this exception for the transform are under consideration for the next version of the FFV1 bitstream.

3.8. Coding of the Sample Difference

Instead of coding the n+1 bits of the Sample Difference with Huffman or Range coding (or n+2 bits, in the case of JPEG 2000 RCT), only the n (or n+1, in the case of JPEG 2000 RCT) least significant bits are used, since this is sufficient to recover the original Sample. In Figure 10, the term bits represents bits_per_raw_sample + 1 for JPEG 2000 RCT or bits_per_raw_sample otherwise:

```
coder_input = ((sample_difference + 2 ^ (bits - 1)) &
              (2 ^ bits - 1)) - 2 ^ (bits - 1)
```

Figure 10: Description of the coding of the Sample Difference in the bitstream.

3.8.1. Range Coding Mode

Early experimental versions of FFV1 used the Context-Adaptive Binary Arithmetic Coding (CABAC) coder from H.264 as defined in [ISO.14496-10.2020], but due to the uncertain patent/royalty situation, as well as its slightly worse performance, CABAC was replaced by a range coder based on an algorithm defined by G. Nigel N. Martin in 1979 [Range-Encoding].

3.8.1.1. Range Binary Values

To encode binary digits efficiently, a range coder is used. A range coder encodes a series of binary symbols by using a probability estimation within each context. The sizes of each of the two subranges are proportional to their estimated probability. The Quantization Table is used to choose the context used from the surrounding image sample values for the case of coding the Sample Differences. The coding of integers is done by coding multiple binary values. The range decoder will read bytes until it can determine into which subrange the input falls to return the next binary symbol.

To describe Range coding for FFV1, the following values are used:

C_i the i-th context.
B_i the i-th byte of the bytestream.
R_i the Range at the i-th symbol.
r_i the boundary between two subranges of R_i: a subrange of r_i values and a subrange R_i - r_i values.
L_i the Low value of the Range at the i-th symbol.
l_i a temporary variable to carry over or adjust the Low value of the Range between range coding operations.
t_i a temporary variable to transmit subranges between range coding operations.
b_i the i-th range-coded binary value.
S_(0, i) the i-th initial state.
j_n the length of the bytestream encoding n binary symbols.

The following range coder state variables are initialized to the following values. The Range is initialized to a value of 65,280 (expressed in base 16 as 0xFF00) as depicted in Figure 11. The Low is initialized according to the value of the first two bytes as depicted in Figure 12. j_i tracks the length of the bytestream encoding while incrementing from an initial value of j_0 to a final value of j_n. j_0 is initialized to 2 as depicted in Figure 13.

```
R_(0) = 65280
```

Figure 11: The initial value for the Range.

```
L_(0) = 2 ^ 8 * B_(0) + B_(1)
```

Figure 12: The initial value for Low is set according to the first two bytes of the bytestream.

```
j_(0) = 2
```

Figure 13: The initial value for j, the length of the bytestream encoding.

The following equations define how the range coder variables evolve as it reads or writes symbols.

```
r_(i) = floor( ( R_(i) * S_(i, C_(i)) ) / 2 ^ 8 )
```

Figure 14: This formula shows the positioning of range split based on the state.

```
           b_(i) =  0                              <==>
           L_(i) <  R_(i) - r_(i)                  ==>
S_(i + 1, C_(i)) =  zero_state_(S_(i, C_(i)))      AND
           l_(i) =  L_(i)                          AND
           t_(i) =  R_(i) - r_(i)

           b_(i) =  1                              <==>
           L_(i) >= R_(i) - r_(i)                  ==>
S_(i + 1, C_(i)) =  one_state_(S_(i, C_(i)))       AND
           l_(i) =  L_(i) - R_(i) + r_(i)          AND
           t_(i) =  r_(i)
```

Figure 15: This formula shows the linking of the decoded symbol (represented as b_i), the updated state (represented as S_(i+1,C_(i))), and the updated range (represented as a range from l_i to t_i).

```
C_(i) != k ==> S_(i + 1, k) = S_(i, k)
```

Figure 16: If the value of k is unequal to the i-th value of context, in other words, if the state is unchanged from the last symbol coding, then the value of the state is carried over to the next symbol coding.

```
t_(i)     <  2 ^ 8                              ==>
R_(i + 1) =  2 ^ 8 * t_(i)                      AND
L_(i + 1) =  2 ^ 8 * l_(i) + B_(j_(i))          AND
j_(i + 1) =  j_(i) + 1

t_(i)     >= 2 ^ 8                              ==>
R_(i + 1) =  t_(i)                              AND
L_(i + 1) =  l_(i)                              AND
j_(i + 1) =  j_(i)
```

Figure 17: This formula shows the linking of the range coder with the reading or writing of the bytestream.

```
    range = 0xFF00;
    end   = 0;
    low   = get_bits(16);
    if (low >= range) {
        low = range;
        end = 1;
    }
```

Figure 18: A pseudocode description of the initialization of range coder variables in Range binary mode.

```
refill() {
    if (range < 256) {
        range = range * 256;
        low   = low * 256;
        if (!end) {
            c.low += get_bits(8);
            if (remaining_bits_in_bitstream( NumBytes ) == 0) {
                end = 1;
            }
        }
    }
}
```

Figure 19: A pseudocode description of refilling the binary value buffer of the range coder.

```
get_rac(state) {
    rangeoff  = (range * state) / 256;
    range    -= rangeoff;
    if (low < range) {
        state = zero_state[state];
        refill();
        return 0;
    } else {
        low   -= range;
        state  = one_state[state];
        range  = rangeoff;
        refill();
        return 1;
    }
}
```

Figure 20: A pseudocode description of the read of a binary value in Range binary mode.

3.8.1.1.1. Termination

The range coder can be used in three modes:

* In Open mode when decoding, every symbol the reader attempts to read is available. In this mode, arbitrary data can have been appended without affecting the range coder output. This mode is not used in FFV1.

* In Closed mode, the length in bytes of the bytestream is provided to the range decoder. Bytes beyond the length are read as 0 by the range decoder. This is generally one byte shorter than the Open mode.

* In Sentinel mode, the exact length in bytes is not known, and thus the range decoder MAY read into the data that follows the range-coded bytestream by one byte. In Sentinel mode, the end of the range-coded bytestream is a binary symbol with state 129, which value SHALL be discarded. After reading this symbol, the range decoder will have read one byte beyond the end of the range-coded bytestream. This way the byte position of the end can be determined. Bytestreams written in Sentinel mode can be read in Closed mode if the length can be determined. In this case, the last (sentinel) symbol will be read uncorrupted and be of value 0.

The above describes the range decoding. Encoding is defined as any process that produces a decodable bytestream.

There are three places where range coder termination is needed in FFV1. The first is in the Configuration Record, which in this case the size of the range coded bytestream is known and handled as Closed mode. The second is the switch from the Slice Header, which is range coded to Golomb-coded Slices as Sentinel mode. The third is the end of range-coded Slices, which need to terminate before the CRC at their end. This can be handled as Sentinel mode or as Closed mode if the CRC position has been determined.

3.8.1.2. Range Non Binary Values

To encode scalar integers, it would be possible to encode each bit separately and use the past bits as context. However, that would mean 255 contexts per 8-bit symbol, which is not only a waste of memory but also requires more past data to reach a reasonably good estimate of the probabilities. Alternatively, it would also be possible to assume a Laplacian distribution and only dealing with its variance and mean (as in Huffman coding). However, for maximum flexibility and simplicity, the chosen method uses a single symbol to encode if a number is 0, and if the number is nonzero, it encodes the

number using its exponent, mantissa, and sign. The exact contexts used are best described by Figure 21.

```
int get_symbol(RangeCoder *c, uint8_t *state, int is_signed) {
    if (get_rac(c, state + 0) {
        return 0;
    }

    int e = 0;
    while (get_rac(c, state + 1 + min(e, 9)) { //1..10
        e++;
    }

    int a = 1;
    for (int i = e - 1; i >= 0; i--) {
        a = a * 2 + get_rac(c, state + 22 + min(i, 9));  // 22..31
    }

    if (!is_signed) {
        return a;
    }

    if (get_rac(c, state + 11 + min(e, 10))) { //11..21
        return -a;
    } else {
        return a;
    }
}
```

Figure 21: A pseudocode description of the contexts of Range nonbinary values.

get_symbol is used for the read out of sample_difference indicated in Figure 10.

get_rac returns a boolean, computed from the bytestream as described by the formula found in Figure 14 and by the pseudocode found in Figure 20.

3.8.1.3. Initial Values for the Context Model

When the keyframe value (see Section 4.4) value is 1, all range coder state variables are set to their initial state.

3.8.1.4. State Transition Table

In this model, a state transition table is used, indicating to which state the decoder will move to, based on the current state and the value extracted from Figure 20.

```
one_state_(i) =
       default_state_transition_(i) + state_transition_delta_(i)
```

Figure 22: Description of the coding of the state transition table for a get_rac readout value of 0.

```
zero_state_(i) = 256 - one_state_(256-i)
```

Figure 23: Description of the coding of the state transition table for a get_rac readout value of 1.

3.8.1.5. default_state_transition

By default, the following state transition table is used:

```
  0,  0,  0,  0,  0,  0,  0,  0, 20, 21, 22, 23, 24, 25, 26, 27,
 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 37, 38, 39, 40, 41, 42,
 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 56, 57,
 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
 74, 75, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
 89, 90, 91, 92, 93, 94, 94, 95, 96, 97, 98, 99,100,101,102,103,
104,105,106,107,108,109,110,111,112,113,114,114,115,116,117,118,
119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,133,
134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,
150,151,152,152,153,154,155,156,157,158,159,160,161,162,163,164,
165,166,167,168,169,170,171,171,172,173,174,175,176,177,178,179,
180,181,182,183,184,185,186,187,188,189,190,190,191,192,194,194,
195,196,197,198,199,200,201,202,202,204,205,206,207,208,209,209,
210,211,212,213,215,215,216,217,218,219,220,220,222,223,224,225,
226,227,227,229,229,230,231,232,234,234,235,236,237,238,239,240,
241,242,243,244,245,246,247,248,248,  0,  0,  0,  0,  0,  0,  0,
```

Figure 24: Default state transition table for Range coding.

3.8.1.6. Alternative State Transition Table

The alternative state transition table has been built using iterative minimization of frame sizes and generally performs better than the default. To use it, the coder_type (see Section 4.2.3) MUST be set to 2, and the difference to the default MUST be stored in the Parameters, see Section 4.2. At the time of this writing, the reference implementation of FFV1 in FFmpeg uses Figure 25 by default when Range coding is used.

```
  0, 10, 10, 10, 10, 16, 16, 16, 28, 16, 16, 29, 42, 49, 20, 49,
 59, 25, 26, 26, 27, 31, 33, 33, 33, 34, 34, 37, 67, 38, 39, 39,
 40, 40, 41, 79, 43, 44, 45, 45, 48, 48, 64, 50, 51, 52, 88, 52,
 53, 74, 55, 57, 58, 58, 74, 60,101, 61, 62, 84, 66, 66, 68, 69,
 87, 82, 71, 97, 73, 73, 82, 75,111, 77, 94, 78, 87, 81, 83, 97,
 85, 83, 94, 86, 99, 89, 90, 99,111, 92, 93,134, 95, 98,105, 98,
105,110,102,108,102,118,103,106,106,113,109,112,114,112,116,125,
115,116,117,117,126,119,125,121,121,123,145,124,126,131,127,129,
165,130,132,138,133,135,145,136,137,139,146,141,143,142,144,148,
147,155,151,149,151,150,152,157,153,154,156,168,158,162,161,160,
172,163,169,164,166,184,167,170,177,174,171,173,182,176,180,178,
175,189,179,181,186,183,192,185,200,187,191,188,190,197,193,196,
197,194,195,196,198,202,199,201,210,203,207,204,205,206,208,214,
209,211,221,212,213,215,224,216,217,218,219,220,222,228,223,225,
226,224,227,229,240,230,231,232,233,234,235,236,238,239,237,242,
241,243,242,244,245,246,247,248,249,250,251,252,252,253,254,255,
```

Figure 25: Alternative state transition table for Range coding.

3.8.2. Golomb Rice Mode

The end of the bitstream of the Frame is padded with zeroes until the bitstream contains a multiple of eight bits.

3.8.2.1. Signed Golomb Rice Codes

This coding mode uses Golomb Rice codes. The VLC is split into two parts: the prefix and suffix. The prefix stores the most significant bits or indicates if the symbol is too large to be stored (this is known as the ESC case. The suffix either stores the k least significant bits or stores the whole number in the ESC case.

```
int get_ur_golomb(k) {
    for (prefix = 0; prefix < 12; prefix++) {
        if (get_bits(1)) {
            return get_bits(k) + (prefix << k);
        }
    }
    return get_bits(bits) + 11;
}
```

Figure 26: A pseudocode description of the read of an unsigned integer in Golomb Rice mode.

```
int get_sr_golomb(k) {
    v = get_ur_golomb(k);
    if (v & 1) return - (v >> 1) - 1;
    else       return   (v >> 1);
}
```

Figure 27: A pseudocode description of the read of a signed integer in Golomb Rice mode.

3.8.2.1.1. Prefix

bits	value
1	0
01	1
...	...
0000 0000 01	9
0000 0000 001	10
0000 0000 0001	11
0000 0000 0000	ESC

Table 1: Description of the coding of the prefix of signed Golomb Rice codes.

ESC is an ESCape symbol to indicate that the symbol to be stored is too large for normal storage and that an alternate storage method is used.

3.8.2.1.2. Suffix

non ESC	the k least significant bits MSB first
ESC	the value - 11, in MSB first order

Table 2: Description of the coding of the suffix of signed Golomb Rice codes.

ESC MUST NOT be used if the value can be coded as non-ESC.

3.8.2.1.3. Examples

Table 3 shows practical examples of how signed Golomb Rice codes are decoded based on the series of bits extracted from the bitstream as described by the method above:

k	bits	value
0	1	0
0	001	2
2	1 00	0
2	1 10	2
2	01 01	5
any	000000000000 10000000	139

Table 3: Examples of decoded, signed Golomb Rice codes.

3.8.2.2. Run Mode

Run mode is entered when the context is 0 and left as soon as a nonzero difference is found. The Sample Difference is identical to the predicted one. The run and the first different Sample Difference are coded are coded as defined in Section 3.8.2.4.1.

3.8.2.2.1. Run Length Coding

The run value is encoded in two parts. The prefix part stores the more significant part of the run as well as adjusting the run_index that determines the number of bits in the less significant part of the run. The second part of the value stores the less significant part of the run as it is. The run_index is reset to zero for each Plane and Slice.

```
log2_run[41] = {
 0, 0, 0, 0, 1, 1, 1, 1,
 2, 2, 2, 2, 3, 3, 3, 3,
 4, 4, 5, 5, 6, 6, 7, 7,
 8, 9,10,11,12,13,14,15,
16,17,18,19,20,21,22,23,
24,
};

if (run_count == 0 && run_mode == 1) {
    if (get_bits(1)) {
        run_count = 1 << log2_run[run_index];
        if (x + run_count <= w) {
            run_index++;
        }
    } else {
        if (log2_run[run_index]) {
            run_count = get_bits(log2_run[run_index]);
        } else {
            run_count = 0;
        }
        if (run_index) {
            run_index--;
        }
        run_mode = 2;
    }
}
```

The log2_run array is also used within [ISO.14495-1.1999].

3.8.2.3. Sign Extension

sign_extend is the function of increasing the number of bits of an input binary number in two's complement signed number representation while preserving the input number's sign (positive/negative) and value, in order to fit in the output bit width. It MAY be computed with the following:

```
sign_extend(input_number, input_bits) {
    negative_bias = 1 << (input_bits - 1);
    bits_mask = negative_bias - 1;
    output_number = input_number & bits_mask; // Remove negative bit
    is_negative = input_number & negative_bias; // Test negative bit
    if (is_negative)
        output_number -= negative_bias;
    return output_number
}
```

3.8.2.4. Scalar Mode

Each difference is coded with the per context mean prediction removed and a per context value for k.

```
   get_vlc_symbol(state) {
       i = state->count;
       k = 0;
       while (i < state->error_sum) {
           k++;
           i += i;
       }

       v = get_sr_golomb(k);

       if (2 * state->drift < -state->count) {
           v = -1 - v;
       }

       ret = sign_extend(v + state->bias, bits);

       state->error_sum += abs(v);
       state->drift     += v;

       if (state->count == 128) {
           state->count     >>= 1;
           state->drift     >>= 1;
           state->error_sum >>= 1;
       }
       state->count++;
       if (state->drift <= -state->count) {
           state->bias = max(state->bias - 1, -128);

           state->drift = max(state->drift + state->count,
                              -state->count + 1);
       } else if (state->drift > 0) {
           state->bias = min(state->bias + 1, 127);

           state->drift = min(state->drift - state->count, 0);
       }

       return ret;
   }
```

3.8.2.4.1. Golomb Rice Sample Difference Coding

Level coding is identical to the normal difference coding with the exception that the 0 value is removed as it cannot occur:

```
       diff = get_vlc_symbol(context_state);
       if (diff >= 0) {
           diff++;
       }
```

Note that this is different from JPEG-LS (lossless JPEG), which doesn't use prediction in run mode and uses a different encoding and context model for the last difference. On a small set of test Samples, the use of prediction slightly improved the compression rate.

3.8.2.5. Initial Values for the VLC Context State

When keyframe (see Section 4.4) value is 1, all VLC coder state variables are set to their initial state.

```
drift     = 0;
error_sum = 4;
bias      = 0;
count     = 1;
```

4. Bitstream

An FFV1 bitstream is composed of a series of one or more Frames and (when required) a Configuration Record.

Within the following subsections, pseudocode as described in Section 2.2.1, is used to explain the structure of each FFV1 bitstream component. Table 4 lists symbols used to annotate that pseudocode in order to define the storage of the data referenced in that line of pseudocode.

symbol	definition
u(n)	Unsigned, big-endian integer symbol using n bits
br	Boolean (1-bit) symbol that is range coded with the method described in Section 3.8.1.1
ur	Unsigned scalar symbol that is range coded with the method described in Section 3.8.1.2
sr	Signed scalar symbol that is range coded with the method described in Section 3.8.1.2
sd	Sample difference symbol that is coded with the method described in Section 3.8

Table 4: Definition of pseudocode symbols for this document.

The following MUST be provided by external means during the initialization of the decoder:

frame_pixel_width is defined as Frame width in pixels.

frame_pixel_height is defined as Frame height in pixels.

Default values at the decoder initialization phase:

ConfigurationRecordIsPresent is set to 0.

4.1. Quantization Table Set

The Quantization Table Sets store a sequence of values that are equal to one less than the count of equal concurrent entries for each set of equal concurrent entries within the first half of the table (represented as <tt>len - 1</tt> in the pseudocode below) using the method described in Section 3.8.1.2. The second half doesnt need to be stored as it is identical to the first with flipped sign. scale and len_count[i][j] are temporary values used for the computing of context_count[i] and are not used outside Quantization Table Set pseudocode.

Example:

Table: 0 0 1 1 1 1 2 2 -2 -2 -2 -1 -1 -1 -1 0

Stored values: 1, 3, 1

QuantizationTableSet has its own initial states, all set to 128.

pseudocode	type

```
QuantizationTableSet( i ) {
    scale = 1
    for (j = 0; j < MAX_CONTEXT_INPUTS; j++) {
        QuantizationTable( i, j, scale )
        scale *= 2 * len_count[ i ][ j ] - 1
    }
    context_count[ i ] = ceil( scale / 2 )
}
```

MAX_CONTEXT_INPUTS is 5.

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
QuantizationTable(i, j, scale) {                                    |
    v = 0                                                           |
    for (k = 0; k < 128;) {                                         |
        len - 1                                                     | ur
        for (n = 0; n < len; n++) {                                 |
            quant_tables[ i ][ j ][ k ] = scale * v                 |
            k++                                                     |
        }                                                           |
        v++                                                         |
    }                                                               |
    for (k = 1; k < 128; k++) {                                     |
        quant_tables[ i ][ j ][ 256 - k ] = \                       |
        -quant_tables[ i ][ j ][ k ]                                |
    }                                                               |
    quant_tables[ i ][ j ][ 128 ] = \                               |
    -quant_tables[ i ][ j ][ 127 ]                                  |
    len_count[ i ][ j ] = v                                         |
}                                                                   |
```

4.1.1. quant_tables

quant_tables[i][j][k] indicates the Quantization Table value of the Quantized Sample Difference k of the Quantization Table j of the Quantization Table Set i.

4.1.2. context_count

context_count[i] indicates the count of contexts for Quantization Table Set i. context_count[i] MUST be less than or equal to 32768.

4.2. Parameters

The Parameters section contains significant characteristics about the decoding configuration used for all instances of Frame (in FFV1 version 0 and 1) or the whole FFV1 bitstream (other versions), including the stream version, color configuration, and quantization tables. Figure 28 describes the contents of the bitstream.

Parameters has its own initial states, all set to 128.

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
Parameters( ) {                                                     |
    version                                                         | ur
    if (version >= 3) {                                             |
        micro_version                                               | ur
    }                                                               |
    coder_type                                                      | ur
    if (coder_type > 1) {                                           |
        for (i = 1; i < 256; i++) {                                 |
            state_transition_delta[ i ]                             | sr
        }                                                           |
    }                                                               |
    colorspace_type                                                 | ur
    if (version >= 1) {                                             |
        bits_per_raw_sample                                         | ur
    }                                                               |
    chroma_planes                                                   | br
    log2_h_chroma_subsample                                         | ur
    log2_v_chroma_subsample                                         | ur
    extra_plane                                                     | br
    if (version >= 3) {                                             |
        num_h_slices - 1                                            | ur
        num_v_slices - 1                                            | ur
        quant_table_set_count                                       | ur
    }                                                               |
    for (i = 0; i < quant_table_set_count; i++) {                   |
        QuantizationTableSet( i )                                   |
    }                                                               |
    if (version >= 3) {                                             |
        for (i = 0; i < quant_table_set_count; i++) {               |
            states_coded                                            | br
            if (states_coded) {                                     |
                for (j = 0; j < context_count[ i ]; j++) {          |
                    for (k = 0; k < CONTEXT_SIZE; k++) {            |
                        initial_state_delta[ i ][ j ][ k ]          | sr
                    }                                               |
                }                                                   |
            }                                                       |
        }                                                           |
        ec                                                          | ur
        intra                                                       | ur
    }                                                               |
}                                                                   |
```

Figure 28: A pseudocode description of the bitstream contents.

CONTEXT_SIZE is 32.

4.2.1. version

version specifies the version of the FFV1 bitstream.

Each version is incompatible with other versions: decoders SHOULD reject FFV1 bitstreams due to an unknown version.

Decoders SHOULD reject FFV1 bitstreams with version <= 1 && ConfigurationRecordIsPresent == 1.

Decoders SHOULD reject FFV1 bitstreams with version >= 3 && ConfigurationRecordIsPresent == 0.

value	version
0	FFV1 version 0
1	FFV1 version 1
2	reserved*
3	FFV1 version 3
4	FFV1 version 4
Other	reserved for future use

Table 5: The definitions for version values.

* Version 2 was experimental and this document does not describe it.

4.2.2. micro_version

micro_version specifies the micro-version of the FFV1 bitstream.

After a version is considered stable (a micro-version value is assigned to be the first stable variant of a specific version), each new micro-version after this first stable variant is compatible with the previous micro-version: decoders SHOULD NOT reject FFV1 bitstreams due to an unknown micro-version equal or above the micro-version considered as stable.

Meaning of micro_version for version 3:

value	micro_version
0...3	reserved*
4	first stable variant
Other	reserved for future use

Table 6: The definitions for micro_version values for FFV1 version 3.

* Development versions may be incompatible with the stable variants.

Meaning of micro_version for version 4 (note: at the time of writing of this specification, version 4 is not considered stable so the first stable micro_version value is to be announced in the future):

value	micro_version
0...TBA	reserved*
TBA	first stable variant
Other	reserved for future use

Table 7: The definitions for micro_version values for FFV1 version 4.

* Development versions which may be incompatible with the stable variants.

4.2.3. coder_type

coder_type specifies the coder used.

value	coder used
0	Golomb Rice
1	Range coder with default state transition table
2	Range coder with custom state transition table
Other	reserved for future use

Table 8: The definitions for coder_type values.

Restrictions:

If coder_type is 0, then bits_per_raw_sample SHOULD NOT be > 8.

Background: At the time of this writing, there is no known implementation of FFV1 bitstream supporting the Golomb Rice algorithm with bits_per_raw_sample greater than eight, and range coder is preferred.

4.2.4. state_transition_delta

state_transition_delta specifies the range coder custom state transition table.

If state_transition_delta is not present in the FFV1 bitstream, all range coder custom state transition table elements are assumed to be 0.

4.2.5. colorspace_type

colorspace_type specifies the color space encoded, the pixel transformation used by the encoder, the extra Plane content, as well as interleave method.

value	color space encoded	pixel transformation	extra Plane content	interleave method
0	YCbCr	None	Transparency	Plane then Line
1	RGB	JPEG 2000 RCT	Transparency	Line then Plane
Other	reserved for future use	reserved for future use	reserved for future use	reserved for future use

Table 9: The definitions for colorspace_type values.

FFV1 bitstreams with colorspace_type == 1 && (chroma_planes != 1 || log2_h_chroma_subsample != 0 || log2_v_chroma_subsample != 0) are not part of this specification.

4.2.6. chroma_planes

chroma_planes indicates if chroma (color) Planes are present.

value	presence
0	chroma Planes are not present
1	chroma Planes are present

Table 10: The definitions for chroma_planes values.

4.2.7. bits_per_raw_sample

bits_per_raw_sample indicates the number of bits for each Sample. Inferred to be 8 if not present.

value	bits for each sample
0	reserved*
Other	the actual bits for each Sample

Table 11: The definitions for bits_per_raw_sample values.

* Encoders MUST NOT store bits_per_raw_sample = 0. Decoders SHOULD accept and interpret bits_per_raw_sample = 0 as 8.

4.2.8. log2_h_chroma_subsample

log2_h_chroma_subsample indicates the subsample factor, stored in powers to which the number 2 is raised, between luma and chroma width (chroma_width = 2 ^ -log2_h_chroma_subsample * luma_width).

4.2.9. log2_v_chroma_subsample

log2_v_chroma_subsample indicates the subsample factor, stored in powers to which the number 2 is raised, between luma and chroma height (chroma_height = 2 ^ -log2_v_chroma_subsample * luma_height).

4.2.10. extra_plane

extra_plane indicates if an extra Plane is present.

value	presence
0	extra Plane is not present
1	extra Plane is present

Table 12: The definitions for extra_plane values.

4.2.11. num_h_slices

num_h_slices indicates the number of horizontal elements of the Slice raster.

Inferred to be 1 if not present.

4.2.12. num_v_slices

num_v_slices indicates the number of vertical elements of the Slice raster.

Inferred to be 1 if not present.

4.2.13. quant_table_set_count

quant_table_set_count indicates the number of Quantization Table Sets. quant_table_set_count MUST be less than or equal to 8.

Inferred to be 1 if not present.

MUST NOT be 0.

4.2.14. states_coded

states_coded indicates if the respective Quantization Table Set has the initial states coded.

Inferred to be 0 if not present.

value	initial states
0	initial states are not present and are assumed to be all 128
1	initial states are present

Table 13: The definitions for states_coded values.

4.2.15. initial_state_delta

initial_state_delta[i][j][k] indicates the initial range coder state, and it is encoded using k as context index for the range coder and the following pseudocode:

```
pred = j ? initial_states[ i ][j - 1][ k ] : 128
```

Figure 29: Predictor value for the coding of initial_state_delta[i][j][k].

```
initial_state[ i ][ j ][ k ] =
       ( pred + initial_state_delta[ i ][ j ][ k ] ) & 255
```

Figure 30: Description of the coding of initial_state_delta[i][j][k].

4.2.16. ec

ec indicates the error detection/correction type.

value	error detection/correction type
0	32-bit CRC in ConfigurationRecord
1	32-bit CRC in Slice and ConfigurationRecord
Other	reserved for future use

Table 14: The definitions for ec values.

4.2.17. intra

intra indicates the constraint on keyframe in each instance of Frame.

Inferred to be 0 if not present.

value	relationship
0	keyframe can be 0 or 1 (non keyframes or keyframes)
1	keyframeMUST be 1 (keyframes only)
Other	reserved for future use

Table 15: The definitions for intra values.

4.3. Configuration Record

In the case of a FFV1 bitstream with version >= 3, a Configuration Record is stored in the underlying container as described in Section 4.3.3. It contains the Parameters used for all instances of Frame. The size of the Configuration Record, NumBytes, is supplied by the underlying container.

```
pseudocode                                                    | type
--------------------------------------------------------------|-----
ConfigurationRecord( NumBytes ) {                             |
    ConfigurationRecordIsPresent = 1                          |
    Parameters( )                                             |
    while (remaining_symbols_in_syntax(NumBytes - 4)) {       |
        reserved_for_future_use                               | br/ur/sr
    }                                                         |
    configuration_record_crc_parity                           | u(32)
}                                                             |
```

4.3.1. reserved_for_future_use

reserved_for_future_use is a placeholder for future updates of this specification.

Encoders conforming to this version of this specification SHALL NOT write reserved_for_future_use.

Decoders conforming to this version of this specification SHALL ignore reserved_for_future_use.

4.3.2. configuration_record_crc_parity

configuration_record_crc_parity is 32 bits that are chosen so that the Configuration Record as a whole has a CRC remainder of zero.

This is equivalent to storing the CRC remainder in the 32-bit parity.

The CRC generator polynomial used is described in Section 4.9.3.

4.3.3. Mapping FFV1 into Containers

This Configuration Record can be placed in any file format that supports Configuration Records, fitting as much as possible with how the file format stores Configuration Records. The Configuration Record storage place and NumBytes are currently defined and supported for the following formats:

4.3.3.1. Audio Video Interleave (AVI) File Format

The Configuration Record extends the stream format chunk ("AVI ", "hdlr", "strl", "strf") with the ConfigurationRecord bitstream.

See [AVI] for more information about chunks.

NumBytes is defined as the size, in bytes, of the "strf" chunk indicated in the chunk header minus the size of the stream format structure.

4.3.3.2. ISO Base Media File Format

The Configuration Record extends the sample description box ("moov", "trak", "mdia", "minf", "stbl", "stsd") with a "glbl" box that contains the ConfigurationRecord bitstream. See [ISO.14496-12.2020] for more information about boxes.

NumBytes is defined as the size, in bytes, of the "glbl" box indicated in the box header minus the size of the box header.

4.3.3.3. NUT File Format

The codec_specific_data element (in stream_header packet) contains the ConfigurationRecord bitstream. See [NUT] for more information about elements.

NumBytes is defined as the size, in bytes, of the codec_specific_data element as indicated in the "length" field of codec_specific_data.

4.3.3.4. Matroska File Format

FFV1 SHOULD use V_FFV1 as the Matroska Codec ID. For FFV1 versions 2 or less, the Matroska CodecPrivate Element SHOULD NOT be used. For FFV1 versions 3 or greater, the Matroska CodecPrivate Element MUST contain the FFV1 Configuration Record structure and no other data. See [I-D.ietf-cellar-matroska] for more information about elements.

NumBytes is defined as the Element Data Size of the CodecPrivate Element.

4.4. Frame

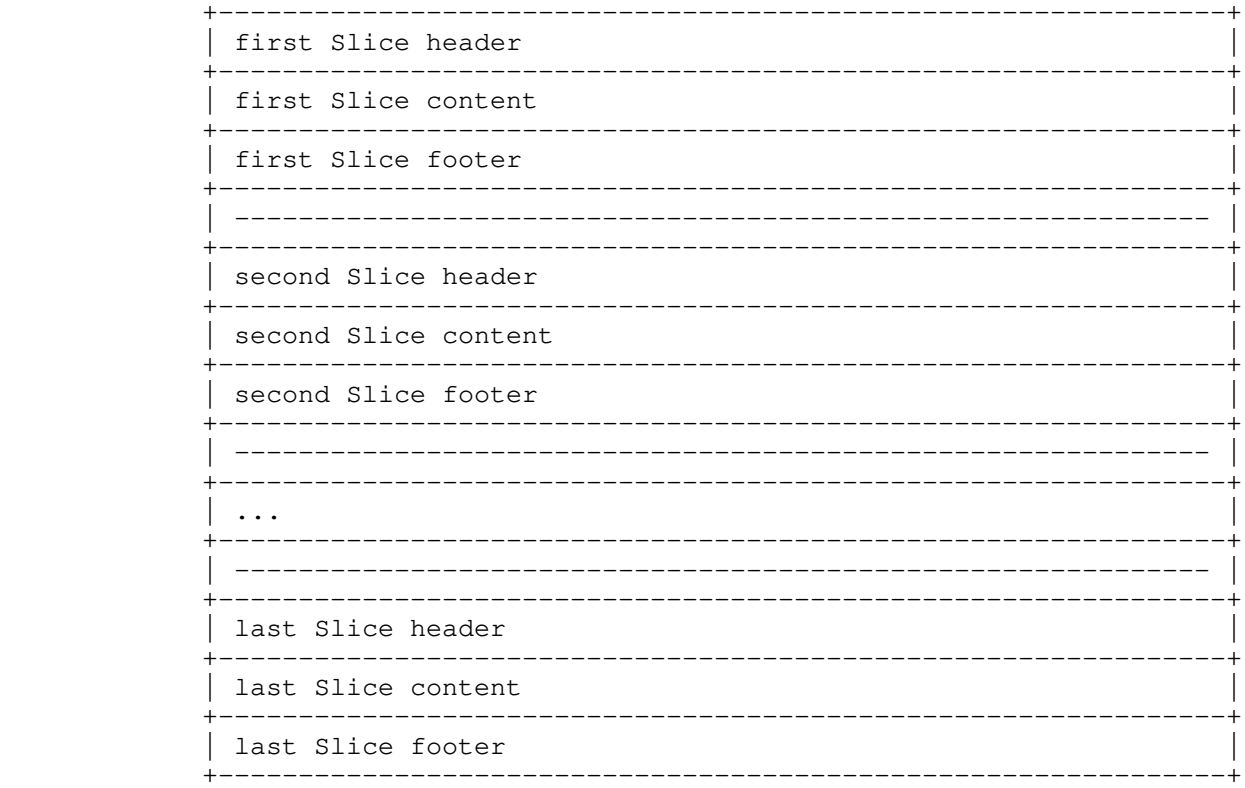
A Frame is an encoded representation of a complete static image. The whole Frame is provided by the underlaying container.

A Frame consists of the keyframe field, Parameters (if version <= 1), and a sequence of independent Slices. The pseudocode below describes the contents of a Frame.

The keyframe field has its own initial state, set to 128.

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
Frame( NumBytes ) {                                                 |
    keyframe                                                        | br
    if (keyframe && !ConfigurationRecordIsPresent {                 |
        Parameters( )                                               |
    }                                                               |
    while (remaining_bits_in_bitstream( NumBytes )) {               |
        Slice( )                                                    |
    }                                                               |
}                                                                   |
```

The following is an architecture overview of Slices in a Frame:



4.5. Slice

A Slice is an independent, spatial subsection of a Frame that is encoded separately from another region of the same Frame. The use of more than one Slice per Frame provides opportunities for taking advantage of multithreaded encoding and decoding.

A Slice consists of a Slice Header (when relevant), a Slice Content, and a Slice Footer (when relevant). The pseudocode below describes the contents of a Slice.

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
Slice( ) {                                                          |
    if (version >= 3) {                                             |
        SliceHeader( )                                              |
    }                                                               |
    SliceContent( )                                                 |
    if (coder_type == 0) {                                          |
        while (!byte_aligned()) {                                   |
            padding                                                 | u(1)
        }                                                           |
    }                                                               |
    if (version <= 1) {                                             |
        while (remaining_bits_in_bitstream( NumBytes ) != 0) {      |
            reserved                                                | u(1)
        }                                                           |
    }                                                               |
    if (version >= 3) {                                             |
        SliceFooter( )                                              |
    }                                                               |
}                                                                   |
```

padding specifies a bit without any significance and used only for byte alignment. padding MUST be 0.

reserved specifies a bit without any significance in this specification but may have a significance in a later revision of this specification.

Encoders SHOULD NOT fill reserved.

Decoders SHOULD ignore reserved.

4.6. Slice Header

A Slice Header provides information about the decoding configuration of the Slice, such as its spatial position, size, and aspect ratio. The pseudocode below describes the contents of the Slice Header.

Slice Header has its own initial states, all set to 128.

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
SliceHeader( ) {                                                    |
    slice_x                                                         | ur
    slice_y                                                         | ur
    slice_width - 1                                                 | ur
    slice_height - 1                                                | ur
    for (i = 0; i < quant_table_set_index_count; i++) {             |
        quant_table_set_index[ i ]                                  | ur
    }                                                               |
    picture_structure                                               | ur
    sar_num                                                         | ur
    sar_den                                                         | ur
    if (version >= 4) {                                             |
        reset_contexts                                              | br
        slice_coding_mode                                           | ur
    }                                                               |
}                                                                   |
```

4.6.1. slice_x

slice_x indicates the x position on the Slice raster formed by num_h_slices.

Inferred to be 0 if not present.

4.6.2. slice_y

slice_y indicates the y position on the Slice raster formed by num_v_slices.

Inferred to be 0 if not present.

4.6.3. slice_width

slice_width indicates the width on the Slice raster formed by num_h_slices.

Inferred to be 1 if not present.

4.6.4. slice_height

slice_height indicates the height on the Slice raster formed by num_v_slices.

Inferred to be 1 if not present.

4.6.5. quant_table_set_index_count

quant_table_set_index_count is defined as the following:

```
1 + ( ( chroma_planes || version <= 3 ) ? 1 : 0 )
    + ( extra_plane ? 1 : 0 )
```

4.6.6. quant_table_set_index

quant_table_set_index indicates the Quantization Table Set index to select the Quantization Table Set and the initial states for the Slice Content.

Inferred to be 0 if not present.

4.6.7. picture_structure

picture_structure specifies the temporal and spatial relationship of each Line of the Frame.

Inferred to be 0 if not present.

value	picture structure used
0	unknown
1	top field first
2	bottom field first
3	progressive
Other	reserved for future use

Table 16: The definitions for picture_structure values.

4.6.8. sar_num

sar_num specifies the Sample aspect ratio numerator.

Inferred to be 0 if not present.

A value of 0 means that aspect ratio is unknown.

Encoders MUST write 0 if the Sample aspect ratio is unknown.

If sar_den is 0, decoders SHOULD ignore the encoded value and consider that sar_num is 0.

4.6.9. sar_den

sar_den specifies the Sample aspect ratio denominator.

Inferred to be 0 if not present.

A value of 0 means that aspect ratio is unknown.

Encoders MUST write 0 if the Sample aspect ratio is unknown.

If sar_num is 0, decoders SHOULD ignore the encoded value and consider that sar_den is 0.

4.6.10. reset_contexts

reset_contexts indicates if Slice contexts MUST be reset.

Inferred to be 0 if not present.

4.6.11. slice_coding_mode

slice_coding_mode indicates the Slice coding mode.

Inferred to be 0 if not present.

value	Slice coding mode
0	Range Coding or Golomb Rice
1	raw PCM
Other	reserved for future use

Table 17: The definitions for slice_coding_mode values.

4.7. Slice Content

A Slice Content contains all Line elements part of the Slice.

Depending on the configuration, Line elements are ordered by Plane then by row (YCbCr) or by row then by Plane (RGB).

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
SliceContent( ) {                                                   |
    if (colorspace_type == 0) {                                     |
        for (p = 0; p < primary_color_count; p++) {                 |
            for (y = 0; y < plane_pixel_height[ p ]; y++) {         |
                Line( p, y )                                        |
            }                                                       |
        }                                                           |
    } else if (colorspace_type == 1) {                              |
        for (y = 0; y < slice_pixel_height; y++) {                  |
            for (p = 0; p < primary_color_count; p++) {             |
                Line( p, y )                                        |
            }                                                       |
        }                                                           |
    }                                                               |
}                                                                   |
```

4.7.1. primary_color_count

primary_color_count is defined as the following:

```
1 + ( chroma_planes ? 2 : 0 ) + ( extra_plane ? 1 : 0 )
```

4.7.2. plane_pixel_height

plane_pixel_height[p] is the height in pixels of Plane p of the Slice. It is defined as the following:

```
chroma_planes == 1 && (p == 1 || p == 2)
    ? ceil(slice_pixel_height / (1 << log2_v_chroma_subsample))
    : slice_pixel_height
```

4.7.3. slice_pixel_height

slice_pixel_height is the height in pixels of the Slice. It is defined as the following:

```
floor(
        ( slice_y + slice_height )
        * slice_pixel_height
        / num_v_slices
    ) - slice_pixel_y.
```

4.7.4. slice_pixel_y

slice_pixel_y is the Slice vertical position in pixels. It is defined as the following:

```
floor( slice_y * frame_pixel_height / num_v_slices )
```

4.8. Line

A Line is a list of the Sample Differences (relative to the predictor) of primary color components. The pseudocode below describes the contents of the Line.

```
pseudocode                                                          | type
--------------------------------------------------------------------|-----
Line( p, y ) {                                                      |
    if (colorspace_type == 0) {                                     |
        for (x = 0; x < plane_pixel_width[ p ]; x++) {              |
            sample_difference[ p ][ y ][ x ]                        | sd
        }                                                           |
    } else if (colorspace_type == 1) {                              |
        for (x = 0; x < slice_pixel_width; x++) {                   |
            sample_difference[ p ][ y ][ x ]                        | sd
        }                                                           |
    }                                                               |
}                                                                   |
```

4.8.1. plane_pixel_width

plane_pixel_width[p] is the width in pixels of Plane p of the Slice. It is defined as the following:

```
chroma_planes == 1 && (p == 1 || p == 2)
    ? ceil( slice_pixel_width / (1 << log2_h_chroma_subsample) )
    : slice_pixel_width.
```

4.8.2. slice_pixel_width

slice_pixel_width is the width in pixels of the Slice. It is defined as the following:

```
floor(
        ( slice_x + slice_width )
        * slice_pixel_width
        / num_h_slices
    ) - slice_pixel_x
```

4.8.3. slice_pixel_x

slice_pixel_x is the Slice horizontal position in pixels. It is defined as the following:

```
floor( slice_x * frame_pixel_width / num_h_slices )
```

4.8.4. sample_difference

sample_difference[p][y][x] is the Sample Difference for Sample at Plane p, y position y, and x position x. The Sample value is computed based on median predictor and context described in Section 3.2.

4.9. Slice Footer

A Slice Footer provides information about Slice size and (optionally) parity. The pseudocode below describes the contents of the Slice Footer.

Note: Slice Footer is always byte aligned.

```
pseudocode                                                    | type
--------------------------------------------------------------|-----
SliceFooter( ) {                                              |
    slice_size                                                | u(24)
    if (ec) {                                                 |
        error_status                                          | u(8)
        slice_crc_parity                                      | u(32)
    }                                                         |
}                                                             |
```

4.9.1. slice_size

slice_size indicates the size of the Slice in bytes.

Note: this allows finding the start of Slices before previous Slices have been fully decoded and allows parallel decoding as well as error resilience.

4.9.2. error_status

error_status specifies the error status.

value	error status
0	no error
1	Slice contains a correctable error
2	Slice contains a uncorrectable error
Other	reserved for future use

Table 18: The definitions for error_status values.

4.9.3. slice_crc_parity

slice_crc_parity is 32 bits that are chosen so that the Slice as a whole has a CRC remainder of 0.

This is equivalent to storing the CRC remainder in the 32-bit parity.

The CRC generator polynomial used is the standard IEEE CRC polynomial (0x104C11DB7) with initial value 0, without pre-inversion, and without post-inversion.

5. Restrictions

To ensure that fast multithreaded decoding is possible, starting with version 3 and if frame_pixel_width * frame_pixel_height is more than 101376, slice_width * slice_height MUST be less or equal to num_h_slices * num_v_slices / 4. Note: 101376 is the frame size in pixels of a 352x288 frame also known as CIF (Common Intermediate Format) frame size format.

For each Frame, each position in the Slice raster MUST be filled by one and only one Slice of the Frame (no missing Slice position and no Slice overlapping).

For each Frame with a keyframe value of 0, each Slice MUST have the same value of slice_x, slice_y, slice_width, and slice_height as a Slice in the previous Frame, except if reset_contexts is 1.

6. Security Considerations

Like any other codec (such as [RFC6716]), FFV1 should not be used with insecure ciphers or cipher modes that are vulnerable to known plaintext attacks. Some of the header bits as well as the padding are easily predictable.

Implementations of the FFV1 codec need to take appropriate security considerations into account. Those related to denial of service are outlined in Section 2.1 of [RFC4732]. It is extremely important for the decoder to be robust against malicious payloads. Malicious payloads MUST NOT cause the decoder to overrun its allocated memory or to take an excessive amount of resources to decode. An overrun in allocated memory could lead to arbitrary code execution by an attacker. The same applies to the encoder, even though problems in encoders are typically rarer. Malicious video streams MUST NOT cause the encoder to misbehave because this would allow an attacker to attack transcoding gateways. A frequent security problem in image and video codecs is failure to check for integer overflows. An example is allocating frame_pixel_width * frame_pixel_height in pixel count computations without considering that the multiplication result may have overflowed the range of the arithmetic type. The range coder could, if implemented naively, read one byte over the end. The implementation MUST ensure that no read outside allocated and initialized memory occurs.

None of the content carried in FFV1 is intended to be executable.

7. IANA Considerations

IANA has registered the following values.

7.1. Media Type Definition

This registration is done using the template defined in [RFC6838] and following [RFC4855].

Type name: video

Subtype name: FFV1

Required parameters: None.

Optional parameters: These parameters are used to signal the capabilities of a receiver implementation. These parameters MUST NOT be used for any other purpose.

version: The version of the FFV1 encoding as defined by

Section 4.2.1.
micro_version: The micro_version of the FFV1 encoding as defined by Section 4.2.2.
coder_type: The coder_type of the FFV1 encoding as defined by Section 4.2.3.
colorspace_type: The colorspace_type of the FFV1 encoding as defined by Section 4.2.5.
bits_per_raw_sample: The bits_per_raw_sample of the FFV1 encoding as defined by Section 4.2.7.
max_slices: The value of max_slices is an integer indicating the maximum count of Slices within a Frame of the FFV1 encoding.

Encoding considerations: This media type is defined for encapsulation in several audiovisual container formats and contains binary data; see Section 4.3.3. This media type is framed binary data; see Section 4.8 of [RFC6838].

Security considerations: See Section 6 of this document.

Interoperability considerations: None.

Published specification: RFC XXXX.

[RFC Editor: Upon publication as an RFC, please replace "XXXX" with the number assigned to this document and remove this note.]

Applications that use this media type: Any application that requires the transport of lossless video can use this media type. Some examples are, but not limited to, screen recording, scientific imaging, and digital video preservation.
Fragment identifier considerations: N/A.
Additional information: None.
Person & email address to contact for further information: Michael N iedermayer (michael@niedermayer.cc (mailto:michael@niedermayer.cc))
Intended usage: COMMON
Restrictions on usage: None.
Author: Dave Rice (dave@dericed.com (mailto:dave@dericed.com))
Change controller: IETF CELLAR Working Group delegated from the IESG.

8. Changelog

See https://github.com/FFmpeg/FFV1/commits/master (https://github.com/FFmpeg/FFV1/commits/master)

[RFC Editor: Please remove this Changelog section prior to publication.]

9. References

9.1. Normative References

[ISO.9899.2018]
International Organization for Standardization, "Information technology - Programming languages - C", ISO/IEC 9899:2018, June 2018.

[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>.

[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>.

[RFC4855] Casner, S., "Media Type Registration of RTP Payload Formats", RFC 4855, DOI 10.17487/RFC4855, February 2007, <https://www.rfc-editor.org/info/rfc4855>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[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>.

9.2. Informative References

[AddressSanitizer]
Clang Project, "AddressSanitizer", Clang 12 documentation, <https://clang.llvm.org/docs/AddressSanitizer.html>.

[AVI] Microsoft, "AVI RIFF File Reference", <https://docs.microsoft.com/en-us/windows/win32/directshow/avi-riff-file-reference>.

[FFV1GO] Buitenhuis, D., "FFV1 Decoder in Go", 2019, <https://github.com/dwbuiten/go-ffv1>.

[HuffYUV] Rudiak-Gould, B., "HuffYUV revisited", December 2003, <https://web.archive.org/web/20040402121343/http://cultact-server.novi.dk/kpo/huffyuv/huffyuv.html>.

[I-D.ietf-cellar-ffv1]
Niedermayer, M., Rice, D., and J. Martinez, "FFV1 Video Coding Format Versions 0, 1, and 3", Work in Progress, Internet-Draft, draft-ietf-cellar-ffv1-20, 23 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-ffv1-20>.

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

[ISO.14495-1.1999]
International Organization for Standardization, "Information technology -- Lossless and near-lossless compression of continuous-tone still images: Baseline", ISO/IEC 14495-1:1999, December 1999.

[ISO.14496-10.2020]
International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 10: Advanced Video Coding", ISO/IEC 14496-10:2020, December 2020.

[ISO.14496-12.2020]
International Organization for Standardization, "Information technology -- Coding of audio-visual objects -- Part 12: ISO base media file format", ISO/IEC 14496-12:2020, December 2020.

[ISO.15444-1.2019]
International Organization for Standardization, "Information technology -- JPEG 2000 image coding system: Core coding system", ISO/IEC 15444-1:2019, October 2019.

[MediaConch]
MediaArea.net, "MediaConch", 2018, <https://mediaarea.net/MediaConch>.

[NUT] Niedermayer, M., "NUT Open Container Format", December 2013, <https://ffmpeg.org/~michael/nut.txt>.

[Range-Encoding]
Martin, G. N. N., "Range encoding: an algorithm for removing redundancy from a digitised message", Proceedings of the Conference on Video and Data Recording, Institution of Electronic and Radio Engineers, Hampshire, England, July 1979.

[REFIMPL] Niedermayer, M., "The reference FFV1 implementation / the FFV1 codec in FFmpeg", <https://ffmpeg.org/doxygen/trunk/ffv1_8h.html>.

[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>.

[Valgrind] Valgrind Developers, "Valgrind website", <https://valgrind.org/>.

[YCbCr] Wikipedia, "YCbCr", 25 May 2021, <https://en.wikipedia.org/w/index.php?title=YCbCr&oldid=1025097882>.

Appendix A. Multithreaded Decoder Implementation Suggestions

This appendix is informative.

The FFV1 bitstream is parsable in two ways: in sequential order as described in this document or with the pre-analysis of the footer of each Slice. Each Slice footer contains a slice_size field so the boundary of each Slice is computable without having to parse the Slice content. That allows multithreading as well as independence of Slice content (a bitstream error in a Slice header or Slice content has no impact on the decoding of the other Slices).

After having checked the keyframe field, a decoder SHOULD parse slice_size fields, from slice_size of the last Slice at the end of the Frame up to slice_size of the first Slice at the beginning of the Frame before parsing Slices, in order to have Slice boundaries. A decoder MAY fall back on sequential order e.g., in case of a corrupted Frame (e.g., frame size unknown or slice_size of Slices not coherent) or if there is no possibility of seeking into the stream.

Appendix B. Future Handling of Some Streams Created by Nonconforming Encoders

This appendix is informative.

Some bitstreams were found with 40 extra bits corresponding to error_status and slice_crc_parity in the reserved bits of Slice. Any revision of this specification should avoid adding 40 bits of content after SliceContent if version == 0 or version == 1, otherwise a decoder conforming to the revised specification could not distinguish between a revised bitstream and such buggy bitstream in the wild.

Appendix C. FFV1 Implementations

This appendix provides references to a few notable implementations of FFV1.

C.1. FFmpeg FFV1 Codec

This reference implementation [REFIMPL] contains no known buffer overflow or cases where a specially crafted packet or video segment could cause a significant increase in CPU load.

The reference implementation [REFIMPL] was validated in the following conditions:

* Sending the decoder valid packets generated by the reference encoder and verifying that the decoder's output matches the encoder's input.
* Sending the decoder packets generated by the reference encoder and then subjected to random corruption.
* Sending the decoder random packets that are not FFV1.

In all of the conditions above, the decoder and encoder was run inside the Valgrind memory debugger [Valgrind] as well as the Clang AddressSanitizer [AddressSanitizer], which tracks reads and writes to invalid memory regions as well as the use of uninitialized memory. There were no errors reported on any of the tested conditions.

C.2. FFV1 Decoder in Go

An FFV1 decoder [FFV1GO] was written in Go by Derek Buitenhuis during the work to develop this document.

C.3. MediaConch

The developers of the MediaConch project [MediaConch] created an independent FFV1 decoder as part of that project to validate FFV1 bitstreams. This work led to the discovery of three conflicts between existing FFV1 implementations and draft versions of this document. These issues are addressed by Section 3.3.1, Section 3.7.2.1, and Appendix B.

Authors' Addresses

Michael Niedermayer
Email: michael@niedermayer.cc

Dave Rice
Email: dave@dericed.com

Jérôme Martinez
Email: jerome@mediaarea.net

cellar
Internet-Draft
Intended status: Standards Track
Expires: 17 July 2024

M.Q.C. van Beurden

A. Weaver

14 January 2024

Free Lossless Audio Codec
draft-ietf-cellar-flac-14

Abstract

This document defines the Free Lossless Audio Codec (FLAC) format and its streamable subset. FLAC is designed to reduce the amount of computer storage space needed to store digital audio signals without losing information in doing so (i.e., lossless). FLAC is free in the sense that its specification is open and its reference implementation is open-source. Compared to other lossless (audio) coding formats, FLAC is a format with low complexity and can be coded to and from with little computing resources. Decoding of FLAC has seen many independent implementations on many different platforms, and both encoding and decoding can be implemented without needing floating-point arithmetic.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 July 2024.

Copyright Notice

Table of Contents

1. Introduction

This document defines the FLAC format and its streamable subset. FLAC files and streams can code for pulse-code modulated (PCM) audio with 1 to 8 channels, sample rates from 1 up to 1048575 hertz and bit depths from 4 up to 32 bits. Most tools for coding to and decoding from the FLAC format have been optimized for CD-audio, which is PCM audio with 2 channels, a sample rate of 44.1 kHz, and a bit depth of 16 bits.

FLAC is able to achieve lossless compression because samples in audio signals tend to be highly correlated with their close neighbors. In contrast with general-purpose compressors, which often use dictionaries, do run-length coding, or exploit long-term repetition, FLAC removes redundancy solely in the very short term, looking back at at most 32 samples.

The coding methods provided by the FLAC format work best on PCM audio signals, of which the samples have a signed representation and are centered around zero. Audio signals in which samples have an unsigned representation must be transformed to a signed representation as described in this document in order to achieve reasonable compression. The FLAC format is not suited for compressing audio that is not PCM.

2. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Values expressed as u(n) represent unsigned big-endian integer using n bits. Values expressed as s(n) represent signed big-endian integer using n bits, signed two's complement. Where necessary n is expressed as an equation using * (multiplication), / (division), + (addition), or - (subtraction). An inclusive range of the number of bits expressed is represented with an ellipsis, such as u(m...n).

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

3. Definitions

* *Lossless compression*: reducing the amount of computer storage space needed to store data without needing to remove or irreversibly alter any of this data in doing so. In other words, decompressing losslessly compressed information returns exactly the original data.

* *Lossy compression*: like lossless compression, but instead removing, irreversibly altering, or only approximating information for the purpose of further reducing the amount of computer storage space needed. In other words, decompressing lossy compressed information returns an approximation of the original data.

* *Block*: A (short) section of linear pulse-code modulated audio with one or more channels.

* *Subblock*: All samples within a corresponding block for one channel. One or more subblocks form a block, and all subblocks in a certain block contain the same number of samples.

* *Frame*: A frame header, one or more subframes, and a frame footer. It encodes the contents of a corresponding block.

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

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

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

* *Bit depth* or *bits per sample*: the number of bits used to contain each sample. This MUST be the same for all subblocks in a block but MAY be different for different subframes in a frame because of interchannel decorrelation. (See Section 4.2 for details on interchannel decorrelation)

* *Predictor*: a model used to predict samples in an audio signal based on past samples. FLAC uses such predictors to remove redundancy in a signal in order to be able to compress it.

* *Linear predictor*: a predictor using linear prediction (see [LinearPrediction]). This is also called *linear predictive coding (LPC)*. With a linear predictor, each prediction is a linear combination of past samples, hence the name. A linear predictor has a causal discrete-time finite impulse response (see [FIR]).

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

* *Fixed predictor*: a linear predictor in which the model parameters are the same across all FLAC files, and thus do not need to be stored.

* *Predictor order*: the number of past samples that a predictor uses. For example, a 4th order predictor uses the 4 samples directly preceding a certain sample to predict it. In FLAC, samples used in a predictor are always consecutive, and are always the samples directly before the sample that is being predicted.

* *Residual*: The audio signal that remains after a predictor has been subtracted from a subblock. If the predictor has been able to remove redundancy from the signal, the samples of the remaining signal (the *residual samples*) will have, on average, a smaller numerical value than the original signal.

* *Rice code*: A variable-length code (see [VarLengthCode]) that compresses data by making use of the observation that, after using an effective predictor, most residual samples are closer to zero than the original samples, while still allowing for a small part of the samples to be much larger.

4. Conceptual overview

Similar to many other audio coders, a FLAC file is encoded following the steps below. On decoding a FLAC file, these steps are undone in reverse order, i.e., from bottom to top.

* *Blocking* (see Section 4.1). The input is split up into many contiguous blocks.

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

* *Prediction* (see Section 4.3). To remove redundancy in a signal, a predictor is stored for each subblock or its transformation as formed in the previous step. A predictor consists of a simple mathematical description that can be used, as the name implies, to predict a certain sample from the samples that preceded it. As this prediction is rarely exact, the error of this prediction is passed on to the next stage. The predictor of each subblock is completely independent from other subblocks. Since the methods of prediction are known to both the encoder and decoder, only the parameters of the predictor need to be included in the compressed stream. If no usable predictor can be found for a certain subblock, the signal is stored uncompressed and the next stage is skipped.

* *Residual Coding* (see Section 4.4). As the predictor does not describe the signal exactly, the difference between the original signal and the predicted signal (called the error or residual signal) is coded losslessly. If the predictor is effective, the residual signal will require fewer bits per sample than the original signal. FLAC uses Rice coding, a subset of Golomb coding, with either 4-bit or 5-bit parameters to code the residual signal.

In addition, FLAC specifies a metadata system (see Section 8), which allows arbitrary information about the stream to be included at the beginning of the stream.

4.1. Blocking

The block size used for audio data has a direct effect on the compression ratio. If the block size is too small, the resulting large number of frames means that a disproportionate amount of bytes will be spent on frame headers. If the block size is too large, the characteristics of the signal may vary so much that the encoder will be unable to find a good predictor. In order to simplify encoder/decoder design, FLAC imposes a minimum block size of 16 samples, except for the last block, and a maximum block size of 65535 samples. The last block is allowed to be smaller than 16 samples to be able to match the length of the encoded audio without using padding.

While the block size does not have to be constant in a FLAC file, it is often difficult to find the optimal arrangement of block sizes for maximum compression. Because of this, the FLAC format explicitly stores whether a file has a constant or a variable block size throughout the stream, and stores a block number instead of a sample number to slightly improve compression if a stream has a constant block size.

4.2. Interchannel Decorrelation

In many audio files, channels are correlated. The FLAC format can exploit this correlation in stereo files by not directly coding subblocks into subframes, but instead coding an average of all samples in both subblocks (a mid channel) or the difference between all samples in both subblocks (a side channel). The following combinations are possible:

* *Independent*. All channels are coded independently. All non-stereo files MUST be encoded this way.

* *Mid-side*. A left and right subblock are converted to mid and side subframes. To calculate a sample for a mid subframe, the corresponding left and right samples are summed and the result is shifted right by 1 bit. To calculate a sample for a side subframe, the corresponding right sample is subtracted from the corresponding left sample. On decoding, all mid channel samples have to be shifted left by 1 bit. Also, if a side channel sample is odd, 1 has to be added to the corresponding mid channel sample after it has been shifted left by one bit. To reconstruct the left channel, the corresponding samples in the mid and side subframes are added and the result shifted right by 1 bit, while for the right channel the side channel has to be subtracted from the mid channel and the result shifted right by 1 bit.

* *Left-side*. The left subblock is coded and the left and right subblocks are used to code a side subframe. The side subframe is constructed in the same way as for mid-side. To decode, the right subblock is restored by subtracting the samples in the side subframe from the corresponding samples in the the left subframe.

* *Side-right*. The left and right subblocks are used to code a side subframe and the right subblock is coded. The side subframe is constructed in the same way as for mid-side. To decode, the left subblock is restored by adding the samples in the side subframe to the corresponding samples in the right subframe.

The side channel needs one extra bit of bit depth as the subtraction can produce sample values twice as large as the maximum possible in any given bit depth. The mid channel in mid-side stereo does not need one extra bit, as it is shifted right one bit. The right shift of the mid channel does not lead to lossy behavior, because an odd sample in the mid subframe must always be accompanied by a corresponding odd sample in the side subframe, which means the lost least-significant bit can be restored by taking it from the sample in the side subframe.

4.3. Prediction

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

1. *Verbatim*. Samples are stored directly, without any modeling. This method is used for inputs with little correlation, like white noise. Since the raw signal is not actually passed through the residual coding stage (it is added to the stream 'verbatim'), this method is different from using a zero-order fixed predictor.

2. *Constant*. A single sample value is stored. This method is used whenever a signal is pure DC ("digital silence"), i.e., a constant value throughout.

3. *Fixed predictor*. Samples are predicted with one of five fixed (i.e., predefined) predictors, and the error of this prediction is processed by the residual coder. These fixed predictors are well suited for predicting simple waveforms. Since the predictors are fixed, no predictor coefficients are stored. From a mathematical point of view, the predictors work by extrapolating the signal from the previous samples. The number of previous samples used is equal to the predictor order. For more information, see Section 9.2.5.

4. *Linear predictor*. Samples are predicted using past samples and a set of predictor coefficients, and the error of this prediction is processed by the residual coder. Compared to a fixed predictor, using a generic linear predictor adds overhead as predictor coefficients need to be stored. Therefore, this method of prediction is best suited for predicting more complex waveforms, where the added overhead is offset by space savings in the residual coding stage resulting from more accurate prediction. A linear predictor in FLAC has two parameters besides the predictor coefficients and the predictor order: the number of bits with which each coefficient is stored (the coefficient precision) and a prediction right shift. A prediction is formed by taking the sum of multiplying each predictor coefficient with the corresponding past sample, and dividing that sum by applying the specified right shift. For more information, see Section 9.2.6.

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

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

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

4.4. Residual Coding

If a subframe uses a predictor to approximate the audio signal, a residual is stored to 'correct' the approximation to the exact value. When an effective predictor is used, the average numerical value of the residual samples is smaller than that of the samples before prediction. While having smaller values on average, it is possible that a few 'outlier' residual samples are much larger than any of the original samples. Sometimes these outliers even exceed the range the bit depth of the original audio offers.

To be able to efficiently code such a stream of relatively small numbers with an occasional outlier, Rice coding (a subset of Golomb coding) is used. Depending on how small the numbers are that have to be coded, a Rice parameter is chosen. The numerical value of each residual sample is split into two parts by dividing it by 2^(Rice parameter), creating a quotient and a remainder. The quotient is

stored in unary form, the remainder in binary form. If indeed most residual samples are close to zero and a suitable Rice parameter is chosen, this form of coding, with a so-called variable-length code, uses fewer bits than the residual in unencoded form.

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

Quite often, the optimal Rice parameter varies over the course of a subframe. To accommodate this, the residual can be split up into partitions, where each partition has its own Rice parameter. To keep overhead and complexity low, the number of partitions used in a subframe is limited to powers of two.

The FLAC format uses two forms of Rice coding, which only differ in the number of bits used for encoding the Rice parameter, either 4 or 5 bits.

5. Format principles

FLAC has no format version information, but it does contain reserved space in several places. Future versions of the format MAY use this reserved space safely without breaking the format of older streams. Older decoders MAY choose to abort decoding when encountering data encoded using methods they do not recognize. Apart from reserved patterns, the format specifies forbidden patterns in certain places, meaning that the patterns MUST NOT appear in any bitstream. They are listed in the following table.

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

Table 1

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

All samples encoded to and decoded from the FLAC format MUST be in a signed representation.

There are several ways to convert unsigned sample representations to signed sample representations, but the coding methods provided by the FLAC format work best on audio signals of which the numerical values of the samples are centered around zero, i.e., have no DC offset. In most unsigned audio formats, signals are centered around halfway the range of the unsigned integer type used. If that is the case, converting sample representations by first copying the number to a signed integer with sufficient range and then subtracting half of the range of the unsigned integer type, results in a signal with samples centered around 0.

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

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

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

6. Format layout overview

A FLAC bitstream consists of the fLaC (i.e., 0x664C6143) marker at the beginning of the stream, followed by a mandatory metadata block (called the STREAMINFO block), any number of other metadata blocks, and then the audio frames.

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

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

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

Furthermore, to allow a decoder to start decoding at any place in the stream even without having received a streaminfo metadata block, each frame header contains some basic information about the stream. This information includes sample rate, bits per sample, number of channels, etc. Since the frame header is overhead, it has a direct effect on the compression ratio. To keep the frame header as small as possible, FLAC uses lookup tables for the most commonly used values for frame properties. When a certain property has a value that is not covered by the lookup table, the decoder is directed to find the value of that property (for example, the sample rate) at the end of the frame header or in the streaminfo metadata block. If a frame header refers to the streaminfo metadata block, the file is not 'streamable', see Section 7 for details. By using lookup tables, the file is streamable and the frame header size small for the most common forms of audio data.

Individual subframes (one for each channel) are coded separately within a frame, and appear serially in the stream. In other words, the encoded audio data is NOT channel-interleaved. This reduces decoder complexity at the cost of requiring larger decode buffers. Each subframe has its own header specifying the attributes of the subframe, like prediction method and order, residual coding parameters, etc. Each subframe header is followed by the encoded audio data for that channel.

7. Streamable subset

The FLAC format specifies a subset of itself as the FLAC streamable subset. The purpose of this is to ensure that any streams encoded according to this subset are truly "streamable", meaning that a decoder that cannot seek within the stream can still pick up in the middle of the stream and start decoding. It also makes hardware decoder implementations more practical by limiting the encoding parameters in such a way that decoder buffer sizes and other resource requirements can be easily determined. The streamable subset makes the following limitations on what MAY be used in the stream:

* The sample rate bits (see Section 9.1.2) in the frame header MUST be 0b0001-0b1110, i.e., the frame header MUST NOT refer to the streaminfo metadata block to describe the sample rate.

* The bit depth bits (see Section 9.1.4) in the frame header MUST be 0b001-0b111, i.e., the frame header MUST NOT refer to the streaminfo metadata block to describe the bit depth.
* The stream MUST NOT contain blocks with more than 16384 interchannel samples, i.e., the maximum block size must not be larger than 16384.
* Audio with a sample rate less than or equal to 48000 Hz MUST NOT be contained in blocks with more than 4608 interchannel samples, i.e., the maximum block size used for this audio must not be larger than 4608.
* Linear prediction subframes (see Section 9.2.6) containing audio with a sample rate less than or equal to 48000 Hz MUST have a predictor order less than or equal to 12, i.e., the subframe type bits in the subframe header (see Section 9.2.1) MUST NOT be 0b101100-0b111111.
* The Rice partition order (see Section 9.2.7) MUST be less than or equal to 8.
* The channel ordering MUST be equal to one defined in Section 9.1.3, i.e., the FLAC file MUST NOT need a WAVEFORMATEXTENSIBLE_CHANNEL_MASK tag to describe the channel ordering. See Section 8.6.2 for details.

8. File-level metadata

At the start of a FLAC file or stream, following the fLaC ASCII file signature, one or more metadata blocks MUST be present before any audio frames appear. The first metadata block MUST be a streaminfo block.

8.1. Metadata block header

Each metadata block starts with a 4 byte header. The first bit in this header flags whether a metadata block is the last one: it is a 0 when other metadata blocks follow, otherwise it is a 1. The 7 remaining bits of the first header byte contain the type of the metadata block as an unsigned number between 0 and 126 according to the following table. A value of 127 (i.e., 0b1111111) is forbidden. The three bytes that follow code for the size of the metadata block in bytes, excluding the 4 header bytes, as an unsigned number coded big-endian.

Value	Metadata block type
0	Streaminfo
1	Padding
2	Application
3	Seektable
4	Vorbis comment
5	Cuesheet
6	Picture
7 - 126	reserved
127	forbidden, to avoid confusion with a frame sync code

Table 2

8.2. Streaminfo

The streaminfo metadata block has information about the whole stream, like sample rate, number of channels, total number of samples, etc. It MUST be present as the first metadata block in the stream. Other metadata blocks MAY follow. There MUST be no more than one streaminfo metadata block per FLAC stream.

If the streaminfo metadata block contains incorrect or incomplete information, decoder behavior is left unspecified (i.e., up to the decoder implementation). A decoder MAY choose to stop further decoding when the information supplied by the streaminfo metadata block turns out to be incorrect or contains forbidden values. A decoder accepting information from the streaminfo block (most-significantly the maximum frame size, maximum block size, number of audio channels, number of bits per sample, and total number of samples) without doing further checks during decoding of audio frames could be vulnerable to buffer overflows. See also Section 12.

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

Data	Description
u(16)	The minimum block size (in samples) used in the stream, excluding the last block.
u(16)	The maximum block size (in samples) used in the stream.
u(24)	The minimum frame size (in bytes) used in the stream. A value of 0 signifies that the value is not known.
u(24)	The maximum frame size (in bytes) used in the stream. A value of 0 signifies that the value is not known.
u(20)	Sample rate in Hz.
u(3)	(number of channels)-1. FLAC supports from 1 to 8 channels.
u(5)	(bits per sample)-1. FLAC supports from 4 to 32 bits per sample.
u(36)	Total number of interchannel samples in the stream. A value of zero here means the number of total samples is unknown.
u(128)	MD5 checksum of the unencoded audio data. This allows the decoder to determine if an error exists in the audio data even when, despite the error, the bitstream itself is valid. A value of 0 signifies that the value is not known.

Table 3

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

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

If the minimum block size is equal to the maximum block size, the file contains a fixed block size stream, as the minimum block size excludes the last block. Note that in the case of a stream with a variable block size, the actual maximum block size MAY be smaller than the maximum block size listed in the streaminfo block, and the actual smallest block size excluding the last block MAY be larger than the minimum block size listed in the streaminfo block. This is because the encoder has to write these fields before receiving any input audio data, and cannot know beforehand what block sizes it will use, only between what bounds these will be chosen.

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

The MD5 checksum is computed by applying the MD5 message-digest algorithm in [RFC1321]. The message to this algorithm consists of all the samples of all channels interleaved, represented in signed, little-endian form. This interleaving is on a per-sample basis, so for a stereo file this means first the first sample of the first channel, then the first sample of the second channel, then the second sample of the first channel etc. Before computing the checksum, all samples must be byte-aligned. If the bit depth is not a whole number of bytes, the value of each sample is sign extended to the next whole number of bytes.

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

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

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

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

8.3. Padding

The padding metadata block allows for an arbitrary amount of padding. This block is useful when it is known that metadata will be edited after encoding; the user can instruct the encoder to reserve a padding block of sufficient size so that when metadata is added, it will simply overwrite the padding (which is relatively quick) instead of having to insert it into the existing file (which would normally require rewriting the entire file). There MAY be one or more padding metadata blocks per FLAC stream.

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

Table 4

8.4. Application

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

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

Table 5

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

8.5. Seektable

The seektable metadata block can be used to store seek points. It is possible to seek to any given sample in a FLAC stream without a seek table, but the delay can be unpredictable since the bitrate may vary widely within a stream. By adding seek points to a stream, this delay can be significantly reduced. There MUST NOT be more than one seektable metadata block in a stream, but the table can have any number of seek points.

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

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

Table 6

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

8.5.1. Seekpoint

Data	Description
u(64)	Sample number of the first sample in the target frame, or 0xFFFFFFFFFFFFFFFF for a placeholder point.
u(64)	Offset (in bytes) from the first byte of the first frame header to the first byte of the target frame's header.
u(16)	Number of samples in the target frame.

Table 7

NOTES

* For placeholder points, the second and third field values are undefined.
* Seek points within a table MUST be sorted in ascending order by sample number.
* Seek points within a table MUST be unique by sample number, with the exception of placeholder points.
* The previous two notes imply that there MAY be any number of placeholder points, but they MUST all occur at the end of the table.
* The sample offsets are those of an unmuxed FLAC stream. The offsets MUST NOT be updated on muxing to reflect the new offsets of FLAC frames in a container.

8.6. Vorbis comment

A Vorbis comment metadata block contains human-readable information coded in UTF-8. The name Vorbis comment points to the fact that the Vorbis codec stores such metadata in almost the same way, see [Vorbis]. A Vorbis comment metadata block consists of a vendor string optionally followed by a number of fields, which are pairs of field names and field contents. Many users refer to these fields as

FLAC tags or simply as tags. A FLAC file MUST NOT contain more than one Vorbis comment metadata block.

In a Vorbis comment metadata block, the metadata block header is directly followed by 4 bytes containing the length in bytes of the vendor string as an unsigned number coded little-endian. The vendor string follows UTF-8 coded, and is not terminated in any way.

Following the vendor string are 4 bytes containing the number of fields that are in the Vorbis comment block, stored as an unsigned number, coded little-endian. If this number is non-zero, it is followed by the fields themselves, each of which is stored with a 4 byte length. First, the 4 byte field length in bytes is stored as an unsigned number, coded little-endian. The field itself is, like the vendor string, UTF-8 coded, not terminated in any way.

Each field consists of a field name and a field content, separated by an = character. The field name MUST only consist of UTF-8 code points U+0020 through U+007E, excluding U+003D, which is the = character. In other words, the field name can contain all printable ASCII characters except the equals sign. The evaluation of the field names MUST be case insensitive, so U+0041 through 0+005A (A-Z) MUST be considered equivalent to U+0061 through U+007A (a-z) respectively. The field contents can contain any UTF-8 character.

Note that the Vorbis comment as used in Vorbis allows for on the order of 2^64 bytes of data whereas the FLAC metadata block is limited to 2^24 bytes. Given the stated purpose of Vorbis comments, i.e., human-readable textual information, the FLAC metadata block limit is unlikely to be restrictive. Also note that the 32-bit field lengths are coded little-endian, as opposed to the usual big-endian coding of fixed-length integers in the rest of the FLAC format.

8.6.1. Standard field names

Only one standard field name is defined: the channel mask field, in Section 8.6.2. No other field names are defined because the applicability of any field name is strongly tied to the content it is associated with. For example, field names useful for describing files that contain a single work of music would be unusable when labeling archived broadcasts, recordings of any kind, or a collection of music works. Even when describing a single work of music, different conventions exist depending on the kind of music: orchestral music differs from music by solo artists or bands.

Despite the fact that no field names are formally defined, there is a general trend among devices and software capable of FLAC playback that are meant to play music. Most of those recognize at least the following field names:

* Title: name of the current work.
* Artist: name of the artist generally responsible for the current work. For orchestral works, this is usually the composer; otherwise, it is often the performer.
* Album: name of the collection the current work belongs to.

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

8.6.2. Channel mask

Besides fields containing information about the work itself, one field is defined for technical reasons, of which the field name is WAVEFORMATEXTENSIBLE_CHANNEL_MASK. This field is used to communicate that the channels in a file differ from the default channels defined in Section 9.1.3. For example, by default, a FLAC file containing two channels is interpreted to contain a left and right channel, but with this field, it is possible to describe different channel contents.

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

Bit number	Channel description
0	Front left
1	Front right
2	Front center
3	Low-frequency effects (LFE)
4	Back left
5	Back right
6	Front left of center
7	Front right of center
8	Back center
9	Side left
10	Side right
11	Top center
12	Top front left
13	Top front center
14	Top front right
15	Top rear left
16	Top rear center
17	Top rear right

Table 8

Following are three examples:

* If a file has a single channel, being a LFE channel, the Vorbis comment field is WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x8.

* If a file has four channels, being front left, front right, top front left, and top front right, the Vorbis comment field is WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x5003.
* If an input has four channels, being back center, top front center, front center, and top rear center in that order, they have to be reordered to front center, back center, top front center and top rear center. The Vorbis comment field added is WAVEFORMATEXTENSIBLE_CHANNEL_MASK=0x12104.

WAVEFORMATEXTENSIBLE_CHANNEL_MASK fields MAY be padded with zeros, for example, 0x0008 for a single LFE channel. Parsing of WAVEFORMATEXTENSIBLE_CHANNEL_MASK fields MUST be case-insensitive for both the field name and the field contents.

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

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

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

8.7. Cuesheet

To either store the track and index point structure of a Compact Disc Digital Audio (CD-DA) along with its audio or to provide a mechanism to store locations of interest within a FLAC file, a cuesheet metadata block can be used. Certain aspects of this metadata block follow directly from the CD-DA specification, called Red Book, which is standardized as [IEC.60908.1999]. The description below is complete and further reference to [IEC.60908.1999] is not needed to implement this metadata block.

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

Data	Description
u(128*8)	Media catalog number, in ASCII printable characters 0x20-0x7E.
u(64)	Number of lead-in samples.
u(1)	1 if the cuesheet corresponds to a CD-DA, else 0.
u(7+258*8)	Reserved. All bits MUST be set to zero.
u(8)	Number of tracks in this cuesheet.
Cuesheet tracks	A number of structures as specified in Section 8.7.1 equal to the number of tracks specified previously.

Table 9

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

The number of lead-in samples has meaning only for CD-DA cuesheets; for other uses, it should be 0. For CD-DA, the lead-in is the TRACK 00 area where the table of contents is stored; more precisely, it is the number of samples from the first sample of the media to the first sample of the first index point of the first track. According to [IEC.60908.1999], the lead-in MUST be silence and CD grabbing software does not usually store it; additionally, the lead-in MUST be at least two seconds but MAY be longer. For these reasons, the lead-in length is stored here so that the absolute position of the first track can be computed. Note that the lead-in stored here is the number of samples up to the first index point of the first track, not necessarily to INDEX 01 of the first track; even the first track MAY have INDEX 00 data.

The number of tracks MUST be at least 1, as a cuesheet block MUST have a lead-out track. For CD-DA, this number MUST be no more than 100 (99 regular tracks and one lead-out track). The lead-out track is always the last track in the cuesheet. For CD-DA, the lead-out track number MUST be 170 as specified by [IEC.60908.1999], otherwise it MUST be 255.

8.7.1. Cuesheet track

Data	Description
u(64)	Track offset of the first index point in samples, relative to the beginning of the FLAC audio stream.
u(8)	Track number.
u(12*8)	Track ISRC.
u(1)	The track type: 0 for audio, 1 for non-audio. This corresponds to the CD-DA Q-channel control bit 3.
u(1)	The pre-emphasis flag: 0 for no pre-emphasis, 1 for pre-emphasis. This corresponds to the CD-DA Q-channel control bit 5.
u(6+13*8)	Reserved. All bits MUST be set to zero.
u(8)	The number of track index points.
Cuesheet track index points	For all tracks except the lead-out track, a number of structures as specified in Section 8.7.1.1 equal to the number of index points specified previously.

Table 10

Note that the track offset differs from the one in CD-DA, where the track's offset in the TOC is that of the track's INDEX 01 even if there is an INDEX 00. For CD-DA, the track offset MUST be evenly divisible by 588 samples (588 samples = 44100 samples/s * 1/75 s).

A track number of 0 is not allowed, because the CD-DA specification reserves this for the lead-in. For CD-DA the number MUST be 1-99, or 170 for the lead-out; for non-CD-DA, the track number MUST be 255 for the lead-out. It is recommended to start with track 1 and increase sequentially. Track numbers MUST be unique within a cuesheet.

The track ISRC (International Standard Recording Code) is a 12-digit alphanumeric code; see [ISRC-handbook]. A value of 12 ASCII 0x00 characters MAY be used to denote the absence of an ISRC.

There MUST be at least one index point in every track in a cuesheet except for the lead-out track, which MUST have zero. For CD-DA, the number of index points MUST NOT be more than 100.

8.7.1.1. Cuesheet track index point

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

Table 11

For CD-DA, the track index point offset MUST be evenly divisible by 588 samples (588 samples = 44100 samples/s * 1/75 s). Note that the offset is from the beginning of the track, not the beginning of the audio data.

For CD-DA, a track index point number of 0 corresponds to the track pre-gap. The first index point in a track MUST have a number of 0 or 1, and subsequently, index point numbers MUST increase by 1. Index point numbers MUST be unique within a track.

8.8. Picture

The picture metadata block contains image data of a picture in some way belonging to the audio contained in the FLAC file. Its format is derived from the APIC frame in the ID3v2 specification, see [ID3v2]. However, contrary to the APIC frame in ID3v2, the media type and description are prepended with a 4-byte length field instead of being 0x00 delimited strings. A FLAC file MAY contain one or more picture

metadata blocks.

Note that while the length fields for media type, description, and picture data are 4 bytes in length and could in theory code for a size up to 4 GiB, the total metadata block size cannot exceed what can be described by the metadata block header, i.e., 16 MiB.

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

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

Data	Description
u(32)	The picture type according to next table
u(32)	The length of the media type string in bytes.
u(n*8)	The media type string as specified by [RFC2046], or the text string --> to signify that the data part is a URI of the picture instead of the picture data itself. This field must be in printable ASCII characters 0x20-0x7E.
u(32)	The length of the description string in bytes.
u(n*8)	The description of the picture, in UTF-8.
u(32)	The width of the picture in pixels.
u(32)	The height of the picture in pixels.
u(32)	The color depth of the picture in bits per pixel.
u(32)	For indexed-color pictures (e.g., GIF), the number of colors used, or 0 for non-indexed pictures.
u(32)	The length of the picture data in bytes.
u(n*8)	The binary picture data.

Table 12

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

The following table contains all the defined picture types. Values other than those listed in the table are reserved. There MAY only be one each of picture types 1 and 2 in a file. In general practice, many FLAC playback devices and software display the contents of a picture metadata block with picture type 3 (front cover) during playback, if present.

Value	Picture type
0	Other
1	PNG file icon of 32x32 pixels, see [RFC2083]
2	General file icon
3	Front cover
4	Back cover
5	Liner notes page
6	Media label (e.g., CD, Vinyl or Cassette label)
7	Lead artist, lead performer, or soloist
8	Artist or performer
9	Conductor
10	Band or orchestra
11	Composer
12	Lyricist or text writer
13	Recording location
14	During recording
15	During performance
16	Movie or video screen capture
17	A bright colored fish
18	Illustration
19	Band or artist logotype
20	Publisher or studio logotype

Table 13

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

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

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

9. Frame structure

Directly after the last metadata block, one or more frames follow. Each frame consists of a frame header, one or more subframes, padding zero bits to achieve byte-alignment, and a frame footer. The number of subframes in each frame is equal to the number of audio channels.

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

Note that storing audio with changing audio properties in FLAC results in various practical problems. For example, these changes of audio properties must happen on a frame boundary, or the process will not be lossless. When a variable block size is chosen to accommodate this, note that blocks smaller than 16 samples are not allowed and it is therefore not possible to store an audio stream in which these properties change within 16 samples of the last change or the start of the file. Also, since the streaminfo metadata block can only accommodate a single set of properties, it is only valid for part of such an audio stream. Instead, it is RECOMMENDED to store an audio stream with changing properties in FLAC encapsulated in a container capable of handling such changes, as these do not suffer from the mentioned limitations. See Section 10 for details.

9.1. Frame header

Each frame MUST start on a byte boundary and starts with the 15-bit frame sync code 0b111111111111100. Following the sync code is the blocking strategy bit, which MUST NOT change during the audio stream. The blocking strategy bit is 0 for a fixed block size stream or 1 for a variable block size stream. If the blocking strategy is known, a decoder can include this bit when searching for the start of a frame to reduce the possibility of encountering a false positive, as the first two bytes of a frame are either 0xFFF8 for a fixed block size stream or 0xFFF9 for a variable block size stream.

9.1.1. Block size bits

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

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

Table 14

9.1.2. Sample rate bits

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

Value	Sample rate
0b0000	sample rate only stored in the streaminfo metadata block
0b0001	88.2 kHz
0b0010	176.4 kHz
0b0011	192 kHz
0b0100	8 kHz
0b0101	16 kHz
0b0110	22.05 kHz
0b0111	24 kHz
0b1000	32 kHz
0b1001	44.1 kHz
0b1010	48 kHz
0b1011	96 kHz
0b1100	uncommon sample rate in kHz stored as an 8-bit number
0b1101	uncommon sample rate in Hz stored as a 16-bit number
0b1110	uncommon sample rate in Hz divided by 10, stored as a 16-bit number
0b1111	forbidden

Table 15

9.1.3. Channels bits

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

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

Value	Channels
0b0000	1 channel: mono
0b0001	2 channels: left, right
0b0010	3 channels: left, right, center
0b0011	4 channels: front left, front right, back left, back right
0b0100	5 channels: front left, front right, front center, back/surround left, back/surround right
0b0101	6 channels: front left, front right, front center, LFE, back/surround left, back/surround right
0b0110	7 channels: front left, front right, front center, LFE, back center, side left, side right
0b0111	8 channels: front left, front right, front center, LFE, back left, back right, side left, side right
0b1000	2 channels, left, right, stored as left/side stereo
0b1001	2 channels, left, right, stored as right/side stereo
0b1010	2 channels, left, right, stored as mid/side stereo
0b1011 - 0b1111	reserved

Table 16

9.1.4. Bit depth bits

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

Value	Bit depth
0b000	bit depth only stored in the streaminfo metadata block
0b001	8 bits per sample
0b010	12 bits per sample
0b011	reserved
0b100	16 bits per sample
0b101	20 bits per sample
0b110	24 bits per sample
0b111	32 bits per sample

Table 17

The next bit is reserved and MUST be zero.

9.1.5. Coded number

Following the reserved bit (starting at the fifth byte of the frame) is either a sample or a frame number, which will be referred to as the coded number. When dealing with variable block size streams, the sample number of the first sample in the frame is encoded. When the file contains a fixed block size stream, the frame number is encoded. See Section 9.1 on the blocking strategy bit which signals whether a stream is a fixed block size stream or a variable block size stream. Also see Appendix B.1.

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

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

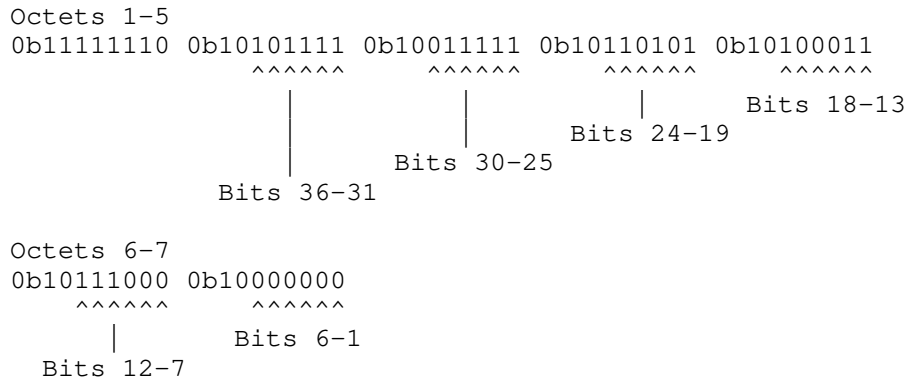
Furthermore, while UTF-8 is specifically used to encode characters, FLAC uses it to encode numbers instead. To encode or decode a coded number, follow the procedures of Section 3 of [RFC3629], but instead of using a character number, use a frame or sample number, and instead of the table in Section 3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

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

Table 18

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

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



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

9.1.6. Uncommon block size

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

9.1.7. Uncommon sample rate

Following the uncommon block size (or the coded number if no uncommon block size is stored) is the sample rate, if the sample rate bits were 0b1100, 0b1101, or 0b1110 (uncommon sample rate stored), as either an 8-bit or a 16-bit unsigned number coded big-endian.

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

9.1.8. Frame header CRC

Finally, after either the frame/sample number, an uncommon block size, or an uncommon sample rate, depending on whether the latter two are stored, is an 8-bit CRC. This CRC is initialized with 0 and has the polynomial x^8 + x^2 + x^1 + x^0. This CRC covers the whole frame header before the CRC, including the sync code.

9.2. Subframes

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

9.2.1. Subframe header

Each subframe starts with a header. The first bit of the header MUST be 0, followed by 6 bits describing which subframe type is used according to the following table, where v is the value of the 6 bits as an unsigned number.

Value	Subframe type
0b000000	Constant subframe
0b000001	Verbatim subframe
0b000010 - 0b000111	reserved
0b001000 - 0b001100	Subframe with a fixed predictor of order v-8, i.e., 0, 1, 2, 3 or 4
0b001101 - 0b011111	reserved
0b100000 - 0b111111	Subframe with a linear predictor of order v-31, i.e., 1 through 32 (inclusive)

Table 19

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

9.2.2. Wasted bits per sample

Most uncompressed audio file formats can only store audio samples with a bit depth that is an integer number of bytes. Samples of which the bit depth is not an integer number of bytes are usually stored in such formats by padding them with least-significant zero bits to a bit depth that is an integer number of bytes. For example, shifting a 14-bit sample right by 2 pads it to a 16-bit sample, which then has two zero least-significant bits. In this specification, these least-significant zero bits are referred to as wasted bits per sample or simply wasted bits. They are wasted in the sense that they contain no information, but are stored anyway.

The FLAC format can optionally take advantage of these wasted bits by signaling their presence and coding the subframe without them. To do this, the wasted bits per sample flag in a subframe header is set to 0 and the number of wasted bits per sample (k) minus 1 follows the flag in an unary encoding. For example, if k is 3, 0b001 follows. If k = 0, the wasted bits per sample flag is 0 and no unary coded k follows. In this document, if a subframe header signals a certain number of wasted bits, it is said it 'uses' these wasted bits.

If a subframe uses wasted bits (i.e., k is not equal to 0), samples are coded ignoring k least-significant bits. For example, if a frame not employing stereo decorrelation specifies a sample size of 16 bits per sample in the frame header and k of a subframe is 3, samples in the subframe are coded as 13 bits per sample. For more details, see Section 9.2.3 on how the bit depth of a subframe is calculated. A decoder MUST add k least-significant zero bits by shifting left (padding) after decoding a subframe sample. If the frame has left/side, right/side, or mid/side stereo, a decoder MUST perform padding on the subframes before restoring the channels to left and right. The number of wasted bits per sample MUST be such that the resulting number of bits per sample (of which the calculation is explained in Section 9.2.3) is larger than zero.

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

9.2.3. Constant subframe

In a constant subframe, only a single sample is stored. This sample is stored as an integer number coded big-endian, signed two's complement. The number of bits used to store this sample depends on the bit depth of the current subframe. The bit depth of a subframe is equal to the bit depth as coded in the frame header (see Section 9.1.4), minus the number of used wasted bits coded in the subframe header (see Section 9.2.2). If a subframe is a side subframe (see Section 4.2), the bit depth of that subframe is increased by 1 bit.

9.2.4. Verbatim subframe

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

9.2.5. Fixed predictor subframe

Five different fixed predictors are defined in the following table, one for each prediction order 0 through 4. In the table is also a derivation, which explains the rationale for choosing these fixed predictors.

Order	Prediction	Derivation
0	0	N/A
1	a(n-1)	N/A
2	2 * a(n-1) - a(n-2)	a(n-1) + a'(n-1)
3	3 * a(n-1) - 3 * a(n-2) + a(n-3)	a(n-1) + a'(n-1) + a''(n-1)
4	4 * a(n-1) - 6 * a(n-2) + 4 * a(n-3) - a(n-4)	a(n-1) + a'(n-1) + a''(n-1) + a'''(n-1)

Table 20

Where

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

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

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

Table 21

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

To encode a signal with a fixed predictor, each sample has the corresponding prediction subtracted and sent to the residual coder. To decode a signal with a fixed predictor, the residual is decoded, and then the prediction can be added for each sample. This means that decoding is necessarily a sequential process within a subframe, as for each sample, enough fully decoded previous samples are needed to calculate the prediction.

For fixed predictor order 0, the prediction is always 0, thus each residual sample is equal to its corresponding input or decoded sample. The difference between a fixed predictor with order 0 and a verbatim subframe, is that a verbatim subframe stores all samples unencoded, while a fixed predictor with order 0 has all its samples processed by the residual coder.

The first order fixed predictor is comparable to how DPCM encoding works, as the resulting residual sample is the difference between the corresponding sample and the sample before it. The higher order fixed predictors can be understood as polynomials fitted to the previous samples.

9.2.6. Linear predictor subframe

Whereas fixed predictors are well suited for simple signals, using a (non-fixed) linear predictor on more complex signals can improve compression by making the residual samples even smaller. There is a certain trade-off however, as storing the predictor coefficients takes up space as well.

In the FLAC format, a predictor is defined by up to 32 predictor coefficients and a shift. To form a prediction, each coefficient is multiplied by its corresponding past sample, the results are summed,

and this sum is then shifted. To encode a signal with a linear predictor, each sample has the corresponding prediction subtracted and sent to the residual coder. To decode a signal with a linear predictor, the residual is decoded, and then the prediction can be added for each sample. This means that decoding MUST be a sequential process within a subframe, as for each sample, enough decoded samples are needed to calculate the prediction.

The table below defines how a linear predictor subframe appears in the bitstream.

Data	Description
s(n)	Unencoded warm-up samples (n = subframe's bits per sample * lpc order).
u(4)	(Predictor coefficient precision in bits)-1 (NOTE: 0b1111 is forbidden).
s(5)	Prediction right shift needed in bits.
s(n)	Predictor coefficients (n = predictor coefficient precision * lpc order).
Coded residual	Coded residual as defined in Section 9.2.7

Table 22

See Section 9.2.3 on how the warm-up samples are stored unencoded. The predictor coefficients are stored as an integer number coded big-endian, signed two's complement, where the number of bits needed for each coefficient is defined by the predictor coefficient precision. While the prediction right shift is signed two's complement, this number MUST NOT be negative, see Appendix B.4 for an explanation why this is.

Please note that the order in which the predictor coefficients appear in the bitstream corresponds to which *past* sample they belong to. In other words, the order of the predictor coefficients is opposite to the chronological order of the samples. So, the first predictor coefficient has to be multiplied with the sample directly before the sample that is being predicted, the second predictor coefficient has to be multiplied with the sample before that, etc.

9.2.7. Coded residual

The first two bits in a coded residual indicate which coding method is used. See the table below.

Value	Description
0b00	partitioned Rice code with 4-bit parameters
0b01	partitioned Rice code with 5-bit parameters
0b10 - 0b11	reserved

Table 23

Both defined coding methods work the same way, but differ in the number of bits used for Rice parameters. The 4 bits that directly follow the coding method bits form the partition order, which is an unsigned number. The rest of the coded residual consists of 2^(partition order) partitions. For example, if the 4 bits are 0b1000, the partition order is 8 and the residual is split up into 2^8 = 256 partitions.

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

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

Each partition starts with a parameter. If the coded residual of a subframe is one with 4-bit Rice parameters (see the table at the start of this section), the first 4 bits of each partition are either a Rice parameter or an escape code. These 4 bits indicate an escape code if they are 0b1111, otherwise they contain the Rice parameter as an unsigned number. If the coded residual of the current subframe is one with 5-bit Rice parameters, the first 5 bits of each partition indicate an escape code if they are 0b11111, otherwise, they contain the Rice parameter as an unsigned number as well.

9.2.7.1. Escaped partition

If an escape code was used, the partition does not contain a variable-length Rice coded residual, but a fixed-length unencoded residual. Directly following the escape code are 5 bits containing the number of bits with which each residual sample is stored, as an unsigned number. The residual samples themselves are stored signed two's complement. For example, when a partition is escaped and each residual sample is stored with 3 bits, the number -1 is represented as 0b111.

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

9.2.7.2. Rice code

If a Rice parameter was provided for a certain partition, that partition contains a Rice coded residual. The residual samples, which are signed numbers, are represented by unsigned numbers in the Rice code. For positive numbers, the representation is the number doubled, for negative numbers, the representation is the number multiplied by -2 and has 1 subtracted. This representation of signed numbers is also known as zigzag encoding. The zigzag encoded residual is called the folded residual.

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

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

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

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

Appendix D shows decoding of a complete coded residual.

9.2.7.3. Residual sample value limit

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

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

9.3. Frame footer

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

10. Container mappings

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

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

10.1. Ogg mapping

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

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

Table 24

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

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

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

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

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

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

10.2. Matroska mapping

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

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

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

10.3. ISO Base Media File Format (MP4) mapping

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

11. Implementation status

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

This section records the status of known implementations of the FLAC format, and is based on a proposal described in [RFC7942]. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

A reference encoder and decoder implementation of the FLAC format exists, known as libFLAC, maintained by Xiph.Org. It can be found at https://xiph.org/flac/ (https://xiph.org/flac/) Note that while all libFLAC components are licensed under 3-clause BSD, the flac and metaflac command line tools often supplied together with libFLAC are licensed under GPL.

Another completely independent implementation of both encoder and decoder of the FLAC format is available in libavcodec, maintained by FFmpeg, licensed under LGPL 2.1 or later. It can be found at https://ffmpeg.org/ (https://ffmpeg.org/)

A list of other implementations and an overview of which parts of the format they implement can be found at [FLAC-wiki-implementations].

12. Security Considerations

Like any other codec (such as [RFC6716]), FLAC should not be used with insecure ciphers or cipher modes that are vulnerable to known plaintext attacks. Some of the header bits as well as the padding are easily predictable.

Implementations of the FLAC codec need to take appropriate security considerations into account. Section 2.1 of [RFC4732] provides general information on DoS attacks on end-systems and describes some mitigation strategies. Areas of concern specific to FLAC follow.

It is extremely important for the decoder to be robust against malformed payloads. Payloads that do not conform to this specification MUST NOT cause the decoder to overrun its allocated memory or take an excessive amount of resources to decode. An overrun in allocated memory could lead to arbitrary code execution by an attacker. The same applies to the encoder, even though problems with encoders are typically rarer. Malformed audio streams MUST NOT cause the encoder to misbehave because this would allow an attacker to attack transcoding gateways.

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

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

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

Metadata blocks can also contain references, e.g., the picture metadata block can contain a URI. When following an URI, the security considerations of [RFC3986] apply. Applications MUST obtain explicit user approval to retrieve resources via remote protocols.

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

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

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

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

FLAC files may contain executable code, although the FLAC format is not designed for it and it is uncommon. One use case where FLAC is occasionally used to store executable code is when compressing images of mixed mode CDs, which contain both audio and non-audio data, of which the non-audio portion can contain executable code. In that case, the executable code is stored as if it were audio and is potentially obscured. Of course, it is also possible to store executable code as metadata, for example as a vorbis comment with help of a binary-to-text encoding or directly in an application metadata block. Applications MUST NOT execute code contained in FLAC files or present parts of FLAC files as executable code to the user,

except when an application has that explicit purpose, e.g., applications reading FLAC files as disc images and presenting it as virtual disc drive.

13. IANA Considerations

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

13.1. Media type registration

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

Type name: audio

Subtype name: flac

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: as per THISRFC

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

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

Published specification: THISRFC

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

Fragment identifier considerations: none

Additional information:

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

Magic number(s): fLaC

File extension(s): flac

Macintosh file type code(s): none

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

Windows Clipboard Format Name: audio/flac

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: IETF CELLAR WG

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

Provisional registration? (standards tree only): NO

13.2. Application ID Registry

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

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

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

Application ID	ASCII rendition (if available)	Description	Specification	Change controller

0x41544348	ATCH	FlacFile	[FlacFile]	IETF
0x42534F4C	BSOL	beSolo		IETF
0x42554753	BUGS	Bugs Player		IETF
0x43756573	Cues	GoldWave cue points		IETF
0x46696361	Fica	CUE Splitter		IETF
0x46746F6C	Ftol	flac-tools		IETF
0x4D4F5442	MOTB	MOTB MetaCzar		IETF
0x4D505345	MPSE	MP3 Stream Editor		IETF
0x4D754D4C	MuML	MusicML: Music Metadata Language		IETF
0x52494646	RIFF	Sound Devices RIFF chunk storage		IETF
0x5346464C	SFFL	Sound Font FLAC		IETF
0x534F4E59	SONY	Sony Creative Software		IETF
0x5351455A	SQEZ	flacsqueeze		IETF
0x54745776	TtWv	TwistedWave		IETF
0x55495453	UITS	UITS Embedding tools		IETF
0x61696666	aiff	FLAC AIFF chunk storage	[Foreign-metadata]	IETF

0x696D6167	imag	flac-image		IETF
0x7065656D	peem	Parseable Embedded Extensible Metadata		IETF
0x71667374	qfst	QFLAC Studio		IETF
0x72696666	riff	FLAC RIFF chunk storage	[Foreign-metadata]	IETF
0x74756E65	tune	TagTuner		IETF
0x773634C0	w64	FLAC Wave64 chunk storage	[Foreign-metadata]	IETF
0x78626174	xbat	XBAT		IETF
0x786D6364	xmcd	xmcd		IETF

Table 25

14. Acknowledgments

FLAC owes much to the many people who have advanced the audio compression field so freely. For instance:

* A. J. Robinson for his work on Shorten; his paper (see [robinson-tr156]) is a good starting point on some of the basic methods used by FLAC. FLAC trivially extends and improves the fixed predictors, LPC coefficient quantization, and Rice coding used in Shorten.
* S. W. Golomb and Robert F. Rice; their universal codes are used by FLAC's entropy coder, see [Rice].
* N. Levinson and J. Durbin; the FLAC reference encoder (see Section 11) uses an algorithm developed and refined by them for determining the LPC coefficients from the autocorrelation coefficients, see [Durbin].
* And of course, Claude Shannon, see [Shannon].

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

15. References

15.1. Normative References

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

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

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

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

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

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

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

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

15.2. Informative References

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

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

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

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

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

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

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

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

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

[HPL-1999-144]
Hans, M. and RW. Schafer, "Lossless Compression of Digital Audio", DOI 10.1109/79.939834, November 1999, <https://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf>.

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

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

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

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

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

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

[RFC4732] Handley, M., Ed., Rescorla, E., Ed., and IAB, "Internet Denial-of-Service Considerations", RFC 4732, DOI 10.17487/RFC4732, December 2006, <https://www.rfc-editor.org/info/rfc4732>.

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

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <https://www.rfc-editor.org/info/rfc6716>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

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

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

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

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

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

[robinson-tr156] Robinson, T., "SHORTEN: Simple lossless and near-lossless waveform compression", December 1994, <https://mi.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html>.

Appendix A. Numerical considerations

In order to maintain lossless behavior, all arithmetic used in encoding and decoding sample values must be done with integer data types to eliminate the possibility of introducing rounding errors associated with floating-point arithmetic. Use of floating-point representations in analysis (e.g., finding a good predictor or Rice parameter) is not a concern, as long as the process of using the found predictor and Rice parameter to encode audio samples is implemented with only integer math.

Furthermore, the possibility of integer overflow can be eliminated by using large enough data types. Choosing a 64-bit signed data type for all arithmetic involving sample values would make sure the possibility for overflow is eliminated, but usually smaller data types are chosen for increased performance, especially in embedded devices. This appendix provides guidelines for choosing the appropriate data type for each step of encoding and decoding FLAC files.

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

A.1. Determining the necessary data type size

To find the smallest data type size that is guaranteed not to overflow for a certain sequence of arithmetic operations, the combination of values producing the largest possible result should be considered.

If, for example, two 16-bit signed integers are added, the largest possible result forms if both values are the largest number that can be represented with a 16-bit signed integer. To store the result, a signed integer data type with at least 17 bits is needed. Similarly, when adding 4 of these values, 18 bits are needed; when adding 8, 19 bits are needed, etc. In general, the number of bits necessary when adding numbers together is increased by the log base 2 of the number of values rounded up to the nearest integer. So, when adding 18 unknown values stored in 8 bit signed integers, we need a signed integer data type of at least 13 bits to store the result, as the log base 2 of 18 rounded up is 5.

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

A.2. Stereo decorrelation

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

This means that while 16-bit signed integers have sufficient range to store samples from a fully decoded FLAC frame with a bit depth of 16 bits, the decoding of a side subframe in such a file will need a data type with at least 17 bits to store decoded subframe samples before undoing stereo decorrelation.

Most FLAC decoders store decoded (subframe) samples as 32-bit values, which is sufficient for files with bit depths up to (and including) 31 bits.

A.3. Prediction

A prediction (which is used to calculate the residual on encoding or added to the residual to calculate the sample value on decoding) is formed by multiplying and summing preceding sample values. In order to eliminate the possibility of integer overflow, the combination of preceding sample values and predictor coefficients producing the largest possible value should be considered.

To determine the size of the data type needed to calculate either a residual sample (on encoding) or an audio sample value (on decoding) in a fixed predictor subframe, the maximal possible value for these is calculated as described in Appendix A.1 in the following table. For example: if a frame codes for 16-bit audio and has some form of stereo decorrelation, the subframe coding for the side channel would need 16+1+3 bits if a third order fixed predictor is used.

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

Table 26

Where

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

For subframes with a linear predictor, the calculation is a little more complicated. Each prediction is the sum of several multiplications. Each of these multiply a sample value with a predictor coefficient. The extra bits needed can be calculated by adding the predictor coefficient precision (in bits) to the bit depth of the audio samples. To account for the summing of these multiplications, the log base 2 of the predictor order rounded up is added.

For example, if the sample bit depth of the source is 24, the current subframe encodes a side channel (see Section 4.2), the predictor order is 12, and the predictor coefficient precision is 15 bits, the minimum required size of the used signed integer data type is at least (24 + 1) + 15 + ceil(log2(12)) = 44 bits. As another example, with a side-channel subframe bit depth of 16, a predictor order of 8, and a predictor coefficient precision of 12 bits, the minimum required size of the used signed integer data type is (16 + 1) + 12 + ceil(log2(8)) = 32 bits.

A.4. Residual

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

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

Taking the last example of the previous section, where 32 bits were needed for the prediction, the required data type size for the residual samples in case of a right shift of 10 bits would be 32 - 10 + 1 = 23 bits, which means it is not necessary to perform the aforementioned check.

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

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

A.5. Rice coding

When folding (i.e., zig-zag encoding) the residual sample values, no extra bits are needed when the absolute value of each residual sample is first stored in an unsigned data type of the size of the last step, then doubled, and then has one subtracted depending on whether the residual sample was positive or negative. Many implementations, however, choose to require one extra bit of data type size so zig-zag encoding can happen in one step and without a cast instead of the procedure described in the previous sentence.

Appendix B. Past format changes

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

The FLAC format was first specified in December 2000 and the bitstream format was considered frozen with the release of FLAC (the reference encoder/decoder) 1.0 in July 2001. Only changes made since this first stable release are considered in this appendix. Changes made to the FLAC streamable subset definition (see Section 7) are not considered.

B.1. Addition of blocking strategy bit

Perhaps the largest backwards incompatible change to the specification was published in July 2007. Before this change, variable block size streams were not explicitly marked as such by a flag bit in the frame header. A decoder had two ways to detect a variable block size stream, either by comparing the minimum and maximum block size in the STREAMINFO metadata block (which are equal for a fixed block size stream), or, if a decoder did not receive a STREAMINFO metadata block, by detecting a change of block size during

a stream, which could in theory not happen at all. As the meaning of the coded number in the frame header depends on whether or not a stream is variable block size, this presented a problem: the meaning of the coded number could not be reliably determined. To fix this problem, one of the reserved bits was changed to be used as a blocking strategy bit. See also Section 9.1.

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

B.2. Restriction of encoded residual samples

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

B.3. Addition of 5-bit Rice parameters

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

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

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

Appendix C. Interoperability considerations

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

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

C.1. Features outside of the streamable subset

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

C.2. Variable block size

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

streams.

C.3. 5-bit Rice parameter

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

C.4. Rice escape code

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

C.5. Uncommon block size

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

C.6. Uncommon bit depth

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

* DVD-Audio has the possibility to store 20 bit PCM audio.
* DAT and DV can store 12 bit PCM audio.
* NICAM-728 samples at 14 bit, which is companded to 10 bit.
* 8-bit μ-law can be losslessly converted to 14 bit (Linear) PCM.
* 8-bit A-law can be losslessly converted to 13 bit (Linear) PCM.

The FLAC format can contain these bit depths directly, but because they are uncommon, some decoders are not able to process the resulting files correctly. It is possible to store these formats in a FLAC file with a more common bit depth without sacrificing

compression by padding each sample with zero bits to a bit depth that is a whole byte. The FLAC format can efficiently compress these wasted bits. See Section 9.2.2 for details.

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

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

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

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

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

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

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

C.8. Changing audio properties mid-stream

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

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

Appendix D. Examples

This informational appendix contains short example FLAC files that are decoded step by step. These examples provide a more engaging way to understand the FLAC format than the formal specification. The text explaining these examples assumes the reader has at least cursorily read the specification and that the reader refers to the specification for explanation of the terminology used. These examples mostly focus on the layout of several metadata blocks and subframe types and the implications of certain aspects (for example, wasted bits and stereo decorrelation) on this layout.

The examples feature files generated by various FLAC encoders. These are presented in hexadecimal or binary format, followed by tables and text referring to various features by their starting bit positions in these representations. Each starting position (shortened to 'start' in the tables) is a hexadecimal byte position and a start bit within that byte, separated by a plus sign. Counts for these start at zero. For example, a feature starting at the 3rd bit of the 17th byte is referred to as starting at 0x10+2. The files that are explored in these examples can be found at [FLAC-specification-github].

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

D.1. Decoding example 1

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

D.1.1. Example file 1 in hexadecimal representation

```
00000000: 664c 6143 8000 0022 1000 1000  fLaC..."....
0000000c: 0000 0f00 000f 0ac4 42f0 0000  ........B...
00000018: 0001 3e84 b418 07dc 6903 0758  ..>.....i..X
00000024: 6a3d ad1a 2e0f fff8 6918 0000  j=......i...
00000030: bf03 58fd 0312 8baa 9a         ..X......
```

D.1.2. Example file 1 in binary representation

```
00000000: 01100110 01001100 01100001 01000011  fLaC
00000004: 10000000 00000000 00000000 00100010  ..."
00000008: 00010000 00000000 00010000 00000000  ....
0000000c: 00000000 00000000 00001111 00000000  ....
00000010: 00000000 00001111 00001010 11000100  ....
00000014: 01000010 11110000 00000000 00000000  B...
00000018: 00000000 00000001 00111110 10000100  ..>.
0000001c: 10110100 00011000 00000111 11011100  ....
00000020: 01101001 00000011 00000111 01011000  i..X
00000024: 01101010 00111101 10101101 00011010  j=..
00000028: 00101110 00001111 11111111 11111000  ....
0000002c: 01101001 00011000 00000000 00000000  i...
00000030: 10111111 00000011 01011000 11111101  ..X.
00000034: 00000011 00010010 10001011 10101010  ....
00000038: 10011010
```

D.1.3. Signature and streaminfo

The first 4 bytes of the file contain the fLaC file signature. Directly following it is a metadata block. The signature and the first metadata block header are broken down in the following table.

Start	Length	Contents	Description
0x00+0	4 bytes	0x664C6143	fLaC
0x04+0	1 bit	0b1	Last metadata block
0x04+1	7 bits	0b0000000	Streaminfo metadata block
0x05+0	3 bytes	0x000022	Length 34 byte

Table 27

As the header indicates that this is the last metadata block, the position of the first audio frame can now be calculated as the position of the first byte after the metadata block header + the length of the block, i.e., 8+34 = 42 or 0x2a. As can be seen, 0x2a indeed contains the frame sync code for fixed block size streams, 0xfff8.

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

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

Table 28

The minimum and maximum block size are both 4096. This was apparently the block size the encoder planned to use, but as only 1 interchannel sample was provided, no frames with 4096 samples are actually present in this file.

Note that anywhere a number of samples is mentioned (block size, total number of samples, sample rate), interchannel samples are meant.

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

D.1.4. Audio frames

The frame header starts at position 0x2a and is broken down in the following table.

Start	Length	Contents	Description
0x2a+0	15 bits	0xff, 0b1111100	frame sync
0x2b+7	1 bit	0b0	blocking strategy
0x2c+0	4 bits	0b0110	8-bit block size further down
0x2c+4	4 bits	0b1001	sample rate 44.1 kHz
0x2d+0	4 bits	0b0001	stereo, no decorrelation
0x2d+4	3 bits	0b100	bit depth 16 bit
0x2d+7	1 bit	0b0	mandatory 0 bit
0x2e+0	1 byte	0x00	frame number 0
0x2f+0	1 byte	0x00	block size 1
0x30+0	1 byte	0xbf	frame header CRC

Table 29

As the stream is a fixed block size stream, the number at 0x2e contains a frame number. As the value is smaller than 128, only 1 byte is used for the encoding.

At byte 0x31, the first subframe starts, which is broken down in the following table.

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

Table 30

As the wasted bits flag is 1 in this subframe, an unary coded number follows. Starting at 0x32, we see 0b01, which unary codes for 1, meaning this subframe uses 2 wasted bits.

As this is a verbatim subframe, the subframe only contains unencoded sample values. With a block size of 1, it contains only a single sample. The bit depth of the audio is 16 bits, but as the subframe header signals the use of 2 wasted bits, only 14 bits are stored. As no stereo decorrelation is used, a bit depth increase for the side channel is not applicable. So, the next 14 bits (starting at position 0x32+2) contain the unencoded sample coded big-endian, signed two's complement. The value reads 0b011000 11111101, or 6397. This value needs to be shifted left by 2 bits, to account for the wasted bits. The value is then 0b011000 11111101 00, or 25588.

The second subframe starts at 0x34, and is broken down in the following table.

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

Table 31

Here the wasted bits flag is also one, but the unary coded number that follows it is 4 bit long, indicating the use of 4 wasted bits. This means the sample is stored in 12 bits. The sample value is 0b0010 10001011, or 651. This value now has to be shifted left by 4 bits, i.e., 0b0010 10001011 0000 or 10416.

At this point, we would undo stereo decorrelation if that was applicable.

As the last subframe ends byte-aligned, no padding bits follow it. The next 2 bytes, starting at 0x38, contain the frame CRC. As this is the only frame in the file, the file ends with the CRC.

To validate the MD5 checksum, we line up the samples interleaved, byte-aligned, little endian, signed two's complement. The first sample, with value 25588, translates to 0xf463, the second sample, with value 10416, translates to 0xb028. When computing the MD5 checksum with 0xf463b028 as input, we get the MD5 checksum found in the header, so decoding was lossless.

D.2. Decoding example 2

This FLAC file is larger than the first example, but still contains very little audio. The focus of this example is on decoding a subframe with a fixed predictor and a coded residual, but it also contains a very short seektable, a Vorbis comment metadata block, and a padding metadata block.

D.2.1. Example file 2 in hexadecimal representation

```
00000000: 664c 6143 0000 0022 0010 0010  fLaC..."....
0000000c: 0000 1700 0044 0ac4 42f0 0000  .....D..B...
00000018: 0013 d5b0 5649 75e9 8b8d 8b93  ....VIu.....
00000024: 0422 757b 8103 0300 0012 0000  ."u{........
00000030: 0000 0000 0000 0000 0000 0000  ............
0000003c: 0000 0010 0400 003a 2000 0000  .......: ...
00000048: 7265 6665 7265 6e63 6520 6c69  reference li
00000054: 6246 4c41 4320 312e 332e 3320  bFLAC 1.3.3
00000060: 3230 3139 3038 3034 0100 0000  20190804....
0000006c: 0e00 0000 5449 544c 453d d7a9  ....TITLE=..
00000078: d79c d795 d79d 8100 0006 0000  ............
00000084: 0000 0000 fff8 6998 000f 9912  ......i.....
00000090: 0867 0162 3d14 4299 8f5d f70d  .g.b=.B..]..
0000009c: 6fe0 0c17 caeb 2100 0ee7 a77a  o.....!....z
000000a8: 24a1 590c 1217 b603 097b 784f  $.Y......{xO
000000b4: aa9a 33d2 85e0 70ad 5b1b 4851  ..3...p.[.HQ
000000c0: b401 0d99 d2cd 1a68 f1e6 b810  .......h....
000000cc: fff8 6918 0102 a402 c382 c40b  ..i.........
000000d8: c14a 03ee 48dd 03b6 7c13 30    .J..H...|.0
```

D.2.2. Example file 2 in binary representation (only audio frames)

```
00000088: 11111111 11111000 01101001 10011000  ..i.
0000008c: 00000000 00001111 10011001 00010010  ....
00000090: 00001000 01100111 00000001 01100010  .g.b
00000094: 00111101 00010100 01000010 10011001  =.B.
00000098: 10001111 01011101 11110111 00001101  .]..
0000009c: 01101111 11100000 00001100 00010111  o...
000000a0: 11001010 11101011 00100001 00000000  ..!.
000000a4: 00001110 11100111 10100111 01111010  ...z
000000a8: 00100100 10100001 01011001 00001100  $.Y.
000000ac: 00010010 00010111 10110110 00000011  ....
000000b0: 00001001 01111011 01111000 01001111  .{xO
000000b4: 10101010 10011010 00110011 11010010  ..3.
000000b8: 10000101 11100000 01110000 10101101  ..p.
000000bc: 01011011 00011011 01001000 01010001  [.HQ
000000c0: 10110100 00000001 00001101 10011001  ....
000000c4: 11010010 11001101 00011010 01101000  ...h
000000c8: 11110001 11100110 10111000 00010000  ....
000000cc: 11111111 11111000 01101001 00011000  ..i.
000000d0: 00000001 00000010 10100100 00000010  ....
000000d4: 11000011 10000010 11000100 00001011  ....
000000d8: 11000001 01001010 00000011 11101110  .J..
000000dc: 01001000 11011101 00000011 10110110  H...
000000e0: 01111100 00010011 00110000           |.0
```

D.2.3. Streaminfo metadata block

Most of the streaminfo block, including its header, is the same as in example 1, so only parts that are different are listed in the following table.

Start	Length	Contents	Description
0x04+0	1 bit	0b0	Not the last metadata block
0x08+0	2 bytes	0x0010	Min. block size 16
0x0a+0	2 bytes	0x0010	Max. block size 16
0x0c+0	3 bytes	0x000017	Min. frame size 23 byte
0x0f+0	3 bytes	0x000044	Max. frame size 68 byte
0x15+4	36 bits	0b0000, 0x00000013	Total no. of samples 19
0x1a	16 bytes	(...)	MD5 checksum

Table 32

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

D.2.4. Seektable

The seektable metadata block only holds one entry. It is not really useful here, as it points to the first frame, but it is enough for this example. The seektable metadata block is broken down in the following table.

Start	Length	Contents	Description
0x2a+0	1 bit	0b0	Not the last metadata block
0x2a+1	7 bits	0b0000011	Seektable metadata block
0x2b+0	3 bytes	0x000012	Length 18 byte
0x2e+0	8 bytes	0x0000000000000000	Seekpoint to sample 0
0x36+0	8 bytes	0x0000000000000000	Seekpoint to offset 0
0x3e+0	2 bytes	0x0010	Seekpoint to block size 16

Table 33

D.2.5. Vorbis comment

The Vorbis comment metadata block contains the vendor string and a single comment. It is broken down in the following table.

Start	Length	Contents	Description
0x40+0	1 bit	0b0	Not the last metadata block
0x40+1	7 bits	0b0000100	Vorbis comment metadata block
0x41+0	3 bytes	0x00003a	Length 58 byte
0x44+0	4 bytes	0x20000000	Vendor string length 32 byte
0x48+0	32 bytes	(...)	Vendor string
0x68+0	4 bytes	0x01000000	Number of fields 1
0x6c+0	4 bytes	0x0e000000	Field length 14 byte
0x70+0	14 bytes	(...)	Field contents

Table 34

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

D.2.6. Padding

The last metadata block is a (very short) padding block.

Start	Length	Contents	Description
0x7e+0	1 bit	0b1	Last metadata block
0x7e+1	7 bits	0b0000001	Padding metadata block
0x7f+0	3 bytes	0x000006	Length 6 byte
0x82+0	6 bytes	0x000000000000	Padding bytes

Table 35

D.2.7. First audio frame

The frame header starts at position 0x88 and is broken down in the following table.

Start	Length	Contents	Description
0x88+0	15 bits	0xff, 0b1111100	frame sync
0x89+7	1 bit	0b0	blocking strategy
0x8a+0	4 bits	0b0110	8-bit block size further down
0x8a+4	4 bits	0b1001	sample rate 44.1 kHz
0x8b+0	4 bits	0b1001	side-right stereo
0x8b+4	3 bits	0b100	bit depth 16 bit
0x8b+7	1 bit	0b0	mandatory 0 bit
0x8c+0	1 byte	0x00	frame number 0
0x8d+0	1 byte	0x0f	block size 16
0x8e+0	1 byte	0x99	frame header CRC

Table 36

The first subframe starts at byte 0x8f, it is broken down in the following table excluding the coded residual. As this subframe codes for a side channel, the bit depth is increased by 1 bit from 16 bit to 17 bit. This is most clearly present in the unencoded warm-up sample.

Start	Length	Contents	Description
0x8f+0	1 bit	0b0	mandatory 0 bit
0x8f+1	6 bits	0b001001	fixed subframe, 1st order
0x8f+7	1 bit	0b0	no wasted bits used
0x90+0	17 bits	0x0867, 0b0	unencoded warm-up sample

Table 37

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

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

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

0xa9+0	1 bit	0b1	Quotient 0
0xa9+1	11 bits	0b01000010101	Remainder 533
0xaa+4	1 bit	0b1	Quotient 0
0xaa+5	11 bits	0b00100001100	Remainder 268

Table 38

At this point, the decoder should know it is done decoding the coded residual, as it received 16 samples: 1 warm-up sample and 15 residual samples. Each residual sample can be calculated from the quotient and remainder, and undoing the zig-zag encoding. For example, the value of the first zig-zag encoded residual sample is 3 * 2^11 + 244 = 6388. As this is an even number, the zig-zag encoding is undone by dividing by 2, the residual sample value is 3194. This is done for all residual samples in the next table.

Quotient	Remainder	Zig-zag encoded	Residual sample value
3	244	6388	3194
1	545	2593	-1297
1	408	2456	1228
0	1885	1885	-943
0	1904	1904	952
0	1391	1391	-696
0	1536	1536	768
0	1047	1047	-524
0	1198	1198	599
0	801	801	-401
12	1767	26343	-13172
0	631	631	-316
0	548	548	274
0	533	533	-267
0	268	268	134

Table 39

It can be calculated that using a Rice code is, in this case, more efficient than storing values unencoded. The Rice code (excluding the partition order and parameter) is 199 bits in length. The largest residual value (-13172) would need 15 bits to be stored unencoded, so storing all 15 samples with 15 bits results in a sequence with a length of 225 bits.

The next step is using the predictor and the residuals to restore the sample values. As this subframe uses a fixed predictor with order 1, this means adding the residual value to the value of the previous sample.

Residual	Sample value
(warm-up)	4302
3194	7496
-1297	6199
1228	7427
-943	6484
952	7436
-696	6740
768	7508
-524	6984
599	7583
-401	7182
-13172	-5990
-316	-6306
274	-6032
-267	-6299
134	-6165

Table 40

With this, the decoding of the first subframe is complete. The decoding of the second subframe is very similar, as it also uses a fixed predictor of order 1, so this is left as an exercise for the reader, the results are in the next table. The next step is undoing stereo decorrelation, which is done in the following table. As the stereo decorrelation is side-right, the samples in the right channel come directly from the second subframe, while the samples in the left channel are found by adding the values of both subframes for each sample.

Subframe 1	Subframe 2	Left	Right
4302	6070	10372	6070
7496	10545	18041	10545
6199	8743	14942	8743
7427	10449	17876	10449
6484	9143	15627	9143
7436	10463	17899	10463
6740	9502	16242	9502
7508	10569	18077	10569
6984	9840	16824	9840
7583	10680	18263	10680
7182	10113	17295	10113
-5990	-8428	-14418	-8428
-6306	-8895	-15201	-8895
-6032	-8476	-14508	-8476
-6299	-8896	-15195	-8896
-6165	-8653	-14818	-8653

Table 41

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

D.2.8. Second audio frame

The second audio frame is very similar to the frame decoded in the first example, but this time not 1 but 3 samples are present.

The frame header starts at position 0xcc and is broken down in the following table.

Start	Length	Contents	Description
0xcc+0	15 bits	0xff, 0b1111100	frame sync
0xcd+7	1 bit	0b0	blocking strategy
0xce+0	4 bits	0b0110	8-bit block size further down
0xce+4	4 bits	0b1001	sample rate 44.1 kHz
0xcf+0	4 bits	0b0001	stereo, no decorrelation
0xcf+4	3 bits	0b100	bit depth 16 bit
0xcf+7	1 bit	0b0	mandatory 0 bit
0xd0+0	1 byte	0x01	frame number 1
0xd1+0	1 byte	0x02	block size 3
0xd2+0	1 byte	0xa4	frame header CRC

Table 42

The first subframe starts at 0xd3+0 and is broken down in the following table.

Start	Length	Contents	Description
0xd3+0	1 bit	0b0	mandatory 0 bit
0xd3+1	6 bits	0b000001	verbatim subframe
0xd3+7	1 bit	0b0	no wasted bits used
0xd4+0	16 bits	0xc382	16-bit unencoded sample
0xd6+0	16 bits	0xc40b	16-bit unencoded sample
0xd8+0	16 bits	0xc14a	16-bit unencoded sample

Table 43

The second subframe starts at 0xda+0 and is broken down in the following table.

Start	Length	Contents	Description
0xda+0	1 bit	0b0	mandatory 0 bit
0xda+1	6 bits	0b000001	verbatim subframe
0xda+7	1 bit	0b1	wasted bits used
0xdb+0	1 bit	0b1	1 wasted bit used
0xdb+1	15 bits	0b110111001001000	15-bit unencoded sample
0xdd+0	15 bits	0b110111010000001	15-bit unencoded sample
0xde+7	15 bits	0b110110110011111	15-bit unencoded sample

Table 44

As this subframe uses wasted bits, the 15-bit unencoded samples need to be shifted left by 1 bit. For example, sample 1 is stored as -4536 and becomes -9072 after shifting left 1 bit.

As the last subframe does not end on byte alignment, 2 padding bits are added before the 2 byte frame CRC follows at 0xe1+0.

D.2.9. MD5 checksum verification

All samples in the file have been decoded, we can now verify the MD5 checksum. All sample values must be interleaved and stored signed, coded little-endian. The result of this follows in groups of 12 samples (i.e., 6 interchannel samples) per line.

```
0x8428 B617 7946 3129 5E3A 2722 D445 D128 0B3D B723 EB45 DF28
0x723f 1E25 9D46 4929 B841 7026 5747 B829 8F43 8127 AEC7 14DF
0x9FC4 41DD 54C7 E4DE A5C4 40DD 1EC6 33DE 82C3 90DC 0BC4 02DD
0x4AC1 3EDB
```

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

D.3. Decoding example 3

This example is once again a very short FLAC file. The focus of this example is on decoding a subframe with a linear predictor and a coded residual with more than one partition.

D.3.1. Example file 3 in hexadecimal representation

```
00000000: 664c 6143 8000 0022 1000 1000  fLaC..."....
0000000c: 0000 1f00 001f 07d0 0070 0000  .........p..
00000018: 0018 f8f9 e396 f5cb cfc6 dc80  ............
00000024: 7f99 7790 6b32 fff8 6802 0017  ..w.k2..h...
00000030: e944 004f 6f31 3d10 47d2 27cb  .D.Oo1=.G.'.
0000003c: 6d09 0831 452b dc28 2222 8057  m..1E+.(""W
00000048: a3                             .
```

D.3.2. Example file 3 in binary representation (only audio frame)

```
0000002a: 11111111 11111000 01101000 00000010  ..h.
0000002e: 00000000 00010111 11101001 01000100  ...D
00000032: 00000000 01001111 01101111 00110001  .Oo1
00000036: 00111101 00010000 01000111 11010010  =.G.
0000003a: 00100111 11001011 01101101 00001001  '.m.
0000003e: 00001000 00110001 01000101 00101011  .1E+
00000042: 11011100 00101000 00100010 00100010  .(""
00000046: 10000000 01010111 10100011           .W.
```

D.3.3. Streaminfo metadata block

Most of the streaminfo metadata block, including its header, is the same as in example 1, so only parts that are different are listed in the following table.

Start	Length	Contents	Description
0x0c+0	3 bytes	0x00001f	Min. frame size 31 byte
0x0f+0	3 bytes	0x00001f	Max. frame size 31 byte
0x12+0	20 bits	0x07d0, 0x0000	Sample rate 32000 hertz
0x14+4	3 bits	0b000	1 channel
0x14+7	5 bits	0b00111	Sample bit depth 8 bit
0x15+4	36 bits	0b0000, 0x00000018	Total no. of samples 24
0x1a	16 bytes	(...)	MD5 checksum

Table 45

D.3.4. Audio frame

The frame header starts at position 0x2a and is broken down in the following table.

Start	Length	Contents	Description
0x2a+0	15 bits	0xff, 0b1111100	Frame sync
0x2b+7	1 bit	0b0	blocking strategy
0x2c+0	4 bits	0b0110	8-bit block size further down
0x2c+4	4 bits	0b1000	Sample rate 32 kHz
0x2d+0	4 bits	0b0000	Mono audio (1 channel)
0x2d+4	3 bits	0b001	Bit depth 8 bit
0x2d+7	1 bit	0b0	Mandatory 0 bit
0x2e+0	1 byte	0x00	Frame number 0
0x2f+0	1 byte	0x17	Block size 24
0x30+0	1 byte	0xe9	Frame header CRC

Table 46

The first and only subframe starts at byte 0x31, it is broken down in the following table, without the coded residual.

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

Table 47

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

Start	Length	Contents	Description
0x37+5	2 bits	0b00	Rice-coded residual, 4-bit parameter
0x37+7	4 bits	0b0010	Partition order 2

0x38+3	4 bits	0b0011	Rice parameter 3
0x38+7	1 bit	0b1	Quotient 0
0x39+0	3 bits	0b110	Remainder 6
0x39+3	1 bit	0b1	Quotient 0
0x39+4	3 bits	0b001	Remainder 1
0x39+7	4 bits	0b0001	Quotient 3
0x3a+3	3 bits	0b001	Remainder 1
0x3a+6	4 bits	0b1111	No Rice parameter, escape code
0x3b+2	5 bits	0b00101	Partition encoded with 5 bits
0x3b+7	5 bits	0b10110	Residual -10
0x3c+4	5 bits	0b11010	Residual -6
0x3d+1	5 bits	0b00010	Residual 2
0x3d+6	5 bits	0b01000	Residual 8
0x3e+3	5 bits	0b01000	Residual 8
0x3f+0	5 bits	0b00110	Residual 6
0x3f+5	4 bits	0b0010	Rice parameter 2
0x40+1	22 bits	(...)	Residual partition 3
0x42+7	4 bits	0b0001	Rice parameter 1
0x43+3	23 bits	(...)	Residual partition 4

Table 48

The frame ends with 6 padding bits and a 2 byte frame CRC

To decode this subframe, 21 predictions have to be calculated and added to their corresponding residuals. This is a sequential process: as each prediction uses previous samples, it is not possible to start this decoding halfway a subframe or decode a subframe with parallel threads.

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

Residual	Predictor w/o shift	Predictor	Sample value
(warm-up)	N/A	N/A	0
(warm-up)	N/A	N/A	79
(warm-up)	N/A	N/A	111
3	303	75	78
-1	38	9	8
-13	-190	-48	-61
-10	-319	-80	-90
-6	-248	-62	-68
2	-58	-15	-13
8	137	34	42
8	236	59	67
6	191	47	53
0	53	13	13
-3	-93	-24	-27
-5	-161	-41	-46
-4	-134	-34	-38
-1	-44	-11	-12

1	52	13	14
1	94	23	24
4	60	15	19
2	17	4	6
2	-24	-6	-4
2	-26	-7	-5
0	1	0	0

Table 49

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

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

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

Authors' Addresses

Martijn van Beurden
Netherlands
Email: mvanb1@gmail.com

Andrew Weaver
Email: theandrewjw@gmail.com

CELLAR Group
Internet-Draft
Intended status: Standards Track
Expires: 24 April 2024

S. Lhomme

M. Bunkus

D. Rice

22 October 2023

Matroska Media Container Format Specifications
draft-ietf-cellar-matroska-21

Abstract

This document defines the Matroska audiovisual data container structure, including definitions of its structural elements, as well as its terminology, vocabulary, and application.

This document updates [RFC8794] to permit the use of a previously reserved EBML Element ID.

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 24 April 2024.

Copyright Notice

Table of Contents

1. Introduction

Matroska is an audiovisual data container format. It was derived from a project called [MCF], but diverges from it significantly because it is based on EBML (Extensible Binary Meta Language) [RFC8794], a binary derivative of XML. EBML provides significant advantages in terms of future format extensibility, without breaking file support in parsers reading the previous versions.

First, it is essential to clarify exactly "What an Audio/Video container is", to avoid any misunderstandings:

* It is NOT a video or audio compression format (codec)
* It is an envelope in which there can be many audio, video, and subtitles streams, allowing the user to store a complete movie or CD in a single file.

Matroska is designed with the future in mind. It incorporates features such as:

* Fast seeking in the file
* Chapter entries
* Full metadata (tags) support
* Selectable subtitle/audio/video streams
* Modularly expandable
* Error resilience (can recover playback even when the stream is damaged)
* Streamable over the internet and local networks (HTTP [RFC9110], FTP [RFC0959], SMB [SMB-CIFS], etc.)
* Menus (like DVDs have [DVD-Video])

2. Status of this document

This document covers Matroska versions 1, 2, 3 and 4. Matroska v4 is the current version. Matroska 1 to 3 are no longer maintained. No new elements are expected in files with version numbers 1, 2, or 3.

3. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document defines specific terms in order to define the format and application of Matroska. Specific terms are defined below:

Matroska: A multimedia container format based on EBML (Extensible Binary Meta Language).
Matroska Reader: A data parser that interprets the semantics of a Matroska document and creates a way for programs to use Matroska.
Matroska Player: A Matroska Reader with a primary purpose of playing audiovisual files, including Matroska documents.
Matroska Writer: A data writer that creates Matroska documents.

4. Matroska Overview

4.1. Principles

Matroska is a Document Type of EBML (Extensible Binary Meta Language). This specification is dependent on the EBML Specification [RFC8794]. For an understanding of Matroska's EBML Schema, see in particular the sections of the EBML Specification covering EBML Element Types (Section 7), EBML Schema (Section 11.1), and EBML Structure (Section 3).

4.2. Updates to RFC 8794

Because of an oversight, [RFC8794] reserved EBML ID 0x80, which is used by deployed Matroska implementations. For this reason, this specification updates [RFC8794] to make 0x80 a legal EBML ID. Specifically, the following are changed in [RFC8794]:

* From Errata 7189

In Section 17.1,

OLD:

One-octet Element IDs MUST be between 0x81 and 0xFE. These items are valuable because they are short, and they need to be used for commonly repeated elements. Element IDs are to be allocated within this range according to the "RFC Required" policy [RFC8126].

The following one-octet Element IDs are RESERVED: 0xFF and 0x80.

NEW:

One-octet Element IDs MUST be between 0x80 and 0xFE. These items are valuable because they are short, and they need to be used for commonly repeated elements. Element IDs are to be allocated within this range according to the "RFC Required" policy [RFC8126].

The following one-octet Element ID is RESERVED: 0xFF.

* From Errata 7191

In Section 5,

OLD:

Element ID Octet Length	Range of Valid Element IDs	Number of Valid Element IDs
1	0x81 - 0xFE	126

NEW:

Element ID Octet Length	Range of Valid Element IDs	Number of Valid Element IDs
1	0x80 - 0xFE	127

4.3. Added EBML Constraints

As an EBML Document Type, Matroska adds the following constraints to the EBML specification.

* The docType of the EBML Header MUST be "matroska".
* The EBMLMaxIDLength of the EBML Header MUST be 4.
* The EBMLMaxSizeLength of the EBML Header MUST be between 1 and 8 inclusive.

4.4. Design Rules

The Root Element and all Top-Levels Elements MUST use 4 octets for their EBML Element ID -- i.e. Segment and direct children of Segment.

Legacy EBML/Matroska parsers did not handle Empty Elements properly, elements present in the file but with a length of zero. They always assumed the value was 0 for integers/dates or 0x0p+0, the textual expression of floats using the [ISO9899] format, no matter the default value of the element which should have been used instead. Therefore, Matroska writers MUST NOT use EBML Empty Elements, if the element has a default value that is not 0 for integers/dates and 0x0p+0 for floats.

When adding new elements to Matroska, these rules apply:

* A non-mandatory integer/date Element MUST NOT have a default value other than 0.
* A non-mandatory float Element MUST NOT have a default value other than 0x0p+0.
* A non-mandatory string Element MUST NOT have a default value, as empty string cannot be defined in the XML Schema.

4.5. Data Layout

A Matroska file MUST be composed of at least one EBML Document using the Matroska Document Type. Each EBML Document MUST start with an EBML Header and MUST be followed by the EBML Root Element, defined as Segment in Matroska. Matroska defines several Top-Level Elements which may occur within the Segment.

As an example, a simple Matroska file consisting of a single EBML Document could be represented like this:

* EBML Header
* Segment

A more complex Matroska file consisting of an EBML Stream (consisting of two EBML Documents) could be represented like this:

* EBML Header
* Segment
* EBML Header
* Segment

The following diagram represents a simple Matroska file, comprised of an EBML Document with an EBML Header, a Segment Element (the Root Element), and all eight Matroska Top-Level Elements. In the

following diagrams of this section, horizontal spacing expresses a parent-child relationship between Matroska Elements (e.g., the Info Element is contained within the Segment Element) whereas vertical alignment represents the storage order within the file.

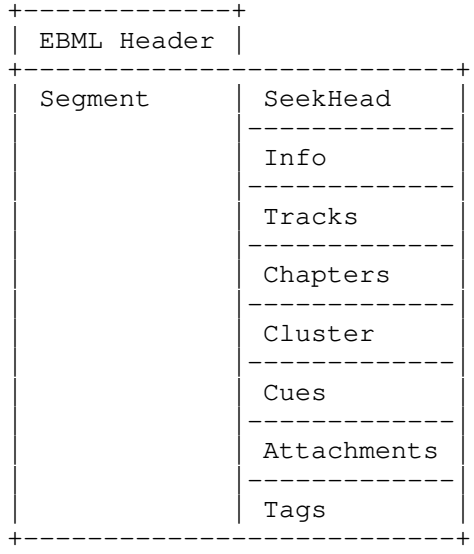


Figure 1: Basic layout of a Matroska file.

The Matroska EBML Schema defines eight Top-Level Elements:

* SeekHead (Section 6.3),
* Info (Section 6.5),
* Tracks (Section 18),
* Chapters (Section 20),
* Cluster (Section 10),
* Cues (Section 22),
* Attachments (Section 21),
* and Tags (Section 6.8).

The SeekHead Element (also known as MetaSeek) contains an index of Top-Level Elements locations within the Segment. Use of the SeekHead Element is RECOMMENDED. Without a SeekHead Element, a Matroska parser would have to search the entire file to find all of the other Top-Level Elements. This is due to Matroska's flexible ordering requirements; for instance, it is acceptable for the Chapters Element to be stored after the Cluster Elements.

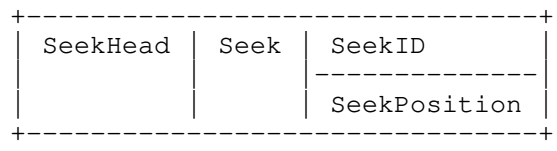


Figure 2: Representation of a SeekHead Element.

The Info Element contains vital information for identifying the whole Segment. This includes the title for the Segment, a randomly generated unique identifier, and the unique identifier(s) of any linked Segment Elements.

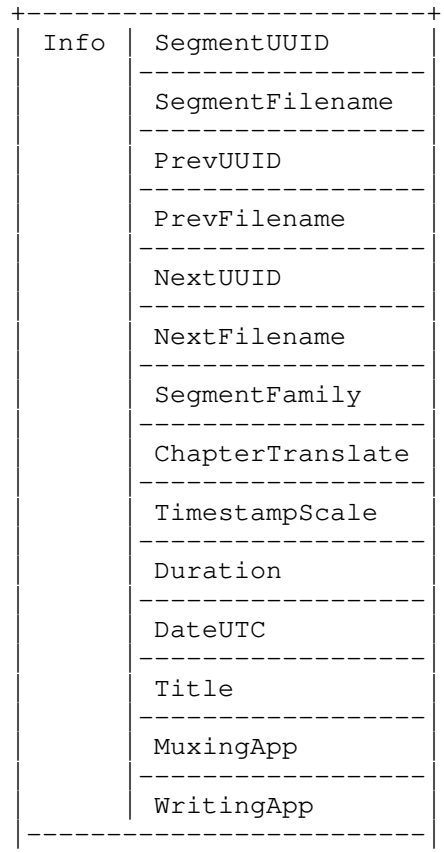


Figure 3: Representation of an Info Element and its Child Elements.

The Tracks Element defines the technical details for each track and can store the name, number, unique identifier, language, and type (audio, video, subtitles, etc.) of each track. For example, the Tracks Element MAY store information about the resolution of a video track or sample rate of an audio track.

The Tracks Element MUST identify all the data needed by the codec to decode the data of the specified track. However, the data required is contingent on the codec used for the track. For example, a Track Element for uncompressed audio only requires the audio bit rate to be present. A codec such as AC-3 would require that the CodecID Element be present for all tracks, as it is the primary way to identify which codec to use to decode the track.

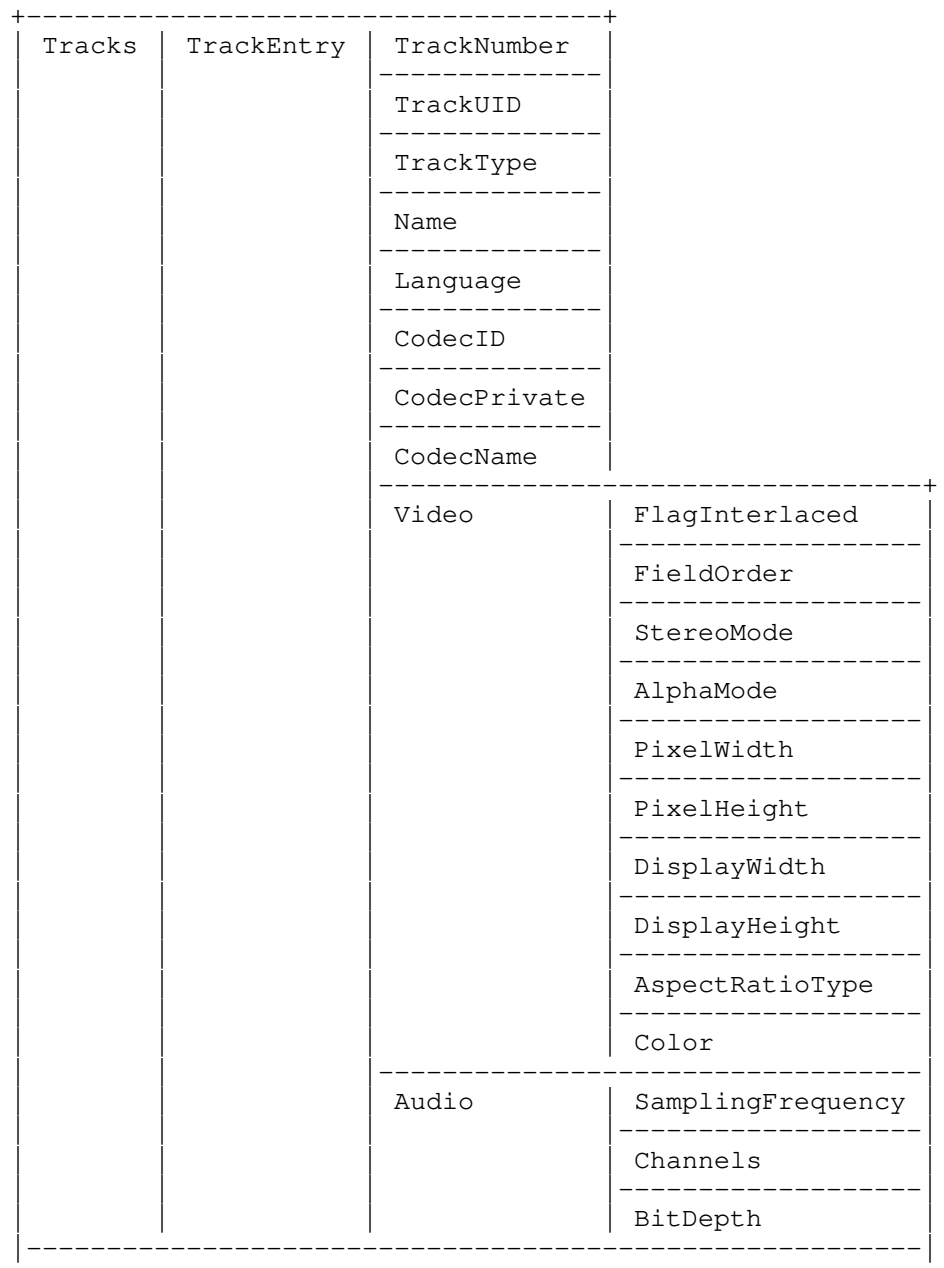


Figure 4: Representation of the Tracks Element and a selection of its Descendant Elements.

The Chapters Element lists all of the chapters. Chapters are a way to set predefined points to jump to in video or audio.

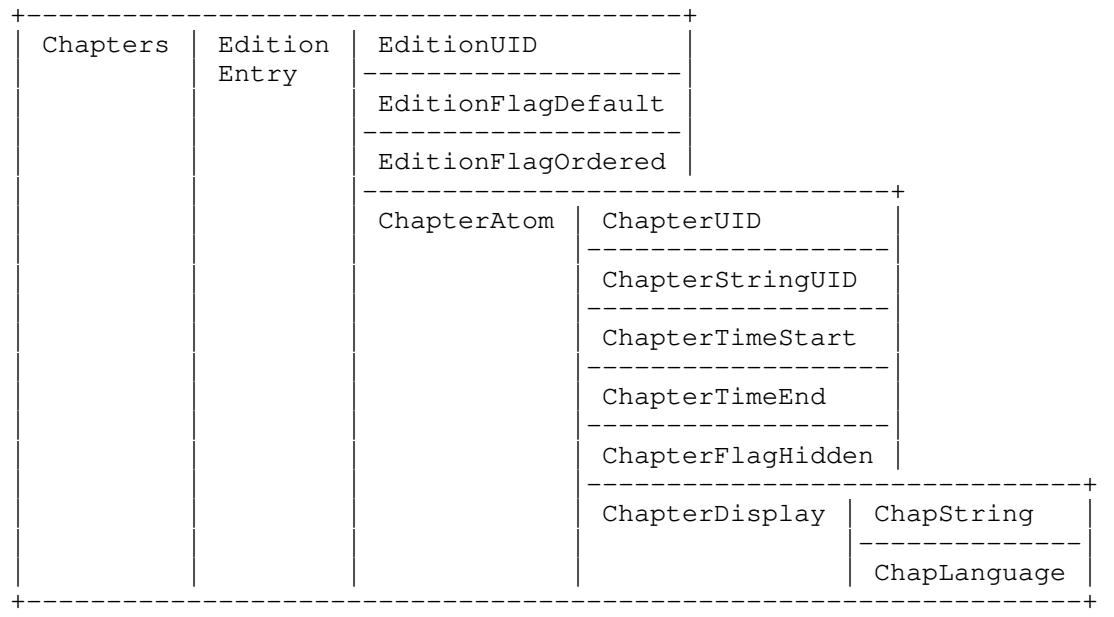


Figure 5: Representation of the Chapters Element and a selection of its Descendant Elements.

Cluster Elements contain the content for each track, e.g., video frames. A Matroska file SHOULD contain at least one Cluster Element. In the rare case it doesn't, there should be a form of Segment linking with other Segments, possibly using Chapters, see Section 17.

The Cluster Element helps to break up SimpleBlock or BlockGroup Elements and helps with seeking and error protection. Every Cluster Element MUST contain a Timestamp Element. This SHOULD be the Timestamp Element used to play the first Block in the Cluster Element, unless a different value is needed to accommodate for more Blocks, see Section 11.2.

Cluster Elements contain one or more block element, such as BlockGroup or SimpleBlock elements. In some situations, a Cluster Element MAY contain no block element, for example in a live recording when no data has been collected.

A BlockGroup Element MAY contain a Block of data and any information relating directly to that Block.

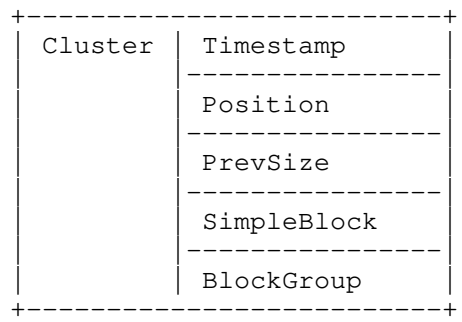


Figure 6: Representation of a Cluster Element and its immediate Child Elements.

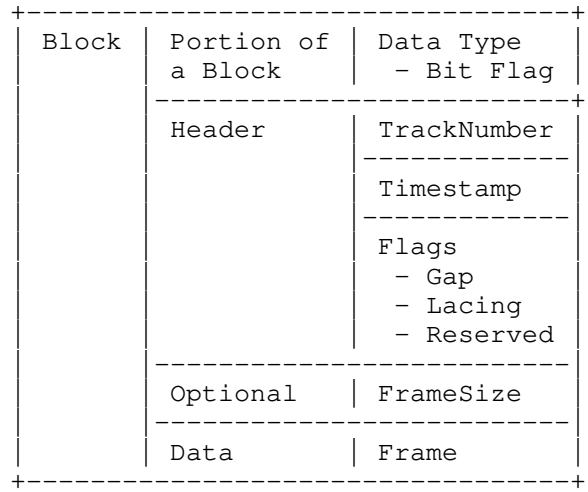


Figure 7: Representation of the Block Element structure.

Each Cluster MUST contain exactly one Timestamp Element. The Timestamp Element value MUST be stored once per Cluster. The Timestamp Element in the Cluster is relative to the entire Segment. The Timestamp Element SHOULD be the first Element in the Cluster it belongs to, or the second Element if that Cluster contains a CRC-32 element (Section 6.2)

Additionally, the Block contains an offset that, when added to the Cluster's Timestamp Element value, yields the Block's effective timestamp. Therefore, timestamp in the Block itself is relative to the Timestamp Element in the Cluster. For example, if the Timestamp Element in the Cluster is set to 10 seconds and a Block in that Cluster is supposed to be played 12 seconds into the clip, the timestamp in the Block would be set to 2 seconds.

The ReferenceBlock in the BlockGroup is used instead of the basic "P-frame"/"B-frame" description. Instead of simply saying that this Block depends on the Block directly before, or directly afterwards, the Timestamp of the necessary Block is used. Because there can be as many ReferenceBlock Elements as necessary for a Block, it allows for some extremely complex referencing.

The Cues Element is used to seek when playing back a file by providing a temporal index for some of the Tracks. It is similar to the SeekHead Element, but used for seeking to a specific time when playing back the file. It is possible to seek without this element, but it is much more difficult because a Matroska Reader would have to 'hunt and peck' through the file looking for the correct timestamp.

The Cues Element SHOULD contain at least one CuePoint Element. Each CuePoint Element stores the position of the Cluster that contains the BlockGroup or SimpleBlock Element. The timestamp is stored in the CueTime Element and location is stored in the CueTrackPositions Element.

The Cues Element is flexible. For instance, Cues Element can be used to index every single timestamp of every Block or they can be indexed selectively.

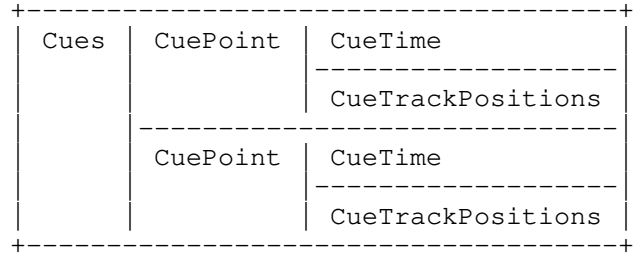


Figure 8: Representation of a Cues Element and two levels of its Descendant Elements.

The Attachments Element is for attaching files to a Matroska file such as pictures, fonts, webpages, etc.

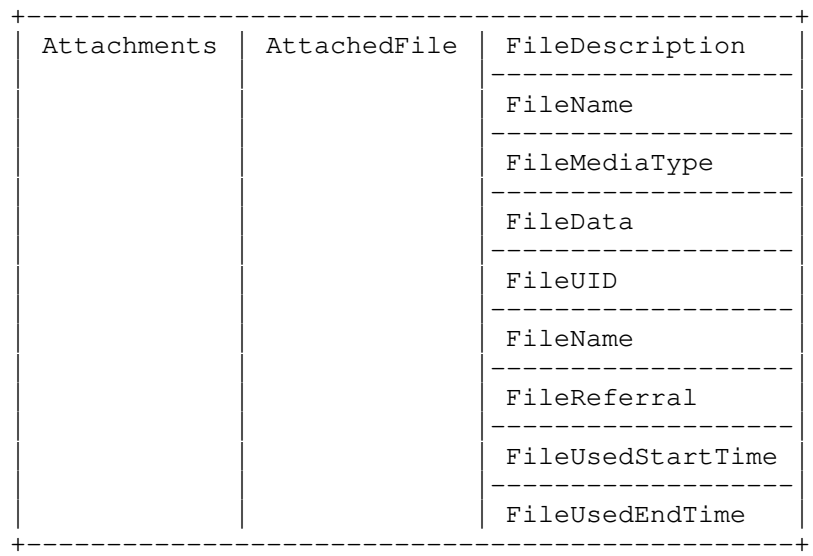


Figure 9: Representation of an Attachments Element.

The Tags Element contains metadata that describes the Segment and potentially its Tracks, Chapters, and Attachments. Each Track or Chapter that those tags applies to has its UID listed in the Tags. The Tags contain all extra information about the file: scriptwriter, singer, actors, directors, titles, edition, price, dates, genre, comments, etc. Tags can contain their values in multiple languages. For example, a movie's "title" Tag might contain both the original English title as well as the title it was released as in Germany.

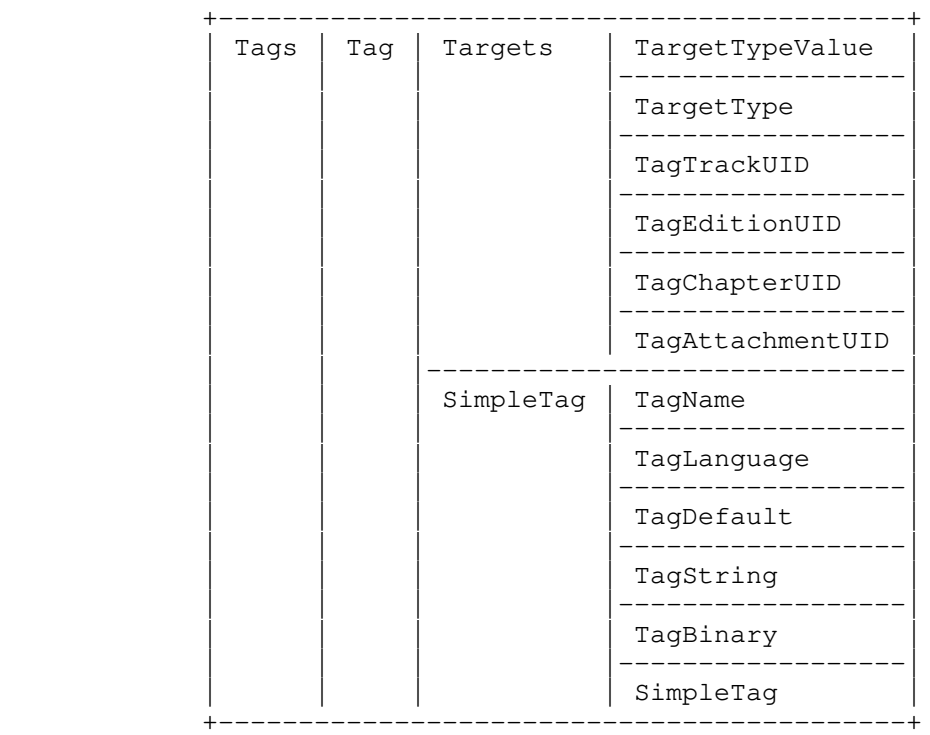


Figure 10: Representation of a Tags Element and three levels of its Children Elements.

5. Matroska Schema

This specification includes an EBML Schema, which defines the Elements and structure of Matroska using the EBML Schema elements and attributes defined in Section 11.1 of [RFC8794]. The EBML Schema defines every valid Matroska element in a manner defined by the EBML specification.

Attributes using their default value like minOccurs, minver, etc. or with undefined values like length, maxver, etc. are omitted.

Here the definition of each Matroska Element is provided.

5.1. Segment Element

id / type: 0x18538067 / master

unknownsizeallowed: True

path: \Segment
minOccurs / maxOccurs: 1 / 1
definition: The Root Element that contains all other Top-Level Elements; see Section 4.5.

5.1.1. SeekHead Element

id / type: 0x114D9B74 / master
path: \Segment\SeekHead
maxOccurs: 2
definition: Contains seeking information of Top-Level Elements; see Section 4.5.

5.1.1.1. Seek Element

id / type: 0x4DBB / master
path: \Segment\SeekHead\Seek
minOccurs: 1
definition: Contains a single seek entry to an EBML Element.

5.1.1.1.1. SeekID Element

id / type: 0x53AB / binary
length: 4
path: \Segment\SeekHead\Seek\SeekID
minOccurs / maxOccurs: 1 / 1
definition: The binary EBML ID of a Top-Level Element.

5.1.1.1.2. SeekPosition Element

id / type: 0x53AC / uinteger
path: \Segment\SeekHead\Seek\SeekPosition
minOccurs / maxOccurs: 1 / 1
definition: The Segment Position (Section 16) of a Top-Level Element.

5.1.2. Info Element

id / type: 0x1549A966 / master
path: \Segment\Info
minOccurs / maxOccurs: 1 / 1

recurring: True

definition: Contains general information about the Segment.

5.1.2.1. SegmentUUID Element

id / type: 0x73A4 / binary
length: 16
path: \Segment\Info\SegmentUUID
maxOccurs: 1
definition: A randomly generated unique ID to identify the Segment amongst many others (128 bits). It is equivalent to a UUID v4 [RFC4122] with all bits randomly (or pseudo-randomly) chosen. An actual UUID v4 value, where some bits are not random, MAY also be used.
usage notes: If the Segment is a part of a Linked Segment, then this Element is REQUIRED. The value of the unique ID MUST contain at least one bit set to 1.

5.1.2.2. SegmentFilename Element

id / type: 0x7384 / utf-8
path: \Segment\Info\SegmentFilename
maxOccurs: 1
definition: A filename corresponding to this Segment.

5.1.2.3. PrevUUID Element

id / type: 0x3CB923 / binary
length: 16
path: \Segment\Info\PrevUUID
maxOccurs: 1
definition: An ID to identify the previous Segment of a Linked Segment.
usage notes: If the Segment is a part of a Linked Segment that uses Hard Linking (Section 17.1), then either the PrevUUID or the NextUUID Element is REQUIRED. If a Segment contains a PrevUUID but not a NextUUID, then it MAY be considered as the last Segment of the Linked Segment. The PrevUUID MUST NOT be equal to the SegmentUUID.

5.1.2.4. PrevFilename Element

id / type: 0x3C83AB / utf-8
path: \Segment\Info\PrevFilename
maxOccurs: 1
definition: A filename corresponding to the file of the previous Linked Segment.
usage notes: Provision of the previous filename is for display convenience, but PrevUUID SHOULD be considered authoritative for identifying the previous Segment in a Linked Segment.

5.1.2.5. NextUUID Element

id / type: 0x3EB923 / binary
length: 16
path: \Segment\Info\NextUUID
maxOccurs: 1
definition: An ID to identify the next Segment of a Linked Segment.
usage notes: If the Segment is a part of a Linked Segment that uses Hard Linking (Section 17.1), then either the PrevUUID or the NextUUID Element is REQUIRED. If a Segment contains a NextUUID but not a PrevUUID, then it MAY be considered as the first Segment of the Linked Segment. The NextUUID MUST NOT be equal to the SegmentUUID.

5.1.2.6. NextFilename Element

id / type: 0x3E83BB / utf-8
path: \Segment\Info\NextFilename
maxOccurs: 1
definition: A filename corresponding to the file of the next Linked Segment.
usage notes: Provision of the next filename is for display convenience, but NextUUID SHOULD be considered authoritative for identifying the Next Segment.

5.1.2.7. SegmentFamily Element

id / type: 0x4444 / binary
length: 16
path: \Segment\Info\SegmentFamily
definition: A unique ID that all Segments of a Linked Segment MUST share (128 bits). It is equivalent to a UUID v4 [RFC4122] with all bits randomly (or pseudo-randomly) chosen. An actual UUID v4 value, where some bits are not random, MAY also be used.
usage notes: If the Segment Info contains a ChapterTranslate element, this Element is REQUIRED.

5.1.2.8. ChapterTranslate Element

id / type: 0x6924 / master
path: \Segment\Info\ChapterTranslate
definition: The mapping between this Segment and a segment value in the given Chapter Codec.
rationale: Chapter Codec may need to address different segments, but

they may not know of the way to identify such segment when stored in Matroska. This element and its child elements add a way to map the internal segments known to the Chapter Codec to the Segment IDs in Matroska. This allows remuxing a file with Chapter Codec without changing the content of the codec data, just the Segment mapping.

5.1.2.8.1. ChapterTranslateID Element

id / type: 0x69A5 / binary
path: \Segment\Info\ChapterTranslate\ChapterTranslateID
minOccurs / maxOccurs: 1 / 1
definition: The binary value used to represent this Segment in the chapter codec data. The format depends on the ChapProcessCodecID used; see Section 5.1.7.1.4.15.

5.1.2.8.2. ChapterTranslateCodec Element

id / type: 0x69BF / uinteger
path: \Segment\Info\ChapterTranslate\ChapterTranslateCodec
minOccurs / maxOccurs: 1 / 1
definition: This ChapterTranslate applies to this chapter codec of the given chapter edition(s); see Section 5.1.7.1.4.15.

defined values:

value	label	definition
0	Matroska Script	Chapter commands using the Matroska Script codec.
1	DVD-menu	Chapter commands using the DVD-like codec.

Table 1: ChapterTranslateCodec values

5.1.2.8.3. ChapterTranslateEditionUID Element

id / type: 0x69FC / uinteger
path: \Segment\Info\ChapterTranslate\ChapterTranslateEditionUID
definition: Specify a chapter edition UID on which this ChapterTranslate applies.
usage notes: When no ChapterTranslateEditionUID is specified in the ChapterTranslate, the ChapterTranslate applies to all chapter editions found in the Segment using the given ChapterTranslateCodec.

5.1.2.9. TimestampScale Element

id / type / default: 0x2AD7B1 / uinteger / 1000000
range: not 0
path: \Segment\Info\TimestampScale
minOccurs / maxOccurs: 1 / 1
definition: Base unit for Segment Ticks and Track Ticks, in nanoseconds. A TimestampScale value of 1000000 means scaled timestamps in the Segment are expressed in milliseconds; see Section 11 on how to interpret timestamps.

5.1.2.10. Duration Element

id / type: 0x4489 / float
range: > 0x0p+0
path: \Segment\Info\Duration
maxOccurs: 1
definition: Duration of the Segment, expressed in Segment Ticks which is based on TimestampScale; see Section 11.1.

5.1.2.11. DateUTC Element

id / type: 0x4461 / date
path: \Segment\Info\DateUTC
maxOccurs: 1
definition: The date and time that the Segment was created by the muxing application or library.

5.1.2.12. Title Element

id / type: 0x7BA9 / utf-8
path: \Segment\Info\Title
maxOccurs: 1
definition: General name of the Segment.

5.1.2.13. MuxingApp Element

id / type: 0x4D80 / utf-8
path: \Segment\Info\MuxingApp
minOccurs / maxOccurs: 1 / 1
definition: Muxing application or library (example: "libmatroska-0.4.3").
usage notes: Include the full name of the application or library followed by the version number.

5.1.2.14. WritingApp Element

id / type: 0x5741 / utf-8

path: \Segment\Info\WritingApp
minOccurs / maxOccurs: 1 / 1
definition: Writing application (example: "mkvmerge-0.3.3").
usage notes: Include the full name of the application followed by the version number.

5.1.3. Cluster Element

id / type: 0x1F43B675 / master

unknownsizeallowed: True

path: \Segment\Cluster
definition: The Top-Level Element containing the (monolithic) Block structure.

5.1.3.1. Timestamp Element

id / type: 0xE7 / uinteger
path: \Segment\Cluster\Timestamp
minOccurs / maxOccurs: 1 / 1
definition: Absolute timestamp of the cluster, expressed in Segment Ticks which is based on TimestampScale; see Section 11.1.
usage notes: This element SHOULD be the first child element of the Cluster it belongs to, or the second if that Cluster contains a CRC-32 element (Section 6.2).

5.1.3.2. Position Element

id / type: 0xA7 / uinteger
path: \Segment\Cluster\Position
maxOccurs: 1
maxver: 4
definition: The Segment Position of the Cluster in the Segment (0 in live streams). It might help to resynchronise offset on damaged streams.

5.1.3.3. PrevSize Element

id / type: 0xAB / uinteger
path: \Segment\Cluster\PrevSize
maxOccurs: 1
definition: Size of the previous Cluster, in octets. Can be useful for backward playing.

5.1.3.4. SimpleBlock Element

id / type: 0xA3 / binary

path: \Segment\Cluster\SimpleBlock
minver: 2
definition: Similar to Block, see Section 10.1, but without all the extra information, mostly used to reduced overhead when no extra feature is needed; see Section 10.2 on SimpleBlock Structure.

5.1.3.5. BlockGroup Element

id / type: 0xA0 / master
path: \Segment\Cluster\BlockGroup
definition: Basic container of information containing a single Block and information specific to that Block.

5.1.3.5.1. Block Element

id / type: 0xA1 / binary
path: \Segment\Cluster\BlockGroup\Block
minOccurs / maxOccurs: 1 / 1
definition: Block containing the actual data to be rendered and a timestamp relative to the Cluster Timestamp; see Section 10.1 on Block Structure.

5.1.3.5.2. BlockAdditions Element

id / type: 0x75A1 / master
path: \Segment\Cluster\BlockGroup\BlockAdditions
maxOccurs: 1
definition: Contain additional binary data to complete the main one; see Codec BlockAdditions section of [MatroskaCodec] for more information. An EBML parser that has no knowledge of the Block structure could still see and use/skip these data.

5.1.3.5.2.1. BlockMore Element

id / type: 0xA6 / master
path: \Segment\Cluster\BlockGroup\BlockAdditions\BlockMore
minOccurs: 1
definition: Contain the BlockAdditional and some parameters.

5.1.3.5.2.2. BlockAdditional Element

id / type: 0xA5 / binary
path: \Segment\Cluster\BlockGroup\BlockAdditions\BlockMore\BlockAdditional
minOccurs / maxOccurs: 1 / 1
definition: Interpreted by the codec as it wishes (using the BlockAddID).

5.1.3.5.2.3. BlockAddID Element

id / type / default: 0xEE / uinteger / 1
range: not 0
path: \Segment\Cluster\BlockGroup\BlockAdditions\BlockMore\BlockAddID
minOccurs / maxOccurs: 1 / 1
definition: An ID to identify how to interpret the BlockAdditional data; see Codec BlockAdditions section of [MatroskaCodec] for more information. A value of 1 indicates that the meaning of the BlockAdditional data is defined by the codec. Any other value indicates the meaning of the BlockAdditional data is found in the BlockAddIDType found in the TrackEntry.
usage notes: Each BlockAddID value MUST be unique between all BlockMore elements found in a BlockAdditions.
usage notes: To keep MaxBlockAdditionID as low as possible, small values SHOULD be used.

5.1.3.5.3. BlockDuration Element

id / type: 0x9B / uinteger
path: \Segment\Cluster\BlockGroup\BlockDuration
minOccurs / maxOccurs: see implementation notes / 1
definition: The duration of the Block, expressed in Track Ticks; see Section 11.1. The BlockDuration Element can be useful at the end of a Track to define the duration of the last frame (as there is no subsequent Block available), or when there is a break in a track like for subtitle tracks.

notes:

attribute	note
minOccurs	BlockDuration MUST be set (minOccurs=1) if the associated TrackEntry stores a DefaultDuration value.
default	When not written and with no DefaultDuration, the value is assumed to be the difference between the timestamp of this Block and the timestamp of the next Block in "display" order (not coding order).

Table 2: BlockDuration implementation notes

5.1.3.5.4. ReferencePriority Element

id / type / default: 0xFA / uinteger / 0
path: \Segment\Cluster\BlockGroup\ReferencePriority
minOccurs / maxOccurs: 1 / 1
definition: This frame is referenced and has the specified cache priority. In cache only a frame of the same or higher priority can replace this frame. A value of 0 means the frame is not referenced.

5.1.3.5.5. ReferenceBlock Element

id / type: 0xFB / integer
path: \Segment\Cluster\BlockGroup\ReferenceBlock
definition: A timestamp value, relative to the timestamp of the Block in this BlockGroup, expressed in Track Ticks; see Section 11.1. This is used to reference other frames necessary to decode this frame. The relative value SHOULD correspond to a valid Block this Block depends on. Historically Matroska Writer didn't write the actual Block(s) this Block depends on, but _some_ Block in the past.

The value "0" MAY also be used to signify this Block cannot be decoded on its own, but without knownledge of which Block is necessary. In this case, other ReferenceBlock MUST NOT be found in the same BlockGroup.

If the BlockGroup doesn't have any ReferenceBlock element, then the Block it contains can be decoded without using any other Block data.

5.1.3.5.6. CodecState Element

id / type: 0xA4 / binary
path: \Segment\Cluster\BlockGroup\CodecState
maxOccurs: 1
minver: 2
definition: The new codec state to use. Data interpretation is private to the codec. This information SHOULD always be referenced by a seek entry.

5.1.3.5.7. DiscardPadding Element

id / type: 0x75A2 / integer
path: \Segment\Cluster\BlockGroup\DiscardPadding
maxOccurs: 1
minver: 4
definition: Duration of the silent data added to the Block,

expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1 (padding at the end of the Block for positive value, at the beginning of the Block for negative value). The duration of DiscardPadding is not calculated in the duration of the TrackEntry and SHOULD be discarded during playback.

5.1.4. Tracks Element

id / type: 0x1654AE6B / master
path: \Segment\Tracks
maxOccurs: 1

recurring: True

definition: A Top-Level Element of information with many tracks described.

5.1.4.1. TrackEntry Element

id / type: 0xAE / master
path: \Segment\Tracks\TrackEntry
minOccurs: 1
definition: Describes a track with all Elements.

5.1.4.1.1. TrackNumber Element

id / type: 0xD7 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\TrackNumber
minOccurs / maxOccurs: 1 / 1
definition: The track number as used in the Block Header.

5.1.4.1.2. TrackUID Element

id / type: 0x73C5 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\TrackUID
minOccurs / maxOccurs: 1 / 1
definition: A unique ID to identify the Track.

stream copy: True (Section 8)

5.1.4.1.3. TrackType Element

id / type: 0x83 / uinteger
path: \Segment\Tracks\TrackEntry\TrackType
minOccurs / maxOccurs: 1 / 1
definition: The TrackType defines the type of each frame found in

the Track. The value SHOULD be stored on 1 octet.

defined values:

value	label	each frame contains
1	video	An image.
2	audio	Audio samples.
3	complex	A mix of different other TrackType. The codec needs to define how the Matroska Player should interpret such data.
16	logo	An image to be rendered over the video track(s).
17	subtitle	Subtitle or closed caption data to be rendered over the video track(s).
18	buttons	Interactive button(s) to be rendered over the video track(s).
32	control	Metadata used to control the player of the Matroska Player.
33	metadata	Timed metadata that can be passed on to the Matroska Player.

Table 3: TrackType values

stream copy: True (Section 8)

5.1.4.1.4. FlagEnabled Element

id / type / default: 0xB9 / uinteger / 1
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagEnabled
minOccurs / maxOccurs: 1 / 1
minver: 2
definition: Set to 1 if the track is usable. It is possible to turn a not usable track into a usable track using chapter codecs or control tracks.

5.1.4.1.5. FlagDefault Element

id / type / default: 0x88 / uinteger / 1
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagDefault
minOccurs / maxOccurs: 1 / 1
definition: Set if that track (audio, video or subs) is eligible for automatic selection by the player; see Section 19 for more details.

5.1.4.1.6. FlagForced Element

id / type / default: 0x55AA / uinteger / 0
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagForced
minOccurs / maxOccurs: 1 / 1
definition: Applies only to subtitles. Set if that track is eligible for automatic selection by the player if it matches the user's language preference, even if the user's preferences would normally not enable subtitles with the selected audio track; this can be used for tracks containing only translations of foreign-language audio or onscreen text. See Section 19 for more details.

5.1.4.1.7. FlagHearingImpaired Element

id / type: 0x55AB / uinteger
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagHearingImpaired
maxOccurs: 1
minver: 4
definition: Set to 1 if and only if that track is suitable for users with hearing impairments.

5.1.4.1.8. FlagVisualImpaired Element

id / type: 0x55AC / uinteger
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagVisualImpaired
maxOccurs: 1
minver: 4
definition: Set to 1 if and only if that track is suitable for users with visual impairments.

5.1.4.1.9. FlagTextDescriptions Element

id / type: 0x55AD / uinteger
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagTextDescriptions

maxOccurs: 1
minver: 4
definition: Set to 1 if and only if that track contains textual descriptions of video content.

5.1.4.1.10. FlagOriginal Element

id / type: 0x55AE / uinteger
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagOriginal
maxOccurs: 1
minver: 4
definition: Set to 1 if and only if that track is in the content's original language.

5.1.4.1.11. FlagCommentary Element

id / type: 0x55AF / uinteger
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagCommentary
maxOccurs: 1
minver: 4
definition: Set to 1 if and only if that track contains commentary.

5.1.4.1.12. FlagLacing Element

id / type / default: 0x9C / uinteger / 1
range: 0-1
path: \Segment\Tracks\TrackEntry\FlagLacing
minOccurs / maxOccurs: 1 / 1
definition: Set to 1 if the track MAY contain blocks using lacing. When set to 0 all blocks MUST have their lacing flags set to No lacing; see Section 10.3 on Block Lacing.

5.1.4.1.13. DefaultDuration Element

id / type: 0x23E383 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\DefaultDuration
maxOccurs: 1
definition: Number of nanoseconds per frame, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1 (frame in the Matroska sense -- one Element put into a (Simple)Block).

stream copy: True (Section 8)

5.1.4.1.14. DefaultDecodedFieldDuration Element

id / type: 0x234E7A / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\DefaultDecodedFieldDuration
maxOccurs: 1
minver: 4
definition: The period between two successive fields at the output of the decoding process, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1. see Section 9 for more information

stream copy: True (Section 8)

5.1.4.1.15. TrackTimestampScale Element

id / type / default: 0x23314F / float / 0x1p+0
range: > 0x0p+0
path: \Segment\Tracks\TrackEntry\TrackTimestampScale
minOccurs / maxOccurs: 1 / 1
maxver: 3
definition: The scale to apply on this track to work at normal speed in relation with other tracks (mostly used to adjust video speed when the audio length differs).

stream copy: True (Section 8)

5.1.4.1.16. MaxBlockAdditionID Element

id / type / default: 0x55EE / uinteger / 0
path: \Segment\Tracks\TrackEntry\MaxBlockAdditionID
minOccurs / maxOccurs: 1 / 1
definition: The maximum value of BlockAddID (Section 5.1.3.5.2.3). A value 0 means there is no BlockAdditions (Section 5.1.3.5.2) for this track.

5.1.4.1.17. BlockAdditionMapping Element

id / type: 0x41E4 / master
path: \Segment\Tracks\TrackEntry\BlockAdditionMapping
minver: 4
definition: Contains elements that extend the track format, by adding content either to each frame, with BlockAddID (Section 5.1.3.5.2.3), or to the track as a whole with BlockAddIDExtraData.

5.1.4.1.17.1. BlockAddIDValue Element

id / type: 0x41F0 / uinteger

range: >=2
path: \Segment\Tracks\TrackEntry\BlockAdditionMapping\BlockAddIDValue
maxOccurs: 1
minver: 4
definition: If the track format extension needs content beside frames, the value refers to the BlockAddID (Section 5.1.3.5.2.3), value being described.
usage notes: To keep MaxBlockAdditionID as low as possible, small values SHOULD be used.

5.1.4.1.17.2. BlockAddIDName Element

id / type: 0x41A4 / string
path: \Segment\Tracks\TrackEntry\BlockAdditionMapping\BlockAddIDName
maxOccurs: 1
minver: 4
definition: A human-friendly name describing the type of BlockAdditional data, as defined by the associated Block Additional Mapping.

5.1.4.1.17.3. BlockAddIDType Element

id / type / default: 0x41E7 / uinteger / 0
path: \Segment\Tracks\TrackEntry\BlockAdditionMapping\BlockAddIDType
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Stores the registered identifier of the Block Additional Mapping to define how the BlockAdditional data should be handled.
usage notes: If BlockAddIDType is 0, the BlockAddIDValue and corresponding BlockAddID values MUST be 1.

5.1.4.1.17.4. BlockAddIDExtraData Element

id / type: 0x41ED / binary
path: \Segment\Tracks\TrackEntry\BlockAdditionMapping\BlockAddIDExtraData
maxOccurs: 1
minver: 4
definition: Extra binary data that the BlockAddIDType can use to interpret the BlockAdditional data. The interpretation of the binary data depends on the BlockAddIDType value and the corresponding Block Additional Mapping.

5.1.4.1.18. Name Element

id / type: 0x536E / utf-8
path: \Segment\Tracks\TrackEntry\Name

maxOccurs: 1
definition: A human-readable track name.

5.1.4.1.19. Language Element

id / type / default: 0x22B59C / string / eng
path: \Segment\Tracks\TrackEntry\Language
minOccurs / maxOccurs: 1 / 1
definition: The language of the track, in the Matroska languages form; see Section 12 on language codes. This Element MUST be ignored if the LanguageBCP47 Element is used in the same TrackEntry.

5.1.4.1.20. LanguageBCP47 Element

id / type: 0x22B59D / string
path: \Segment\Tracks\TrackEntry\LanguageBCP47
maxOccurs: 1
minver: 4
definition: The language of the track, in the [BCP47] form; see Section 12 on language codes. If this Element is used, then any Language Elements used in the same TrackEntry MUST be ignored.

5.1.4.1.21. CodecID Element

id / type: 0x86 / string
path: \Segment\Tracks\TrackEntry\CodecID
minOccurs / maxOccurs: 1 / 1
definition: An ID corresponding to the codec, see [MatroskaCodec] for more info.

stream copy: True (Section 8)

5.1.4.1.22. CodecPrivate Element

id / type: 0x63A2 / binary
path: \Segment\Tracks\TrackEntry\CodecPrivate
maxOccurs: 1
definition: Private data only known to the codec.

stream copy: True (Section 8)

5.1.4.1.23. CodecName Element

id / type: 0x258688 / utf-8
path: \Segment\Tracks\TrackEntry\CodecName
maxOccurs: 1
definition: A human-readable string specifying the codec.

5.1.4.1.24. AttachmentLink Element

id / type: 0x7446 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\AttachmentLink
maxOccurs: 1
maxver: 3
definition: The UID of an attachment that is used by this codec.
usage notes: The value MUST match the FileUID value of an attachment found in this Segment.

5.1.4.1.25. CodecDelay Element

id / type / default: 0x56AA / uinteger / 0
path: \Segment\Tracks\TrackEntry\CodecDelay
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: CodecDelay is The codec-built-in delay, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1. It represents the amount of codec samples that will be discarded by the decoder during playback. This timestamp value MUST be subtracted from each frame timestamp in order to get the timestamp that will be actually played. The value SHOULD be small so the muxing of tracks with the same actual timestamp are in the same Cluster.

stream copy: True (Section 8)

5.1.4.1.26. SeekPreRoll Element

id / type / default: 0x56BB / uinteger / 0
path: \Segment\Tracks\TrackEntry\SeekPreRoll
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: After a discontinuity, SeekPreRoll is the duration of the data the decoder MUST decode before the decoded data is valid, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1.

stream copy: True (Section 8)

5.1.4.1.27. TrackTranslate Element

id / type: 0x6624 / master
path: \Segment\Tracks\TrackEntry\TrackTranslate
definition: The mapping between this TrackEntry and a track value in the given Chapter Codec.
rationale: Chapter Codec may need to address content in specific

track, but they may not know of the way to identify tracks in Matroska. This element and its child elements add a way to map the internal tracks known to the Chapter Codec to the track IDs in Matroska. This allows remuxing a file with Chapter Codec without changing the content of the codec data, just the track mapping.

5.1.4.1.27.1. TrackTranslateTrackID Element

id / type: 0x66A5 / binary
path: \Segment\Tracks\TrackEntry\TrackTranslate\TrackTranslateTrackID
minOccurs / maxOccurs: 1 / 1
definition: The binary value used to represent this TrackEntry in the chapter codec data. The format depends on the ChapProcessCodecID used; see Section 5.1.7.1.4.15.

5.1.4.1.27.2. TrackTranslateCodec Element

id / type: 0x66BF / uinteger
path: \Segment\Tracks\TrackEntry\TrackTranslate\TrackTranslateCodec
minOccurs / maxOccurs: 1 / 1
definition: This TrackTranslate applies to this chapter codec of the given chapter edition(s); see Section 5.1.7.1.4.15.

defined values:

value	label	definition
0	Matroska Script	Chapter commands using the Matroska Script codec.
1	DVD-menu	Chapter commands using the DVD-like codec.

Table 4: TrackTranslateCodec values

5.1.4.1.27.3. TrackTranslateEditionUID Element

id / type: 0x66FC / uinteger
path: \Segment\Tracks\TrackEntry\TrackTranslate\TrackTranslateEditionUID
definition: Specify a chapter edition UID on which this TrackTranslate applies.
usage notes: When no TrackTranslateEditionUID is specified in the TrackTranslate, the TrackTranslate applies to all chapter editions found in the Segment using the given TrackTranslateCodec.

5.1.4.1.28. Video Element

id / type: 0xE0 / master
path: \Segment\Tracks\TrackEntry\Video
maxOccurs: 1
definition: Video settings.

5.1.4.1.28.1. FlagInterlaced Element

id / type / default: 0x9A / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\FlagInterlaced
minOccurs / maxOccurs: 1 / 1
minver: 2
definition: Specify whether the video frames in this track are interlaced.

defined values:

value	label	definition
0	undetermined	Unknown status.This value SHOULD be avoided.
1	interlaced	Interlaced frames.
2	progressive	No interlacing.

Table 5: FlagInterlaced values

stream copy: True (Section 8)

5.1.4.1.28.2. FieldOrder Element

id / type / default: 0x9D / uinteger / 2
path: \Segment\Tracks\TrackEntry\Video\FieldOrder
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Specify the field ordering of video frames in this track.

defined values:

value	label	definition
0	progressive	Interlaced frames.This value SHOULD be avoided, setting FlagInterlaced to 2 is sufficient.
1	tff	Top field displayed first. Top field stored first.
2	undetermined	Unknown field order.This value SHOULD be avoided.
6	bff	Bottom field displayed first. Bottom field stored first.
9	bff(swapped)	Top field displayed first. Fields are interleaved in storage with the top line of the top field stored first.
14	tff(swapped)	Bottom field displayed first. Fields are interleaved in storage with the top line of the top field stored first.

Table 6: FieldOrder values

usage notes: If FlagInterlaced is not set to 1, this Element MUST be ignored.

stream copy: True (Section 8)

5.1.4.1.28.3. StereoMode Element

id / type / default: 0x53B8 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\StereoMode
minOccurs / maxOccurs: 1 / 1
minver: 3
definition: Stereo-3D video mode. There are some more details in Section 18.10.

restrictions:

value	label
0	mono
1	side by side (left eye first)
2	top - bottom (right eye is first)
3	top - bottom (left eye is first)
4	checkboard (right eye is first)
5	checkboard (left eye is first)
6	row interleaved (right eye is first)
7	row interleaved (left eye is first)
8	column interleaved (right eye is first)
9	column interleaved (left eye is first)
10	anaglyph (cyan/red)
11	side by side (right eye first)
12	anaglyph (green/magenta)
13	both eyes laced in one Block (left eye is first)
14	both eyes laced in one Block (right eye is first)

Table 7: StereoMode values

stream copy: True (Section 8)

5.1.4.1.28.4. AlphaMode Element

id / type / default: 0x53C0 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\AlphaMode
minOccurs / maxOccurs: 1 / 1
minver: 3
definition: Indicate whether the BlockAdditional Element with

BlockAddID of "1" contains Alpha data, as defined by to the Codec Mapping for the CodecID. Undefined values SHOULD NOT be used as the behavior of known implementations is different (considered either as 0 or 1).

defined values:

value	label	definition
0	none	The BlockAdditional Element with BlockAddID of "1" does not exist or SHOULD NOT be considered as containing such data.
1	present	The BlockAdditional Element with BlockAddID of "1" contains alpha channel data.

Table 8: AlphaMode values

stream copy: True (Section 8)

5.1.4.1.28.5. OldStereoMode Element

id / type: 0x53B9 / uinteger
path: \Segment\Tracks\TrackEntry\Video\OldStereoMode
maxOccurs: 1
maxver: 2
definition: Bogus StereoMode value used in old versions of libmatroska.

restrictions:

value	label
0	mono
1	right eye
2	left eye
3	both eyes

Table 9: OldStereoMode values

usage notes: This Element MUST NOT be used. It was an incorrect value used in libmatroska up to 0.9.0.

5.1.4.1.28.6. PixelWidth Element

id / type: 0xB0 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\Video\PixelWidth
minOccurs / maxOccurs: 1 / 1
definition: Width of the encoded video frames in pixels.

stream copy: True (Section 8)

5.1.4.1.28.7. PixelHeight Element

id / type: 0xBA / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\Video\PixelHeight
minOccurs / maxOccurs: 1 / 1
definition: Height of the encoded video frames in pixels.

stream copy: True (Section 8)

5.1.4.1.28.8. PixelCropBottom Element

id / type / default: 0x54AA / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\PixelCropBottom
minOccurs / maxOccurs: 1 / 1
definition: The number of video pixels to remove at the bottom of the image.

stream copy: True (Section 8)

5.1.4.1.28.9. PixelCropTop Element

id / type / default: 0x54BB / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\PixelCropTop
minOccurs / maxOccurs: 1 / 1
definition: The number of video pixels to remove at the top of the image.

stream copy: True (Section 8)

5.1.4.1.28.10. PixelCropLeft Element

id / type / default: 0x54CC / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\PixelCropLeft
minOccurs / maxOccurs: 1 / 1

definition: The number of video pixels to remove on the left of the image.

stream copy: True (Section 8)

5.1.4.1.28.11. PixelCropRight Element

id / type / default: 0x54DD / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\PixelCropRight
minOccurs / maxOccurs: 1 / 1
definition: The number of video pixels to remove on the right of the image.

stream copy: True (Section 8)

5.1.4.1.28.12. DisplayWidth Element

id / type: 0x54B0 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\Video\DisplayWidth
maxOccurs: 1
definition: Width of the video frames to display. Applies to the video frame after cropping (PixelCrop* Elements).

notes:

attribute	note
default	If the DisplayUnit of the same TrackEntry is 0, then the default value for DisplayWidth is equal toPixelWidth - PixelCropLeft - PixelCropRight, else there is no default value.

Table 10: DisplayWidth implementation notes

stream copy: True (Section 8)

5.1.4.1.28.13. DisplayHeight Element

id / type: 0x54BA / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\Video\DisplayHeight
maxOccurs: 1
definition: Height of the video frames to display. Applies to the video frame after cropping (PixelCrop* Elements).

notes:

attribute	note
default	If the DisplayUnit of the same TrackEntry is 0, then the default value for DisplayHeight is equal toPixelHeight - PixelCropTop - PixelCropBottom, else there is no default value.

Table 11: DisplayHeight implementation notes

stream copy: True (Section 8)

5.1.4.1.28.14. DisplayUnit Element

id / type / default: 0x54B2 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\DisplayUnit
minOccurs / maxOccurs: 1 / 1
definition: How DisplayWidth & DisplayHeight are interpreted.

restrictions:

value	label
0	pixels
1	centimeters
2	inches
3	display aspect ratio
4	unknown

Table 12: DisplayUnit values

5.1.4.1.28.15. UncompressedFourCC Element

id / type: 0x2EB524 / binary
length: 4
path: \Segment\Tracks\TrackEntry\Video\UncompressedFourCC
minOccurs / maxOccurs: see implementation notes / 1
definition: Specify the uncompressed pixel format used for the

Track's data as a FourCC. This value is similar in scope to the biCompression value of AVI's BITMAPINFO [AVIFormat]. There is no definitive list of FourCC values, nor an official registry. Some common values for YUV pixel formats can be found at [MSYUV8], [MSYUV16] and [FourCC-YUV]. Some common values for uncompressed RGB pixel formats can be found at [MSRGB] and [FourCC-RGB].

notes:

attribute	note
minOccurs	UncompressedFourCC MUST be set (minOccurs=1) in TrackEntry, when the CodecID Element of the TrackEntry is set to "V_UNCOMPRESSED".

Table 13: UncompressedFourCC implementation notes

stream copy: True (Section 8)

5.1.4.1.28.16. Colour Element

id / type: 0x55B0 / master
path: \Segment\Tracks\TrackEntry\Video\Colour
maxOccurs: 1
minver: 4
definition: Settings describing the colour format.

stream copy: True (Section 8)

5.1.4.1.28.17. MatrixCoefficients Element

id / type / default: 0x55B1 / uinteger / 2
path: \Segment\Tracks\TrackEntry\Video\Colour\MatrixCoefficients
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: The Matrix Coefficients of the video used to derive luma and chroma values from red, green, and blue color primaries. For clarity, the value and meanings for MatrixCoefficients are adopted from Table 4 of [ITU-H.273].

restrictions:

value	label
0	Identity
1	ITU-R BT.709
2	unspecified
3	reserved
4	US FCC 73.682
5	ITU-R BT.470BG
6	SMPTE 170M
7	SMPTE 240M
8	YCoCg
9	BT2020 Non-constant Luminance
10	BT2020 Constant Luminance
11	SMPTE ST 2085
12	Chroma-derived Non-constant Luminance
13	Chroma-derived Constant Luminance
14	ITU-R BT.2100-0

Table 14: MatrixCoefficients values

stream copy: True (Section 8)

5.1.4.1.28.18. BitsPerChannel Element

id / type / default: 0x55B2 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\Colour\BitsPerChannel
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Number of decoded bits per channel. A value of 0 indicates that the BitsPerChannel is unspecified.

stream copy: True (Section 8)

5.1.4.1.28.19. ChromaSubsamplingHorz Element

id / type: 0x55B3 / uinteger
path: \Segment\Tracks\TrackEntry\Video\Colour\ChromaSubsamplingHorz
maxOccurs: 1
minver: 4
definition: The amount of pixels to remove in the Cr and Cb channels for every pixel not removed horizontally. Example: For video with 4:2:0 chroma subsampling, the ChromaSubsamplingHorz SHOULD be set to 1.

stream copy: True (Section 8)

5.1.4.1.28.20. ChromaSubsamplingVert Element

id / type: 0x55B4 / uinteger
path: \Segment\Tracks\TrackEntry\Video\Colour\ChromaSubsamplingVert
maxOccurs: 1
minver: 4
definition: The amount of pixels to remove in the Cr and Cb channels for every pixel not removed vertically. Example: For video with 4:2:0 chroma subsampling, the ChromaSubsamplingVert SHOULD be set to 1.

stream copy: True (Section 8)

5.1.4.1.28.21. CbSubsamplingHorz Element

id / type: 0x55B5 / uinteger
path: \Segment\Tracks\TrackEntry\Video\Colour\CbSubsamplingHorz
maxOccurs: 1
minver: 4
definition: The amount of pixels to remove in the Cb channel for every pixel not removed horizontally. This is additive with ChromaSubsamplingHorz. Example: For video with 4:2:1 chroma subsampling, the ChromaSubsamplingHorz SHOULD be set to 1 and CbSubsamplingHorz SHOULD be set to 1.

stream copy: True (Section 8)

5.1.4.1.28.22. CbSubsamplingVert Element

id / type: 0x55B6 / uinteger
path: \Segment\Tracks\TrackEntry\Video\Colour\CbSubsamplingVert
maxOccurs: 1
minver: 4
definition: The amount of pixels to remove in the Cb channel for

every pixel not removed vertically. This is additive with ChromaSubsamplingVert.

stream copy: True (Section 8)

5.1.4.1.28.23. ChromaSitingHorz Element

id / type / default: 0x55B7 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\Colour\ChromaSitingHorz
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: How chroma is subsampled horizontally.

restrictions:

value	label
0	unspecified
1	left collocated
2	half

Table 15:
ChromaSitingHorz values

stream copy: True (Section 8)

5.1.4.1.28.24. ChromaSitingVert Element

id / type / default: 0x55B8 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\Colour\ChromaSitingVert
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: How chroma is subsampled vertically.

restrictions:

value	label
0	unspecified
1	top collocated
2	half

Table 16:
ChromaSitingVert
values

stream copy: True (Section 8)

5.1.4.1.28.25. Range Element

id / type / default: 0x55B9 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\Colour\Range
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Clipping of the color ranges.

restrictions:

value	label
0	unspecified
1	broadcast range
2	full range (no clipping)
3	defined by MatrixCoefficients / TransferCharacteristics

Table 17: Range values

stream copy: True (Section 8)

5.1.4.1.28.26. TransferCharacteristics Element

id / type / default: 0x55BA / uinteger / 2
path: \Segment\Tracks\TrackEntry\Video\Colour\TransferCharacteristics
minOccurs / maxOccurs: 1 / 1

minver: 4
definition: The transfer characteristics of the video. For clarity, the value and meanings for TransferCharacteristics are adopted from Table 3 of [ITU-H.273].

restrictions:

value	label
0	reserved
1	ITU-R BT.709
2	unspecified
3	reserved2
4	Gamma 2.2 curve - BT.470M
5	Gamma 2.8 curve - BT.470BG
6	SMPTE 170M
7	SMPTE 240M
8	Linear
9	Log
10	Log Sqrt
11	IEC 61966-2-4
12	ITU-R BT.1361 Extended Colour Gamut
13	IEC 61966-2-1
14	ITU-R BT.2020 10 bit
15	ITU-R BT.2020 12 bit
16	ITU-R BT.2100 Perceptual Quantization
17	SMPTE ST 428-1
18	ARIB STD-B67 (HLG)

Table 18: TransferCharacteristics values

stream copy: True (Section 8)

5.1.4.1.28.27. Primaries Element

id / type / default: 0x55BB / uinteger / 2
path: \Segment\Tracks\TrackEntry\Video\Colour\Primaries
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: The colour primaries of the video. For clarity, the value and meanings for Primaries are adopted from Table 2 of [ITU-H.273].

restrictions:

value	label
0	reserved
1	ITU-R BT.709
2	unspecified
3	reserved2
4	ITU-R BT.470M
5	ITU-R BT.470BG - BT.601 625
6	ITU-R BT.601 525 - SMPTE 170M
7	SMPTE 240M
8	FILM
9	ITU-R BT.2020
10	SMPTE ST 428-1
11	SMPTE RP 432-2
12	SMPTE EG 432-2
22	EBU Tech. 3213-E - JEDEC P22 phosphors

Table 19: Primaries values

stream copy: True (Section 8)

5.1.4.1.28.28. MaxCLL Element

id / type: 0x55BC / uinteger
path: \Segment\Tracks\TrackEntry\Video\Colour\MaxCLL
maxOccurs: 1
minver: 4
definition: Maximum brightness of a single pixel (Maximum Content Light Level) in candelas per square meter (cd/m^2).

stream copy: True (Section 8)

5.1.4.1.28.29. MaxFALL Element

id / type: 0x55BD / uinteger
path: \Segment\Tracks\TrackEntry\Video\Colour\MaxFALL
maxOccurs: 1
minver: 4
definition: Maximum brightness of a single full frame (Maximum Frame-Average Light Level) in candelas per square meter (cd/m^2).

stream copy: True (Section 8)

5.1.4.1.28.30. MasteringMetadata Element

id / type: 0x55D0 / master
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata
maxOccurs: 1
minver: 4
definition: SMPTE 2086 mastering data.

stream copy: True (Section 8)

5.1.4.1.28.31. PrimaryRChromaticityX Element

id / type: 0x55D1 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\PrimaryRChromaticityX
maxOccurs: 1
minver: 4
definition: Red X chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.32. PrimaryRChromaticityY Element

id / type: 0x55D2 / float
range: 0x0p+0-0x1p+0

path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\PrimaryRChromaticityY
maxOccurs: 1
minver: 4
definition: Red Y chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.33. PrimaryGChromaticityX Element

id / type: 0x55D3 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\PrimaryGChromaticityX
maxOccurs: 1
minver: 4
definition: Green X chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.34. PrimaryGChromaticityY Element

id / type: 0x55D4 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\PrimaryGChromaticityY
maxOccurs: 1
minver: 4
definition: Green Y chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.35. PrimaryBChromaticityX Element

id / type: 0x55D5 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\PrimaryBChromaticityX
maxOccurs: 1
minver: 4
definition: Blue X chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.36. PrimaryBChromaticityY Element

id / type: 0x55D6 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\PrimaryBChromaticityY
maxOccurs: 1
minver: 4
definition: Blue Y chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.37. WhitePointChromaticityX Element

id / type: 0x55D7 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\WhitePointChromaticityX
maxOccurs: 1
minver: 4
definition: White X chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.38. WhitePointChromaticityY Element

id / type: 0x55D8 / float
range: 0x0p+0-0x1p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\WhitePointChromaticityY
maxOccurs: 1
minver: 4
definition: White Y chromaticity coordinate, as defined by [CIE-1931].

stream copy: True (Section 8)

5.1.4.1.28.39. LuminanceMax Element

id / type: 0x55D9 / float
range: >= 0x0p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\LuminanceMax
maxOccurs: 1
minver: 4
definition: Maximum luminance. Represented in candelas per square

meter (cd/m^2).

stream copy: True (Section 8)

5.1.4.1.28.40. LuminanceMin Element

id / type: 0x55DA / float
range: >= 0x0p+0
path: \Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\LuminanceMin
maxOccurs: 1
minver: 4
definition: Minimum luminance. Represented in candelas per square meter (cd/m^2).

stream copy: True (Section 8)

5.1.4.1.28.41. Projection Element

id / type: 0x7670 / master
path: \Segment\Tracks\TrackEntry\Video\Projection
maxOccurs: 1
minver: 4
definition: Describes the video projection details. Used to render spherical, VR videos or flipping videos horizontally/vertically.

stream copy: True (Section 8)

5.1.4.1.28.42. ProjectionType Element

id / type / default: 0x7671 / uinteger / 0
path: \Segment\Tracks\TrackEntry\Video\Projection\ProjectionType
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Describes the projection used for this video track.

restrictions:

value	label
0	rectangular
1	equirectangular
2	cubemap
3	mesh

Table 20:
ProjectionType values

stream copy: True (Section 8)

5.1.4.1.28.43. ProjectionPrivate Element

id / type: 0x7672 / binary
path: \Segment\Tracks\TrackEntry\Video\Projection\ProjectionPrivate
maxOccurs: 1
minver: 4
definition: Private data that only applies to a specific projection.

* If ProjectionType equals 0 (Rectangular), then this element MUST NOT be present.
* If ProjectionType equals 1 (Equirectangular), then this element MUST be present and contain the same binary data that would be stored inside an ISOBMFF Equirectangular Projection Box ('equi').
* If ProjectionType equals 2 (Cubemap), then this element MUST be present and contain the same binary data that would be stored inside an ISOBMFF Cubemap Projection Box ('cbmp').
* If ProjectionType equals 3 (Mesh), then this element MUST be present and contain the same binary data that would be stored inside an ISOBMFF Mesh Projection Box ('mshp').

usage notes: ISOBMFF box size and fourcc fields are not included in the binary data, but the FullBox version and flag fields are. This is to avoid redundant framing information while preserving versioning and semantics between the two container formats.

stream copy: True (Section 8)

5.1.4.1.28.44. ProjectionPoseYaw Element

id / type / default: 0x7673 / float / 0x0p+0
range: >= -0xB4p+0, <= 0xB4p+0

path: \Segment\Tracks\TrackEntry\Video\Projection\ProjectionPoseYaw
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Specifies a yaw rotation to the projection.

Value represents a clockwise rotation, in degrees, around the up vector. This rotation must be applied before any ProjectionPosePitch or ProjectionPoseRoll rotations. The value of this element MUST be in the -180 to 180 degree range, both included.

Setting ProjectionPoseYaw to 180 or -180 degrees, with the ProjectionPoseRoll and ProjectionPosePitch set to 0 degrees flips the image horizontally.

stream copy: True (Section 8)

5.1.4.1.28.45. ProjectionPosePitch Element

id / type / default: 0x7674 / float / 0x0p+0
range: >= -0x5Ap+0, <= 0x5Ap+0
path: \Segment\Tracks\TrackEntry\Video\Projection\ProjectionPosePitch
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Specifies a pitch rotation to the projection.

Value represents a counter-clockwise rotation, in degrees, around the right vector. This rotation must be applied after the ProjectionPoseYaw rotation and before the ProjectionPoseRoll rotation. The value of this element MUST be in the -90 to 90 degree range, both included.

stream copy: True (Section 8)

5.1.4.1.28.46. ProjectionPoseRoll Element

id / type / default: 0x7675 / float / 0x0p+0
range: >= -0xB4p+0, <= 0xB4p+0
path: \Segment\Tracks\TrackEntry\Video\Projection\ProjectionPoseRoll
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: Specifies a roll rotation to the projection.

Value represents a counter-clockwise rotation, in degrees, around the forward vector. This rotation must be applied after the ProjectionPoseYaw and ProjectionPosePitch rotations. The value of this element MUST be in the -180 to 180 degree range, both included.

Setting ProjectionPoseRoll to 180 or -180 degrees, the ProjectionPoseYaw to 180 or -180 degrees with ProjectionPosePitch set to 0 degrees flips the image vertically.

Setting ProjectionPoseRoll to 180 or -180 degrees, with the ProjectionPoseYaw and ProjectionPosePitch set to 0 degrees flips the image horizontally and vertically.

stream copy: True (Section 8)

5.1.4.1.29. Audio Element

id / type: 0xE1 / master
path: \Segment\Tracks\TrackEntry\Audio
maxOccurs: 1
definition: Audio settings.

5.1.4.1.29.1. SamplingFrequency Element

id / type / default: 0xB5 / float / 0x1.f4p+12
range: > 0x0p+0
path: \Segment\Tracks\TrackEntry\Audio\SamplingFrequency
minOccurs / maxOccurs: 1 / 1
definition: Sampling frequency in Hz.

stream copy: True (Section 8)

5.1.4.1.29.2. OutputSamplingFrequency Element

id / type: 0x78B5 / float
range: > 0x0p+0
path: \Segment\Tracks\TrackEntry\Audio\OutputSamplingFrequency
maxOccurs: 1
definition: Real output sampling frequency in Hz (used for SBR techniques).

notes:

attribute	note
default	The default value for OutputSamplingFrequency of the same TrackEntry is equal to the SamplingFrequency.

Table 21: OutputSamplingFrequency implementation notes

5.1.4.1.29.3. Channels Element

id / type / default: 0x9F / uinteger / 1
range: not 0
path: \Segment\Tracks\TrackEntry\Audio\Channels
minOccurs / maxOccurs: 1 / 1
definition: Numbers of channels in the track.

stream copy: True (Section 8)

5.1.4.1.29.4. BitDepth Element

id / type: 0x6264 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\Audio\BitDepth
maxOccurs: 1
definition: Bits per sample, mostly used for PCM.

stream copy: True (Section 8)

5.1.4.1.30. TrackOperation Element

id / type: 0xE2 / master
path: \Segment\Tracks\TrackEntry\TrackOperation
maxOccurs: 1
minver: 3
definition: Operation that needs to be applied on tracks to create this virtual track. For more details look at Section 18.8.

stream copy: True (Section 8)

5.1.4.1.30.1. TrackCombinePlanes Element

id / type: 0xE3 / master
path: \Segment\Tracks\TrackEntry\TrackOperation\TrackCombinePlanes
maxOccurs: 1
minver: 3
definition: Contains the list of all video plane tracks that need to be combined to create this 3D track

stream copy: True (Section 8)

5.1.4.1.30.2. TrackPlane Element

id / type: 0xE4 / master
path: \Segment\Tracks\TrackEntry\TrackOperation\TrackCombinePlanes\TrackPlane
minOccurs: 1

minver: 3
definition: Contains a video plane track that need to be combined to create this 3D track

stream copy: True (Section 8)

5.1.4.1.30.3. TrackPlaneUID Element

id / type: 0xE5 / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\TrackOperation\TrackCombinePlanes\TrackPlane\TrackPlaneUID
minOccurs / maxOccurs: 1 / 1
minver: 3
definition: The trackUID number of the track representing the plane.

stream copy: True (Section 8)

5.1.4.1.30.4. TrackPlaneType Element

id / type: 0xE6 / uinteger
path: \Segment\Tracks\TrackEntry\TrackOperation\TrackCombinePlanes\TrackPlane\TrackPlaneType
minOccurs / maxOccurs: 1 / 1
minver: 3
definition: The kind of plane this track corresponds to.

restrictions:

value	label
0	left eye
1	right eye
2	background

Table 22:
TrackPlaneType values

stream copy: True (Section 8)

5.1.4.1.30.5. TrackJoinBlocks Element

id / type: 0xE9 / master
path: \Segment\Tracks\TrackEntry\TrackOperation\TrackJoinBlocks

maxOccurs: 1
minver: 3
definition: Contains the list of all tracks whose Blocks need to be combined to create this virtual track

stream copy: True (Section 8)

5.1.4.1.30.6. TrackJoinUID Element

id / type: 0xED / uinteger
range: not 0
path: \Segment\Tracks\TrackEntry\TrackOperation\TrackJoinBlocks\TrackJoinUID
minOccurs: 1
minver: 3
definition: The trackUID number of a track whose blocks are used to create this virtual track.

stream copy: True (Section 8)

5.1.4.1.31. ContentEncodings Element

id / type: 0x6D80 / master
path: \Segment\Tracks\TrackEntry\ContentEncodings
maxOccurs: 1
definition: Settings for several content encoding mechanisms like compression or encryption.

stream copy: True (Section 8)

5.1.4.1.31.1. ContentEncoding Element

id / type: 0x6240 / master
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding
minOccurs: 1
definition: Settings for one content encoding like compression or encryption.

stream copy: True (Section 8)

5.1.4.1.31.2. ContentEncodingOrder Element

id / type / default: 0x5031 / uinteger / 0
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncodingOrder
minOccurs / maxOccurs: 1 / 1
definition: Tell in which order to apply each ContentEncoding of the

ContentEncodings. The decoder/demuxer MUST start with the ContentEncoding with the highest ContentEncodingOrder and work its way down to the ContentEncoding with the lowest ContentEncodingOrder. This value MUST be unique over for each ContentEncoding found in the ContentEncodings of this TrackEntry.

stream copy: True (Section 8)

5.1.4.1.31.3. ContentEncodingScope Element

id / type / default: 0x5032 / uinteger / 1
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncodingScope
minOccurs / maxOccurs: 1 / 1
definition: A bit field that describes which Elements have been modified in this way. Values (big-endian) can be OR'ed.

defined values:

value	label	definition
1	Block	All frame contents, excluding lacing data.
2	Private	The track's CodecPrivate data.
4	Next	The next ContentEncoding (next ContentEncodingOrder. Either the data inside ContentCompression and/or ContentEncryption).This value SHOULD NOT be used as it's not supported by players.

Table 23: ContentEncodingScope values

stream copy: True (Section 8)

5.1.4.1.31.4. ContentEncodingType Element

id / type / default: 0x5033 / uinteger / 0
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncodingType
minOccurs / maxOccurs: 1 / 1
definition: A value describing what kind of transformation is applied.

restrictions:

value	label
0	Compression
1	Encryption

Table 24:
ContentEncodingType
values

stream copy: True (Section 8)

5.1.4.1.31.5. ContentCompression Element

id / type: 0x5034 / master
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentCompression
maxOccurs: 1
definition: Settings describing the compression used. This Element MUST be present if the value of ContentEncodingType is 0 and absent otherwise. Each block MUST be decompressable even if no previous block is available in order not to prevent seeking.

stream copy: True (Section 8)

5.1.4.1.31.6. ContentCompAlgo Element

id / type / default: 0x4254 / uinteger / 0
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentCompression\ContentCompAlgo
minOccurs / maxOccurs: 1 / 1
definition: The compression algorithm used.

defined values:

value	label	definition
0	zlib	zlib compression [RFC1950].
1	bzlib	bzip2 compression [BZIP2], SHOULD NOT be used; see usage notes.
2	lzo1x	Lempel-Ziv-Oberhumer compression [LZO], SHOULD NOT be used; see usage notes.
3	Header Stripping	Octets in ContentCompSettings (Section 5.1.4.1.31.7) have been stripped from each frame.

Table 25: ContentCompAlgo values

usage notes: Compression method "1" (bzlib) and "2" (lzo1x) are lacking proper documentation on the format which limits implementation possibilities. Due to licensing conflicts on commonly available libraries compression methods "2" (lzo1x) does not offer widespread interoperability. A Matroska Writer SHOULD NOT use these compression methods by default. A Matroska Reader MAY support methods "1" and "2" as possible, and SHOULD support other methods.

stream copy: True (Section 8)

5.1.4.1.31.7. ContentCompSettings Element

id / type: 0x4255 / binary
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentCompression\ContentCompSettings
maxOccurs: 1
definition: Settings that might be needed by the decompressor. For Header Stripping (ContentCompAlgo=3), the bytes that were removed from the beginning of each frames of the track.

stream copy: True (Section 8)

5.1.4.1.31.8. ContentEncryption Element

id / type: 0x5035 / master
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption
maxOccurs: 1
definition: Settings describing the encryption used. This Element

MUST be present if the value of ContentEncodingType is 1 (encryption) and MUST be ignored otherwise. A Matroska Player MAY support encryption.

stream copy: True (Section 8)

5.1.4.1.31.9. ContentEncAlgo Element

id / type / default: 0x47E1 / uinteger / 0
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentEncAlgo
minOccurs / maxOccurs: 1 / 1
definition: The encryption algorithm used.

defined values:

value	label	definition
0	Not encrypted	The data are not encrypted.
1	DES	Data Encryption Standard (DES) [FIPS.46-3].This value SHOULD be avoided.
2	3DES	Triple Data Encryption Algorithm [SP.800-67].This value SHOULD be avoided.
3	Twofish	Twofish Encryption Algorithm [Twofish].
4	Blowfish	Blowfish Encryption Algorithm [Blowfish].This value SHOULD be avoided.
5	AES	Advanced Encryption Standard (AES) [FIPS.197].

Table 26: ContentEncAlgo values

stream copy: True (Section 8)

5.1.4.1.31.10. ContentEncKeyID Element

id / type: 0x47E2 / binary
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentEncKeyID
maxOccurs: 1
definition: For public key algorithms this is the ID of the public

key the data was encrypted with.

stream copy: True (Section 8)

5.1.4.1.31.11. ContentEncAESSettings Element

id / type: 0x47E7 / master
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentEncAESSettings
maxOccurs: 1
minver: 4
definition: Settings describing the encryption algorithm used.

notes:

attribute	note
maxOccurs	ContentEncAESSettings MUST NOT be set (maxOccurs=0) if ContentEncAlgo is not AES (5).

Table 27: ContentEncAESSettings implementation notes

stream copy: True (Section 8)

5.1.4.1.31.12. AESSettingsCipherMode Element

id / type: 0x47E8 / uinteger
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentEncAESSettings\AESSettingsCipherMode
minOccurs / maxOccurs: 1 / 1
minver: 4
definition: The AES cipher mode used in the encryption.

defined values:

value	label	definition
1	AES-CTR	Counter [SP.800-38A].
2	AES-CBC	Cipher Block Chaining [SP.800-38A].

Table 28: AESSettingsCipherMode values

notes:

attribute	note
maxOccurs	AESSettingsCipherMode MUST NOT be set (maxOccurs=0) if ContentEncAlgo is not AES (5).

Table 29: AESSettingsCipherMode implementation notes

stream copy: True (Section 8)

5.1.5. Cues Element

id / type: 0x1C53BB6B / master
path: \Segment\Cues
minOccurs / maxOccurs: see implementation notes / 1
definition: A Top-Level Element to speed seeking access. All entries are local to the Segment.

notes:

attribute	note
minOccurs	This Element SHOULD be set when the Segment is not transmitted as a live stream; see Section 23.2.

Table 30: Cues implementation notes

5.1.5.1. CuePoint Element

id / type: 0xBB / master
path: \Segment\Cues\CuePoint
minOccurs: 1
definition: Contains all information relative to a seek point in the Segment.

5.1.5.1.1. CueTime Element

id / type: 0xB3 / uinteger
path: \Segment\Cues\CuePoint\CueTime
minOccurs / maxOccurs: 1 / 1
definition: Absolute timestamp of the seek point, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1.

5.1.5.1.2. CueTrackPositions Element

id / type: 0xB7 / master
path: \Segment\Cues\CuePoint\CueTrackPositions
minOccurs: 1
definition: Contain positions for different tracks corresponding to the timestamp.

5.1.5.1.2.1. CueTrack Element

id / type: 0xF7 / uinteger
range: not 0
path: \Segment\Cues\CuePoint\CueTrackPositions\CueTrack
minOccurs / maxOccurs: 1 / 1
definition: The track for which a position is given.

5.1.5.1.2.2. CueClusterPosition Element

id / type: 0xF1 / uinteger
path: \Segment\Cues\CuePoint\CueTrackPositions\CueClusterPosition
minOccurs / maxOccurs: 1 / 1
definition: The Segment Position (Section 16) of the Cluster containing the associated Block.

5.1.5.1.2.3. CueRelativePosition Element

id / type: 0xF0 / uinteger
path: \Segment\Cues\CuePoint\CueTrackPositions\CueRelativePosition
maxOccurs: 1
minver: 4
definition: The relative position inside the Cluster of the referenced SimpleBlock or BlockGroup with 0 being the first possible position for an Element inside that Cluster.

5.1.5.1.2.4. CueDuration Element

id / type: 0xB2 / uinteger
path: \Segment\Cues\CuePoint\CueTrackPositions\CueDuration
maxOccurs: 1
minver: 4
definition: The duration of the block, expressed in Segment Ticks which is based on TimestampScale; see Section 11.1. If missing, the track's DefaultDuration does not apply and no duration information is available in terms of the cues.

5.1.5.1.2.5. CueBlockNumber Element

id / type: 0x5378 / uinteger

range: not 0
path: \Segment\Cues\CuePoint\CueTrackPositions\CueBlockNumber
maxOccurs: 1
definition: Number of the Block in the specified Cluster.

5.1.5.1.2.6. CueCodecState Element

id / type / default: 0xEA / uinteger / 0
path: \Segment\Cues\CuePoint\CueTrackPositions\CueCodecState
minOccurs / maxOccurs: 1 / 1
minver: 2
definition: The Segment Position (Section 16) of the Codec State corresponding to this Cue Element. 0 means that the data is taken from the initial Track Entry.

5.1.5.1.2.7. CueReference Element

id / type: 0xDB / master
path: \Segment\Cues\CuePoint\CueTrackPositions\CueReference
minver: 2
definition: The Clusters containing the referenced Blocks.

5.1.5.1.2.8. CueRefTime Element

id / type: 0x96 / uinteger
path: \Segment\Cues\CuePoint\CueTrackPositions\CueReference\CueRefTime
minOccurs / maxOccurs: 1 / 1
minver: 2
definition: Timestamp of the referenced Block, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1.

5.1.6. Attachments Element

id / type: 0x1941A469 / master
path: \Segment\Attachments
maxOccurs: 1
definition: Contain attached files.

5.1.6.1. AttachedFile Element

id / type: 0x61A7 / master
path: \Segment\Attachments\AttachedFile
minOccurs: 1
definition: An attached file.

5.1.6.1.1. FileDescription Element

id / type: 0x467E / utf-8
path: \Segment\Attachments\AttachedFile\FileDescription
maxOccurs: 1
definition: A human-friendly name for the attached file.

5.1.6.1.2. FileName Element

id / type: 0x466E / utf-8
path: \Segment\Attachments\AttachedFile\FileName
minOccurs / maxOccurs: 1 / 1
definition: Filename of the attached file.

5.1.6.1.3. FileMediaType Element

id / type: 0x4660 / string
path: \Segment\Attachments\AttachedFile\FileMediaType
minOccurs / maxOccurs: 1 / 1
definition: Media type of the file following the [RFC6838] format.

stream copy: True (Section 8)

5.1.6.1.4. FileData Element

id / type: 0x465C / binary
path: \Segment\Attachments\AttachedFile\FileData
minOccurs / maxOccurs: 1 / 1
definition: The data of the file.

stream copy: True (Section 8)

5.1.6.1.5. FileUID Element

id / type: 0x46AE / uinteger
range: not 0
path: \Segment\Attachments\AttachedFile\FileUID
minOccurs / maxOccurs: 1 / 1
definition: Unique ID representing the file, as random as possible.

stream copy: True (Section 8)

5.1.7. Chapters Element

id / type: 0x1043A770 / master
path: \Segment\Chapters
maxOccurs: 1

recurring: True

definition: A system to define basic menus and partition data. For more detailed information, look at the Chapters explanation in Section 20.

5.1.7.1. EditionEntry Element

id / type: 0x45B9 / master
path: \Segment\Chapters\EditionEntry
minOccurs: 1
definition: Contains all information about a Segment edition.

5.1.7.1.1. EditionUID Element

id / type: 0x45BC / uinteger
range: not 0
path: \Segment\Chapters\EditionEntry\EditionUID
maxOccurs: 1
definition: A unique ID to identify the edition. It's useful for tagging an edition.

stream copy: True (Section 8)

5.1.7.1.2. EditionFlagDefault Element

id / type / default: 0x45DB / uinteger / 0
range: 0-1
path: \Segment\Chapters\EditionEntry\EditionFlagDefault
minOccurs / maxOccurs: 1 / 1
definition: Set to 1 if the edition SHOULD be used as the default one.

5.1.7.1.3. EditionFlagOrdered Element

id / type / default: 0x45DD / uinteger / 0
range: 0-1
path: \Segment\Chapters\EditionEntry\EditionFlagOrdered
minOccurs / maxOccurs: 1 / 1
definition: Set to 1 if the chapters can be defined multiple times and the order to play them is enforced; see Section 20.1.3.

5.1.7.1.4. ChapterAtom Element

id / type: 0xB6 / master
path: \Segment\Chapters\EditionEntry\+ChapterAtom
minOccurs: 1

recursive: True

definition: Contains the atom information to use as the chapter atom (apply to all tracks).

5.1.7.1.4.1. ChapterUID Element

id / type: 0x73C4 / uinteger
range: not 0
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterUID
minOccurs / maxOccurs: 1 / 1
definition: A unique ID to identify the Chapter.

stream copy: True (Section 8)

5.1.7.1.4.2. ChapterStringUID Element

id / type: 0x5654 / utf-8
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterStringUID
maxOccurs: 1
minver: 3
definition: A unique string ID to identify the Chapter. For example it is used as the storage for [WebVTT] cue identifier values.

5.1.7.1.4.3. ChapterTimeStart Element

id / type: 0x91 / uinteger
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterTimeStart
minOccurs / maxOccurs: 1 / 1
definition: Timestamp of the start of Chapter, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1.

5.1.7.1.4.4. ChapterTimeEnd Element

id / type: 0x92 / uinteger
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterTimeEnd
minOccurs / maxOccurs: see implementation notes / 1
definition: Timestamp of the end of Chapter timestamp excluded, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1. The value MUST be greater than or equal to the ChapterTimeStart of the same ChapterAtom.
usage notes: The ChapterTimeEnd timestamp value being excluded, it MUST take in account the duration of the last frame it includes, especially for the ChapterAtom using the last frames of the Segment.

notes:

attribute	note
minOccurs	ChapterTimeEnd MUST be set (minOccurs=1) if the Edition is an ordered edition; see Section 20.1.3, unless it's a Parent Chapter; see Section 20.2.3

Table 31: ChapterTimeEnd implementation notes

5.1.7.1.4.5. ChapterFlagHidden Element

id / type / default: 0x98 / uinteger / 0
range: 0-1
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterFlagHidden
minOccurs / maxOccurs: 1 / 1
definition: Set to 1 if a chapter is hidden. Hidden chapters SHOULD NOT be available to the user interface (but still to Control Tracks; see Section 20.2.5 on Chapter flags).

5.1.7.1.4.6. ChapterSegmentUUID Element

id / type: 0x6E67 / binary
length: 16
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterSegmentUUID
minOccurs / maxOccurs: see implementation notes / 1
definition: The SegmentUUID of another Segment to play during this chapter.
usage notes: The value MUST NOT be the SegmentUUID value of the Segment it belongs to.

notes:

attribute	note
minOccurs	ChapterSegmentUUID MUST be set (minOccurs=1) if ChapterSegmentEditionUID is used; see Section 17.2 on medium-linking Segments.

Table 32: ChapterSegmentUUID implementation notes

5.1.7.1.4.7. ChapterSegmentEditionUID Element

id / type: 0x6EBC / uinteger
range: not 0
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterSegmentEdit

ionUID
maxOccurs: 1
definition: The EditionUID to play from the Segment linked in ChapterSegmentUUID. If ChapterSegmentEditionUID is undeclared, then no Edition of the linked Segment is used; see Section 17.2 on medium-linking Segments.

5.1.7.1.4.8. ChapterPhysicalEquiv Element

id / type: 0x63C3 / uinteger
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterPhysicalEquiv
maxOccurs: 1
definition: Specify the physical equivalent of this ChapterAtom like "DVD" (60) or "SIDE" (50); see Section 20.4 for a complete list of values.

5.1.7.1.4.9. ChapterDisplay Element

id / type: 0x80 / master
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterDisplay
definition: Contains all possible strings to use for the chapter display.

5.1.7.1.4.10. ChapString Element

id / type: 0x85 / utf-8
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterDisplay\ChapString
minOccurs / maxOccurs: 1 / 1
definition: Contains the string to use as the chapter atom.

5.1.7.1.4.11. ChapLanguage Element

id / type / default: 0x437C / string / eng
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterDisplay\ChapLanguage
minOccurs: 1
definition: A language corresponding to the string, in the Matroska languages form; see Section 12 on language codes. This Element MUST be ignored if a ChapLanguageBCP47 Element is used within the same ChapterDisplay Element.

5.1.7.1.4.12. ChapLanguageBCP47 Element

id / type: 0x437D / string
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterDisplay\ChapLanguageBCP47

minver: 4
definition: A language corresponding to the ChapString, in the [BCP47] form; see Section 12 on language codes. If a ChapLanguageBCP47 Element is used, then any ChapLanguage and ChapCountry Elements used in the same ChapterDisplay MUST be ignored.

5.1.7.1.4.13. ChapCountry Element

id / type: 0x437E / string
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapterDisplay\ChapCountry
definition: A country corresponding to the string, in the Matroska countries form; see Section 13 on country codes. This Element MUST be ignored if a ChapLanguageBCP47 Element is used within the same ChapterDisplay Element.

5.1.7.1.4.14. ChapProcess Element

id / type: 0x6944 / master
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapProcess
definition: Contains all the commands associated to the Atom.

5.1.7.1.4.15. ChapProcessCodecID Element

id / type / default: 0x6955 / uinteger / 0
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapProcess\ChapProcessCodecID
minOccurs / maxOccurs: 1 / 1
definition: Contains the type of the codec used for the processing. A value of 0 means built-in Matroska processing (to be defined), a value of 1 means the DVD command set is used; see Section 20.3 on DVD menus. More codec IDs can be added later.

5.1.7.1.4.16. ChapProcessPrivate Element

id / type: 0x450D / binary
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapProcess\ChapProcessPrivate
maxOccurs: 1
definition: Some optional data attached to the ChapProcessCodecID information. For ChapProcessCodecID = 1, it is the "DVD level" equivalent; see Section 20.3 on DVD menus.

5.1.7.1.4.17. ChapProcessCommand Element

id / type: 0x6911 / master
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapProcess\ChapPr

ocessCommand
definition: Contains all the commands associated to the Atom.

5.1.7.1.4.18. ChapProcessTime Element

id / type: 0x6922 / uinteger
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapProcess\ChapPr ocessCommand\ChapProcessTime
minOccurs / maxOccurs: 1 / 1
definition: Defines when the process command SHOULD be handled

restrictions:

value	label
0	during the whole chapter
1	before starting playback
2	after playback of the chapter

Table 33: ChapProcessTime values

5.1.7.1.4.19. ChapProcessData Element

id / type: 0x6933 / binary
path: \Segment\Chapters\EditionEntry\+ChapterAtom\ChapProcess\ChapPr ocessCommand\ChapProcessData
minOccurs / maxOccurs: 1 / 1
definition: Contains the command information. The data SHOULD be interpreted depending on the ChapProcessCodecID value. For ChapProcessCodecID = 1, the data correspond to the binary DVD cell pre/post commands; see Section 20.3 on DVD menus.

5.1.8. Tags Element

id / type: 0x1254C367 / master
path: \Segment\Tags
definition: Element containing metadata describing Tracks, Editions, Chapters, Attachments, or the Segment as a whole. A list of valid tags can be found in [MatroskaTags].

5.1.8.1. Tag Element

id / type: 0x7373 / master
path: \Segment\Tags\Tag

minOccurs: 1
definition: A single metadata descriptor.

5.1.8.1.1. Targets Element

id / type: 0x63C0 / master
path: \Segment\Tags\Tag\Targets
minOccurs / maxOccurs: 1 / 1
definition: Specifies which other elements the metadata represented by the Tag applies to. If empty or omitted, then the Tag describes everything in the Segment.

5.1.8.1.1.1. TargetTypeValue Element

id / type / default: 0x68CA / uinteger / 50
path: \Segment\Tags\Tag\Targets\TargetTypeValue
minOccurs / maxOccurs: 1 / 1
definition: A number to indicate the logical level of the target.

defined values:

value	label	definition
70	COLLECTION	The highest hierarchical level that tags can describe.
60	EDITION / ISSUE / VOLUME / OPUS / SEASON / SEQUEL	A list of lower levels grouped together.
50	ALBUM / OPERA / CONCERT / MOVIE / EPISODE	The most common grouping level of music and video (equals to an episode for TV series).
40	PART / SESSION	When an album or episode has different logical parts.
30	TRACK / SONG / CHAPTER	The common parts of an album or movie.
20	SUBTRACK / MOVEMENT / SCENE	Corresponds to parts of a track for audio like a movement, or a scene in a movie.
10	SHOT	The lowest hierarchy found in music or movies.

Table 34: TargetTypeValue values

5.1.8.1.1.2. TargetType Element

id / type: 0x63CA / string
path: \Segment\Tags\Tag\Targets\TargetType
maxOccurs: 1
definition: An informational string that can be used to display the logical level of the target like "ALBUM", "TRACK", "MOVIE", "CHAPTER", etc.

restrictions:

value	label
COLLECTION	TargetTypeValue 70
EDITION	TargetTypeValue 60
ISSUE	TargetTypeValue 60
VOLUME	TargetTypeValue 60
OPUS	TargetTypeValue 60
SEASON	TargetTypeValue 60
SEQUEL	TargetTypeValue 60
ALBUM	TargetTypeValue 50
OPERA	TargetTypeValue 50
CONCERT	TargetTypeValue 50
MOVIE	TargetTypeValue 50
EPISODE	TargetTypeValue 50
PART	TargetTypeValue 40
SESSION	TargetTypeValue 40
TRACK	TargetTypeValue 30
SONG	TargetTypeValue 30
CHAPTER	TargetTypeValue 30
SUBTRACK	TargetTypeValue 20
MOVEMENT	TargetTypeValue 20
SCENE	TargetTypeValue 20
SHOT	TargetTypeValue 10

Table 35: TargetType values

5.1.8.1.1.3. TagTrackUID Element

id / type / default: 0x63C5 / uinteger / 0
path: \Segment\Tags\Tag\Targets\TagTrackUID
definition: A unique ID to identify the Track(s) the tags belong to.
usage notes: If the value is 0 at this level, the tags apply to all tracks in the Segment. If set to any other value, it MUST match the TrackUID value of a track found in this Segment.

5.1.8.1.1.4. TagEditionUID Element

id / type / default: 0x63C9 / uinteger / 0
path: \Segment\Tags\Tag\Targets\TagEditionUID
definition: A unique ID to identify the EditionEntry(s) the tags belong to.
usage notes: If the value is 0 at this level, the tags apply to all editions in the Segment. If set to any other value, it MUST match the EditionUID value of an edition found in this Segment.

5.1.8.1.1.5. TagChapterUID Element

id / type / default: 0x63C4 / uinteger / 0
path: \Segment\Tags\Tag\Targets\TagChapterUID
definition: A unique ID to identify the Chapter(s) the tags belong to.
usage notes: If the value is 0 at this level, the tags apply to all chapters in the Segment. If set to any other value, it MUST match the ChapterUID value of a chapter found in this Segment.

5.1.8.1.1.6. TagAttachmentUID Element

id / type / default: 0x63C6 / uinteger / 0
path: \Segment\Tags\Tag\Targets\TagAttachmentUID
definition: A unique ID to identify the Attachment(s) the tags belong to.
usage notes: If the value is 0 at this level, the tags apply to all the attachments in the Segment. If set to any other value, it MUST match the FileUID value of an attachment found in this Segment.

5.1.8.1.2. SimpleTag Element

id / type: 0x67C8 / master
path: \Segment\Tags\Tag\+SimpleTag
minOccurs: 1

recursive: True

definition: Contains general information about the target.

5.1.8.1.2.1. TagName Element

id / type: 0x45A3 / utf-8
path: \Segment\Tags\Tag\+SimpleTag\TagName
minOccurs / maxOccurs: 1 / 1
definition: The name of the Tag that is going to be stored.

5.1.8.1.2.2. TagLanguage Element

id / type / default: 0x447A / string / und
path: \Segment\Tags\Tag\+SimpleTag\TagLanguage
minOccurs / maxOccurs: 1 / 1
definition: Specifies the language of the tag specified, in the Matroska languages form; see Section 12 on language codes. This Element MUST be ignored if the TagLanguageBCP47 Element is used within the same SimpleTag Element.

5.1.8.1.2.3. TagLanguageBCP47 Element

id / type: 0x447B / string
path: \Segment\Tags\Tag\+SimpleTag\TagLanguageBCP47
maxOccurs: 1
minver: 4
definition: The language used in the TagString, in the [BCP47] form; see Section 12 on language codes. If this Element is used, then any TagLanguage Elements used in the same SimpleTag MUST be ignored.

5.1.8.1.2.4. TagDefault Element

id / type / default: 0x4484 / uinteger / 1
range: 0-1
path: \Segment\Tags\Tag\+SimpleTag\TagDefault
minOccurs / maxOccurs: 1 / 1
definition: A boolean value to indicate if this is the default/original language to use for the given tag.

5.1.8.1.2.5. TagString Element

id / type: 0x4487 / utf-8
path: \Segment\Tags\Tag\+SimpleTag\TagString
maxOccurs: 1
definition: The value of the Tag.

5.1.8.1.2.6. TagBinary Element

id / type: 0x4485 / binary
path: \Segment\Tags\Tag\+SimpleTag\TagBinary
maxOccurs: 1
definition: The values of the Tag, if it is binary. Note that this cannot be used in the same SimpleTag as TagString.

6. Matroska Element Ordering

Except for the EBML Header and the CRC-32 Element, the EBML specification does not require any particular storage order for Elements. This specification however defines mandates and recommendations for ordering certain Elements in order to facilitate better playback, seeking, and editing efficiency. This section describes and offers rationale for ordering requirements and recommendations for Matroska.

6.1. Top-Level Elements

The Info Element is the only REQUIRED Top-Level Element in a Matroska file. To be playable, Matroska MUST also contain at least one Tracks Element and Cluster Element. The first Info Element and the first Tracks Element MUST either be stored before the first Cluster Element or both SHALL be referenced by a SeekHead Element occurring before the first Cluster Element.

All Top-Level Elements MUST use a 4-octet long EBML Element ID.

When using Medium Linking, chapters are used to reference other Segments to play in a given order Section 17.2. A Segment containing these linked Chapters does not require a Track Element or a Cluster Element.

It is possible to edit a Matroska file after it has been created. For example, chapters, tags, or attachments can be added. When new Top-Level Elements are added to a Matroska file, the SeekHead Element(s) MUST be updated so that the SeekHead Element(s) itemize the identity and position of all Top-Level Elements.

Editing, removing, or adding Elements to a Matroska file often requires that some existing Elements be voided or extended. Transforming the existing Elements into Void Elements as padding can be used as a method to avoid moving large amounts of data around.

6.2. CRC-32

As noted by the EBML specification, if a CRC-32 Element is used, then the CRC-32 Element MUST be the first ordered Element within its Parent Element.

In Matroska all Top-Level Elements of an EBML Document SHOULD include a CRC-32 Element as their first Child Element. The Segment Element, which is the Root Element, SHOULD NOT have a CRC-32 Element.

6.3. SeekHead

If used, the first SeekHead Element MUST be the first non-CRC-32 Child Element of the Segment Element. If a second SeekHead Element is used, then the first SeekHead Element MUST reference the identity and position of the second SeekHead.

Additionally, the second SeekHead Element MUST only reference Cluster Elements and not any other Top-Level Element already contained within the first SeekHead Element.

The second SeekHead Element MAY be stored in any order relative to the other Top-Level Elements. Whether one or two SeekHead Element(s) are used, the SeekHead Element(s) MUST collectively reference the identity and position of all Top-Level Elements except for the first SeekHead Element.

6.4. Cues (index)

The Cues Element is RECOMMENDED to optimize seeking access in Matroska. It is programmatically simpler to add the Cues Element after all Cluster Elements have been written because this does not require a prediction of how much space to reserve before writing the Cluster Elements. However, storing the Cues Element before the Cluster Elements can provide some seeking advantages. If the Cues Element is present, then it SHOULD either be stored before the first Cluster Element or be referenced by a SeekHead Element.

6.5. Info

The first Info Element SHOULD occur before the first Tracks Element and first Cluster Element except when referenced by a SeekHead Element.

6.6. Chapters Element

The Chapters Element SHOULD be placed before the Cluster Element(s). The Chapters Element can be used during playback even if the user does not need to seek. It immediately gives the user information about what section is being read and what other sections are available. In the case of Ordered Chapters it is RECOMMENDED to evaluate the logical linking even before playing. The Chapters Element SHOULD be placed before the first Tracks Element and after the first Info Element.

6.7. Attachments

The Attachments Element is not intended to be used by default when playing the file, but could contain information relevant to the content, such as cover art or fonts. Cover art is useful even before the file is played and fonts could be needed before playback starts for initialization of subtitles. The Attachments Element MAY be placed before the first Cluster Element; however, if the Attachments Element is likely to be edited, then it SHOULD be placed after the last Cluster Element.

6.8. Tags

The Tags Element is most subject to changes after the file was originally created. For easier editing, the Tags Element can be placed at the end of the Segment Element, even after the Attachments Element. On the other hand, it is inconvenient to have to seek in the Segment for tags, especially for network streams. So it’s better if the Tags Element is found early in the stream. When editing the Tags Element, the original Tags Element at the beginning can be overwritten with a Void Element and a new Tags Element written at the end of the Segment Element. The file and Segment sizes will only marginally change.

7. Matroska versioning

Matroska is based upon the principle that a reading application does not have to support 100% of the specifications in order to be able to play the file. A Matroska file therefore contains version indicators that tell a reading application what to expect.

It is possible and valid to have the version fields indicate that the file contains Matroska Elements from a higher specification version number while signaling that a reading application MUST only support a lower version number properly in order to play it back (possibly with a reduced feature set).

The EBML Header of each Matroska document informs the reading application on what version of Matroska to expect. The Elements within EBML Header with jurisdiction over this information are DocTypeVersion and DocTypeReadVersion.

DocTypeVersion MUST be equal to or greater than the highest Matroska version number of any Element present in the Matroska file. For example, a file using the SimpleBlock Element (Section 5.1.3.4) MUST have a DocTypeVersion equal to or greater than 2. A file containing CueRelativePosition Elements (Section 5.1.5.1.2.3) MUST have a DocTypeVersion equal to or greater than 4.

The DocTypeReadVersion MUST contain the minimum version number that a reading application can minimally support in order to play the file back -- optionally with a reduced feature set. For example, if a file contains only Elements of version 2 or lower except for CueRelativePosition (which is a version 4 Matroska Element), then DocTypeReadVersion SHOULD still be set to 2 and not 4 because evaluating CueRelativePosition is not necessary for standard playback -- it makes seeking more precise if used.

A reading application supporting Matroska version V MUST NOT refuse to read a file with DocReadTypeVersion equal to or lower than V even if DocTypeVersion is greater than V.

A reading application supporting at least Matroska version V reading a file whose DocTypeReadVersion field is equal to or lower than V MUST skip Matroska/EBML Elements it encounters but does not know about if that unknown element fits into the size constraints set by the current Parent Element.

8. Stream Copy

It is sometimes necessary to create a Matroska file from another Matroska file, for example to add subtitles in a language or to edit out a portion of the content. Some values from the original Matroska file need to be kept the same in the destination file. For example, the SamplingFrequency of an audio track wouldn’t change between the two files. Some other values may change between the two files, for example the TrackNumber of an audio track when another track has been added.

An Element is marked with a property: stream copy: True when the values of that Element need to be kept identical between the source and destination file. If that property is not set, elements may or may not keep the same value between the source and destination.

9. DefaultDecodedFieldDuration

The DefaultDecodedFieldDuration Element can signal to the displaying application how often fields of a video sequence will be available for displaying. It can be used for both interlaced and progressive content.

If the video sequence is signaled as interlaced Section 5.1.4.1.28.1, then DefaultDecodedFieldDuration equals the period between two successive fields at the output of the decoding process. For video sequences signaled as progressive, DefaultDecodedFieldDuration is half of the period between two successive frames at the output of the decoding process.

These values are valid at the end of the decoding process before post-processing (such as deinterlacing or inverse telecine) is applied.

Examples:

* Blu-ray movie: 1000000000 ns/(48/1.001) = 20854167 ns
* PAL broadcast/DVD: 1000000000 ns/(50/1.000) = 20000000 ns
* N/ATSC broadcast: 1000000000 ns/(60/1.001) = 16683333 ns
* hard-telecined DVD: 1000000000 ns/(60/1.001) = 16683333 ns (60 encoded interlaced fields per second)
* soft-telecined DVD: 1000000000 ns/(60/1.001) = 16683333 ns (48 encoded interlaced fields per second, with "repeat_first_field = 1")

10. Cluster Blocks

Frames using references SHOULD be stored in "coding order". That means the references first, and then the frames referencing them. A consequence is that timestamps might not be consecutive. But a frame with a past timestamp MUST reference a frame already known, otherwise it’s considered bad/void.

Matroska has two similar ways to store frames in a block:

* in a Block which is contained inside a BlockGroup,
* or in a SimpleBlock which is directly in the Cluster.

The SimpleBlock is usually preferred unless some extra elements of the BlockGroup need to be used. A Matroska Reader MUST support both types of blocks.

Each block contains the same parts in the following order:

* a variable length header,
* optionally the lacing information,
* the consecutive frame(s)

The block header starts with the number of the Track it corresponds to. The value MUST corresponding to the TrackNumber (Section 5.1.4.1.1) of a TrackEntry of the Segment.

The TrackNumber is coded using the VINT mechanism described in Section 4 of [RFC8794]. To save space, the shortest VINT form SHOULD be used. The value can be coded on up to 8 octets. This is the only element with a variable size in the block header.

The timestamp is expressed in Track Ticks; see Section 11.1. The value is stored as a signed value on 16 bits.

10.1. Block Structure

This section describes the binary data contained in the Block Element Section 5.1.3.5.1. Bit 0 is the most significant bit.

As the TrackNumber size can vary between 1 and 8 octets, there are 8 different sizes for the Block header. We only provide the definitions for TrackNumber sizes of 1 and 2. The other variants can be deduced by extending the size of the TrackNumber by multiples of 8 bits.

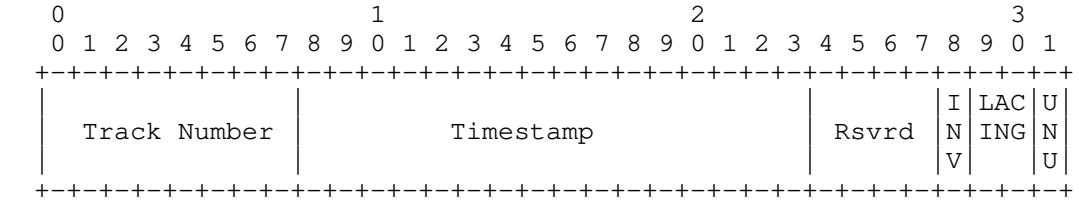


Figure 11: Block Header with 1 octet TrackNumber

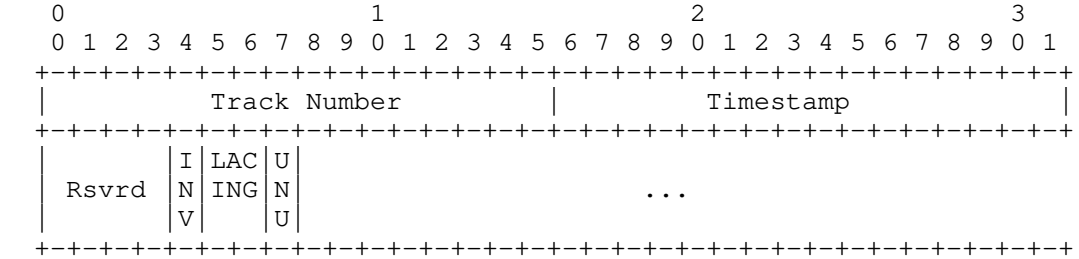


Figure 12: Block Header with 2 octets TrackNumber

where:

Track Number: 8, 16, 24, 32, 40, 48 or 64 bits
an EBML VINT coded track number

Timestamp: 16 bits
signed timestamp in Track Ticks

Rsvrd: 4 bits
Reserved bits MUST be set to 0

INV: 1 bit
Invisible, the codec SHOULD decode this frame but not display it

LACING: 2 bits
using lacing mode

* 00b : no lacing (Section 10.3.1)
* 01b : Xiph lacing (Section 10.3.2)
* 11b : EBML lacing (Section 10.3.3)
* 10b : fixed-size lacing (Section 10.3.4)

UNU: 1 bit
unused bit

The following data in the Block correspond to the lacing data and frames usage as described in each respective lacing mode.

10.2. SimpleBlock Structure

This section describes the binary data contained in the SimpleBlock Element Section 5.1.3.4. Bit 0 is the most significant bit.

The SimpleBlock is inspired by the Block structure; see Section 10.1. The main differences are the added Keyframe flag and Discardable flag. Otherwise, everything is the same.

As the TrackNumber size can vary between 1 and 8 octets, there are 8 different sizes for the SimpleBlock header. We only provide the definitions for TrackNumber sizes of 1 and 2. The other variants can be deduced by extending the size of the TrackNumber by multiples of 8 bits.

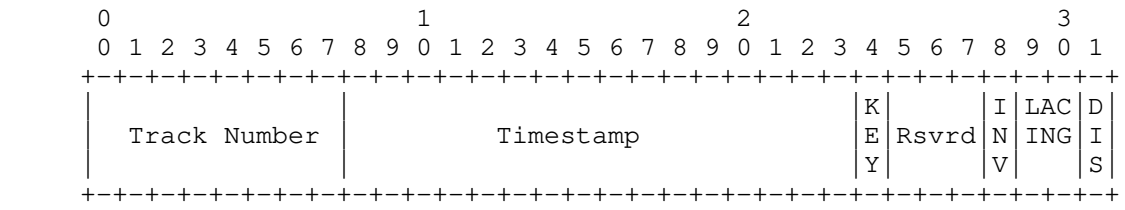


Figure 13: SimpleBlock Header with 1 octet TrackNumber

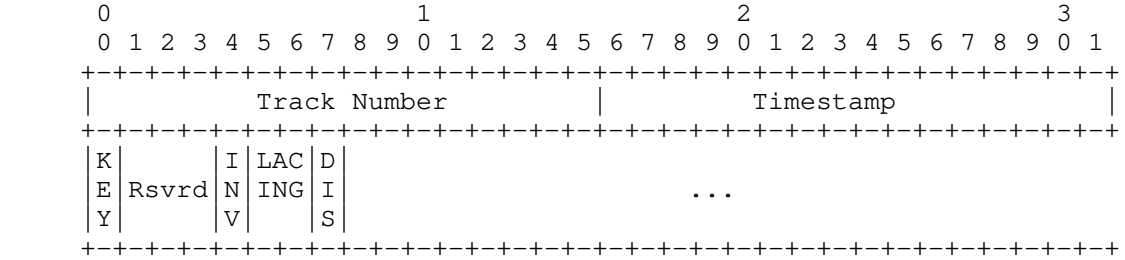


Figure 14: SimpleBlock Header with 2 octets TrackNumber

where:

Track Number: 8, 16, 24, 32, 40, 48 or 64 bits
an EBML VINT coded track number

Timestamp: 16 bits
signed timestamp in Track Ticks

KEY: 1 bit
Keyframe, set when the Block contains only keyframes

Rsvrd: 3 bits
Reserved bits MUST be set to 0

INV: 1 bit
Invisible, the codec SHOULD decode this frame but not display it

LACING: 2 bits
using lacing mode

* 00b : no lacing (Section 10.3.1)
* 01b : Xiph lacing (Section 10.3.2)
* 11b : EBML lacing (Section 10.3.3)
* 10b : fixed-size lacing (Section 10.3.4)

DIS: 1 bit
: Discardable, the frames of the Block can be discarded during playing if needed

The following data in the SimpleBlock correspond to the lacing data and frames usage as described in each respective lacing mode.

10.3. Block Lacing

Lacing is a mechanism to save space when storing data. It is typically used for small blocks of data (referred to as frames in Matroska). It packs multiple frames into a single Block or SimpleBlock.

Lacing MUST NOT be used to store a single frame in a Block or SimpleBlock.

There are 3 types of lacing:

1. Xiph, inspired by what is found in the Ogg container [RFC3533]
2. EBML, which is the same with sizes coded differently
3. fixed-size, where the size is not coded

When lacing is not used, i.e. to store a single frame, the lacing bits 5 and 6 of the Block or SimpleBlock MUST be set to zero.

For example, a user wants to store 3 frames of the same track. The first frame is 800 octets long, the second is 500 octets long and the third is 1000 octets long. As these data are small, they can be stored in a lace to save space.

It is possible not to use lacing at all and just store a single frame without any extra data. When the FlagLacing -- Section 5.1.4.1.12 -- is set to "0" all blocks of that track MUST NOT use lacing.

10.3.1. No lacing

When no lacing is used, the number of frames in the lace is ommitted and only one frame can be stored in the Block. The bits 5-6 of the Block Header flags are set to 0b00.

The Block for an 800 octets frame is as follows:

Block Octets	Value	Description
4-803	<frame>	Single frame data

Table 36: No lacing

When a Block contains a single frame, it MUST use this No lacing mode.

10.3.2. Xiph lacing

The Xiph lacing uses the same coding of size as found in the Ogg container [RFC3533]. The bits 5-6 of the Block Header flags are set to 0b01.

The Block data with laced frames is stored as follows:

* Lacing Head on 1 Octet: Number of frames in the lace minus 1.
* Lacing size of each frame except the last one.
* Binary data of each frame consecutively.

The lacing size is split into 255 values, stored as unsigned octets -- for example, 500 is coded 255;245 or [0xFF 0xF5]. A frame with a size multiple of 255 is coded with a 0 at the end of the size -- for example, 765 is coded 255;255;255;0 or [0xFF 0xFF 0xFF 0x00].

The size of the last frame is deduced from the size remaining in the Block after the other frames.

Because large sizes result in large coding of the sizes, it is RECOMMENDED to use Xiph lacing only with small frames.

In our example, the 800, 500 and 1000 frames are stored with Xiph lacing in a Block as follows:

Block Octet	Value	Description
4	0x02	Number of frames minus 1
5-8	0xFF 0xFF 0xFF 0x23	Size of the first frame (255;255;255;35)
9-10	0xFF 0xF5	Size of the second frame (255;245)
11-810		First frame data
811-1310		Second frame data
1311-2310		Third frame data

Table 37: Xiph lacing example

The Block is 2311 octets large and the last frame starts at 1311, so we can deduce the size of the last frame is 2311 - 1311 = 1000.

10.3.3. EBML lacing

The EBML lacing encodes the frame size with an EBML-like encoding [RFC8794]. The bits 5-6 of the Block Header flags are set to 0b11.

The Block data with laced frames is stored as follows:

* Lacing Head on 1 Octet: Number of frames in the lace minus 1.
* Lacing size of each frame except the last one.
* Binary data of each frame consecutively.

The first frame size is encoded as an EBML Variable-Size Integer value, also known as VINT in [RFC8794]. The remaining frame sizes are encoded as signed values using the difference between the frame size and the previous frame size. These signed values are encoded as VINT, with a mapping from signed to unsigned numbers. Decoding the unsigned number stored in the VINT to a signed number is done by subtracting 2^((7*n)-1)-1, where n is the octet size of the VINT.

Bit Representation of signed VINT	Possible Value Range
1xxx xxxx	2^7 values from -(2^6-1) to 2^6
01xx xxxx xxxx xxxx	2^14 values from -(2^13-1) to 2^13
001x xxxx xxxx xxxx xxxx xxxx	2^21 values from -(2^20-1) to 2^20
0001 xxxx xxxx xxxx xxxx xxxx xxxx xxxx	2^28 values from -(2^27-1) to 2^27
0000 1xxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx	2^35 values from -(2^34-1) to 2^34

Table 38: EBML Lacing signed VINT bits usage

In our example, the 800, 500 and 1000 frames are stored with EBML lacing in a Block as follows:

Block Octets	Value	Description
4	0x02	Number of frames minus 1
5-6	0x43 0x20	Size of the first frame (800 = 0x320 + 0x4000)
7-8	0x5E 0xD3	Size of the second frame (500 - 800 = -300 = - 0x12C + 0x1FFF + 0x4000)
8-807	<frame1>	First frame data
808-1307	<frame2>	Second frame data
1308-2307	<frame3>	Third frame data

Table 39: EBML lacing example

The Block is 2308 octets large and the last frame starts at 1308, so we can deduce the size of the last frame is 2308 - 1308 = 1000.

10.3.4. Fixed-size lacing

The Fixed-size lacing doesn't store the frame size, only the number of frames in the lace. Each frame MUST have the same size. The frame size of each frame is deduced from the total size of the Block. The bits 5-6 of the Block Header flags are set to 0b10.

The Block data with laced frames is stored as follows:

* Lacing Head on 1 Octet: Number of frames in the lace minus 1.
* Binary data of each frame consecutively.

For example, for 3 frames of 800 octets each:

Block Octets	Value	Description
4	0x02	Number of frames minus 1
5-804	<frame1>	First frame data
805-1604	<frame2>	Second frame data
1605-2404	<frame3>	Third frame data

Table 40: Fixed-size lacing example

This gives a Block of 2405 octets. When reading the Block we find that there are 3 frames (Octet 4). The data start at Octet 5, so the size of each frame is (2405 - 5) / 3 = 800.

10.3.5. Laced Frames Timestamp

A Block only contains a single timestamp value. But when lacing is used, it contains more than one frame. Each frame originally has its own timestamp, or Presentation Timestamp (PTS). That timestamp applies to the first frame in the lace.

In the lace, each frame after the first one has an underdetermined timestamp. But each of these frames MUST be contiguous -- i.e. the decoded data MUST NOT contain any gap between them. If there is a gap in the stream, the frames around the gap MUST NOT be in the same Block.

Lacing is only useful for small contiguous data to save space. This is usually the case for audio tracks and not the case for video -- which use a lot of data -- or subtitle tracks -- which have long

gaps. For audio, there is usually a fixed output sampling frequency for the whole track. So the decoder should be able to recover the timestamp of each sample, knowing each output sample is contiguous with a fixed frequency. For subtitles this is usually not the case so lacing SHOULD NOT be used.

10.4. Random Access Points

Random Access Points (RAP) are positions where the parser can seek to and start playback without decoding of what was before. In Matroska BlockGroups and SimpleBlocks can be RAPs. To seek to these elements it is still necessary to seek to the Cluster containing them, read the Cluster Timestamp and start playback from the BlockGroup or SimpleBlock that is a RAP.

Because a Matroska File is usually composed of multiple tracks playing at the same time -- video, audio and subtitles -- to seek properly to a RAP, each selected track must be taken in account. Usually all audio and subtitle BlockGroup or SimpleBlock are RAP. They are independent of each other and can be played randomly.

Video tracks on the other hand often use references to previous and future frames for better coding efficiency. Frames with such reference MUST either contain one or more ReferenceBlock Elements in their BlockGroup or MUST be marked as non-keyframe in a SimpleBlock; see Section 10.2.

* BlockGroup with a frame that references another frame, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <BlockGroup>
    <!-- References a Block 40 Track Ticks before this one -->
    <ReferenceBlock>-40</ReferenceBlock>
    <Block/>
  </BlockGroup>
  ...
</Cluster>
```

* SimpleBlock with a frame that references another frame, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <SimpleBlock/> (octet 3 bit 0 not set)
  ...
</Cluster>
```

Frames that are RAP -- i.e. they don't depend on other frames -- MUST set the keyframe flag if they are in a SimpleBlock or their parent BlockGroup MUST NOT contain a ReferenceBlock.

* BlockGroup with a frame that references no other frame, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <BlockGroup>
    <!-- No ReferenceBlock allowed in this BlockGroup -->
    <Block/>
  </BlockGroup>
  ...
</Cluster>
```

* SimpleBlock with a frame that references no other frame, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <SimpleBlock/> (octet 3 bit 0 set)
  ...
</Cluster>
```

There may be cases where the use of BlockGroup is necessary, as the frame may need a BlockDuration, BlockAdditions, CodecState or a DiscardPadding element. For thoses cases a SimpleBlock MUST NOT be used, the reference information SHOULD be recovered for non-RAP frames.

* SimpleBlock with a frame that references another frame, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <SimpleBlock/> (octet 3 bit 0 not set)
  ...
</Cluster>
```

* Same frame that references another frame put inside a BlockGroup to add BlockDuration, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <BlockGroup>
    <!-- ReferenceBlock value recovered based on the codec -->
    <ReferenceBlock>-40</ReferenceBlock>
    <BlockDuration>20<BlockDuration>
    <Block/>
  </BlockGroup>
  ...
</Cluster>
```

When a frame in a BlockGroup is not a RAP, the BlockGroup MUST contain at least a ReferenceBlock. The ReferenceBlocks MUST be used in one of the following ways:

* each reference frame listed as a ReferenceBlock,
* some referenced frame listed as a ReferenceBlock, even if the timestamp value is accurate,
* or one ReferenceBlock with the timestamp value "0" corresponding to a self or unknown reference.

The lack of ReferenceBlock would mean such a frame is a RAP and seeking on that frame that actually depends on other frames may create bogus output or even crash.

* Same frame that references another frame put inside a BlockGroup but the reference could not be recovered, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <BlockGroup>
    <!-- ReferenceBlock value not recovered from the codec -->
    <ReferenceBlock>0</ReferenceBlock>
    <BlockDuration>20<BlockDuration>
    <Block/>
  </BlockGroup>
  ...
</Cluster>
```

* BlockGroup with a frame that references two other frames, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <BlockGroup>
    <!-- References a Block 80 Track Ticks before this one -->
    <ReferenceBlock>-80</ReferenceBlock>
    <!-- References a Block 40 Track Ticks after this one -->
    <ReferenceBlock>40</ReferenceBlock>
    <Block/>
  </BlockGroup>
  ...
</Cluster>
```

Intra-only video frames, such as the ones found in AV1 or VP9, can be decoded without any other frame, but they don't reset the codec state. So seeking to these frames is not possible as the next frames may need frames that are not known from this seeking point. Such intra-only frames MUST NOT be considered as keyframes so the keyframe flag MUST NOT be set in the SimpleBlock or a ReferenceBlock MUST be used to signify the frame is not a RAP. The timestamp value of the ReferenceBlock MUST be "0", meaning it's referencing itself.

* Intra-only frame not an RAP, with the EBML tree shown as XML:

```
<Cluster>
  <Timestamp>123456</Timestamp>
  <BlockGroup>
    <!-- References itself to mark it should not be used as RAP -->
    <ReferenceBlock>0</ReferenceBlock>
    <Block/>
  </BlockGroup>
  ...
</Cluster>
```

Because a video SimpleBlock has less references information than a video BlockGroup, it is possible to remux a video track using BlockGroup into a SimpleBlock, as long as it doesn't use any other BlockGroup features than ReferenceBlock.

11. Timestamps

Historically timestamps in Matroska were mistakenly called timecodes. The Timestamp Element was called Timecode, the TimestampScale Element was called TimecodeScale, the TrackTimestampScale Element was called TrackTimecodeScale and the ReferenceTimestamp Element was called ReferenceTimeCode.

11.1. Timestamp Ticks

All timestamp values in Matroska are expressed in multiples of a tick. They are usually stored as integers. There are three types of ticks possible:

11.1.1. Matroska Ticks

For such elements, the timestamp value is stored directly in nanoseconds.

The elements storing values in Matroska Ticks/nanoseconds are:

* TrackEntry\DefaultDuration; defined in Section 5.1.4.1.13
* TrackEntry\DefaultDecodedFieldDuration; defined in Section 5.1.4.1.14
* TrackEntry\SeekPreRoll; defined in Section 5.1.4.1.26
* TrackEntry\CodecDelay; defined in Section 5.1.4.1.25
* BlockGroup\DiscardPadding; defined in Section 5.1.3.5.7
* ChapterAtom\ChapterTimeStart; defined in Section 5.1.7.1.4.3
* ChapterAtom\ChapterTimeEnd; defined in Section 5.1.7.1.4.4
* CuePoint\CueTime; defined in Section 5.1.5.1.1
* CueReference\CueRefTime; defined in Section 5.1.5.1.1

11.1.2. Segment Ticks

Elements in Segment Ticks involve the use of the TimestampScale Element of the Segment to get the timestamp in nanoseconds of the element, with the following formula:

timestamp in nanosecond = element value * TimestampScale

This allows storing smaller integer values in the elements.

When using the default value of TimestampScale of "1,000,000", one Segment Tick represents one millisecond.

The elements storing values in Segment Ticks are:

* Cluster\Timestamp; defined in Section 5.1.3.1
* Info\Duration is stored as a floating-point but the same formula applies; defined in Section 5.1.2.10
* CuePoint\CueTrackPositions\CueDuration; defined in Section 5.1.5.1.2.4

11.1.3. Track Ticks

Elements in Track Ticks involve the use of the TimestampScale Element of the Segment and the TrackTimestampScale Element of the Track to get the timestamp in nanoseconds of the element, with the following formula:

```
timestamp in nanoseconds =
    element value * TrackTimestampScale * TimestampScale
```

This allows storing smaller integer values in the elements. The resulting floating-point values of the timestamps are still expressed in nanoseconds.

When using the default values for TimestampScale and TrackTimestampScale of "1,000,000" and of "1.0" respectively, one Track Tick represents one millisecond.

The elements storing values in Track Ticks are:

* Cluster\BlockGroup\Block and Cluster\SimpleBlock timestamps; detailed in Section 11.2
* Cluster\BlockGroup\BlockDuration; defined in Section 5.1.3.5.3
* Cluster\BlockGroup\ReferenceBlock; defined in Section 5.1.3.5.5

When the TrackTimestampScale is interpreted as "1.0", Track Ticks are equivalent to Segment Ticks and give an integer value in nanoseconds. This is the most common case as TrackTimestampScale is usually omitted.

A value of TrackTimestampScale other than "1.0" MAY be used to scale the timestamps more in tune with each Track sampling frequency. For historical reasons, a lot of Matroska readers don’t take the TrackTimestampScale value in account. So using a value other than "1.0" might not work in many places.

11.2. Block Timestamps

A Block Element and SimpleBlock Element timestamp is the time when the decoded data of the first frame in the Block/SimpleBlock MUST be presented, if the track of that Block/SimpleBlock is selected for playback. This is also known as the Presentation Timestamp (PTS).

The Block Element and SimpleBlock Element store their timestamps as signed integers, relative to the Cluster\Timestamp value of the Cluster they are stored in. To get the timestamp of a Block or SimpleBlock in nanoseconds you have to use the following formula:

(Cluster\Timestamp + (block timestamp * TrackTimestampScale)) * TimestampScale

The Block Element and SimpleBlock Element store their timestamps as 16bit signed integers, allowing a range from "-32768" to "+32767" Track Ticks. Although these values can be negative, when added to the Cluster\Timestamp, the resulting frame timestamp SHOULD NOT be negative.

When a CodecDelay Element is set, its value MUST be substracted from each Block timestamp of that track. To get the timestamp in nanoseconds of the first frame in a Block or SimpleBlock, the formula becomes:

((Cluster\Timestamp + (block timestamp * TrackTimestampScale)) * TimestampScale) - CodecDelay

The resulting frame timestamp SHOULD NOT be negative.

During playback, when a frame has a negative timestamp, the content MUST be decoded by the decoder but not played to the user.

11.3. TimestampScale Rounding

The default Track Tick duration is one millisecond.

The TimestampScale is a floating-point value, which is usually 1.0. But when it's not, the multiplied Block Timestamp is a floating-point value in nanoseconds. The Matroska Reader SHOULD use the nearest rounding value in nanosecond to get the proper nanosecond timestamp of a Block. This allows some clever TimestampScale values to have more refined timestamp precision per frame.

12. Language Codes

Matroska from version 1 through 3 uses language codes that can be either the 3 letters bibliographic ISO-639-2 form [ISO639-2] (like "fre" for French), or such a language code followed by a dash and a country code for specialities in languages (like "fre-ca" for Canadian French). The ISO 639-2 Language Elements are "Language Element", "TagLanguage Element", and "ChapLanguage Element".

Starting in Matroska version 4, either [ISO639-2] or [BCP47] MAY be used, although BCP 47 is RECOMMENDED. The BCP 47 Language Elements are "LanguageBCP47 Element", "TagLanguageBCP47 Element", and "ChapLanguageBCP47 Element". If a BCP 47 Language Element and an ISO 639-2 Language Element are used within the same Parent Element, then the ISO 639-2 Language Element MUST be ignored and precedence given to the BCP 47 Language Element.

13. Country Codes

Country codes are the [BCP47] two-letter region subtag, without the UK exception.

14. Encryption

This Matroska specification provides no interoperable solution for securing the data container with any assurances of confidentiality, integrity, authenticity, or to provide authorization. The ContentEncryption Element (Section 5.1.4.1.31.8) and associated sub-fields (Section 5.1.4.1.31.9 to Section 5.1.4.1.31.12) are defined only for the benefit of implementers to construct their own proprietary solution or as the basis for further standardization activities. How to use these fields to secure a Matroska data container is out of scope, as are any related issues such as key management and distribution.

A Matroska Reader who encounters containers that use the fields defined in this section MUST rely on out-of-scope guidance to decode the associated content.

Because encryption occurs within the Block Element, it is possible to manipulate encrypted streams without decrypting them. The streams could potentially be copied, deleted, cut, appended, or any number of other possible editing techniques without decryption. The data can be used without having to expose it or go through the decrypting process.

Encryption can also be layered within Matroska. This means that two completely different types of encryption can be used, requiring two separate keys to be able to decrypt a stream.

Encryption information is stored in the ContentEncodings Element under the ContentEncryption Element.

For encryption systems sharing public/private keys, the creation of the keys and the exchange of keys are not covered by this document. They have to be handled by the system using Matroska.

The algorithms described in Table 26 support different modes of operations and key sizes. The specification of these parameters is required for a complete solution, but is out of scope of this document and left to the proprietary implementations using them or subsequent profiles of this document.

The ContentEncodingScope Element gives an idea of which part of the track are encrypted. But each ContentEncAlgo Element and its sub elements like AESSettingsCipherMode really define how the encrypted should be exactly interpreted.

An example of an extension that builds upon these security-related fields in this specification is [WebM-Enc]. It uses AES-CTR, ContentEncAlgo = 5 (Section 5.1.4.1.31.9) and AESSettingsCipherMode = 1 (Section 5.1.4.1.31.12).

A Matroska Writer MUST NOT use insecure cryptographic algorithms to create new archives or streams, but a Matroska Reader MAY support these algorithms to read previously made archives or stream.

15. Image Presentation

15.1. Cropping

The PixelCrop Elements (PixelCropTop, PixelCropBottom, PixelCropRight, and PixelCropLeft) indicate when, and by how much, encoded videos frames SHOULD be cropped for display. These Elements allow edges of the frame that are not intended for display, such as the sprockets of a full-frame film scan or the VANC area of a digitized analog videotape, to be stored but hidden. PixelCropTop and PixelCropBottom store an integer of how many rows of pixels SHOULD be cropped from the top and bottom of the image (respectively). PixelCropLeft and PixelCropRight store an integer of how many columns of pixels SHOULD be cropped from the left and right of the image (respectively).

For example, a pillar-boxed video that stores a 1440x1080 visual image within the center of a padded 1920x1080 encoded image may set both PixelCropLeft and PixelCropRight to "240", so that a Matroska Player should crop off 240 columns of pixels from the left and right of the encoded image to present the image with the pillar-boxes hidden.

Cropping has to be performed before resizing and the display dimensions given by DisplayWidth, DisplayHeight and DisplayUnit apply to the already cropped image.

15.2. Rotation

The ProjectionPoseRoll Element (see Section 5.1.4.1.28.46) can be used to indicate that the image from the associated video track SHOULD be rotated for presentation. For instance, the following representation of the Projection Element Section 5.1.4.1.28.41) and the ProjectionPoseRoll Element represents a video track where the image SHOULD be presented with a 90-degree counter-clockwise rotation, with the EBML tree shown as XML :

```
<Projection>
  <ProjectionPoseRoll>90</ProjectionPoseRoll>
</Projection>
```

Figure 15: Rotation example.

16. Segment Position

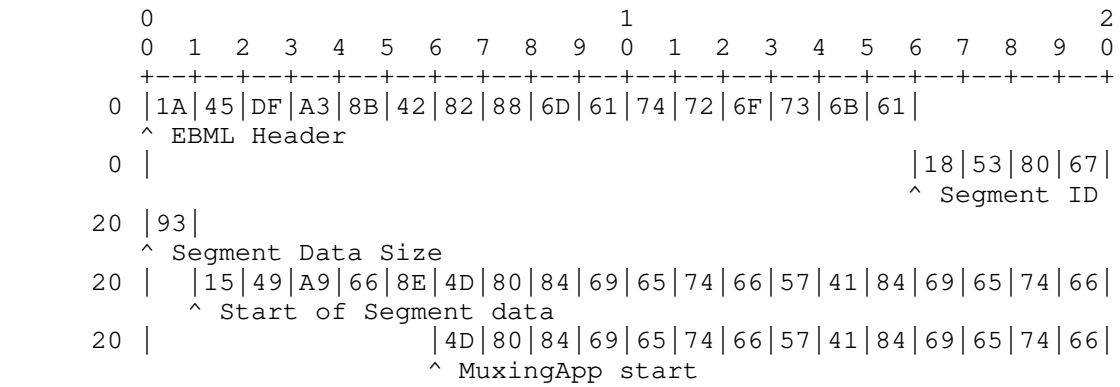
The Segment Position of an Element refers to the position of the first octet of the Element ID of that Element, measured in octets, from the beginning of the Element Data section of the containing Segment Element. In other words, the Segment Position of an Element is the distance in octets from the beginning of its containing Segment Element minus the size of the Element ID and Element Data Size of that Segment Element. The Segment Position of the first Child Element of the Segment Element is 0. An Element which is not stored within a Segment Element, such as the Elements of the EBML Header, do not have a Segment Position.

16.1. Segment Position Exception

Elements that are defined to store a Segment Position MAY define reserved values to indicate a special meaning.

16.2. Example of Segment Position

This table presents an example of Segment Position by showing a hexadecimal representation of a very small Matroska file with labels to show the offsets in octets. The file contains a Segment Element with an Element ID of "0x18538067" and a MuxingApp Element with an Element ID of "0x4D80".



In the above example, the Element ID of the Segment Element is stored at offset 16, the Element Data Size of the Segment Element is stored at offset 20, and the Element Data of the Segment Element is stored at offset 21.

The MuxingApp Element is stored at offset 26. Since the Segment Position of an Element is calculated by subtracting the position of the Element Data of the containing Segment Element from the position of that Element, the Segment Position of MuxingApp Element in the above example is ’26 - 21’ or ’5’.

17. Linked Segments

Matroska provides several methods to link two or more Segment Elements together to create a Linked Segment. A Linked Segment is a set of multiple Segments linked together into a single presentation by using Hard Linking or Medium Linking.

All Segments within a Linked Segment MUST have a SegmentUUID.

All Segments within a Linked Segment SHOULD be stored within the same directory or be accessible quickly based on their SegmentUUID in order to have seamless transition between segments.

All Segments within a Linked Segment MAY set a SegmentFamily with a common value to make it easier for a Matroska Player to know which Segments are meant to be played together.

The SegmentFilename, PrevFilename and NextFilename elements MAY also give hints on the original filenames that were used when the Segment links were created, in case some SegmentUUID are damaged.

17.1. Hard Linking

Hard Linking, also called splitting, is the process of creating a Linked Segment by linking multiple Segment Elements using the NextUUID and PrevUUID Elements.

All Segments within a Hard Linked Segment MUST use the same Tracks list and TimestampScale.

Within a Linked Segment, the timestamps of Block and SimpleBlock MUST follow consecutively the timestamps of Block and SimpleBlock from the previous Segment in linking order.

With Hard Linking, the chapters of any Segment within the Linked Segment MUST only reference the current Segment. The NextUUID and PrevUUID reference the respective SegmentUUID values of the next and previous Segments.

The first Segment of a Linked Segment MUST NOT have a PrevUUID Element. The last Segment of a Linked Segment MUST NOT have a NextUUID Element.

For each node of the chain of Segments of a Linked Segment at least one Segment MUST reference the other Segment within the chain.

In a chain of Segments of a Linked Segment the NextUUID always takes precedence over the PrevUUID. So if SegmentA has a NextUUID to SegmentB and SegmentB has a PrevUUID to SegmentC, the link to use is NextUUID between SegmentA and SegmentB, SegmentC is not part of the Linked Segment.

If SegmentB has a PrevUUID to SegmentA but SegmentA has no NextUUID, then the Matroska Player MAY consider these two Segments linked as SegmentA followed by SegmentB.

As an example, three Segments can be Hard Linked as a Linked Segment through cross-referencing each other with SegmentUUID, PrevUUID, and NextUUID, as in this table:

file name	SegmentUUID	PrevUUID	NextUUID
start.mkv	71000c23cd310998 53fbc94dd984a5dd	Invalid	a77b3598941cb803 eac0fcdafe44fac9
middle.mkv	a77b3598941cb803 eac0fcdafe44fac9	71000c23cd310998 53fbc94dd984a5dd	6c92285fa6d3e827 b198d120ea3ac674
end.mkv	6c92285fa6d3e827 b198d120ea3ac674	a77b3598941cb803 eac0fcdafe44fac9	Invalid

Table 41: Usual Hard Linking UIDs

An other example where only the NextUUID Element is used:

file name	SegmentUUID	PrevUUID	NextUUID
start.mkv	71000c23cd310998 53fbc94dd984a5dd	Invalid	a77b3598941cb803 eac0fcdafe44fac9
middle.mkv	a77b3598941cb803 eac0fcdafe44fac9	n/a	6c92285fa6d3e827 b198d120ea3ac674
end.mkv	6c92285fa6d3e827 b198d120ea3ac674	n/a	Invalid

Table 42: Hard Linking without PrevUUID

An example where only the PrevUUID Element is used:

file name	SegmentUUID	PrevUUID	NextUUID
start.mkv	71000c23cd310998 53fbc94dd984a5dd	Invalid	n/a
middle.mkv	a77b3598941cb803 eac0fcdafe44fac9	71000c23cd310998 53fbc94dd984a5dd	n/a
end.mkv	6c92285fa6d3e827 b198d120ea3ac674	a77b3598941cb803 eac0fcdafe44fac9	Invalid

Table 43: Hard Linking without NextUUID

In this example only the middle.mkv is using the PrevUUID and NextUUID Elements:

file name	SegmentUUID	PrevUUID	NextUUID
start.mkv	71000c23cd310998 53fbc94dd984a5dd	Invalid	n/a
middle.mkv	a77b3598941cb803 eac0fcdafe44fac9	71000c23cd310998 53fbc94dd984a5dd	6c92285fa6d3e827 b198d120ea3ac674
end.mkv	6c92285fa6d3e827 b198d120ea3ac674	n/a	Invalid

Table 44: Hard Linking with mixed UID links

17.2. Medium Linking

Medium Linking creates relationships between Segments using Ordered Chapters (Section 20.1.3) and the ChapterSegmentUUID Element. A Chapter Edition with Ordered Chapters MAY contain Chapter elements that reference timestamp ranges from other Segments. The Segment referenced by the Ordered Chapter via the ChapterSegmentUUID Element SHOULD be played as part of a Linked Segment.

The timestamps of Segment content referenced by Ordered Chapters MUST be adjusted according to the cumulative duration of the previous Ordered Chapters.

As an example a file named intro.mkv could have a SegmentUUID of "0xb16a58609fc7e60653a60c984fc11ead". Another file called program.mkv could use a Chapter Edition that contains two Ordered Chapters. The first chapter references the Segment of intro.mkv with the use of a ChapterSegmentUUID, ChapterSegmentEditionUID, ChapterTimeStart, and optionally a ChapterTimeEnd element. The second chapter references content within the Segment of program.mkv. A Matroska Player SHOULD recognize the Linked Segment created by the use of ChapterSegmentUUID in an enabled Edition and present the reference content of the two Segments as a single presentation.

The ChapterSegmentUUID represents the Segment that holds the content to play in place of the Linked Chapter. The ChapterSegmentUUID MUST NOT be the SegmentUUID of its own Segment.

There are 2 ways to use a chapter link:

* Linked-Duration linking,

* Linked-Edition linking

17.2.1. Linked-Duration

A Matroska Player MUST play the content of the linked Segment from the ChapterTimeStart until ChapterTimeEnd timestamp in place of the Linked Chapter.

ChapterTimeStart and ChapterTimeEnd represent timestamps in the Linked Segment matching the value of ChapterSegmentUUID. Their values MUST be in the range of the linked Segment duration.

The ChapterTimeEnd value MUST be set when using linked-duration chapter linking. ChapterSegmentEditionUID MUST NOT be set.

17.2.2. Linked-Edition

A Matroska Player MUST play the whole linked Edition of the linked Segment in place of the Linked Chapter.

ChapterSegmentEditionUID represents a valid Edition from the Linked Segment matching the value of ChapterSegmentUUID.

When using linked-edition chapter linking. ChapterTimeEnd is OPTIONAL.

18. Track Flags

18.1. Default flag

The "default track" flag is a hint for a Matroska Player indicating that a given track SHOULD be eligible to be automatically selected as the default track for a given language. If no tracks in a given language have the default track flag set, then all tracks in that language are eligible for automatic selection. This can be used to indicate that a track provides "regular service" suitable for users with default settings, as opposed to specialized services, such as commentary, hearing-impaired captions, or descriptive audio.

The Matroska Player MAY override the "default track" flag for any reason, including user preferences to prefer tracks providing accessibility services.

18.2. Forced flag

The "forced" flag tells the Matroska Player that it SHOULD display this subtitle track, even if user preferences usually would not call for any subtitles to be displayed alongside the current selected audio track. This can be used to indicate that a track contains translations of onscreen text, or of dialogue spoken in a different language than the track's primary one.

18.3. Hearing-impaired flag

The "hearing impaired" flag tells the Matroska Player that it SHOULD prefer this track when selecting a default track for a hearing-impaired user, and that it MAY prefer to select a different track when selecting a default track for a non-hearing-impaired user.

18.4. Visual-impaired flag

The "visual impaired" flag tells the Matroska Player that it SHOULD prefer this track when selecting a default track for a visually-impaired user, and that it MAY prefer to select a different track when selecting a default track for a non-visually-impaired user.

18.5. Descriptions flag

The "descriptions" flag tells the Matroska Player that this track is suitable to play via a text-to-speech system for a visually-impaired user, and that it SHOULD NOT automatically select this track when selecting a default track for a non-visually-impaired user.

18.6. Original flag

The "original" flag tells the Matroska Player that this track is in the original language, and that it SHOULD prefer it if configured to prefer original-language tracks of this track's type.

18.7. Commentary flag

The "commentary" flag tells the Matroska Player that this track contains commentary on the content.

18.8. Track Operation

TrackOperation allows combining multiple tracks to make a virtual one. It uses two separate system to combine tracks. One to create a 3D "composition" (left/right/background planes) and one to simplify join two tracks together to make a single track.

A track created with TrackOperation is a proper track with a UID and all its flags. However, the codec ID is meaningless because each "sub" track needs to be decoded by its own decoder before the "operation" is applied. The Cues Elements corresponding to such a virtual track SHOULD be the union of the Cues Elements for each of the tracks it's composed of (when the Cues are defined per track).

In the case of TrackJoinBlocks, the Block Elements (from BlockGroup and SimpleBlock) of all the tracks SHOULD be used as if they were defined for this new virtual Track. When two Block Elements have overlapping start or end timestamps, it's up to the underlying system to either drop some of these frames or render them the way they overlap. This situation SHOULD be avoided when creating such tracks as you can never be sure of the end result on different platforms.

18.9. Overlay Track

Overlay tracks SHOULD be rendered in the same channel as the track it's linked to. When content is found in such a track, it SHOULD be played on the rendering channel instead of the original track.

18.10. Multi-planar and 3D videos

There are two different ways to compress 3D videos: have each eye track in a separate track and have one track have both eyes combined inside (which is more efficient, compression-wise). Matroska supports both ways.

For the single track variant, there is the StereoMode Element, which defines how planes are assembled in the track (mono or left-right combined). Odd values of StereoMode means the left plane comes first for more convenient reading. The pixel count of the track (PixelWidth/PixelHeight) is the raw amount of pixels, for example 3840x1080 for full HD side by side, and the DisplayWidth/DisplayHeight in pixels is the amount of pixels for one plane (1920x1080 for that full HD stream). Old stereo 3D were displayed using anaglyph (cyan and red colors separated). For compatibility with such movies, there is a value of the StereoMode that corresponds to AnaGlyph.

There is also a "packed" mode (values 13 and 14) which consists of packing two frames together in a Block using lacing. The first frame is the left eye and the other frame is the right eye (or vice versa). The frames SHOULD be decoded in that order and are possibly dependent on each other (P and B frames).

For separate tracks, Matroska needs to define exactly which track does what. TrackOperation with TrackCombinePlanes do that. For more details look at Section 18.8 on how TrackOperation works.

The 3D support is still in infancy and may evolve to support more features.

The StereoMode used to be part of Matroska v2 but it didn't meet the requirement for multiple tracks. There was also a bug in libmatroska prior to 0.9.0 that would save/read it as 0x53B9 instead of 0x53B8; see OldStereoMode (Section 5.1.4.1.28.5). Matroska Readers MAY support these legacy files by checking Matroska v2 or 0x53B9. The older values of StereoMode were 0: mono, 1: right eye, 2: left eye, 3: both eyes, the only values that can be found in OldStereoMode. They are not compatible with the StereoMode values found in Matroska v3 and above.

19. Default track selection

This section provides some example sets of Tracks and hypothetical user settings, along with indications of which ones a similarly-configured Matroska Player SHOULD automatically select for playback by default in such a situation. A player MAY provide additional settings with more detailed controls for more nuanced scenarios. These examples are provided as guidelines to illustrate the intended usages of the various supported Track flags, and their expected behaviors.

Track names are shown in English for illustrative purposes; actual files may have titles in the language of each track, or provide titles in multiple languages.

19.1. Audio Selection

Example track set:

No.	Type	Lang	Layout	Original	Default	Other flags	Name
1	Video	und	N/A	N/A	N/A	None	
2	Audio	eng	5.1	1	1	None	
3	Audio	eng	2.0	1	1	None	
4	Audio	eng	2.0	1	0	Visual-impaired	Descriptive audio
5	Audio	esp	5.1	0	1	None	
6	Audio	esp	2.0	0	0	Visual-impaired	Descriptive audio
7	Audio	eng	2.0	1	0	Commentary	Director’s Commentary
8	Audio	eng	2.0	1	0	None	Karaoke

Table 45: Audio Tracks for default selection

Here we have a file with 7 audio tracks, of which 5 are in English and 2 are in Spanish.

The English tracks all have the Original flag, indicating that English is the original content language.

Generally the player will first consider the track languages: if the player has an option to prefer original-language audio and the user has enabled it, then it should prefer one of the Original-flagged tracks. If configured to specifically prefer audio tracks in English or Spanish, the player should select one of the tracks in the corresponding language. The player may also wish to prefer an Original-flagged track if no tracks matching any of the user’s explicitly-preferred languages are available.

Two of the tracks have the Visual-impaired flag. If the player has been configured to prefer such tracks, it should select one; otherwise, it should avoid them if possible.

If selecting an English track, when other settings have left multiple possible options, it may be useful to exclude the tracks that lack the Default flag: here, one provides descriptive service for the visually impaired (which has its own flag and may be automatically

selected by user configuration, but is unsuitable for users with default-configured players), one is a commentary track (which has its own flag, which the player may or may not have specialized handling for), and the last contains karaoke versions of the music that plays during the film, which is an unusual specialized audio service that Matroska has no built-in support for indicating, so it's indicated in the track name instead. By not setting the Default flag on these specialized tracks, the file's author hints that they should not be automatically selected by a default-configured player.

Having narrowed its choices down, our example player now may have to select between tracks 2 and 3. The only difference between these tracks is their channel layouts: 2 is 5.1 surround, while 3 is stereo. If the player is aware that the output device is a pair of headphones or stereo speakers, it may wish to prefer the stereo mix automatically. On the other hand, if it knows that the device is a surround system, it may wish to prefer the surround mix.

If the player finishes analyzing all of the available audio tracks and finds that multiple seems equally and maximally preferable, it SHOULD default to the first of the group.

19.2. Subtitle selection

Example track set:

No.	Type	Lang	Original	Default	Forced	Other flags	Name
1	Video	und	N/A	N/A	N/A	None	
2	Audio	fra	1	1	N/A	None	
3	Audio	por	0	1	N/A	None	
4	Subtitles	fra	1	1	0	None	
5	Subtitles	fra	1	0	0	Hearing-impaired	Captions for the hearing-impaired
6	Subtitles	por	0	1	0	None	
7	Subtitles	por	0	0	1	None	Signs
8	Subtitles	por	0	0	0	Hearing-impaired	SDH

Table 46: Subtitle Tracks for default selection

Here we have 2 audio tracks and 5 subtitle tracks. As we can see, French is the original language.

We'll start by discussing the case where the user prefers French (or Original-language) audio (or has explicitly selected the French audio track), and also prefers French subtitles.

In this case, if the player isn't configured to display captions when the audio matches their preferred subtitle languages, the player doesn't need to select a subtitle track at all.

If the user _has_ indicated that they want captions to be displayed, the selection simply comes down to whether Hearing-impaired subtitles are preferred.

The situation for a user who prefers Portuguese subtitles starts out somewhat analogous. If they select the original French audio (either by explicit audio language preference, preference for Original-language tracks, or by explicitly selecting that track), then the selection once again comes down to the hearing-impaired preference.

However, the case where the Portuguese audio track is selected has an important catch: a Forced track in Portuguese is present. This may contain translations of onscreen text from the video track, or of portions of the audio that are not translated (music, for instance). This means that even if the user's preferences wouldn't normally call for captions here, the Forced track should be selected nonetheless, rather than selecting no track at all. On the other hand, if the user's preferences _do_ call for captions, the non-Forced tracks should be preferred, as the Forced track will not contain captioning for the dialogue.

20. Chapters

The Matroska Chapters system can have multiple Editions and each Edition can consist of Simple Chapters where a chapter start time is used as marker in the timeline only. An Edition can be more complex with Ordered Chapters where a chapter end time stamp is additionally used or much more complex with Linked Chapters. The Matroska Chapters system can also have a menu structure, borrowed from the DVD menu system [DVD-Video], or have its own built-in Matroska menu structure.

20.1. EditionEntry

The EditionEntry is also called an Edition. An Edition contains a set of Edition flags and MUST contain at least one ChapterAtom Element. Chapters are always inside an Edition (or a Chapter itself part of an Edition). Multiple Editions are allowed. Some of these Editions MAY be ordered and others not.

20.1.1. EditionFlagDefault

Only one Edition SHOULD have an EditionFlagDefault flag set to true.

20.1.2. Default Edition

The Default Edition is the Edition that a Matroska Player SHOULD use for playback by default.

The first Edition with the EditionFlagDefault flag set to true is the Default Edition.

When all EditionFlagDefault flags are set to false, then the first Edition is the Default Edition.

Edition	FlagDefault	Default Edition
Edition 1	true	X
Edition 2	true	
Edition 3	true	

Table 47: Default edition, all default

Edition	FlagDefault	Default Edition
Edition 1	false	X
Edition 2	false	
Edition 3	false	

Table 48: Default edition, no default

Edition	FlagDefault	Default Edition
Edition 1	false	
Edition 2	true	X
Edition 3	false	

Table 49: Default edition, with default

20.1.3. EditionFlagOrdered

The EditionFlagOrdered Flag is a significant feature as it enables an Edition of Ordered Chapters which defines and arranges a virtual timeline rather than simply labeling points within the timeline. For example, with Editions of Ordered Chapters a single Matroska file can present multiple edits of a film without duplicating content. Alternatively, if a videotape is digitized in full, one Ordered Edition could present the full content (including colorbars, countdown, slate, a feature presentation, and black frames), while another Edition of Ordered Chapters can use Chapters that only mark the intended presentation with the colorbars and other ancillary

visual information excluded. If an Edition of Ordered Chapters is enabled, then the Matroska Player MUST play those Chapters in their stored order from the timestamp marked in the ChapterTimeStart Element to the timestamp marked in to ChapterTimeEnd Element.

If the EditionFlagOrdered Flag evaluates to "0", Simple Chapters are used and only the ChapterTimeStart of a Chapter is used as chapter mark to jump to the predefined point in the timeline. With Simple Chapters, a Matroska Player MUST ignore certain Chapter Elements. In that case these elements are informational only.

The following list shows the different Chapter elements only found in Ordered Chapters.

Ordered Chapter elements
ChapterAtom/ChapterSegmentUUID
ChapterAtom/ChapterSegmentEditionUID
ChapterAtom/ChapterTrack
ChapterAtom/ChapProcess
Info/ChapterTranslate
TrackEntry/TrackTranslate

Table 50: elements only found in ordered chapters

Furthermore, there are other EBML Elements which could be used if the EditionFlagOrdered evaluates to "1".

20.1.3.1. Ordered-Edition and Matroska Segment-Linking

* Hard Linking: Ordered-Chapters supersedes the Hard Linking.

* Medium Linking: Ordered Chapters are used in a normal way and can be combined with the ChapterSegmentUUID element which establishes a link to another Segment.

See Section 17 on the Linked Segments for more information about Hard Linking and Medium Linking.

20.2. ChapterAtom

The ChapterAtom is also called a Chapter.

20.2.1. ChapterTimeStart

The timestamp of the start of Chapter with nanosecond accuracy, not scaled by TimestampScale. For Simple Chapters this is the position of the chapter markers in the timeline.

20.2.2. ChapterTimeEnd

The timestamp of the end of Chapter with nanosecond accuracy, not scaled by TimestampScale. The timestamp defined by the ChapterTimeEnd is not part of the Chapter. A Matroska Player calculates the duration of this Chapter using the difference between the ChapterTimeEnd and ChapterTimeStart. The end timestamp MUST be greater than or equal to the start timestamp.

When the ChapterTimeEnd timestamp is equal to the ChapterTimeStart timestamp, the timestamps is included in the Chapter. It can be useful to put markers in a file or add chapter commands with ordered chapter commands without having to play anything; see Section 5.1.7.1.4.14.

Chapter	Start timestamp	End timestamp	Duration
Chapter 1	0	1000000000	1000000000
Chapter 2	1000000000	5000000000	4000000000
Chapter 3	6000000000	6000000000	0
Chapter 4	9000000000	8000000000	Invalid (-1000000000)

Table 51: ChapterTimeEnd usage possibilities

20.2.3. Nested Chapters

A ChapterAtom element can contain other ChapterAtom elements. That element is a Parent Chapter and the ChapterAtom elements it contains are Nested Chapters.

Nested Chapters can be useful to tag small parts of a Segment that already have tags or add Chapter Codec commands on smaller parts of a Segment that already have Chapter Codec commands.

The ChapterTimeStart of a Nested Chapter MUST be greater than or equal to the ChapterTimeStart its Parent Chapter.

If the Parent Chapter of a Nested Chapter has a ChapterTimeEnd, the ChapterTimeStart of that Nested Chapter MUST be smaller than or equal to the ChapterTimeEnd of the Parent Chapter.

20.2.4. Nested Chapters in Ordered Chapters

The ChapterTimeEnd of the lowest level of Nested Chapters MUST be set for Ordered Chapters.

When used with Ordered Chapters, the ChapterTimeEnd value of a Parent Chapter is useless for playback as the proper playback sections are described in its Nested Chapters. The ChapterTimeEnd SHOULD NOT be set in Parent Chapters and MUST be ignored for playback.

20.2.5. ChapterFlagHidden

Each Chapter ChapterFlagHidden flag works independently of parent chapters. A Nested Chapter with a ChapterFlagHidden that evaluates to "0" remains visible in the user interface even if the Parent Chapter ChapterFlagHidden flag is set to "1".

Chapter + Nested Chapter	ChapterFlagHidden	visible
Chapter 1	0	yes
Nested Chapter 1.1	0	yes
Nested Chapter 1.2	1	no
Chapter 2	1	no
Nested Chapter 2.1	0	yes
Nested Chapter 2.2	1	no

Table 52: ChapterFlagHidden nested visibility

20.3. Menu features

The menu features are handled like a chapter codec. That means each codec has a type, some private data and some data in the chapters.

The type of the menu system is defined by the ChapProcessCodecID parameter. For now, only 2 values are supported : 0 matroska script, 1 menu borrowed from the DVD [DVD-Video]. The private data depend on the type of menu system (stored in ChapProcessPrivate), idem for the data in the chapters (stored in ChapProcessData).

The menu system, as well as Chapter Codecs in general, can do actions on the Matroska Player like jumping to another Chapter or Edition, selecting different tracks and possibly more. The scope of all the possibilities of Chapter Codecs is not covered in this document as it depends on the Chapter Codec features and its integration in a Matroska Player.

20.4. Physical Types

Each level can have different meanings for audio and video. The ORIGINAL_MEDIA_TYPE tag [MatroskaTags] can be used to specify a string for ChapterPhysicalEquiv = 60. Here is the list of possible levels for both audio and video:

Value	Audio	Video	Comment
70	SET / PACKAGE	SET / PACKAGE	the collection of different media
60	CD / 12" / 10" / 7" / TAPE / MINIDISC / DAT	DVD / VHS / LASERDISC	the physical medium like a CD or a DVD
50	SIDE	SIDE	when the original medium (LP/DVD) has different sides
40	-	LAYER	another physical level on DVDs
30	SESSION	SESSION	as found on CDs and DVDs
20	TRACK	-	as found on audio CDs
10	INDEX	-	the first logical level of the side/ medium

Table 53: ChapterPhysicalEquiv meaning per track type

20.5. Chapter Examples

20.5.1. Example 1 : basic chaptering

In this example a movie is split in different chapters. It could also just be an audio file (album) on which each track corresponds to a chapter.

* 00000 ms - 05000 ms : Intro
* 05000 ms - 25000 ms : Before the crime
* 25000 ms - 27500 ms : The crime
* 27500 ms - 38000 ms : The killer arrested
* 38000 ms - 43000 ms : Credits

This would translate in the following matroska form, with the EBML tree shown as XML :

```
   <Chapters>
     <EditionEntry>
       <EditionUID>16603393396715046047</EditionUID>
       <ChapterAtom>
         <ChapterUID>1193046</ChapterUID>
         <ChapterTimeStart>0</ChapterTimeStart>
         <ChapterTimeEnd>5000000000</ChapterTimeEnd>
         <ChapterDisplay>
           <ChapString>Intro</ChapString>
         </ChapterDisplay>
       </ChapterAtom>
       <ChapterAtom>
         <ChapterUID>2311527</ChapterUID>
         <ChapterTimeStart>5000000000</ChapterTimeStart>
         <ChapterTimeEnd>25000000000</ChapterTimeEnd>
         <ChapterDisplay>
           <ChapString>Before the crime</ChapString>
         </ChapterDisplay>
         <ChapterDisplay>
           <ChapString>Avant le crime</ChapString>
           <ChapLanguage>fra</ChapLanguage>
         </ChapterDisplay>
       </ChapterAtom>
       <ChapterAtom>
         <ChapterUID>3430008</ChapterUID>
         <ChapterTimeStart>25000000000</ChapterTimeStart>
         <ChapterTimeEnd>27500000000</ChapterTimeEnd>
         <ChapterDisplay>
           <ChapString>The crime</ChapString>
         </ChapterDisplay>
         <ChapterDisplay>
           <ChapString>Le crime</ChapString>
           <ChapLanguage>fra</ChapLanguage>
         </ChapterDisplay>
       </ChapterAtom>
       <ChapterAtom>
         <ChapterUID>4548489</ChapterUID>
         <ChapterTimeStart>27500000000</ChapterTimeStart>
         <ChapterTimeEnd>38000000000</ChapterTimeEnd>
         <ChapterDisplay>
           <ChapString>After the crime</ChapString>
         </ChapterDisplay>
         <ChapterDisplay>
           <ChapString>Apres le crime</ChapString>
           <ChapLanguage>fra</ChapLanguage>
         </ChapterDisplay>
       </ChapterAtom>
       <ChapterAtom>
```

```
        <ChapterUID>5666960</ChapterUID>
        <ChapterTimeStart>38000000000</ChapterTimeStart>
        <ChapterTimeEnd>43000000000</ChapterTimeEnd>
        <ChapterDisplay>
          <ChapString>Credits</ChapString>
        </ChapterDisplay>
        <ChapterDisplay>
          <ChapString>Generique</ChapString>
          <ChapLanguage>fra</ChapLanguage>
        </ChapterDisplay>
      </ChapterAtom>
    </EditionEntry>
  </Chapters>
```

Figure 16: Basic Chapters Example.

20.5.2. Example 2 : nested chapters

In this example an (existing) album is split into different chapters, and one of them contains another splitting.

20.5.2.1. The Micronauts "Bleep To Bleep"

* 00:00 - 12:28 : Baby Wants To Bleep/Rock
  - 00:00 - 04:38 : Baby wants to bleep (pt.1)
  - 04:38 - 07:12 : Baby wants to rock
  - 07:12 - 10:33 : Baby wants to bleep (pt.2)
  - 10:33 - 12:28 : Baby wants to bleep (pt.3)
* 12:30 - 19:38 : Bleeper_O+2
* 19:40 - 22:20 : Baby wants to bleep (pt.4)
* 22:22 - 25:18 : Bleep to bleep
* 25:20 - 33:35 : Baby wants to bleep (k)
* 33:37 - 44:28 : Bleeper

This would translate in the following matroska form, with the EBML tree shown as XML :

```
<Chapters>
  <EditionEntry>
    <EditionUID>1281690858003401414</EditionUID>
    <ChapterAtom>
      <ChapterUID>1</ChapterUID>
      <ChapterTimeStart>0</ChapterTimeStart>
      <ChapterTimeEnd>748000000</ChapterTimeEnd>
      <ChapterDisplay>
        <ChapString>Baby wants to Bleep/Rock</ChapString>
      </ChapterDisplay>
      <ChapterAtom>
```

```
          <ChapterUID>2</ChapterUID>
          <ChapterTimeStart>0</ChapterTimeStart>
          <ChapterTimeEnd>278000000</ChapterTimeEnd>
          <ChapterDisplay>
            <ChapString>Baby wants to bleep (pt.1)</ChapString>
          </ChapterDisplay>
        </ChapterAtom>
        <ChapterAtom>
          <ChapterUID>3</ChapterUID>
          <ChapterTimeStart>278000000</ChapterTimeStart>
          <ChapterTimeEnd>432000000</ChapterTimeEnd>
          <ChapterDisplay>
            <ChapString>Baby wants to rock</ChapString>
          </ChapterDisplay>
        </ChapterAtom>
        <ChapterAtom>
          <ChapterUID>4</ChapterUID>
          <ChapterTimeStart>432000000</ChapterTimeStart>
          <ChapterTimeEnd>633000000</ChapterTimeEnd>
          <ChapterDisplay>
            <ChapString>Baby wants to bleep (pt.2)</ChapString>
          </ChapterDisplay>
        </ChapterAtom>
        <ChapterAtom>
          <ChapterUID>5</ChapterUID>
          <ChapterTimeStart>633000000</ChapterTimeStart>
          <ChapterTimeEnd>748000000</ChapterTimeEnd>
          <ChapterDisplay>
            <ChapString>Baby wants to bleep (pt.3)</ChapString>
          </ChapterDisplay>
        </ChapterAtom>
      </ChapterAtom>
      <ChapterAtom>
        <ChapterUID>6</ChapterUID>
        <ChapterTimeStart>750000000</ChapterTimeStart>
        <ChapterTimeEnd>1178500000</ChapterTimeEnd>
        <ChapterDisplay>
          <ChapString>Bleeper_O+2</ChapString>
        </ChapterDisplay>
      </ChapterAtom>
      <ChapterAtom>
        <ChapterUID>7</ChapterUID>
        <ChapterTimeStart>1180500000</ChapterTimeStart>
        <ChapterTimeEnd>1340000000</ChapterTimeEnd>
        <ChapterDisplay>
          <ChapString>Baby wants to bleep (pt.4)</ChapString>
        </ChapterDisplay>
      </ChapterAtom>
```

```
      <ChapterAtom>
        <ChapterUID>8</ChapterUID>
        <ChapterTimeStart>1342000000</ChapterTimeStart>
        <ChapterTimeEnd>1518000000</ChapterTimeEnd>
        <ChapterDisplay>
          <ChapString>Bleep to bleep</ChapString>
        </ChapterDisplay>
      </ChapterAtom>
      <ChapterAtom>
        <ChapterUID>9</ChapterUID>
        <ChapterTimeStart>1520000000</ChapterTimeStart>
        <ChapterTimeEnd>2015000000</ChapterTimeEnd>
        <ChapterDisplay>
          <ChapString>Baby wants to bleep (k)</ChapString>
        </ChapterDisplay>
      </ChapterAtom>
      <ChapterAtom>
        <ChapterUID>10</ChapterUID>
        <ChapterTimeStart>2017000000</ChapterTimeStart>
        <ChapterTimeEnd>2668000000</ChapterTimeEnd>
        <ChapterDisplay>
          <ChapString>Bleeper</ChapString>
        </ChapterDisplay>
      </ChapterAtom>
    </EditionEntry>
  </Chapters>
```

Figure 17: Nested Chapters Example.

21. Attachments

Matroska supports storage of related files and data in the Attachments Element (a Top-Level Element). Attachment Elements can be used to store related cover art, font files, transcripts, reports, error recovery files, picture, or text-based annotations, copies of specifications, or other ancillary files related to the Segment.

Matroska Readers MUST NOT execute files stored as Attachment Elements.

21.1. Cover Art

This section defines a set of guidelines for the storage of cover art in Matroska files. A Matroska Reader MAY use embedded cover art to display a representational still-image depiction of the multimedia contents of the Matroska file.

Only [JPEG] and PNG [RFC2083] image formats SHOULD be used for cover art pictures.

There can be two different covers for a movie/album: a portrait style (e.g., a DVD case) and a landscape style (e.g., a wide banner ad).

There can be two versions of the same cover, the normal cover and the small cover. The dimension of the normal cover SHOULD be 600 pixels on the smallest side -- for example, 960x600 for landscape, 600x800 for portrait, or 600x600 for square. The dimension of the small cover SHOULD be 120 pixels on the smallest side -- for example, 192x120 or 120x160.

Versions of cover art can be differentiated by the filename, which is stored in the FileName Element. The default filename of the normal cover in square or portrait mode is cover.(jpg|png). When stored, the normal cover SHOULD be the first Attachment in storage order. The small cover SHOULD be prefixed with "small_", such as small_cover.(jpg|png). The landscape variant SHOULD be suffixed with "_land", such as cover_land.(jpg|png). The filenames are case-sensitive.

The following table provides examples of file names for cover art in Attachments.

FileName	Image Orientation	Pixel Length of Smallest Side
cover.jpg	Portrait or square	600
small_cover.png	Portrait or square	120
cover_land.png	Landscape	600
small_cover_land.jpg	Landscape	120

Table 54: Cover Art Filenames

21.2. Font files

Font files MAY be added to a Matroska file as Attachments so that the font file may be used to display an associated subtitle track. This allows the presentation of a Matroska file to be consistent in various environments where the needed fonts might not be available on the local system.

Depending on the font format in question, each font file can contain multiple font variants. Each font variant has a name which will be referred to as Font Name from now on. This Font Name can be different from the Attachment's FileName, even when disregarding the extension. In order to select a font for display, a Matroska player SHOULD consider both the Font Name and the base name of the Attachment's FileName, preferring the former when there are multiple matches.

Subtitle codecs, such as SubStation Alpha (SSA/ASS), usually refer to a font by its Font Name, not by its filename. If none of the Attachments are a match for the Font Name, the Matroska player SHOULD attempt to find a system font whose Font Name matches the one used in the subtitle track.

Since loading fonts temporarily can take a while, a Matroska player usually loads or installs all the fonts found in attachments so they are ready to be used during playback. Failure to use the font attachment might result in incorrect rendering of the subtitles.

If a selected subtitle track has some AttachmentLink elements, the player MAY restrict its font rendering to use only these fonts.

A Matroska player SHOULD handle the official font media types from [RFC8081] when the system can handle the type:

* font/sfnt: Generic SFNT Font Type,
* font/ttf: TTF Font Type,
* font/otf: OpenType Layout (OTF) Font Type,
* font/collection: Collection Font Type,
* font/woff: WOFF 1.0,
* font/woff2: WOFF 2.0.

Fonts in Matroska existed long before [RFC8081]. A few unofficial media types for fonts were used in existing files. Therefore, it is RECOMMENDED for a Matroska player to support the following legacy media types for font attachments:

* application/x-truetype-font: Truetype fonts, equivalent to font/ttf and sometimes font/otf,

* application/x-font-ttf: TTF fonts, equivalent to font/ttf,

* application/vnd.ms-opentype: OpenType Layout fonts, equivalent to font/otf

* application/font-sfnt: Generic SFNT Font Type, equivalent to font/sfnt

* application/font-woff: WOFF 1.0, equivalent to font/woff

There may also be some font attachments with the application/octet-stream media type. In that case the Matroska player MAY try to guess the font type by checking the file extension of the AttachedFile\FileName string. Common file extensions for fonts are:

* .ttf for Truetype fonts, equivalent to font/ttf,

* .otf for OpenType Layout fonts, equivalent to font/otf,

* .ttc for Collection fonts, equivalent to font/collection

The file extension check MUST be case-insensitive.

Matroska writers SHOULD use a valid font media type from [RFC8081] in the AttachedFile\FileMediaType of the font attachment. They MAY use the media types found in older files when compatibility with older players is necessary.

22. Cues

The Cues Element provides an index of certain Cluster Elements to allow for optimized seeking to absolute timestamps within the Segment. The Cues Element contains one or many CuePoint Elements which each MUST reference an absolute timestamp (via the CueTime Element), a Track (via the CueTrack Element), and a Segment Position (via the CueClusterPosition Element). Additional non-mandated Elements are part of the CuePoint Element such as CueDuration, CueRelativePosition, CueCodecState and others which provide any Matroska Reader with additional information to use in the optimization of seeking performance.

22.1. Recommendations

The following recommendations are provided to optimize Matroska performance.

* Unless Matroska is used as a live stream, it SHOULD contain a Cues Element.

* For each video track, each keyframe SHOULD be referenced by a CuePoint Element.

* It is RECOMMENDED to not reference non-keyframes of video tracks in Cues unless it references a Cluster Element which contains a CodecState Element but no keyframes.

* For each subtitle track present, each subtitle frame SHOULD be referenced by a CuePoint Element with a CueDuration Element.

* References to audio tracks MAY be skipped in CuePoint Elements if a video track is present. When included the CuePoint Elements SHOULD reference audio keyframes at most once every 500 milliseconds.

* If the referenced frame is not stored within the first SimpleBlock, or first BlockGroup within its Cluster Element, then the CueRelativePosition Element SHOULD be written to reference where in the Cluster the reference frame is stored.

* If a CuePoint Element references Cluster Element that includes a CodecState Element, then that CuePoint Element MUST use a CueCodecState Element.

* CuePoint Elements SHOULD be numerically sorted in storage order by the value of the CueTime Element.

23. Matroska Streaming

In Matroska, there are two kinds of streaming: file access and livestreaming.

23.1. File Access

File access can simply be reading a file located on your computer, but also includes accessing a file from an HTTP (web) server or CIFS (Windows share) server. These protocols are usually safe from reading errors and seeking in the stream is possible. However, when a file is stored far away or on a slow server, seeking can be an expensive operation and should be avoided. The guidelines in

Section 25, when followed, help reduce the number of seeking operations for regular playback and also have the playback start quickly without a lot of data needed to read first (like a Cues Element, Attachment Element or SeekHead Element).

Matroska, having a small overhead, is well suited for storing music/videos on file servers without a big impact on the bandwidth used. Matroska does not require the index to be loaded before playing, which allows playback to start very quickly. The index can be loaded only when seeking is requested the first time.

23.2. Livestreaming

Livestreaming is the equivalent of television broadcasting on the internet. There are 2 families of servers for livestreaming: RTP/RTSP and HTTP. Matroska is not meant to be used over RTP. RTP already has timing and channel mechanisms that would be wasted if doubled in Matroska. Additionally, having the same information at the RTP and Matroska level would be a source of confusion if they do not match. Livestreaming of Matroska over file-like protocols like HTTP, QUIC, etc. is possible.

A live Matroska stream is different from a file because it usually has no known end (only ending when the client disconnects). For this, all bits of the "size" portion of the Segment Element MUST be set to 1. Another option is to concatenate Segment Elements with known sizes, one after the other. This solution allows a change of codec/resolution between each segment. For example, this allows for a switch between 4:3 and 16:9 in a television program.

When Segment Elements are continuous, certain Elements, like SeekHead, Cues, Chapters, and Attachments, MUST NOT be used.

It is possible for a Matroska Player to detect that a stream is not seekable. If the stream has neither a SeekHead list nor a Cues list at the beginning of the stream, it SHOULD be considered non-seekable. Even though it is possible to seek forward in the stream, it is NOT RECOMMENDED.

In the context of live radio or web TV, it is possible to "tag" the content while it is playing. The Tags Element can be placed between Clusters each time it is necessary. In that case, the new Tags Element MUST reset the previously encountered Tags Elements and use the new values instead.

24. Tags

24.1. Tags Precedence

Tags allow tagging all kinds of Matroska parts with very detailed metadata in multiple languages.

Some Matroska elements also contain their own string value like the Track Name (Section 5.1.4.1.18) or the Chapter String (Section 5.1.7.1.4.10).

The following Matroska elements can also be defined with tags:

* The Track Name Element (Section 5.1.4.1.18) corresponds to a tag with the TagTrackUID (Section 5.1.8.1.1.3) set to the given track, a TagName of TITLE (Section 5.1.8.1.2.1) and a TagLanguage (Section 5.1.8.1.2.2) or TagLanguageBCP47 (Section 5.1.8.1.2.3) of "und".

* The Chapter String Element (Section 5.1.7.1.4.10) corresponds to a tag with the TagChapterUID (Section 5.1.8.1.1.5) set to the same chapter UID, a TagName of TITLE (Section 5.1.8.1.2.1) and a TagLanguage (Section 5.1.8.1.2.2) or TagLanguageBCP47 (Section 5.1.8.1.2.3) matching the ChapLanguage (Section 5.1.7.1.4.11) or ChapLanguageBCP47 (Section 5.1.7.1.4.12) respectively.

* The FileDescription Element (Section 5.1.6.1.1) of an attachment corresponds to a tag with the TagAttachmentUID (Section 5.1.8.1.1.6) set to the given attachment, a TagName of TITLE (Section 5.1.8.1.2.1) and a TagLanguage (Section 5.1.8.1.2.2) or TagLanguageBCP47 (Section 5.1.8.1.2.3) of "und".

When both values exist in the file, the value found in Tags takes precedence over the value found in original location of the element. For example, if you have a TrackEntry\Name element and Tag TITLE for that track in a Matroska Segment, the Tag string SHOULD be used and not the TrackEntry\Name string to identify the track.

As the Tag element is optional, a lot of Matroska Readers do not handle it and will not use the tags value when it’s found. So for maximum compatibility, it’s usually better to put the strings in the TrackEntry, ChapterAtom and Attachment and keep the tags matching these values if tags are also used.

24.2. Tag Levels

Tag elements allow tagging information on multiple levels, each level having a TargetTypeValue Section 5.1.8.1.1.1. An element for a given TargetTypeValue also applies to the lower levels denoted by smaller TargetTypeValue values. If an upper value doesn't apply to a level but the actual value to use is not known, an empty TagString (Section 5.1.8.1.2.5) or an empty TagBinary (Section 5.1.8.1.2.6) element MUST be used as the tag value for this level.

See [MatroskaTags] for more details on common tag names, types and descriptions.

25. Implementation Recommendations

25.1. Cluster

It is RECOMMENDED that each individual Cluster Element contains no more than 5 seconds or 5 megabytes of content.

25.2. SeekHead

It is RECOMMENDED that the first SeekHead Element be followed by a Void Element to allow for the SeekHead Element to be expanded to cover new Top-Level Elements that could be added to the Matroska file, such as Tags, Chapters, and Attachments Elements.

The size of this Void Element should be adjusted depending on the Matroska file already having Tags, Chapters, and Attachments Elements.

25.3. Optimum Layouts

While there can be Top-Level Elements in any order, some ordering of Elements are better than others. Here are few optimum layouts for different use case:

25.3.1. Optimum layout for a muxer

This is the basic layout muxers should be using for an efficient playback experience.

* SeekHead
* Info
* Tracks
* Chapters
* Attachments
* Tags

* Clusters
* Cues

25.3.2. Optimum layout after editing tags

When tags from the previous layout need to be extended, they are moved to the end with the extra information. The location where the old tags were located is voided.

* SeekHead
* Info
* Tracks
* Chapters
* Attachments
* Void
* Clusters
* Cues
* Tags

25.3.3. Optimum layout with Cues at the front

Cues are usually a big chunk of data referencing a lot of locations in the file. For players that want to seek in the file they need to seek to the end of the file to access these locations. It is often better if they are placed early in the file. On the other hand that means players that don't intend to seek will have to read/skip these data no matter what.

Because the Cues reference locations further in the file, it's often complicated to allocate the proper space for that element before all the locations are known. Therefore, this layout is rarely used.

* SeekHead
* Info
* Tracks
* Chapters
* Attachments
* Tags
* Cues
* Clusters

25.3.4. Optimum layout for livestreaming

In Livestreaming (Section 23.2) only a few elements make sense. SeekHead and Cues are useless for example. All elements other than the Clusters MUST be placed before the Clusters.

* Info

* Tracks
* Attachments (rare)
* Tags
* Clusters

26. Security Considerations

Matroska inherits security considerations from EBML.

Attacks on a Matroska Reader could include:

* Storage of an arbitrary and potentially executable data within an Attachment Element. Matroska Readers that extract or use data from Matroska Attachments SHOULD check that the data adheres to expectations or not use the attachement.

* A Matroska Attachment with an inaccurate media type.

* Damage to the Encryption and Compression fields (Section 14) that would result in bogus binary data interpreted by the decoder.

* Chapter Codecs running unwanted commands on the host system.

The same error handling done for EBML applies to Matroska files. Particular error handling is not covered in this specification as this is depends on the goal of the Matroska Readers. It is up to the decision of the Matroska Readers on how to handle the errors if they are recoverable in their code or not. For example, if the checksum of the \Segment\Tracks is invalid some could decide to try to read the data anyway, some will just reject the file, most will not even check it.

Matroska Reader implementations need to be robust against malicious payloads. Those related to denial of service are outlined in Section 2.1 of [RFC4732].
Although rarer, the same may apply to a Matroska Writer. Malicious stream data must not cause the Writer to misbehave, as this might allow an attacker access to transcoding gateways.

As an audio and visual container format, a Matroska file or stream will potentially encapsulate numerous byte streams created with a variety of codecs. Implementers will need to consider the security considerations of these encapsulated formats.

27. IANA Considerations

27.1. Matroska Element IDs Registry

This document creates a new IANA registry called the "Matroska Element IDs" registry.

To register a new Element ID in this registry, one needs an Element ID, a Change Controller (IETF or email of registrant) and an optional Reference to a document describing the Element ID.

Element IDs are encoded using the VINT mechanism described in Section 4 of [RFC8794] and can be between one and five octets long. Five-octet-long Element IDs are possible only if declared in the EBML header.

Element IDs are described in Section 5 of [RFC8794] with errata 7189 and 7191.

One-octet Matroska Element IDs are to be allocated according to the "RFC Required" policy [RFC8126].

Two-octet Matroska Element IDs are to be allocated according to the "Specification Required" policy [RFC8126].

Three-octet and four-octet Matroska Element IDs are to be allocated according to the "First Come First Served" policy [RFC8126].

The allowed values in the Elements IDs registry are similar to the ones found in the EBML Element IDs registry defined in Section 17.1 of [RFC8794].

EBML IDs defined for the EBML Header -- as defined in Section 17.1 of [RFC8794] -- MUST NOT be used as Matroska Element IDs.

Given the scarcity of the One-octet Element IDs, they should only be created to save space for elements found many times in a file. For example, within a BlockGroup or Chapters. The Four-octet Element IDs are mostly for synchronization of large elements. They should only be used for such high level elements. Elements that are not expected to be used often should use Three-octet Element IDs.

Elements found in Section 28 have an assigned Matroska Element ID for historical reasons. These elements are not in use and SHOULD NOT be reused unless there is no other IDs available with the desired size. Such IDs are considered as reclaimed to the IANA registry as they could be used for other things in the future.

Matroska Element IDs Values found in this document are assigned as initial values as follows:

Element ID	Element Name	Reference
0x80	ChapterDisplay	Described in Section 5.1.7.1.4.9
0x83	TrackType	Described in Section 5.1.4.1.3
0x85	ChapString	Described in Section 5.1.7.1.4.10
0x86	CodecID	Described in Section 5.1.4.1.21
0x88	FlagDefault	Described in Section 5.1.4.1.5
0x8E	Slices	Reclaimed (Section 28.5)
0x91	ChapterTimeStart	Described in Section 5.1.7.1.4.3
0x92	ChapterTimeEnd	Described in Section 5.1.7.1.4.4
0x96	CueRefTime	Described in Section 5.1.5.1.2.8
0x97	CueRefCluster	Reclaimed (Section 28.37)
0x98	ChapterFlagHidden	Described in Section 5.1.7.1.4.5
0x9A	FlagInterlaced	Described in Section 5.1.4.1.28.1
0x9B	BlockDuration	Described in Section 5.1.3.5.3
0x9C	FlagLacing	Described in Section 5.1.4.1.12
0x9D	FieldOrder	Described in Section 5.1.4.1.28.2

0x9F	Channels	Described in Section 5.1.4.1.29.3
0xA0	BlockGroup	Described in Section 5.1.3.5
0xA1	Block	Described in Section 5.1.3.5.1
0xA2	BlockVirtual	Reclaimed (Section 28.3)
0xA3	SimpleBlock	Described in Section 5.1.3.4
0xA4	CodecState	Described in Section 5.1.3.5.6
0xA5	BlockAdditional	Described in Section 5.1.3.5.2.2
0xA6	BlockMore	Described in Section 5.1.3.5.2.1
0xA7	Position	Described in Section 5.1.3.2
0xAA	CodecDecodeAll	Reclaimed (Section 28.22)
0xAB	PrevSize	Described in Section 5.1.3.3
0xAE	TrackEntry	Described in Section 5.1.4.1
0xAF	EncryptedBlock	Reclaimed (Section 28.15)
0xB0	PixelWidth	Described in Section 5.1.4.1.28.6
0xB2	CueDuration	Described in Section 5.1.5.1.2.4
0xB3	CueTime	Described in Section 5.1.5.1.1

0xB5	SamplingFrequency	Described in Section 5.1.4.1.29.1
0xB6	ChapterAtom	Described in Section 5.1.7.1.4
0xB7	CueTrackPositions	Described in Section 5.1.5.1.2
0xB9	FlagEnabled	Described in Section 5.1.4.1.4
0xBA	PixelHeight	Described in Section 5.1.4.1.28.7
0xBB	CuePoint	Described in Section 5.1.5.1
0xC0	TrickTrackUID	Reclaimed (Section 28.28)
0xC1	TrickTrackSegmentUID	Reclaimed (Section 28.29)
0xC4	TrickMasterTrackSegmentUID	Reclaimed (Section 28.32)
0xC6	TrickTrackFlag	Reclaimed (Section 28.30)
0xC7	TrickMasterTrackUID	Reclaimed (Section 28.31)
0xC8	ReferenceFrame	Reclaimed (Section 28.12)
0xC9	ReferenceOffset	Reclaimed (Section 28.13)
0xCA	ReferenceTimestamp	Reclaimed (Section 28.14)
0xCB	BlockAdditionID	Reclaimed (Section 28.9)
0xCC	LaceNumber	Reclaimed (Section 28.7)

0xCD	FrameNumber	Reclaimed (Section 28.8)
0xCE	Delay	Reclaimed (Section 28.10)
0xCF	SliceDuration	Reclaimed (Section 28.11)
0xD7	TrackNumber	Described in Section 5.1.4.1.1
0xDB	CueReference	Described in Section 5.1.5.1.2.7
0xE0	Video	Described in Section 5.1.4.1.28
0xE1	Audio	Described in Section 5.1.4.1.29
0xE2	TrackOperation	Described in Section 5.1.4.1.30
0xE3	TrackCombinePlanes	Described in Section 5.1.4.1.30.1
0xE4	TrackPlane	Described in Section 5.1.4.1.30.2
0xE5	TrackPlaneUID	Described in Section 5.1.4.1.30.3
0xE6	TrackPlaneType	Described in Section 5.1.4.1.30.4
0xE7	Timestamp	Described in Section 5.1.3.1
0xE8	TimeSlice	Reclaimed (Section 28.6)
0xE9	TrackJoinBlocks	Described in Section 5.1.4.1.30.5
0xEA	CueCodecState	Described in Section 5.1.5.1.2.6

0xEB	CueRefCodecState	Reclaimed (Section 28.39)
0xED	TrackJoinUID	Described in Section 5.1.4.1.30.6
0xEE	BlockAddID	Described in Section 5.1.3.5.2.3
0xF0	CueRelativePosition	Described in Section 5.1.5.1.2.3
0xF1	CueClusterPosition	Described in Section 5.1.5.1.2.2
0xF7	CueTrack	Described in Section 5.1.5.1.2.1
0xFA	ReferencePriority	Described in Section 5.1.3.5.4
0xFB	ReferenceBlock	Described in Section 5.1.3.5.5
0xFD	ReferenceVirtual	Reclaimed (Section 28.4)
0x41A4	BlockAddIDName	Described in Section 5.1.4.1.17.2
0x41E4	BlockAdditionMapping	Described in Section 5.1.4.1.17
0x41E7	BlockAddIDType	Described in Section 5.1.4.1.17.3
0x41ED	BlockAddIDExtraData	Described in Section 5.1.4.1.17.4
0x41F0	BlockAddIDValue	Described in Section 5.1.4.1.17.1
0x4254	ContentCompAlgo	Described in Section 5.1.4.1.31.6
0x4255	ContentCompSettings	Described in Section 5.1.4.1.31.7

0x437C	ChapLanguage	Described in Section 5.1.7.1.4.11
0x437D	ChapLanguageBCP47	Described in Section 5.1.7.1.4.12
0x437E	ChapCountry	Described in Section 5.1.7.1.4.13
0x4444	SegmentFamily	Described in Section 5.1.2.7
0x4461	DateUTC	Described in Section 5.1.2.11
0x447A	TagLanguage	Described in Section 5.1.8.1.2.2
0x447B	TagLanguageBCP47	Described in Section 5.1.8.1.2.3
0x4484	TagDefault	Described in Section 5.1.8.1.2.4
0x4485	TagBinary	Described in Section 5.1.8.1.2.6
0x4487	TagString	Described in Section 5.1.8.1.2.5
0x4489	Duration	Described in Section 5.1.2.10
0x44B4	TagDefaultBogus	Reclaimed (Section 28.43)
0x450D	ChapProcessPrivate	Described in Section 5.1.7.1.4.16
0x45A3	TagName	Described in Section 5.1.8.1.2.1
0x45B9	EditionEntry	Described in Section 5.1.7.1
0x45BC	EditionUID	Described in Section 5.1.7.1.1

0x45DB	EditionFlagDefault	Described in Section 5.1.7.1.2
0x45DD	EditionFlagOrdered	Described in Section 5.1.7.1.3
0x465C	FileData	Described in Section 5.1.6.1.4
0x4660	FileMediaType	Described in Section 5.1.6.1.3
0x4661	FileUsedStartTime	Reclaimed (Section 28.41)
0x4662	FileUsedEndTime	Reclaimed (Section 28.42)
0x466E	FileName	Described in Section 5.1.6.1.2
0x4675	FileReferral	Reclaimed (Section 28.40)
0x467E	FileDescription	Described in Section 5.1.6.1.1
0x46AE	FileUID	Described in Section 5.1.6.1.5
0x47E1	ContentEncAlgo	Described in Section 5.1.4.1.31.9
0x47E2	ContentEncKeyID	Described in Section 5.1.4.1.31.10
0x47E3	ContentSignature	Reclaimed (Section 28.33)
0x47E4	ContentSigKeyID	Reclaimed (Section 28.34)
0x47E5	ContentSigAlgo	Reclaimed (Section 28.35)
0x47E6	ContentSigHashAlgo	Reclaimed (Section 28.36)

0x47E7	ContentEncAESSettings	Described in Section 5.1.4.1.31.11
0x47E8	AESSettingsCipherMode	Described in Section 5.1.4.1.31.12
0x4D80	MuxingApp	Described in Section 5.1.2.13
0x4DBB	Seek	Described in Section 5.1.1.1
0x5031	ContentEncodingOrder	Described in Section 5.1.4.1.31.2
0x5032	ContentEncodingScope	Described in Section 5.1.4.1.31.3
0x5033	ContentEncodingType	Described in Section 5.1.4.1.31.4
0x5034	ContentCompression	Described in Section 5.1.4.1.31.5
0x5035	ContentEncryption	Described in Section 5.1.4.1.31.8
0x535F	CueRefNumber	Reclaimed (Section 28.38)
0x536E	Name	Described in Section 5.1.4.1.18
0x5378	CueBlockNumber	Described in Section 5.1.5.1.2.5
0x537F	TrackOffset	Reclaimed (Section 28.18)
0x53AB	SeekID	Described in Section 5.1.1.1.1
0x53AC	SeekPosition	Described in Section 5.1.1.1.2
0x53B8	StereoMode	Described in Section 5.1.4.1.28.3

0x53B9	OldStereoMode	Described in Section 5.1.4.1.28.5
0x53C0	AlphaMode	Described in Section 5.1.4.1.28.4
0x54AA	PixelCropBottom	Described in Section 5.1.4.1.28.8
0x54B0	DisplayWidth	Described in Section 5.1.4.1.28.12
0x54B2	DisplayUnit	Described in Section 5.1.4.1.28.14
0x54B3	AspectRatioType	Reclaimed (Section 28.24)
0x54BA	DisplayHeight	Described in Section 5.1.4.1.28.13
0x54BB	PixelCropTop	Described in Section 5.1.4.1.28.9
0x54CC	PixelCropLeft	Described in Section 5.1.4.1.28.10
0x54DD	PixelCropRight	Described in Section 5.1.4.1.28.11
0x55AA	FlagForced	Described in Section 5.1.4.1.6
0x55AB	FlagHearingImpaired	Described in Section 5.1.4.1.7
0x55AC	FlagVisualImpaired	Described in Section 5.1.4.1.8
0x55AD	FlagTextDescriptions	Described in Section 5.1.4.1.9
0x55AE	FlagOriginal	Described in Section 5.1.4.1.10
0x55AF	FlagCommentary	Described in Section 5.1.4.1.11

0x55B0	Colour	Described in Section 5.1.4.1.28.16
0x55B1	MatrixCoefficients	Described in Section 5.1.4.1.28.17
0x55B2	BitsPerChannel	Described in Section 5.1.4.1.28.18
0x55B3	ChromaSubsamplingHorz	Described in Section 5.1.4.1.28.19
0x55B4	ChromaSubsamplingVert	Described in Section 5.1.4.1.28.20
0x55B5	CbSubsamplingHorz	Described in Section 5.1.4.1.28.21
0x55B6	CbSubsamplingVert	Described in Section 5.1.4.1.28.22
0x55B7	ChromaSitingHorz	Described in Section 5.1.4.1.28.23
0x55B8	ChromaSitingVert	Described in Section 5.1.4.1.28.24
0x55B9	Range	Described in Section 5.1.4.1.28.25
0x55BA	TransferCharacteristics	Described in Section 5.1.4.1.28.26
0x55BB	Primaries	Described in Section 5.1.4.1.28.27
0x55BC	MaxCLL	Described in Section 5.1.4.1.28.28
0x55BD	MaxFALL	Described in Section 5.1.4.1.28.29
0x55D0	MasteringMetadata	Described in Section 5.1.4.1.28.30
0x55D1	PrimaryRChromaticityX	Described in Section 5.1.4.1.28.31

0x55D2	PrimaryRChromaticityY	Described in Section 5.1.4.1.28.32
0x55D3	PrimaryGChromaticityX	Described in Section 5.1.4.1.28.33
0x55D4	PrimaryGChromaticityY	Described in Section 5.1.4.1.28.34
0x55D5	PrimaryBChromaticityX	Described in Section 5.1.4.1.28.35
0x55D6	PrimaryBChromaticityY	Described in Section 5.1.4.1.28.36
0x55D7	WhitePointChromaticityX	Described in Section 5.1.4.1.28.37
0x55D8	WhitePointChromaticityY	Described in Section 5.1.4.1.28.38
0x55D9	LuminanceMax	Described in Section 5.1.4.1.28.39
0x55DA	LuminanceMin	Described in Section 5.1.4.1.28.40
0x55EE	MaxBlockAdditionID	Described in Section 5.1.4.1.16
0x5654	ChapterStringUID	Described in Section 5.1.7.1.4.2
0x56AA	CodecDelay	Described in Section 5.1.4.1.25
0x56BB	SeekPreRoll	Described in Section 5.1.4.1.26
0x5741	WritingApp	Described in Section 5.1.2.14
0x5854	SilentTracks	Reclaimed (Section 28.1)
0x58D7	SilentTrackNumber	Reclaimed (Section 28.2)

0x61A7	AttachedFile	Described in Section 5.1.6.1
0x6240	ContentEncoding	Described in Section 5.1.4.1.31.1
0x6264	BitDepth	Described in Section 5.1.4.1.29.4
0x63A2	CodecPrivate	Described in Section 5.1.4.1.22
0x63C0	Targets	Described in Section 5.1.8.1.1
0x63C3	ChapterPhysicalEquiv	Described in Section 5.1.7.1.4.8
0x63C4	TagChapterUID	Described in Section 5.1.8.1.1.5
0x63C5	TagTrackUID	Described in Section 5.1.8.1.1.3
0x63C6	TagAttachmentUID	Described in Section 5.1.8.1.1.6
0x63C9	TagEditionUID	Described in Section 5.1.8.1.1.4
0x63CA	TargetType	Described in Section 5.1.8.1.1.2
0x6624	TrackTranslate	Described in Section 5.1.4.1.27
0x66A5	TrackTranslateTrackID	Described in Section 5.1.4.1.27.1
0x66BF	TrackTranslateCodec	Described in Section 5.1.4.1.27.2
0x66FC	TrackTranslateEditionUID	Described in Section 5.1.4.1.27.3
0x67C8	SimpleTag	Described in Section 5.1.8.1.2

0x68CA	TargetTypeValue	Described in Section 5.1.8.1.1.1
0x6911	ChapProcessCommand	Described in Section 5.1.7.1.4.17
0x6922	ChapProcessTime	Described in Section 5.1.7.1.4.18
0x6924	ChapterTranslate	Described in Section 5.1.2.8
0x6933	ChapProcessData	Described in Section 5.1.7.1.4.19
0x6944	ChapProcess	Described in Section 5.1.7.1.4.14
0x6955	ChapProcessCodecID	Described in Section 5.1.7.1.4.15
0x69A5	ChapterTranslateID	Described in Section 5.1.2.8.1
0x69BF	ChapterTranslateCodec	Described in Section 5.1.2.8.2
0x69FC	ChapterTranslateEditionUID	Described in Section 5.1.2.8.3
0x6D80	ContentEncodings	Described in Section 5.1.4.1.31
0x6DE7	MinCache	Reclaimed (Section 28.16)
0x6DF8	MaxCache	Reclaimed (Section 28.17)
0x6E67	ChapterSegmentUUID	Described in Section 5.1.7.1.4.6
0x6EBC	ChapterSegmentEditionUID	Described in Section 5.1.7.1.4.7
0x6FAB	TrackOverlay	Reclaimed (Section 28.23)

0x7373	Tag	Described in Section 5.1.8.1
0x7384	SegmentFilename	Described in Section 5.1.2.2
0x73A4	SegmentUUID	Described in Section 5.1.2.1
0x73C4	ChapterUID	Described in Section 5.1.7.1.4.1
0x73C5	TrackUID	Described in Section 5.1.4.1.2
0x7446	AttachmentLink	Described in Section 5.1.4.1.24
0x75A1	BlockAdditions	Described in Section 5.1.3.5.2
0x75A2	DiscardPadding	Described in Section 5.1.3.5.7
0x7670	Projection	Described in Section 5.1.4.1.28.41
0x7671	ProjectionType	Described in Section 5.1.4.1.28.42
0x7672	ProjectionPrivate	Described in Section 5.1.4.1.28.43
0x7673	ProjectionPoseYaw	Described in Section 5.1.4.1.28.44
0x7674	ProjectionPosePitch	Described in Section 5.1.4.1.28.45
0x7675	ProjectionPoseRoll	Described in Section 5.1.4.1.28.46
0x78B5	OutputSamplingFrequency	Described in Section 5.1.4.1.29.2
0x7BA9	Title	Described in Section 5.1.2.12

0x7D7B	ChannelPositions	Reclaimed (Section 28.27)
0x22B59C	Language	Described in Section 5.1.4.1.19
0x22B59D	LanguageBCP47	Described in Section 5.1.4.1.20
0x23314F	TrackTimestampScale	Described in Section 5.1.4.1.15
0x234E7A	DefaultDecodedFieldDuration	Described in Section 5.1.4.1.14
0x2383E3	FrameRate	Reclaimed (Section 28.26)
0x23E383	DefaultDuration	Described in Section 5.1.4.1.13
0x258688	CodecName	Described in Section 5.1.4.1.23
0x26B240	CodecDownloadURL	Reclaimed (Section 28.21)
0x2AD7B1	TimestampScale	Described in Section 5.1.2.9
0x2EB524	UncompressedFourCC	Described in Section 5.1.4.1.28.15
0x2FB523	GammaValue	Reclaimed (Section 28.25)
0x3A9697	CodecSettings	Reclaimed (Section 28.19)
0x3B4040	CodecInfoURL	Reclaimed (Section 28.20)
0x3C83AB	PrevFilename	Described in Section 5.1.2.4
0x3CB923	PrevUUID	Described in Section 5.1.2.3

0x3E83BB	NextFilename	Described in Section 5.1.2.6
0x3EB923	NextUUID	Described in Section 5.1.2.5
0x1043A770	Chapters	Described in Section 5.1.7
0x114D9B74	SeekHead	Described in Section 5.1.1
0x1254C367	Tags	Described in Section 5.1.8
0x1549A966	Info	Described in Section 5.1.2
0x1654AE6B	Tracks	Described in Section 5.1.4
0x18538067	Segment	Described in Section 5.1
0x1941A469	Attachments	Described in Section 5.1.6
0x1C53BB6B	Cues	Described in Section 5.1.5
0x1F43B675	Cluster	Described in Section 5.1.3

Table 55: IDs and Names for Matroska Element IDs assigned by this document

27.2. Chapter Codec IDs Registry

This document creates a new IANA registry called the "Matroska Chapter Codec IDs" registry. The values correspond to the unsigned integer ChapProcessCodecID value described in Section 5.1.7.1.4.15.

To register a new Chapter Codec ID in this registry, one needs a Chapter Codec ID, a Change Controller (IETF or email of registrant) and an optional Reference to a document describing the Chapter Codec ID.

The Chapter Codec IDs are to be allocated according to the "First Come First Served" policy [RFC8126].

ChapProcessCodecID values of "0" and "1" are RESERVED to the IETF for future use.

27.3. Media Types

Matroska files and streams are found in three main forms: audio-video files, audio-only and occasionally with stereoscopic video tracks.

Historically Matroska files and streams have used the following media types with a "x-" prefix. For better compatibility a system SHOULD be able to handle both formats. Newer systems SHOULD NOT use the historic format and use the format that follows the [RFC6838] format instead.

Please register three media types, the [RFC6838] templates are below:

27.3.1. For files containing video tracks

Type name: video
Subtype name: matroska
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: as per this document and RFC8794
Security considerations: See Section 26.
Interoperability considerations: Due to the extensibility of Matroska, it is possible to encounter files with unknown but valid EBML Elements. Readers should be ready to handle this case. The fixed byte order, octet boundaries and UTF-8 usage allow for broad interoparability.
Published specification: THISRFC
Applications that use this media type: FFmpeg, VLC, ...
Fragment identifier considerations: N/A

Additional information:

* Deprecated alias names for this type: video/x-matroska

* Magic number(s): N/A

* File extension(s): mkv

* Macintosh file type code(s): N/A

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

Intended usage: COMMON
Restrictions on usage: None
Author: IETF CELLAR WG
Change controller: IETF
Provisional registration? (standards tree only): No

27.3.2. For files containing audio tracks with no video tracks

Type name: audio
Subtype name: matroska
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: as per this document and RFC8794
Security considerations: See Section 26.
Interoperability considerations: Due to the extensibility of Matroska, it is possible to encounter files with unknown but valid EBML Elements. Readers should be ready to handle this case. The fixed byte order, octet boundaries and UTF-8 usage allow for broad interoparability.
Published specification: THISRFC
Applications that use this media type: FFmpeg, VLC, ...
Fragment identifier considerations: N/A

Additional information:

* Deprecated alias names for this type: audio/x-matroska

* Magic number(s): N/A

* File extension(s): mka

* Macintosh file type code(s): N/A

Person & email address to contact for further information: IETF CELLAR WG cellar@ietf.org
Intended usage: COMMON
Restrictions on usage: None
Author: IETF CELLAR WG
Change controller: IETF
Provisional registration? (standards tree only): No

27.3.3. For files containing a stereoscopic video track

Type name: video
Subtype name: matroska-3d
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: as per this document and RFC8794

Security considerations: See Section 26.
Interoperability considerations: Due to the extensibility of Matroska, it is possible to encounter files with unknown but valid EBML Elements. Readers should be ready to handle this case. The fixed byte order, octet boundaries and UTF-8 usage allow for broad interoparability.
Published specification: THISRFC
Applications that use this media type: FFmpeg, VLC, ...
Fragment identifier considerations: N/A

Additional information:

* Deprecated alias names for this type: video/x-matroska-3d

* Magic number(s): N/A

* File extension(s): mk3d

* Macintosh file type code(s): N/A

Person & email address to contact for further information: IETF CELLAR WG cellar@ietf.org
Intended usage: COMMON
Restrictions on usage: None
Author: IETF CELLAR WG
Change controller: IETF
Provisional registration? (standards tree only): No

28. Annex A: Historic Deprecated Elements

As Matroska evolved since 2002 many parts that were considered for use in the format were never used and often incorrectly designed. Many of the elements that were then defined are not found in any known files but were part of public specs. DivX also had a few custom elements that were designed for custom features.

We list these elements that have a known ID that SHOULD NOT be reused to avoid colliding with existing files. They might be reassigned by IANA in the future if there are no more IDs for a given size. A short description of what each ID was used for is included, but the text is not normative.

28.1. SilentTracks Element

type / id: master / 0x5854
path: \Segment\Cluster\SilentTracks
documentation: The list of tracks that are not used in that part of

the stream. It is useful when using overlay tracks on seeking or to decide what track to use.

28.2. SilentTrackNumber Element

type / id: uinteger / 0x58D7
path: \Segment\Cluster\SilentTracks\SilentTrackNumber
documentation: One of the track number that are not used from now on in the stream. It could change later if not specified as silent in a further Cluster.

28.3. BlockVirtual Element

type / id: binary / 0xA2
path: \Segment\Cluster\BlockGroup\BlockVirtual
documentation: A Block with no data. It must be stored in the stream at the place the real Block would be in display order.

28.4. ReferenceVirtual Element

type / id: integer / 0xFD
path: \Segment\Cluster\BlockGroup\ReferenceVirtual
documentation: The Segment Position of the data that would otherwise be in position of the virtual block.

28.5. Slices Element

type / id: master / 0x8E
path: \Segment\Cluster\BlockGroup\Slices
documentation: Contains slices description.

28.6. TimeSlice Element

type / id: master / 0xE8
path: \Segment\Cluster\BlockGroup\Slices\TimeSlice
documentation: Contains extra time information about the data contained in the Block. Being able to interpret this Element is not required for playback.

28.7. LaceNumber Element

type / id: uinteger / 0xCC
path: \Segment\Cluster\BlockGroup\Slices\TimeSlice\LaceNumber
documentation: The reverse number of the frame in the lace (0 is the last frame, 1 is the next to last, etc.). Being able to interpret this Element is not required for playback.

28.8. FrameNumber Element

type / id: uinteger / 0xCD
path: \Segment\Cluster\BlockGroup\Slices\TimeSlice\FrameNumber
documentation: The number of the frame to generate from this lace with this delay (allow you to generate many frames from the same Block/Frame).

28.9. BlockAdditionID Element

type / id: uinteger / 0xCB
path: \Segment\Cluster\BlockGroup\Slices\TimeSlice\BlockAdditionID
documentation: The ID of the BlockAdditional Element (0 is the main Block).

28.10. Delay Element

type / id: uinteger / 0xCE
path: \Segment\Cluster\BlockGroup\Slices\TimeSlice\Delay
documentation: The delay to apply to the Element, expressed in Track Ticks; see Section 11.1.

28.11. SliceDuration Element

type / id: uinteger / 0xCF
path: \Segment\Cluster\BlockGroup\Slices\TimeSlice\SliceDuration
documentation: The duration to apply to the Element, expressed in Track Ticks; see Section 11.1.

28.12. ReferenceFrame Element

type / id: master / 0xC8
path: \Segment\Cluster\BlockGroup\ReferenceFrame
documentation: Contains information about the last reference frame. See [DivXTrickTrack].

28.13. ReferenceOffset Element

type / id: uinteger / 0xC9
path: \Segment\Cluster\BlockGroup\ReferenceFrame\ReferenceOffset
documentation: The relative offset, in bytes, from the previous BlockGroup element for this Smooth FF/RW video track to the containing BlockGroup element. See [DivXTrickTrack].

28.14. ReferenceTimestamp Element

type / id: uinteger / 0xCA
path: \Segment\Cluster\BlockGroup\ReferenceFrame\ReferenceTimestamp

documentation: The timestamp of the BlockGroup pointed to by ReferenceOffset, expressed in Track Ticks; see Section 11.1. See [DivXTrickTrack].

28.15. EncryptedBlock Element

type / id: binary / 0xAF
path: \Segment\Cluster\EncryptedBlock
documentation: Similar to SimpleBlock, see Section 10.2, but the data inside the Block are Transformed (encrypt and/or signed).

28.16. MinCache Element

type / id: uinteger / 0x6DE7
path: \Segment\Tracks\TrackEntry\MinCache
documentation: The minimum number of frames a player should be able to cache during playback. If set to 0, the reference pseudo-cache system is not used.

28.17. MaxCache Element

type / id: uinteger / 0x6DF8
path: \Segment\Tracks\TrackEntry\MaxCache
documentation: The maximum cache size necessary to store referenced frames in and the current frame. 0 means no cache is needed.

28.18. TrackOffset Element

type / id: integer / 0x537F
path: \Segment\Tracks\TrackEntry\TrackOffset
documentation: A value to add to the Block's Timestamp, expressed in Matroska Ticks -- i.e., in nanoseconds; see Section 11.1. This can be used to adjust the playback offset of a track.

28.19. CodecSettings Element

type / id: utf-8 / 0x3A9697
path: \Segment\Tracks\TrackEntry\CodecSettings
documentation: A string describing the encoding setting used.

28.20. CodecInfoURL Element

type / id: string / 0x3B4040
path: \Segment\Tracks\TrackEntry\CodecInfoURL
documentation: A URL to find information about the codec used.

28.21. CodecDownloadURL Element

type / id: string / 0x26B240
path: \Segment\Tracks\TrackEntry\CodecDownloadURL
documentation: A URL to download about the codec used.

28.22. CodecDecodeAll Element

type / id: uinteger / 0xAA
path: \Segment\Tracks\TrackEntry\CodecDecodeAll
documentation: Set to 1 if the codec can decode potentially damaged data.

28.23. TrackOverlay Element

type / id: uinteger / 0x6FAB
path: \Segment\Tracks\TrackEntry\TrackOverlay
documentation: Specify that this track is an overlay track for the Track specified (in the u-integer). That means when this track has a gap on SilentTracks, the overlay track should be used instead. The order of multiple TrackOverlay matters, the first one is the one that should be used. If not found it should be the second, etc.

28.24. AspectRatioType Element

type / id: uinteger / 0x54B3
path: \Segment\Tracks\TrackEntry\Video\AspectRatioType
documentation: Specify the possible modifications to the aspect ratio.

28.25. GammaValue Element

type / id: float / 0x2FB523
path: \Segment\Tracks\TrackEntry\Video\GammaValue
documentation: Gamma Value.

28.26. FrameRate Element

type / id: float / 0x2383E3
path: \Segment\Tracks\TrackEntry\Video\FrameRate
documentation: Number of frames per second. This value is Informational only. It is intended for constant frame rate streams, and should not be used for a variable frame rate TrackEntry.

28.27. ChannelPositions Element

type / id: binary / 0x7D7B
path: \Segment\Tracks\TrackEntry\Audio\ChannelPositions
documentation: Table of horizontal angles for each successive channel.

28.28. TrickTrackUID Element

type / id: uinteger / 0xC0
path: \Segment\Tracks\TrackEntry\TrickTrackUID
documentation: The TrackUID of the Smooth FF/RW video in the paired EBML structure corresponding to this video track. See [DivXTrickTrack].

28.29. TrickTrackSegmentUID Element

type / id: binary / 0xC1
path: \Segment\Tracks\TrackEntry\TrickTrackSegmentUID
documentation: The SegmentUID of the Segment containing the track identified by TrickTrackUID. See [DivXTrickTrack].

28.30. TrickTrackFlag Element

type / id: uinteger / 0xC6
path: \Segment\Tracks\TrackEntry\TrickTrackFlag
documentation: Set to 1 if this video track is a Smooth FF/RW track. If set to 1, MasterTrackUID and MasterTrackSegUID should be present and BlockGroups for this track must contain ReferenceFrame structures. Otherwise, TrickTrackUID and TrickTrackSegUID must be present if this track has a corresponding Smooth FF/RW track. See [DivXTrickTrack].

28.31. TrickMasterTrackUID Element

type / id: uinteger / 0xC7
path: \Segment\Tracks\TrackEntry\TrickMasterTrackUID
documentation: The TrackUID of the video track in the paired EBML structure that corresponds to this Smooth FF/RW track. See [DivXTrickTrack].

28.32. TrickMasterTrackSegmentUID Element

type / id: binary / 0xC4
path: \Segment\Tracks\TrackEntry\TrickMasterTrackSegmentUID
documentation: The SegmentUID of the Segment containing the track identified by MasterTrackUID. See [DivXTrickTrack].

28.33. ContentSignature Element

type / id: binary / 0x47E3
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentSignature
documentation: A cryptographic signature of the contents.

28.34. ContentSigKeyID Element

type / id: binary / 0x47E4
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentSigKeyID
documentation: This is the ID of the private key the data was signed with.

28.35. ContentSigAlgo Element

type / id: uinteger / 0x47E5
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentSigAlgo
documentation: The algorithm used for the signature.

28.36. ContentSigHashAlgo Element

type / id: uinteger / 0x47E6
path: \Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncryption\ContentSigHashAlgo
documentation: The hash algorithm used for the signature.

28.37. CueRefCluster Element

type / id: uinteger / 0x97
path: \Segment\Cues\CuePoint\CueTrackPositions\CueReference\CueRefCluster
documentation: The Segment Position of the Cluster containing the referenced Block.

28.38. CueRefNumber Element

type / id: uinteger / 0x535F
path: \Segment\Cues\CuePoint\CueTrackPositions\CueReference\CueRefNumber
documentation: Number of the referenced Block of Track X in the specified Cluster.

28.39. CueRefCodecState Element

type / id: uinteger / 0xEB

path: \Segment\Cues\CuePoint\CueTrackPositions\CueReference\CueRefCodecState
documentation: The Segment Position of the Codec State corresponding to this referenced Element. 0 means that the data is taken from the initial Track Entry.

28.40. FileReferral Element

type / id: binary / 0x4675
path: \Segment\Attachments\AttachedFile\FileReferral
documentation: A binary value that a track/codec can refer to when the attachment is needed.

28.41. FileUsedStartTime Element

type / id: uinteger / 0x4661
path: \Segment\Attachments\AttachedFile\FileUsedStartTime
documentation: The timestamp at which this optimized font attachment comes into context, expressed in Segment Ticks which is based on TimestampScale. See [DivXWorldFonts].

28.42. FileUsedEndTime Element

type / id: uinteger / 0x4662
path: \Segment\Attachments\AttachedFile\FileUsedEndTime
documentation: The timestamp at which this optimized font attachment goes out of context, expressed in Segment Ticks which is based on TimestampScale. See [DivXWorldFonts].

28.43. TagDefaultBogus Element

type / id: uinteger / 0x44B4
path: \Segment\Tags\Tag\+SimpleTag\TagDefaultBogus
documentation: A variant of the TagDefault element with a bogus Element ID; see Section 5.1.8.1.2.4.

29. Normative References

[BCP47] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[CIE-1931] Commission Internationale de l'Eclairage, "CIE 1931 Standard Colorimetric System", 1931, <https://en.wikipedia.org/wiki/CIE_1931_color_space>.

[ISO639-2] United States Library Of Congress, "Codes for the Representation of Names of Languages", ISO 639-2:1998, 21 December 2017, <https://www.loc.gov/standards/iso639-2/php/code_list.php>.

[ISO9899] International Organization for Standardization, "Information technology -- Programming languages -- C", ISO/IEC 9899:2011, 2011, <https://www.iso.org/standard/57853.html>.

[ITU-H.273] International Telecommunication Union, "Coding-independent code points for video signal type identification", ITU H.273, 24 September 2021, <https://www.itu.int/rec/T-REC-H.273/en>.

[RFC1950] Deutsch, P. and J. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/info/rfc1950>.

[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>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8081] Lilley, C., "The "font" Top-Level Media Type", RFC 8081, DOI 10.17487/RFC8081, February 2017, <https://www.rfc-editor.org/info/rfc8081>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8794] Lhomme, S., Rice, D., and M. Bunkus, "Extensible Binary Meta Language", RFC 8794, DOI 10.17487/RFC8794, July 2020, <https://www.rfc-editor.org/info/rfc8794>.

30. Informative References

[AVIFormat]
Microsoft, "AVI RIFF File Reference", 31 May 2018, <https://docs.microsoft.com/en-us/windows/win32/directshow/avi-riff-file-reference>.

[Blowfish] Schneier, B., "The Blowfish Encryption Algorithm", 1993, <https://www.schneier.com/academic/blowfish/>.

[BZIP2] Seward, J., "bzip2", 18 July 1996, <https://sourceware.org/bzip2/>.

[DivXTrickTrack]
"DivX Trick Track Extensions", 14 December 2010, <https://web.archive.org/web/20101222001148/http://labs.divx.com/node/16601>.

[DivXWorldFonts]
"DivX World Fonts Extensions", 14 December 2010, <https://web.archive.org/web/20110214132246/http://labs.divx.com/node/16602>.

[DVD-Video]
DVD Forum, "DVD-Books: Part 3 DVD-Video Book", 1 November 1995, <http://www.dvdforum.org/>.

[FIPS.197] US National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", FIPS PUB 197, DOI 10.6028/NIST.FIPS.197, 26 November 2001, <https://csrc.nist.gov/publications/detail/fips/197/final>.

[FIPS.46-3]
US National Institute of Standards and Technology, "Data Encryption Standard (DES)", FIPS PUB 46, 25 October 1999, <https://csrc.nist.gov/publications/detail/fips/46/3/archive/1999-10-25>.

[FourCC-RGB]
Silicon.dk ApS, "RGB Pixel Format FourCCs", <https://web.archive.org/web/20160609214806/https://www.fourcc.org/rgb.php>.

[FourCC-YUV]
Silicon.dk ApS, "YUV Pixel Format FourCCs", <https://web.archive.org/web/20160609214806/https://www.fourcc.org/yuv.php>.

[JPEG] International Telegraph and Telephone Consultative Committee, "Digital Compression and Coding of Continuous-Tone Still Images", ITU T.81, September 1992, <https://www.w3.org/Graphics/JPEG/itu-t81.pdf>.

[LZO] Tarreau, W., Rodgman, R., and M. Oberhumer, "Lempel-Ziv-Oberhumer compression", 30 October 2018, <https://www.kernel.org/doc/Documentation/lzo.txt>.

[MatroskaCodec]
Lhomme, S., Bunkus, M., and D. Rice, "Media Container Codec Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-codec-10, 12 April 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-codec-10>.

[MatroskaTags]
Lhomme, S., Bunkus, M., and D. Rice, "Matroska Media Container Tag Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-tags-10, 12 April 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-tags-10>.

[MCF] "Media Container Format", 17 July 2002, <http://mukoli.free.fr/mcf/>.

[MSRGB] Microsoft, "WMF Compression Enumeration", <https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-wmf/4e588f70-bd92-4a6f-b77f-35d0feaf7a57>.

[MSYUV16] Microsoft, "10-bit and 16-bit YUV Video Formats", <https://learn.microsoft.com/en-us/windows/win32/medfound/10-bit-and-16-bit-yuv-video-formats>.

[MSYUV8] Microsoft, "Recommended 8-Bit YUV Formats for Video Rendering", <https://learn.microsoft.com/en-us/windows/win32/medfound/recommended-8-bit-yuv-formats-for-video-rendering>.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, DOI 10.17487/RFC0959, October 1985, <https://www.rfc-editor.org/info/rfc959>.

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

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

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[SMB-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol", 1 October 2020, <https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-CIFS/%5bMS-CIFS%5d.pdf>.

[SP.800-38A] US National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: Methods and Techniques", DOI 10.6028/NIST.SP.800-38A, 1 December 2001, <https://csrc.nist.gov/publications/detail/fips/197/final>.

[SP.800-67] US National Institute of Standards and Technology, "Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher", DOI 10.6028/10.6028/NIST.SP.800-67r2, 1 November 2017, <https://csrc.nist.gov/publications/detail/sp/800-67/rev-2/final>.

[Twofish] Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., and N. Ferguson, "Twofish: A 128-Bit Block Cipher", 15 June 1998, <https://www.schneier.com/academic/twofish/>.

[WebM-Enc] Galligan, F., "WebM Encryption", 19 September 2016, <https://www.webmproject.org/docs/webm-encryption/>.

[WebVTT] Pieters, S., Pfeiffer, S., Ed., Jaegenstedt, P., and I. Hickson, "WebVTT Cue Identifier", 4 April 2019, <https://www.w3.org/TR/webvtt1/#webvtt-cue-identifier>.

Authors' Addresses

Steve Lhomme
Email: slhomme@matroska.org

Moritz Bunkus
Email: moritz@bunkus.org

Dave Rice
Email: dave@dericed.com

cellar
Internet-Draft
Intended status: Standards Track
Expires: 6 November 2024

S. Lhomme

M. Bunkus

D. Rice

5 May 2024

Matroska Media Container Tag Specifications
draft-ietf-cellar-tags-13

Abstract

This document defines the Matroska tags, namely the tag names and their respective semantic meaning.

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 6 November 2024.

Table of Contents

1. Introduction

Matroska is a multimedia container format defined in [Matroska]. It can store timestamped multimedia data but also chapters and tags. The Tag Elements add important metadata to identify and classify the information found in a Matroska Segment. It can tag a whole Segment, separate Track Elements, individual Chapter Elements or Attachment Elements.

While the Matroska tagging framework allows anyone to create their own custom tags, it's important to have a common set of values for interoperability. This document intends to define a set of common tag names used in Matroska.

2. Status of This Document

This document is a work-in-progress specification defining the Matroska file format as part of the IETF Cellar working group (https://datatracker.ietf.org/wg/cellar/charter/). It uses basic elements and concept already defined in the Matroska specifications defined by this workgroup [Matroska].

3. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Tagging

When a Tag is nested within another Tag, the nested Tag becomes an attribute of the base tag. For instance, if you wanted to store the dates that a singer used certain addresses for, that singer being the lead singer for a track that included multiple bands simultaneously, then your tag tree would look something like this:

* Targets
  - TrackUID
* BAND
  - LEADPERFORMER
    - o ADDRESS
      + DATE
      + DATEEND
    - o ADDRESS
      + DATE

In this way, it becomes possible to store any Tag as attributes of another tag.

Multiple items SHOULD never be stored as a list in a single TagString. If there is more than one tag of a certain type to be stored, then more than one SimpleTag SHOULD be used.

4.1. Why Official Tags Matter

There is a debate between people who think all tags SHOULD be free and those who think all tags SHOULD be strict. If you look at this page you will realize we are in between.

Advanced-users application might let you put any tag in your file. But for the rest of the applications, they usually give you a basic list of tags you can use. Both have their needs. But it's usually a bad idea to use custom/exotic tags because you will probably be the only person to use this information even though everyone else could benefit from it. So hopefully, when someone wants to put information in one's file, they will find an official one that fit them and hopefully use it ! If it's not in the list, this person can contact us any time for addition of such a missing tag. But it doesn't mean it will be accepted... Matroska files are not meant the become a whole database of people who made costumes for a film. A website would be better for that... It's hard to define what SHOULD be in and what doesn't make sense in a file; thus, we'll treat each request carefully.

We also need an official list simply for developers to be able to display relevant information in their own design (if they choose to support a list of meta-information they SHOULD know which tag has the wanted meaning so that other apps could understand the same meaning).

4.2. Tag Formatting

* The TagName SHOULD consists of capital letters, numbers and the underscore character '_'.

* The TagName SHOULD NOT contain any space.

* TagNames starting with the underscore character '_' are not official tags; see Section 4.1.

* The fields with dates SHOULD have the following format: YYYY-MM-DD hh:mm:ss.mss YYYY = Year, MM = Month, DD = Days, HH = Hours, mm = Minutes, ss = Seconds, mss = Milliseconds. To store less accuracy, you remove items starting from the right. To store only the year, you would use, "2004". To store a specific day such as May 1st, 2003, you would use "2003-05-01".

* Fields that require a Float SHOULD use the "." mark instead of the "," mark. To display it differently for another local, applications SHOULD support auto replacement on display. Also, a thousandths separator SHOULD NOT be used.

* For currency amounts, there SHOULD only be a numeric value in the Tag. Only numbers, no letters or symbols other than ".". For instance, you would store "15.59" instead of "$15.59USD".

4.3. Target Types

The TargetType element allows tagging of different parts that are inside or outside a given file. For example, in an audio file with one song you could have information about the album it comes from and even the CD set even if it’s not found in the file.

For application to know the kind of information (like TITLE) relates to a certain level (CD title or track title), we also need a set of official TargetType names. For now audio and video will have different values and names. That also means the same tag name can have different meanings depending on where it is (otherwise, we would end up with 15 TITLE_ tags).

TargetTypeValue	Audio strings	Video strings	Comment
70	COLLECTION	COLLECTION	the high hierarchy consisting of many different lower items
60	EDITION / ISSUE / VOLUME / OPUS	SEASON / SEQUEL / VOLUME	a list of lower levels grouped together
50	ALBUM / OPERA / CONCERT	MOVIE / EPISODE / CONCERT	the most common grouping level of music and video (e.g., an episode for TV series)
40	PART / SESSION	PART / SESSION	when an album or episode has different logical parts
30	TRACK / SONG	CHAPTER	the common parts of an album or a movie
20	SUBTRACK / PART / MOVEMENT	SCENE	corresponds to parts of a track for audio (like a movement)
10	-	SHOT	the lowest hierarchy found in music or movies

Table 1: TargetTypeValue Values Semantic Description

An upper level value tag applies to the lower level. This means that if a CD has the same artist for all tracks, you just need to set the ARTIST tag at level 50 (ALBUM) and not to each TRACK (but you can). That also means that, if some parts of the CD have no known ARTIST, the value MUST be set to nothing (a void string "").

When a level doesn't exist it MUST NOT be specified in the files, so that the TOTAL_PARTS and PART_NUMBER elements match the same levels.

Here is an example of how these organizational tags work: If you set 10 TOTAL_PARTS to the ALBUM level (40) it means the album contains 10 lower parts. The lower part in question is the first lower level that is specified in the file. So, if it's TRACK (30), then that means it contains 10 tracks. If it's MOVEMENT (20), that means it's 10 movements, etc.

5. Official tags

The following is a complete list of the supported Matroska Tags. While it is possible to use Tag names that are not listed below, this is not recommended as compatibility will be compromised. If you find that there is a Tag missing that you would like to use, then please contact the persons mentioned in the IANA Mastroska Tags Registry for its inclusion; see Section 7.1.

5.1. Nesting Information

Nesting Information tags are intended to contain other tags.

Tag Name	Type	Description
ORIGINAL	nested	A special tag that is meant to have other tags inside (using nested tags) to describe the original work of art that this item is based on. All tags in this list can be used "under" the ORIGINAL tag like LYRICIST, PERFORMER, etc.
SAMPLE	nested	A tag that contains other tags to describe a sample used in the targeted item taken from another work of art. All tags in this list can be used "under" the SAMPLE tag like TITLE, ARTIST, DATE_RELEASED, etc.
COUNTRY	UTF-8	The name of the country that is meant to have other tags inside (using nested tags) to country specific information about the item, in the Matroska countries form, i.e. [RFC5646] two-letter region subtags, without the UK exception. All tags in this list can be used "under" the COUNTRY_SPECIFIC tag like LABEL, PUBLISH_RATING, etc.

Table 2: Nesting Information tags

5.2. Organization Information

Tag Name	Type	Description
TOTAL_PARTS	UTF-8	Total number of parts defined at the first lower level. (e.g., if TargetType is ALBUM, the total number of tracks of an audio CD).
PART_NUMBER	UTF-8	Number of the current part of the current level. (e.g., if TargetType is TRACK, the track number of an audio CD).
PART_OFFSET	UTF-8	A number to add to PART_NUMBER, when the parts at that level don’t start at 1 (e.g., if TargetType is TRACK, the track number of the second audio CD).

Table 3: Organization Information tags

5.3. Titles

Tag Name	Type	Description
TITLE	UTF-8	The title of this item. For example, for music you might label this "Canon in D", or for video's audio track you might use "English 5.1" This is akin to the "TIT2" tag in [ID3v2].
SUBTITLE	UTF-8	Sub Title of the entity.

Table 4: Titles tags

5.4. Nested Information

Nested Information includes tags contained in other tags.

Tag Name	Type	Description
URL	UTF-8	URL corresponding to the tag it's included in.
SORT_WITH	UTF-8	A child element to indicate what alternative value the parent tag can have to be sorted -- for example, "Pet Shop Boys" instead of "The Pet Shop Boys". Or "Marley Bob" and "Marley Ziggy" (no comma needed).
INSTRUMENTS	UTF-8	The instruments that are being used/ played, separated by a comma. It SHOULD be a child of the following tags: ARTIST, LEAD_PERFORMER, or ACCOMPANIMENT.
EMAIL	UTF-8	Email corresponding to the tag it's included in.
ADDRESS	UTF-8	The physical address of the entity. The address SHOULD include a country code. It can be useful for a recording label.
FAX	UTF-8	The fax number corresponding to the tag it's included in. It can be useful for a recording label.
PHONE	UTF-8	The phone number corresponding to the tag it's included in. It can be useful for a recording label.

Table 5: Nested Information tags

5.5. Entities

Tag Name	Type	Description
ARTIST	UTF-8	A person or band/collective generally considered responsible for the work. This is akin to the "TPE1" tag in [ID3v2].

LEAD_PERFORMER	UTF-8	Lead Performer/Soloist(s). This can sometimes be the same as ARTIST.
ACCOMPANIMENT	UTF-8	Band/orchestra/accompaniment/ musician. This is akin to the "TPE2" tag in [ID3v2].
COMPOSER	UTF-8	The name of the composer of this item. This is akin to the "TCOM" tag in [ID3v2].
ARRANGER	UTF-8	The person who arranged the piece (e.g., Ravel).
LYRICS	UTF-8	The lyrics corresponding to a song (in case audio synchronization is not known or as a doublon to a subtitle track). Editing this value, when subtitles are found, SHOULD also result in editing the subtitle track for more consistency.
LYRICIST	UTF-8	The person who wrote the lyrics for a musical item. This is akin to the "TEXT" tag in [ID3v2].
CONDUCTOR	UTF-8	Conductor/performer refinement. This is akin to the "TPE3" tag in [ID3v2].
DIRECTOR	UTF-8	This is akin to the "IART" tag [RIFF.tags].
ASSISTANT_DIRECTOR	UTF-8	The name of the assistant director.
DIRECTOR_OF_PHOTOGRAPHY	UTF-8	The name of the director of photography, also known as cinematographer. This is akin to the "ICNM" tag in [RIFF.tags].
SOUND_ENGINEER	UTF-8	The name of the sound

		engineer or sound recordist.
ART_DIRECTOR	UTF-8	The person who oversees the artists and craftspeople who build the sets.
PRODUCTION_DESIGNER	UTF-8	Artist responsible for designing the overall visual appearance of a movie.
CHOREGRAPHER	UTF-8	The name of the choregrapher
COSTUME_DESIGNER	UTF-8	The name of the costume designer
ACTOR	UTF-8	An actor or actress playing a role in this movie. This is the person's real name, not the character's name the person is playing.
CHARACTER	UTF-8	The name of the character an actor or actress plays in this movie. This SHOULD be a sub-tag of an ACTOR tag in order to not cause ambiguities.
WRITTEN_BY	UTF-8	The author of the story or script (used for movies and TV shows).
SCREENPLAY_BY	UTF-8	The author of the screenplay or scenario (used for movies and TV shows).
EDITED_BY	UTF-8	This is akin to the "IEDT" tag in [RIFF.tags].
PRODUCER	UTF-8	Produced by. This is akin to the "IPRO" tag in [RIFF.tags].
COPRODUCER	UTF-8	The name of a co-producer.
EXECUTIVE_PRODUCER	UTF-8	The name of an executive producer.

DISTRIBUTED_BY	UTF-8	This is akin to the "IDST" tag in [RIFF.tags].
MASTERED_BY	UTF-8	The engineer who mastered the content for a physical medium or for digital distribution.
ENCODED_BY	UTF-8	This is akin to the "TENC" tag in [ID3v2].
MIXED_BY	UTF-8	DJ mix by the artist specified
REMIXED_BY	UTF-8	Interpreted, remixed, or otherwise modified by. This is akin to the "TPE4" tag in [ID3v2].
PRODUCTION_STUDIO	UTF-8	This is akin to the "ISTD" tag in [RIFF.tags].
THANKS_TO	UTF-8	A very general tag for everyone else that wants to be listed.
PUBLISHER	UTF-8	This is akin to the "TPUB" tag in [ID3v2].
LABEL	UTF-8	The record label or imprint on the disc.

Table 6: Entities tags

5.6. Search and Classification

Tag Name	Type	Description
GENRE	UTF-8	The main genre (classical, ambient-house, synthpop, sci-fi, drama, etc.). The format follows the infamous "TCON" tag in [ID3v2].
MOOD	UTF-8	Intended to reflect the mood of the item with a few keywords (e.g., "Romantic", "Sad" or

		"Uplifting"). The format follows that of the "TMOO" tag in [ID3v2].
ORIGINAL_MEDIA_TYPE	UTF-8	Describes the original type of the media, such as, "DVD", "CD", "computer image," "drawing," "lithograph," and so forth. This is akin to the "TMED" tag in [ID3v2].
CONTENT_TYPE	UTF-8	The type of the item (e.g., Documentary, Feature Film, Cartoon, Music Video, Music, Sound FX).
SUBJECT	UTF-8	Describes the topic of the file, such as "Aerial view of Seattle."
DESCRIPTION	UTF-8	A short description of the content, such as "Two birds flying."
KEYWORDS	UTF-8	Keywords to the item separated by a comma, used for searching.
SUMMARY	UTF-8	A plot outline or a summary of the story.
SYNOPSIS	UTF-8	A description of the story line of the item.
INITIAL_KEY	UTF-8	The initial key that a musical track starts in. The format is identical to "TKEY" tag in [ID3v2].
PERIOD	UTF-8	Describes the period that the piece is from or about. For example, "Renaissance".
LAW_RATING	UTF-8	Depending on the COUNTRY it’s the format of the rating of a movie (P, R, X in the USA, an age in other countries or a URI defining a logo).

Table 7: Search and Classification tags

5.7. Temporal Information

Tag Name	Type	Description
DATE_RELEASED	UTF-8	The time that the item was originally released. This is akin to the "TDRL" tag in [ID3v2].
DATE_RECORDED	UTF-8	The time that the recording began. This is akin to the "TDRC" tag in [ID3v2].
DATE_ENCODED	UTF-8	The time that the encoding of this item was completed began. This is akin to the "TDEN" tag in [ID3v2].
DATE_TAGGED	UTF-8	The time that the tags were done for this item. This is akin to the "TDTG" tag in [ID3v2].
DATE_DIGITIZED	UTF-8	The time that the item was transferred to a digital medium. This is akin to the "IDIT" tag in [RIFF.tags].
DATE_WRITTEN	UTF-8	The time that the writing of the music/script began.
DATE_PURCHASED	UTF-8	Information on when the file was purchased; see also Section 5.12 on purchase tags.

Table 8: Temporal Information tags

5.8. Spatial Information

Tag Name	Type	Description
RECORDING_LOCATION	UTF-8	The location where the item was recorded, in the Matroska countries form, i.e. [RFC5646] two-letter region subtags, without the UK exception. This code is followed by a comma, then more detailed information such as state/province, another comma,

		and then city. For example, "US, Texas, Austin". This will allow for easy sorting. It is okay to only store the country, or the country and the state/province. More detailed information can be added after the city through the use of additional commas. In cases where the province/state is unknown, but you want to store the city, simply leave a space between the two commas. For example, "US, , Austin".
COMPOSITION_LOCATION	UTF-8	Location that the item was originally designed/written, in the Matroska countries form, i.e. [RFC5646] two-letter region subtags, without the UK exception. This code is followed by a comma, then more detailed information such as state/ province, another comma, and then city. For example, "US, Texas, Austin". This will allow for easy sorting. It is okay to only store the country, or the country and the state/province. More detailed information can be added after the city through the use of additional commas. In cases where the province/state is unknown, but you want to store the city, simply leave a space between the two commas. For example, "US, , Austin".
COMPOSER_NATIONALITY	UTF-8	Nationality of the main composer of the item, mostly for classical music, in the Matroska countries form, i.e. [RFC5646] two-letter region subtags, without the UK exception.

Table 9: Spatial Information tags

5.9. Personal

Tag Name	Type	Description
COMMENT	UTF-8	Any comment related to the content.
PLAY_COUNTER	UTF-8	The number of time the item has been played.
RATING	UTF-8	A numeric value defining how much a person likes the song/movie. The number is between 0 and 5 with decimal values possible (e.g., 2.7), 5(.0) being the highest possible rating. Other rating systems with different ranges will have to be scaled.

Table 10: Personal tags

5.10. Technical Information

Tag Name	Type	Description
ENCODER	UTF-8	The software or hardware used to encode this item. ("LAME" or "XviD")
ENCODER_SETTINGS	UTF-8	A list of the settings used for encoding this item. No specific format.
BPS	UTF-8	The average bits per second of the specified item. This is only the data in the Blocks, and excludes headers and any container overhead.
FPS	UTF-8	The average frames per second of the specified item. This is typically the average number of Blocks per second. In the event that lacing is used, each laced chunk is to be counted as a separate frame.
BPM	UTF-8	Average number of beats per minute

		in the complete target (e.g., a chapter). Usually a decimal number.
MEASURE	UTF-8	In music, a measure is a unit of time in Western music like "4/4". It represents a regular grouping of beats, a meter, as indicated in musical notation by the time signature. The majority of the contemporary rock and pop music you hear on the radio these days is written in the 4/4 time signature.
TUNING	UTF-8	It is saved as a frequency in hertz to allow near-perfect tuning of instruments to the same tone as the musical piece (e.g., "441.34" in Hertz). The default value is 440.0 Hz.
REPLAYGAIN_GAIN	binary	The gain to apply to reach 89dB SPL on playback. This is based on the [ReplayGain] standard. Note that ReplayGain information can be found at all TargetType levels (track, album, etc).
REPLAYGAIN_PEAK	binary	The maximum absolute peak value of the item. This is based on the [ReplayGain] standard.

Table 11: Technical Information tags

5.11. Identifiers

Tag Name	Type	Description
ISRC	UTF-8	The International Standard Recording Code [ISRC], excluding the "ISRC" prefix and including hyphens.
MCDI	binary	This is a binary dump of the TOC of the CDROM that this item was taken from. This holds the same

		information as the "MCDI" in [ID3v2].
ISBN	UTF-8	International Standard Book Number [ISBN].
BARCODE	UTF-8	European Article Numbering EAN-13 barcode defined in [GS1] General Specifications.
CATALOG_NUMBER	UTF-8	A label-specific string used to identify the release -- for example, TIC 01.
LABEL_CODE	UTF-8	A 4-digit or 5-digit number to identify the record label, typically printed as (LC) xxxx or (LC) 0xxxx on CDs medias or covers (only the number is stored).
LCCN	UTF-8	Library of Congress Control Number [LCCN].
IMDB	UTF-8	Internet Movie Database [IMDb] identifier. "tt" followed by at least 7 digits for Movies, TV Shows, and Episodes.
TMDB	UTF-8	The Movie DB "movie_id" or "tv_id" identifier for movies/TV shows [MovieDB]. The variable length digits string MUST be prefixed with either "movie/" or "tv/".
TVDB	UTF-8	The TV Database "Series ID" or "Episode ID" identifier for TV shows [TheTVDB]. Variable length all-digits string identifying a TV Show.
TVDB2	UTF-8	The TV Database [TheTVDB] tag which can include movies. The variable length digits string representing a "Series ID", "Episode ID" or "Movie ID" identifier MUST be prefixed with "series/", "episodes/", or "movies/", respectively.

Table 12: Identifiers tags

5.12. Commercial

Tag Name	Type	Description
PURCHASE_ITEM	UTF-8	URL to purchase this file. This is akin to the "WPAY" tag in [ID3v2].
PURCHASE_INFO	UTF-8	Information on where to purchase this album. This is akin to the "WCOM" tag in [ID3v2].
PURCHASE_OWNER	UTF-8	Information on the person who purchased the file. This is akin to the "TOWN" tag in [ID3v2].
PURCHASE_PRICE	UTF-8	The amount paid for entity. There SHOULD only be a numeric value in here. Only numbers, no letters or symbols other than ".". For instance, you would store "15.59" instead of "$15.59USD".
PURCHASE_CURRENCY	UTF-8	The currency type used to pay for the entity. Use [ISO4217] for the 3 letter alphabetic code.

Table 13: Commercial tags

5.13. Legal

Tag Name	Type	Description
COPYRIGHT	UTF-8	The copyright information as per the copyright holder. This is akin to the "TCOP" tag in [ID3v2].
PRODUCTION_COPYRIGHT	UTF-8	The copyright information as per the production copyright holder. This is akin to the "TPRO" tag in [ID3v2].

LICENSE	UTF-8	The license applied to the content (like Creative Commons variants).
TERMS_OF_USE	UTF-8	The terms of use for this item. This is akin to the "USER" tag in [ID3v2].

Table 14: Legal tags

5.14. Notes

In the Target list, a logical OR is applied on all tracks, a logical OR is applied on all chapters. Then a logical AND is applied between the Tracks list and the Chapters list to know if an element belongs to this Target.

6. Security Considerations

Tag values can be either strings or binary blobs. This document inherits security considerations from the EBML [RFC8794] and Matroska [Matroska] documents.

7. IANA Considerations

7.1. Matroska Tags Names Registry

IANA has created a new registry called the "Matroska Tag Names" registry.

To register a new Tag Name in this registry, one needs a Name, a Type, a Change Controller (IETF or email of registrant), and an optional Reference to a document describing the Element ID.

The Name corresponds to the value stored in the TagName Element. The Name SHOULD always be written in all capital letters and contain no space as defined in Section 4.2,

The Type corresponds to which element will be stored the tag value. There can be 3 values for the Type:

* UTF-8: the value of the Tag is stored in TagString,
* binary: the value of the Tag is stored in TagBinary,
* nested: the tag doesn’t contain a value, only nested tags inside.

Matroska Tag Names Values found in this document are assigned as initial values as follows:

Tag Name	Tag Type	Reference
ORIGINAL	nested	Described in this Section 5.1
SAMPLE	nested	Described in this Section 5.1
COUNTRY	UTF-8	Described in this Section 5.1
TOTAL_PARTS	UTF-8	Described in this Section 5.2
PART_NUMBER	UTF-8	Described in this Section 5.2
PART_OFFSET	UTF-8	Described in this Section 5.2
TITLE	UTF-8	Described in this Section 5.3
SUBTITLE	UTF-8	Described in this Section 5.3
URL	UTF-8	Described in this Section 5.4
SORT_WITH	UTF-8	Described in this Section 5.4
INSTRUMENTS	UTF-8	Described in this Section 5.4
EMAIL	UTF-8	Described in this Section 5.4
ADDRESS	UTF-8	Described in this Section 5.4
FAX	UTF-8	Described in this Section 5.4
PHONE	UTF-8	Described in this Section 5.4

ARTIST	UTF-8	Described in this Section 5.5
LEAD_PERFORMER	UTF-8	Described in this Section 5.5
ACCOMPANIMENT	UTF-8	Described in this Section 5.5
COMPOSER	UTF-8	Described in this Section 5.5
ARRANGER	UTF-8	Described in this Section 5.5
LYRICS	UTF-8	Described in this Section 5.5
LYRICIST	UTF-8	Described in this Section 5.5
CONDUCTOR	UTF-8	Described in this Section 5.5
DIRECTOR	UTF-8	Described in this Section 5.5
ASSISTANT_DIRECTOR	UTF-8	Described in this Section 5.5
DIRECTOR_OF_PHOTOGRAPHY	UTF-8	Described in this Section 5.5
SOUND_ENGINEER	UTF-8	Described in this Section 5.5
ART_DIRECTOR	UTF-8	Described in this Section 5.5
PRODUCTION_DESIGNER	UTF-8	Described in this Section 5.5
CHOREGRAPHER	UTF-8	Described in this Section 5.5
COSTUME_DESIGNER	UTF-8	Described in this Section 5.5

ACTOR	UTF-8	Described in this Section 5.5
CHARACTER	UTF-8	Described in this Section 5.5
WRITTEN_BY	UTF-8	Described in this Section 5.5
SCREENPLAY_BY	UTF-8	Described in this Section 5.5
EDITED_BY	UTF-8	Described in this Section 5.5
PRODUCER	UTF-8	Described in this Section 5.5
COPRODUCER	UTF-8	Described in this Section 5.5
EXECUTIVE_PRODUCER	UTF-8	Described in this Section 5.5
DISTRIBUTED_BY	UTF-8	Described in this Section 5.5
MASTERED_BY	UTF-8	Described in this Section 5.5
ENCODED_BY	UTF-8	Described in this Section 5.5
MIXED_BY	UTF-8	Described in this Section 5.5
REMIXED_BY	UTF-8	Described in this Section 5.5
PRODUCTION_STUDIO	UTF-8	Described in this Section 5.5
THANKS_TO	UTF-8	Described in this Section 5.5
PUBLISHER	UTF-8	Described in this Section 5.5

LABEL	UTF-8	Described in this Section 5.5
GENRE	UTF-8	Described in this Section 5.6
MOOD	UTF-8	Described in this Section 5.6
ORIGINAL_MEDIA_TYPE	UTF-8	Described in this Section 5.6
CONTENT_TYPE	UTF-8	Described in this Section 5.6
SUBJECT	UTF-8	Described in this Section 5.6
DESCRIPTION	UTF-8	Described in this Section 5.6
KEYWORDS	UTF-8	Described in this Section 5.6
SUMMARY	UTF-8	Described in this Section 5.6
SYNOPSIS	UTF-8	Described in this Section 5.6
INITIAL_KEY	UTF-8	Described in this Section 5.6
PERIOD	UTF-8	Described in this Section 5.6
LAW_RATING	UTF-8	Described in this Section 5.6
DATE_RELEASED	UTF-8	Described in this Section 5.7
DATE_RECORDED	UTF-8	Described in this Section 5.7
DATE_ENCODED	UTF-8	Described in this Section 5.7

DATE_TAGGED	UTF-8	Described in this Section 5.7
DATE_DIGITIZED	UTF-8	Described in this Section 5.7
DATE_WRITTEN	UTF-8	Described in this Section 5.7
DATE_PURCHASED	UTF-8	Described in this Section 5.7
RECORDING_LOCATION	UTF-8	Described in this Section 5.8
COMPOSITION_LOCATION	UTF-8	Described in this Section 5.8
COMPOSER_NATIONALITY	UTF-8	Described in this Section 5.8
COMMENT	UTF-8	Described in this Section 5.9
PLAY_COUNTER	UTF-8	Described in this Section 5.9
RATING	UTF-8	Described in this Section 5.9
ENCODER	UTF-8	Described in this Section 5.10
ENCODER_SETTINGS	UTF-8	Described in this Section 5.10
BPS	UTF-8	Described in this Section 5.10
FPS	UTF-8	Described in this Section 5.10
BPM	UTF-8	Described in this Section 5.10
MEASURE	UTF-8	Described in this Section 5.10

TUNING	UTF-8	Described in this Section 5.10
REPLAYGAIN_GAIN	binary	Described in this Section 5.10
REPLAYGAIN_PEAK	binary	Described in this Section 5.10
ISRC	UTF-8	Described in this Section 5.11
MCDI	binary	Described in this Section 5.11
ISBN	UTF-8	Described in this Section 5.11
BARCODE	UTF-8	Described in this Section 5.11
CATALOG_NUMBER	UTF-8	Described in this Section 5.11
LABEL_CODE	UTF-8	Described in this Section 5.11
LCCN	UTF-8	Described in this Section 5.11
IMDB	UTF-8	Described in this Section 5.11
TMDB	UTF-8	Described in this Section 5.11
TVDB	UTF-8	Described in this Section 5.11
TVDB2	UTF-8	Described in this Section 5.11
PURCHASE_ITEM	UTF-8	Described in this Section 5.12
PURCHASE_INFO	UTF-8	Described in this Section 5.12

PURCHASE_OWNER	UTF-8	Described in this Section 5.12
PURCHASE_PRICE	UTF-8	Described in this Section 5.12
PURCHASE_CURRENCY	UTF-8	Described in this Section 5.12
COPYRIGHT	UTF-8	Described in this Section 5.13
PRODUCTION_COPYRIGHT	UTF-8	Described in this Section 5.13
LICENSE	UTF-8	Described in this Section 5.13
TERMS_OF_USE	UTF-8	Described in this Section 5.13

Table 15: Initial Contents of "Matroska Tag Names" Registry

8. References

8.1. Normative References

[GS1] "GS1 General Specifications", GS1 20.0, January 2020, <https://www.gs1.org/standards/barcodes-epcrfid-id-keys/gs1-general-specifications>.

[ID3v2] Nilsson, M., Mahoney, D., Ed., and J. Sundstrom, Ed., "ID3 tag version 2.3.0", 3 February 1999, <https://id3.org/id3v2.3.0>.

[IMDb] Internet Movie Database, "IMDb API Documentation", <https://imdb-api.com/api>.

[ISBN] International ISBN Agency, "ISBN Users' Manual", December 2017, <https://www.isbn-international.org/content/isbn-users-manual>.

[ISO4217] International Organization for Standardization, "ISO 4217 Currency codes", ISO 4217:2015, August 2015, <https://www.iso.org/iso-4217-currency-codes.html>.

[ISRC] IFPI Secretariat, "International Standard Recording Code (ISRC) Handbook", IFPI 3rd Edition, 2009, <https://www.ifpi.org/wp-content/uploads/2020/08/ISRC_Handbook.pdf>.

[LCCN] United States Library Of Congress, "Library Of Congress Control Number", October 1999, <https://www.loc.gov/marc/lccn.html>.

[Matroska] Lhomme, S., Bunkus, M., and D. Rice, "Media Container Specifications", Work in Progress, Internet-Draft, draft-ietf-cellar-matroska-20, 8 October 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-cellar-matroska-20>.

[MovieDB] The Movie Database, "The Movie Database API", <https://developers.themoviedb.org/3/movies/get-movie-details>.

[ReplayGain] Robinson, D., "ReplayGain 1.0 specification", 10 July 2001, <http://wiki.hydrogenaud.io/index.php?title=Replay_Gain_specification>.

[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>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[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>.

[RFC8794] Lhomme, S., Rice, D., and M. Bunkus, "Extensible Binary Meta Language", RFC 8794, DOI 10.17487/RFC8794, July 2020, <https://www.rfc-editor.org/info/rfc8794>.

[TheTVDB] The TVDB, "API documentation", <https://www.thetvdb.com/api-information>.

8.2. Informative References

[RIFF.tags]
Exiftool, "RIFF Tags",
<https://exiftool.org/TagNames/RIFF.html>.

Authors' Addresses

Steve Lhomme
Email: slhomme@matroska.org

Moritz Bunkus
Email: moritz@bunkus.org

Dave Rice
Email: dave@dericed.com