UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Lecture notes for Multimedia Data Processing

Matroska

Last updated on: 28/05/2021

Container formats

- Container formats were born from the need to combine multiple data types (audio, video, subtitles and other metadata) in a single object.
- A container format is a file format whose specifications allow you to store multiple streams of data in a non-contiguous way regardless of how they were encoded.

- A program capable of reading a container format may not be able to decode the audio or video streams contained in it.

- The most famous examples of container formats are the AVI format (Audio Video Interleaved, extension .avi), the Matroska format (extension .mkv) and the MPEG4 part 14 format (associated with .mp4 and .m4a files, usually the extension .m4a identifies files containing only audio streams).

Extensible Binary Meta Language - EBML

- The Matroska format uses the EBML format to store data within the binary file.
- EBML is a byte-aligned binary markup format (in EBML terminology called octet in the meaning of an eight-bit group).
- Similarly to XML or JSON, syntax and semantics are completely separate so EBML can be used to represent arbitrary data within a file.
- To be compliant with the EBML format, a file must have only one header, specified in the format, and can subsequently contain any other type of element.

- EBML supports different data types of varying sizes: integers with and without sign, floating point numbers, strings, dates and not interpreted binary data.
- EBML also allows you to organize its elements hierarchically, some of them (called masters) may contain other EBML elements.

Supported data types

- The basic data types supported by EBML are:
  - *Signed Integer*: Big-endian, any size from 1 to 8 octets.
  - *Unsigned Integer*: Big-endian, any size from 1 to 8 octets.
  - *Float:* Big-endian, defined for 4 and 8 octets (32, 64 bit).
  - *String*: printable ASCII characters (0x20-0x7E)
  - *UTF-8:* sequence of Unicode characters in UTF-8 format
  - *Date*: signed 8 octets integer in nanoseconds with 0 indicating the precise beginning of the millennium (2001-01-01T00:00:00,000000000 UTC).
  - *Master element*: contains other EBML sub-elements of the next lower level.
  - *Binary*: not interpreted by the parser.

- String e UTF-8 can have some bytes set to 0 in the end (zero-padding).

EBML element

- The structure of each EBML element is composed of the following fields:

  `<ID> <size> <data>`

- Each element has an identifier, a size and some data:
  - The `ID` field is an identifier of the EBML element *type* (not of the data type contained in the `data` field).
  - The `size` field indicates the size of the data contained in the data field.
  - The `data` field contains the actual data.

- Both the `ID` field and the `size` field are integers and are stored with a variable number of bytes (*variable size integer* or VINT).
- Their length in bytes can be determined by checking the contents of the first byte (the first byte encountered in reading from the file).

EBML ID

- In EBML format an ID can be represented using from 1 to 4 bytes.
- The size in bytes can be determined using the number of zeros + 1 starting from the most significant bit:

Width	Size	Representation			
1	2^7	1xxx.xxxx			
2	2^14	01xx.xxxx	xxxx.xxxx		
3	2^21	001x.xxxx	xxxx.xxxx	xxxx.xxxx	
4	2^28	0001.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx

- Since it is mandatory for IDs to represent values with as few bytes as possible, fewer values are available for each class:

Class	Base	Width	Number of IDs		
A	0x8x	1	2^7 - 2	=	126
B	0x4x	2	2^14 - 2^7	=	16 256
C	0x2x	3	2^21 - 2^14	=	2 080 768
D	0x1x	4	2^28 - 2^21	=	266 338 304

- for each class of IDs the combinations with all the bits indicated with `x` at 1 or at 0 is reserved.

EBML size

- In the EBML format, a size can be represented using from 1 to 8 bytes, although a maximum number of bytes less than 8 can be indicated in the header.
- The size in bytes can be determined using the same technique used for IDs:

Q

Q

bits, big-endian								
1xxx.xxxx								- 0 to $2^{7}-2$
01xx.xxxx	xxxx.xxxx							- 0 to $2^{14}-2$
001x.xxxx	xxxx.xxxx	xxxx.xxxx						- 0 to $2^{21}-2$
0001.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx					- 0 to $2^{28}-2$
0000.1xxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx				- 0 to $2^{35}-2$
0000.01xx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx			- 0 to $2^{42}-2$
0000.001x	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx		- 0 to $2^{49}-2$
0000.0001	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	xxxx.xxxx	- 0 to $2^{56}-2$

- In all cases the representable values range from 0 to $2^{nbit} - 2$ because the combination with all the bits indicated by x to 1 is used to indicate an unknown size.

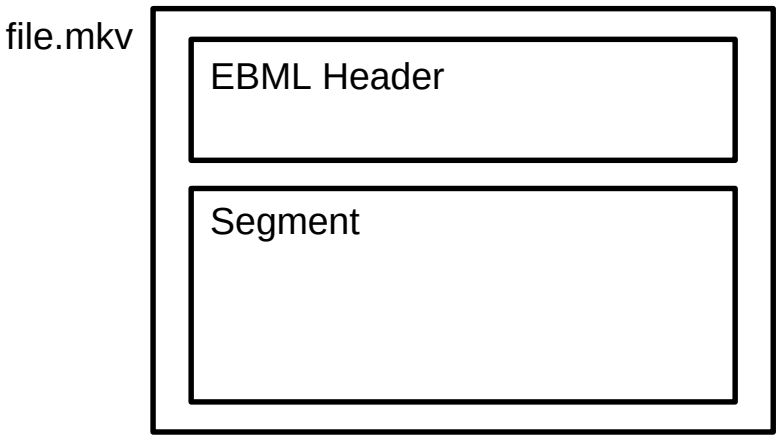
EBML Header

- At the beginning of each file that respects the EBML format there is a header that contains some information on how the file is formatted and what type of document it contains (the header also respects the format).
- The main fields of the header are:

Element Name	Level	Class-ID	Default	Element Type	Description
EBML	0	[1A][45][DF][A3]	-	sub-elements	Set the EBML characteristics of the data to follow. Each EBML document has to start with this.
EBMLVersion	1	[42][86]	1	u-integer	The version of EBML parser used to create the file.
EBMLReadVersion	1	[42][F7]	1	u-integer	The minimum EBML version a parser has to support to read this file.
EBMLMaxIDLength	1	[42][F2]	4	u-integer	The maximum length of the IDs you'll find in this file (4 or less in Matroska).
EBMLMaxSizeLength	1	[42][F3]	8	u-integer	The maximum length of the sizes you'll find in this file (8 or less in Matroska). This does not override the element size indicated at the beginning of an element. Elements that have an indicated size which is larger than what is allowed by EBMLMaxSizeLength shall be considered invalid.
DocType	1	[42][82]	-	string	A string that describes the type of document that follows this EBML header ('matroska' in our case).
DocTypeVersion	1	[42][87]	1	u-integer	The version of DocType interpreter used to create the file.
DocTypeReadVersion	1	[42][85]	1	u-integer	The minimum DocType version an interpreter has to support to read this file.

Matroska

- The Matroska format is an open container format that allows you to store in the same file an arbitrary number of *tracks*, of video, audio or text type (for subtitles).
- It is important to note that the format only indicates which elements a file is made of, not how to divide a stream into the EBML elements available.
- It is completely based on the EBML format and a Matroska file at its highest level (level 0) is composed of two elements:
  - The standard EBML header (with DocType containing the string «matroska»).
  - An element called «Segment» which contains all other elements and data.



Segment

- The *Segment* contains all the level 1 EBML elements indicated in the Matroska format:
  - *SeekHead*: contains an index of where all the other level 1 elements are located within the file (since they can be in any order).
  - *SegmentInfo*: contains generic information on the entire file.
  - *Tracks*: contains information on all the tracks present in the file.
  - *Chapters*: contains information on the chapters in which the file is divided.
  - *Clusters*: contains clusters, each of which contains some blocks of different tracks.
  - *Cues*: contains information on where you can jump in playing the file.
  - *Attachment*: is a section that can contain any type of file.
  - *Tagging*: contains all kinds of information about the tracks, like the ID3 tags of MP3 files.

Q

SeekHead

- The *SeekHead* element is not mandatory because to find out the position of the other level 1 elements you can simply scroll through the file (even if it is more expensive).
- It has a very simple structure, it maintains key-value pairs called *Seeks* that contain IDs and locations:

```
<SeekID> <SeekPosition>
```

- Where:
  - *SeekID*: is the EBML ID of a level 1 element.
  - *SeekPosition*: is the position in bytes with respect to the first level 1 element.

SeekHead

- SeekHead example from an mkv file:

- SeekHead (master) [position #59, identifier 0x114D9B74 (4), size 60 (2)]
  - Seek (master) [position #65, identifier 0x4DBB (2), size 11 (1)]
    - SeekID (binary) [position #68, identifier 0x53AB (2), size 4 (1)]
    - SeekPosition (uinteger) [position #75, identifier 0x53AC (2), size 1 (1)] = 223 (0xdf)
  - Seek (master) [position #79, identifier 0x4DBB (2), size 12 (1)]
    - SeekID (binary) [position #82, identifier 0x53AB (2), size 4 (1)]
    - SeekPosition (uinteger) [position #89, identifier 0x53AC (2), size 2 (1)] = 304 (0x130)
  - Seek (master) [position #94, identifier 0x4DBB (2), size 12 (1)]
    - SeekID (binary) [position #97, identifier 0x53AB (2), size 4 (1)]
    - SeekPosition (uinteger) [position #104, identifier 0x53AC (2), size 2 (1)] = 503 (0x1f7)
  - Seek (master) [position #109, identifier 0x4DBB (2), size 13 (1)]
    - SeekID (binary) [position #112, identifier 0x53AB (2), size 4 (1)]
    - SeekPosition (uinteger) [position #119, identifier 0x53AC (2), size 3 (1)] = 383.790 (0x5db2e)

SegmentInfo

- The *SegmentInfo* element is mandatory and contains generic information about the file.
- Some relevant elements are:
  - *SegmentUID*: a randomly generated unique ID to identify the file (128 bit).
  - *SegmentFilename*: a filename corresponding to the segment.
  - *TimecodeScale* (mandatory): contains a scale factor expressed in nanoseconds used to represent all the timecodes contained within the file.
  - *Duration*: duration of the file expressed using the TimecodeScale parameter.

- It also contains information on the applications used to create the container and to write the file (respectively *MuxingApp* and *WritingApp*).

SegmentInfo

- Example from an mkv file:

▼ Info (master) [position #282, identifier 0x1549A966 (4), size 69 (8)]
- TimecodeScale (uinteger) [position #294, identifier 0x2AD7B1 (3), size 3 (1)] = 1.000.000 (0xf4240)
- MuxingApp (utf-8) [position #301, identifier 0x4D80 (2), size 13 (1)] = Lavf57.37.101
- WritingApp (utf-8) [position #317, identifier 0x5741 (2), size 13 (1)] = Lavf57.37.101
- SegmentUID (binary) [position #333, identifier 0x73A4 (2), size 16 (1)]
- Duration (float) [position #352, identifier 0x4489 (2), size 8 (1)] = 5.568,000000 (5568.0)

Tracks

- The *Tracks* segment contains information on all the tracks in the file.
- For each track there is a *TrackEntry* element which mainly contains the following elements:
  - *TrackNumber*: track number used later in clusters.
  - *TrackUID*: a unique identifier for the track.
  - *TrackType*: an 8-bit integer for the track type (1: video, 2: audio, 3: complex=audio and video combined, 0x10: logo, 0x11: subtitle, 0x12: buttons, 0x20: control)
  - *Language*: the language of the track.
  - *CodecID*: the identifier of the codec used to encode the track; the format specifications indicate the IDs available for the various codecs.
  - *CodecPrivate:* specific additional information of the codec.
  - *Video*: if the track is video, this master element contains all the video settings.
  - *Audio*: if the track is audio, this master element contains all the audio settings.

Q

Tracks

- Example from an mkv file:

- Tracks (master) [position #363, identifier 0x1654AE6B (4), size 187 (8)]
  - TrackEntry (master) [position #375, identifier 0xAE (1), size 111 (8)]
    - TrackNumber (uinteger) [position #384, identifier 0xD7 (1), size 1 (1)] = 1 (0x1)
    - TrackUID (uinteger) [position #387, identifier 0x73C5 (2), size 1 (1)] = 1 (0x1)
    - FlagLacing (uinteger) [position #391, identifier 0x9C (1), size 1 (1)] = 0 (0x0)
    - Language (string) [position #394, identifier 0x22B59C (3), size 3 (1)] = und
    - CodecID (string) [position #401, identifier 0x86 (1), size 15 (1)] = V_MPEG4/ISO/AVC
    - TrackType (uinteger) [position #418, identifier 0x83 (1), size 1 (1)] = 1 (0x1)
    - DefaultDuration (uinteger) [position #421, identifier 0x23E383 (3), size 4 (1)] = 33.333.333 (0x1fca055)
    - Video (master) [position #429, identifier 0xE0 (1), size 11 (8)]
      - PixelWidth (uinteger) [position #438, identifier 0xB0 (1), size 2 (1)] = 560 (0x230)
      - PixelHeight (uinteger) [position #442, identifier 0xBA (1), size 2 (1)] = 320 (0x140)
      - FlagInterlaced (uinteger) [position #446, identifier 0x9A (1), size 1 (1)] = 0 (0x0)
    - CodecPrivate (binary) [position #449, identifier 0x63A2 (2), size 43 (1)]
  - TrackEntry (master) [position #495, identifier 0xAE (1), size 58 (8)]
    - TrackNumber (uinteger) [position #504, identifier 0xD7 (1), size 1 (1)] = 2 (0x2)
    - TrackUID (uinteger) [position #507, identifier 0x73C5 (2), size 1 (1)] = 2 (0x2)
    - FlagLacing (uinteger) [position #511, identifier 0x9C (1), size 1 (1)] = 0 (0x0)
    - Language (string) [position #514, identifier 0x22B59C (3), size 3 (1)] = eng
    - CodecID (string) [position #521, identifier 0x86 (1), size 5 (1)] = A_AAC
    - TrackType (uinteger) [position #528, identifier 0x83 (1), size 1 (1)] = 2 (0x2)
    - Audio (master) [position #531, identifier 0xE1 (1), size 17 (8)]
      - Channels (uinteger) [position #540, identifier 0x9F (1), size 1 (1)] = 1 (0x1)
      - SamplingFrequency (float) [position #543, identifier 0xB5 (1), size 8 (1)] = 48.000,00000 (48000.0)
      - BitDepth (uinteger) [position #553, identifier 0x6264 (2), size 1 (1)] = 16 (0x10)
    - CodecPrivate (binary) [position #557, identifier 0x63A2 (2), size 2 (1)]

Clusters

- *Clusters* are the elements that contain the encoded binary data of the various tracks.
- Within the *Segment* there are many clusters, which contain blocks (called *SimpleBlock* or *BlockGroup*) of data, each of which is tied to a single track.

- Generally a cluster contains data for a total of a few MB or a few seconds of playback. The choice of how much data to include is left to the application that creates the file.

- In addition to blocks, a cluster contains some other important elements:
  - *Timecode* (mandatory): contains an absolute timestamp of the cluster (always based on the TimecodeScale present in the SegmentInfo). **Q**
  - *PrevSize*: contains the size in bytes of the previous cluster, not mandatory but useful for playback or backward search. **Q**

SimpleBlock e BlockGroup

- You can use two types of element to hold the encoded tracks data, SimpleBlock or BlockGroup.
- Both perform the same function but BlockGroups can add more information to the data block.
- SimpleBlock is a binary element that directly contains data.
- The BlockGroup is instead a master element that can also contain other elements, among which the most important are:
  - *Block*: the binary element containing the data.
  - *BlockDuration*: the duration of the block expressed in relation to the TimecodeScale.
  - *ReferenceBlock*: contains the timestamp of another frame to be used as a reference (to indicate relationships between frames of type B and P and a frame of type I).
- When the software that creates the file assumes that no additional information is needed for the Blocks, it can use SimpleBlocks directly.

SimpleBlock e BlockGroup

- Example of a series of clusters containing only SimpleBlock:

▼ Cluster (master) [position #957, identifier 0x1F43B675 (4), size 359.331 (8)]
- Timecode (uinteger) [position #969, identifier 0xE7 (1), size 1 (1)] = 0 (0x0)
- SimpleBlock (binary) [position #972, identifier 0xA3 (1), size 22.059 (3)]
- SimpleBlock (binary) [position #23035, identifier 0xA3 (1), size 251 (2)]
- SimpleBlock (binary) [position #23289, identifier 0xA3 (1), size 223 (2)]
- SimpleBlock (binary) [position #23515, identifier 0xA3 (1), size 2.852 (2)]
- SimpleBlock (binary) [position #26370, identifier 0xA3 (1), size 229 (2)]
- SimpleBlock (binary) [position #26602, identifier 0xA3 (1), size 233 (2)]
- SimpleBlock (binary) [position #26838, identifier 0xA3 (1), size 1.472 (2)]
- SimpleBlock (binary) [position #28313, identifier 0xA3 (1), size 237 (2)]
- SimpleBlock (binary) [position #28553, identifier 0xA3 (1), size 1.510 (2)]
- SimpleBlock (binary) [position #30066, identifier 0xA3 (1), size 236 (2)]
- SimpleBlock (binary) [position #30305, identifier 0xA3 (1), size 244 (2)]
- SimpleBlock (binary) [position #30552, identifier 0xA3 (1), size 1.477 (2)]
- SimpleBlock (binary) [position #32032, identifier 0xA3 (1), size 245 (2)]
- SimpleBlock (binary) [position #32280, identifier 0xA3 (1), size 1.083 (2)]
- SimpleBlock (binary) [position #33366, identifier 0xA3 (1), size 243 (2)]

⋮

SimpleBlock e BlockGroup

- Example of a series of clusters containing only BlockGroup:

- Cluster (master) [position #276636, identifier 0x1F43B675 (4), size 32.557 (8)]
  - Timecode (uinteger) [position #276648, identifier 0xE7 (1), size 1 (1)] = 0 (0x0)
  - BlockGroup (master) [position #276651, identifier 0xA0 (1), size 2.853 (8)]
    - Block (binary) [position #276660, identifier 0xA1 (1), size 2.850 (2)]
  - BlockGroup (master) [position #279513, identifier 0xA0 (1), size 302 (8)]
    - Block (binary) [position #279522, identifier 0xA1 (1), size 296 (2)]
    - ReferenceBlock (integer) [position #279821, identifier 0xFB (1), size 1 (1)] = -125 (0xffffffffffffff83)
  - BlockGroup (master) [position #279824, identifier 0xA0 (1), size 273 (8)]
    - Block (binary) [position #279833, identifier 0xA1 (1), size 267 (2)]
    - ReferenceBlock (integer) [position #280103, identifier 0xFB (1), size 1 (1)] = 84 (0x54)
  - BlockGroup (master) [position #280106, identifier 0xA0 (1), size 586 (8)]
    - Block (binary) [position #280115, identifier 0xA1 (1), size 580 (2)]
    - BlockDuration (uinteger) [position #280698, identifier 0x9B (1), size 1 (1)] = 24 (0x18)
  - BlockGroup (master) [position #280701, identifier 0xA0 (1), size 586 (8)]
    - Block (binary) [position #280710, identifier 0xA1 (1), size 580 (2)]
    - BlockDuration (uinteger) [position #281293, identifier 0x9B (1), size 1 (1)] = 24 (0x18)
  - BlockGroup (master) [position #281296, identifier 0xA0 (1), size 288 (8)]
    - Block (binary) [position #281305, identifier 0xA1 (1), size 285 (2)]
  - BlockGroup (master) [position #281593, identifier 0xA0 (1), size 57 (8)]
    - Block (binary) [position #281602, identifier 0xA1 (1), size 55 (1)]
  - BlockGroup (master) [position #281659, identifier 0xA0 (1), size 586 (8)]

⋮

Block

- Inside, a Block contains binary data preceded by a header. The data must be interpreted in a way specified by the format (this time not EBML).
- Within a Block we find a header with the following elements:
  - *Track Number*: the Track number to which the data contained in the Block belongs, it is an integer stored as a VINT of the EBML format.
  - *Timecode*: a 16-bit signed integer relating to the timecode of the cluster to which the Block belongs.
  - *Flags*: one byte of flags:

Q

Bit	Player	Description
0-3	-	Reserved, set to 0
4	-	Invisible, the codec should decode this frame but not display it
5-6	must	Lacing: • 00 : no lacing • 01 : Xiph lacing • 11 : EBML lacing • 10 : fixed-size lacing
7	-	not used

- *Lacing* saves space by encoding multiple frames together.

Block

- If lacing is used, there are two more fields:
  - *Number of frames*: an 8-bit unsigned integer indicating how many frames are stored within the Block.
  - *Dimensions*: the size of each frame is indicated. The way of representing the dimensions can vary according to the type of lacing.

- After the lacing information (if any), the block frame(s) are directly stored.

SimpleBlock

- The structure of a SimpleBlock is almost the same as that of a Block, the only difference is in the byte of Flags:

Bit	Player	Description
0	-	Keyframe, set when the Block contains only keyframes
1-3	-	Reserved, set to 0
4	-	Invisible, the codec should decode this frame but not display it
5-6	must	Lacing: • 00: no lacing • 01: Xiph lacing • 11: EBML lacing • 10: fixed-size lacing
7	-	Discardable, the frames of the Block can be discarded during playing if needed

- In this case the most significant bit of the Flags is used to indicate whether the data contained in the current SimpleBlock belong to a keyframe.

Lacing

- Lacing is a mechanism to save space.
- It is used when some small-sized frames must be saved to the file in succession and you don’t want to use new SimpleBlocks or BlockGroups for each of them.

- The Matroska format supports three different types of lacing:
  - Xiph lacing.
  - EBML lacing.
  - Fixed-size lacing.

- The mechanism is the same in all three cases, what changes is the way of representing the number and size of the frames in the corresponding field of the Block or SimpleBlock.

Cues

- The *Cues* element contains a series of *CuePoints* that allow the application to skip with playback in different points without having to scroll through all the Clusters to find the right point.
- The most important elements contained in the CuePoints are:
  - *CueTime*: an absolute timestamp that uses the scale indicated in the SegmentInfo which indicates the time position of the CuePoint.
  - *CueTrackPosition*: is a master element that contains all the information on how to reach the timestamp indicated in CueTime within a specific track:
    - *CueTrack*: indicates the track number to which it refers.
    - *CueClusterPosition*: indicates the position in bytes within the Segment of the cluster that contains the indicated Block.
    - *CueRelativePosition*: relative position of the Block within the cluster.

Cues

- Example of CuePoints in an mkv file:

- Cues (master) [position #754542366, identifier 0x1C53BB6B (4), size 9.943 (8)]
  - CuePoint (master) [position #754542378, identifier 0xBB (1), size 20 (8)]
    - CueTime (uinteger) [position #754542387, identifier 0xB3 (1), size 1 (1)] = 0 (0x0)
    - CueTrackPositions (master) [position #754542390, identifier 0xB7 (1), size 8 (8)]
      - CueTrack (uinteger) [position #754542399, identifier 0xF7 (1), size 1 (1)] = 1 (0x1)
      - CueClusterPosition (uinteger) [position #754542402, identifier 0xF1 (1), size 3 (1)] = 276.577 (0x43861)
  - CuePoint (master) [position #754542407, identifier 0xBB (1), size 21 (8)]
    - CueTime (uinteger) [position #754542416, identifier 0xB3 (1), size 2 (1)] = 4.000 (0xfa0)
    - CueTrackPositions (master) [position #754542420, identifier 0xB7 (1), size 8 (8)]
      - CueTrack (uinteger) [position #754542429, identifier 0xF7 (1), size 1 (1)] = 1 (0x1)
      - CueClusterPosition (uinteger) [position #754542432, identifier 0xF1 (1), size 3 (1)] = 541.461 (0x84315)
  - CuePoint (master) [position #754542437, identifier 0xBB (1), size 21 (8)]
    - CueTime (uinteger) [position #754542446, identifier 0xB3 (1), size 2 (1)] = 8.000 (0x1f40)
    - CueTrackPositions (master) [position #754542450, identifier 0xB7 (1), size 8 (8)]
      - CueTrack (uinteger) [position #754542459, identifier 0xF7 (1), size 1 (1)] = 1 (0x1)
      - CueClusterPosition (uinteger) [position #754542462, identifier 0xF1 (1), size 3 (1)] = 788.805 (0xc0945)
  - CuePoint (master) [position #754542467, identifier 0xBB (1), size 21 (8)]
    - CueTime (uinteger) [position #754542476, identifier 0xB3 (1), size 2 (1)] = 12.000 (0x2ee0)
    - CueTrackPositions (master) [position #754542480, identifier 0xB7 (1), size 8 (8)]
      - CueTrack (uinteger) [position #754542489, identifier 0xF7 (1), size 1 (1)] = 1 (0x1)
      - CueClusterPosition (uinteger) [position #754542492, identifier 0xF1 (1), size 3 (1)] = 2.430.626 (0x2516a2)
  - CuePoint (master) [position #754542497, identifier 0xBB (1), size 21 (8)]
    - CueTime (uinteger) [position #754542506, identifier 0xB3 (1), size 2 (1)] = 16.000 (0x3e80)
    - CueTrackPositions (master) [position #754542510, identifier 0xB7 (1), size 8 (8)]

⋮

Chapters

- *Chapters* are a mechanism to locate precise points in the file from which to start playback.
- They differ from the CuePoint because Chapters can be associated with strings in different languages to allow the user to understand exactly where a certain Chapter will lead into the file.
- Chapters indicate semantically interesting points of a file; in the case of a film it could be for example the different scenes of which it is composed while in the case of a music album it could be the different songs.
- For each Chapter different data are indicated, for example:
  - *ChapterUID*: unique ID of the chapter.
  - *ChapterTimeStart*: chapter start timestamp (unscaled).
  - *ChapterTimeEnd*: chapter end timestamp (unscaled).
  - *ChapterDisplay*: chapter related strings (there may be multiple languages in the same ChapterDisplay):
    - *ChapterString*: chapter description.
    - *ChapterLanguage*: corresponding ChapterString language.

Chapters

- Chapters example from an mkv file:

- Chapters (master) [position #274080, identifier 0x1043A770 (4), size 2.544 (8)]
  - EditionEntry (master) [position #274092, identifier 0x45B9 (2), size 2.534 (8)]
    - EditionUID (uinteger) [position #274102, identifier 0x45BC (2), size 8 (1)] = -4.497.399.309.340.229.568 (0xc196072809b04040)
    - EditionFlagHidden (uinteger) [position #274113, identifier 0x45BD (2), size 1 (1)] = 0 (0x0)
    - EditionFlagOrdered (uinteger) [position #274117, identifier 0x45DD (2), size 1 (1)] = 0 (0x0)
    - ChapterAtom (master) [position #274121, identifier 0xB6 (1), size 206 (8)]
    - ChapterAtom (master) [position #274336, identifier 0xB6 (1), size 246 (8)]
      - ChapterUID (uinteger) [position #274345, identifier 0x73C4 (2), size 8 (1)] = 6.182.029.356.179.473.180 (0x55cafaf2161d431c)
      - ChapterTimeStart (uinteger) [position #274356, identifier 0x91 (1), size 5 (1)] = 40.417.000.000 (0x9690a7a40)
      - ChapterTimeEnd (uinteger) [position #274363, identifier 0x92 (1), size 5 (1)] = 51.875.000.000 (0xc13fdaec0)
      - ChapterFlagHidden (uinteger) [position #274370, identifier 0x98 (1), size 1 (1)] = 0 (0x0)
      - ChapterFlagEnabled (uinteger) [position #274373, identifier 0x4598 (2), size 1 (1)] = 1 (0x1)
      - ChapterDisplay (master) [position #274377, identifier 0x80 (1), size 18 (8)]
        - ChapString (utf-8) [position #274386, identifier 0x85 (1), size 10 (1)] = Snow Field
        - ChapLanguage (string) [position #274398, identifier 0x437C (2), size 3 (1)] = eng
      - ChapterDisplay (master) [position #274404, identifier 0x80 (1), size 28 (8)]
        - ChapString (utf-8) [position #274413, identifier 0x85 (1), size 20 (1)] = Neige sur le terrain
        - ChapLanguage (string) [position #274435, identifier 0x437C (2), size 3 (1)] = fre

⋮

Tags

- The *Tags* element can contain generic metadata about the file or about particular elements of the file.
- The Tags element contains one or more Tag type elements, each with its sub-elements:
  - *Targets*: it contains the unique IDs of the elements to which the tags refer (not the EBML IDs, the unique IDs of the Matroska format). If this element is empty, then the tags refer to the whole segment.
  - *SimpleTag*: a master element that can contain generic metadata, its possible sub-elements are:
    - *TagName*: name of the stored tag.
    - *TagLanguage*: language in which the tag is stored.
    - *TagString*: the value of the tag as a string.
    - *TagBinary*: the value of the tag as binary data.
    - *SimpleTag*: another element of type SimpleTag.

    Note: TagString and TagBinary cannot be used together.

- There are some official tags supported by Matroska but it is possible to add arbitrary ones.

Tags

- Example of Tags, in this case the Targets element is empty, so all the metadata refer to the entire Segment:

```
▼ Tags (master) [position #754553479, identifier 0x1254C367 (4), size 12.428 (8)]
  ▼ Tag (master) [position #754553491, identifier 0x7373 (2), size 12.418 (8)]
      Targets (master) [position #754553501, identifier 0x63C0 (2), size 0 (8)]
    ▼ SimpleTag (master) [position #754553511, identifier 0x67C8 (2), size 68 (8)]
        TagName (utf-8) [position #754553521, identifier 0x45A3 (2), size 5 (1)] = TITLE
        TagString (utf-8) [position #754553529, identifier 0x4487 (2), size 6 (1)] = Sintel
      ► SimpleTag (master) [position #754553538, identifier 0x67C8 (2), size 41 (8)]
    ▼ SimpleTag (master) [position #754553589, identifier 0x67C8 (2), size 45 (8)]
        TagName (utf-8) [position #754553599, identifier 0x45A3 (2), size 6 (1)] = ARTIST
        TagString (utf-8) [position #754553608, identifier 0x4487 (2), size 33 (1)] = Durian Blender Open Movie Project
    ▼ SimpleTag (master) [position #754553644, identifier 0x67C8 (2), size 28 (8)]
        TagName (utf-8) [position #754553654, identifier 0x45A3 (2), size 8 (1)] = PRODUCER
        TagString (utf-8) [position #754553665, identifier 0x4487 (2), size 14 (1)] = Ton Roosendaal
    ▼ SimpleTag (master) [position #754553682, identifier 0x67C8 (2), size 64 (8)]
        TagName (utf-8) [position #754553692, identifier 0x45A3 (2), size 9 (1)] = COPYRIGHT
        TagString (utf-8) [position #754553704, identifier 0x4487 (2), size 49 (1)] = Copyright Blender Foundation | durian.blender.org
    ▼ SimpleTag (master) [position #754553756, identifier 0x67C8 (2), size 120 (8)]
        TagName (utf-8) [position #754553766, identifier 0x45A3 (2), size 12 (1)] = TERMS_OF_USE
        TagString (utf-8) [position #754553781, identifier 0x4487 (2), size 40 (1)] = Creative Commons Attribution 3.0 License
      ▼ SimpleTag (master) [position #754553824, identifier 0x67C8 (2), size 52 (8)]
          TagName (utf-8) [position #754553834, identifier 0x45A3 (2), size 3 (1)] = URL
          TagString (utf-8) [position #754553840, identifier 0x4487 (2), size 43 (1)] = http://creativecommons.org/licenses/by/3.0/
  ⋮
```

Attachments

- In the *Attachments* element, generic files can be attached to the Matroska container.
- An example of a possible attached file is a file containing the font to be used for subtitles.
- The Attachments element, if present, contains one or more AttachedFile elements, each of which has the following sub-elements:
  - *FileName*: contains the name of the attached file.
  - *FileMimeType*: contains the mime type of the file.
  - *FileUID*: contains a unique ID that represents the file.
  - *FileDescription*: contains a description of the file.
  - *FileData*: a binary element that contains the actual file.

Attachments

- Example of Attachment from an mkv file which, in this case, contains a font file for subtitles named «AlteHaasGroteskRegular.ttf»: