

cellar  
Internet-Draft  
Intended status: Standards Track  
Expires: 30 July 2020

S. Lhomme  
D. Rice  
M. Bunkus  
27 January 2020

Extensible Binary Meta Language  
draft-ietf-cellar-ebml-17

Abstract

This document defines the Extensible Binary Meta Language (EBML) format as a binary container format designed for audio/video storage. EBML is designed as a binary equivalent to XML and uses a storage-efficient approach to build nested Elements with identifiers, lengths, and values. Similar to how an XML Schema defines the structure and semantics of an XML Document, this document defines how EBML Schemas are created to convey the semantics of an EBML Document.

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 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components



extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Notation and Conventions . . . . .	4
3. Structure . . . . .	6
4. Variable Size Integer . . . . .	7
4.1. VINT_WIDTH . . . . .	7
4.2. VINT_MARKER . . . . .	7
4.3. VINT_DATA . . . . .	7
4.4. VINT Examples . . . . .	8
5. Element ID . . . . .	9
6. Element Data Size . . . . .	11
6.1. Data Size Format . . . . .	11
6.2. Unknown Data Size . . . . .	12
6.3. Data Size Values . . . . .	14
7. EBML Element Types . . . . .	15
7.1. Signed Integer Element . . . . .	15
7.2. Unsigned Integer Element . . . . .	16
7.3. Float Element . . . . .	16
7.4. String Element . . . . .	16
7.5. UTF-8 Element . . . . .	16
7.6. Date Element . . . . .	17
7.7. Master Element . . . . .	17
7.8. Binary Element . . . . .	17
8. EBML Document . . . . .	18
8.1. EBML Header . . . . .	18
8.2. EBML Body . . . . .	18
9. EBML Stream . . . . .	19
10. EBML Versioning . . . . .	19
10.1. EBML Header Version . . . . .	19
10.2. EBML Document Version . . . . .	19
11. Elements semantic . . . . .	19
11.1. EBML Schema . . . . .	19
11.1.1. EBML Schema Example . . . . .	20
11.1.2. <EBMLSchema> Element . . . . .	21
11.1.3. <EBMLSchema> Attributes . . . . .	21
11.1.4. <element> Element . . . . .	22
11.1.5. <element> Attributes . . . . .	22
11.1.6. <documentation> Element . . . . .	30
11.1.7. <documentation> Attributes . . . . .	30
11.1.8. <implementation_note> Element . . . . .	31
11.1.9. <implementation_note> Attributes . . . . .	32
11.1.10. <restriction> Element . . . . .	33
11.1.11. <enum> Element . . . . .	33



11.1.12.	<enum> Attributes . . . . .	34
11.1.13.	<extension> Element . . . . .	34
11.1.14.	<extension> Attributes . . . . .	34
11.1.15.	XML Schema for EBML Schema . . . . .	35
11.1.16.	Identically Recurring Elements . . . . .	38
11.1.17.	Textual expression of floats . . . . .	39
11.1.18.	Note on the use of default attributes to define Mandatory EBML Elements . . . . .	40
11.2.	EBML Header Elements . . . . .	41
11.2.1.	EBML Element . . . . .	41
11.2.2.	EBMLVersion Element . . . . .	41
11.2.3.	EBMLReadVersion Element . . . . .	41
11.2.4.	EBMLMaxIDLength Element . . . . .	42
11.2.5.	EBMLMaxSizeLength Element . . . . .	42
11.2.6.	DocType Element . . . . .	43
11.2.7.	DocTypeVersion Element . . . . .	43
11.2.8.	DocTypeReadVersion Element . . . . .	44
11.2.9.	DocTypeExtension Element . . . . .	44
11.2.10.	DocTypeExtensionName Element . . . . .	45
11.2.11.	DocTypeExtensionVersion Element . . . . .	45
11.3.	Global Elements . . . . .	46
11.3.1.	CRC-32 Element . . . . .	46
11.3.2.	Void Element . . . . .	47
12.	Considerations for Reading EBML Data . . . . .	47
13.	Terminating Elements . . . . .	48
14.	Guidelines for Updating Elements . . . . .	49
14.1.	Reducing a Element Data in Size . . . . .	49
14.1.1.	Adding a Void Element . . . . .	49
14.1.2.	Extending the Element Data Size . . . . .	49
14.1.3.	Terminating Element Data . . . . .	50
14.2.	Considerations when Updating Elements with Cyclic Redundancy Check (CRC) . . . . .	51
15.	Backward and Forward Compatibility . . . . .	51
15.1.	Backward Compatibility . . . . .	51
15.2.	Forward Compatibility . . . . .	52
16.	Security Considerations . . . . .	52
17.	IANA Considerations . . . . .	53
17.1.	EBML Element ID Registry . . . . .	54
17.2.	EBML DocType Registry . . . . .	57
18.	Normative References . . . . .	57
19.	Informative References . . . . .	59
	Authors' Addresses . . . . .	59

## 1. Introduction

EBML, short for Extensible Binary Meta Language, specifies a binary and octet (byte) aligned format inspired by the principle of XML (a framework for structuring data).



The goal of this document is to define a generic, binary, space-efficient format that can be used to define more complex formats using an EBML Schema. EBML is used by the multimedia container, Matroska [Matroska]. The applicability of EBML for other use cases is beyond the scope of this document.

The definition of the EBML format recognizes the idea behind HTML and XML as a good one: separate structure and semantics allowing the same structural layer to be used with multiple, possibly widely differing semantic layers. Except for the EBML Header and a few Global Elements this specification does not define particular EBML format semantics; however this specification is intended to define how other EBML-based formats can be defined, such as the audio-video container formats Matroska and WebM [WebM].

EBML uses a simple approach of building Elements upon three pieces of data (tag, length, and value) as this approach is well known, easy to parse, and allows selective data parsing. The EBML structure additionally allows for hierarchical arrangement to support complex structural formats in an efficient manner.

A typical EBML file has the following structure:

```
EBML Header (master)
+ DocType (string)
+ DocTypeVersion (unsigned integer)
EBML Body Root (master)
+ ElementA (utf-8)
+ Parent (master)
  + ElementB (integer)
+ Parent (master)
  + ElementB (integer)
```

## 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.

This document defines specific terms in order to define the format and application of "EBML". Specific terms are defined below:

"EBML": Extensible Binary Meta Language

"EBML Document Type": A name provided by an "EBML Schema" to



designate a particular implementation of "EBML" for a data format (e.g.: matroska and webm).

"EBML Schema": A standardized definition for the structure of an "EBML Document Type".

"EBML Document": A datastream comprised of only two components, an "EBML Header" and an "EBML Body".

"EBML Reader": A data parser that interprets the semantics of an "EBML Document" and creates a way for programs to use "EBML".

"EBML Stream": A file that consists of one or more "EBML Documents" that are concatenated together.

"EBML Header": A declaration that provides processing instructions and identification of the "EBML Body". The "EBML Header" is analogous to an XML Declaration [W3C.REC-xml-20081126] (see section 2.8 on Prolog and Document Type Declaration).

"EBML Body": All data of an "EBML Document" following the "EBML Header".

"Variable Size Integer": A compact variable-length binary value which defines its own length.

"VINT": Also known as "Variable Size Integer".

"EBML Element": A foundation block of data that contains three parts: an "Element ID", an "Element Data Size", and "Element Data".

"Element ID": The "Element ID" is a binary value, encoded as a "Variable Size Integer", used to uniquely identify a defined "EBML Element" within a specific "EBML Schema".

"Element Data Size": An expression, encoded as a "Variable Size Integer", of the length in octets of "Element Data".

"VINTMAX": The maximum possible value that can be stored as "Element Data Size".

"Unknown-Sized Element": An "Element" with an unknown "Element Data Size".

"Element Data": The value(s) of the "EBML Element" which is identified by its "Element ID" and "Element Data Size". The form of the "Element Data" is defined by this document and the corresponding "EBML Schema" of the Element's "EBML Document Type".



"Root Level": The starting level in the hierarchy of an "EBML Document".

"Root Element": A mandatory, non-repeating "EBML Element" which occurs at the top level of the path hierarchy within an "EBML Body" and contains all other "EBML Elements" of the "EBML Body", excepting optional "Void Elements".

"Top-Level Element": An "EBML Element" defined to only occur as a "Child Element" of the "Root Element".

"Master Element": The "Master Element" contains zero, one, or many other "EBML Elements".

"Child Element": A "Child Element" is a relative term to describe the "EBML Elements" immediately contained within a "Master Element".

"Parent Element": A relative term to describe the "Master Element" which contains a specified element. For any specified "EBML Element" that is not at "Root Level", the "Parent Element" refers to the "Master Element" in which that "EBML Element" is directly contained.

"Descendant Element": A relative term to describe any "EBML Elements" contained within a "Master Element", including any of the "Child Elements" of its "Child Elements", and so on.

"Void Element": A "Void Element" is an "Element" used to overwrite data or reserve space within a "Master Element" for later use.

"Element Name": The human-readable name of the "EBML Element".

"Element Path": The hierarchy of "Parent Element" where the "EBML Element" is expected to be found in the "EBML Body".

"Empty Element": An "EBML Element" that has an "Element Data Size" with all "VINT\_DATA" bits set to zero, which indicates that the "Element Data" of the "Element" is zero octets in length.

### 3. Structure

EBML uses a system of Elements to compose an EBML Document. EBML Elements incorporate three parts: an Element ID, an Element Data Size, and Element Data. The Element Data, which is described by the Element ID, includes either binary data, one or more other EBML Elements, or both.



#### 4. Variable Size Integer

The Element ID and Element Data Size are both encoded as a Variable Size Integer. The Variable Size Integer is composed of a VINT\_WIDTH, VINT\_MARKER, and VINT\_DATA, in that order. Variable Size Integers MUST left-pad the VINT\_DATA value with zero bits so that the whole Variable Size Integer is octet-aligned. Variable Size Integer will be referred to as VINT for shorthand.

##### 4.1. VINT\_WIDTH

Each Variable Size Integer starts with a VINT\_WIDTH followed by a VINT\_MARKER. VINT\_WIDTH is a sequence of zero or more bits of value "0", and is terminated by the VINT\_MARKER, which is a single bit of value "1". The total length in bits of both VINT\_WIDTH and VINT\_MARKER is the total length in octets in of the Variable Size Integer.

The single bit "1" starts a Variable Size Integer with a length of one octet. The sequence of bits "01" starts a Variable Size Integer with a length of two octets. "001" starts a Variable Size Integer with a length of three octets, and so on, with each additional 0-bit adding one octet to the length of the Variable Size Integer.

##### 4.2. VINT\_MARKER

The VINT\_MARKER serves as a separator between the VINT\_WIDTH and VINT\_DATA. Each Variable Size Integer MUST contain exactly one VINT\_MARKER. The VINT\_MARKER is one bit in length and contain a bit with a value of one. The first bit with a value of one within the Variable Size Integer is the VINT\_MARKER.

##### 4.3. VINT\_DATA

The VINT\_DATA portion of the Variable Size Integer includes all data that follows (but not including) the VINT\_MARKER until end of the Variable Size Integer whose length is derived from the VINT\_WIDTH. The bits required for the VINT\_WIDTH and the VINT\_MARKER use one out of every eight bits of the total length of the Variable Size Integer. Thus a Variable Size Integer of 1 octet length supplies 7 bits for VINT\_DATA, a 2 octet length supplies 14 bits for VINT\_DATA, and a 3 octet length supplies 21 bits for VINT\_DATA. If the number of bits required for VINT\_DATA are less than the bit size of VINT\_DATA, then VINT\_DATA MUST be zero-padded to the left to a size that fits. The VINT\_DATA value MUST be expressed as a big-endian unsigned integer.



#### 4.4. VINT Examples

Table 1 shows examples of Variable Size Integers with lengths from 1 to 5 octets. The Usable Bits column refers to the number of bits that can be used in the VINT\_DATA. The Representation column depicts a binary expression of Variable Size Integers where VINT\_WIDTH is depicted by "0", the VINT\_MARKER as "1", and the VINT\_DATA as "x".

Octet Length	Usable Bits	Representation
1	7	1xxx xxxx
2	14	01xx xxxx xxxx xxxx
3	21	001x xxxx xxxx xxxx xxxx xxxx
4	28	0001 xxxx xxxx xxxx xxxx xxxx xxxx xxxx
5	35	0000 1xxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx

Table 1: VINT examples depicting usable bits

A Variable Size Integer may be rendered at octet lengths larger than needed to store the data in order to facilitate overwriting it at a later date, e.g. when its final size isn't known in advance. In Table 2 an integer "2" (with a corresponding binary value of 0b10) is shown encoded as different Variable Size Integers with lengths from one octet to four octets. All four encoded examples have identical semantic meaning though the VINT\_WIDTH and the padding of the VINT\_DATA vary.



Integer	Octet Length	As Represented in VINT (binary)	As Represented in VINT (hexadecimal)
2	1	1000 0010	0x82
2	2	0100 0000 0000 0010	0x4002
2	3	0010 0000 0000 0000 0000 0010	0x200002
2	4	0001 0000 0000 0000 0000 0000 0000 0010	0x10000002

Table 2: VINT examples depicting the same integer value rendered at different VINT lengths

## 5. Element ID

An Element ID is a Variable Size Integer. By default, Element IDs are from one octet to four octets in length, although Element IDs of greater lengths MAY be used if the EBMLMaxIDLength Element of the EBML Header is set to a value greater than four (see Section 11.2.4). The bits of the VINT\_DATA component of the Element ID MUST NOT be all "0" values or all "1" values. The VINT\_DATA component of the Element ID MUST be encoded at the shortest valid length. For example, an Element ID with binary encoding of "1011 1111" is valid, whereas an Element ID with binary encoding of "0100 0000 0011 1111" stores a semantically equal VINT\_DATA but is invalid because a shorter VINT encoding is possible. Additionally, an Element ID with binary encoding of "1111 1111" is invalid since the VINT\_DATA section is set to all one values, whereas an Element ID with binary encoding of "0100 0000 0111 1111" stores a semantically equal VINT\_DATA and is the shortest possible VINT encoding.

Table 3 details these specific examples further:



VINT_WIDTH	VINT_MARKER	VINT_DATA	Element ID Status
	1	0000000	Invalid: VINT_DATA MUST NOT be set to all 0
0	1	000000000000000	Invalid: VINT_DATA MUST NOT be set to all 0
	1	0000001	Valid
0	1	000000000000001	Invalid: A shorter VINT_DATA encoding is available.
	1	0111111	Valid
0	1	000000001111111	Invalid: A shorter VINT_DATA encoding is available.
	1	1111111	Invalid: VINT_DATA MUST NOT be set to all 1
0	1	000000011111111	Valid

Table 3: Examples of valid and invalid Element IDs

The range and count of possible Element IDs are determined by their octet length. Examples of this are provided in Table 4.



Element ID Octet Length	Range of Valid Element IDs	Number of Valid Element IDs
1	0x81 - 0xFE	126
2	0x407F - 0x7FFE	16,256
3	0x203FFF - 0x3FFFFE	2,080,768
4	0x101FFFFF - 0x1FFFFFFE	268,338,304

Table 4: Examples of count and range for Element IDs at various octet lengths

## 6. Element Data Size

### 6.1. Data Size Format

The Element Data Size expresses the length in octets of Element Data. The Element Data Size itself is encoded as a Variable Size Integer. By default, Element Data Sizes can be encoded in lengths from one octet to eight octets, although Element Data Sizes of greater lengths MAY be used if the octet length of the longest Element Data Size of the EBML Document is declared in the EBMLMaxSizeLength Element of the EBML Header (see Section 11.2.5). Unlike the VINT\_DATA of the Element ID, the VINT\_DATA component of the Element Data Size is not mandated to be encoded at the shortest valid length. For example, an Element Data Size with binary encoding of 1011 1111 or a binary encoding of 0100 0000 0011 1111 are both valid Element Data Sizes and both store a semantically equal value (both 0b00000000111111 and 0b01111111, the VINT\_DATA sections of the examples, represent the integer 63).

Although an Element ID with all VINT\_DATA bits set to zero is invalid, an Element Data Size with all VINT\_DATA bits set to zero is allowed for EBML Element Types which do not mandate a non-zero length (see Section 7). An Element Data Size with all VINT\_DATA bits set to zero indicates that the Element Data is zero octets in length. Such an EBML Element is referred to as an Empty Element. If an Empty Element has a default value declared then the EBML Reader MUST interpret the value of the Empty Element as the default value. If an Empty Element has no default value declared then the EBML Reader MUST use the value of the Empty Element for the corresponding EBML Element



Type of the Element ID, 0 for numbers and an empty string for strings.

## 6.2. Unknown Data Size

An Element Data Size with all VINT\_DATA bits set to one is reserved as an indicator that the size of the EBML Element is unknown. The only reserved value for the VINT\_DATA of Element Data Size is all bits set to one. An EBML Element with an unknown Element Data Size is referred to as an Unknown-Sized Element. Only a Master Element is allowed to be of unknown size, and it can only be so if the unknownsizeallowed attribute of its EBML Schema is set to true (see Section 11.1.5.10).

The use of Unknown-Sized Elements allows for an EBML Element to be written and read before the size of the EBML Element is known. Unknown-Sized Elements MUST only be used if the Element Data Size is not known before the Element Data is written, such as in some cases of data streaming. The end of an Unknown-Sized Element is determined by whichever comes first:

- \* Any EBML Element that is a valid Parent Element of the Unknown-Sized Element according to the EBML Schema, Global Elements excluded.
- \* Any valid EBML Element according to the EBML Schema, Global Elements excluded, that is not a Descendant Element of the Unknown-Sized Element but share a common direct parent, such as a Top-Level Element.
- \* Any EBML Element that is a valid Root Element according to the EBML Schema, Global Elements excluded.
- \* The end of the Parent Element with a known size has been reached.
- \* The end of the EBML Document, either when reaching the end of the file or because a new EBML Header started.

Consider an Unknown-Sized Element which EBML path is "\root\level1\level2\elt". When reading a new Element ID, assuming the EBML Path of that new Element is valid, here are some possible and impossible ways that this new Element is ending "elt":



EBML Path of new element	Status
"\root\level1\level2"	Ends the Unknown-Sized Element; as it is a new Parent Element
"\root\level1"	Ends the Unknown-Sized Element; as it is a new Parent Element
"\root"	Ends the Unknown-Sized Element; as it is a new Root Element
"\root2"	Ends the Unknown-Sized Element; as it is a new Root Element
"\root\level1\level2\other"	Ends the Unknown-Sized Element; as they share the same parent
"\root\level1\level2\elt"	Ends the Unknown-Sized Element; as they share the same parent
"\root\level1\level2\elt\inside"	Doesn't end the Unknown-Sized Element; it's a child of "elt"
"\root\level1\level2\elt<global>"	Global Element is valid; it's a child of "elt"
"\root\level1\level2<global>"	Global Element cannot be interpreted with this path; while parsing "elt" a Global Element can only be a child of "elt"

Table 5: Examples of determining the end of an Unknown-Sized Element



### 6.3. Data Size Values

For Element Data Sizes encoded at octet lengths from one to eight, Table 6 depicts the range of possible values that can be encoded as an Element Data Size. An Element Data Size with an octet length of 8 is able to express a size of  $2^{56}-2$  or 72,057,594,037,927,934 octets (or about 72 petabytes). The maximum possible value that can be stored as Element Data Size is referred to as VINTMAX.

Octet Length	Possible Value Range
1	0 to $2^7 - 2$
2	0 to $2^{14} - 2$
3	0 to $2^{21} - 2$
4	0 to $2^{28} - 2$
5	0 to $2^{35} - 2$
6	0 to $2^{42} - 2$
7	0 to $2^{49} - 2$
8	0 to $2^{56} - 2$

Table 6: Possible range of values  
that can be stored in VINTs by  
octet length.

If the length of Element Data equals  $2^{(n*7)-1}$  then the octet length of the Element Data Size MUST be at least  $n+1$ . This rule prevents an Element Data Size from being expressed as the unknown size value. Table 7 clarifies this rule by showing a valid and invalid expression of an Element Data Size with a VINT\_DATA of 127 (which is equal to  $2^{(1*7)-1}$ ) and 16,383 (which is equal to  $2^{((2*7)-1)}$ .)



VINT_WIDTH	VINT_MARKER	VINT_DATA	Element Data Size Status
	1	1111111	Reserved (meaning Unknown)
0	1	00000001111111	Valid (meaning 127 octets)
00	1	00000000000000111111	Valid (meaning 127 octets)
0	1	11111111111111	Reserved (meaning Unknown)
00	1	00000001111111111111	Valid (16,383 octets)

Table 7: Demonstration of VINT\_DATA reservation for VINTs of unknown size.

## 7. EBML Element Types

EBML Elements are defined by an EBML Schema (see Section 11.1) which MUST declare one of the following EBML Element Types for each EBML Element. An EBML Element Type defines a concept of storing data within an EBML Element that describes such characteristics as length, endianness, and definition.

EBML Elements which are defined as a Signed Integer Element, Unsigned Integer Element, Float Element, or Date Element use big endian storage.

### 7.1. Signed Integer Element

A Signed Integer Element MUST declare a length from zero to eight octets. If the EBML Element is not defined to have a default value, then a Signed Integer Element with a zero-octet length represents an integer value of zero.



A Signed Integer Element stores an integer (meaning that it can be written without a fractional component) which could be negative, positive, or zero. Signed Integers are stored with two's complement notation with the leftmost bit being the sign bit. Because EBML limits Signed Integers to 8 octets in length a Signed Integer Element stores a number from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

## 7.2. Unsigned Integer Element

An Unsigned Integer Element MUST declare a length from zero to eight octets. If the EBML Element is not defined to have a default value, then an Unsigned Integer Element with a zero-octet length represents an integer value of zero.

An Unsigned Integer Element stores an integer (meaning that it can be written without a fractional component) which could be positive or zero. Because EBML limits Unsigned Integers to 8 octets in length an Unsigned Integer Element stores a number from 0 to 18,446,744,073,709,551,615.

## 7.3. Float Element

A Float Element MUST declare a length of either zero octet (0 bit), four octets (32 bit) or eight octets (64 bit). If the EBML Element is not defined to have a default value, then a Float Element with a zero-octet length represents a numerical value of zero.

A Float Element stores a floating-point number in the 32-bit and 64-bit binary interchange format as defined in [IEEE.754.1985].

## 7.4. String Element

A String Element MUST declare a length in octets from zero to VINTMAX. If the EBML Element is not defined to have a default value, then a String Element with a zero-octet length represents an empty string.

A String Element MUST either be empty (zero-length) or contain printable ASCII characters [RFC0020] in the range of 0x20 to 0x7E, with an exception made for termination (see Section 13).

## 7.5. UTF-8 Element

A UTF-8 Element MUST declare a length in octets from zero to VINTMAX. If the EBML Element is not defined to have a default value, then a UTF-8 Element with a zero-octet length represents an empty string.



A UTF-8 Element contains only a valid Unicode string as defined in [RFC3629], with an exception made for termination (see Section 13).

#### 7.6. Date Element

A Date Element MUST declare a length of either zero octets or eight octets. If the EBML Element is not defined to have a default value, then a Date Element with a zero-octet length represents a timestamp of 2001-01-01T00:00:00.000000000 UTC [RFC3339].

The Date Element stores an integer in the same format as the Signed Integer Element that expresses a point in time referenced in nanoseconds from the precise beginning of the third millennium of the Gregorian Calendar in Coordinated Universal Time (also known as 2001-01-01T00:00:00.000000000 UTC). This provides a possible expression of time from 1708-09-11T00:12:44.854775808 UTC to 2293-04-11T11:47:16.854775807 UTC.

#### 7.7. Master Element

A Master Element MUST declare a length in octets from zero to VINTMAX or be of unknown length. See Section 6 for rules that apply to elements of unknown length.

The Master Element contains zero or more other elements. EBML Elements contained within a Master Element MUST have the EBMLParentPath of their Element Path equal to the EBMLFullPath of the Master Element Element Path (see Section 11.1.5.2). Element Data stored within Master Elements SHOULD only consist of EBML Elements and SHOULD NOT contain any data that is not part of an EBML Element. The EBML Schema identifies what Element IDs are valid within the Master Elements for that version of the EBML Document Type. Any data contained within a Master Element that is not part of a Child Element MUST be ignored.

#### 7.8. Binary Element

A Binary Element MUST declare a length in octets from zero to VINTMAX.

The contents of a Binary Element should not be interpreted by the EBML Reader.



## 8. EBML Document

An EBML Document is composed of only two components, an EBML Header and an EBML Body. An EBML Document MUST start with an EBML Header that declares significant characteristics of the entire EBML Body. An EBML Document consists of EBML Elements and MUST NOT contain any data that is not part of an EBML Element.

### 8.1. EBML Header

The EBML Header is a declaration that provides processing instructions and identification of the EBML Body. The EBML Header of an EBML Document is analogous to the XML Declaration of an XML Document.

The EBML Header documents the EBML Schema (also known as the EBML DocType) that is used to semantically interpret the structure and meaning of the EBML Document. Additionally the EBML Header documents the versions of both EBML and the EBML Schema that were used to write the EBML Document and the versions required to read the EBML Document.

The EBML Header MUST contain a single Master Element with an Element Name of EBML and Element ID of 0x1A45DFA3 (see Section 11.2.1) and any number of additional EBML Elements within it. The EBML Header of an EBML Document that uses an EBMLVersion of 1 MUST only contain EBML Elements that are defined as part of this document.

Elements within an EBML Header can be at most 4 octets long, except for the EBML Element with Element Name EBML and Element ID "0x1A45DFA3" (see Section 11.2.1), which can be up to 8 octets long.

### 8.2. EBML Body

All data of an EBML Document following the EBML Header is the EBML Body. The end of the EBML Body, as well as the end of the EBML Document that contains the EBML Body, is reached at whichever comes first: the beginning of a new EBML Header at the Root Level or the end of the file. This document defines precisely which EBML Elements are to be used within the EBML Header, but does not name or define which EBML Elements are to be used within the EBML Body. The definition of which EBML Elements are to be used within the EBML Body is defined by an EBML Schema.

Within the EBML Body, the maximum octet length allowed for any Element ID is set by the EBMLMaxIDLength Element of the EBML Header and the maximum octet length allowed for any Element Data Size is set by the EBMLMaxSizeLength Element of the EBML Header.



## 9. EBML Stream

An EBML Stream is a file that consists of one or more EBML Documents that are concatenated together. An occurrence of a EBML Header at the Root Level marks the beginning of an EBML Document.

## 10. EBML Versioning

An EBML Document handles 2 different versions: the version of the EBML Header and the version of the EBML Body. Both versions are meant to be backward compatible.

### 10.1. EBML Header Version

The version of the EBML Header is found in EBMLVersion. An EBML parser can read an EBML Header if it can read either the EBMLVersion version or a version equal or higher than the one found in EBMLReadVersion.

### 10.2. EBML Document Version

The version of the EBML Body is found in EBMLDocTypeVersion. A parser for the particular DocType format can read the EBML Document if it can read either the EBMLDocTypeVersion version of that format or a version equal or higher than the one found in EBMLDocTypeReadVersion.

## 11. Elements semantic

### 11.1. EBML Schema

An EBML Schema is a well-formed XML Document [W3C.REC-xml-20081126] that defines the properties, arrangement, and usage of EBML Elements that compose a specific EBML Document Type. The relationship of an EBML Schema to an EBML Document is analogous to the relationship of an XML Schema [W3C.REC-xmlschema-0-20041028] to an XML Document [W3C.REC-xml-20081126]. An EBML Schema MUST be clearly associated with one or more EBML Document Types. An EBML Document Type is identified by a string stored within the EBML Header in the DocType Element; for example matroska or webm (see Section 11.2.6). The DocType value for an EBML Document Type MUST be unique, persistent and described in the IANA Registry (see Section 17.2).

An EBML Schema MUST declare exactly one EBML Element at Root Level (referred to as the Root Element) that occurs exactly once within an EBML Document. The Void Element MAY also occur at Root Level but is not a Root Element (see Section 11.3.2).



The EBML Schema MUST document all Elements of the EBML Body. The EBML Schema does not document Global Elements that are defined by this document (namely the Void Element and the CRC-32 Element).

The EBML Schema MUST NOT use the Element ID "0x1A45DFA3" which is reserved for the EBML Header for resynchronization purpose.

An EBML Schema MAY constrain the use of EBML Header Elements (see Section 11.2) by adding or constraining that Element's "range" attribute. For example, an EBML Schema MAY constrain the EBMLMaxSizeLength to a maximum value of "8" or MAY constrain the EBMLVersion to only support a value of "1". If an EBML Schema adopts the EBML Header Element as-is, then it is not required to document that Element within the EBML Schema. If an EBML Schema constrains the range of an EBML Header Element, then that Element MUST be documented within an "<element>" node of the EBML Schema. This document provides an example of an EBML Schema, see Section 11.1.1.

#### 11.1.1. EBML Schema Example

```
<?xml version="1.0" encoding="utf-8"?>
<EBMLSchema xmlns="https://ietf.org/cellar/ebml"
  docType="files-in-ebml-demo" version="1">
  <!-- constraints to the range of two EBML Header Elements -->
  <element name="EBMLReadVersion" path="\EBML\EBMLReadVersion"
    id="0x42F7" minOccurs="1" maxOccurs="1" range="1" default="1"
    type="uinteger"/>
  <element name="EBMLMaxSizeLength"
    path="\EBML\EBMLMaxSizeLength" id="0x42F3" minOccurs="1"
    maxOccurs="1" range="8" default="8" type="uinteger"/>
  <!-- Root Element-->
  <element name="Files" path="\Files" id="0x1946696C"
    type="master">
    <documentation lang="en" purpose="definition">Container of data and
      attributes representing one or many files.</documentation>
  </element>
  <element name="File" path="\Files\File" id="0x6146"
    type="master" minOccurs="1">
    <documentation lang="en" purpose="definition">
      An attached file.
    </documentation>
  </element>
  <element name="FileName" path="\Files\File\FileName"
    id="0x614E" type="utf-8"
    minOccurs="1">
    <documentation lang="en" purpose="definition">
      Filename of the attached file.
    </documentation>
```



```

</element>
<element name="MimeType" path="\Files\File\MimeType"
  id="0x464D" type="string"
  minOccurs="1">
  <documentation lang="en" purpose="definition">
    MIME type of the file.
  </documentation>
</element>
<element name="ModificationTimestamp"
  path="\Files\File\ModificationTimestamp" id="0x4654"
  type="date" minOccurs="1">
  <documentation lang="en" purpose="definition">
    Modification timestamp of the file.
  </documentation>
</element>
<element name="Data" path="\Files\File\Data" id="0x4664"
  type="binary" minOccurs="1">
  <documentation lang="en" purpose="definition">
    The data of the file.
  </documentation>
</element>
</EBMLSchema>

```

#### 11.1.1.2. <EBMLSchema> Element

Within an EBML Schema, the XPath [W3C.REC-xpath-19991116] of "<EBMLSchema>" element is "/EBMLSchema".

As an XML Document, the EBML Schema MUST use "<EBMLSchema>" as the top level element. The "<EBMLSchema>" element can contain "<element>" sub-elements.

#### 11.1.1.3. <EBMLSchema> Attributes

Within an EBML Schema the "<EBMLSchema>" element uses the following attributes:

##### 11.1.1.3.1. docType

Within an EBML Schema, the XPath of "@docType" attribute is "/EBMLSchema/@docType".

The docType lists the official name of the EBML Document Type that is defined by the EBML Schema; for example, "<EBMLSchema docType='matroska'>".

The docType attribute is REQUIRED within the "<EBMLSchema>" Element.



#### 11.1.3.2. version

Within an EBML Schema, the XPath of "@version" attribute is "/EBMLSchema/@version".

The version lists a non-negative integer that specifies the version of the docType documented by the EBML Schema. Unlike XML Schemas, an EBML Schema documents all versions of a docType's definition rather than using separate EBML Schemas for each version of a docType. EBML Elements may be introduced and deprecated by using the minver and maxver attributes of "<element>".

The version attribute is REQUIRED within the "<EBMLSchema>" Element.

#### 11.1.3.3. ebml

Within an EBML Schema, the XPath of "@ebml" attribute is "/EBMLSchema/@ebml".

The ebml attribute is a positive integer that specifies the version of the EBML Header (see Section 11.2.2) used by the EBML Schema. If the attribute is omitted, the EBML Header version is 1.

#### 11.1.4. <element> Element

Within an EBML Schema, the XPath of "<element>" element is "/EBMLSchema/element".

Each "<element>" defines one EBML Element through the use of several attributes that are defined in Section 11.1.5. EBML Schemas MAY contain additional attributes to extend the semantics but MUST NOT conflict with the definitions of the "<element>" attributes defined within this document.

The "<element>" nodes contain a description of the meaning and use of the EBML Element stored within one or more "<documentation>" sub-elements, followed by optional "<implementation\_note>" sub-elements, followed by zero or one "<restriction>" sub-element, followed by optional "<extension>" sub-elements. All "<element>" nodes MUST be sub-elements of the "<EBMLSchema>".

#### 11.1.5. <element> Attributes

Within an EBML Schema the "<element>" uses the following attributes to define an EBML Element:



## 11.1.5.1. name

Within an EBML Schema, the XPath of "@name" attribute is  
"/EBMLSchema/element/@name".

The name provides the human-readable name of the EBML Element. The value of the name MUST be in the form of characters "A" to "Z", "a" to "z", "0" to "9", "-" and ".". The first character of the name MUST be in the form of an "A" to "Z", "a" to "z", "0" to "9" character.

The name attribute is REQUIRED.

## 11.1.5.2. path

Within an EBML Schema, the XPath of "@path" attribute is  
"/EBMLSchema/element/@path".

The path defines the allowed storage locations of the EBML Element within an EBML Document. This path MUST be defined with the full hierarchy of EBML Elements separated with a "\". The top EBML Element in the path hierarchy being the first in the value. The syntax of the path attribute is defined using this Augmented Backus-Naur Form (ABNF) [RFC5234] with the case sensitive update [RFC7405] notation:

The path attribute is REQUIRED.

EBMLFullPath	= EBMLParentPath EBMLElement
EBMLParentPath	= PathDelimiter [EBMLParents]
EBMLParents	= 0*IntermediatePathAtom EBMLLastParent
IntermediatePathAtom	= EBMLPathAtom / GlobalPlaceholder
EBMLLastParent	= EBMLPathAtom / GlobalPlaceholder
EBMLPathAtom	= [IsRecursive] EBMLAtomName PathDelimiter
EBMLElement	= [IsRecursive] EBMLAtomName
PathDelimiter	= "\"
IsRecursive	= "+"
EBMLAtomName	= ALPHA / DIGIT 0*EBMLNameChar
EBMLNameChar	= ALPHA / DIGIT / "-" / "."
GlobalPlaceholder	= "(" GlobalParentOccurence ")"
GlobalParentOccurence	= [PathMinOccurrence] "-" [PathMaxOccurrence]
PathMinOccurrence	= 1*DIGIT ; no upper limit
PathMaxOccurrence	= 1*DIGIT ; no upper limit



The "\*", "(" and ")" symbols are interpreted as defined in [RFC5234].

The EBMLAtomName of the EBMLElement part MUST be equal to the "@name" attribute of the EBML Schema. If the EBMLElement part contains an IsRecursive part, the EBML Element can occur within itself recursively (see Section 11.1.5.11).

The starting PathDelimiter of EBMLParentPath corresponds to the root of the EBML Document.

The "@path" value MUST be unique within the EBML Schema. The "@id" value corresponding to this "@path" MUST NOT be defined for use within another EBML Element with the same EBMLParentPath as this "@path".

A path with a GlobalPlaceholder as the EBMLLastParent defines a Global Element; see Section 11.3. If the element has no EBMLLastParent part or the EBMLLastParent part is not a GlobalPlaceholder then the Element is not a Global Element.

The GlobalParentOccurrence part is interpreted as the amount of valid EBMLPathAtom parts that can replace the GlobalPlaceholder in the path. PathMinOccurrence represents the minimum amount of EBMLPathAtom required to replace the GlobalPlaceholder. PathMaxOccurrence represents the maximum amount of EBMLPathAtom possible to replace the GlobalPlaceholder.

If PathMinOccurrence is not present then that GlobalParentOccurrence has a PathMinOccurrence value of 0. If PathMaxOccurrence is not present then there is no upper bound for the permitted amount of EBMLPathAtom possible to replace the GlobalPlaceholder. PathMaxOccurrence MUST NOT have the value 0 as it would mean no EBMLPathAtom can replace the GlobalPlaceholder and the EBMLFullPath would be the same without that GlobalPlaceholder part. PathMaxOccurrence MUST be bigger or equal to PathMinOccurrence.

For example in "\a\0-1\global", the Element path "\a\x\global" corresponds to an EBMLPathAtom occurrence of 1. The Element "\a\x\y\global" corresponds to an EBMLPathAtom occurrence of 2, etc. In those case "\a\x" or "\a\x\y" MUST be valid pathes to be able to contain the element "global".

Consider another EBML Path "\a\1-\global". There has to be at least one EBMLPathAtom between the "\a\" part and "global". So the "global" EBML Element cannot be found inside the "\a" EBML Element as it means the resulting path "\a\global" has no EBMLPathAtom between the "\a\" and "global". But the "global" EBML Element can be found inside the "\a\b" EBML Element as the resulting path "\a\b\global"



has one EBMLPathAtom between the "\a\" and "global". Or it can be found inside the "\a\b\c" EBML Element (two EBMLPathAtom), or inside the "\a\b\c\d" EBML Element (three EBMLPathAtom), etc.

Consider another EBML Path "\a\{0-1\}global". There has to be at most one EBMLPathAtom between the "\a\" part and "global". So the "global" EBML Element can be found inside the "\a" EBML Element (0 EBMLPathAtom replacing GlobalPlaceholder) or inside the "\a\b" EBML Element (one replacement EBMLPathAtom). But it cannot be found inside the "\a\b\c" EBML Element as the resulting path "\a\b\c\global" has two EBMLPathAtom between "\a\" and "global".

#### 11.1.5.3. id

Within an EBML Schema, the XPath of "@id" attribute is "/EBMLSchema/element/@id".

The Element ID encoded as a Variable Size Integer expressed in hexadecimal notation prefixed by a 0x that is read and stored in big-endian order. To reduce the risk of false positives while parsing EBML Streams, the Element IDs of the Root Element and Top-Level Elements SHOULD be at least 4 octets in length. Element IDs defined for use at Root Level or directly under the Root Level MAY use shorter octet lengths to facilitate padding and optimize edits to EBML Documents; for instance, the Void Element uses an Element ID with a one octet length to allow its usage in more writing and editing scenarios.

The Element ID of any Element found within an EBML Document MUST only match a single "@path" value of its corresponding EBML Schema, but a separate instance of that Element ID value defined by the EBML Schema MAY occur within a different "@path". If more than one Element is defined to use the same "@id" value, then the "@path" values of those Elements MUST NOT share the same EBMLParentPath. Elements MUST NOT be defined to use the same "@id" value if one of their common Parent Elements could be an Unknown-Size Element.

The id attribute is REQUIRED.

#### 11.1.5.4. minOccurs

Within an EBML Schema, the XPath of "@minOccurs" attribute is "/EBMLSchema/element/@minOccurs".

The minOccurs is a non-negative integer expressing the minimum permitted number of occurrences of this EBML Element within its Parent Element.



Each instance of the Parent Element MUST contain at least this many instances of this EBML Element. If the EBML Element has an empty EBMLParentPath then minOccurs refers to constraints on the occurrence of the EBML Element within the EBML Document. EBML Elements with minOccurs set to "1" that also have a default value (see Section 11.1.5.8) declared are not REQUIRED to be stored but are REQUIRED to be interpreted, see Section 11.1.18.

An EBML Element defined with a minOccurs value greater than zero is called a Mandatory EBML Element.

The minOccurs attribute is OPTIONAL. If the minOccurs attribute is not present then that EBML Element has a minOccurs value of 0.

The semantic meaning of minOccurs within an EBML Schema is analogous to the meaning of minOccurs within an XML Schema.

#### 11.1.5.5. maxOccurs

Within an EBML Schema, the XPath of "@maxOccurs" attribute is "/EBMLSchema/element/@maxOccurs".

The maxOccurs is a non-negative integer expressing the maximum permitted number of occurrences of this EBML Element within its Parent Element.

Each instance of the Parent Element MUST contain at most this many instances of this EBML Element, including the unwritten mandatory element with a default value, see Section 11.1.18. If the EBML Element has an empty EBMLParentPath then maxOccurs refers to constraints on the occurrence of the EBML Element within the EBML Document.

The maxOccurs attribute is OPTIONAL. If the maxOccurs attribute is not present then there is no upper bound for the permitted number of occurrences of this EBML Element within its Parent Element or within the EBML Document depending on whether the EBMLParentPath of the EBML Element is empty or not.

The semantic meaning of maxOccurs within an EBML Schema is analogous to the meaning of maxOccurs within an XML Schema, when it is not present it's similar to xml:maxOccurs="unbounded" in an XML Schema.

#### 11.1.5.6. range

Within an EBML Schema, the XPath of "@range" attribute is "/EBMLSchema/element/@range".



A numerical range for EBML Elements which are of numerical types (Unsigned Integer, Signed Integer, Float, and Date). If specified the value of the EBML Element MUST be within the defined range. See Section 11.1.5.6.1 for rules applied to expression of range values.

The range attribute is OPTIONAL. If the range attribute is not present then any value legal for the type attribute is valid.

#### 11.1.5.6.1. Expression of range

The range attribute MUST only be used with EBML Elements that are either signed integer, unsigned integer, float, or date. The expression defines the upper, lower, exact or excluded value of the EBML Element and optionally an upper boundary value combined with a lower boundary. The range expression may contain whitespace (using the ASCII 0x20 character) for readability but whitespace within a range expression MUST NOT convey meaning.

To set a fixed value for the range, the value is used as the attribute value. For example "1234" means the EBML element always has the value 1234. The value can be prefixed with "not" to indicate that the fixed value MUST NOT be used for that Element. For example "not 1234" means the Element can use all values of its type except 1234.

For an exclusive lower boundary the ">" sign is used and the ">=" sign is used for an inclusive lower boundary. For example ">3" meaning the Element value MUST be greater than 3 or ">=0x1p+0" meaning the Element value MUST be greater than or equal to the floating value 1.0, see Section 11.1.17.

For an exclusive upper boundary the "<" sign is used and the "<=" sign is used for an inclusive upper boundary. For example "<-2" meaning the Element value MUST be less than -2 or "<=10" meaning the Element value MUST be less than or equal to the 10.

The lower and upper bounds can be combined into an expression to form a closed boundary. The lower boundary coming first followed by the upper boundary, separated by a comma. For example ">3,<= 20" means the Element value MUST be greater than 3 and less than or equal to 20.

A special form of lower and upper bounds using the "-" separator is possible, meaning the Element value MUST be greater than or to the first value and MUST be less than or equal to the second value. For example "1-10" is equivalent to ">=1,<=10". If the upper boundary is negative, only the latter form MUST be used.



## 11.1.5.7. length

Within an EBML Schema, the XPath of "@length" attribute is  
"/EBMLSchema/element/@length".

A value to express the valid length of the Element Data as written measured in octets. The length provides a constraint in addition to the Length value of the definition of the corresponding EBML Element Type. This length MUST be expressed as either a non-negative integer or a range (see Section 11.1.5.6.1) that consists of only non-negative integers and valid operators.

The length attribute is OPTIONAL. If the length attribute is not present for that EBML Element then that EBML Element is only limited in length by the definition of the associated EBML Element Type.

## 11.1.5.8. default

Within an EBML Schema, the XPath of "@default" attribute is  
"/EBMLSchema/element/@default".

If an Element is mandatory (has a minOccurs value greater than zero) but not written within its Parent Element or stored as an Empty Element, then the EBML Reader of the EBML Document MUST semantically interpret the EBML Element as present with this specified default value for the EBML Element. An unwritten mandatory Element with a declared default value is semantically equivalent to that Element if written with the default value stored as the Element Data. EBML Elements that are Master Elements MUST NOT declare a default value. EBML Elements with a minOccurs value greater than 1 MUST NOT declare a default value.

The default attribute is OPTIONAL.

## 11.1.5.9. type

Within an EBML Schema, the XPath of "@type" attribute is  
"/EBMLSchema/element/@type".

The type MUST be set to one of the following values: "integer" (signed integer), "uinteger" (unsigned integer), "float", "string", "date", "utf-8", "master", or "binary". The content of each type is defined within Section 7.

The type attribute is REQUIRED.



## 11.1.5.10. unknownsizeallowed

Within an EBML Schema, the XPath of "@unknownsizeallowed" attribute is "/EBMLSchema/element/@unknownsizeallowed".

A boolean to express if an EBML Element is permitted to be Unknown-Sized Element (having all VINT\_DATA bits of Element Data Size set to 1). EBML Elements that are not Master Elements MUST NOT set unknownsizeallowed to true. An EBML Element that is defined with an unknownsizeallowed attribute set to 1 MUST also have the unknownsizeallowed attribute of its Parent Element set to 1.

An EBML Element with the unknownsizeallowed attribute set to 1 MUST NOT have its recursive attribute set to 1.

The unknownsizeallowed attribute is OPTIONAL. If the unknownsizeallowed attribute is not used then that EBML Element is not allowed to use an unknown Element Data Size.

## 11.1.5.11. recursive

Within an EBML Schema, the XPath of "@recursive" attribute is "/EBMLSchema/element/@recursive".

A boolean to express if an EBML Element is permitted to be stored recursively. In this case the EBML Element MAY be stored within another EBML Element that has the same Element ID. Which itself can be stored in an EBML Element that has the same Element ID, and so on. EBML Elements that are not Master Elements MUST NOT set recursive to true.

If the EBMLElement part of the "@path" contains an IsRecursive part then the recursive value MUST be true and false otherwise.

An EBML Element with the recursive attribute set to 1 MUST NOT have its unknownsizeallowed attribute set to 1.

The recursive attribute is OPTIONAL. If the recursive attribute is not present then the EBML Element MUST NOT be used recursively.

## 11.1.5.12. recurring

Within an EBML Schema, the XPath of "@recurring" attribute is "/EBMLSchema/element/@recurring".

A boolean to express if an EBML Element is defined as an Identically Recurring Element or not; see Section 11.1.16.



The recurring attribute is OPTIONAL. If the recurring attribute is not present then the EBML Element is not an Identically Recurring Element.

#### 11.1.5.13. minver

Within an EBML Schema, the XPath of "@minver" attribute is "/EBMLSchema/element/@minver".

The minver (minimum version) attribute stores a non-negative integer that represents the first version of the docType to support the EBML Element.

The minver attribute is OPTIONAL. If the minver attribute is not present, then the EBML Element has a minimum version of "1".

#### 11.1.5.14. maxver

Within an EBML Schema, the XPath of "@maxver" attribute is "/EBMLSchema/element/@maxver".

The maxver (maximum version) attribute stores a non-negative integer that represents the last or most recent version of the docType to support the element. maxver MUST be greater than or equal to minver.

The maxver attribute is OPTIONAL. If the maxver attribute is not present then the EBML Element has a maximum version equal to the value stored in the version attribute of "<EBMLSchema>".

#### 11.1.6. <documentation> Element

Within an EBML Schema, the XPath of "<documentation>" attribute is "/EBMLSchema/element/documentation".

The "<documentation>" element provides additional information about the EBML Element. Within the "<documentation>" element the following XHTML [W3C.SPSD-xhtml-basic-20180327] elements MAY be used: "<a>", "<br>", "<strong>".

#### 11.1.7. <documentation> Attributes

##### 11.1.7.1. lang

Within an EBML Schema, the XPath of "@lang" attribute is "/EBMLSchema/element/documentation/@lang".

A lang attribute which is set to the [RFC5646] value of the language of the element's documentation.



The lang attribute is OPTIONAL.

#### 11.1.7.2. purpose

Within an EBML Schema, the XPath of "@purpose" attribute is `"/EBMLSchema/element/documentation/@purpose"`.

A purpose attribute distinguishes the meaning of the documentation. Values for the "<documentation>" sub-element's purpose attribute MUST include one of the values listed in Table 8.

value of purpose attribute	definition
definition	A 'definition' is recommended for every defined EBML Element. This documentation explains the semantic meaning of the EBML Element.
rationale	An explanation about the reason or catalyst for the definition of the Element.
usage notes	Recommended practices or guideline for both reading, writing, or interpreting the Element.
references	Informational references to support the contextualization and understanding of the value of the Element.

Table 8: Definitions of the permitted values for the purpose attribute of the documentation Element.

The purpose attribute is REQUIRED.

#### 11.1.8. <implementation\_note> Element

Within an EBML Schema, the XPath of "<implementation\_note>" attribute is `"/EBMLSchema/element/implementation_note"`.

In some cases within an EBML Document Type, the attributes of the "<element>" element are not sufficient to clearly communicate how the defined EBML Element is intended to be implemented. For instance, one EBML Element might only be mandatory if another EBML Element is present, or as another example, the default value of an EBML Element might derive from a related Element's content. In these cases where the Element's definition is conditional or advanced implementation



notes are needed, one or many "<implementation\_note>" elements can be used to store that information. The "<implementation\_note>" refer to a specific attribute of the parent "<element>" as expressed by the "note\_attribute" attribute Section 11.1.9.1.

#### 11.1.9. <implementation\_note> Attributes

##### 11.1.9.1. note\_attribute

Within an EBML Schema, the XPath of "@note\_attribute" attribute is "/EBMLSchema/element/implementation\_note/@note\_attribute".

The note\_attribute attribute references which of the "<element>"'s attributes that the implementation\_note is in regards to. The note\_attribute attribute MUST be set to one of the following values (corresponding to that attribute of the parent "<element>"): "minOccurs", "maxOccurs", "range", "length", "default", "minver", or "maxver". The "<implementation\_note>" SHALL supersede the parent "<element>"'s attribute that is named in the "note\_attribute" attribute. An "<element>" SHALL NOT have more than one "<implementation\_note>" of the same "note\_attribute".

The note\_attribute attribute is REQUIRED.

##### 11.1.9.2. <implementation\_note> Example

The following fragment of an EBML Schema demonstrates how an "<implementation\_note>" is used. In this case an EBML Schema documents a list of items that are described with an optional cost. The Currency Element uses an "<implementation\_note>" to say that the Currency Element is REQUIRED if the Cost Element is set, otherwise not.



```
<element name="Items" path="\Items" id="0x4025" type="master"
  minOccurs="1" maxOccurs="1">
  <documentation lang="en" purpose="definition">
    A set of items.
  </documentation>
</element>
<element name="Item" path="\Items\Item" id="0x4026"
  type="master">
  <documentation lang="en" purpose="definition">
    An item.
  </documentation>
</element>
<element name="Cost" path="\Items\Item\Cost" id="0x4024"
  type="float" maxOccurs="1">
  <documentation lang="en" purpose="definition">
    The cost of the item, if any.
  </documentation>
</element>
<element name="Currency" path="\Items\Item\Currency" id="0x403F"
  type="string" maxOccurs="1">
  <documentation lang="en" purpose="definition">
    The currency of the item's cost.
  </documentation>
  <implementation_note note_attribute="minOccurs">
    Currency MUST be set (minOccurs=1) if the associated Item stores
    a Cost, else Currency MAY be unset (minOccurs=0).
  </implementation_note>
</element>
```

#### 11.1.10. <restriction> Element

Within an EBML Schema, the XPath of "<restriction>" attribute is  
"/EBMLSchema/element/restriction".

The "<restriction>" element provides information about restrictions  
to the allowable values for the EBML Element which are listed in  
"<enum>" elements.

#### 11.1.11. <enum> Element

Within an EBML Schema, the XPath of "<enum>" attribute is  
"/EBMLSchema/element/restriction/enum".

The "<enum>" element stores a list of values allowed for storage in  
the EBML Element. The values MUST match the type of the EBML Element  
(for example "<enum value='Yes'>" cannot be a valid value for a EBML  
Element that is defined as an unsigned integer). An "<enum>" element



MAY also store "<documentation>" elements to further describe the "<enum>".

#### 11.1.12. <enum> Attributes

##### 11.1.12.1. label

Within an EBML Schema, the XPath of "@label" attribute is "/EBMLSchema/element/restriction/enum/@label".

The label provides a concise expression for human consumption that describes what the value of the "<enum>" represents.

The label attribute is OPTIONAL.

##### 11.1.12.2. value

Within an EBML Schema, the XPath of "@value" attribute is "/EBMLSchema/element/restriction/enum/@value".

The value represents data that MAY be stored within the EBML Element.

The value attribute is REQUIRED.

#### 11.1.13. <extension> Element

Within an EBML Schema, the XPath of "<extension>" attribute is "/EBMLSchema/element/extension".

The "<extension>" element provides an unconstrained element to contain information about the associated EBML "<element>" which is undefined by this document but MAY be defined by the associated EBML Document Type. The "<extension>" element MUST contain a "type" attribute and also MAY contain any other attribute or sub-element as long as the EBML Schema remains as a well-formed XML Document. All "<extension>" elements MUST be sub-elements of the "<element>".

#### 11.1.14. <extension> Attributes

##### 11.1.14.1. type

Within an EBML Schema, the XPath of "@type" attribute is "/EBMLSchema/element/extension/@type".

The type attribute should reference a name or identifier of the project or authority associated with the contents of the "<extension>" element.



The type attribute is REQUIRED.

#### 11.1.15. XML Schema for EBML Schema

This following provides an XML Schema [W3C.REC-xmlschema-0-20041028] for facilitating verification of an EBML Schema to the definition described in Section 8.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="https://ietf.org/cellar/ebml"
  targetNamespace="https://ietf.org/cellar/ebml"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xhtml="http://www.w3.org/1999/xhtml"
  elementFormDefault="qualified" version="01">

  <!-- for HTML in comments -->
  <xs:import namespace="http://www.w3.org/1999/xhtml"
    schemaLocation="http://www.w3.org/MarkUp/SCHEMA/xhtml11.xsd"/>

  <xs:element name="EBMLSchema" type="EBMLSchemaType"/>

  <xs:complexType name="EBMLSchemaType">
    <xs:sequence>
      <xs:element name="element" type="elementType"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="docType" use="required"/>
    <xs:attribute name="version" use="required" type="xs:integer"/>
    <xs:attribute name="ebml" type="xs:positiveInteger"
      default="1"/>
  </xs:complexType>

  <xs:complexType name="elementType">
    <xs:sequence>
      <xs:element name="documentation" type="documentationType"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="implementation_note" type="noteType"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="restriction" type="restrictionType"
        minOccurs="0" maxOccurs="1"/>
      <xs:element name="extension" type="extensionType"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="[0-9A-Za-z.-] ([0-9A-Za-z.-]) *"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
```



```
</xs:simpleType>
</xs:attribute>
<xs:attribute name="path" use="required">
  <!-- <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9]*\[0-9]*()" />
    </xs:restriction>
  </xs:simpleType> -->
</xs:attribute>
<xs:attribute name="id" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="0x([0-9A-F][0-9A-F])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="minOccurs" default="0">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="maxOccurs" default="1">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="range"/>
<xs:attribute name="length"/>
<xs:attribute name="default"/>
<xs:attribute name="type" use="required">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="integer"/>
      <xs:enumeration value="uinteger"/>
      <xs:enumeration value="float"/>
      <xs:enumeration value="string"/>
      <xs:enumeration value="date"/>
      <xs:enumeration value="utf-8"/>
      <xs:enumeration value="master"/>
      <xs:enumeration value="binary"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="unknownsizeallowed" type="xs:boolean"
```



```
        default="false"/>
      <xs:attribute name="recursive" type="xs:boolean"
        default="false"/>
      <xs:attribute name="recurring" type="xs:boolean"
        default="false"/>
      <xs:attribute name="minver" default="1">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="maxver">
        <xs:simpleType>
          <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>

    <xs:complexType name="restrictionType">
      <xs:sequence>
        <xs:element name="enum" type="enumType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>

    <xs:complexType name="extensionType">
      <xs:sequence>
        <xs:any processContents="skip"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="type" use="required"/>
      <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="enumType">
      <xs:sequence>
        <xs:element name="documentation" type="documentationType"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="label"/>
      <xs:attribute name="value" use="required"/>
    </xs:complexType>

    <xs:complexType name="documentationType" mixed="true">
      <xs:sequence>
```



```

    <xs:element name="a"          type="xhtml:xhtml.a.type"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="br"         type="xhtml:xhtml.br.type"
      minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="strong" type="xhtml:xhtml.strong.type"
      minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="lang"/>
  <xs:attribute name="purpose" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="definition"/>
        <xs:enumeration value="rationale"/>
        <xs:enumeration value="references"/>
        <xs:enumeration value="usage notes"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

<xs:complexType name="noteType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="note_attribute" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="minOccurs"/>
            <xs:enumeration value="maxOccurs"/>
            <xs:enumeration value="range"/>
            <xs:enumeration value="length"/>
            <xs:enumeration value="default"/>
            <xs:enumeration value="minver"/>
            <xs:enumeration value="maxver"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:schema>

```

#### 11.1.16. Identically Recurring Elements

An Identically Recurring Element is an EBML Element that MAY occur within its Parent Element more than once but that each recurrence within that Parent Element MUST be identical both in storage and semantics. Identically Recurring Elements are permitted to be stored multiple times within the same Parent Element in order to increase



data resilience and optimize the use of EBML in transmission. For instance a pertinent Top-Level Element could be periodically resent within a data stream so that an EBML Reader which starts reading the stream from the middle could better interpret the contents. Identically Recurring Elements SHOULD include a CRC-32 Element as a Child Element; this is especially recommended when EBML is used for long-term storage or transmission. If a Parent Element contains more than one copy of an Identically Recurring Element which includes a CRC-32 Element as a Child Element then the first instance of the Identically Recurring Element with a valid CRC-32 value should be used for interpretation. If a Parent Element contains more than one copy of an Identically Recurring Element which does not contain a CRC-32 Element or if CRC-32 Elements are present but none are valid then the first instance of the Identically Recurring Element should be used for interpretation.

#### 11.1.17. Textual expression of floats

When a float value is represented textually in an EBML Schema, such as within a default or range value, the float values MUST be expressed as Hexadecimal Floating-Point Constants as defined in the C11 standard [ISO.9899.2011] (see section 6.4.4.2 on Floating Constants). Table 9 provides examples of expressions of float ranges.

as decimal	as Hexadecimal Floating-Point Constants
0.0	"0x0p+1"
0.0-1.0	"0x0p+1-0x1p+0"
1.0-256.0	"0x1p+0-0x1p+8"
0.857421875	"0x1.b7p-1"
-1.0--0.857421875	"-0x1p+0--0x1.b7p-1"

Table 9: Example of floating point values and ranges as decimal and as Hexadecimal Floating-Point Constants.

Within an expression of a float range, as in an integer range, the - (hyphen) character is the separator between the minimal and maximum value permitted by the range. Hexadecimal Floating-Point Constants also use a - (hyphen) when indicating a negative binary power. Within a float range, when a - (hyphen) is immediately preceded by a letter p, then the - (hyphen) is a part of the Hexadecimal Floating-



Point Constant which notes negative binary power. Within a float range, when a - (hyphen) is not immediately preceded by a letter p, then the - (hyphen) represents the separator between the minimal and maximum value permitted by the range.

#### 11.1.18. Note on the use of default attributes to define Mandatory EBML Elements

If a Mandatory EBML Element has a default value declared by an EBML Schema and the value of the EBML Element is equal to the declared default value then that EBML Element is not required to be present within the EBML Document if its Parent Element is present. In this case, the default value of the Mandatory EBML Element MUST be read by the EBML Reader although the EBML Element is not present within its Parent Element.

If a Mandatory EBML Element has no default value declared by an EBML Schema and its Parent Element is present then the EBML Element MUST be present as well. If a Mandatory EBML Element has a default value declared by an EBML Schema and its Parent Element is present and the value of the EBML Element is NOT equal to the declared default value then the EBML Element MUST be present.

Table 10 clarifies if a Mandatory EBML Element MUST be written, according to if the default value is declared, if the value of the EBML Element is equal to the declared default value, and if the Parent Element is used.

Is the default value declared?	Is the value equal to default?	Is the Parent Element present?	Then is storing the EBML Element REQUIRED?
Yes	Yes	Yes	No
Yes	Yes	No	No
Yes	No	Yes	Yes
Yes	No	No	No
No	n/a	Yes	Yes
No	n/a	No	No

Table 10: Demonstration of the conditional requirements of VINT Storage.



## 11.2. EBML Header Elements

This document contains definitions of all EBML Elements of the EBML Header.

### 11.2.1. EBML Element

name: EBML

path: "\EBML"

id: 0x1A45DFA3

minOccurs: 1

maxOccurs: 1

type: Master Element

description: Set the EBML characteristics of the data to follow. Each EBML Document has to start with this.

### 11.2.2. EBMLVersion Element

name: EBMLVersion

path: "\EBML\EBMLVersion"

id 0x4286

minOccurs: 1

maxOccurs: 1

range: not 0

default: 1

type: Unsigned Integer

description: The version of EBML specifications used to create the EBML Document. The version of EBML defined in this document is 1, so EBMLVersion SHOULD be 1.

### 11.2.3. EBMLReadVersion Element

name: EBMLReadVersion



path: "\EBML\EBMLReadVersion"

id: 0x42F7

minOccurs: 1

maxOccurs: 1

range: 1

default: 1

type: Unsigned Integer

description: The minimum EBML version an EBML Reader has to support to read this EBML Document. The EBMLReadVersion Element MUST be less than or equal to EBMLVersion.

#### 11.2.4. EBMLMaxIDLength Element

name: EBMLMaxIDLength

path: "\EBML\EBMLMaxIDLength"

id 0x42F2

minOccurs: 1

maxOccurs: 1

range: >=4

default: 4

type: Unsigned Integer

description: The EBMLMaxIDLength Element stores the maximum permitted length in octets of the Element IDs to be found within the EBML Body. An EBMLMaxIDLength Element value of four is RECOMMENDED, though larger values are allowed.

#### 11.2.5. EBMLMaxSizeLength Element

name: EBMLMaxSizeLength

path: "\EBML\EBMLMaxSizeLength"

id 0x42F3



minOccurs: 1

maxOccurs: 1

range: not 0

default: 8

type: Unsigned Integer

description: The EBMLMaxSizeLength Element stores the maximum permitted length in octets of the expressions of all Element Data Sizes to be found within the EBML Body. The EBMLMaxSizeLength Element documents an upper bound for the "length" of all Element Data Size expressions within the EBML Body and not an upper bound for the "value" of all Element Data Size expressions within the EBML Body. EBML Elements that have an Element Data Size expression which is larger in octets than what is expressed by EBMLMaxSizeLength Element are invalid.

#### 11.2.6. DocType Element

name: DocType

path: "\EBML\DocType"

id 0x4282

minOccurs: 1

maxOccurs: 1

length: >0

type: String

description: A string that describes and identifies the content of the EBML Body that follows this EBML Header.

#### 11.2.7. DocTypeVersion Element

name: DocTypeVersion

path: "\EBML\DocTypeVersion"

id 0x4287

minOccurs: 1



maxOccurs: 1

range: not 0

default: 1

type: Unsigned Integer

description: The version of DocType interpreter used to create the EBML Document.

#### 11.2.8. DocTypeReadVersion Element

name: DocTypeReadVersion

path: "\EBML\DocTypeReadVersion"

id 0x4285

minOccurs: 1

maxOccurs: 1

range: not 0

default: 1

type: Unsigned Integer

description: The minimum DocType version an EBML Reader has to support to read this EBML Document. The value of the DocTypeReadVersion Element MUST be less than or equal to the value of the DocTypeVersion Element.

#### 11.2.9. DocTypeExtension Element

name: DocTypeExtension

path: "\EBML\DocTypeExtension"

id 0x4281

minOccurs: 0

type: Master Element

description: A DocTypeExtension adds extra Elements to the main DocType+DocTypeVersion tuple it's attached to. An EBML Reader MAY



know these extra Elements and how to use them. A DocTypeExtension MAY be used to iterate between experimental Elements before they are integrated in a regular DocTypeVersion. Reading one DocTypeExtension version of a DocType+DocTypeVersion tuple doesn't imply one should be able to read upper versions of this DocTypeExtension.

#### 11.2.10. DocTypeExtensionName Element

name: DocTypeExtensionName

path: "\\EBML\\DocTypeExtension\\Name"

id 0x4283

minOccurs: 1

maxOccurs: 1

length: >0

type: String

description: The name of the DocTypeExtension to differentiate it from other DocTypeExtension of the same DocType+DocTypeVersion tuple. A DocTypeExtensionName value MUST be unique within the EBML Header.

#### 11.2.11. DocTypeExtensionVersion Element

name: DocTypeExtensionVersion

path: "\\EBML\\DocTypeExtension\\Version"

id 0x4284

minOccurs: 1

maxOccurs: 1

range: not 0

type: Unsigned Integer

description: The version of the DocTypeExtension. Different DocTypeExtensionVersion values of the same DocType+DocTypeVersion+DocTypeExtensionName tuple MAY contain completely different sets of extra Elements. An EBML Reader MAY support multiple versions of the same DocTypeExtension, only one or none.



### 11.3. Global Elements

EBML allows some special Elements to be found within more than one parent in an EBML Document or optionally at the Root Level of an EBML Body. These Elements are called Global Elements. There are 2 Global Elements that can be found in any EBML Document: the CRC-32 Element and the Void Element. An EBML Schema MAY add other Global Elements to the format it defines. These extra elements apply only to the EBML Body, not the EBML Header.

Global Elements are EBML Elements whose EBMLLastParent part of the path has a GlobalPlaceholder. Because it is the last Parent part of the path, a Global Element might also have an EBMLParentPath parts in its path. In this case the Global Element can only be found within this EBMLParentPath path, i.e. it's not fully "global".

A Global Element can be found in many Parent Elements, allowing the same number of occurrences in each Parent where this Element is found.

#### 11.3.1. CRC-32 Element

name: CRC-32

path: "(1-)\CRC-32"

id: 0xBF

minOccurs: 0

maxOccurs: 1

length: 4

type: Binary

description: The CRC-32 Element contains a 32-bit Cyclic Redundancy Check value of all the Element Data of the Parent Element as stored except for the CRC-32 Element itself. When the CRC-32 Element is present, the CRC-32 Element MUST be the first ordered EBML Element within its Parent Element for easier reading. All Top-Level Elements of an EBML Document that are Master Elements SHOULD include a CRC-32 Element as a Child Element. The CRC in use is the IEEE-CRC-32 algorithm as used in the [ISO.3309.1979] standard and in section 8.1.1.6.2 of [ITU.V42.1994], with initial value of 0xFFFFFFFF. The CRC value MUST be computed on a little endian bytestream and MUST use little endian storage.



### 11.3.2. Void Element

name: Void

path: "\\(-\\)Void"

id: 0xEC

minOccurs: 0

type: Binary

description: Used to void data or to avoid unexpected behaviors when using damaged data. The content is discarded. Also used to reserve space in a sub-element for later use.

## 12. Considerations for Reading EBML Data

The following scenarios describe events to consider when reading EBML Documents and the recommended design of an EBML Reader.

If a Master Element contains a CRC-32 Element that doesn't validate, then the EBML Reader MAY ignore all contained data except for Descendant Elements that contain their own valid CRC-32 Element.

In the following XML representation of a simple, hypothetical EBML fragment, a Master Element called CONTACT contains two Child Elements, NAME and ADDRESS. In this example, some data within the NAME Element had been altered, so that the CRC-32 of the NAME Element does not validate and thus any Ancestor Element with a CRC-32 would therefore also no longer validate. However, even though the CONTACT Element has a CRC-32 that does not validate (because of the changed data within the NAME Element), the CRC-32 of the ADDRESS Element does validate and thus the contents and semantics of the ADDRESS Element MAY be used.



```

<CONTACT>
  <CRC-32>c119a69b</CRC-32><!-- does not validate -->
  <NAME>
    <CRC-32>1f59ee2b</CRC-32><!-- does not validate -->
    <FIRST-NAME>invalid data</FIRST-NAME>
    <LAST-NAME>invalid data</LAST-NAME>
  </NAME>
  <ADDRESS>
    <CRC-32>df941cc9</CRC-32><!-- validates -->
    <STREET>valid data</STREET>
    <CITY>valid data</CITY>
  </ADDRESS>
</CONTACT>

```

If a Master Element contains more occurrences of a Child Master Element than permitted according to the maxOccurs and recurring attributes of the definition of that Element then the occurrences in addition to maxOccurs MAY be ignored.

If a Master Element contains more occurrences of a Child Element than permitted according to the maxOccurs attribute of the definition of that Element then all instances of that Element after the first maxOccur occurrences from the beginning of its Parent Element SHOULD be ignored.

### 13. Terminating Elements

Null Octets, which are octets with all bits set to zero, MAY follow the value of a String Element or UTF-8 Element to serve as a terminator. An EBML Writer MAY terminate a String Element or UTF-8 Element with Null Octets in order to overwrite a stored value with a new value of lesser length while maintaining the same Element Data Size (this can prevent the need to rewrite large portions of an EBML Document); otherwise the use of Null Octets within a String Element or UTF-8 Element is NOT RECOMMENDED. The Element Data of a UTF-8 Element MUST be a valid UTF-8 string up to whichever comes first: the end of the Element or the first occurring Null octet. Within the Element Data of a String or UTF-8 Element, any Null octet itself and any following data within that Element SHOULD be ignored. A string value and a copy of that string value terminated by one or more Null Octets are semantically equal.

Table 11 shows examples of semantics and validation for the use of Null Octets. Values to represent Stored Values and the Semantic Meaning as represented as hexadecimal values.



Stored Value	Semantic Meaning
0x65 0x62 0x6D 0x6C	0x65 0x62 0x6D 0x6C
0x65 0x62 0x00 0x6C	0x65 0x62
0x65 0x62 0x00 0x00	0x65 0x62
0x65 0x62	0x65 0x62

Table 11: Exmaples of semantics for Null Octets in VINT\_DATA.

#### 14. Guidelines for Updating Elements

An EBML Document can be updated without requiring that the entire EBML Document be rewritten. These recommendations describe strategies to change the Element Data of a written EBML Element with minimal disruption to the rest of the EBML Document.

##### 14.1. Reducing a Element Data in Size

There are three methods to reduce the size of Element Data of a written EBML Element.

###### 14.1.1. Adding a Void Element

When an EBML Element is changed to reduce its total length by more than one octet, an EBML Writer SHOULD fill the freed space with a Void Element.

###### 14.1.2. Extending the Element Data Size

The same value for Element Data Size MAY be written in various lengths, so for minor reductions of the Element Data, the Element Size MAY be written to a longer octet length to fill the freed space.

For example, the first row of Table 12 depicts a String Element that stores an Element ID (3 octets), Element Data Size (1 octet), and Element Data (4 octets). If the Element Data is changed to reduce the length by one octet and if the current length of the Element Data Size is less than its maximum permitted length, then the Element Data Size of that Element MAY be rewritten to increase its length by one octet. Thus before and after the change the EBML Element maintains the same length of 8 octets and data around the Element does not need to be moved.



Status	Element ID	Element Data Size	Element Data
Before edit	0x3B4040	0x84	0x65626D6C
After edit	0x3B4040	0x4003	0x6D6B76

Table 12: Example of editing a VINT to reduce VINT\_DATA length by one octet.

This method is RECOMMENDED when the Element Data is reduced by a single octet; for reductions by two or more octets it is RECOMMENDED to fill the freed space with a Void Element.

Note that if the Element Data length needs to be rewritten as shortened by one octet and the Element Data Size could be rewritten as a shorter VINT then it is RECOMMENDED to rewrite the Element Data Size as one octet shorter, shorten the Element Data by one octet, and follow that Element with a Void Element. For example, Table 13 depicts a String Element that stores an Element ID (3 octets), Element Data Size (2 octets, but could be rewritten in one octet), and Element Data (3 octets). If the Element Data is to be rewritten to a two octet length, then another octet can be taken from Element Data Size so that there is enough space to add a two octet Void Element.

Status	Element ID	Element Data Size	Element Data	Void Element
Before	0x3B4040	0x4003	0x6D6B76	
After	0x3B4040	0x82	0x6869	0xEC80

Table 13: Example of editing a VINT to reduce VINT\_DATA length by more than one octet.

#### 14.1.3. Terminating Element Data

For String Elements and UTF-8 Elements the length of Element Data could be reduced by adding Null Octets to terminate the Element Data (see Section 13).

In Table 14, a four octets long Element Data is changed to a three octet long value followed by a Null Octet; the Element Data Size



includes any Null Octets used to terminate Element Data so remains unchanged.

Status	Element ID	Element Data Size	Element Data
Before edit	0x3B4040	0x84	0x65626D6C
After edit	0x3B4040	0x84	0x6D6B7600

Table 14: Example of terminating VINT\_DATA with a Null Octet when reducing VINT length during an edit.

Note that this method is NOT RECOMMENDED. For reductions of one octet, the method for Extending the Element Data Size SHOULD be used. For reduction by more than one octet, the method for Adding a Void Element SHOULD be used.

#### 14.2. Considerations when Updating Elements with Cyclic Redundancy Check (CRC)

If the Element to be changed is a Descendant Element of any Master Element that contains a CRC-32 Element (see Section 11.3.1) then the CRC-32 Element MUST be verified before permitting the change. Additionally the CRC-32 Element value MUST be subsequently updated to reflect the changed data.

### 15. Backward and Forward Compatibility

Elements of an EBML format SHOULD be designed with backward and forward compatibility in mind.

#### 15.1. Backward Compatibility

Backward compatibility of new EBML Elements can be achieved by using default values for mandatory elements. The default value MUST represent the state that was assumed for previous versions of the EBML Schema, without this new EBML Element. If such a state doesn't make sense for previous versions, then the new EBML Element SHOULD NOT be mandatory.

Non mandatory EBML Elements can be added in a new EBMLDocTypeVersion. Since they are not mandatory they won't be found in older versions of the EBMLDocTypeVersion, just as they might not be found in newer versions. This causes no compatibility issue.



## 15.2. Forward Compatibility

EBML Elements MAY be marked as deprecated in a new EBMLDocTypeVersion using the maxver attribute of the EBML Schema. If such an Element is found in an EBML Document with newer version of the EBMLDocTypeVersion it SHOULD be discarded.

## 16. Security Considerations

EBML itself does not offer any kind of security and does not provide confidentiality. EBML does not provide any kind of authorization. EBML only offers marginally useful and effective data integrity options, such as CRC elements.

Even if the semantic layer offers any kind of encryption, EBML itself could leak information at both the semantic layer (as declared via the DocType Element) and within the EBML structure (the presence of EBML Elements can be derived even with an unknown semantic layer using a heuristic approach; not without errors, of course, but with a certain degree of confidence).

An EBML Document that has the following issues may still be handled by the EBML Reader and the data accepted as such, depending on how strict the EBML Reader wants to be:

- \* Invalid Element IDs that are longer than the limit stated in the EBMLMaxIDLength Element of the EBML Header.
- \* Invalid Element IDs that are not encoded in the shortest-possible way.
- \* Invalid Element Data Size values that are longer than the limit stated in the EBMLMaxSizeLength Element of the EBML Header.

Element IDs that are unknown to the EBML Reader MAY be accepted as valid EBML IDs in order to skip such elements.

EBML Elements with a string type may contain extra data after the first 0x00. These data MUST be discarded according to the Section 13 rules.

An EBML Reader may discard some or all data if the following errors are found in the EBML Document:

- \* Invalid Element Data Size values (e.g. extending the length of the EBML Element beyond the scope of the Parent Element; possibly triggering access-out-of-bounds issues).



- \* Very high lengths in order to force out-of-memory situations resulting in a denial of service, access-out-of-bounds issues etc.
- \* Missing EBML Elements that are mandatory in a Master Element and have no declared default value, making the semantic invalid at that Master Element level.
- \* Usage of invalid UTF-8 encoding in EBML Elements of UTF-8 type (e.g. in order to trigger access-out-of-bounds or buffer overflow issues).
- \* Usage of invalid data in EBML Elements with a date type, triggering bogus date accesses.
- \* The CRC-32 Element (see Section 11.3.1) of a Master Element doesn't match the rest of the content of that Master Element.

Side channel attacks could exploit:

- \* The semantic equivalence of the same string stored in a String Element or UTF-8 Element with and without zero-bit padding, making comparison at the semantic level invalid.
- \* The semantic equivalence of VINT\_DATA within Element Data Size with two different lengths due to left-padding zero bits, making comparison at the semantic level invalid.
- \* Data contained within a Master Element which is not itself part of a Child Element can trigger incorrect parsing behavior in EBML Readers.
- \* Extraneous copies of Identically Recurring Element, making parsing unnecessarily slow to the point of not being usable.
- \* Copies of Identically Recurring Element within a Parent Element that contain invalid CRC-32 Elements. EBML Readers not checking the CRC-32 might use the version of the element with mismatching CRC-32.
- \* Use of Void Elements which could be used to hide content or create bogus resynchronization points seen by some EBML Reader and not others.

## 17. IANA Considerations



### 17.1. EBML Element ID Registry

This document creates a new IANA Registry called "EBML Element ID Registry".

Element IDs are described in section Element ID. Element IDs are encoded using the VINT mechanism described in section Section 4 can be between one and five octets long. Five octet long Element IDs are possible only if declared in the header.

This IANA Registry only applies to Elements that can be contained in the EBML Header, thus including Global Elements. Elements only found in the EBML Body have their own set of independent Element IDs and are not part of this IANA Registry.

One-octet Element IDs MUST be between 0x81 and 0xFE. These items are valuable because they are short, and 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.

The one-octet range of 0x00 to 0x7F are not valid for use as an Element ID.

Two-octet Element IDs MUST be between 0x407F and 0x7FFE. Element IDs are to be allocated within this range according to the "Specification Required" policy [RFC8126].

The following two-octet Element IDs are RESERVED: 0x7FFF and 0x4000.

The two-octet ranges of 0x0000 to 0x3FFF and 0x8000 to 0xFFFF are not valid for use as an Element ID.

Three-octet Element IDs MUST be between 0x203FFF and 0x3FFFFE. Element IDs are to be allocated within this range according to the "First Come First Served" policy [RFC8126].

The following three-octet Element IDs are RESERVED: 0x3FFFFFF and 0x200000.

The three-octet ranges of 0x000000 to 0x1FFFFFF and 0x400000 to 0xFFFFFFF are not valid for use as an Element ID.

Four-octet Element IDs MUST be between 0x101FFFFFF and 0x1FFFFFFFE. Four-octet Element IDs are somewhat special in that they are useful for resynchronizing to major structures in the event of data corruption or loss. As such four-octet Element IDs are split into



two categories. Four-octet Element IDs whose lower three octets (as encoded) would make printable 7-bit ASCII values (0x20 to 0x7E, inclusive) MUST be allocated by the "Specification Required" policy. Sequential allocation of values is not required: specifications SHOULD include a specific request, and are encouraged to do early allocations.

To be clear about the above category: four-octet Element IDs always start with hex 0x10 to 0x1F, and that octet may be chosen so that the entire VINT has some desirable property, such as a specific CRC. The other three octets, when ALL having values between 0x20 (32, ASCII Space) and 0x7E (126, ASCII "~"), fall into this category.

Other four-octet Element IDs may be allocated by the "First Come First Served" policy.

The following four-octet Element IDs are RESERVED: 0x1FFFFFFFF and 0x10000000.

The four-octet ranges of 0x00000000 to 0x0FFFFFFF and 0x20000000 to 0xFFFFFFFF are not valid for use as an Element ID.

Five-octet Element IDs (values from 0x080FFFFFFF to 0x0FFFFFFFFF) are RESERVED according to the "Experimental Use" policy [RFC8126]: they may be used by anyone at any time, but there is no coordination.

ID Values found in this document are assigned as initial values as follows:



Element ID	Element Name	Reference
0x1A45DFA3	EBML	Described in Section 11.2.1
0x4286	EBMLVersion	Described in Section 11.2.2
0x42F7	EBMLReadVersion	Described in Section 11.2.3
0x42F2	EBMLMaxIDLength	Described in Section 11.2.4
0x42F3	EBMLMaxSizeLength	Described in Section 11.2.5
0x4282	DocType	Described in Section 11.2.6
0x4287	DocTypeVersion	Described in Section 11.2.7
0x4285	DocTypeReadVersion	Described in Section 11.2.8
0x4281	DocTypeExtension	Described in Section 11.2.9
0x4283	DocTypeExtensionName	Described in Section 11.2.10
0x4284	DocTypeExtensionVersion	Described in Section 11.2.11
0xBF	CRC-32	Described in Section 11.3.1
0xEC	Void	Described in Section 11.3.2

Table 15: IDs and Names for EBML Elements assigned by this document.



## 17.2. EBML DocType Registry

This document creates a new IANA Registry called "EBML DocType Registry".

To register a new DocType in this registry one needs a DocType name, a Description of the DocType, a Change Controller (IESG or email of registrant) and an optional Reference to a document describing the DocType.

DocType values are described in Section 11.1.3.1. DocTypes are ASCII strings, defined in Section 7.4, which label the official name of the EBML Document Type. The strings may be allocated according to the "First Come First Served" policy.

The use of ASCII corresponds to the types and code already in use, the value is not meant to be visible to the user.

DocType string values of "matroska" and "webm" are RESERVED to the IETF for future use. These can be assigned via the "IESG Approval" or "RFC Required" policies [RFC8126].

## 18. Normative References

- [W3C.REC-xmlschema-0-20041028]  
Fallside, D. and P. Walmsley, "XML Schema Part 0: Primer Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-0-20041028, 28 October 2004, <<http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [ITU.V42.1994]  
International Telecommunications Union, "Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion", 1994.
- [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>>.
- [RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <<https://www.rfc-editor.org/info/rfc20>>.



- [ISO.9899.2011]  
International Organization for Standardization,  
"Programming languages - C", 2011.
- [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>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF",  
RFC 7405, DOI 10.17487/RFC7405, December 2014,  
<<https://www.rfc-editor.org/info/rfc7405>>.
- [W3C.SPSPD-xhtml-basic-20180327]  
McCarron, S., "XHTML(tm) Basic 1.1 - Second Edition", 27  
March 2018.
- [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>>.
- [IEEE.754.1985]  
Institute of Electrical and Electronics Engineers,  
"Standard for Binary Floating-Point Arithmetic", August  
1985.
- [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>>.
- [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>>.
- [W3C.REC-xml-20081126]  
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and  
F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth  
Edition)", World Wide Web Consortium Recommendation REC-  
xml-20081126, 26 November 2008,  
<<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [ISO.3309.1979]  
International Organization for Standardization, "Data



communication - High-level data link control procedures -  
Frame structure", 1979.

#### 19. Informative References

- [Matroska] IETF, "Matroska Specifications", 2019,  
<<https://datatracker.ietf.org/doc/draft-ietf-cellar-matroska/>>.
- [WebM] The WebM Project, "WebM Container Guidelines", November  
2017, <<https://www.webmproject.org/docs/container/>>.
- [W3C.REC-xpath-19991116]  
Clark, J. and S. DeRose, "XML Path Language (XPath)  
Version 1.0", World Wide Web Consortium Recommendation  
REC-xpath-19991116, 16 November 1999,  
<<http://www.w3.org/TR/1999/REC-xpath-19991116>>.

#### Authors' Addresses

Steve Lhomme

Email: [slhomme@matroska.org](mailto:slhomme@matroska.org)

Dave Rice

Email: [dave@dericed.com](mailto:dave@dericed.com)

Moritz Bunkus

Email: [moritz@bunkus.org](mailto:moritz@bunkus.org)



cellar  
Internet-Draft  
Intended status: Informational  
Expires: 27 August 2021

M. Niedermayer  
D. Rice  
J. Martinez  
23 February 2021

FFV1 Video Coding Format Version 0, 1, and 3  
draft-ietf-cellar-ffv1-20

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 27 August 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.



This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
2. Notation and Conventions . . . . .	5
2.1. Definitions . . . . .	5
2.2. Conventions . . . . .	6
2.2.1. Pseudo-code . . . . .	6
2.2.2. Arithmetic Operators . . . . .	6
2.2.3. Assignment Operators . . . . .	7
2.2.4. Comparison Operators . . . . .	7
2.2.5. Mathematical Functions . . . . .	8
2.2.6. Order of Operation Precedence . . . . .	8
2.2.7. Range . . . . .	9
2.2.8. NumBytes . . . . .	9
2.2.9. Bitstream Functions . . . . .	9
3. Sample Coding . . . . .	10
3.1. Border . . . . .	10
3.2. Samples . . . . .	11
3.3. Median Predictor . . . . .	11
3.3.1. Exception . . . . .	12
3.4. Quantization Table Sets . . . . .	12
3.5. Context . . . . .	13
3.6. Quantization Table Set Indexes . . . . .	13
3.7. Color spaces . . . . .	13
3.7.1. YCbCr . . . . .	14
3.7.2. RGB . . . . .	14
3.8. Coding of the Sample Difference . . . . .	16
3.8.1. Range Coding Mode . . . . .	16
3.8.2. Golomb Rice Mode . . . . .	24
4. Bitstream . . . . .	30
4.1. Quantization Table Set . . . . .	31
4.1.1. quant_tables . . . . .	32
4.1.2. context_count . . . . .	33
4.2. Parameters . . . . .	33
4.2.1. version . . . . .	35
4.2.2. micro_version . . . . .	35
4.2.3. coder_type . . . . .	36
4.2.4. state_transition_delta . . . . .	36
4.2.5. colorspace_type . . . . .	37



4.2.6.	chroma_planes . . . . .	37
4.2.7.	bits_per_raw_sample . . . . .	38
4.2.8.	log2_h_chroma_subsample . . . . .	38
4.2.9.	log2_v_chroma_subsample . . . . .	38
4.2.10.	extra_plane . . . . .	38
4.2.11.	num_h_slices . . . . .	39
4.2.12.	num_v_slices . . . . .	39
4.2.13.	quant_table_set_count . . . . .	39
4.2.14.	states_coded . . . . .	39
4.2.15.	initial_state_delta . . . . .	39
4.2.16.	ec . . . . .	40
4.2.17.	intra . . . . .	40
4.3.	Configuration Record . . . . .	41
4.3.1.	reserved_for_future_use . . . . .	41
4.3.2.	configuration_record_crc_parity . . . . .	41
4.3.3.	Mapping FFV1 into Containers . . . . .	41
4.4.	Frame . . . . .	42
4.5.	Slice . . . . .	44
4.6.	Slice Header . . . . .	45
4.6.1.	slice_x . . . . .	46
4.6.2.	slice_y . . . . .	46
4.6.3.	slice_width . . . . .	46
4.6.4.	slice_height . . . . .	46
4.6.5.	quant_table_set_index_count . . . . .	46
4.6.6.	quant_table_set_index . . . . .	47
4.6.7.	picture_structure . . . . .	47
4.6.8.	sar_num . . . . .	47
4.6.9.	sar_den . . . . .	48
4.7.	Slice Content . . . . .	48
4.7.1.	primary_color_count . . . . .	48
4.7.2.	plane_pixel_height . . . . .	48
4.7.3.	slice_pixel_height . . . . .	49
4.7.4.	slice_pixel_y . . . . .	49
4.8.	Line . . . . .	49
4.8.1.	plane_pixel_width . . . . .	49
4.8.2.	slice_pixel_width . . . . .	50
4.8.3.	slice_pixel_x . . . . .	50
4.8.4.	sample_difference . . . . .	50
4.9.	Slice Footer . . . . .	50
4.9.1.	slice_size . . . . .	51
4.9.2.	error_status . . . . .	51
4.9.3.	slice_crc_parity . . . . .	51
5.	Restrictions . . . . .	51
6.	Security Considerations . . . . .	52
7.	IANA Considerations . . . . .	52
7.1.	Media Type Definition . . . . .	52
8.	Changelog . . . . .	54
9.	Normative References . . . . .	54



10. Informative References . . . . .	55
Appendix A. Multi-threaded decoder implementation suggestions . .	56
Appendix B. Future handling of some streams created by non conforming encoders . . . . .	57
Appendix C. FFV1 Implementations . . . . .	57
C.1. FFmpeg FFV1 Codec . . . . .	57
C.2. FFV1 Decoder in Go . . . . .	58
C.3. MediaConch . . . . .	58
Authors' Addresses . . . . .	58

## 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 version 0, 1 and 3 of FFV1. The distinctions of the versions are provided throughout the document, but in summary:

- \* Version 0 of FFV1 was the original implementation of FFV1 and has been flagged as stable on April 14, 2006 [FFV1\_V0].
- \* Version 1 of FFV1 adds support of more video bit depths and has been has been flagged as stable on April 24, 2009 [FFV1\_V1].
- \* Version 2 of FFV1 only existed in experimental form and is not described by this document, but is available as a LyX file at <https://github.com/FFmpeg/FFV1/blob/8ad772b6d61c3dd8b0171979a2cd9f11924d5532/ffv1.lyx> (<https://github.com/FFmpeg/FFV1/blob/8ad772b6d61c3dd8b0171979a2cd9f11924d5532/ffv1.lyx>).
- \* Version 3 of FFV1 adds several features such as increased description of the characteristics of the encoding images and embedded CRC data to support fixity verification of the encoding. Version 3 has been flagged as stable on August 17, 2013 [FFV1\_V3].

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



This specification describes the valid bitstream and how to decode such valid bitstream. Bitstreams not conforming to this specification or how they are handled is outside this specification. A decoder could reject every invalid bitstream or attempt to perform error concealment or re-download or use a redundant copy of the invalid part or any other action it deems appropriate.

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

"Container": 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). YCbCr word is used for historical reasons and currently references any color space relying on 1 luma Sample and 2 chroma Samples, e.g. YCbCr, YCgCo or ICtCp. The exact meaning of the three numeric values is unspecified.

## 2.2. Conventions

### 2.2.1. Pseudo-code

The FFV1 bitstream is described in this document using pseudo-code. Note that the pseudo-code is used for clarity in order to illustrate the structure of FFV1 and not intended to specify any particular implementation. The pseudo-code 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, pseudo-code 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 pseudo-code.

pseudo-code	type
-----	-----
ExamplePseudoCode( ) {	
value	ur
}	

Figure 1: A depiction of type-labelled pseudo-code 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 re-defined 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 bit-wise "and" of a and b.

"a | b" means bit-wise "or" of a and b.

"a >> b" means arithmetic right shift of 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 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.  $\text{"abs(a)" = "sign(a) * a"}$ .

"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 non-negative 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 RangeCoder 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 9 Samples "a,b,c,d,e,f,g,h,i" in a 3x3 arrangement along with its assumed border.



0	0		0	0	0		0
0	0		0	0	0		0
0	0		a	b	c		c
0	a		d	e	f		f
0	d		g	h	i		i

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

		T	
	tl	t	tr
L	l	X	

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:

$\text{median}(l, t, l + t - tl)$

Note, 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 = 1  >= 32768 ? ( 1  - 65536 ) : 1
top16s  = t  >= 32768 ? ( t  - 65536 ) : t
diag16s = t1 >= 32768 ? ( t1 - 65536 ) : t1
```

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 configuration of all known implementations being impacted is 16-bit YCbCr with no Pixel transformation with Range Coder coder, as other potentially impacted configurations (e.g. 15/16-bit JPEG2000-RCT with Range Coder coder, or 16-bit content with Golomb Rice coder) were implemented nowhere [ISO.15444-1.2016]. In the meanwhile, 16-bit JPEG2000-RCT with Range Coder coder was implemented without this issue in one implementation and validated by one conformance checker. It is expected (to be confirmed) to remove this exception for the median predictor 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 5 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 8 least significant bits of the Quantized Sample Difference are used as index:

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

Figure 4



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

### 3.5. Context

Relative to any Sample "X", the Quantized Sample Differences "L-1", "l-tl", "tl-t", "T-t", and "t-tr" are used as context:

$$\begin{aligned} \text{context} = & Q\_ (0) [l - tl] + \\ & Q\_ (1) [tl - t] + \\ & Q\_ (2) [t - tr] + \\ & Q\_ (3) [L - l] + \\ & Q\_ (4) [T - t] \end{aligned}$$

Figure 5

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 first implementations of FFV1 bitstream, the index for Cb and Cr Planes was stored even if it is 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 JPEG2000-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 1 to 4 Planes.

The Cb and Cr Planes are optional, but if used then MUST be used together. Omitting the Cb and Cr Planes codes the frames in grayscale 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 3 or 4 Planes.

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

JPEG2000-RCT is a Reversible Color Transform that codes RGB (red, green, blue) Planes losslessly in a modified YCbCr color space [ISO.15444-1.2016]. Reversible Pixel transformations between YCbCr and RGB use the following formulae.

$$\begin{aligned} \text{Cb} &= \text{b} - \text{g} \\ \text{Cr} &= \text{r} - \text{g} \\ \text{Y} &= \text{g} + (\text{Cb} + \text{Cr}) \gg 2 \end{aligned}$$

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 " $1 \ll \text{bits\_per\_raw\_sample}$ " after the conversion from RGB to the modified YCbCr and are negatively offset by the same value before the conversion from the modified YCbCr to RGB, in order to have only non-negative values after the conversion.

When FFV1 uses the JPEG2000-RCT, the horizontal Lines are interleaved to improve caching efficiency since it is most likely that the JPEG2000-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 JPEG2000-RCT, the coding order would be 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. Exception

If " $\text{bits\_per\_raw\_sample}$ " is between 9 and 15 inclusive and " $\text{extra\_plane}$ " 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 FFV1 bitstream, when "bits\_per\_raw\_sample" was between 9 and 15 inclusive and "extra\_plane" is 0, GBR Planes were used as BGR Planes during both encoding and decoding. In the meanwhile, 16-bit JPEG2000-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 JPEG2000-RCT), only the  $n$  (or  $n+1$ , in the case of JPEG2000-RCT) least significant bits are used, since this is sufficient to recover the original Sample. In the equation below, the term "bits" represents "bits\_per\_raw\_sample + 1" for JPEG2000-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 CABAC Arithmetic coder from H.264 as defined in [ISO.14496-10.2014] 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-coding].



### 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 2 sub-ranges 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. Coding integers is done by coding multiple binary values. The range decoder will read bytes until it can determine which sub-range the input falls into 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 sub-ranges of  $R_{(i)}$ : a sub-range of  $r_{(i)}$  values and a sub-range  $R_{(i)} - r_{(i)}$  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 sub-ranges 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 "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.

$$r_{-}(i) = \text{floor}((R_{-}(i) * S_{-}(i, C_{-}(i))) / 2^{8})$$

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

$b_{-}(i) = 0$	$\Leftarrow$
$L_{-}(i) < R_{-}(i) - r_{-}(i)$	$\Rightarrow$
$S_{-}(i + 1, C_{-}(i)) = \text{zero\_state\_}(S_{-}(i, C_{-}(i)))$	AND
$l_{-}(i) = L_{-}(i)$	AND
$t_{-}(i) = R_{-}(i) - r_{-}(i)$	
$b_{-}(i) = 1$	$\Leftarrow$
$L_{-}(i) \geq R_{-}(i) - r_{-}(i)$	$\Rightarrow$
$S_{-}(i + 1, C_{-}(i)) = \text{one\_state\_}(S_{-}(i, C_{-}(i)))$	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  $t_{-}(i)$ ).

$$C_{-}(i) \neq k \Rightarrow S_{-}(i + 1, k) = S_{-}(i, k)$$

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.



$t_{-}(i)$	$< 2^8$	$\Rightarrow$
$R_{-}(i + 1)$	$= 2^8 * t_{-}(i)$	AND
$L_{-}(i + 1)$	$= 2^8 * l_{-}(i) + B_{-}(j_{-}(i))$	AND
$j_{-}(i + 1)$	$= j_{-}(i) + 1$	
$t_{-}(i)$	$\geq 2^8$	$\Rightarrow$
$R_{-}(i + 1)$	$= t_{-}(i)$	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 pseudo-code 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 pseudo-code description of refilling the Range Binary Value coder buffer.



```
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 pseudo-code 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 non-corrupted and be of value 0.

Above describes the range decoding. Encoding is defined as any process which produces a decodable bytestream.



There are three places where range coder termination is needed in FFV1. First is in the "Configuration Record", in this case the size of the range coded bytestream is known and handled as "Closed mode". Second is the switch from the "Slice Header" which is range coded to Golomb coded slices as "Sentinel mode". 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 that is not only a waste of memory but also requires more past data to reach a reasonably good estimate of the probabilities. Alternatively assuming a Laplacian distribution and only dealing with its variance and mean (as in Huffman coding) would also be possible, however, for maximum flexibility and simplicity, the chosen method uses a single Symbol to encode if a number is 0, and if not, 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 pseudo-code description of the contexts of Range Non Binary 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 in Figure 14 as a formula and in Figure 20 as pseudo-code.

#### 3.8.1.3. Initial Values for the Context Model

When "keyframe" (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 mode a State Transition Table is used, indicating in 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

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

Figure 23

#### 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,

#### 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. The reference implementation of FFV1 in FFmpeg uses Figure 24 by default at the time of this writing 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 24: Alternative state transition table for Range coding.

### 3.8.2. Golomb Rice Mode

The end of the bitstream of the Frame is padded with 0-bits until the bitstream contains a multiple of 8 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 stores the most significant bits and the suffix 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 25: A pseudo-code 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 26: A pseudo-code 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

"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

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 non-0 difference is found. The sample difference is identical to the predicted one. The run and the first different sample difference 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 for each Plane and slice to 0.

```
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 twos 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:



```
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, this is different from JPEG-LS, 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 sub-sections, pseudo-code is used, as described in Section 2.2.1, to explain the structure of each FFV1 bitstream component. Table 4 lists symbols used to annotate that pseudo-code in order to define the storage of the data referenced in that line of pseudo-code.



Symbol	Definition
u(n)	unsigned big endian integer Symbol using n bits
sg	Golomb Rice coded signed scalar Symbol coded with the method described in Section 3.8.2
br	Range coded Boolean (1-bit) Symbol with the method described in Section 3.8.1.1
ur	Range coded unsigned scalar Symbol coded with the method described in Section 3.8.1.2
sr	Range coded signed scalar Symbol coded with the method described in Section 3.8.1.2
sd	Sample difference Symbol coded with the method described in Section 3.8

Table 4: Definition of pseudo-code symbols for this document.

The following MUST be provided by external means during 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 are stored by storing the number of equal entries -1 of the first half of the table (represented as "len - 1" in the pseudo-code below) using the method described in Section 3.8.1.2. The second half doesn't 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 pseudo-code.

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.

pseudo-code	type
<pre>QuantizationTableSet( i ) {     scale = 1     for (j = 0; j &lt; MAX_CONTEXT_INPUTS; j++) {         QuantizationTable( i, j, scale )         scale *= 2 * len_count[ i ][ j ] - 1     }     context_count[ i ] = ceil( scale / 2 ) }</pre>	

"MAX\_CONTEXT\_INPUTS" is 5.

pseudo-code	type
<pre>QuantizationTable(i, j, scale) {     v = 0     for (k = 0; k &lt; 128;) {         len = 1         for (n = 0; n &lt; len; n++) {             quant_tables[ i ][ j ][ k ] = scale * v             k++         }         v++     }     for (k = 1; k &lt; 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 }</pre>	ur

#### 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 27 describes the contents of the bitstream.

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



pseudo-code	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 27: A pseudo-code 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  $\leq 1$  && ConfigurationRecordIsPresent == 1.

Decoders SHOULD reject FFV1 bitstreams with version  $\geq 3$  && ConfigurationRecordIsPresent == 0.

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

Table 5

\* 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.

#### 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 7

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 Golomb Rice algorithm with "bits\_per\_raw\_sample" greater than 8, 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	JPEG2000-RCT	Transparency	Line then Plane
Other	reserved for future use	reserved for future use	reserved for future use	reserved for future use

Table 8

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 9



## 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 10

\* 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 11



## 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 12

## 4.2.15. initial\_state\_delta

"initial\_state\_delta[ i ][ j ][ k ]" indicates the initial Range coder state, it is encoded using "k" as context index and

pred = j ? initial\_states[ i ][ j - 1 ][ k ] : 128



Figure 28

```

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

```

Figure 29

## 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 13

## 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	"keyframe" MUST be 1 (keyframes only)
Other	reserved for future use

Table 14



### 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.

pseudo-code	type
ConfigurationRecord( NumBytes ) { ConfigurationRecordIsPresent = 1 Parameters( ) while (remaining_symbols_in_syntax(NumBytes - 4)) { reserved_for_future_use } configuration_record_crc_parity }	     br/ur/sr  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" 32 bits that are chosen so that the "Configuration Record" 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 described in Section 4.9.3.

### 4.3.3. Mapping FFV1 into Containers

This "Configuration Record" can be placed in any file format supporting "Configuration Records", fitting as much as possible with how the file format uses to store "Configuration Records". The "Configuration Record" storage place and "NumBytes" are currently defined and supported by this version of this specification for the following formats:



#### 4.3.3.1. 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.14496-12.2015] 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 [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"  $\leq 1$ ), and a sequence of independent slices. The pseudo-code below describes the contents of a Frame.

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

pseudo-code	type
<pre> Frame( NumBytes ) {     keyframe     if (keyframe &amp;&amp; !ConfigurationRecordIsPresent {         Parameters( )     }     while (remaining_bits_in_bitstream( NumBytes )) {         Slice( )     } } </pre>	br

Architecture overview of slices in a Frame:



+=====+		
+=====+		
	first slice header	
+-----+		
	first slice content	
+-----+		
	first slice footer	
+-----+		
	-----	
+-----+		
	second slice header	
+-----+		
	second slice content	
+-----+		
	second slice footer	
+-----+		
	-----	
+-----+		
	...	
+-----+		
	-----	
+-----+		
	last slice header	
+-----+		
	last slice content	
+-----+		
	last slice footer	
+-----+		

Table 15

#### 4.5. Slice

A "Slice" is an independent spatial sub-section of a Frame that is encoded separately from another region of the same Frame. The use of more than one "Slice" per Frame can be useful for taking advantage of the opportunities of multithreaded encoding and decoding.

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



pseudo-code	type
<pre> Slice( ) {     if (version &gt;= 3) {         SliceHeader( )     }     SliceContent( )     if (coder_type == 0) {         while (!byte_aligned()) {             padding         }     }     if (version &lt;= 1) {         while (remaining_bits_in_bitstream( NumBytes ) != 0) {             reserved         }     }     if (version &gt;= 3) {         SliceFooter( )     } } </pre>	<p>u(1)</p> <p>u(1)</p>

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

"reserved" specifies a bit without any significance in this revision of the specification and 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 pseudo-code below describes the contents of the "Slice Header".

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



pseudo-code	type
<pre> SliceHeader( ) {     slice_x     slice_y     slice_width - 1     slice_height - 1     for (i = 0; i &lt; quant_table_set_index_count; i++) {         quant_table_set_index[ i ]     }     picture_structure     sar_num     sar_den } </pre>	<pre> ur ur ur ur ur ur ur ur ur </pre>

#### 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:



```
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

#### 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 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 Sample aspect ratio is unknown.

If "sar\_num" is 0, decoders SHOULD ignore the encoded value and consider that "sar\_den" is 0.

## 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).

pseudo-code	type
<pre> SliceContent( ) {     if (colorspace_type == 0) {         for (p = 0; p &lt; primary_color_count; p++) {             for (y = 0; y &lt; plane_pixel_height[ p ]; y++) {                 Line( p, y )             }         }     } else if (colorspace_type == 1) {         for (y = 0; y &lt; slice_pixel_height; y++) {             for (p = 0; p &lt; primary_color_count; p++) {                 Line( p, y )             }         }     } } </pre>	

## 4.7.1. primary\_color\_count

"primary\_color\_count" is defined as:

$1 + (\text{chroma\_planes} ? 2 : 0) + (\text{extra\_plane} ? 1 : 0)$

## 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:



```

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:

```

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:

```

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 pseudo-code below describes the contents of the Line.

pseudo-code	type
<pre> Line( p, y ) {   if (colorspace_type == 0) {     for (x = 0; x &lt; plane_pixel_width[ p ]; x++) {       sample_difference[ p ][ y ][ x ]     }   } else if (colorspace_type == 1) {     for (x = 0; x &lt; slice_pixel_width; x++) {       sample_difference[ p ][ y ][ x ]     }   } } </pre>	<pre> sd sd </pre>

#### 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:



```

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:

```

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:

```

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 pseudo-code below describes the contents of the "Slice Footer".

Note: "Slice Footer" is always byte aligned.

pseudo-code	type
<pre> SliceFooter( ) {     slice_size     if (ec) {         error_status         slice_crc_parity     } } </pre>	<pre> u(24) u(8) u(32) </pre>



## 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 17

## 4.9.3. slice\_crc\_parity

"slice\_crc\_parity" 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, no slice overlapping).

For each Frame with "keyframe" value of 0, each slice MUST have the same value of "slice\_x", "slice\_y", "slice\_width", "slice\_height" as a slice in the previous Frame.

## 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 arithmetic types range. 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

The IANA is requested to register 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 with a frames 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 which 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 Niedermayer 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. Normative References

[ISO.15444-1.2016]

International Organization for Standardization,  
"Information technology -- JPEG 2000 image coding system:  
Core coding system", October 2016.

[ISO.9899.2018]

International Organization for Standardization,  
"Programming languages - C", ISO Standard 9899, 2018.

[Matroska] IETF, "Matroska", 2019, <<https://datatracker.ietf.org/doc/draft-ietf-cellar-matroska/>>.

[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>>.



- [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>>.
- [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>>.

## 10. Informative References

- [Address-Sanitizer] The Clang Team, "ASAN AddressSanitizer website", undated, <<https://clang.llvm.org/docs/AddressSanitizer.html>>.
- [AVI] Microsoft, "AVI RIFF File Reference", undated, <<https://msdn.microsoft.com/en-us/library/windows/desktop/dd318189%28v=vs.85%29.aspx>>.
- [FFV1GO] Buitenhuis, D., "FFV1 Decoder in Go", 2019, <<https://github.com/dwbuiten/go-ffv1>>.
- [FFV1\_V0] Niedermayer, M., "Commit to mark FFV1 version 0 as non-experimental", April 2006, <<https://git.videolan.org/?p=ffmpeg.git;a=commit;h=b548f2b91b701e1235608ac882ea6df915167c7e>>.
- [FFV1\_V1] Niedermayer, M., "Commit to release FFV1 version 1", April 2009, <<https://git.videolan.org/?p=ffmpeg.git;a=commit;h=68f8d33becbd73b4d0aa277f472a6e8e72ea6849>>.
- [FFV1\_V3] Niedermayer, M., "Commit to mark FFV1 version 3 as non-experimental", August 2013, <<https://git.videolan.org/?p=ffmpeg.git;a=commit;h=abe76b851c05eea8743f6c899cbe5f7409b0f301>>.
- [HuffyYUV] Rudiak-Gould, B., "HuffyYUV", December 2003, <<https://web.archive.org/web/20040402121343/http://cultact-server.novi.dk/kpo/huffyuv/huffyuv.html>>.



- [ISO.14495-1.1999]  
International Organization for Standardization,  
"Information technology -- Lossless and near-lossless  
compression of continuous-tone still images: Baseline",  
December 1999.
- [ISO.14496-10.2014]  
International Organization for Standardization,  
"Information technology -- Coding of audio-visual objects  
-- Part 10: Advanced Video Coding", September 2014.
- [ISO.14496-12.2015]  
International Organization for Standardization,  
"Information technology -- Coding of audio-visual objects  
-- Part 12: ISO base media file format", December 2015.
- [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-coding]  
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", undated, <<https://ffmpeg.org>>.
- [VALGRIND] Valgrind Developers, "Valgrind website", undated,  
<<https://valgrind.org/>>.
- [YCbCr] Wikipedia, "YCbCr", undated,  
<<https://en.wikipedia.org/w/index.php?title=YCbCr>>.

#### Appendix A. Multi-threaded 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 multi-threading 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 "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 slices boundaries. A decoder MAY fallback on sequential order e.g. in case of a corrupted "Frame" (frame size unknown, "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 non conforming 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 care about avoiding to add 40 bits of content after "SliceContent" if "version" == 0 or "version" == 1. Else 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 as well as clangs address sanitizer [Address-Sanitizer], which track 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 was [FFV1GO] written in Go by Derek Buitenhuis during the work to development 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 this document without the added exceptions.

### Authors' Addresses

Michael Niedermayer

Email: michael@niedermayer.cc

Dave Rice

Email: dave@dericed.com

Jerome Martinez

Email: jerome@mediaarea.net



cellar  
Internet-Draft  
Intended status: Standards Track  
Expires: July 7, 2018

S. Lhomme  
M. Bunkus  
D. Rice  
January 3, 2018

Matroska Specifications  
draft-lhomme-cellar-matroska-04

Abstract

This document defines the Matroska audiovisual container, including definitions of its structural elements, as well as its terminology, vocabulary, and application.

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 July 7, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Introduction . . . . .	9
2. Status of this document . . . . .	10
3. Security Considerations . . . . .	10
4. IANA Considerations . . . . .	10
5. Notation and Conventions . . . . .	10
6. Basis in EBML . . . . .	11
6.1. Added Constraints on EBML . . . . .	11
6.2. Matroska Design . . . . .	11
6.2.1. Language Codes . . . . .	11
6.2.2. Physical Types . . . . .	12
6.2.3. Block Structure . . . . .	12
6.2.4. Lacing . . . . .	14
7. Matroska Structure . . . . .	18
8. Matroska Schema . . . . .	26
8.1. Matroska Additions to Schema Element Attributes . . . . .	26
8.2. Matroska Schema . . . . .	27
8.2.1. EBMLMaxIDLength Element . . . . .	27
8.2.2. EBMLMaxSizeLength Element . . . . .	27
8.2.3. Segment Element . . . . .	28
8.2.4. SeekHead Element . . . . .	28
8.2.5. Seek Element . . . . .	28
8.2.6. SeekID Element . . . . .	29
8.2.7. SeekPosition Element . . . . .	29
8.2.8. Info Element . . . . .	30
8.2.9. SegmentUID Element . . . . .	30
8.2.10. SegmentFilename Element . . . . .	30
8.2.11. PrevUID Element . . . . .	31
8.2.12. PrevFilename Element . . . . .	31
8.2.13. NextUID Element . . . . .	32
8.2.14. NextFilename Element . . . . .	32
8.2.15. SegmentFamily Element . . . . .	33
8.2.16. ChapterTranslate Element . . . . .	33
8.2.17. ChapterTranslateEditionUID Element . . . . .	33
8.2.18. ChapterTranslateCodec Element . . . . .	34
8.2.19. ChapterTranslateID Element . . . . .	34
8.2.20. TimecodeScale Element . . . . .	35
8.2.21. Duration Element . . . . .	35
8.2.22. DateUTC Element . . . . .	35
8.2.23. Title Element . . . . .	36
8.2.24. MuxingApp Element . . . . .	36
8.2.25. WritingApp Element . . . . .	37
8.2.26. Cluster Element . . . . .	37
8.2.27. Timecode Element . . . . .	37
8.2.28. SilentTracks Element . . . . .	38
8.2.29. SilentTrackNumber Element . . . . .	38
8.2.30. Position Element . . . . .	38



8.2.31. PrevSize Element . . . . .	39
8.2.32. SimpleBlock Element . . . . .	39
8.2.33. BlockGroup Element . . . . .	40
8.2.34. Block Element . . . . .	40
8.2.35. BlockVirtual Element . . . . .	40
8.2.36. BlockAdditions Element . . . . .	41
8.2.37. BlockMore Element . . . . .	41
8.2.38. BlockAddID Element . . . . .	41
8.2.39. BlockAdditional Element . . . . .	42
8.2.40. BlockDuration Element . . . . .	42
8.2.41. ReferencePriority Element . . . . .	43
8.2.42. ReferenceBlock Element . . . . .	43
8.2.43. ReferenceVirtual Element . . . . .	44
8.2.44. CodecState Element . . . . .	44
8.2.45. DiscardPadding Element . . . . .	44
8.2.46. Slices Element . . . . .	45
8.2.47. TimeSlice Element . . . . .	45
8.2.48. LaceNumber Element . . . . .	45
8.2.49. FrameNumber Element . . . . .	46
8.2.50. BlockAdditionID Element . . . . .	46
8.2.51. Delay Element . . . . .	47
8.2.52. SliceDuration Element . . . . .	47
8.2.53. ReferenceFrame Element . . . . .	48
8.2.54. ReferenceOffset Element . . . . .	48
8.2.55. ReferenceTimeCode Element . . . . .	48
8.2.56. EncryptedBlock Element . . . . .	49
8.2.57. Tracks Element . . . . .	49
8.2.58. TrackEntry Element . . . . .	50
8.2.59. TrackNumber Element . . . . .	50
8.2.60. TrackUID Element . . . . .	50
8.2.61. TrackType Element . . . . .	51
8.2.62. FlagEnabled Element . . . . .	51
8.2.63. FlagDefault Element . . . . .	52
8.2.64. FlagForced Element . . . . .	52
8.2.65. FlagLacing Element . . . . .	53
8.2.66. MinCache Element . . . . .	53
8.2.67. MaxCache Element . . . . .	54
8.2.68. DefaultDuration Element . . . . .	54
8.2.69. DefaultDecodedFieldDuration Element . . . . .	54
8.2.70. TrackTimecodeScale Element . . . . .	55
8.2.71. TrackOffset Element . . . . .	55
8.2.72. MaxBlockAdditionID Element . . . . .	56
8.2.73. Name Element . . . . .	56
8.2.74. Language Element . . . . .	57
8.2.75. LanguageIETF Element . . . . .	57
8.2.76. CodecID Element . . . . .	57
8.2.77. CodecPrivate Element . . . . .	58
8.2.78. CodecName Element . . . . .	58



8.2.79. AttachmentLink Element . . . . .	58
8.2.80. CodecSettings Element . . . . .	59
8.2.81. CodecInfoURL Element . . . . .	59
8.2.82. CodecDownloadURL Element . . . . .	60
8.2.83. CodecDecodeAll Element . . . . .	60
8.2.84. TrackOverlay Element . . . . .	60
8.2.85. CodecDelay Element . . . . .	61
8.2.86. SeekPreRoll Element . . . . .	61
8.2.87. TrackTranslate Element . . . . .	62
8.2.88. TrackTranslateEditionUID Element . . . . .	62
8.2.89. TrackTranslateCodec Element . . . . .	62
8.2.90. TrackTranslateTrackID Element . . . . .	63
8.2.91. Video Element . . . . .	63
8.2.92. FlagInterlaced Element . . . . .	63
8.2.93. FieldOrder Element . . . . .	64
8.2.94. StereoMode Element . . . . .	64
8.2.95. AlphaMode Element . . . . .	65
8.2.96. OldStereoMode Element . . . . .	65
8.2.97. PixelWidth Element . . . . .	66
8.2.98. PixelHeight Element . . . . .	66
8.2.99. PixelCropBottom Element . . . . .	66
8.2.100. PixelCropTop Element . . . . .	67
8.2.101. PixelCropLeft Element . . . . .	67
8.2.102. PixelCropRight Element . . . . .	68
8.2.103. DisplayWidth Element . . . . .	68
8.2.104. DisplayHeight Element . . . . .	68
8.2.105. DisplayUnit Element . . . . .	69
8.2.106. AspectRatioType Element . . . . .	69
8.2.107. ColourSpace Element . . . . .	70
8.2.108. GammaValue Element . . . . .	70
8.2.109. FrameRate Element . . . . .	71
8.2.110. Colour Element . . . . .	71
8.2.111. MatrixCoefficients Element . . . . .	71
8.2.112. BitsPerChannel Element . . . . .	72
8.2.113. ChromaSubsamplingHorz Element . . . . .	72
8.2.114. ChromaSubsamplingVert Element . . . . .	73
8.2.115. CbSubsamplingHorz Element . . . . .	73
8.2.116. CbSubsamplingVert Element . . . . .	73
8.2.117. ChromaSitingHorz Element . . . . .	74
8.2.118. ChromaSitingVert Element . . . . .	74
8.2.119. Range Element . . . . .	75
8.2.120. TransferCharacteristics Element . . . . .	75
8.2.121. Primaries Element . . . . .	75
8.2.122. MaxCLL Element . . . . .	76
8.2.123. MaxFALL Element . . . . .	76
8.2.124. MasteringMetadata Element . . . . .	77
8.2.125. PrimaryRChromaticityX Element . . . . .	77
8.2.126. PrimaryRChromaticityY Element . . . . .	77



8.2.127. PrimaryGChromaticityX Element . . . . .	78
8.2.128. PrimaryGChromaticityY Element . . . . .	78
8.2.129. PrimaryBChromaticityX Element . . . . .	79
8.2.130. PrimaryBChromaticityY Element . . . . .	79
8.2.131. WhitePointChromaticityX Element . . . . .	79
8.2.132. WhitePointChromaticityY Element . . . . .	80
8.2.133. LuminanceMax Element . . . . .	80
8.2.134. LuminanceMin Element . . . . .	81
8.2.135. Projection Element . . . . .	81
8.2.136. ProjectionType Element . . . . .	81
8.2.137. ProjectionPrivate Element . . . . .	82
8.2.138. ProjectionPoseYaw Element . . . . .	82
8.2.139. ProjectionPosePitch Element . . . . .	83
8.2.140. ProjectionPoseRoll Element . . . . .	84
8.2.141. Audio Element . . . . .	84
8.2.142. SamplingFrequency Element . . . . .	84
8.2.143. OutputSamplingFrequency Element . . . . .	85
8.2.144. Channels Element . . . . .	85
8.2.145. ChannelPositions Element . . . . .	86
8.2.146. BitDepth Element . . . . .	86
8.2.147. TrackOperation Element . . . . .	87
8.2.148. TrackCombinePlanes Element . . . . .	87
8.2.149. TrackPlane Element . . . . .	87
8.2.150. TrackPlaneUID Element . . . . .	88
8.2.151. TrackPlaneType Element . . . . .	88
8.2.152. TrackJoinBlocks Element . . . . .	89
8.2.153. TrackJoinUID Element . . . . .	89
8.2.154. TrickTrackUID Element . . . . .	89
8.2.155. TrickTrackSegmentUID Element . . . . .	90
8.2.156. TrickTrackFlag Element . . . . .	90
8.2.157. TrickMasterTrackUID Element . . . . .	91
8.2.158. TrickMasterTrackSegmentUID Element . . . . .	91
8.2.159. ContentEncodings Element . . . . .	91
8.2.160. ContentEncoding Element . . . . .	92
8.2.161. ContentEncodingOrder Element . . . . .	92
8.2.162. ContentEncodingScope Element . . . . .	93
8.2.163. ContentEncodingType Element . . . . .	93
8.2.164. ContentCompression Element . . . . .	94
8.2.165. ContentCompAlgo Element . . . . .	94
8.2.166. ContentCompSettings Element . . . . .	95
8.2.167. ContentEncryption Element . . . . .	95
8.2.168. ContentEncAlgo Element . . . . .	95
8.2.169. ContentEncKeyID Element . . . . .	96
8.2.170. ContentSignature Element . . . . .	96
8.2.171. ContentSigKeyID Element . . . . .	97
8.2.172. ContentSigAlgo Element . . . . .	97
8.2.173. ContentSigHashAlgo Element . . . . .	97
8.2.174. Cues Element . . . . .	98



8.2.175. CuePoint Element . . . . .	98
8.2.176. CueTime Element . . . . .	99
8.2.177. CueTrackPositions Element . . . . .	99
8.2.178. CueTrack Element . . . . .	99
8.2.179. CueClusterPosition Element . . . . .	100
8.2.180. CueRelativePosition Element . . . . .	100
8.2.181. CueDuration Element . . . . .	101
8.2.182. CueBlockNumber Element . . . . .	101
8.2.183. CueCodecState Element . . . . .	101
8.2.184. CueReference Element . . . . .	102
8.2.185. CueRefTime Element . . . . .	102
8.2.186. CueRefCluster Element . . . . .	102
8.2.187. CueRefNumber Element . . . . .	103
8.2.188. CueRefCodecState Element . . . . .	103
8.2.189. Attachments Element . . . . .	104
8.2.190. AttachedFile Element . . . . .	104
8.2.191. FileDescription Element . . . . .	105
8.2.192. FileName Element . . . . .	105
8.2.193. FileMimeType Element . . . . .	105
8.2.194. FileData Element . . . . .	106
8.2.195. FileUID Element . . . . .	106
8.2.196. FileReferral Element . . . . .	106
8.2.197. FileUsedStartTime Element . . . . .	107
8.2.198. FileUsedEndTime Element . . . . .	107
8.2.199. Chapters Element . . . . .	108
8.2.200. EditionEntry Element . . . . .	108
8.2.201. EditionUID Element . . . . .	108
8.2.202. EditionFlagHidden Element . . . . .	109
8.2.203. EditionFlagDefault Element . . . . .	109
8.2.204. EditionFlagOrdered Element . . . . .	110
8.2.205. ChapterAtom Element . . . . .	110
8.2.206. ChapterUID Element . . . . .	110
8.2.207. ChapterStringUID Element . . . . .	111
8.2.208. ChapterTimeStart Element . . . . .	111
8.2.209. ChapterTimeEnd Element . . . . .	112
8.2.210. ChapterFlagHidden Element . . . . .	112
8.2.211. ChapterFlagEnabled Element . . . . .	113
8.2.212. ChapterSegmentUID Element . . . . .	113
8.2.213. ChapterSegmentEditionUID Element . . . . .	114
8.2.214. ChapterPhysicalEquiv Element . . . . .	114
8.2.215. ChapterTrack Element . . . . .	114
8.2.216. ChapterTrackNumber Element . . . . .	115
8.2.217. ChapterDisplay Element . . . . .	115
8.2.218. ChapString Element . . . . .	116
8.2.219. ChapLanguage Element . . . . .	116
8.2.220. ChapLanguageIETF Element . . . . .	116
8.2.221. ChapCountry Element . . . . .	117
8.2.222. ChapProcess Element . . . . .	117



8.2.223.	ChapProcessCodecID Element	118
8.2.224.	ChapProcessPrivate Element	118
8.2.225.	ChapProcessCommand Element	118
8.2.226.	ChapProcessTime Element	119
8.2.227.	ChapProcessData Element	119
8.2.228.	Tags Element	120
8.2.229.	Tag Element	120
8.2.230.	Targets Element	120
8.2.231.	TargetTypeValue Element	121
8.2.232.	TargetType Element	121
8.2.233.	TagTrackUID Element	122
8.2.234.	TagEditionUID Element	122
8.2.235.	TagChapterUID Element	122
8.2.236.	TagAttachmentUID Element	123
8.2.237.	SimpleTag Element	123
8.2.238.	TagName Element	123
8.2.239.	TagLanguage Element	124
8.2.240.	TagLanguageIETF Element	124
8.2.241.	TagDefault Element	125
8.2.242.	TagString Element	125
8.2.243.	TagBinary Element	126
9.	Matroska Element Ordering Guidelines	126
9.1.	Top-Level Elements	126
9.2.	CRC-32	127
9.3.	SeekHead	127
9.4.	Cues (index)	127
9.5.	Info	127
9.6.	Chapters	128
9.7.	Attachments	128
9.8.	Tags	128
9.9.	Optimum layout from a muxer	128
9.10.	Optimum layout after editing tags	129
9.11.	Optimum layout with Cues at the front	129
9.12.	Cluster Timecode	129
10.	Chapters	130
10.1.	Edition and Chapter Flags	130
10.1.1.	Chapter Flags	130
10.1.2.	Edition Flags	130
10.2.	Menu features	132
10.2.1.	Matroska Script (0)	132
10.2.2.	DVD menu (1)	133
10.3.	Example 1 : basic chaptering	134
10.4.	Example 2 : nested chapters	136
10.4.1.	The Micronauts "Bleep To Bleep"	136
11.	Attachments	139
11.1.	Introduction	139
11.2.	Cover Art	139
12.	Cues	140



12.1.	Introduction . . . . .	140
12.2.	Recommendations . . . . .	140
13.	Matroska Streaming . . . . .	141
13.1.	File Access . . . . .	141
13.2.	Livestreaming . . . . .	142
14.	Menu Specifications . . . . .	142
14.1.	Introduction . . . . .	142
14.2.	Requirements . . . . .	143
14.2.1.	Highlights/Hotspots . . . . .	143
14.2.2.	Playback features . . . . .	144
14.2.3.	Player requirements . . . . .	144
14.3.	Working Graph . . . . .	144
14.4.	Ideas . . . . .	145
14.5.	Data Structure . . . . .	145
15.	Unknown elements . . . . .	145
16.	Default Values . . . . .	145
17.	DefaultDecodedFieldDuration . . . . .	145
18.	Encryption . . . . .	146
19.	Image cropping . . . . .	147
20.	Matroska versioning . . . . .	147
21.	MIME Types . . . . .	148
22.	Segment Position . . . . .	148
22.1.	Segment Position Exception . . . . .	148
22.2.	Example of Segment Position . . . . .	148
23.	Linked Segments . . . . .	149
23.1.	Hard Linking . . . . .	149
23.2.	Medium Linking . . . . .	150
23.3.	Soft Linking . . . . .	150
24.	Track Flags . . . . .	151
24.1.	Default flag . . . . .	151
24.2.	Forced flag . . . . .	151
24.3.	Track Operation . . . . .	151
24.4.	Overlay Track . . . . .	152
24.5.	Multi-planar and 3D videos . . . . .	152
25.	Timecodes . . . . .	153
25.1.	Timecode Types . . . . .	153
25.2.	Block Timecodes . . . . .	153
25.3.	Raw Timecode . . . . .	153
25.4.	TimecodeScale . . . . .	153
25.5.	TimecodeScale Rounding . . . . .	154
25.6.	TrackTimecodeScale . . . . .	156
26.1.	URIs . . . . .	157
	Authors' Addresses . . . . .	158



## 1. Introduction

Matroska aims to become THE standard of multimedia container formats. It was derived from a project called MCF [1], but differentiates from it significantly because it is based on EBML [2] (Extensible Binary Meta Language), a binary derivative of XML. EBML enables significant advantages in terms of future format extensibility, without breaking file support in old parsers.

First, it is essential to clarify exactly "What an Audio/Video container is", to avoid any misunderstandings:

- o It is NOT a video or audio compression format (codec)
- o It is an envelope for 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 like:

- o Fast seeking in the file
- o Chapter entries
- o Full metadata (tags) support
- o Selectable subtitle/audio/video streams
- o Modularly expandable
- o Error resilience (can recover playback even when the stream is damaged)
- o Streamable over the internet and local networks (HTTP, CIFS, FTP, etc)
- o Menus (like DVDs have)

Matroska is an open standards project. This means for personal use it is absolutely free to use and that the technical specifications describing the bitstream are open to everybody, even to companies that would like to support it in their products.



## 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 [3]. But since it's quite complete it is used as a reference for the development of libmatroska. Legacy versions of the specification can be found here [4] (PDF doc by Alexander Noe -- outdated).

For a simplified diagram of the layout of a Matroska file, see the Diagram page [5].

A more refined and detailed version of the EBML specifications is being worked on here [6].

The table found below is now generated from the "source" of the Matroska specification. This XML file [7] is also used to generate the semantic data used in libmatroska and libmatroska2. We encourage anyone to use and monitor its changes so your code is spec-proof and always up to date.

Note that versions 1, 2 and 3 have been finalized. Version 4 is currently work in progress. There MAY be further additions to v4.

## 3. Security Considerations

Matroska inherits security considerations from EBML.

Attacks on a "Matroska Reader" could include:

- o Storage of a 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.
- o A "Matroska Attachment" with an inaccurate mime-type.

## 4. IANA Considerations

To be determined.

## 5. Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [8].

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 "Matroska Reader" is a data parser that interprets the semantics of a Matroska document and creates a way for programs to use "Matroska".

"Matroska Player": A "Matroska Player" is a "Matroska Reader" with a primary purpose of playing audiovisual files, including "Matroska" documents.

## 6. Basis in EBML

Matroska is a Document Type of EBML (Extensible Binary Meta Language). This specification is dependent on the EBML Specification [9]. For an understanding of Matroska's EBML Schema, see in particular the sections of the EBML Specification covering EBML Element Types [10], EBML Schema [11], and EBML Structure [12].

### 6.1. Added Constraints on EBML

As an EBML Document Type, Matroska adds the following constraints to the EBML specification.

- o The "docType" of the "EBML Header" MUST be 'matroska'.
- o The "EBMLMaxIDLength" of the "EBML Header" MUST be "4".
- o The "EBMLMaxSizeLength" of the "EBML Header" MUST be between "1" and "8" inclusive.

### 6.2. Matroska Design

All top-levels elements (Segment and direct sub-elements) are coded on 4 octets, i.e. class D elements.

#### 6.2.1. Language Codes

Matroska from version 1 through 3 uses language codes that can be either the 3 letters bibliographic ISO-639-2 [13] form (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 "ISO 639-2" or BCP 47 [14] MAY be used, although "BCP 47" is RECOMMENDED. The "BCP 47 Language Elements" are "LanguageIETF Element", "TagLanguageIETF Element", and



"ChapLanguageIETF 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".

Country codes are the same as used for internet domains [15].

### 6.2.2. Physical Types

Each level can have different meanings for audio and video. The ORIGINAL\_MEDIUM tag can be used to specify a string for ChapterPhysicalEquiv = 60. Here is the list of possible levels for both audio and video :

ChapterPhysicalEquiv	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

### 6.2.3. Block Structure

Size = 1 + (1-8) + 4 + (4 + (4)) octets. So from 6 to 21 octets.

Bit 0 is the most significant bit.



Frames using references SHOULD be stored in "coding order". That means the references first and then the frames referencing them. A consequence is that timecodes MAY NOT be consecutive. But a frame with a past timecode MUST reference a frame already known, otherwise it's considered bad/void.

There can be many Blocks in a BlockGroup provided they all have the same timecode. It is used with different parts of a frame with different priorities.

#### 6.2.3.1. Block Header

Offset	Player	Description
0x00+	MUST	Track Number (Track Entry). It is coded in EBML like form (1 octet if the value is < 0x80, 2 if < 0x4000, etc) (most significant bits set to increase the range).
0x01+	MUST	Timecode (relative to Cluster timecode, signed int16)

#### 6.2.3.2. Block Header Flags

Offset	Bit	Player	Description
0x03+	0-3	-	Reserved, set to 0
0x03+	4	-	Invisible, the codec SHOULD decode this frame but not display it
0x03+	5-6	MUST	Lacing * 00 : no lacing * 01 : Xiph lacing * 11 : EBML lacing * 10 : fixed-size lacing
0x03+	7	-	not used

#### 6.2.3.3. Laced Data

When lacing bit is set.



Offset	Player	Description
0x00	MUST	Number of frames in the lace-1 (uint8)
0x01 / 0xFF	MUST*	Lace-coded size of each frame of the lace, except for the last one (multiple uint8). *This is not used with Fixed-size lacing as it is calculated automatically from (total size of lace) / (number of frames in lace).

For (possibly) Laced Data

Offset	Player	Description
0x00	MUST	Consecutive laced frames

#### 6.2.4. 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). There are 3 types of lacing:

1. Xiph, inspired by what is found in the Ogg container
2. EBML, which is the same with sizes coded differently
3. fixed-size, where the size is not coded

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. They will then be stored in the same block as follows:

##### 6.2.4.1. Xiph lacing

- o Block head (with lacing bits set to 01)
- o Lacing head: Number of frames in the lace -1, i.e. 2 (the 800 and 500 octets one)
- o Lacing sizes: only the 2 first ones will be coded, 800 gives 255;255;255;35, 500 gives 255;245. The size of the last frame is deduced from the total size of the Block.



- o Data in frame 1
- o Data in frame 2
- o Data in frame 3

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.

#### 6.2.4.2. EBML lacing

In this case, the size is not coded as blocks of 255 bytes, but as a difference with the previous size and this size is coded as in EBML. The first size in the lace is unsigned as in EBML. The others use a range shifting to get a sign on each value:

Bit Representation	Value
1xxx xxxx	value $-(2^6-1)$ to $2^6-1$ (ie 0 to $2^7-2$ minus $2^6-1$ , half of the range)
01xx xxxx    xxxx xxxx	value $-(2^{13}-1)$ to $2^{13}-1$
001x xxxx    xxxx xxxx    xxxx xxxx	value $-(2^{20}-1)$ to $2^{20}-1$
0001 xxxx    xxxx xxxx    xxxx xxxx    xxxx xxxx	value $-(2^{27}-1)$ to $2^{27}-1$
0000 1xxx    xxxx xxxx    xxxx xxxx    xxxx xxxx	value $-(2^{34}-1)$ to $2^{34}-1$
0000 01xx    xxxx xxxx    xxxx xxxx    xxxx xxxx	value $-(2^{41}-1)$ to $2^{41}-1$
0000 001x    xxxx xxxx    xxxx xxxx    xxxx xxxx    xxxx xxxx	value $-(2^{48}-1)$ to $2^{48}-1$

- o Block head (with lacing bits set to 11)
- o Lacing head: Number of frames in the lace -1, i.e. 2 (the 800 and 400 octets one)
- o Lacing sizes: only the 2 first ones will be coded, 800 gives 0x320 0x4000 = 0x4320, 500 is coded as -300 : - 0x12C + 0x1FFF + 0x4000 = 0x5ED3. The size of the last frame is deduced from the total size of the Block.
- o Data in frame 1
- o Data in frame 2



- o Data in frame 3

#### 6.2.4.3. Fixed-size lacing

In this case, only the number of frames in the lace is saved, the size of each frame is deduced from the total size of the Block. For example, for 3 frames of 800 octets each:

- o Block head (with lacing bits set to 10)
- o Lacing head: Number of frames in the lace -1, i.e. 2
- o Data in frame 1
- o Data in frame 2
- o Data in frame 3

#### 6.2.4.4. SimpleBlock Structure

The "SimpleBlock" is inspired by the Section 6.2.3. The main differences are the added Keyframe flag and Discardable flag. Otherwise everything is the same.

Size = 1 + (1-8) + 4 + (4 + (4)) octets. So from 6 to 21 octets.

Bit 0 is the most significant bit.

Frames using references SHOULD be stored in "coding order". That means the references first and then the frames referencing them. A consequence is that timecodes MAY NOT be consecutive. But a frame with a past timecode MUST reference a frame already known, otherwise it's considered bad/void.

There can be many "Block Elements" in a "BlockGroup" provided they all have the same timecode. It is used with different parts of a frame with different priorities.

##### 6.2.4.4.1. SimpleBlock Header



Offset	Player	Description
0x00+	MUST	Track Number (Track Entry). It is coded in EBML like form (1 octet if the value is < 0x80, 2 if < 0x4000, etc) (most significant bits set to increase the range).
0x01+	MUST	Timecode (relative to Cluster timecode, signed int16)

#### 6.2.4.4.2. SimpleBlock Header Flags

Offset	Bit	Player	Description
0x03+	0	-	Keyframe, set when the Block contains only keyframes
0x03+	1-3	-	Reserved, set to 0
0x03+	4	-	Invisible, the codec SHOULD decode this frame but not display it
0x03+	5-6	MUST	Lacing * 00 : no lacing * 01 : Xiph lacing * 11 : EBML lacing * 10 : fixed-size lacing
0x03+	7	-	Discardable, the frames of the Block can be discarded during playing if needed

#### 6.2.4.5. Laced Data

When lacing bit is set.

Offset	Player	Description
0x00	MUST	Number of frames in the lace-1 (uint8)
0x01 / 0xXX	MUST*	Lace-coded size of each frame of the lace, except for the last one (multiple uint8). *This is not used with Fixed-size lacing as it is calculated automatically from (total size of lace) / (number of frames in lace).

For (possibly) Laced Data



Offset	Player	Description
0x00	MUST	Consecutive laced frames

## 7. Matroska Structure

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:

- o "EBML Header"
- o "Segment"

A more complex Matroska file consisting of an "EBML Stream" (consisting of two "EBML Documents") could be represented like this:

- o "EBML Header"
- o "Segment"
- o "EBML Header"
- o "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.



EBML Header	
Segment	SeekHead
	Info
	Tracks
	Chapters
	Cluster
	Cues
	Attachments
	Tags

The Matroska "EBML Schema" defines eight "Top Level Elements": "SeekHead", "Info", "Tracks", "Chapters", "Cluster", "Cues", "Attachments", and "Tags".

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".

SeekHead	Seek	SeekID
		SeekPosition

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".



Info	SegmentUID
	SegmentFilename
	PrevUID
	PrevFilename
	NextUID
	NextFilename
	SegmentFamily
	ChapterTranslate
	TimecodeScale
	Duration
	DateUTC
	Title
	MuxingApp
	WritingApp

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.



Tracks	TrackEntry	TrackNumber	
		TrackUID	
		TrackType	
		Name	
		Language	
		CodecID	
		CodecPrivate	
		CodecName	
		Video	FlagInterlaced
			FieldOrder
			StereoMode
			AlphaMode
			PixelWidth
			PixelHeight
			DisplayWidth
			DisplayHeight
			AspectRatioType
			Color
		Audio	SamplingFrequency
			Channels
			BitDepth

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.

Chapters	Edition Entry	EditionUID		
		EditionFlagHidden		
		EditionFlagDefault		
		EditionFlagOrdered		
	ChapterAtom	ChapterUID		
		ChapterStringUID		
		ChapterTimeStart		
		ChapterTimeEnd		
		ChapterFlagHidden		
ChapterDisplay		ChapString		
		ChapLanguage		

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". The "Cluster Element" helps to break up "SimpleBlock" or "BlockGroup Elements" and helps with seeking and error protection. It is RECOMMENDED that the size of each individual "Cluster Element" be limited to store no more than 5 seconds or 5 megabytes. Every "Cluster Element" MUST contain a "Timecode Element". This SHOULD be the "Timecode Element" used to play the first "Block" in the "Cluster Element". There SHOULD be one or more "BlockGroup" or "SimpleBlock Element" in each "Cluster Element". A "BlockGroup Element" MAY contain a "Block" of data and any information relating directly to that "Block".



Cluster	Timecode
	SilentTracks
	Position
	PrevSize
	SimpleBlock
	BlockGroup
	EncryptedBlock

Representation of a Cluster Element and its immediate Child Elements.

Block	Portion of a Block	Data Type - Bit Flag
	Header	TrackNumber
		Timecode
		Flags - Gap - Lacing - Reserved
	Optional	FrameSize
	Data	Frame

Representation of the Block Element structure.

Each "Cluster" MUST contain exactly one "Timecode Element". The "Timecode Element" value MUST be stored once per "Cluster". The "Timecode Element" in the "Cluster" is relative to the entire "Segment". The "Timecode Element" SHOULD be the first "Element" in the "Cluster".

Additionally, the "Block" contains an offset that, when added to the "Cluster"'s "Timecode Element" value, yields the "Block"'s effective timecode. Therefore, timecode in the "Block" itself is relative to the "Timecode Element" in the "Cluster". For example, if the



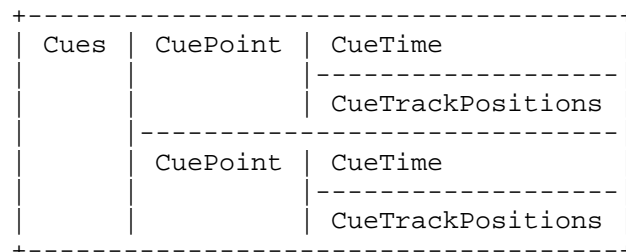
"Timecode Element" in the "Cluster" is set to 10 seconds and a "Block" in that "Cluster" is supposed to be played 12 seconds into the clip, the timecode in the "Block" would be set to 2 seconds.

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 "Timecode" 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 timecode.

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 timecode 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 timecode of every "Block" or they can be indexed selectively. For video files, it is RECOMMENDED to index at least the keyframes of the video track.



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, webpages, programs, or even the codec needed to play back the file.



Attachments	AttachedFile	FileDescription
		FileName
		FileMimeType
		FileData
		FileUID
		FileName
		FileReferral
		FileUsedStartTime
		FileUsedEndTime

Representation of a 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.



Tags	Tag	Targets	TargetTypeValue
			TargetType
			TagTrackUID
			TagEditionUID
			TagChapterUID
			TagAttachmentUID
		SimpleTag	TagName
			TagLanguage
			TagDefault
			TagString
			TagBinary
			SimpleTag

Representation of a Tags Element and three levels of its Children Elements.

## 8. Matroska Schema

This specification includes an "EBML Schema" which defines the Elements and structure of Matroska as an EBML Document Type. The EBML Schema defines every valid Matroska element in a manner defined by the EBML specification.

### 8.1. Matroska Additions to Schema Element Attributes

In addition to the EBML Schema definition provided by the EBML Specification, Matroska adds the following additional attributes:



attribute name	required	definition
webm	No	A boolean to express if the Matroska Element is also supported within version 2 of the "webm" specification. Please consider the webm specification [16] as the authoritative on "webm".

## 8.2. Matroska Schema

Here the definition of each Matroska Element is provided.

% concatenate with Matroska EBML Schema converted to markdown %

### 8.2.1. EBMLMaxIDLength Element

```
name: "EBMLMaxIDLength"
path: "1*1(\\EBML\\EBMLMaxIDLength)"
id: "0x42F2"
minOccurs: "1"
maxOccurs: "1"
range: "4"
default: "4"
type: "uinteger"
```

### 8.2.2. EBMLMaxSizeLength Element

```
name: "EBMLMaxSizeLength"
path: "1*1(\\EBML\\EBMLMaxSizeLength)"
id: "0x42F3"
minOccurs: "1"
maxOccurs: "1"
range: "1-8"
```



default: "8"

type: "uinteger"

#### 8.2.3. Segment Element

name: "Segment"

path: "1\*1(\\Segment)"

id: "0x18538067"

minOccurs: "1"

maxOccurs: "1"

type: "master"

unknownsizeallowed: "1"

minver: "1"

documentation: The Root Element that contains all other Top-Level Elements (Elements defined only at Level 1). A Matroska file is composed of 1 Segment.

#### 8.2.4. SeekHead Element

name: "SeekHead"

path: "0\*2(\\Segment\\SeekHead)"

id: "0x114D9B74"

maxOccurs: "2"

type: "master"

minver: "1"

documentation: Contains the Segment Position of other Top-Level Elements.

#### 8.2.5. Seek Element

name: "Seek"

path: "1\*(\\Segment\\SeekHead\\Seek)"



id: "0x4DBB"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Contains a single seek entry to an EBML Element.

#### 8.2.6. SeekID Element

name: "SeekID"

path: "1\*1(\Segment\SeekHead\Seek\SeekID)"

id: "0x53AB"

minOccurs: "1"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: The binary ID corresponding to the Element name.

#### 8.2.7. SeekPosition Element

name: "SeekPosition"

path: "1\*1(\Segment\SeekHead\Seek\SeekPosition)"

id: "0x53AC"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: The Segment Position of the Element.



## 8.2.8. Info Element

name: "Info"

path: "1\*(\Segment\Info)"

id: "0x1549A966"

minOccurs: "1"

type: "master"

minver: "1"

definition: Contains general information about the Segment.

## 8.2.9. SegmentUID Element

name: "SegmentUID"

path: "0\*1(\Segment\Info\SegmentUID)"

id: "0x73A4"

maxOccurs: "1"

range: "not 0"

size: "16"

type: "binary"

minver: "1"

definition: A randomly generated unique ID to identify the Segment amongst many others (128 bits).

usage notes: If the Segment is a part of a Linked Segment then this Element is REQUIRED.

## 8.2.10. SegmentFilename Element

name: "SegmentFilename"

path: "0\*1(\Segment\Info\SegmentFilename)"

id: "0x7384"



maxOccurs: "1"

type: "utf-8"

minver: "1"

definition: A filename corresponding to this Segment.

#### 8.2.11. PrevUID Element

name: "PrevUID"

path: "0\*1(\\Segment\\Info\\PrevUID)"

id: "0x3CB923"

maxOccurs: "1"

size: "16"

type: "binary"

minver: "1"

definition: A unique ID to identify the previous Segment of a Linked Segment (128 bits).

usage notes: If the Segment is a part of a Linked Segment that uses Hard Linking then either the PrevUID or the NextUID Element is REQUIRED. If a Segment contains a PrevUID but not a NextUID then it MAY be considered as the last Segment of the Linked Segment. The PrevUID MUST NOT be equal to the SegmentUID.

#### 8.2.12. PrevFilename Element

name: "PrevFilename"

path: "0\*1(\\Segment\\Info\\PrevFilename)"

id: "0x3C83AB"

maxOccurs: "1"

type: "utf-8"

minver: "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 PrevUID SHOULD be considered authoritative for identifying the previous Segment in a Linked Segment.

#### 8.2.13. NextUID Element

name: "NextUID"

path: "0\*1(\\Segment\\Info\\NextUID)"

id: "0x3EB923"

maxOccurs: "1"

size: "16"

type: "binary"

minver: "1"

definition: A unique ID to identify the next Segment of a Linked Segment (128 bits).

usage notes: If the Segment is a part of a Linked Segment that uses Hard Linking then either the PrevUID or the NextUID Element is REQUIRED. If a Segment contains a NextUID but not a PrevUID then it MAY be considered as the first Segment of the Linked Segment. The NextUID MUST NOT be equal to the SegmentUID.

#### 8.2.14. NextFilename Element

name: "NextFilename"

path: "0\*1(\\Segment\\Info\\NextFilename)"

id: "0x3E83BB"

maxOccurs: "1"

type: "utf-8"

minver: "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 NextUID SHOULD be considered authoritative for identifying the Next Segment.

#### 8.2.15. SegmentFamily Element

name: "SegmentFamily"

path: "0\*(\\Segment\\Info\\SegmentFamily)"

id: "0x4444"

size: "16"

type: "binary"

minver: "1"

definition: A randomly generated unique ID that all Segments of a Linked Segment MUST share (128 bits).

usage notes: If the Segment is a part of a Linked Segment that uses Soft Linking then this Element is REQUIRED.

#### 8.2.16. ChapterTranslate Element

name: "ChapterTranslate"

path: "0\*(\\Segment\\Info\\ChapterTranslate)"

id: "0x6924"

type: "master"

minver: "1"

documentation: A tuple of corresponding ID used by chapter codecs to represent this Segment.

#### 8.2.17. ChapterTranslateEditionUID Element

name: "ChapterTranslateEditionUID"

path: "0\*(\\Segment\\Info\\ChapterTranslate\\ChapterTranslateEditionUID)"

id: "0x69FC"

type: "uinteger"



minver: "1"

documentation: Specify an edition UID on which this correspondance applies. When not specified, it means for all editions found in the Segment.

#### 8.2.18. ChapterTranslateCodec Element

name: "ChapterTranslateCodec"

path: "1\*1(\\Segment\\Info\\ChapterTranslate\\ChapterTranslateCodec)"

id: "0x69BF"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: The chapter codec

#### 8.2.19. ChapterTranslateID Element

name: "ChapterTranslateID"

path: "1\*1(\\Segment\\Info\\ChapterTranslate\\ChapterTranslateID)"

id: "0x69A5"

minOccurs: "1"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: The binary value used to represent this Segment in the chapter codec data. The format depends on the ChapProcessCodecID used.



## 8.2.20. TimecodeScale Element

name: "TimecodeScale"  
path: "1\*1(\\Segment\\Info\\TimecodeScale)"  
id: "0x2AD7B1"  
minOccurs: "1"  
maxOccurs: "1"  
range: "not 0"  
default: "1000000"  
type: "uinteger"  
minver: "1"  
  
documentation: Timestamp scale in nanoseconds (1.000.000 means all timestamps in the Segment are expressed in milliseconds).

## 8.2.21. Duration Element

name: "Duration"  
path: "0\*1(\\Segment\\Info\\Duration)"  
id: "0x4489"  
maxOccurs: "1"  
range: "> 0x0p+0"  
type: "float"  
minver: "1"  
  
definition: Duration of the Segment in nanoseconds based on TimecodeScale.

## 8.2.22. DateUTC Element

name: "DateUTC"  
path: "0\*1(\\Segment\\Info\\DateUTC)"



id: "0x4461"

maxOccurs: "1"

type: "date"

minver: "1"

documentation: The date and time that the Segment was created by the muxing application or library.

#### 8.2.23. Title Element

name: "Title"

path: "0\*1(\\Segment\\Info\\Title)"

id: "0x7BA9"

maxOccurs: "1"

type: "utf-8"

minver: "1"

documentation: General name of the Segment.

#### 8.2.24. MuxingApp Element

name: "MuxingApp"

path: "1\*1(\\Segment\\Info\\MuxingApp)"

id: "0x4D80"

minOccurs: "1"

maxOccurs: "1"

type: "utf-8"

minver: "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.



## 8.2.25. WritingApp Element

name: "WritingApp"  
path: "1\*1(\\Segment\\Info\\WritingApp)"  
id: "0x5741"  
minOccurs: "1"  
maxOccurs: "1"  
type: "utf-8"  
minver: "1"  
definition: Writing application (example: "mkvmerge-0.3.3").  
usage notes: Include the full name of the application followed by the version number.

## 8.2.26. Cluster Element

name: "Cluster"  
path: "0\*(\\Segment\\Cluster)"  
id: "0x1F43B675"  
type: "master"  
unknownsizeallowed: "1"  
minver: "1"  
documentation: The Top-Level Element containing the (monolithic) Block structure.

## 8.2.27. Timecode Element

name: "Timecode"  
path: "1\*1(\\Segment\\Cluster\\Timecode)"  
id: "0xE7"  
minOccurs: "1"



maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: Absolute timestamp of the cluster (based on TimecodeScale).

#### 8.2.28. SilentTracks Element

name: "SilentTracks"

path: "0\*1(\\Segment\\Cluster\\SilentTracks)"

id: "0x5854"

maxOccurs: "1"

type: "master"

minver: "1"

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.

#### 8.2.29. SilentTrackNumber Element

name: "SilentTrackNumber"

path: "0\*(\\Segment\\Cluster\\SilentTracks\\SilentTrackNumber)"

id: "0x58D7"

type: "uinteger"

minver: "1"

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.

#### 8.2.30. Position Element

name: "Position"

path: "0\*1(\\Segment\\Cluster\\Position)"



id: "0xA7"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: The Segment Position of the Cluster in the Segment (0 in live broadcast streams). It might help to resynchronise offset on damaged streams.

#### 8.2.31. PrevSize Element

name: "PrevSize"

path: "0\*1(\\Segment\\Cluster\\PrevSize)"

id: "0xAB"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: Size of the previous Cluster, in octets. Can be useful for backward playing.

#### 8.2.32. SimpleBlock Element

name: "SimpleBlock"

path: "0\*(\\Segment\\Cluster\\SimpleBlock)"

id: "0xA3"

type: "binary"

minver: "2"

documentation: Similar to Block but without all the extra information, mostly used to reduced overhead when no extra feature is needed. (see SimpleBlock Structure)



## 8.2.33. BlockGroup Element

name: "BlockGroup"

path: "0\*(\Segment\Cluster\BlockGroup)"

id: "0xA0"

type: "master"

minver: "1"

documentation: Basic container of information containing a single Block and information specific to that Block.

## 8.2.34. Block Element

name: "Block"

path: "1\*1(\Segment\Cluster\BlockGroup\Block)"

id: "0xA1"

minOccurs: "1"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: Block containing the actual data to be rendered and a timestamp relative to the Cluster Timecode. (see Block Structure)

## 8.2.35. BlockVirtual Element

name: "BlockVirtual"

path: "0\*1(\Segment\Cluster\BlockGroup\BlockVirtual)"

id: "0xA2"

maxOccurs: "1"

type: "binary"

minver: "0"



maxver: "0"

documentation: A Block with no data. It MUST be stored in the stream at the place the real Block would be in display order. (see Block Virtual)

#### 8.2.36. BlockAdditions Element

name: "BlockAdditions"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\BlockAdditions)"

id: "0x75A1"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: Contain additional blocks to complete the main one. An EBML parser that has no knowledge of the Block structure could still see and use/skip these data.

#### 8.2.37. BlockMore Element

name: "BlockMore"

path: "1\*(\\Segment\\Cluster\\BlockGroup\\BlockAdditions\\BlockMore)"

id: "0xA6"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Contain the BlockAdditional and some parameters.

#### 8.2.38. BlockAddID Element

name: "BlockAddID"

path: "1\*1(\\Segment\\Cluster\\BlockGroup\\BlockAdditions\\BlockMore\\BlockAddID)"

id: "0xEE"



```
minOccurs: "1"
maxOccurs: "1"
range: "not 0"
default: "1"
type: "uinteger"
minver: "1"

documentation: An ID to identify the BlockAdditional level.
```

#### 8.2.39. BlockAdditional Element

```
name: "BlockAdditional"

path: "1*1(\\Segment\\Cluster\\BlockGroup\\BlockAdditions\\BlockMore\\Block
Additional)"

id: "0xA5"

minOccurs: "1"
maxOccurs: "1"
type: "binary"
minver: "1"

documentation: Interpreted by the codec as it wishes (using the
BlockAddID).
```

#### 8.2.40. BlockDuration Element

```
name: "BlockDuration"

path: "0*1(\\Segment\\Cluster\\BlockGroup\\BlockDuration)"

id: "0x9B"

maxOccurs: "1"
default: "DefaultDuration"
type: "uinteger"
```



minver: "1"

documentation: The duration of the Block (based on TimecodeScale). This Element is mandatory when DefaultDuration is set for the track (but can be omitted as other default values). 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). This Element can be useful at the end of a Track (as there is not other Block available), or when there is a break in a track like for subtitle tracks.

#### 8.2.41. ReferencePriority Element

name: "ReferencePriority"

path: "1\*1(\Segment\Cluster\BlockGroup\ReferencePriority)"

id: "0xFA"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: 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.

#### 8.2.42. ReferenceBlock Element

name: "ReferenceBlock"

path: "0\*(\Segment\Cluster\BlockGroup\ReferenceBlock)"

id: "0xFB"

type: "integer"

minver: "1"

documentation: Timestamp of another frame used as a reference (ie: B or P frame). The timestamp is relative to the block it's attached to.



## 8.2.43. ReferenceVirtual Element

name: "ReferenceVirtual"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\ReferenceVirtual)"

id: "0xFD"

maxOccurs: "1"

type: "integer"

minver: "0"

maxver: "0"

documentation: The Segment Position of the data that would otherwise be in position of the virtual block.

## 8.2.44. CodecState Element

name: "CodecState"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\CodecState)"

id: "0xA4"

maxOccurs: "1"

type: "binary"

minver: "2"

documentation: The new codec state to use. Data interpretation is private to the codec. This information SHOULD always be referenced by a seek entry.

## 8.2.45. DiscardPadding Element

name: "DiscardPadding"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\DiscardPadding)"

id: "0x75A2"

maxOccurs: "1"

type: "integer"



minver: "4"

documentation: Duration in nanoseconds of the silent data added to the Block (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.

#### 8.2.46. Slices Element

name: "Slices"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\Slices)"

id: "0x8E"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: Contains slices description.

#### 8.2.47. TimeSlice Element

name: "TimeSlice"

path: "0\*(\\Segment\\Cluster\\BlockGroup\\Slices\\TimeSlice)"

id: "0xE8"

type: "master"

minver: "1"

maxver: "1"

documentation: Contains extra time information about the data contained in the Block. Being able to interpret this Element is not REQUIRED for playback.

#### 8.2.48. LaceNumber Element

name: "LaceNumber"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\Slices\\TimeSlice\\LaceNumber)"



id: "0xCC"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

maxver: "1"

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.

#### 8.2.49. FrameNumber Element

name: "FrameNumber"

path: "0\*1(\\Segment\\Cluster\\BlockGroup\\Slices\\TimeSlice\\FrameNumber)"

id: "0xCD"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "0"

maxver: "0"

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).

#### 8.2.50. BlockAdditionID Element

name: "BlockAdditionID"

path:

"0\*1(\\Segment\\Cluster\\BlockGroup\\Slices\\TimeSlice\\BlockAdditionID)"

id: "0xCB"

maxOccurs: "1"



```
default: "0"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: The ID of the BlockAdditional Element (0 is the main
Block).
```

#### 8.2.51. Delay Element

```
name: "Delay"

path: "0*1(\Segment\Cluster\BlockGroup\Slices\TimeSlice\Delay)"

id: "0xCE"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: The (scaled) delay to apply to the Element.
```

#### 8.2.52. SliceDuration Element

```
name: "SliceDuration"

path:
"0*1(\Segment\Cluster\BlockGroup\Slices\TimeSlice\SliceDuration)"

id: "0xCF"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "0"
```



maxver: "0"

documentation: The (scaled) duration to apply to the Element.

#### 8.2.53. ReferenceFrame Element

name: "ReferenceFrame"

path: "0\*1(\Segment\Cluster\BlockGroup\ReferenceFrame)"

id: "0xC8"

maxOccurs: "1"

type: "master"

minver: "0"

maxver: "0"

documentation: DivX trick track extensions

#### 8.2.54. ReferenceOffset Element

name: "ReferenceOffset"

path:  
"1\*1(\Segment\Cluster\BlockGroup\ReferenceFrame\ReferenceOffset)"

id: "0xC9"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: DivX trick track extensions

#### 8.2.55. ReferenceTimeCode Element

name: "ReferenceTimeCode"



```
path:
  "1*1(\Segment\Cluster\BlockGroup\ReferenceFrame\ReferenceTimeCode)"

id: "0xCA"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: DivX trick track extensions
```

#### 8.2.56. EncryptedBlock Element

```
name: "EncryptedBlock"

path: "0*(\Segment\Cluster\EncryptedBlock)"

id: "0xAF"

type: "binary"

minver: "0"

maxver: "0"

documentation: Similar to SimpleBlock but the data inside the Block
are Transformed (encrypt and/or signed). (see EncryptedBlock
Structure)
```

#### 8.2.57. Tracks Element

```
name: "Tracks"

path: "0*(\Segment\Tracks)"

id: "0x1654AE6B"

type: "master"

minver: "1"
```



documentation: A Top-Level Element of information with many tracks described.

#### 8.2.58. TrackEntry Element

name: "TrackEntry"

path: "1\*(\Segment\Tracks\TrackEntry)"

id: "0xAE"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Describes a track with all Elements.

#### 8.2.59. TrackNumber Element

name: "TrackNumber"

path: "1\*1(\Segment\Tracks\TrackEntry\TrackNumber)"

id: "0xD7"

minOccurs: "1"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "1"

documentation: The track number as used in the Block Header (using more than 127 tracks is not encouraged, though the design allows an unlimited number).

#### 8.2.60. TrackUID Element

name: "TrackUID"

path: "1\*1(\Segment\Tracks\TrackEntry\TrackUID)"

id: "0x73C5"



minOccurs: "1"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "1"

documentation: A unique ID to identify the Track. This SHOULD be kept the same when making a direct stream copy of the Track to another file.

#### 8.2.61. TrackType Element

name: "TrackType"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\TrackType)"

id: "0x83"

minOccurs: "1"

maxOccurs: "1"

range: "1-254"

type: "uinteger"

minver: "1"

documentation: A set of track types coded on 8 bits.

#### 8.2.62. FlagEnabled Element

name: "FlagEnabled"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\FlagEnabled)"

id: "0xB9"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"



default: "1"  
type: "uinteger"  
minver: "2"  
documentation: Set if the track is usable. (1 bit)

#### 8.2.63. FlagDefault Element

name: "FlagDefault"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\FlagDefault)"  
id: "0x88"  
minOccurs: "1"  
maxOccurs: "1"  
range: "0-1"  
default: "1"  
type: "uinteger"  
minver: "1"  
documentation: Set if that track (audio, video or subs) SHOULD be active if no language found matches the user preference. (1 bit)

#### 8.2.64. FlagForced Element

name: "FlagForced"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\FlagForced)"  
id: "0x55AA"  
minOccurs: "1"  
maxOccurs: "1"  
range: "0-1"  
default: "0"  
type: "uinteger"



minver: "1"

documentation: Set if that track MUST be active during playback. There can be many forced track for a kind (audio, video or subs), the player SHOULD select the one which language matches the user preference or the default + forced track. Overlay MAY happen between a forced and non-forced track of the same kind. (1 bit)

#### 8.2.65. FlagLacing Element

name: "FlagLacing"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\FlagLacing)"

id: "0x9C"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"

default: "1"

type: "uinteger"

minver: "1"

documentation: Set if the track MAY contain blocks using lacing. (1 bit)

#### 8.2.66. MinCache Element

name: "MinCache"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\MinCache)"

id: "0x6DE7"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"



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.

#### 8.2.67. MaxCache Element

name: "MaxCache"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\MaxCache)"

id: "0x6DF8"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: The maximum cache size necessary to store referenced frames in and the current frame. 0 means no cache is needed.

#### 8.2.68. DefaultDuration Element

name: "DefaultDuration"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\DefaultDuration)"

id: "0x23E383"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "1"

documentation: Number of nanoseconds (not scaled via TimecodeScale) per frame ('frame' in the Matroska sense -- one Element put into a (Simple)Block).

#### 8.2.69. DefaultDecodedFieldDuration Element

name: "DefaultDecodedFieldDuration"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\DefaultDecodedFieldDuration)"

id: "0x234E7A"



maxOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "4"  
  
documentation: The period in nanoseconds (not scaled by  
TimecodeScale) between two successive fields at the output of the  
decoding process (see the notes)

#### 8.2.70. TrackTimecodeScale Element

name: "TrackTimecodeScale"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\TrackTimecodeScale)"  
id: "0x23314F"  
minOccurs: "1"  
maxOccurs: "1"  
range: "> 0x0p+0"  
default: "0x1p+0"  
type: "float"  
minver: "1"  
maxver: "3"  
  
documentation: DEPRECATED, DO NOT USE. 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).

#### 8.2.71. TrackOffset Element

name: "TrackOffset"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\TrackOffset)"  
id: "0x537F"  
maxOccurs: "1"



default: "0"

type: "integer"

minver: "0"

maxver: "0"

documentation: A value to add to the Block's Timestamp. This can be used to adjust the playback offset of a track.

#### 8.2.72. MaxBlockAdditionID Element

name: "MaxBlockAdditionID"

path: "1\*1(\Segment\Tracks\TrackEntry\MaxBlockAdditionID)"

id: "0x55EE"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The maximum value of BlockAddID. A value 0 means there is no BlockAdditions for this track.

#### 8.2.73. Name Element

name: "Name"

path: "0\*1(\Segment\Tracks\TrackEntry\Name)"

id: "0x536E"

maxOccurs: "1"

type: "utf-8"

minver: "1"

documentation: A human-readable track name.



## 8.2.74. Language Element

name: "Language"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Language)"

id: "0x22B59C"

maxOccurs: "1"

default: "eng"

type: "string"

minver: "1"

documentation: Specifies the language of the track in the Matroska languages form. This Element MUST be ignored if the LanguageIETF Element is used in the same TrackEntry.

## 8.2.75. LanguageIETF Element

name: "LanguageIETF"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\LanguageIETF)"

id: "0x22B59D"

maxOccurs: "1"

type: "string"

minver: "4"

documentation: Specifies the language of the track according to BCP 47 and using the IANA Language Subtag Registry. If this Element is used, then any Language Elements used in the same TrackEntry MUST be ignored.

## 8.2.76. CodecID Element

name: "CodecID"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\CodecID)"

id: "0x86"

minOccurs: "1"



maxOccurs: "1"

type: "string"

minver: "1"

documentation: An ID corresponding to the codec, see the codec page for more info.

#### 8.2.77. CodecPrivate Element

name: "CodecPrivate"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\CodecPrivate)"

id: "0x63A2"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: Private data only known to the codec.

#### 8.2.78. CodecName Element

name: "CodecName"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\CodecName)"

id: "0x258688"

maxOccurs: "1"

type: "utf-8"

minver: "1"

documentation: A human-readable string specifying the codec.

#### 8.2.79. AttachmentLink Element

name: "AttachmentLink"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\AttachmentLink)"

id: "0x7446"



maxOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "1"  
maxver: "3"

documentation: The UID of an attachment that is used by this codec.

#### 8.2.80. CodecSettings Element

name: "CodecSettings"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\CodecSettings)"  
id: "0x3A9697"  
maxOccurs: "1"  
type: "utf-8"  
minver: "0"  
maxver: "0"

documentation: A string describing the encoding setting used.

#### 8.2.81. CodecInfoURL Element

name: "CodecInfoURL"  
path: "0\*(\\Segment\\Tracks\\TrackEntry\\CodecInfoURL)"  
id: "0x3B4040"  
type: "string"  
minver: "0"  
maxver: "0"

documentation: A URL to find information about the codec used.



## 8.2.82. CodecDownloadURL Element

name: "CodecDownloadURL"  
path: "0\*(\Segment\Tracks\TrackEntry\CodecDownloadURL)"  
id: "0x26B240"  
type: "string"  
minver: "0"  
maxver: "0"  
documentation: A URL to download about the codec used.

## 8.2.83. CodecDecodeAll Element

name: "CodecDecodeAll"  
path: "1\*1(\Segment\Tracks\TrackEntry\CodecDecodeAll)"  
id: "0xAA"  
minOccurs: "1"  
maxOccurs: "1"  
range: "0-1"  
default: "1"  
type: "uinteger"  
minver: "2"  
documentation: The codec can decode potentially damaged data (1 bit).

## 8.2.84. TrackOverlay Element

name: "TrackOverlay"  
path: "0\*(\Segment\Tracks\TrackEntry\TrackOverlay)"  
id: "0x6FAB"  
type: "uinteger"



minver: "1"

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 (see 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.

#### 8.2.85. CodecDelay Element

name: "CodecDelay"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\CodecDelay)"

id: "0x56AA"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "4"

documentation: CodecDelay is The codec-built-in delay in nanoseconds. This value MUST be subtracted from each block timestamp in order to get the actual timestamp. The value SHOULD be small so the muxing of tracks with the same actual timestamp are in the same Cluster.

#### 8.2.86. SeekPreRoll Element

name: "SeekPreRoll"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\SeekPreRoll)"

id: "0x56BB"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "4"



documentation: After a discontinuity, SeekPreRoll is the duration in nanoseconds of the data the decoder MUST decode before the decoded data is valid.

#### 8.2.87. TrackTranslate Element

name: "TrackTranslate"

path: "0\*(\\Segment\\Tracks\\TrackEntry\\TrackTranslate)"

id: "0x6624"

type: "master"

minver: "1"

documentation: The track identification for the given Chapter Codec.

#### 8.2.88. TrackTranslateEditionUID Element

name: "TrackTranslateEditionUID"

path: "0\*(\\Segment\\Tracks\\TrackEntry\\TrackTranslate\\TrackTranslateEditionUID)"

id: "0x66FC"

type: "uinteger"

minver: "1"

documentation: Specify an edition UID on which this translation applies. When not specified, it means for all editions found in the Segment.

#### 8.2.89. TrackTranslateCodec Element

name: "TrackTranslateCodec"

path:  
"1\*1(\\Segment\\Tracks\\TrackEntry\\TrackTranslate\\TrackTranslateCodec)"

id: "0x66BF"

minOccurs: "1"

maxOccurs: "1"



type: "uinteger"

minver: "1"

documentation: The chapter codec.

#### 8.2.90. TrackTranslateTrackID Element

name: "TrackTranslateTrackID"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\TrackTranslate\\TrackTranslateTrackID)"

id: "0x66A5"

minOccurs: "1"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: The binary value used to represent this track in the chapter codec data. The format depends on the ChapProcessCodecID used.

#### 8.2.91. Video Element

name: "Video"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video)"

id: "0xE0"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: Video settings.

#### 8.2.92. FlagInterlaced Element

name: "FlagInterlaced"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\Video\\FlagInterlaced)"



id: "0x9A"  
minOccurs: "1"  
maxOccurs: "1"  
range: "0-2"  
default: "0"  
type: "uinteger"  
minver: "2"  
  
documentation: A flag to declare is the video is known to be progressive or interlaced and if applicable to declare details about the interlacement.

#### 8.2.93. FieldOrder Element

name: "FieldOrder"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\Video\\FieldOrder)"  
id: "0x9D"  
minOccurs: "1"  
maxOccurs: "1"  
range: "0-14"  
default: "2"  
type: "uinteger"  
minver: "4"  
  
documentation: Declare the field ordering of the video. If FlagInterlaced is not set to 1, this Element MUST be ignored.

#### 8.2.94. StereoMode Element

name: "StereoMode"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\StereoMode)"  
id: "0x53B8"



maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "3"

documentation: Stereo-3D video mode. There are some more details on 3D support in the Specification Notes.

#### 8.2.95. AlphaMode Element

name: "AlphaMode"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\AlphaMode)"

id: "0x53C0"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "3"

documentation: Alpha Video Mode. Presence of this Element indicates that the BlockAdditional Element could contain Alpha data.

#### 8.2.96. OldStereoMode Element

name: "OldStereoMode"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\OldStereoMode)"

id: "0x53B9"

maxOccurs: "1"

type: "uinteger"

maxver: "0"

documentation: DEPRECATED, DO NOT USE. Bogus StereoMode value used in old versions of libmatroska.



## 8.2.97. PixelWidth Element

name: "PixelWidth"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\Video\\PixelWidth)"  
id: "0xB0"  
minOccurs: "1"  
maxOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "1"  
documentation: Width of the encoded video frames in pixels.

## 8.2.98. PixelHeight Element

name: "PixelHeight"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\Video\\PixelHeight)"  
id: "0xBA"  
minOccurs: "1"  
maxOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "1"  
documentation: Height of the encoded video frames in pixels.

## 8.2.99. PixelCropBottom Element

name: "PixelCropBottom"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\PixelCropBottom)"  
id: "0x54AA"



maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The number of video pixels to remove at the bottom of the image (for HDTV content).

#### 8.2.100. PixelCropTop Element

name: "PixelCropTop"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\PixelCropTop)"

id: "0x54BB"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The number of video pixels to remove at the top of the image.

#### 8.2.101. PixelCropLeft Element

name: "PixelCropLeft"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\PixelCropLeft)"

id: "0x54CC"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The number of video pixels to remove on the left of the image.



## 8.2.102. PixelCropRight Element

name: "PixelCropRight"

path: "0\*1(\Segment\Tracks\TrackEntry\Video\PixelCropRight)"

id: "0x54DD"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The number of video pixels to remove on the right of the image.

## 8.2.103. DisplayWidth Element

name: "DisplayWidth"

path: "0\*1(\Segment\Tracks\TrackEntry\Video\DisplayWidth)"

id: "0x54B0"

maxOccurs: "1"

range: "not 0"

default: "PixelWidth - PixelCropLeft - PixelCropRight"

type: "uinteger"

minver: "1"

documentation: Width of the video frames to display. Applies to the video frame after cropping (PixelCrop\* Elements). The default value is only valid when DisplayUnit is 0.

## 8.2.104. DisplayHeight Element

name: "DisplayHeight"

path: "0\*1(\Segment\Tracks\TrackEntry\Video\DisplayHeight)"

id: "0x54BA"



maxOccurs: "1"  
range: "not 0"  
default: "PixelHeight - PixelCropTop - PixelCropBottom"  
type: "uinteger"  
minver: "1"  
documentation: Height of the video frames to display. Applies to the video frame after cropping (PixelCrop\* Elements). The default value is only valid when DisplayUnit is 0.

#### 8.2.105. DisplayUnit Element

name: "DisplayUnit"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\DisplayUnit)"  
id: "0x54B2"  
maxOccurs: "1"  
default: "0"  
type: "uinteger"  
minver: "1"  
documentation: How DisplayWidth & DisplayHeight are interpreted.

#### 8.2.106. AspectRatioType Element

name: "AspectRatioType"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\AspectRatioType)"  
id: "0x54B3"  
maxOccurs: "1"  
default: "0"  
type: "uinteger"  
minver: "1"



documentation: Specify the possible modifications to the aspect ratio.

#### 8.2.107. ColourSpace Element

name: "ColourSpace"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\ColourSpace)"

id: "0x2EB524"

maxOccurs: "1"

size: "4"

type: "binary"

minver: "1"

documentation: Specify the pixel format used for the Track's data as a FourCC. This value is similar in scope to the biCompression value of AVI's BITMAPINFOHEADER. This Element is MANDATORY in TrackEntry when the CodecID Element of the TrackEntry is set to "V\_UNCOMPRESSED".

#### 8.2.108. GammaValue Element

name: "GammaValue"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\GammaValue)"

id: "0x2FB523"

maxOccurs: "1"

range: "> 0x0p+0"

type: "float"

minver: "0"

maxver: "0"

documentation: Gamma Value.



## 8.2.109. FrameRate Element

name: "FrameRate"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\FrameRate)"  
id: "0x2383E3"  
maxOccurs: "1"  
range: "> 0x0p+0"  
type: "float"  
minver: "0"  
maxver: "0"  
documentation: Number of frames per second. Informational only.

## 8.2.110. Colour Element

name: "Colour"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour)"  
id: "0x55B0"  
maxOccurs: "1"  
type: "master"  
minver: "4"  
documentation: Settings describing the colour format.

## 8.2.111. MatrixCoefficients Element

name: "MatrixCoefficients"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MatrixCoefficients)"  
id: "0x55B1"  
maxOccurs: "1"  
default: "2"



type: "uinteger"

minver: "4"

documentation: 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 ISO/IEC 23001-8:2013/DCOR1.

#### 8.2.112. BitsPerChannel Element

name: "BitsPerChannel"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\BitsPerChannel)"

id: "0x55B2"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "4"

documentation: Number of decoded bits per channel. A value of 0 indicates that the BitsPerChannel is unspecified.

#### 8.2.113. ChromaSubsamplingHorz Element

name: "ChromaSubsamplingHorz"

path:  
"0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\ChromaSubsamplingHorz)"

id: "0x55B3"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: 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.



## 8.2.114. ChromaSubsamplingVert Element

name: "ChromaSubsamplingVert"

path:

"0\*1(\Segment\Tracks\TrackEntry\Video\Colour\ChromaSubsamplingVert)"

id: "0x55B4"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: 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.

## 8.2.115. CbSubsamplingHorz Element

name: "CbSubsamplingHorz"

path:

"0\*1(\Segment\Tracks\TrackEntry\Video\Colour\CbSubsamplingHorz)"

id: "0x55B5"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: 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.

## 8.2.116. CbSubsamplingVert Element

name: "CbSubsamplingVert"

path:

"0\*1(\Segment\Tracks\TrackEntry\Video\Colour\CbSubsamplingVert)"



id: "0x55B6"  
maxOccurs: "1"  
type: "uinteger"  
minver: "4"  
documentation: The amount of pixels to remove in the Cb channel for every pixel not removed vertically. This is additive with ChromaSubsamplingVert.

#### 8.2.117. ChromaSitingHorz Element

name: "ChromaSitingHorz"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\ChromaSitingHorz)"  
id: "0x55B7"  
maxOccurs: "1"  
default: "0"  
type: "uinteger"  
minver: "4"  
documentation: How chroma is subsampled horizontally.

#### 8.2.118. ChromaSitingVert Element

name: "ChromaSitingVert"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\ChromaSitingVert)"  
id: "0x55B8"  
maxOccurs: "1"  
default: "0"  
type: "uinteger"  
minver: "4"  
documentation: How chroma is subsampled vertically.



## 8.2.119. Range Element

name: "Range"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\Range)"  
id: "0x55B9"  
maxOccurs: "1"  
default: "0"  
type: "uinteger"  
minver: "4"  
documentation: Clipping of the color ranges.

## 8.2.120. TransferCharacteristics Element

name: "TransferCharacteristics"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\TransferCharacteristics)"  
id: "0x55BA"  
maxOccurs: "1"  
default: "2"  
type: "uinteger"  
minver: "4"  
documentation: The transfer characteristics of the video. For clarity, the value and meanings for TransferCharacteristics 1-15 are adopted from Table 3 of ISO/IEC 23001-8:2013/DCOR1. TransferCharacteristics 16-18 are proposed values.

## 8.2.121. Primaries Element

name: "Primaries"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\Primaries)"  
id: "0x55BB"



maxOccurs: "1"

default: "2"

type: "uinteger"

minver: "4"

documentation: The colour primaries of the video. For clarity, the value and meanings for Primaries are adopted from Table 2 of ISO/IEC 23001-8:2013/DCOR1.

#### 8.2.122. MaxCLL Element

name: "MaxCLL"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MaxCLL)"

id: "0x55BC"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: Maximum brightness of a single pixel (Maximum Content Light Level) in candelas per square meter (cd/m<sup>2</sup>).

#### 8.2.123. MaxFALL Element

name: "MaxFALL"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MaxFALL)"

id: "0x55BD"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: Maximum brightness of a single full frame (Maximum Frame-Average Light Level) in candelas per square meter (cd/m<sup>2</sup>).



## 8.2.124. MasteringMetadata Element

name: "MasteringMetadata"  
path:  
"0\*1(\Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata)"  
id: "0x55D0"  
maxOccurs: "1"  
type: "master"  
minver: "4"  
documentation: SMPTE 2086 mastering data.

## 8.2.125. PrimaryRChromaticityX Element

name: "PrimaryRChromaticityX"  
path: "0\*1(\Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\  
PrimaryRChromaticityX)"  
id: "0x55D1"  
maxOccurs: "1"  
range: "0-1"  
type: "float"  
minver: "4"  
documentation: Red X chromaticity coordinate as defined by CIE 1931.

## 8.2.126. PrimaryRChromaticityY Element

name: "PrimaryRChromaticityY"  
path: "0\*1(\Segment\Tracks\TrackEntry\Video\Colour\MasteringMetadata\  
PrimaryRChromaticityY)"  
id: "0x55D2"  
maxOccurs: "1"  
range: "0-1"



type: "float"

minver: "4"

documentation: Red Y chromaticity coordinate as defined by CIE 1931.

#### 8.2.127. PrimaryGChromaticityX Element

name: "PrimaryGChromaticityX"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\PrimaryGChromaticityX)"

id: "0x55D3"

maxOccurs: "1"

range: "0-1"

type: "float"

minver: "4"

documentation: Green X chromaticity coordinate as defined by CIE 1931.

#### 8.2.128. PrimaryGChromaticityY Element

name: "PrimaryGChromaticityY"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\PrimaryGChromaticityY)"

id: "0x55D4"

maxOccurs: "1"

range: "0-1"

type: "float"

minver: "4"

documentation: Green Y chromaticity coordinate as defined by CIE 1931.



## 8.2.129. PrimaryBChromaticityX Element

name: "PrimaryBChromaticityX"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\PrimaryBChromaticityX)"

id: "0x55D5"

maxOccurs: "1"

range: "0-1"

type: "float"

minver: "4"

documentation: Blue X chromaticity coordinate as defined by CIE 1931.

## 8.2.130. PrimaryBChromaticityY Element

name: "PrimaryBChromaticityY"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\PrimaryBChromaticityY)"

id: "0x55D6"

maxOccurs: "1"

range: "0-1"

type: "float"

minver: "4"

documentation: Blue Y chromaticity coordinate as defined by CIE 1931.

## 8.2.131. WhitePointChromaticityX Element

name: "WhitePointChromaticityX"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\WhitePointChromaticityX)"

id: "0x55D7"

maxOccurs: "1"



range: "0-1"

type: "float"

minver: "4"

documentation: White X chromaticity coordinate as defined by CIE 1931.

#### 8.2.132. WhitePointChromaticityY Element

name: "WhitePointChromaticityY"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\WhitePointChromaticityY)"

id: "0x55D8"

maxOccurs: "1"

range: "0-1"

type: "float"

minver: "4"

documentation: White Y chromaticity coordinate as defined by CIE 1931.

#### 8.2.133. LuminanceMax Element

name: "LuminanceMax"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\LuminanceMax)"

id: "0x55D9"

maxOccurs: "1"

range: ">= 0x0p+0"

type: "float"

minver: "4"

documentation: Maximum luminance. Represented in candelas per square meter (cd/m<sup>2</sup>).



## 8.2.134. LuminanceMin Element

name: "LuminanceMin"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Colour\\MasteringMetadata\\LuminanceMin)"

id: "0x55DA"

maxOccurs: "1"

range: ">= 0x0p+0"

type: "float"

minver: "4"

documentation: Minimum luminance. Represented in candelas per square meter (cd/m<sup>2</sup>).

## 8.2.135. Projection Element

name: "Projection"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Projection)"

id: "0x7670"

maxOccurs: "1"

type: "master"

minver: "4"

documentation: Describes the video projection details. Used to render spherical and VR videos.

## 8.2.136. ProjectionType Element

name: "ProjectionType"

path:  
"1\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Projection\\ProjectionType)"

id: "0x7671"

minOccurs: "1"



maxOccurs: "1"  
range: "0-3"  
default: "0"  
type: "uinteger"  
minver: "4"  
documentation: Describes the projection used for this video track.

#### 8.2.137. ProjectionPrivate Element

name: "ProjectionPrivate"  
path:  
"0\*1(\\Segment\\Tracks\\TrackEntry\\Video\\Projection\\ProjectionPrivate)"  
id: "0x7672"  
maxOccurs: "1"  
type: "binary"  
minver: "4"  
documentation: Private data that only applies to a specific projection. Semantics: 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'). Note: 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.

#### 8.2.138. ProjectionPoseYaw Element

name: "ProjectionPoseYaw"



path:  
"1\*1(\Segment\Tracks\TrackEntry\Video\Projection\ProjectionPoseYaw)"  
  
id: "0x7673"  
  
minOccurs: "1"  
  
maxOccurs: "1"  
  
default: "0x0p+0"  
  
type: "float"  
  
minver: "4"  
  
documentation: Specifies a yaw rotation to the  
projection.SemanticsValue 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 field should be in the -180 to 180 degree range.

#### 8.2.139. ProjectionPosePitch Element

name: "ProjectionPosePitch"  
  
path: "1\*1(\Segment\Tracks\TrackEntry\Video\Projection\ProjectionPose  
Pitch)"  
  
id: "0x7674"  
  
minOccurs: "1"  
  
maxOccurs: "1"  
  
default: "0x0p+0"  
  
type: "float"  
  
minver: "4"  
  
documentation: Specifies a pitch rotation to the  
projection.SemanticsValue 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 field should be in  
the -90 to 90 degree range.



## 8.2.140. ProjectionPoseRoll Element

name: "ProjectionPoseRoll"

path:  
"1\*1(\Segment\Tracks\TrackEntry\Video\Projection\ProjectionPoseRoll)"

id: "0x7675"

minOccurs: "1"

maxOccurs: "1"

default: "0x0p+0"

type: "float"

minver: "4"

documentation: Specifies a roll rotation to the projection.SemanticsValue 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 field should be in the -180 to 180 degree range.

## 8.2.141. Audio Element

name: "Audio"

path: "0\*1(\Segment\Tracks\TrackEntry\Audio)"

id: "0xE1"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: Audio settings.

## 8.2.142. SamplingFrequency Element

name: "SamplingFrequency"

path: "1\*1(\Segment\Tracks\TrackEntry\Audio\SamplingFrequency)"

id: "0xB5"



minOccurs: "1"  
maxOccurs: "1"  
range: "> 0x0p+0"  
default: "0x1.f4p+12"  
type: "float"  
minver: "1"  
documentation: Sampling frequency in Hz.

#### 8.2.143. OutputSamplingFrequency Element

name: "OutputSamplingFrequency"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Audio\\OutputSamplingFrequency)"  
id: "0x78B5"  
maxOccurs: "1"  
range: "> 0x0p+0"  
default: "SamplingFrequency"  
type: "float"  
minver: "1"  
documentation: Real output sampling frequency in Hz (used for SBR techniques).

#### 8.2.144. Channels Element

name: "Channels"  
path: "1\*1(\\Segment\\Tracks\\TrackEntry\\Audio\\Channels)"  
id: "0x9F"  
minOccurs: "1"  
maxOccurs: "1"  
range: "not 0"



default: "1"  
type: "uinteger"  
minver: "1"  
documentation: Numbers of channels in the track.

#### 8.2.145. ChannelPositions Element

name: "ChannelPositions"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Audio\\ChannelPositions)"  
id: "0x7D7B"  
maxOccurs: "1"  
type: "binary"  
minver: "0"  
maxver: "0"  
documentation: Table of horizontal angles for each successive channel, see appendix.

#### 8.2.146. BitDepth Element

name: "BitDepth"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\Audio\\BitDepth)"  
id: "0x6264"  
maxOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "1"  
documentation: Bits per sample, mostly used for PCM.



## 8.2.147. TrackOperation Element

name: "TrackOperation"

path: "0\*1(\Segment\Tracks\TrackEntry\TrackOperation)"

id: "0xE2"

maxOccurs: "1"

type: "master"

minver: "3"

documentation: Operation that needs to be applied on tracks to create this virtual track. For more details look at the Specification Notes on the subject.

## 8.2.148. TrackCombinePlanes Element

name: "TrackCombinePlanes"

path:  
"0\*1(\Segment\Tracks\TrackEntry\TrackOperation\TrackCombinePlanes)"

id: "0xE3"

maxOccurs: "1"

type: "master"

minver: "3"

documentation: Contains the list of all video plane tracks that need to be combined to create this 3D track

## 8.2.149. TrackPlane Element

name: "TrackPlane"

path: "1\*(\Segment\Tracks\TrackEntry\TrackOperation\TrackCombinePlanes\TrackPlane)"

id: "0xE4"

minOccurs: "1"

type: "master"



minver: "3"

documentation: Contains a video plane track that need to be combined to create this 3D track

#### 8.2.150. TrackPlaneUID Element

name: "TrackPlaneUID"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\TrackOperation\\TrackCombinePlanes\\TrackPlane\\TrackPlaneUID) "

id: "0xE5"

minOccurs: "1"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "3"

documentation: The trackUID number of the track representing the plane.

#### 8.2.151. TrackPlaneType Element

name: "TrackPlaneType"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\TrackOperation\\TrackCombinePlanes\\TrackPlane\\TrackPlaneType) "

id: "0xE6"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "3"

documentation: The kind of plane this track corresponds to.



## 8.2.152. TrackJoinBlocks Element

name: "TrackJoinBlocks"

path:  
"0\*1(\Segment\Tracks\TrackEntry\TrackOperation\TrackJoinBlocks)"

id: "0xE9"

maxOccurs: "1"

type: "master"

minver: "3"

documentation: Contains the list of all tracks whose Blocks need to be combined to create this virtual track

## 8.2.153. TrackJoinUID Element

name: "TrackJoinUID"

path: "1\*(\Segment\Tracks\TrackEntry\TrackOperation\TrackJoinBlocks\TrackJoinUID)"

id: "0xED"

minOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "3"

documentation: The trackUID number of a track whose blocks are used to create this virtual track.

## 8.2.154. TrickTrackUID Element

name: "TrickTrackUID"

path: "0\*1(\Segment\Tracks\TrackEntry\TrickTrackUID)"

id: "0xC0"

maxOccurs: "1"



```
type: "uinteger"
minver: "0"
maxver: "0"
documentation: DivX trick track extensions
```

#### 8.2.155. TrickTrackSegmentUID Element

```
name: "TrickTrackSegmentUID"
path: "0*1(\\Segment\\Tracks\\TrackEntry\\TrickTrackSegmentUID)"
id: "0xC1"
maxOccurs: "1"
size: "16"
type: "binary"
minver: "0"
maxver: "0"
documentation: DivX trick track extensions
```

#### 8.2.156. TrickTrackFlag Element

```
name: "TrickTrackFlag"
path: "0*1(\\Segment\\Tracks\\TrackEntry\\TrickTrackFlag)"
id: "0xC6"
maxOccurs: "1"
default: "0"
type: "uinteger"
minver: "0"
maxver: "0"
documentation: DivX trick track extensions
```



## 8.2.157. TrickMasterTrackUID Element

name: "TrickMasterTrackUID"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\TrickMasterTrackUID)"  
id: "0xC7"  
maxOccurs: "1"  
type: "uinteger"  
minver: "0"  
maxver: "0"  
documentation: DivX trick track extensions

## 8.2.158. TrickMasterTrackSegmentUID Element

name: "TrickMasterTrackSegmentUID"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\TrickMasterTrackSegmentUID)"  
id: "0xC4"  
maxOccurs: "1"  
size: "16"  
type: "binary"  
minver: "0"  
maxver: "0"  
documentation: DivX trick track extensions

## 8.2.159. ContentEncodings Element

name: "ContentEncodings"  
path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings)"  
id: "0x6D80"  
maxOccurs: "1"



type: "master"

minver: "1"

documentation: Settings for several content encoding mechanisms like compression or encryption.

#### 8.2.160. ContentEncoding Element

name: "ContentEncoding"

path:

"1\*(\Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding)"

id: "0x6240"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Settings for one content encoding like compression or encryption.

#### 8.2.161. ContentEncodingOrder Element

name: "ContentEncodingOrder"

path: "1\*1(\Segment\Tracks\TrackEntry\ContentEncodings\ContentEncoding\ContentEncodingOrder)"

id: "0x5031"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: Tells when this modification was used during encoding/muxing starting with 0 and counting upwards. The decoder/demuxer has to start with the highest order number it finds and work its way



down. This value has to be unique over all ContentEncodingOrder Elements in the Segment.

#### 8.2.162. ContentEncodingScope Element

name: "ContentEncodingScope"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncodingScope)"

id: "0x5032"

minOccurs: "1"

maxOccurs: "1"

range: "not 0"

default: "1"

type: "uinteger"

minver: "1"

documentation: A bit field that describes which Elements have been modified in this way. Values (big endian) can be OR'ed. Possible values: 1 - all frame contents, 2 - the track's private data, 4 - the next ContentEncoding (next ContentEncodingOrder. Either the data inside ContentCompression and/or ContentEncryption)

#### 8.2.163. ContentEncodingType Element

name: "ContentEncodingType"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncodingType)"

id: "0x5033"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"



documentation: A value describing what kind of transformation has been done. Possible values: 0 - compression, 1 - encryption

#### 8.2.164. ContentCompression Element

name: "ContentCompression"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentCompression)"

id: "0x5034"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: 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.

#### 8.2.165. ContentCompAlgo Element

name: "ContentCompAlgo"

path: "1\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentCompression\\ContentCompAlgo)"

id: "0x4254"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The compression algorithm used. Algorithms that have been specified so far are: 0 - zlib, 1 - bzlib, 2 - lzolx 3 - Header Stripping



## 8.2.166. ContentCompSettings Element

name: "ContentCompSettings"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentCompression\\ContentCompSettings)"

id: "0x4255"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: 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.

## 8.2.167. ContentEncryption Element

name: "ContentEncryption"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption)"

id: "0x5035"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: Settings describing the encryption used. This Element MUST be present if the value of ContentEncodingType is 1 and absent otherwise.

## 8.2.168. ContentEncAlgo Element

name: "ContentEncAlgo"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption\\ContentEncAlgo)"

id: "0x47E1"

maxOccurs: "1"



default: "0"

type: "uinteger"

minver: "1"

documentation: The encryption algorithm used. The value '0' means that the contents have not been encrypted but only signed. Predefined values: 1 - DES, 2 - 3DES, 3 - Twofish, 4 - Blowfish, 5 - AES

#### 8.2.169. ContentEncKeyID Element

name: "ContentEncKeyID"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption\\ContentEncKeyID)"

id: "0x47E2"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: For public key algorithms this is the ID of the public key the the data was encrypted with.

#### 8.2.170. ContentSignature Element

name: "ContentSignature"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption\\ContentSignature)"

id: "0x47E3"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: A cryptographic signature of the contents.



## 8.2.171. ContentSigKeyID Element

name: "ContentSigKeyID"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption\\ContentSigKeyID)"

id: "0x47E4"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: This is the ID of the private key the data was signed with.

## 8.2.172. ContentSigAlgo Element

name: "ContentSigAlgo"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption\\ContentSigAlgo)"

id: "0x47E5"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The algorithm used for the signature. A value of '0' means that the contents have not been signed but only encrypted.  
Predefined values: 1 - RSA

## 8.2.173. ContentSigHashAlgo Element

name: "ContentSigHashAlgo"

path: "0\*1(\\Segment\\Tracks\\TrackEntry\\ContentEncodings\\ContentEncoding\\ContentEncryption\\ContentSigHashAlgo)"

id: "0x47E6"



maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: The hash algorithm used for the signature. A value of '0' means that the contents have not been signed but only encrypted. Predefined values: 1 - SHA1-160 2 - MD5

#### 8.2.174. Cues Element

name: "Cues"

path: "0\*1(\\Segment\\Cues)"

id: "0x1C53BB6B"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: A Top-Level Element to speed seeking access. All entries are local to the Segment. This Element SHOULD be mandatory for non "live" streams.

#### 8.2.175. CuePoint Element

name: "CuePoint"

path: "1\*(\\Segment\\Cues\\CuePoint)"

id: "0xBB"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Contains all information relative to a seek point in the Segment.



## 8.2.176. CueTime Element

name: "CueTime"

path: "1\*1(\Segment\Cues\CuePoint\CueTime)"

id: "0xB3"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: Absolute timestamp according to the Segment time base.

## 8.2.177. CueTrackPositions Element

name: "CueTrackPositions"

path: "1\*(\Segment\Cues\CuePoint\CueTrackPositions)"

id: "0xB7"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Contain positions for different tracks corresponding to the timestamp.

## 8.2.178. CueTrack Element

name: "CueTrack"

path: "1\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueTrack)"

id: "0xF7"

minOccurs: "1"

maxOccurs: "1"

range: "not 0"



type: "uinteger"

minver: "1"

documentation: The track for which a position is given.

#### 8.2.179. CueClusterPosition Element

name: "CueClusterPosition"

path:  
"1\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueClusterPosition)"

id: "0xF1"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: The Segment Position of the Cluster containing the associated Block.

#### 8.2.180. CueRelativePosition Element

name: "CueRelativePosition"

path:  
"0\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueRelativePosition)"

id: "0xF0"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: The relative position of the referenced block inside the cluster with 0 being the first possible position for an Element inside that cluster.



## 8.2.181. CueDuration Element

name: "CueDuration"

path: "0\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueDuration)"

id: "0xB2"

maxOccurs: "1"

type: "uinteger"

minver: "4"

documentation: The duration of the block according to the Segment time base. If missing the track's DefaultDuration does not apply and no duration information is available in terms of the cues.

## 8.2.182. CueBlockNumber Element

name: "CueBlockNumber"

path: "0\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueBlockNumber)"

id: "0x5378"

maxOccurs: "1"

range: "not 0"

default: "1"

type: "uinteger"

minver: "1"

documentation: Number of the Block in the specified Cluster.

## 8.2.183. CueCodecState Element

name: "CueCodecState"

path: "0\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueCodecState)"

id: "0xEA"

maxOccurs: "1"



default: "0"

type: "uinteger"

minver: "2"

documentation: The Segment Position of the Codec State corresponding to this Cue Element. 0 means that the data is taken from the initial Track Entry.

#### 8.2.184. CueReference Element

name: "CueReference"

path: "0\*(\Segment\Cues\CuePoint\CueTrackPositions\CueReference)"

id: "0xDB"

type: "master"

minver: "2"

documentation: The Clusters containing the referenced Blocks.

#### 8.2.185. CueRefTime Element

name: "CueRefTime"

path: "1\*1(\Segment\Cues\CuePoint\CueTrackPositions\CueReference\CueRefTime)"

id: "0x96"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "2"

documentation: Timestamp of the referenced Block.

#### 8.2.186. CueRefCluster Element

name: "CueRefCluster"



path: "1\*1(\\Segment\\Cues\\CuePoint\\CueTrackPositions\\CueReference\\CueRefCluster)"

id: "0x97"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: The Segment Position of the Cluster containing the referenced Block.

#### 8.2.187. CueRefNumber Element

name: "CueRefNumber"

path: "0\*1(\\Segment\\Cues\\CuePoint\\CueTrackPositions\\CueReference\\CueRefNumber)"

id: "0x535F"

maxOccurs: "1"

range: "not 0"

default: "1"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: Number of the referenced Block of Track X in the specified Cluster.

#### 8.2.188. CueRefCodecState Element

name: "CueRefCodecState"

path: "0\*1(\\Segment\\Cues\\CuePoint\\CueTrackPositions\\CueReference\\CueRefCodecState)"



id: "0xEB"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "0"

maxver: "0"

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.

#### 8.2.189. Attachments Element

name: "Attachments"

path: "0\*1(\\Segment\\Attachments)"

id: "0x1941A469"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: Contain attached files.

#### 8.2.190. AttachedFile Element

name: "AttachedFile"

path: "1\*(\\Segment\\Attachments\\AttachedFile)"

id: "0x61A7"

minOccurs: "1"

type: "master"

minver: "1"

documentation: An attached file.



## 8.2.191. FileDescription Element

name: "FileDescription"  
path: "0\*1(\\Segment\\Attachments\\AttachedFile\\FileDescription)"  
id: "0x467E"  
maxOccurs: "1"  
type: "utf-8"  
minver: "1"  
documentation: A human-friendly name for the attached file.

## 8.2.192. FileName Element

name: "FileName"  
path: "1\*1(\\Segment\\Attachments\\AttachedFile\\FileName)"  
id: "0x466E"  
minOccurs: "1"  
maxOccurs: "1"  
type: "utf-8"  
minver: "1"  
documentation: Filename of the attached file.

## 8.2.193. FileMimeType Element

name: "FileMimeType"  
path: "1\*1(\\Segment\\Attachments\\AttachedFile\\FileMimeType)"  
id: "0x4660"  
minOccurs: "1"  
maxOccurs: "1"  
type: "string"



minver: "1"

documentation: MIME type of the file.

#### 8.2.194. FileData Element

name: "FileData"

path: "1\*1(\\Segment\\Attachments\\AttachedFile\\FileData)"

id: "0x465C"

minOccurs: "1"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: The data of the file.

#### 8.2.195. FileUID Element

name: "FileUID"

path: "1\*1(\\Segment\\Attachments\\AttachedFile\\FileUID)"

id: "0x46AE"

minOccurs: "1"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "1"

documentation: Unique ID representing the file, as random as possible.

#### 8.2.196. FileReferral Element

name: "FileReferral"

path: "0\*1(\\Segment\\Attachments\\AttachedFile\\FileReferral)"



id: "0x4675"

maxOccurs: "1"

type: "binary"

minver: "0"

maxver: "0"

documentation: A binary value that a track/codec can refer to when the attachment is needed.

#### 8.2.197. FileUsedStartTime Element

name: "FileUsedStartTime"

path: "0\*1(\\Segment\\Attachments\\AttachedFile\\FileUsedStartTime)"

id: "0x4661"

maxOccurs: "1"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: DivX font extension

#### 8.2.198. FileUsedEndTime Element

name: "FileUsedEndTime"

path: "0\*1(\\Segment\\Attachments\\AttachedFile\\FileUsedEndTime)"

id: "0x4662"

maxOccurs: "1"

type: "uinteger"

minver: "0"

maxver: "0"

documentation: DivX font extension



## 8.2.199. Chapters Element

name: "Chapters"

path: "0\*1(\Segment\Chapters)"

id: "0x1043A770"

maxOccurs: "1"

type: "master"

minver: "1"

documentation: A system to define basic menus and partition data.  
For more detailed information, look at the Chapters Explanation.

## 8.2.200. EditionEntry Element

name: "EditionEntry"

path: "1\*(\Segment\Chapters\EditionEntry)"

id: "0x45B9"

minOccurs: "1"

type: "master"

minver: "1"

documentation: Contains all information about a Segment edition.

## 8.2.201. EditionUID Element

name: "EditionUID"

path: "0\*1(\Segment\Chapters\EditionEntry\EditionUID)"

id: "0x45BC"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "1"



documentation: A unique ID to identify the edition. It's useful for tagging an edition.

#### 8.2.202. EditionFlagHidden Element

name: "EditionFlagHidden"

path: "1\*1(\Segment\Chapters\EditionEntry\EditionFlagHidden)"

id: "0x45BD"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"

default: "0"

type: "uinteger"

minver: "1"

documentation: If an edition is hidden (1), it SHOULD NOT be available to the user interface (but still to Control Tracks; see flag notes). (1 bit)

#### 8.2.203. EditionFlagDefault Element

name: "EditionFlagDefault"

path: "1\*1(\Segment\Chapters\EditionEntry\EditionFlagDefault)"

id: "0x45DB"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"

default: "0"

type: "uinteger"

minver: "1"



documentation: If a flag is set (1) the edition SHOULD be used as the default one. (1 bit)

#### 8.2.204. EditionFlagOrdered Element

name: "EditionFlagOrdered"

path: "0\*1(\\Segment\\Chapters\\EditionEntry\\EditionFlagOrdered)"

id: "0x45DD"

maxOccurs: "1"

range: "0-1"

default: "0"

type: "uinteger"

minver: "1"

documentation: Specify if the chapters can be defined multiple times and the order to play them is enforced. (1 bit)

#### 8.2.205. ChapterAtom Element

name: "ChapterAtom"

path: "1\*(\\Segment\\Chapters\\EditionEntry(1\*(\\ChapterAtom)))"

id: "0xB6"

minOccurs: "1"

type: "master"

recursive: "1"

minver: "1"

documentation: Contains the atom information to use as the chapter atom (apply to all tracks).

#### 8.2.206. ChapterUID Element

name: "ChapterUID"

path: "1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterUID)"



id: "0x73C4"  
minOccurs: "1"  
maxOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "1"  
documentation: A unique ID to identify the Chapter.

#### 8.2.207. ChapterStringUID Element

name: "ChapterStringUID"  
path:  
"0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterStringUID)"  
id: "0x5654"  
maxOccurs: "1"  
type: "utf-8"  
minver: "3"  
documentation: A unique string ID to identify the Chapter. Use for  
WebVTT cue identifier storage.

#### 8.2.208. ChapterTimeStart Element

name: "ChapterTimeStart"  
path:  
"1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterTimeStart)"  
id: "0x91"  
minOccurs: "1"  
maxOccurs: "1"  
type: "uinteger"  
minver: "1"



documentation: Timestamp of the start of Chapter (not scaled).

#### 8.2.209. ChapterTimeEnd Element

name: "ChapterTimeEnd"

path:

"0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterTimeEnd)"

id: "0x92"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: Timestamp of the end of Chapter (timestamp excluded, not scaled).

#### 8.2.210. ChapterFlagHidden Element

name: "ChapterFlagHidden"

path:

"1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterFlagHidden)"

id: "0x98"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"

default: "0"

type: "uinteger"

minver: "1"

documentation: If a chapter is hidden (1), it SHOULD NOT be available to the user interface (but still to Control Tracks; see flag notes). (1 bit)



## 8.2.211. ChapterFlagEnabled Element

name: "ChapterFlagEnabled"

path:

"1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterFlagEnabled)"

id: "0x4598"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"

default: "1"

type: "uinteger"

minver: "1"

documentation: Specify whether 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 flag notes). (1 bit)

## 8.2.212. ChapterSegmentUID Element

name: "ChapterSegmentUID"

path:

"0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterSegmentUID)"

id: "0x6E67"

maxOccurs: "1"

range: ">0"

size: "16"

type: "binary"

minver: "1"

documentation: The SegmentUID of another Segment to play during this chapter.



usage notes: ChapterSegmentUID is mandatory if ChapterSegmentEditionUID is used.

#### 8.2.213. ChapterSegmentEditionUID Element

name: "ChapterSegmentEditionUID"

path: "0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterSegmentEditionUID)"

id: "0x6EBC"

maxOccurs: "1"

range: "not 0"

type: "uinteger"

minver: "1"

documentation: The EditionUID to play from the Segment linked in ChapterSegmentUID. If ChapterSegmentEditionUID is undeclared then no Edition of the linked Segment is used.

#### 8.2.214. ChapterPhysicalEquiv Element

name: "ChapterPhysicalEquiv"

path: "0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterPhysicalEquiv)"

id: "0x63C3"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: Specify the physical equivalent of this ChapterAtom like "DVD" (60) or "SIDE" (50), see complete list of values.

#### 8.2.215. ChapterTrack Element

name: "ChapterTrack"

path: "0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterTrack)"



id: "0x8F"  
maxOccurs: "1"  
type: "master"  
minver: "1"  
documentation: List of tracks on which the chapter applies. If this Element is not present, all tracks apply

#### 8.2.216. ChapterTrackNumber Element

name: "ChapterTrackNumber"  
path: "1\*(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterTrack\\ChapterTrackNumber)"  
id: "0x89"  
minOccurs: "1"  
range: "not 0"  
type: "uinteger"  
minver: "1"  
documentation: UID of the Track to apply this chapter too. 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.

#### 8.2.217. ChapterDisplay Element

name: "ChapterDisplay"  
path: "0\*(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterDisplay)"  
id: "0x80"  
type: "master"  
minver: "1"  
documentation: Contains all possible strings to use for the chapter display.



## 8.2.218. ChapString Element

name: "ChapString"

path: "1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterDisplay\\ChapString)"

id: "0x85"

minOccurs: "1"

maxOccurs: "1"

type: "utf-8"

minver: "1"

documentation: Contains the string to use as the chapter atom.

## 8.2.219. ChapLanguage Element

name: "ChapLanguage"

path: "1\*(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterDisplay\\ChapLanguage)"

id: "0x437C"

minOccurs: "1"

default: "eng"

type: "string"

minver: "1"

documentation: The languages corresponding to the string, in the bibliographic ISO-639-2 form. This Element MUST be ignored if the ChapLanguageIETF Element is used within the same ChapterDisplay Element.

## 8.2.220. ChapLanguageIETF Element

name: "ChapLanguageIETF"

path: "0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapterDisplay\\ChapLanguageIETF)"



id: "0x437D"

maxOccurs: "1"

type: "string"

minver: "4"

documentation: Specifies the language used in the ChapString according to BCP 47 and using the IANA Language Subtag Registry. If this Element is used, then any ChapLanguage Elements used in the same ChapterDisplay MUST be ignored.

#### 8.2.221. ChapCountry Element

name: "ChapCountry"

path: "0\*(\Segment\Chapters\EditionEntry\ChapterAtom\ChapterDisplay\ChapCountry)"

id: "0x437E"

type: "string"

minver: "1"

documentation: The countries corresponding to the string, same 2 octets as in Internet domains. This Element MUST be ignored if the ChapLanguageIETF Element is used within the same ChapterDisplay Element.

#### 8.2.222. ChapProcess Element

name: "ChapProcess"

path: "0\*(\Segment\Chapters\EditionEntry\ChapterAtom\ChapProcess)"

id: "0x6944"

type: "master"

minver: "1"

documentation: Contains all the commands associated to the Atom.



## 8.2.223. ChapProcessCodecID Element

name: "ChapProcessCodecID"

path: "1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapProcess\\ChapProcessCodecID)"

id: "0x6955"

minOccurs: "1"

maxOccurs: "1"

default: "0"

type: "uinteger"

minver: "1"

documentation: Contains the type of the codec used for the processing. A value of 0 means native Matroska processing (to be defined), a value of 1 means the DVD command set is used. More codec IDs can be added later.

## 8.2.224. ChapProcessPrivate Element

name: "ChapProcessPrivate"

path: "0\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapProcess\\ChapProcessPrivate)"

id: "0x450D"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: Some optional data attached to the ChapProcessCodecID information. For ChapProcessCodecID = 1, it is the "DVD level" equivalent.

## 8.2.225. ChapProcessCommand Element

name: "ChapProcessCommand"



path: "0\*(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapProcess\\ChapProcessCommand)"

id: "0x6911"

type: "master"

minver: "1"

documentation: Contains all the commands associated to the Atom.

#### 8.2.226. ChapProcessTime Element

name: "ChapProcessTime"

path: "1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapProcess\\ChapProcessCommand\\ChapProcessTime)"

id: "0x6922"

minOccurs: "1"

maxOccurs: "1"

type: "uinteger"

minver: "1"

documentation: Defines when the process command SHOULD be handled

#### 8.2.227. ChapProcessData Element

name: "ChapProcessData"

path: "1\*1(\\Segment\\Chapters\\EditionEntry\\ChapterAtom\\ChapProcess\\ChapProcessCommand\\ChapProcessData)"

id: "0x6933"

minOccurs: "1"

maxOccurs: "1"

type: "binary"

minver: "1"



documentation: 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.

#### 8.2.228. Tags Element

name: "Tags"

path: "0\*(\Segment\Tags)"

id: "0x1254C367"

type: "master"

minver: "1"

documentation: Element containing metadata describing Tracks, Editions, Chapters, Attachments, or the Segment as a whole. A list of valid tags can be found [here](#).

#### 8.2.229. Tag Element

name: "Tag"

path: "1\*(\Segment\Tags\Tag)"

id: "0x7373"

minOccurs: "1"

type: "master"

minver: "1"

documentation: A single metadata descriptor.

#### 8.2.230. Targets Element

name: "Targets"

path: "1\*1(\Segment\Tags\Tag\Targets)"

id: "0x63C0"

minOccurs: "1"

maxOccurs: "1"



type: "master"

minver: "1"

documentation: Specifies which other elements the metadata represented by the Tag applies to. If empty or not present, then the Tag describes everything in the Segment.

#### 8.2.231. TargetTypeValue Element

name: "TargetTypeValue"

path: "0\*1(\\Segment\\Tags\\Tag\\Targets\\TargetTypeValue)"

id: "0x68CA"

maxOccurs: "1"

default: "50"

type: "uinteger"

minver: "1"

documentation: A number to indicate the logical level of the target.

#### 8.2.232. TargetType Element

name: "TargetType"

path: "0\*1(\\Segment\\Tags\\Tag\\Targets\\TargetType)"

id: "0x63CA"

maxOccurs: "1"

type: "string"

minver: "1"

documentation: An informational string that can be used to display the logical level of the target like "ALBUM", "TRACK", "MOVIE", "CHAPTER", etc (see TargetType).



## 8.2.233. TagTrackUID Element

name: "TagTrackUID"

path: "0\*(\Segment\Tags\Tag\Targets\TagTrackUID)"

id: "0x63C5"

default: "0"

type: "uinteger"

minver: "1"

documentation: A unique ID to identify the Track(s) the tags belong to. If the value is 0 at this level, the tags apply to all tracks in the Segment.

## 8.2.234. TagEditionUID Element

name: "TagEditionUID"

path: "0\*(\Segment\Tags\Tag\Targets\TagEditionUID)"

id: "0x63C9"

default: "0"

type: "uinteger"

minver: "1"

documentation: A unique ID to identify the EditionEntry(s) the tags belong to. If the value is 0 at this level, the tags apply to all editions in the Segment.

## 8.2.235. TagChapterUID Element

name: "TagChapterUID"

path: "0\*(\Segment\Tags\Tag\Targets\TagChapterUID)"

id: "0x63C4"

default: "0"

type: "uinteger"



minver: "1"

documentation: A unique ID to identify the Chapter(s) the tags belong to. If the value is 0 at this level, the tags apply to all chapters in the Segment.

#### 8.2.236. TagAttachmentUID Element

name: "TagAttachmentUID"

path: "0\*(\Segment\Tags\Tag\Targets\TagAttachmentUID)"

id: "0x63C6"

default: "0"

type: "uinteger"

minver: "1"

documentation: A unique ID to identify the Attachment(s) the tags belong to. If the value is 0 at this level, the tags apply to all the attachments in the Segment.

#### 8.2.237. SimpleTag Element

name: "SimpleTag"

path: "1\*(\Segment\Tags\Tag(1\*(\SimpleTag)))"

id: "0x67C8"

minOccurs: "1"

type: "master"

recursive: "1"

minver: "1"

documentation: Contains general information about the target.

#### 8.2.238. TagName Element

name: "TagName"

path: "1\*1(\Segment\Tags\Tag\SimpleTag\TagName)"



id: "0x45A3"  
minOccurs: "1"  
maxOccurs: "1"  
type: "utf-8"  
minver: "1"  
documentation: The name of the Tag that is going to be stored.

#### 8.2.239. TagLanguage Element

name: "TagLanguage"  
path: "1\*1(\\Segment\\Tags\\Tag\\SimpleTag\\TagLanguage)"  
id: "0x447A"  
minOccurs: "1"  
maxOccurs: "1"  
default: "und"  
type: "string"  
minver: "1"  
documentation: Specifies the language of the tag specified, in the Matroska languages form. This Element MUST be ignored if the TagLanguageIETF Element is used within the same SimpleTag Element.

#### 8.2.240. TagLanguageIETF Element

name: "TagLanguageIETF"  
path: "0\*1(\\Segment\\Tags\\Tag\\SimpleTag\\TagLanguageIETF)"  
id: "0x447B"  
maxOccurs: "1"  
type: "string"  
minver: "4"



documentation: Specifies the language used in the TagString according to BCP 47 and using the IANA Language Subtag Registry. If this Element is used, then any TagLanguage Elements used in the same SimpleTag MUST be ignored.

#### 8.2.241. TagDefault Element

name: "TagDefault"

path: "1\*1(\\Segment\\Tags\\Tag\\SimpleTag\\TagDefault)"

id: "0x4484"

minOccurs: "1"

maxOccurs: "1"

range: "0-1"

default: "1"

type: "uinteger"

minver: "1"

documentation: A boolean value to indicate if this is the default/original language to use for the given tag.

#### 8.2.242. TagString Element

name: "TagString"

path: "0\*1(\\Segment\\Tags\\Tag\\SimpleTag\\TagString)"

id: "0x4487"

maxOccurs: "1"

type: "utf-8"

minver: "1"

documentation: The value of the Tag.



#### 8.2.243. TagBinary Element

name: "TagBinary"

path: "0\*1(\\Segment\\Tags\\Tag\\SimpleTag\\TagBinary)"

id: "0x4485"

maxOccurs: "1"

type: "binary"

minver: "1"

documentation: The values of the Tag if it is binary. Note that this cannot be used in the same SimpleTag as TagString.

### 9. Matroska Element Ordering Guidelines

Except for the "EBML Header" and the "CRC-32Element", the EBML specification does not require any particular storage order for "Elements". The Matroska 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.

#### 9.1. 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".

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; therefore, it is RECOMMENDED to use "Void Elements" as padding in between "Top-Level Elements".



## 9.2. 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". The Matroska specification recommends that "CRC-32 Elements" SHOULD NOT be used as an immediate "Child Element" of the "Segment Element"; however all "Top-Level Elements" of an "EBML Document" SHOULD include a "CRC-32 Element" as a "Child Element".

## 9.3. SeekHead

If used, the first "SeekHead Element" SHOULD 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".

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".

## 9.4. 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".

## 9.5. Info

The first "Info Element" SHOULD occur before the first "Tracks Element" and first "Cluster Element" except when referenced by a "SeekHead Element".



## 9.6. Chapters

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 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".

## 9.7. 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".

## 9.8. Tags

The "Tags Element" is most subject to changes after the file was originally created. For easier editing, the "Tags Element" SHOULD 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 size will only marginally change.

## 9.9. Optimum layout from a muxer

- o SeekHead
- o Info
- o Tracks
- o Chapters
- o Attachments
- o Tags



- o Clusters
- o Cues

#### 9.10. Optimum layout after editing tags

- o SeekHead
- o Info
- o Tracks
- o Chapters
- o Attachments
- o Void
- o Clusters
- o Cues
- o Tags

#### 9.11. Optimum layout with Cues at the front

- o SeekHead
- o Info
- o Tracks
- o Chapters
- o Attachments
- o Tags
- o Cues
- o Clusters

#### 9.12. Cluster Timecode

The "Timecode Element" MUST occur as in storage order before any "SimpleBlock", "BlockGroup", or "EncryptedBlock" within the "Cluster Element".



## 10. Chapters

### 10.1. Edition and Chapter Flags

#### 10.1.1. Chapter Flags

Two "Chapter Flags" are defined to describe the behavior of the "ChapterAtom Element": "ChapterFlagHidden" and "ChapterFlagEnabled".

If a "ChapterAtom Element" is the "Child Element" of another "ChapterAtom Element" with a "Chapter Flag" set to "true", then the "Child ChapterAtom Element" MUST be interpreted as having its same "Chapter Flag" set to "true". If a "ChapterAtom Element" is the "Child Element" of another "ChapterAtom Element" with a "Chapter Flag" set to "false" or if the "ChapterAtom Element" does not have a "ChapterAtom Element" as its "Parent Element", then it MUST be interpreted according to its own "Chapter Flag".

As an example, consider a "Parent ChapterAtom Element" that has its "ChapterFlagHidden" set to "true" and also contains two child "ChapterAtoms", the first with "ChapterFlagHidden" set to "true" and the second with "ChapterFlagHidden" either set to "false" or not present at all (in which case the default value of the Element applies, which is "false"). Since the parent "ChapterAtom" has its "ChapterFlagHidden" set to "true" then all of its children "ChapterAtoms" MUST also be interpreted as if their "ChapterFlagHidden" is also set to "true". However, 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".

#### 10.1.2. Edition Flags

Three "Edition Flags" are defined to describe the behavior of the "EditionEntry Element": "EditionFlagHidden", "EditionFlagDefault" and "EditionFlagOrdered".

##### 10.1.2.1. EditionFlagHidden

The "EditionFlagHidden Flag" behaves similarly to the "ChapterFlagHidden Flag": if "EditionFlagHidden" is set to "true", its "Child ChapterAtoms Elements" MUST also be interpreted as if their "ChapterFlagHidden" is also set to "true", regardless of their own "ChapterFlagHidden Flags". If "EditionFlagHidden" is toggled by a "Control Track" to "false", the "ChapterFlagHidden Flags" of the



"Child ChapterAtoms Elements" SHALL determine whether the "ChapterAtom" is hidden or not.

#### 10.1.2.2. 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" is the "Default Edition".

#### 10.1.2.3. 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 timecode marked in the "ChapterTimeStart Element" to the timecode marked in to "ChapterTimeEnd Element".

If the "EditionFlagOrdered Flag" is set to "false", "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". All these elements are now informational only.

The following list shows the different usage of "Chapter Elements" between an ordered and non-ordered "Edition".

Chapter elements / ordered Edition	False	True	ChapterUID	X	X
ChapterStringUID	X	X	ChapterTimeStart	X	X
ChapterTimeEnd	-	X	ChapterFlagHidden	X	X
ChapterFlagEnabled	X	X	ChapterSegmentUID	-	X
ChapterSegmentEditionUID	-	X	ChapterPhysicalEquiv	X	X
ChapterTrack	-	X	ChapterDisplay	X	X
ChapProcess	-	X			

Furthermore there are other EBML "Elements" which could be used if the "EditionFlagOrdered Flag" is set to "true".



```
Other elements / ordered Edition | False | True Info/SegmentFamily |
- | X Info/ChapterTranslate | - | X Track/TrackTranslate | - | X
```

These other "Elements" belong to the Matroska DVD menu system and are only used when the "ChapProcessCodecID Element" is set to 1.

#### 10.1.2.3.1. Ordered-Edition and Matroska Segment-Linking

- o Hard Linking: "Ordered-Chapters" supersedes the "Hard Linking".
- o Soft Linking: In this complex system "Ordered Chapters" are REQUIRED and a "Chapter CODEC" MUST interpret the "ChapProcess" of all chapters.
- o Medium Linking: "Ordered Chapters" are used in a normal way and can be combined with the "ChapterSegmentUID" element which establishes a link to another Matroska file/Segment.

See Section 23) for more information about "Hard Linking", "Soft Linking" and "Medium Linking".

#### 10.2. 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. The private data depend on the type of menu system (stored in ChapProcessPrivate), idem for the data in the chapters (stored in ChapProcessData).

##### 10.2.1. 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.



## 10.2.2. DVD menu (1)

This is the case when "ChapProcessCodecID" = 1. Each level of a chapter corresponds to a logical level in the DVD system 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 | - |
```

You can also recover whether 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)]
```

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

- o 0x00: entry only/basic PGC
- o 0x82: Title+Entry Menu (only found in the Video Manager domain)



- o 0x83: Root Menu (only found in the VTSM domain)
- o 0x84: Subpicture Menu (only found in the VTSM domain)
- o 0x85: Audio Menu (only found in the VTSM domain)
- o 0x86: Angle Menu (only found in the VTSM domain)
- o 0x87: Chapter Menu (only found in the VTSM domain)

The next 4 following octets correspond to the User Operation flags [17] 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-replica [18], where we got most of the info about it. You can also get information on DVD from the DVDinfo project [19].

### 10.3. 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.

- o 00000ms - 05000ms : Intro
- o 05000ms - 25000ms : Before the crime
- o 25000ms - 27500ms : The crime
- o 27500ms - 38000ms : The killer arrested
- o 38000ms - 43000ms : Credits

This would translate in the following matroska form :

```
<Chapters>
  <EditionEntry>
    <EditionUID>16603393396715046047</EditionUID>
    <ChapterAtom>
      <ChapterUID>1193046</ChapterUID>
      <ChapterTimeStart>0</ChapterTimeStart>
      <ChapterTimeEnd>5000000000</ChapterTimeEnd>
```



```
<ChapterDisplay>
  <ChapString>Intro</ChapString>
  <ChapLanguage>eng</ChapLanguage>
</ChapterDisplay>
<ChapterFlagHidden>0</ChapterFlagHidden>
<ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>2311527</ChapterUID>
  <ChapterTimeStart>5000000000</ChapterTimeStart>
  <ChapterTimeEnd>25000000000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>Before the crime</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterDisplay>
    <ChapString>Avant le crime</ChapString>
    <ChapLanguage>fra</ChapLanguage>
  </ChapterDisplay>
  <ChapterFlagHidden>0</ChapterFlagHidden>
  <ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>3430008</ChapterUID>
  <ChapterTimeStart>25000000000</ChapterTimeStart>
  <ChapterTimeEnd>27500000000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>The crime</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterDisplay>
    <ChapString>Le crime</ChapString>
    <ChapLanguage>fra</ChapLanguage>
  </ChapterDisplay>
  <ChapterFlagHidden>0</ChapterFlagHidden>
  <ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>4548489</ChapterUID>
  <ChapterTimeStart>27500000000</ChapterTimeStart>
  <ChapterTimeEnd>38000000000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>After the crime</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterDisplay>
    <ChapString>Apres le crime</ChapString>
    <ChapLanguage>fra</ChapLanguage>
```



```

        </ChapterDisplay>
        <ChapterFlagHidden>0</ChapterFlagHidden>
        <ChapterFlagEnabled>1</ChapterFlagEnabled>
    </ChapterAtom>
    <ChapterAtom>
        <ChapterUID>5666960</ChapterUID>
        <ChapterTimeStart>38000000000</ChapterTimeStart>
        <ChapterTimeEnd>43000000000</ChapterTimeEnd>
        <ChapterDisplay>
            <ChapString>Credits</ChapString>
            <ChapLanguage>eng</ChapLanguage>
        </ChapterDisplay>
        <ChapterDisplay>
            <ChapString>Generique</ChapString>
            <ChapLanguage>fra</ChapLanguage>
        </ChapterDisplay>
        <ChapterFlagHidden>0</ChapterFlagHidden>
        <ChapterFlagEnabled>1</ChapterFlagEnabled>
    </ChapterAtom>
    <EditionFlagDefault>0</EditionFlagDefault>
    <EditionFlagHidden>0</EditionFlagHidden>
</EditionEntry>
</Chapters>

```

#### 10.4. Example 2 : nested chapters

In this example an (existing) album is split into different chapters, and one of them contain another splitting.

##### 10.4.1. The Micronauts "Bleep To Bleep"

- o 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)
- o 12:30 - 19:38 : Bleeper\_O+2
- o 19:40 - 22:20 : Baby wants to bleep (pt.4)
- o 22:22 - 25:18 : Bleep to bleep
- o 25:20 - 33:35 : Baby wants to bleep (k)



- o 33:37 - 44:28 : Bleeper

```
<Chapters>
  <EditionEntry>
    <EditionUID>1281690858003401414</EditionUID>
    <ChapterAtom>
      <ChapterUID>1</ChapterUID>
      <ChapterTimeStart>0</ChapterTimeStart>
      <ChapterTimeEnd>748000000</ChapterTimeEnd>
      <ChapterDisplay>
        <ChapString>Baby wants to Bleep/Rock</ChapString>
        <ChapLanguage>eng</ChapLanguage>
      </ChapterDisplay>
    </ChapterAtom>
    <ChapterAtom>
      <ChapterUID>2</ChapterUID>
      <ChapterTimeStart>0</ChapterTimeStart>
      <ChapterTimeEnd>278000000</ChapterTimeEnd>
      <ChapterDisplay>
        <ChapString>Baby wants to bleep (pt.1)</ChapString>
        <ChapLanguage>eng</ChapLanguage>
      </ChapterDisplay>
      <ChapterFlagHidden>0</ChapterFlagHidden>
      <ChapterFlagEnabled>1</ChapterFlagEnabled>
    </ChapterAtom>
    <ChapterAtom>
      <ChapterUID>3</ChapterUID>
      <ChapterTimeStart>278000000</ChapterTimeStart>
      <ChapterTimeEnd>432000000</ChapterTimeEnd>
      <ChapterDisplay>
        <ChapString>Baby wants to rock</ChapString>
        <ChapLanguage>eng</ChapLanguage>
      </ChapterDisplay>
      <ChapterFlagHidden>0</ChapterFlagHidden>
      <ChapterFlagEnabled>1</ChapterFlagEnabled>
    </ChapterAtom>
    <ChapterAtom>
      <ChapterUID>4</ChapterUID>
      <ChapterTimeStart>432000000</ChapterTimeStart>
      <ChapterTimeEnd>633000000</ChapterTimeEnd>
      <ChapterDisplay>
        <ChapString>Baby wants to bleep (pt.2)</ChapString>
        <ChapLanguage>eng</ChapLanguage>
      </ChapterDisplay>
      <ChapterFlagHidden>0</ChapterFlagHidden>
      <ChapterFlagEnabled>1</ChapterFlagEnabled>
    </ChapterAtom>
    <ChapterAtom>
      <ChapterUID>5</ChapterUID>
```



```
<ChapterTimeStart>633000000</ChapterTimeStart>
<ChapterTimeEnd>748000000</ChapterTimeEnd>
<ChapterDisplay>
  <ChapString>Baby wants to bleep (pt.3)</ChapString>
  <ChapLanguage>eng</ChapLanguage>
</ChapterDisplay>
<ChapterFlagHidden>0</ChapterFlagHidden>
<ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterFlagHidden>0</ChapterFlagHidden>
<ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>6</ChapterUID>
  <ChapterTimeStart>750000000</ChapterTimeStart>
  <ChapterTimeEnd>1178500000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>Bleeper_0+2</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterFlagHidden>0</ChapterFlagHidden>
  <ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>7</ChapterUID>
  <ChapterTimeStart>1180500000</ChapterTimeStart>
  <ChapterTimeEnd>1340000000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>Baby wants to bleep (pt.4)</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterFlagHidden>0</ChapterFlagHidden>
  <ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>8</ChapterUID>
  <ChapterTimeStart>1342000000</ChapterTimeStart>
  <ChapterTimeEnd>1518000000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>Bleep to bleep</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterFlagHidden>0</ChapterFlagHidden>
  <ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>9</ChapterUID>
  <ChapterTimeStart>1520000000</ChapterTimeStart>
```



```
<ChapterTimeEnd>2015000000</ChapterTimeEnd>
<ChapterDisplay>
  <ChapString>Baby wants to bleep (k)</ChapString>
  <ChapLanguage>eng</ChapLanguage>
</ChapterDisplay>
<ChapterFlagHidden>0</ChapterFlagHidden>
<ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<ChapterAtom>
  <ChapterUID>10</ChapterUID>
  <ChapterTimeStart>2017000000</ChapterTimeStart>
  <ChapterTimeEnd>2668000000</ChapterTimeEnd>
  <ChapterDisplay>
    <ChapString>Bleeper</ChapString>
    <ChapLanguage>eng</ChapLanguage>
  </ChapterDisplay>
  <ChapterFlagHidden>0</ChapterFlagHidden>
  <ChapterFlagEnabled>1</ChapterFlagEnabled>
</ChapterAtom>
<EditionFlagDefault>0</EditionFlagDefault>
<EditionFlagHidden>0</EditionFlagHidden>
</EditionEntry>
</Chapters>
```

## 11. Attachments

### 11.1. Introduction

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".

### 11.2. 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 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

## 12. Cues

### 12.1. Introduction

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.

### 12.2. Recommendations

The following recommendations are provided to optimize Matroska performance.



- o Unless Matroska is used as a live stream, it SHOULD contain a "Cues Element".
- o For each video track, each keyframe SHOULD be referenced by a "CuePoint Element".
- o 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.
- o For each subtitle track present, each subtitle frame SHOULD be referenced by a "CuePoint Element" with a "CueDuration Element".
- o 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.
- o 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.
- o If a "CuePoint Element" references "Cluster Element" that includes a "CodecState Element", then that "CuePoint Element" MUST use a "CueCodecState Element".
- o "CuePoint Elements" SHOULD be numerically sorted in storage order by the value of the "CueTime Element".

### 13. Matroska Streaming

In Matroska, there are two kinds of streaming: file access and livestreaming.

#### 13.1. 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 following guidelines, 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.

### 13.2. 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 HTTP (or any other plain protocol based on TCP) 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 "MetaSeek", "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 "MetaSeek" list or a "Cues" list at the beginning of the stream, it SHOULD be considered non-seekable. Even though it is possible to seek blindly 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.

## 14. Menu Specifications

### 14.1. Introduction

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, when you have some screens to select the audio format, subtitles or scene selection.

#### 14.2. 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.

##### 14.2.1. 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 14.2.2).

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) 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.



#### 14.2.2. Playback features

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

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

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

#### 14.2.3. Player requirements

Some "Matroska Players" might not support the control track. That means 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" encounters 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.

#### 14.3. Working Graph

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



#### 14.4. Ideas

!!!! KNOW Where the main/audio/subs menu starts wherever we are (use chapters) !  
!!!

!!!! Keep in mind the state of the selected tracks of each kind (more than 1 for each possible) !!!!  
!!!! Order of blending !!!!  
!!!! What if a command is not supported by the player ? !!!!  
!!!! Track selection issue, only applies when 'quitting' the menu (but still possible to change live too) !!!!  
!!!! Allow to hide (not render) some parts of a movie for certain editions !!!!  
!!!! Get the parental level of the player (can be changed live) !!!!

#### 14.5. Data Structure

As a Matroska side project, the obvious choice for storing binary data is EBML.

#### 15. Unknown elements

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). For example, 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".

#### 16. Default Values

The default value of an "Element" is assumed when not present in the data stream. It is assumed only in the scope of its "Parent Element". For example, the "Language Element" is in the scope of the "Track Element". If the "Parent Element" is not present or assumed, then the "Child Element" cannot be assumed.

#### 17. 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, then the period between two successive fields at the output of the decoding process equals "DefaultDecodedFieldDuration".

For video sequences signaled as progressive, it is twice the value of "DefaultDecodedFieldDuration".

These values are valid at the end of the decoding process before post-processing (such as deinterlacing or inverse telecine) is applied.

Examples:

- o Blu-ray movie:  $1000000000\text{ns}/(48/1.001) = 20854167\text{ns}$
- o PAL broadcast/DVD:  $1000000000\text{ns}/(50/1.000) = 20000000\text{ns}$
- o N/ATSC broadcast:  $1000000000\text{ns}/(60/1.001) = 16683333\text{ns}$
- o hard-telecined DVD:  $1000000000\text{ns}/(60/1.001) = 16683333\text{ns}$  (60 encoded interlaced fields per second)
- o soft-telecined DVD:  $1000000000\text{ns}/(60/1.001) = 16683333\text{ns}$  (48 encoded interlaced fields per second, with "repeat\_first\_field = 1")

## 18. Encryption

Encryption in Matroska is designed in a very generic style to allow people to implement whatever form of encryption is best for them. It is possible to use the encryption framework in Matroska as a type of DRM (Digital Rights Management).

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".



## 19. Image 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.

## 20. Matroska versioning

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" MUST have a "DocTypeVersion" equal to or greater than 2. A file containing "CueRelativePosition" Elements 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.

"DocTypeVersion" MUST always be equal to or greater than "DocTypeReadVersion".

A reading application supporting Matroska version "V" MUST NOT refuse to read an application with "DocReadTypeVersion" equal to or lower



than "V" even if "DocTypeVersion" is greater than "V". See also the note about Section 15.

## 21. MIME Types

There is no IETF endorsed MIME type for Matroska files. These definitions can be used:

- o .mka : Matroska audio "audio/x-matroska"
- o .mkv : Matroska video "video/x-matroska"
- o .mk3d : Matroska 3D video "video/x-matroska-3d"

## 22. 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".

### 22.1. Segment Position Exception

"Elements" that are defined to store a "Segment Position" MAY define reserved values to indicate a special meaning.

### 22.2. 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".

	0										1										2									
	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0									
	+---																													
0	1A	45	DF	A3	8B	42	82	88	6D	61	74	72	6F	73	6B	61	18	53	80	67										
20	93	15	49	A9	66	8E	4D	80	84	69	65	74	66	57	41	84	69	65	74	66										



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".

## 23. Linked Segments

Matroska provides several methods to link two or many "Segment Elements" together to create a "Linked Segment". A "Linked Segment" is a set of multiple "Segments" related together into a single presentation by using Hard Linking, Medium Linking, or Soft Linking. All "Segments" within a "Linked Segment" MUST utilize the same track numbers and timescale. All "Segments" within a "Linked Segment" MUST be stored within the same directory. All "Segments" within a "Linked Segment" MUST store a "SegmentUID".

### 23.1. Hard Linking

Hard Linking (also called splitting) is the process of creating a "Linked Segment" by relating multiple "Segment Elements" using the "PrevUID" and "NextUID Elements". Within a "Linked Segment", the timestamps of each "Segment" MUST follow consecutively in linking order. With Hard Linking, the chapters of any "Segment" within the "Linked Segment" MUST only reference the current "Segment". With Hard Linking, the "NextUID" and "PrevUID" MUST reference the respective "SegmentUID" values of the next and previous "Segments". The first "Segment" of a "Linked Segment" MUST have a "NextUID Element" and MUST NOT have a "PrevUID Element". The last "Segment" of a "Linked Segment" MUST have a "PrevUID Element" and MUST NOT have a "NextUID Element". The middle "Segments" of a "Linked Segment" MUST have both a "NextUID Element" and a "PrevUID Element".

As an example, four "Segments" can be Hard Linked as a "Linked Segment" through cross-referencing each other with "SegmentUID", "PrevUID", and "NextUID", as in this table.



file name	SegmentUID	PrevUID	NextUID
"start.mkv"	"71000c23cd31099853fbc94dd984a5dd"	n/a	"a77b3598941cb803eac0fcdafe44fac9"
"middle.mkv"	"a77b3598941cb803eac0fcdafe44fac9"	"71000c23cd31099853fbc94dd984a5dd"	"6c92285fa6d3e827b198d120ea3ac674"
"end.mkv"	"6c92285fa6d3e827b198d120ea3ac674"	"a77b3598941cb803eac0fcdafe44fac9"	n/a

### 23.2. Medium Linking

Medium Linking creates relationships between "Segments" using Ordered Chapters and the "ChapterSegmentUID Element". A "Segment Edition" with Ordered Chapters MAY contain "Chapter Elements" that reference timestamp ranges from other "Segments". The "Segment" referenced by the Ordered Chapter via the "ChapterSegmentUID 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 the previous Ordered Chapters.

As an example a file named "intro.mkv" could have a "SegmentUID" of "0xb16a58609fc7e60653a60c984fc1lead". 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 "ChapterSegmentUID", "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 "ChapterSegmentUID" in an enabled "Edition" and present the reference content of the two "Segments" together.

### 23.3. Soft Linking

Soft Linking is used by codec chapters. They can reference another "Segment" and jump to that "Segment". The way the "Segments" are described are internal to the chapter codec and unknown to the Matroska level. But there are "Elements" within the "Info Element" (such as "ChapterTranslate") that can translate a value representing a "Segment" in the chapter codec and to the current "SegmentUID". All "Segments" that could be used in a "Linked Segment" in this way SHOULD be marked as members of the same family via the "SegmentFamily



Element", so that the "Matroska Player" can quickly switch from one to the other.

## 24. Track Flags

### 24.1. Default flag

The "default track" flag is a hint for a "Matroska Player" and SHOULD always be changeable by the user. If the user wants to see or hear a track of a certain kind (audio, video, subtitles) and hasn't chosen a specific track, the "Matroska Player" SHOULD use the first track of that kind whose "default track" flag is set to "1". If no such track is found then the first track of this kind SHOULD be chosen.

Only one track of a kind MAY have its "default track" flag set in a segment. If a track entry does not contain the "default track" flag element then its default value "1" is to be used.

### 24.2. Forced flag

The "forced" flag tells the "Matroska Player" that it MUST display/play this track or another track of the same kind that also has its "forced" flag set. When there are multiple "forced" tracks, the "Matroska Player" SHOULD determine the track based upon the language of the forced flag or use the default flag if no track matches the use languages. Another track of the same kind without the "forced" flag may be use simultaneously with the "forced" track (like DVD subtitles for example).

### 24.3. 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 sum 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 timecodes, 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.

#### 24.4. Overlay Track

Overlay tracks SHOULD be rendered in the same 'channel' as the track its linked to. When content is found in such a track, it SHOULD be played on the rendering channel instead of the original track.

#### 24.5. 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 colours 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 24.3.

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. "Matroska Readers" may support these legacy files by checking Matroska v2 or 0x53B9. The older values [20] were 0: mono, 1: right eye, 2: left eye, 3: both eyes.



## 25. Timecodes

### 25.1. Timecode Types

- o Absolute Timecode = Block+Cluster
- o Relative Timecode = Block
- o Scaled Timecode = Block+Cluster
- o Raw Timecode = (Block+Cluster)\*TimecodeScale\*TrackTimecodeScale

### 25.2. Block Timecodes

The "Block Element"'s timecode MUST be a signed integer that represents the "Raw Timecode" relative to the "Cluster"'s "Timecode Element", multiplied by the "TimecodeScale Element". See Section 25.4 for more information.

The "Block Element"'s timecode MUST be represented by a 16bit signed integer (sint16). The "Block"'s timecode has a range of -32768 to +32767 units. When using the default value of the "TimecodeScale Element", each integer represents lms. The maximum time span of "Block Elements" in a "Cluster" using the default "TimecodeScale Element" of lms is 65536ms.

If a "Cluster"'s "Timecode Element" is set to zero, it is possible to have "Block Elements" with a negative "Raw Timecode". "Block Elements" with a negative "Raw Timecode" are not valid.

### 25.3. Raw Timecode

The exact time of an object SHOULD be represented in nanoseconds. To find out a "Block"'s "Raw Timecode", you need the "Block"'s "Timecode Element", the "Cluster"'s "Timecode Element", and the "TimecodeScale Element".

### 25.4. TimecodeScale

The "TimecodeScale Element" is used to calculate the "Raw Timecode" of a "Block". The timecode is obtained by adding the "Block"'s timecode to the "Cluster"'s "Timecode Element", and then multiplying that result by the "TimecodeScale". The result will be the "Block"'s "Raw Timecode" in nanoseconds. The formula for this would look like:



```
(a + b) * c

a = 'Block''s Timecode
b = 'Cluster''s Timecode
c = 'TimeCodeScale'
```

For example, assume a "Cluster"'s "Timecode" has a value of 564264, the "Block" has a "Timecode" of 1233, and the "TimecodeScale Element" is the default of 1000000.

$$(1233 + 564264) * 1000000 = 565497000000$$

So, the "Block" in this example has a specific time of 565497000000 in nanoseconds. In milliseconds this would be 565497ms.

#### 25.5. TimecodeScale Rounding

Because the default value of "TimecodeScale" is 1000000, which makes each integer in the "Cluster" and "Block" "Timecode Elements" equal 1ms, this is the most commonly used. When dealing with audio, this causes inaccuracy when seeking. When the audio is combined with video, this is not an issue. For most cases, the the synch of audio to video does not need to be more than 1ms accurate. This becomes obvious when one considers that sound will take 2-3ms to travel a single meter, so distance from your speakers will have a greater effect on audio/visual synch than this.

However, when dealing with audio-only files, seeking accuracy can become critical. For instance, when storing a whole CD in a single track, a user will want to be able to seek to the exact sample that a song begins at. If seeking a few sample ahead or behind, a 'crack' or 'pop' may result as a few odd samples are rendered. Also, when performing precise editing, it may be very useful to have the audio accuracy down to a single sample.

When storing timecodes for an audio stream, the "TimecodeScale Element" SHOULD have an accuracy of at least that of the audio sample rate, otherwise there are rounding errors that prevent users from knowing the precise location of a sample. Here's how a program has to round each timecode in order to be able to recreate the sample number accurately.

Let's assume that the application has an audio track with a sample rate of 44100. As written above the "TimecodeScale" MUST have at least the accuracy of the sample rate itself:  $1000000000 / 44100 = 22675.7369614512$ . This value MUST always be truncated. Otherwise the accuracy will not suffice. So in this example the application will use 22675 for the "TimecodeScale". The application could even



use some lower value like 22674 which would allow it to be a little bit imprecise about the original timecodes. But more about that in a minute.

Next the application wants to write sample number 52340 and calculates the timecode. This is easy. In order to calculate the "Raw Timecode" in ns all it has to do is calculate `"Raw Timecode = round(1000000000 * sample_number / sample_rate)"`. Rounding at this stage is very important! The application might skip it if it choses a slightly smaller value for the "TimecodeScale" factor instead of the truncated one like shown above. Otherwise it has to round or the results won't be reversible. For our example we get `"Raw Timecode = round(1000000000 * 52340 / 44100) = round(1186848072.56236) = 1186848073"`.

The next step is to calculate the "Absolute Timecode" - that is the timecode that will be stored in the Matroska file. Here the application has to divide the "Raw Timecode" from the previous paragraph by the "TimecodeScale" factor and round the result: `"Absolute Timecode = round(Raw Timecode / TimecodeScale_factor)"` which will result in the following for our example: `"Absolute Timecode = round(1186848073 / 22675) = round(52341.7011245866) = 52342"`. This number is the one the application has to write to the file.

Now our file is complete, and we want to play it back with another application. Its task is to find out which sample the first application wrote into the file. So it starts reading the Matroska file and finds the "TimecodeScale" factor 22675 and the audio sample rate 44100. Later it finds a data block with the "Absolute Timecode" of 52342. But how does it get the sample number from these numbers?

First it has to calculate the "Raw Timecode" of the block it has just read. Here's no rounding involved, just an integer multiplication: `"Raw Timecode = Absolute Timecode * TimecodeScale_factor"`. In our example: `"Raw Timecode = 52342 * 22675 = 1186854850"`.

The conversion from the "Raw Timecode" to the sample number again requires rounding: `"sample_number = round(Raw Timecode * sample_rate / 1000000000)"`. In our example: `"sample_number = round(1186854850 * 44100 / 1000000000) = round(52340.298885) = 52340"`. This is exactly the sample number that the previous program started with.

Some general notes for a program:

1. Always calculate the timestamps / sample numbers with floating point numbers of at least 64bit precision (called 'double' in



most modern programming languages). If you're calculating with integers then make sure they're 64bit long, too.

2. Always round if you divide. Always! If you don't you'll end up with situations in which you have a timecode in the Matroska file that does not correspond to the sample number that it started with. Using a slightly lower timecode scale factor can help here in that it removes the need for proper rounding in the conversion from sample number to "Raw Timecode".

#### 25.6. TrackTimecodeScale

The "TrackTimecodeScale Element" is used align tracks that would otherwise be played at different speeds. An example of this would be if you have a film that was originally recorded at 24fps video. When playing this back through a PAL broadcasting system, it is standard to speed up the film to 25fps to match the 25fps display speed of the PAL broadcasting standard. However, when broadcasting the video through NTSC, it is typical to leave the film at its original speed. If you wanted to make a single file where there was one video stream, and an audio stream used from the PAL broadcast, as well as an audio stream used from the NTSC broadcast, you would have the problem that the PAL audio stream would be 1/24th faster than the NTSC audio stream, quickly leading to problems. It is possible to stretch out the PAL audio track and re-encode it at a slower speed, however when dealing with lossy audio codecs, this often results in a loss of audio quality and/or larger file sizes.

This is the type of problem that "TrackTimecodeScale" was designed to fix. Using it, the video can be played back at a speed that will synch with either the NTSC or the PAL audio stream, depending on which is being used for playback. To continue the above example:

```
Track 1: Video
Track 2: NTSC Audio
Track 3: PAL Audio
```

Because the NTSC track is at the original speed, it will used as the default value of 1.0 for its "TrackTimecodeScale". The video will also be aligned to the NTSC track with the default value of 1.0.

The "TrackTimecodeScale" value to use for the PAL track would be calculated by determining how much faster the PAL track is than the NTSC track. In this case, because we know the video for the NTSC audio is being played back at 24fps and the video for the PAL audio is being played back at 25fps, the calculation would be:

```
25/24 &#8776; 1.04166666666666666667
```



When writing a file that uses a non-default "TrackTimecodeScale", the values of the "Block"'s timecode are whatever they would be when normally storing the track with a default value for the "TrackTimecodeScale". However, the data is interleaved a little differently. Data SHOULD be interleaved by its Section 25.3 in the order handed back from the encoder. The "Raw Timecode" of a "Block" from a track using "TrackTimecodeScale" is calculated using:

$$\text{"(Block's Timecode + Cluster's Timecode) * TimecodeScale * TrackTimecodeScale"}$$

So, a Block from the PAL track above that had a Section 25.1 of 100 seconds would have a "Raw Timecode" of 104.66666667 seconds, and so would be stored in that part of the file.

When playing back a track using the "TrackTimecodeScale", if the track is being played by itself, there is no need to scale it. From the above example, when playing the Video with the NTSC Audio, neither are scaled. However, when playing back the Video with the PAL Audio, the timecodes from the PAL Audio track are scaled using the "TrackTimecodeScale", resulting in the video playing back in synch with the audio.

It would be possible for a "Matroska Player" to also adjust the audio's samplerate at the same time as adjusting the timecodes if you wanted to play the two audio streams synchronously. It would also be possible to adjust the video to match the audio's speed. However, for playback, the selected track(s) timecodes SHOULD be adjusted if they need to be scaled.

While the above example deals specifically with audio tracks, this element can be used to align video, audio, subtitles, or any other type of track contained in a Matroska file.

## 26. References

### 26.1. URIs

- [1] <http://mukoli.free.fr/mcf/mcf.html>
- [2] <https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown>
- [3] <https://datatracker.ietf.org/wg/cellar/charter/>
- [4] <https://matroska.org/files/matroska.pdf>
- [5] diagram.md



- [6] <https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown>
- [7] <https://github.com/Matroska-Org/foundation-source/blob/master/spectool/specdata.xml>
- [8] <https://tools.ietf.org/html/rfc2119>
- [9] <https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown>
- [10] <https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown#ebml-element-types>
- [11] <https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown#ebml-schema>
- [12] <https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown#structure>
- [13] [https://www.loc.gov/standards/iso639-2/php/English\\_list.php](https://www.loc.gov/standards/iso639-2/php/English_list.php)
- [14] <https://tools.ietf.org/html/bcp47>
- [15] <https://www.iana.org/domains/root/db>
- [16] <http://www.webmproject.org/docs/container/>
- [17] <http://dvd.sourceforge.net/dvdinfo/uops.html>
- [18] <http://www.dvd-replica.com/DVD/>
- [19] <http://dvd.sourceforge.net/dvdinfo/>
- [20] <http://www.matroska.org/node/1/revisions/74/view#StereoMode>

#### Authors' Addresses

Steve Lhomme

Email: [slhomme@matroska.org](mailto:slhomme@matroska.org)

Moritz Bunkus

Email: [moritz@bunkus.org](mailto:moritz@bunkus.org)



Dave Rice

Email: [dave@dericed.com](mailto:dave@dericed.com)



cellar  
Internet-Draft  
Intended status: Standards Track  
Expires: December 7, 2017

J. Coalson

Xiph.Org Foundation  
June 5, 2017

Free Lossless Audio Codec  
draft-xiph-cellar-flac-00

## Abstract

This document defines FLAC, which stands for Free Lossless Audio Codec, a free, open source codec for lossless audio compression and decompression.

## 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 <http://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 December 7, 2017.

## Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.



## Table of Contents

1. Introduction . . . . .	3
2. Acknowledgments . . . . .	3
3. Scope . . . . .	3
4. Architecture . . . . .	4
5. Definitions . . . . .	5
6. Blocking . . . . .	5
7. Interchannel Decorrelation . . . . .	6
8. Prediction . . . . .	6
9. Residual Coding . . . . .	7
10. Format . . . . .	8
10.1. Conventions . . . . .	11
10.2. STREAM . . . . .	11
10.3. METADATA_BLOCK . . . . .	12
10.4. METADATA_BLOCK_HEADER . . . . .	12
10.5. BLOCK_TYPE . . . . .	12
10.6. METADATA_BLOCK_DATA . . . . .	13
10.7. METADATA_BLOCK_STREAMINFO . . . . .	13
10.8. METADATA_BLOCK_PADDING . . . . .	14
10.9. METADATA_BLOCK_APPLICATION . . . . .	15
10.10. METADATA_BLOCK_SEEKTABLE . . . . .	15
10.11. SEEKPOINT . . . . .	15
10.12. METADATA_BLOCK_VORBIS_COMMENT . . . . .	16
10.13. METADATA_BLOCK_CUESHEET . . . . .	16
10.14. CUESHEET_TRACK . . . . .	18
10.15. CUESHEET_TRACK_INDEX . . . . .	20
10.16. METADATA_BLOCK_PICTURE . . . . .	20
10.17. PICTURE_TYPE . . . . .	20
10.18. FRAME . . . . .	21
10.19. FRAME_HEADER . . . . .	21
10.19.1. FRAME_HEADER_RESERVED . . . . .	22
10.19.2. BLOCKING_STRATEGY . . . . .	22
10.19.3. INTERCHANNEL_SAMPLE_BLOCK_SIZE . . . . .	23
10.19.4. SAMPLE_RATE . . . . .	23
10.19.5. CHANNEL_ASSIGNMENT . . . . .	24
10.19.6. SAMPLE_SIZE . . . . .	24
10.19.7. FRAME_HEADER_RESERVED2 . . . . .	25
10.19.8. CODED_NUMBER . . . . .	25
10.19.9. BLOCK_SIZE_INT . . . . .	25
10.19.10. SAMPLE_RATE_INT . . . . .	25
10.19.11. FRAME_CRC . . . . .	25
10.20. FRAME_FOOTER . . . . .	25
10.21. SUBFRAME . . . . .	26
10.22. SUBFRAME_HEADER . . . . .	26
10.22.1. SUBFRAME_TYPE . . . . .	26
10.22.2. WASTED_BITS_PER_SAMPLE_FLAG . . . . .	26
10.23. SUBFRAME_CONSTANT . . . . .	27



10.24. SUBFRAME_FIXED . . . . .	27
10.25. SUBFRAME_LPC . . . . .	27
10.26. SUBFRAME_VERBATIM . . . . .	28
10.27. RESIDUAL . . . . .	28
10.27.1. RESIDUAL_CODING_METHOD . . . . .	28
10.27.2. RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB . . . . .	28
10.27.3. RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB2 . . . . .	29
10.27.4. ENCODED RESIDUAL . . . . .	30
11.1. URIs . . . . .	30
Authors' Addresses . . . . .	31

## 1. Introduction

This is a detailed description of the FLAC format. There is also a companion document that describes FLAC-to-Ogg mapping [1].

For a user-oriented overview, see About the FLAC Format [2].

## 2. Acknowledgments

FLAC owes much to the many people who have advanced the audio compression field so freely. For instance: - A. J. Robinson [3] for his work on Shorten [4]; his paper 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 Exponential-Golomb coding used in Shorten. - S. W. Golomb [5] and Robert F. Rice; their universal codes are used by FLAC's entropy coder. - N. Levinson and J. Durbin; the reference encoder uses an algorithm developed and refined by them for determining the LPC coefficients from the autocorrelation coefficients. - And of course, Claude Shannon [6]

## 3. Scope

It is a known fact that no algorithm can losslessly compress all possible input, so most compressors restrict themselves to a useful domain and try to work as well as possible within that domain. FLAC's domain is audio data. Though it can losslessly \*code\* any input, only certain kinds of input will get smaller. FLAC exploits the fact that audio data typically has a high degree of sample-to-sample correlation.

Within the audio domain, there are many possible subdomains. For example: low bitrate speech, high-bitrate multi-channel music, etc. FLAC itself does not target a specific subdomain but many of the default parameters of the reference encoder are tuned to CD-quality music data (i.e. 44.1 kHz, 2 channel, 16 bits per sample). The



effect of the encoding parameters on different kinds of audio data will be examined later.

#### 4. Architecture

Similar to many audio coders, a FLAC encoder has the following stages:

- o "Blocking" (see Section 6). The input is broken up into many contiguous blocks. With FLAC, the blocks may vary in size. The optimal size of the block is usually affected by many factors, including the sample rate, spectral characteristics over time, etc. Though FLAC allows the block size to vary within a stream, the reference encoder uses a fixed block size.
- o "Interchannel Decorrelation" (see Section 7). In the case of stereo streams, the encoder will create mid and side signals based on the average and difference (respectively) of the left and right channels. The encoder will then pass the best form of the signal to the next stage.
- o "Prediction" (see Section 8). The block is passed through a prediction stage where the encoder tries to find a mathematical description (usually an approximate one) of the signal. This description is typically much smaller than the raw signal itself. Since the methods of prediction are known to both the encoder and decoder, only the parameters of the predictor need be included in the compressed stream. FLAC currently uses four different classes of predictors, but the format has reserved space for additional methods. FLAC allows the class of predictor to change from block to block, or even within the channels of a block.
- o "Residual Coding" (See Section 9). If the predictor does not describe the signal exactly, the difference between the original signal and the predicted signal (called the error or residual signal) must be coded losslessly. If the predictor is effective, the residual signal will require fewer bits per sample than the original signal. FLAC currently uses only one method for encoding the residual, but the format has reserved space for additional methods. FLAC allows the residual coding method to change from block to block, or even within the channels of a block.

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



## 5. Definitions

Many terms like "block" and "frame" are used to mean different things in different encoding schemes. For example, a frame in MP3 corresponds to many samples across several channels, whereas an S/PDIF frame represents just one sample for each channel. The definitions we use for FLAC follow. Note that when we talk about blocks and subblocks we are referring to the raw unencoded audio data that is the input to the encoder, and when we talk about frames and subframes, we are referring to the FLAC-encoded data.

- o **\*Block\***: One or more audio samples that span several channels.
- o **\*Subblock\***: One or more audio samples within a channel. So a block contains one subblock for each channel, and all subblocks contain the same number of samples.
- o **\*Blocksize\***: The number of samples in any of a block's subblocks. For example, a one second block sampled at 44.1 kHz has a blocksize of 44100, regardless of the number of channels.
- o **\*Frame\***: A frame header plus one or more subframes.
- o **\*Subframe\***: A subframe header plus one or more encoded samples from a given channel. All subframes within a frame will contain the same number of samples.
- o **\*Exponential-Golomb coding\***: One of Robert Rice's universal coding schemes, FLAC's residual coder, compresses data by writing the number of bits to be read minus 1, before writing the actual value.
- o **\*LPC\***: Linear predictive coding [7].

## 6. Blocking

The size used for blocking the audio data has a direct effect on the compression ratio. If the block size is too small, the resulting large number of frames mean that excess bits will be wasted 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, and a maximum block size of 65535 samples. This range covers the optimal size for all of the audio data FLAC supports.



Currently the reference encoder uses a fixed block size, optimized on the sample rate of the input. Future versions may vary the block size depending on the characteristics of the signal.

Blocked data is passed to the predictor stage one subblock (channel) at a time. Each subblock is independently coded into a subframe, and the subframes are concatenated into a frame. Because each channel is coded separately, it means that one channel of a stereo frame may be encoded as a constant subframe, and the other an LPC subframe.

## 7. Interchannel Decorrelation

In stereo streams, many times there is an exploitable amount of correlation between the left and right channels. FLAC allows the frames of stereo streams to have different channel assignments, and an encoder may choose to use the best representation on a frame-by-frame basis.

- o *\*Independent\**. The left and right channels are coded independently.
- o *\*Mid-side\**. The left and right channels are transformed into mid and side channels. The mid channel is the midpoint (average) of the left and right signals, and the side is the difference signal (left minus right).
- o *\*Left-side\**. The left channel and side channel are coded.
- o *\*Right-side\**. The right channel and side channel are coded

Surprisingly, the left-side and right-side forms can be the most efficient in many frames, even though the raw number of bits per sample needed for the original signal is slightly more than that needed for independent or mid-side coding.

## 8. Prediction

FLAC uses four methods for modeling the input signal:

- o *\*Verbatim\**. This is essentially a zero-order predictor of the signal. The predicted signal is zero, meaning the residual is the signal itself, and the compression is zero. This is the baseline against which the other predictors are measured. If you feed random data to the encoder, the verbatim predictor will probably be used for every subblock. Since the raw signal is not actually passed through the residual coding stage (it is added to the stream 'verbatim'), the encoding results will not be the same as a zero-order linear predictor.



- o *\*Constant\**. This predictor is used whenever the subblock is pure DC ("digital silence"), i.e. a constant value throughout. The signal is run-length encoded and added to the stream.
- o *\*Fixed linear predictor\**. FLAC uses a class of computationally-efficient fixed linear predictors (for a good description, see audiopak [8] and shorten [9]). FLAC adds a fourth-order predictor to the zero-to-third-order predictors used by Shorten. Since the predictors are fixed, the predictor order is the only parameter that needs to be stored in the compressed stream. The error signal is then passed to the residual coder.
- o *\*FIR Linear prediction\**. For more accurate modeling (at a cost of slower encoding), FLAC supports up to 32nd order FIR linear prediction (again, for information on linear prediction, see audiopak [10] and shorten [11]). The reference encoder uses the Levinson-Durbin method for calculating the LPC coefficients from the autocorrelation coefficients, and the coefficients are quantized before computing the residual. Whereas encoders such as Shorten used a fixed quantization for the entire input, FLAC allows the quantized coefficient precision to vary from subframe to subframe. The FLAC reference encoder estimates the optimal precision to use based on the block size and dynamic range of the original signal.

## 9. Residual Coding

FLAC uses Exponential-Golomb (a variant of Rice) coding as it's residual encoder. You can learn more about exp-golomb coding on Wikipedia [12]

FLAC currently defines two similar methods for the coding of the error signal from the prediction stage. The error signal is coded using Exponential-Golomb codes in one of two ways:

1. the encoder estimates a single exp-golomb parameter based on the variance of the residual and exp-golomb codes the entire residual using this parameter;
2. the residual is partitioned into several equal-length regions of contiguous samples, and each region is coded with its own exp-golomb parameter based on the region's mean. (Note that the first method is a special case of the second method with one partition, except the exp-golomb parameter is based on the residual variance instead of the mean.)

The FLAC format has reserved space for other coding methods. Some possibilities for volunteers would be to explore better context-



modeling of the exp-golomb parameter, or Huffman coding. See LOCO-I [13] and pucrunch [14] for descriptions of several universal codes.

## 10. Format

This section specifies the FLAC bitstream format. 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 or skip data encoded with newer methods. Apart from reserved patterns, in places the format specifies invalid patterns, meaning that the patterns may never appear in any valid bitstream, in any prior, present, or future versions of the format. These invalid patterns are usually used to make the synchronization mechanism more robust.

All numbers used in a FLAC bitstream are integers; there are no floating-point representations. All numbers are big-endian coded. All numbers are unsigned unless otherwise specified.

Before the formal description of the stream, an overview might be helpful.

- o A FLAC bitstream consists of the "fLaC" marker at the beginning of the stream, followed by a mandatory metadata block (called the STREAMINFO block), any number of other metadata blocks, then the audio frames.
- o FLAC supports up to 128 kinds of metadata blocks; currently the following are defined:
  - \* "STREAMINFO": This 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, and ones that the decoder doesn't understand, it will skip.
  - \* "APPLICATION": This block is for use by third-party applications. The only mandatory field is a 32-bit identifier. This ID is granted upon request to an application by the FLAC maintainers. The remainder of the block is defined by the registered application. Visit the registration page [15] if you would like to register an ID for your application with FLAC.
  - \* "PADDING": This block allows for an arbitrary amount of padding. The contents of a PADDING block have no meaning. 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 right place in the existing file (which would normally require rewriting the entire file).

- \* "SEEKTABLE": This is an optional block for storing 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. Each seek point takes 18 bytes, so 1% resolution within a stream adds less than 2K. There can be only one SEEKTABLE in a stream, but the table can have any number of seek points. There is also a special 'placeholder' seekpoint which will be ignored by decoders but which can be used to reserve space for future seek point insertion.
- \* "VORBIS\_COMMENT": This block is for storing a list of human-readable name/value pairs. Values are encoded using UTF-8. It is an implementation of the Vorbis comment specification [16] (without the framing bit). This is the only officially supported tagging mechanism in FLAC. There may be only one VORBIS\_COMMENT block in a stream. In some external documentation, Vorbis comments are called FLAC tags to lessen confusion.
- \* "CUESHEET": This block is for storing various information that can be used in a cue sheet. It supports track and index points, compatible with Red Book CD digital audio discs, as well as other CD-DA metadata such as media catalog number and track ISRCs. The CUESHEET block is especially useful for backing up CD-DA discs, but it can be used as a general purpose cueing mechanism for playback.
- \* "PICTURE": This block is for storing pictures associated with the file, most commonly cover art from CDs. There may be more than one PICTURE block in a file. The picture format is similar to the APIC frame in ID3v2 [17]. The PICTURE block has a type, MIME type, and UTF-8 description like ID3v2, and supports external linking via URL (though this is discouraged). The differences are that there is no uniqueness constraint on the description field, and the MIME type is mandatory. The FLAC PICTURE block also includes the resolution, color depth, and palette size so that the client can search for a suitable picture without having to scan them all.



- o 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, number of channels, et cetera, and an 8-bit CRC. The frame header also contains either the sample number of the first sample in the frame (for variable-blocksize streams), or the frame number (for fixed-blocksize streams). This allows for fast, sample-accurate seeking to be performed. Following the frame header are encoded subframes, one for each channel, and finally, the frame is zero-padded to a byte boundary. Each subframe has its own header that specifies how the subframe is encoded.
- o Since a decoder may start decoding in the middle of a stream, there must be a method to determine the start of a frame. A 14-bit sync code begins each frame. The sync code will not appear anywhere else in the frame header. However, since it may appear in the subframes, the decoder has two other ways of ensuring a correct sync. The first is to check that the rest of the frame header contains no invalid data. Even this is not foolproof since valid header patterns can still occur within the subframes. The decoder's final check is to generate an 8-bit CRC of the frame header and compare this to the CRC stored at the end of the frame header.
- o Again, since a decoder may start decoding at an arbitrary frame in the stream, each frame header must contain some basic information about the stream because the decoder may not have access to the STREAMINFO metadata block at the start of the stream. This information includes sample rate, bits per sample, number of channels, etc. Since the frame header is pure 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 parameters. For instance, the sample rate part of the frame header is specified using 4 bits. Eight of the bit patterns correspond to the commonly used sample rates of 8/16/22.05/24/32/44.1/48/96 kHz. However, odd sample rates can be specified by using one of the 'hint' bit patterns, directing the decoder to find the exact sample rate at the end of the frame header. The same method is used for specifying the block size and bits per sample. In this way, the frame header size stays small for all of the most common forms of audio data.
- o 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. The header is followed by the encoded audio data for that channel.

- o "FLAC" specifies a subset of itself as the Subset format. The purpose of this is to ensure that any streams encoded according to the 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 such that decoder buffer sizes and other resource requirements can be easily determined. \*flac\* generates Subset streams by default unless the "--lax" command-line option is used. The Subset makes the following limitations on what may be used in the stream:

- \* The blocksize bits in the "FRAME\_HEADER" (see Section 10.19) must be 0001-1110. The blocksize must be  $\leq 16384$ ; if the sample rate is  $\leq 48000$  Hz, the blocksize must be  $\leq 4608$ .
- \* The sample rate bits in the "FRAME\_HEADER" must be 0001-1110.
- \* The bits-per-sample bits in the "FRAME\_HEADER" must be 001-111.
- \* If the sample rate is  $\leq 48000$  Hz, the filter order in "LPC subframes" (see Section 10.25) must be less than or equal to 12, i.e. the subframe type bits in the "SUBFRAME\_HEADER" (see Section 10.22) may not be 101100-111111.
- \* The Rice partition order in an "exp-golomb coded residual section" (see Section 10.27.2) must be less than or equal to 8.

#### 10.1. Conventions

The following tables constitute a formal description of the FLAC format. Values expressed as "u(n)" represent unsigned big-endian integer using "n" bits. "n" may be expressed as an equation using "\*" (multiplication), "/" (division), "+" (addition), or "-" (subtraction). An inclusive range of the number of bits expressed may be represented with an ellipsis, such as "u(m...n)". The name of a value followed by an asterisk "\*" indicates zero or more occurrences of the value. The name of a value followed by a plus sign "+" indicates one or more occurrences of the value.

#### 10.2. STREAM



Data	Description
"u(32)"	"fLaC", the FLAC stream marker in ASCII, meaning byte 0 of the stream is 0x66, followed by 0x4C 0x61 0x43
"METADATA_BLOCK_STREAMINFO"	This is the mandatory STREAMINFO metadata block that has the basic properties of the stream.
"METADATA_BLOCK" *	Zero or more metadata blocks
"FRAME" +	One or more audio frames

## 10.3. METADATA\_BLOCK

Data	Description
"METADATA_BLOCK_HEADER"	A block header that specifies the type and size of the metadata block data.
"METADATA_BLOCK_DATA"	

## 10.4. METADATA\_BLOCK\_HEADER

Data	Description
"u(1)"	Last-metadata-block flag: '1' if this block is the last metadata block before the audio blocks, '0' otherwise.
"u(7)"	"BLOCK_TYPE"
"u(24)"	Length (in bytes) of metadata to follow (does not include the size of the "METADATA_BLOCK_HEADER")

## 10.5. BLOCK\_TYPE



Value	Description
0	STREAMINFO
1	PADDING
2	APPLICATION
3	SEEKTABLE
4	VORBIS_COMMENT
5	CUESHEET
6	PICTURE
7 - 126	reserved
127	invalid, to avoid confusion with a frame sync code

## 10.6. METADATA\_BLOCK\_DATA

Data	Description
"METADATA_BLOCK_STREAMINFO"    "METADATA_BLOCK_PADDING"    "METADATA_BLOCK_APPLICATION"    "METADATA_BLOCK_SEEKTABLE"    "METADATA_BLOCK_VORBIS_COMMENT"    "METADATA_BLOCK_CUESHEET"    "METADATA_BLOCK_PICTURE"	The block data must match the block type in the block header.

## 10.7. METADATA\_BLOCK\_STREAMINFO



Data	Description
"u(16)"	The minimum block size (in samples) used in the stream.
"u(16)"	The maximum block size (in samples) used in the stream. (Minimum blocksize == maximum blocksize) implies a fixed-blocksize stream.
"u(24)"	The minimum frame size (in bytes) used in the stream. May be 0 to imply the value is not known.
"u(24)"	The maximum frame size (in bytes) used in the stream. May be 0 to imply the value is not known.
"u(20)"	Sample rate in Hz. Though 20 bits are available, the maximum sample rate is limited by the structure of frame headers to 655350 Hz. Also, a value of 0 is invalid.
"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. Currently the reference encoder and decoders only support up to 24 bits per sample.
"u(36)"	Total samples in stream. 'Samples' means inter-channel sample, i.e. one second of 44.1 kHz audio will have 44100 samples regardless of the number of channels. A value of zero here means the number of total samples is unknown.
"u(128)"	MD5 signature of the unencoded audio data. This allows the decoder to determine if an error exists in the audio data even when the error does not result in an invalid bitstream.

## NOTE

- o FLAC specifies a minimum block size of 16 and a maximum block size of 65535, meaning the bit patterns corresponding to the numbers 0-15 in the minimum blocksize and maximum blocksize fields are invalid.

## 10.8. METADATA\_BLOCK\_PADDING

Data	Description
"u(n)"	n '0' bits (n must be a multiple of 8)



## 10.9. METADATA\_BLOCK\_APPLICATION

Data	Description
"u(32)"	Registered application ID. (Visit the registration page [18] to register an ID with FLAC.)
"u(n)"	Application data (n must be a multiple of 8)

## 10.10. METADATA\_BLOCK\_SEEKTABLE

Data	Description
"SEEKPOINT"+	One or more seek points.

NOTE - The number of seek points is implied by the metadata header 'length' field, i.e. equal to length / 18.

## 10.11. SEEKPOINT

Data	Description
"u(64)"	Sample number of 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.

## NOTES

- o For placeholder points, the second and third field values are undefined.
- o Seek points within a table must be sorted in ascending order by sample number.
- o Seek points within a table must be unique by sample number, with the exception of placeholder points.
- o 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.



## 10.12. METADATA\_BLOCK\_VORBIS\_COMMENT

Data	Description
"u(n)"	Also known as FLAC tags, the contents of a vorbis comment packet as specified here [19] (without the framing bit). Note that the vorbis comment spec allows for on the order of $2^{64}$ bytes of data where as the FLAC metadata block is limited to $2^{24}$ bytes. Given the stated purpose of vorbis comments, i.e. human-readable textual information, this limit is unlikely to be restrictive. Also note that the 32-bit field lengths are little-endian coded according to the vorbis spec, as opposed to the usual big-endian coding of fixed-length integers in the rest of FLAC.

## 10.13. METADATA\_BLOCK\_CUESHEET



Data	Description
"u(128*8)"	Media catalog number, in ASCII printable characters 0x20-0x7e. In general, the media catalog number may be 0 to 128 bytes long; any unused characters should be right-padded with NUL characters. For CD-DA, this is a thirteen digit number, followed by 115 NUL bytes.
"u(64)"	The number of lead-in samples. This field 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 the Red Book, 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.
"u(1)"	"1" if the CUESHEET corresponds to a Compact Disc, else "0".
"u(7+258*8)"	Reserved. All bits must be set to zero.
"u(8)"	The number of tracks. Must be at least 1 (because of the requisite lead-out track). For CD-DA, this number must be no more than 100 (99 regular tracks and one lead-out track).
"CUESHEET_TRACK"+	One or more tracks. A CUESHEET block is required to have a lead-out track; it is always the last track in the CUESHEET. For CD-DA, the lead-out track number must be 170 as specified by the Red Book, otherwise is must be 255.



Internet-Draft

FLAC

June 2017

10.14. CUESHEET\_TRACK



Data	Description
"u(64)"	Track offset in samples, relative to the beginning of the FLAC audio stream. It is the offset to the first index point of the track. (Note how this differs from 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 offset must be evenly divisible by 588 samples (588 samples = 44100 samples/sec * 1/75th of a sec).
"u(8)"	Track number. A track number of 0 is not allowed to avoid conflicting with the CD-DA spec, which 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 for 255 for the lead-out. It is not required but encouraged to start with track 1 and increase sequentially. Track numbers must be unique within a CUESHEET.
"u(12\*8)"	Track ISRC. This is a 12-digit alphanumeric code; see here [20] and here [21]. A value of 12 ASCII NUL characters may be used to denote absence of an 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; see here [22].
"u(6+13*8)"	Reserved. All bits must be set to zero.
"u(8)"	The number of track index points. There must be at least one index in every track in a CUESHEET except for the lead-out track, which must have zero. For CD-DA, this number may be no more than 100.
"CUESHEET_TRACK_INDEX"+	For all tracks except the lead-out track, one or more track index points.



## 10.15. CUESHEET\_TRACK\_INDEX

Data	Description
"u(64)"	Offset in samples, relative to the track offset, of the index point. For CD-DA, the offset must be evenly divisible by 588 samples (588 samples = 44100 samples/sec * 1/75 sec). Note that the offset is from the beginning of the track, not the beginning of the audio data.
"u(8)"	The index point number. For CD-DA, an index number of 0 corresponds to the track pre-gap. The first index in a track must have a number of 0 or 1, and subsequently, index numbers must increase by 1. Index numbers must be unique within a track.
"u(3*8)"	Reserved. All bits must be set to zero.

## 10.16. METADATA\_BLOCK\_PICTURE

Data	Description
"u(32)"	The PICTURE_TYPE according to the ID3v2 APIC frame:
"u(32)"	The length of the MIME type string in bytes.
"u(n*8)"	The MIME type string, in printable ASCII characters 0x20-0x7e. The MIME type may also be "-->" to signify that the data part is a URL of the picture instead of the picture data itself.
"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.

## 10.17. PICTURE\_TYPE



Value	Description
0	Other
1	32x32 pixels 'file icon' (PNG only)
2	Other file icon
3	Cover (front)
4	Cover (back)
5	Leaflet page
6	Media (e.g. label side of CD)
7	Lead artist/lead performer/soloist
8	Artist/performer
9	Conductor
10	Band/Orchestra
11	Composer
12	Lyricist/text writer
13	Recording Location
14	During recording
15	During performance
16	Movie/video screen capture
17	A bright colored fish
18	Illustration
19	Band/artist logotype
20	Publisher/Studio logotype

Other values are reserved and should not be used. There may only be one each of picture type 1 and 2 in a file.

#### 10.18. FRAME

Data	Description
"FRAME_HEADER"	
"SUBFRAME"+	One SUBFRAME per channel.
"u(?)"	Zero-padding to byte alignment.
"FRAME_FOOTER"	

#### 10.19. FRAME\_HEADER



Data	Description
"u(14)"	Sync code '0b11111111111110'
"u(1)"	"FRAME HEADER RESERVED"
"u(1)"	"BLOCKING STRATEGY"
"u(4)"	"INTERCHANNEL SAMPLE BLOCK SIZE"
"u(4)"	"SAMPLE RATE"
"u(4)"	"CHANNEL ASSIGNMENT"
"u(3)"	"SAMPLE SIZE"
"u(1)"	"FRAME HEADER RESERVED2"
"u(?)"	"CODED NUMBER"
"u(?)"	"BLOCK SIZE INT"
"u(?)"	"SAMPLE RATE INT"
"u(8)"	"FRAME CRC"

#### 10.19.1. FRAME HEADER RESERVED

Value	Description
0	mandatory value
1	reserved for future use

FRAME HEADER RESERVED must remain reserved for "0" in order for a FLAC frame's initial 15 bits to be distinguishable from the start of an MPEG audio frame (see also [23]).

#### 10.19.2. BLOCKING STRATEGY

Value	Description
0	fixed-blocksize stream; frame header encodes the frame number
1	variable-blocksize stream; frame header encodes the sample number

The "BLOCKING STRATEGY" bit must be the same throughout the entire stream.

The "BLOCKING STRATEGY" bit determines how to calculate the sample number of the first sample in the frame. If the bit is "0" (fixed-blocksize), the frame header encodes the frame number as above, and the frame's starting sample number will be the frame number times the



blocksize. If it is "1" (variable-blocksize), the frame header encodes the frame's starting sample number itself. (In the case of a fixed-blocksize stream, only the last block may be shorter than the stream blocksize; its starting sample number will be calculated as the frame number times the previous frame's blocksize, or zero if it is the first frame).

#### 10.19.3. INTERCHANNEL SAMPLE BLOCK SIZE

Value	Description
0b0000	reserved
0b0001	192 samples
0b0010 - 0b0101	576 * (2 <sup>(n-2)</sup> ) samples, i.e. 576/1152/2304/4608
0b0110	get 8 bit (blocksize-1) from end of header
0b0111	get 16 bit (blocksize-1) from end of header
0b1000 - 0b1111	256 * (2 <sup>(n-8)</sup> ) samples, i.e. 256/512/1024/2048/4096/8192/16384/32768

#### 10.19.4. SAMPLE RATE

Value	Description
0b0000	get from 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	get 8 bit sample rate (in kHz) from end of header
0b1101	get 16 bit sample rate (in Hz) from end of header
0b1110	get 16 bit sample rate (in tens of Hz) from end of header
0b1111	invalid, to prevent sync-fooling string of 1s



## 10.19.5. CHANNEL ASSIGNMENT

For values 0000-0111, the value represents the (number of independent channels)-1. Where defined, the channel order follows SMPTE/ITU-R recommendations.

Value	Description
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	left/side stereo: channel 0 is the left channel, channel 1 is the side(difference) channel
0b1001	right/side stereo: channel 0 is the side(difference) channel, channel 1 is the right channel
0b1010	mid/side stereo: channel 0 is the mid(average) channel, channel 1 is the side(difference) channel
0b1011 - 0b1111	reserved

## 10.19.6. SAMPLE SIZE

Value	Description
0b000	get from 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	reserved



## 10.19.7. FRAME HEADER RESERVED2

Value	Description
0	mandatory value
1	reserved for future use

## 10.19.8. CODED NUMBER

The "UTF-8" coding used for the sample/frame number is the same variable length code used to store compressed UCS-2, extended to handle larger input.

```
if(variable blocksize)
    'u(8...56)`: "UTF-8" coded sample number (decoded number is 36 bits)
else
    'u(8...48)`: "UTF-8" coded frame number (decoded number is 31 bits)
```

## 10.19.9. BLOCK SIZE INT

```
if('INTERCHANNEL SAMPLE BLOCK SIZE' == 0b0110)
    8 bit (blocksize-1)
else if('INTERCHANNEL SAMPLE BLOCK SIZE' == 0b0111)
    16 bit (blocksize-1)
```

## 10.19.10. SAMPLE RATE INT

```
if('SAMPLE RATE' == 0b1100)
    8 bit sample rate (in kHz)
else if('SAMPLE RATE' == 0b1101)
    16 bit sample rate (in Hz)
else if('SAMPLE RATE' == 0b1110)
    16 bit sample rate in tens of Hz)
```

## 10.19.11. FRAME CRC

CRC-8 (polynomial =  $x^8 + x^2 + x^1 + x^0$ , initialized with 0) of everything before the CRC, including the sync code

## 10.20. FRAME\_FOOTER



Data	Description
"u(16)"	CRC-16 (polynomial = $x^{16} + x^{15} + x^2 + x^0$ , initialized with 0) of everything before the CRC, back to and including the frame header sync code

## 10.21. SUBFRAME

Data	Description
"SUBFRAME_HEADER"	
"SUBFRAME_CONSTANT"    "SUBFRAME_FIXED"	The SUBFRAME_HEADER specifies which one.
"SUBFRAME_LPC"    "SUBFRAME_VERBATIM"	

## 10.22. SUBFRAME\_HEADER

Data	Description
"u(1)"	Zero bit padding, to prevent sync-fooling string of 1s
"u(6)"	"SUBFRAME TYPE" (see Section 10.22.1)
"u(1+k)"	"WASTED BITS PER SAMPLE FLAG" (see Section 10.22.2)

## 10.22.1. SUBFRAME TYPE

Value	Description
0b000000	"SUBFRAME_CONSTANT"
0b000001	"SUBFRAME_VERBATIM"
0b00001x	reserved
0b0001xx	reserved
0b001xxx	if( $xxx \leq 4$ ) "SUBFRAME_FIXED", $xxx=order$ ; else reserved
0b01xxxx	reserved
0b1xxxxx	"SUBFRAME_LPC", $xxxxx=order-1$

## 10.22.2. WASTED BITS PER SAMPLE FLAG



Value	Description
0	no wasted bits-per-sample in source subblock, k=0
1	k wasted bits-per-sample in source subblock, k-1 follows, unary coded; e.g. k=3 => 001 follows, k=7 => 0000001 follows.

## 10.23. SUBFRAME\_CONSTANT

Data	Description
"u(n)"	Unencoded constant value of the subblock, n = frame's bits-per-sample.

## 10.24. SUBFRAME\_FIXED

Data	Description
"u(n)"	Unencoded warm-up samples (n = frame's bits-per-sample * predictor order).
"RESIDUAL"	Encoded residual

## 10.25. SUBFRAME\_LPC

Data	Description
"u(n)"	Unencoded warm-up samples (n = frame's bits-per-sample * lpc order).
"u(4)"	(Quantized linear predictor coefficients' precision in bits)-1 (0b1111 = invalid).
"u(5)"	Quantized linear predictor coefficient shift needed in bits (NOTE: this number is signed two's-complement).
"u(n)"	Unencoded predictor coefficients (n = qlp coeff precision * lpc order) (NOTE: the coefficients are signed two's-complement).
"RESIDUAL"	Encoded residual



## 10.26. SUBFRAME\_VERBATIM

Data	Description
"u(n\*i)"	Unencoded subblock; n = frame's bits-per-sample, i = frame's blocksize.

## 10.27. RESIDUAL

Data	Description
"u(2)"	"RESIDUAL_CODING_METHOD"
"RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB"    "RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB2"	"OD"

## 10.27.1. RESIDUAL\_CODING\_METHOD

Value	Description
0b00	partitioned Exp-Golomb coding with 4-bit Exp-Golomb parameter; RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB follows
0b01	partitioned Exp-Golomb coding with 5-bit Exp-Golomb parameter; RESIDUAL_CODING_METHOD_PARTITIONED_EXP_GOLOMB2 follows
0b10 - 0b11	reserved

## 10.27.2. RESIDUAL\_CODING\_METHOD\_PARTITIONED\_EXP\_GOLOMB

Data	Description
"u(4)"	Partition order.
"EXP_GOLOMB_PARTITION"+	There will be 2^order partitions.



## 10.27.2.1. EXP\_GOLOMB\_PARTITION

Data	Description
"u(4(+5))"	"EXP-GOLOMB PARTITION ENCODING PARAMETER" (see Section 10.27.2.2)
"u(?)"	"ENCODED RESIDUAL" (see Section 10.27.4)

## 10.27.2.2. EXP-GOLOMB PARTITION ENCODING PARAMETER

Value	Description
0b0000 - 0b1110	Exp-golomb parameter.
0b1111	Escape code, meaning the partition is in unencoded binary form using n bits per sample; n follows as a 5-bit number.

## 10.27.3. RESIDUAL\_CODING\_METHOD\_PARTITIONED\_EXP\_GOLOMB2

Data	Description
"u(4)"	Partition order.
"EXP-GOLOMB2_PARTITION"+	There will be 2^order partitions.

## 10.27.3.1. EXP\_GOLOMB2\_PARTITION

Data	Description
"u(5(+5))"	"EXP-GOLOMB2 PARTITION ENCODING PARAMETER" (see Section 10.27.3.2)
"u(?)"	"ENCODED RESIDUAL" (see Section 10.27.4)

## 10.27.3.2. EXP-GOLOMB2 PARTITION ENCODING PARAMETER



Value	Description
0b00000	Exp-golomb parameter.
0b11110	Escape code, meaning the partition is in unencoded binary form using n bits per sample; n follows as a 5-bit number.
0b11111	

#### 10.27.4. ENCODED RESIDUAL

The number of samples (n) in the partition is determined as follows:

- o if the partition order is zero,  $n = \text{frame's blocksize} - \text{predictor order}$
- o else if this is not the first partition of the subframe,  $n = (\text{frame's blocksize} / (2^{\text{partition order}}))$
- o else  $n = (\text{frame's blocksize} / (2^{\text{partition order}})) - \text{predictor order}$

Copyright (c) 2000-2009 Josh Coalson, 2011-2014 Xiph.Org Foundation

## 11. References

### 11.1. URIs

- [1] [ogg\\_mapping.html](#)
- [2] [documentation\\_format\\_overview.html](#)
- [3] <http://svr-www.eng.cam.ac.uk/~ajr/>
- [4] [http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson\\_tr156.html](http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html)
- [5] <https://web.archive.org/web/20040215005354/http://csi.usc.edu/faculty/golomb.html>
- [6] [http://en.wikipedia.org/wiki/Claude\\_Shannon](http://en.wikipedia.org/wiki/Claude_Shannon)
- [7] [https://en.wikipedia.org/wiki/Linear\\_predictive\\_coding](https://en.wikipedia.org/wiki/Linear_predictive_coding)
- [8] <http://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf>



- [9] [http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson\\_tr156.html](http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html)
- [10] <http://www.hpl.hp.com/techreports/1999/HPL-1999-144.pdf>
- [11] [http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson\\_tr156.html](http://svr-www.eng.cam.ac.uk/reports/abstracts/robinson_tr156.html)
- [12] [https://en.wikipedia.org/wiki/Exponential-Golomb\\_coding](https://en.wikipedia.org/wiki/Exponential-Golomb_coding)
- [13] <http://www.hpl.hp.com/techreports/98/HPL-98-193.html>
- [14] <http://web.archive.org/web/20140827133312/http://www.cs.tut.fi/~albert/Dev/pucrunch/packing.html>
- [15] <https://xiph.org/flac/id.html>
- [16] <http://xiph.org/vorbis/doc/v-comment.html>
- [17] <http://www.id3.org/id3v2.4.0-frames>
- [18] [id.html](#)
- [19] <http://www.xiph.org/vorbis/doc/v-comment.html>
- [20] <http://isrc.ifpi.org/>
- [21] [http://www.disctronics.co.uk/technology/cdaudio/cdaud\\_isrc.htm](http://www.disctronics.co.uk/technology/cdaudio/cdaud_isrc.htm)
- [22] <http://www.chipchapin.com/CDMedia/cdda9.php3>
- [23] <http://lists.xiph.org/pipermail/flac-dev/2008-December/002607.html>

#### Authors' Addresses

Josh Coalson

Xiph.Org Foundation