

cellar  
Internet-Draft  
Intended status: Standards Track  
Expires: November 28, 2019

S. Lhomme  
  
D. Rice  
  
M. Bunkus  
May 27, 2019

**Extensible Binary Meta Language**  
**draft-ietf-cellar-ebml-10**

Abstract

This document defines the Extensible Binary Meta Language (EBML) format as a generalized file format for any type of data in a hierarchical form. 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 November 28, 2019.

Copyright Notice

Copyright (c) 2019 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

|                          |  |                    |
|--------------------------|--|--------------------|
| <a href="#">1.</a>       | <a href="#">Introduction . . . . .</a>                     | <a href="#">3</a>  |
| <a href="#">2.</a>       | <a href="#">Notation and Conventions . . . . .</a>         | <a href="#">4</a>  |
| <a href="#">3.</a>       | <a href="#">Structure . . . . .</a>                        | <a href="#">6</a>  |
| <a href="#">4.</a>       | <a href="#">Variable Size Integer . . . . .</a>            | <a href="#">7</a>  |
| <a href="#">4.1.</a>     | <a href="#">VINT_WIDTH . . . . .</a>                       | <a href="#">7</a>  |
| <a href="#">4.2.</a>     | <a href="#">VINT_MARKER . . . . .</a>                      | <a href="#">7</a>  |
| <a href="#">4.3.</a>     | <a href="#">VINT_DATA . . . . .</a>                        | <a href="#">7</a>  |
| <a href="#">4.4.</a>     | <a href="#">VINT Examples . . . . .</a>                    | <a href="#">8</a>  |
| <a href="#">5.</a>       | <a href="#">Element ID . . . . .</a>                       | <a href="#">9</a>  |
| <a href="#">6.</a>       | <a href="#">Element Data Size . . . . .</a>                | <a href="#">10</a> |
| <a href="#">7.</a>       | <a href="#">EBML Element Types . . . . .</a>               | <a href="#">12</a> |
| <a href="#">7.1.</a>     | <a href="#">Signed Integer Element . . . . .</a>           | <a href="#">12</a> |
| <a href="#">7.2.</a>     | <a href="#">Unsigned Integer Element . . . . .</a>         | <a href="#">13</a> |
| <a href="#">7.3.</a>     | <a href="#">Float Element . . . . .</a>                    | <a href="#">13</a> |
| <a href="#">7.4.</a>     | <a href="#">String Element . . . . .</a>                   | <a href="#">13</a> |
| <a href="#">7.5.</a>     | <a href="#">UTF-8 Element . . . . .</a>                    | <a href="#">13</a> |
| <a href="#">7.6.</a>     | <a href="#">Date Element . . . . .</a>                     | <a href="#">13</a> |
| <a href="#">7.7.</a>     | <a href="#">Master Element . . . . .</a>                   | <a href="#">14</a> |
| <a href="#">7.8.</a>     | <a href="#">Binary Element . . . . .</a>                   | <a href="#">14</a> |
| <a href="#">8.</a>       | <a href="#">EBML Document . . . . .</a>                    | <a href="#">14</a> |
| <a href="#">8.1.</a>     | <a href="#">EBML Header . . . . .</a>                      | <a href="#">15</a> |
| <a href="#">8.2.</a>     | <a href="#">EBML Body . . . . .</a>                        | <a href="#">15</a> |
| <a href="#">9.</a>       | <a href="#">EBML Stream . . . . .</a>                      | <a href="#">15</a> |
| <a href="#">10.</a>      | <a href="#">EBML Versioning . . . . .</a>                  | <a href="#">15</a> |
| <a href="#">10.1.</a>    | <a href="#">EBML Header Version . . . . .</a>              | <a href="#">16</a> |
| <a href="#">10.2.</a>    | <a href="#">EBML Document Version . . . . .</a>            | <a href="#">16</a> |
| <a href="#">11.</a>      | <a href="#">Elements semantic . . . . .</a>                | <a href="#">16</a> |
| <a href="#">11.1.</a>    | <a href="#">EBML Schema . . . . .</a>                      | <a href="#">16</a> |
| <a href="#">11.1.1.</a>  | <a href="#">EBML Schema Example . . . . .</a>              | <a href="#">17</a> |
| <a href="#">11.1.2.</a>  | <a href="#">&lt;EBMLSchema&gt; Element . . . . .</a>       | <a href="#">18</a> |
| <a href="#">11.1.3.</a>  | <a href="#">&lt;EBMLSchema&gt; Attributes . . . . .</a>    | <a href="#">18</a> |
| <a href="#">11.1.4.</a>  | <a href="#">&lt;element&gt; Element . . . . .</a>          | <a href="#">18</a> |
| <a href="#">11.1.5.</a>  | <a href="#">&lt;element&gt; Attributes . . . . .</a>       | <a href="#">19</a> |
| <a href="#">11.1.6.</a>  | <a href="#">&lt;documentation&gt; Element . . . . .</a>    | <a href="#">24</a> |
| <a href="#">11.1.7.</a>  | <a href="#">&lt;documentation&gt; Attributes . . . . .</a> | <a href="#">24</a> |
| <a href="#">11.1.8.</a>  | <a href="#">&lt;restriction&gt; Element . . . . .</a>      | <a href="#">25</a> |
| <a href="#">11.1.9.</a>  | <a href="#">&lt;enum&gt; Element . . . . .</a>             | <a href="#">25</a> |
| <a href="#">11.1.10.</a> | <a href="#">&lt;enum&gt; Attributes . . . . .</a>          | <a href="#">25</a> |
| <a href="#">11.1.11.</a> | <a href="#">XML Schema for EBML Schema . . . . .</a>       | <a href="#">26</a> |
| <a href="#">11.1.12.</a> | <a href="#">Identically Recurring Elements . . . . .</a>   | <a href="#">27</a> |



|          |   |    |
|----------|---|----|
| 11.1.13. | Textual expression of floats . . . . .  | 28 |
| 11.1.14. | Note on the use of default attributes to define<br>Mandatory EBML Elements . . . . .  | 28 |
| 11.2.    | EBML Header Elements . . . . .  | 29 |
| 11.2.1.  | EBML Element . . . . .  | 29 |
| 11.2.2.  | EBMLVersion Element . . . . .   | 29 |
| 11.2.3.  | EBMLReadVersion Element . . . . .   | 30 |
| 11.2.4.  | EBMLMaxIDLength Element . . . . .   | 30 |
| 11.2.5.  | EBMLMaxSizeLength Element . . . . .   | 31 |
| 11.2.6.  | DocType Element . . . . .   | 31 |
| 11.2.7.  | DocTypeVersion Element . . . . .  | 32 |
| 11.2.8.  | DocTypeReadVersion Element . . . . .  | 32 |
| 11.2.9.  | DocTypeExtension Element . . . . .  | 33 |
| 11.2.10. | DocTypeExtensionName Element . . . . .  | 33 |
| 11.2.11. | DocTypeExtensionVersion Element . . . . .   | 34 |
| 11.3.    | Global Elements . . . . .   | 34 |
| 11.3.1.  | CRC-32 Element . . . . .  | 34 |
| 11.3.2.  | Void Element . . . . .  | 35 |
| 12.      | Considerations for Reading EBML Data . . . . .  | 35 |
| 13.      | Terminating Elements . . . . .  | 36 |
| 14.      | Guidelines for Updating Elements . . . . .  | 36 |
| 14.1.    | Reducing a Element Data in Size . . . . .   | 36 |
| 14.1.1.  | Adding a Void Element . . . . .   | 36 |
| 14.1.2.  | Extending the Element Data Size . . . . .   | 37 |
| 14.1.3.  | Terminating Element Data . . . . .  | 38 |
| 14.2.    | Considerations when Updating Elements with Cyclic<br>Redundancy Check (CRC) . . . . . | 38 |
| 15.      | Backward and Forward Compatibility . . . . .  | 38 |
| 15.1.    | Backward Compatibility . . . . .  | 38 |
| 15.2.    | Forward Compatibility . . . . .   | 39 |
| 16.      | Security Considerations . . . . .   | 39 |
| 17.      | IANA Considerations . . . . .   | 41 |
| 17.1.    | CELLAR EBML Element ID Registry . . . . .   | 41 |
| 17.2.    | CELLAR EBML DocType Registry . . . . .  | 43 |
| 18.      | References . . . . .  | 43 |
| 18.1.    | Normative References . . . . .  | 43 |
| 18.2.    | Informative References . . . . .  | 44 |
| 18.3.    | URIs . . . . .  | 45 |
|          | Authors' Addresses . . . . .  | 45 |

## 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 [1]. 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.

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".

"EBML Class": A representation of the octet length of an "Element ID".

"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 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 begins with a VINT\_WIDTH which consists of zero or many zero-value bits. The count of consecutive zero-values of the VINT\_WIDTH plus one equals the length in octets of the Variable Size Integer. For example, a Variable Size Integer that starts with a VINT\_WIDTH which contains zero consecutive zero-value bits is one octet in length and a Variable Size Integer that starts with one consecutive zero-value bit is two octets in length. The VINT\_WIDTH MUST only contain zero-value bits or be empty.

Within the EBML Header the VINT\_WIDTH of a VINT MUST NOT exceed three bits in length (meaning that the Variable Size Integer MUST NOT exceed four octets in length) except if said VINT is used to express the Element Data Size of an EBML Element with Element Name EBML and Element ID "0x1A45DFA3" (see [Section 11.2.1](#)) in which case the VINT\_WIDTH MUST NOT exceed seven bits in length. Within the EBML Body, when a VINT is used to express an Element ID, the maximum length allowed for the VINT\_WIDTH is one less than the value set in the EBMLMaxIDLength Element. Within the EBML Body, when a VINT is used to express an Element Data Size, the maximum length allowed for the VINT\_WIDTH is one less than the value set in the EBMLMaxSizeLength Element.

### **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

This table 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 xxxx                                    |

Data encoded as 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 this table a 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.

| Binary Value | Octet Length | As Represented in Variable Size Integer |
|--------------|--------------|---|
| 10           | 1            | 1000 0010                               |
| 10           | 2            | 0100 0000 0000 0010                     |
| 10           | 3            | 0010 0000 0000 0000 0000 0010           |
| 10           | 4            | 0001 0000 0000 0000 0000 0000 0000 0010 |



## 5. Element ID

The Element ID is encoded as a Variable Size Integer. By default, Element IDs are encoded in lengths from one octet to four octets, 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 VINT\_DATA component of the Element ID MUST NOT be either defined or written as either all zero values or all one values. Any Element ID with the VINT\_DATA component set as all zero values or all one values MUST be ignored. 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.

The following table 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           | 00000001         | Valid   |
| 0          | 1           | 0000000000000001 | 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           | 00000001111111   | Valid   |

The octet length of an Element ID determines its EBML Class.





| EBML Class | Length | Possible IDs            | Number of IDs |
|------------|--------|-------------------------|---------------|
| Class A    | 1      | 0x81 - 0xFE             | 126           |
| Class B    | 2      | 0x407F - 0x7FFE         | 16,256        |
| Class C    | 3      | 0x203FFF - 0x3FFFFE     | 2,080,768     |
| Class D    | 4      | 0x101FFFFF - 0x1FFFFFFE | 268,338,304   |

## 6. Element Data Size

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.

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. A Master Element MAY be an Unknown-Sized Element; however an EBML Element that is not a Master Element MUST NOT be an Unknown-Sized Element. Master Elements MUST NOT use an unknown size unless the unknownsizeallowed attribute of their 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 Element MUST NOT be used or defined unnecessarily; however if the Element Data Size is not known before the Element Data is written, such as in some cases of data streaming, then Unknown-Sized Elements MAY be used. The end of an Unknown-Sized Element is determined by whichever comes first: the end of the file or the beginning of the next EBML Element, defined by this document or the corresponding EBML Schema, that is not independently valid as Descendant Element of the Unknown-Sized Element.

For Element Data Sizes encoded at octet lengths from one to eight, this table 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 $2^{56}-2$ |
| +-----+      |   | +-----+              |                 |

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 a reserved value. The following table 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<br>Size Status      |
|------------|-------------|--------------------------|----------------------------------|
|            | 1           | 11111111                 | Reserved<br>(meaning<br>Unknown) |
| 0          | 1           | 0000000011111111         | Valid (meaning<br>127 octets)    |
| 00         | 1           | 000000000000000011111111 | Valid (meaning<br>127 octets)    |
| 0          | 1           | 1111111111111111         | Reserved<br>(meaning<br>Unknown) |
| 00         | 1           | 000000001111111111111111 | Valid (16,383<br>octets)         |

## 7. EBML Element Types

EBML Elements are defined by an EBML Schema 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 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. The Master Element MAY also use an unknown length. See [Section 6](#) for rules that apply to elements of unknown length.

The Master Element contains zero, one, or many other elements. EBML Elements contained within a Master Element MUST have the EBMLParentPath of their Element Path equal to the EBMLMasterPath 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 comprised 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.

### **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. The EBML Body MUST NOT contain any data that is not part of an EBML Element. 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.

## **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 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-0-20010502](#)] 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 and persistent.

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="1*1(\EBML\EBMLReadVersion)"
    id="0x42F7" minOccurs="1" maxOccurs="1" range="1" default="1"
    type="uinteger"/>
  <element name="EBMLMaxSizeLength" path="1*1(\EBML\EBMLMaxSizeLength)"
    id="0x42F3" minOccurs="1" maxOccurs="1" range="8" default="8"
    type="uinteger"/>
  <!-- Root Element-->
  <element name="Files" path="*1(\Files)" id="0x1946696C" type="master">
    <documentation lang="en" type="definition">Container of data and
      attributes representing one or many files.</documentation>
  </element>
  <element name="File" path="1*(\Files\File)" id="0x6146" type="master"
    minOccurs="1">
    <documentation lang="en" type="definition">
      An attached file.
    </documentation>
  </element>
  <element name="FileName" path="1*1(\Files\File\FileName)"
    id="0x614E" type="utf-8"
    minOccurs="1">
    <documentation lang="en" type="definition">
      Filename of the attached file.
    </documentation>
  </element>
  <element name="MimeType" path="1*1(\Files\File\MimeType)"
    id="0x464D" type="string"
    minOccurs="1">
    <documentation lang="en" type="definition">
      MIME type of the file.
    </documentation>
  </element>
  <element name="ModificationTimestamp"
    path="1*1(\Files\File\ModificationTimestamp)" id="0x4654"
    type="date" minOccurs="1">
    <documentation lang="en" type="definition">
      Modification timestamp of the file.
```





```
</documentation>
</element>
<element name="Data" path="1*1(\\Files\\File\\Data)" id="0x4664"
  type="binary" minOccurs="1">
  <documentation lang="en" type="definition">
    The data of the file.
  </documentation>
</element>
</EBMLSchema>
```

#### **11.1.2. <EBMLSchema> Element**

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

#### **11.1.3. <EBMLSchema> Attributes**

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

##### **11.1.3.1. 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**

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.

#### **11.1.4. <element> Element**

Each "<element>" defines one EBML Element through the use of several attributes that are defined in [Section 11.1.3](#). 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 and zero or one "<restriction>" sub-element. All "<element>" nodes MUST be sub-elements of the "<EBMLSchema>".

#### [11.1.1.5.](#) <element> Attributes

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

##### [11.1.1.5.1.](#) 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 name attribute is REQUIRED.

##### [11.1.1.5.2.](#) 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           = EBMLElementOccurrence "(" EBMLMasterPath ")"
EBMLMasterPath         = [EBMLParentPath] EBMLElementPath
EBMLParentPath         = EBMLFixedParent EBMLLastParent
EBMLFixedParent        = *(EBMLPathAtom)
EBMLElementPath       = EBMLPathAtom / EBMLPathAtomRecursive
EBMLPathAtom           = PathDelimiter EBMLAtomName
EBMLPathAtomRecursive = "(1*(" EBMLPathAtom "))"
EBMLLastParent         = EBMLPathAtom / EBMLVariableParent
EBMLVariableParent     = "(" VariableParentOccurrence "\"
EBMLAtomName           = 1*(EBMLNameChar)
EBMLNameChar           = ALPHA / DIGIT / "-" / "."
PathDelimiter          = "\"
EBMLElementOccurrence = [EBMLMinOccurrence] "*" [EBMLMaxOccurrence]
EBMLMinOccurrence      = 1*DIGIT
EBMLMaxOccurrence      = 1*DIGIT
VariableParentOccurrence = [PathMinOccurrence] "*" [PathMaxOccurrence]
PathMinOccurrence      = 1*DIGIT
PathMaxOccurrence      = 1*DIGIT
```



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

The EBMLPathAtom part of the EBMLElementPath MUST be equal to the name attribute of the EBML Schema.

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

The EBMLElementOccurrence part is interpreted as an ABNF Variable Repetition. The repetition amounts correspond to how many times the EBML Element can be found in its Parent Element.

The EBMLMinOccurrence represents 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 EBMLMinOccurrence refers to constraints on the occurrence of the EBML Element within the EBML Document. If EBMLMinOccurrence is not present then that EBML Element has an EBMLMinOccurrence value of 0. The semantic meaning of EBMLMinOccurrence within an EBML Schema is analogous to the meaning of minOccurs within an XML Schema. EBML Elements with EBMLMinOccurrence 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 11.1.14](#). An EBML Element defined with a EBMLMinOccurrence value greater than zero is called a Mandatory EBML Element.

The EBMLMaxOccurrence represents 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. If the EBML Element has an empty EBMLParentPath then EBMLMaxOccurrence refers to constraints on the occurrence of the EBML Element within the EBML Document. If EBMLMaxOccurrence is not present then there is no upper bound for the permitted number of occurrences of this EBML Element within its Parent Element resp. within the EBML Document depending on whether the EBMLParentPath of the EBML Element is empty or not. The semantic meaning of EBMLMaxOccurrence within an EBML Schema path is analogous to the meaning of maxOccurs within an XML Schema.

The VariableParentOccurrence part is interpreted as an ABNF Variable Repetition. The repetition amounts correspond to the amount of unspecified Parent Element levels there can be between the EBMLFixedParent and the actual EBMLElementPath.

If the path contains an EBMLPathAtomRecursive part, the EBML Element can occur within itself recursively (see the [Section 11.1.5.11](#)).



As an example, a "path" of "1\*(\Segment\Info)" means the element Info is found inside the Segment elements at least once and with no maximum iteration. An element SeekHead with path "0\*2(\Segment\SeekHead)" may not be found at all in its Segment parent, once or twice but no more than that.

#### **11.1.5.3. 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 id attribute is REQUIRED.

#### **11.1.5.4. minOccurs**

An integer expressing the minimum permitted number of occurrences of this EBML Element within its Parent Element. The minOccurs value MUST be equal to the EBMLMinOccurrence value of the path.

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

#### **11.1.5.5. maxOccurs**

An integer expressing the maximum permitted number of occurrences of this EBML Element within its Parent Element. The maxOccurs value MUST be equal to the EBMLMaxOccurrence value of the path.

The maxOccurs attribute is OPTIONAL. If the maxOccurs attribute is not present then that EBML Element has no maximum occurrence, similar to unbounded in the XML world.

#### **11.1.5.6. 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 11.1.13](#).

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**

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**

If an Element is mandatory (has a EBMLMinOccurrence 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. 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**

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**

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.

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**

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 path contains an EBMLPathAtomRecursive part then the recursive value MUST be true and false otherwise.

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**

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

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**

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**

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**

The "<documentation>" element provides additional information about the EBML Element.

#### [11.1.7.](#) **<documentation> Attributes**



#### [11.1.7.1.](#) **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**

A purpose attribute distinguishes the meaning of the documentation. Values for the sub-element's purpose attribute MUST include one of the following: "definition", "rationale", "usage notes", and "references".

The purpose attribute is REQUIRED.

#### [11.1.8.](#) **<restriction> Element**

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

#### [11.1.9.](#) **<enum> Element**

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>".

#### [11.1.10.](#) **<enum> Attributes**

##### [11.1.10.1.](#) **label**

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

The label attribute is OPTIONAL.

##### [11.1.10.2.](#) **value**

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

The value attribute is REQUIRED.





### [11.1.11.](#) XML Schema for EBML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="https://ietf.org/cellar/ebml"
  targetNamespace="https://ietf.org/cellar/ebml"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" version="01">
  <xsd:element name="EBMLSchema" type="EBMLSchemaType"/>
  <xsd:complexType name="EBMLSchemaType">
    <xsd:sequence>
      <xsd:element name="element" type="elementType" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="docType" use="required"/>
    <xsd:attribute name="version" use="required"/>
  </xsd:complexType>
  <xsd:complexType name="elementType">
    <xsd:sequence>
      <xsd:element name="documentation" type="documentationType"
        minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="restriction" type="restrictionType"
        minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="name" use="required"/>
    <xsd:attribute name="path" use="required"/>
    <xsd:attribute name="id" use="required"/>
    <xsd:attribute name="minOccurs" default="0"/>
    <xsd:attribute name="maxOccurs" default="1"/>
    <xsd:attribute name="range"/>
    <xsd:attribute name="length"/>
    <xsd:attribute name="default"/>
    <xsd:attribute name="type" use="required"/>
    <xsd:attribute name="unknownsizeallowed"/>
    <xsd:attribute name="recursive"/>
    <xsd:attribute name="minver" default="1"/>
    <xsd:attribute name="maxver"/>
  </xsd:complexType>
  <xsd:complexType name="restrictionType">
    <xsd:sequence>
      <xsd:element name="enum" type="enumType" minOccurs="0"
        maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="enumType">
    <xsd:sequence>
      <xsd:element name="documentation" type="documentationType"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
```



```
<xsd:attribute name="label"/>
<xsd:attribute name="value" use="required"/>
</xsd:complexType>
<xsd:complexType name="documentationType" mixed="true">
  <xsd:sequence>
    <xsd:any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="lang"/>
  <xsd:attribute name="purpose" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="definition"/>
        <xsd:enumeration value="rationale"/>
        <xsd:enumeration value="references"/>
        <xsd:enumeration value="usage notes"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>
</xsd:schema>
```

#### **11.1.12. 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.



**11.1.13. 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). The following table provides examples of expressions of float ranges.

| +-----+<br>  as decimal<br>+-----+ | +-----+<br>  as Hexadecimal Floating-Point Constants  <br>+-----+ |
|------------------------------------|---|
| 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"  |
| +-----+                            | +-----+   |

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.

**11.1.14. 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.



This table 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<br>default value<br>declared? | Is the value<br>equal to<br>default? | Is the Parent<br>Element<br>present? | Then is storing<br>the EBML Element<br>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   |

## [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: "1\*1(\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: "1\*1(\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: "1\*1(\\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: "1\*1(\\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: "1\*1(\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: "1\*1(\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: "1\*1(\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: "1\*1(\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: "0\*(\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: "1\*1(\EBML\DocTypeExtension\Name)"

id 0x4283

minOccurs: 1

maxOccurs: 1

length: >0

type: String

description: The name of the DocTypeExtension to identify 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: "1\*1(\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 defines these Global Elements which MAY be stored within any Master Element of an EBML Document as defined by their Element Path.

#### **11.3.1. CRC-32 Element**

name: CRC-32

path: "\*1((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 bitstream 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.

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 that is not a Master Element than permitted according to the maxOccurs attribute of the definition of that Element then all but the instance of that Element with the smallest byte offset 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. An EBML Reader MUST consider the value of the String Element or UTF-8 Element to be terminated upon the first read Null Octet and MUST ignore any data following the first Null Octet within that Element. A string value and a copy of that string value terminated by one or more Null Octets are semantically equal.

The following table 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           |  |  |  |
| +-----+-----+       |  |  |  | +-----+-----+       |  |  |  |

### **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 variable lengths, so for minor reductions in octet length the Element Data Size MAY be written to a longer octet length to fill the freed space.

For example, the first row of the following table 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     |

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, the following table 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       |





### **14.1.3. Terminating Element Data**

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

In the following table, 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   |

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:

- o Invalid Element IDs that are longer than the limit stated in the EBMLMaxIDLength Element of the EBML Header.
- o Invalid Element IDs that are not encoded in the shortest-possible way.
- o 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:



- o 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).
- o Very high lengths in order to force out-of-memory situations resulting in a denial of service, access-out-of-bounds issues etc.
- o 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.
- o 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).
- o Usage of invalid data in EBML Elements with a date type, triggering bogus date accesses.

Side channel attacks could exploit:

- o 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.
- o 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.
- o Data contained within a Master Element which is not itself part of a Child Element can trigger incorrect parsing behavior in EBML Readers.
- o Extraneous copies of Identically Recurring Element, making parsing unnecessarily slow to the point of not being usable.
- o 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.
- o Use of Void Elements which could be used to hide content or create bogus resynchronization points seen by some EBML Reader and not others.

An EBML Reader MAY use the data if it considers it doesn't create any security issue.



## **17. IANA Considerations**

### **17.1. CELLAR EBML Element ID Registry**

This document creates a new IANA Registry called "CELLAR 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.

The VINT Data value of one-octet Element IDs MUST be between 0x01 and 0x7E. These items are valuable because they are short, and need to be used for commonly repeated elements. Values from 1 to 126 are to be allocated according to the "RFC Required" policy [[RFC8126](#)].

The VINT Data value of two-octet Element IDs MUST be between 0x007F and 0x3FFE. Numbers are to be allocated within this range according to the "Specification Required" policy [[RFC8126](#)].

The numbers 0x3FFF and 0x4000 are RESERVED.

The VINT Data value of three-octet Element IDs MUST be between 0x4001 and 0x1FFFFE. Numbers may be allocated within this range according to the "First Come First Served" policy [[RFC8126](#)].

The numbers 0x1FFFFFF and 0x200000 are RESERVED.

Four-octet Element IDs are numbers 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 0x7F) 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 number has some desirable property, such as a specific CRC.





The other three octets, when ALL having values between 0x21 (33, ASCII !) and 0x7E (126, ASCII ~), fall into this category.

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

The numbers 0xFFFFFFFF and 0x1000000 are RESERVED.

Five octet Element IDs (values from 0x10000001 upwards) 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:

| 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  |



## **17.2. CELLAR EBML DocType Registry**

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

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. References**

### **18.1. Normative References**

[IEEE.754.1985]

Institute of Electrical and Electronics Engineers,  
"Standard for Binary Floating-Point Arithmetic",  
IEEE Standard 754, August 1985.

[ISO.3309.1979]

International Organization for Standardization, "Data  
communication - High-level data link control procedures -  
Frame structure", ISO Standard 3309, 1979.

[ISO.9899.2011]

International Organization for Standardization,  
"Programming languages - C", ISO Standard 9899, 2011.

[ITU.V42.1994]

International Telecommunications Union, "Error-correcting  
Procedures for DCEs Using Asynchronous-to-Synchronous  
Conversion", ITU-T Recommendation V.42, 1994.

[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>>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", [BCP 14](#), [RFC 2119](#),  
DOI 10.17487/RFC2119, March 1997,  
<<https://www.rfc-editor.org/info/rfc2119>>.



- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", [RFC 3339](#), DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/info/rfc3339>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 8126](#), DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [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, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.

## **18.2. Informative References**

- [W3C.REC-xmlschema-0-20010502]  
Fallside, D., "XML Schema Part 0: Primer", World Wide Web Consortium Recommendation REC-xmlschema-0-20010502, May 2001, <<http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>>.



### **18.3. URIs**

[1] <https://github.com/Matroska-Org/matroska-specification/>

#### 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)