cellar S. Lhomme
Internet-Draft
Intended status: Standards Track D. Rice
Expires: January 7, 2017 July 6, 2016

# Extensible Binary Meta Language
## draft-lhomme-cellar-ebml-00

Abstract

This document defines the Extensible Binary Meta Language (EBML) format as a genearlized file format for any type of data in a hierarchical form. EBML is designed as a binary equivalent to XML and utilizes a storage-efficient approach to building 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 an EBML Schema to convey the semantics of an EBML Document.

Table of Contents



# 1. EBML specifications

## 1.1. Introduction

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

The goal of the EBML Specification is to define a generic, binary, space-efficient format that may be utilized to define more complex formats (such as containers for multimedia content) using an EBML Schema. 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 may be defined.

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.

## 1.2. 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 [RFC2119](<https://tools.ietf.org/html/rfc2119>).

## 1.3. Security Considerations

EBML itself does not offer any kind of security. It has nothing to do with authentication, it does not provide confidentiality, only marginally useful and effective data integrity options (CRC elements).

EBML does not provide any kind of authorization.

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

Attacks on an EBML reader may include: - 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 IDs comprised of reserved values. - Invalid Element Data Size values that are longer than the limit stated in the EBMLMaxSizeLength Element of the EBML Header. - Invalid Element Data Size values (e.g. extending the length of the 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 Elements that are mandatory and have no declared default value. - Usage of "0x00" octets in EBML Elements with a string type. - 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.

## 1.4. Structure

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

## 1.5. Variable Size Integer

The Element ID and Element Data Size are both encoded as a Variable Size Integer, developed according to a UTF-8 like system. The Variable Size Integer is composed of a VINT_WIDTH, VINT_MARKER, and VINT_DATA, in that order. Variable Size Integers shall be referred to as VINT for shorthand.

### 1.5.1. 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.

### 1.5.2. 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 MUST be 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.

### 1.5.3. 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 combined use one bit per octet 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 may be zero-padded to the left to a size that fits. The VINT_DATA value MUST be expressed a big-endian unsigned integer.

### 1.5.4. VINT Examples

This table shows examples of Variable Size Integers at widths of 1 to 5 octets. 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 Width | Size | Representation |
|---|---|---|
| 1 | 2^7 | 1xxx xxxx |
| 2 | 2^14 | 01xx xxxx xxxx xxxx |
| 3 | 2^21 | 001x xxxx xxxx xxxx xxxx xxxx |
| 4 | 2^28 | 0001 xxxx xxxx xxxx xxxx xxxx xxxx xxxx |
| 5 | 2^35 | 0000 1xxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx |

Note that data encoded as a Variable Size Integer may be rendered at octet widths larger than needed to store the data. In this table a binary value of 0b10 is shown encoded as different Variable Size Integers with widths from one octet to four octet. All four encoded examples have identical semantic meaning though the VINT_WIDTH and the padding of the VINT_DATA vary.

| Binary Value | Octet Width | 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 |

## 1.6. Element ID

The Element ID MUST be encoded as a Variable Size Integer. By default, EBML Element IDs may be encoded in lengths from one octet to four octets, although Element IDs of greater lengths may be used if the octet length of the EBML Document's longest Element ID is declared in the EBMLMaxIDLength Element of the EBML Header. The VINT_DATA component of the Element ID MUST NOT be set to either all zero values or all one 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. The following table details this specific example further:

| VINT_WIDTH | VINT_MARKER | VINT_DATA | Element ID Status |
|---|---|---|---|
| | 1 | 0111111 | Valid |
| 0 | 1 | 00000000111111 | Invalid |

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

| EBML Class | Octet Length | Number of Possible Element IDs |
|---|---|---|
| Class A | 1 | 2^7 - 2 = 126 |
| Class B | 2 | 2^14 - 2^7 - 1 = 16,255 |
| Class C | 3 | 2^21 - 2^14 - 1 = 2,080,767 |
| Class D | 4 | 2^28 - 2^21 - 1 = 266,388,303 |

## 1.7. Element Data Size

The Element Data Size expresses the length in octets of Element Data. The Element Data Size itself MUST be encoded as a Variable Size Integer. By default, EBML 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 EBML Document's longest Element Data Size is declared in the EBMLMaxSizeLength Element of the EBML Header. Unlike the VINT_DATA of the Element ID, the VINT_DATA component of the Element Data Size is not required 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.

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 Data Types which do not mandate a non-zero length. An Element Data Size with all VINT_DATA bits set to zero indicates that the Element Data of the Element is zero octets in length. Such an Element is referred to as an Empty Element. If an Empty Element has a "default" value declared then that default value MUST be interpreted as the value of the Empty Element. If an Empty Element has no "default" value declared then the semantic meaning of Empty Element is defined as part of the definition of the EBML Element Types.

An Element Data Size with all VINT_DATA bits set to one is reserved as an indicator that the size of the Element is unknown. The only reserved value for the VINT_DATA of Element Data Size is all bits set to one. This rule allows for an Element to be written and read before the size of the Element is known; however unknown Element Data Size values SHOULD NOT be used unnecessarily. An Element with an unknown Element Data Size MUST be a Master-element in that it contains other EBML Elements as sub-elements. Master-elements MAY only use an unknown size if the "unknownsizeallowed" attribute of the EBML Schema is set to true. The end of a Master-element with unknown size is determined by the beginning of the next element that is not a valid sub-element of that Master-element. An Element with an unknown Element Data Size is referred to as an "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. For example, an Element with an octet length of 127 MUST NOT be encoded in an Element Data Size encoding with a one octet length. 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).

| VINT_WIDTH | VINT_MARKER | VINT_DATA | Element Data Size Status |
|---|---|---|---|
| | 1 | 1111111 | Reserved (meaning Unknown) |
| 0 | 1 | 00000001111111 | Valid (meaning 127 octets) |

## 1.8. EBML Element Types

Each defined EBML Element MUST have a declared Element Type. The Element Type defines a concept for storing data that may be constrained by length, endianness, and purpose.

| Element Data Type | Signed Integer |
|---|---|
| Endianness | Big-endian |
| Length | A Signed Integer Element MUST declare a length that is no greater than 8 octets. An Signed Integer Element with a zero-octet length represents an integer value of zero. |
| Definition | A Signed Integer stores an integer (meaning that it can be written without a fractional component) which may be negative, positive, or zero. Because EBML limits Signed Integers to 8 octets in length a Signed Element may store a number from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807. |

| Element Data Type | Unsigned Integer |
|---|---|
| Endianness | Big-endian |
| Length | A Unsigned Integer Element MUST declare a length that is no greater than 8 octets. An Unsigned Integer Element with a zero-octet length represents an integer value of zero. |
| Definition | An Unsigned Integer stores an integer (meaning that it can be written without a fractional component) which may be positive or zero. Because EBML limits Unsigned Integers to 8 octets in length an unsigned Element may store a number from 0 to 18,446,744,073,709,551,615. |

| Element Data Type | Float |
|---|---|
| Endianness | Big-endian |
| Length | A Float Element MUST declare of length of either 0 octets (0 bit), 4 octets (32 bit) or 8 octets (64 bit). A Float Element with a zero-octet length represents a numerical value of zero. |
| Definition | A Float Elements stores a floating-point number as defined in IEEE 754. |

| Element Data Type | String |
|---|---|
| Endianness | None |
| Length | A String Element may declare any length from zero to "VINTMAX". |
| Definition | A String Element may either be empty (zero-length) or contain Printable ASCII characters in the range of "0x20" to "0x7E". Octets with all bits set to zero may follow the string value when needed. |

| Element Data Type | UTF-8 |
| --- | --- |
| Endianness | None |
| Length | A UTF-8 Element may declare any length from zero to "VINTMAX". |
| Definition | A UTF-8 Element shall contain only a valid Unicode string as defined in [RFC2279](<http://www.faqs.org/rfcs/rfc2279.html>). Octets with all bits set to zero may follow the UTF-8 value when needed. |

| Element Data Type | Date |
| --- | --- |
| Endianness | None |
| Length | A Date Element MUST declare a length of either 0 octets or 8 octets. A Date Element with a zero-octet length represents a timestamp of 2001-01-01T00:00:00.000000000 UTC. |
| Definition | The Date Element MUST contain a Signed Integer 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. |

| Element Data Type | Master-element |
| --- | --- |
| Endianness | None |
| Length | A Master-element may declare any length from zero to "VINTMAX". The Master-element may also use an unknown length. See the section on Element Data Size for rules that apply to elements of unknown length. |
| Definition | The Master-element contains zero, one, or many other elements. Elements contained within a Master-element must be defined for use at levels greater than the level of the Master-element. For instance is a Master-element occurs on level 2 then all contained Elements must be valid at level 3. 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. When EBML is used in transmission or streaming, data that is not part of an EBML Element is permitted to be present within a Master-element if "unknownsizeallowed" is enabled within that Master-element's definition. In this case, the reader should skip data until a valid Element ID of the same level or the next greater level of the Master-element is found. What Element IDs are considered valid within a Master-element is identified by the EBML Schema for that version of the EBML Document Type. Any data contained with a Master-element that is not part of an Element SHOULD be ignored. |

| Element Data Type | Binary |
| --- | --- |
| Endianness | None |
| Length | A binary element may declare any length from zero to "VINTMAX". |
| Definition | The contents of a Binary element should not be interpreted by the EBML parser. |

## 1.9. 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 which declares significant characteristics of the entire EBML Body.

An EBML Document MAY only consist of EBML Elements and MUST NOT contain any data that is not part of an EBML Element. The initial EBML Element of an EBML Document and the Elements that follow it are considered Level 0 Elements. If an EBML Master-element is considered to be at level N and it contains one or many other EBML Elements then the contained Elements shall be considered at Level N+1. Thus a Level 2 Element would have to be contained by a Master-element (at Level 1) that is contained by another Master-element (at Level 0).

### 1.9.1. EBML Header

The EBML Header is a declaration that provides processing instructions and identification of the EBML Body. The EBML Header may be considered as analogous to an XML Declaration. All EBML Documents MUST begin with a valid EBML Header.

The EBML Header documents the EBML Schema (also known as the EBML DocType) that may be 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 consists of a single Master-element with an Element ID of 'EBML'. The EBML Header MUST ONLY contain EBML Elements that are defined as part of the EBML Specification.

All EBML Elements within the EBML Header MUST NOT utilize any Element ID with a length greater than 4 octets. All EBML Elements within the EBML Header MUST NOT utilize any Element Data Size with a length greater than 4 octets.

### 1.9.2. EBML Body

All data of an EBML Document following the EBML Header may be considered the EBML Body. The end of the EBML Body, as well as the end of the EBML Document that contains the EBML Body, is considered as whichever comes first: the beginning of a new level 0 EBML Header or the end of the file. The EBML Body MAY only consist of EBML Elements and MUST NOT contain any data that is not part of an EBML Element. Although the EBML specification itself defines precisely what EBML Elements are to be used within the EBML Header, the EBML specification does not name or define what EBML Elements are to be used within the EBML Body. The definition of what EBML Elements are to be used within the EBML Body is defined by an EBML Schema.

## 1.10. EBML Stream

An EBML Stream is a file that consists of one or many EBML Documents that are concatenated together. An occurrence of a Level 0 EBML Header marks the beginning of an EBML Document.

## 1.11. Elements semantic

### 1.11.1. EBML Schema

An EBML Schema is an 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 may be considered analogous to the relationship of an XML Schema [1] to an XML Document [2]. An EBML Schema MUST be clearly associated with one or many EBML Document Types. An EBML Schema must be expressed as well-formed XML. An EBML Document Type is identified by a unique string stored within the EBML Header element called DocType; for example "matroska" or "webm".

As an XML Document, the EBML Schema MUST use "<EBMLSchema>" as the top level element. The "<EBMLSchema>" element MAY contain "<element>" sub-elements. Each "<element>" defines one EBML Element through the use of several attributes which are defined in the section on Section 1.11.1.1. EBML Schemas MAY contain additional attributes to extend the semantics but MUST NOT conflict is the definitions of the "<element>" attributes defined within this specification.

Within the EBML Schema each EBML Element is defined to occur at a specific level. For any specificied EBML Element that is not at level 0, the Parent EBML Element refers to the EBML Master-element that that EBML Element is contained within. For any specifiied EBML Master-element the Child EBML Element refers to the EBML Elements that may be immediately contained within that Master-element. For any EBML Element that is not defined at level 0, the Parent EBML Element may be identified by the preceding "<element>" node which has a lower value as the defined "level" attribute. The only exception for this rule are Global EBML Elements which may occur within any Parent EBML Element within the restriction of the Global EBML Element's range declaration.

An EBML Schema MUST declare exactly one Element at Level 0 (referred to as the Root Element) that MUST occur exactly once within an EBML Document. The Root Element MUST be mandatory (with minOccurs set to 1) and MUST be defined to occur exactly once (maxOccurs set to 1). Note that the EBML and Void Elements may also occur at Level 0 but are not considered to be Root Elements.

Elements defined to only occur at Level 1 are known as Top-Level Elements.

The EBML Schema does not itself document the EBML Header, but documents all data of the EBML Document that follows the EBML Header. The EBML Header itself is documented by this specification in the Section 1.11.2 section. The EBML Schema also does not document Global Elements that are defined by the EBML Specification (namely Void and CRC-32).

#### 1.11.1.1. EBML Schema Element Attributes

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

| attribute name | required | definition |
|---|---|---|
| docType | Yes | The "docType" lists the official name of the EBML Document Type that is defined by the EBML Schema; for example, "<EBMLSchema docType="matroska">". |
| version | Yes | The "version" lists an incremental 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. Elements may be introduced and deprecated by using the "minver" and "maxver" attributes of . |

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

| attribute name | required | definition |
|---|---|---|
| name | Yes | The official human-readable name of the EBML Element. The value of the name MUST be in the form of an NCName as defined by the XML Schema specification [3]. |
| level | Yes | The level notes at what hierarchical depth the EBML |

| | | |
|---|---|---|
| | | Element may occur within an EBML Document. The Root Element of an EBML Document is at level 0 and the Elements that it may contain are at level 1. The level MUST be expressed as an integer. Note that Elements defined as "global" and "recursive" MAY occur at a level greater than or equal to the defined "level". |
| global | No | A boolean to express if an EBML Element MUST occur at its defined level or may occur within any Parent EBML Element. If the "global" attribute is not expressed for that Element then that element is to be considered not global. |
| id | Yes | The Element ID expressed in hexadecimal notation prefixed by a "0x". To reduce the risk of false positives while parsing EBML Streams, the IDs of the Root Element and Top-Level Elements SHOULD be at least 4 octets in length. Element IDs defined for use at Level 0 or Level 1 MAY use shorter octet lengths to facilitate padding and optimize edits to EBML Documents; for instance, the EBML Void Element uses an Element ID with a one octet length to allow its usage in more writing and editing scenarios. |
| minOccurs | No | An integer to express the minimal number of occurrences that the EBML Element MUST occur within its Parent Element if its Parent Element is used. If the Element has no Parent Level (as is the case with Elements at Level 0), then minOccurs refers to constaints on the Element's occurrence within the EBML Document. If the minOccurs attribute is not expressed for that Element then that Element |

| | | |
|---|---|---|
| | | shall be considered to have a minOccurs value of 0. This value of minOccurs MUST be a positive integer. The semantic meaning of minOccurs within an EBML Schema is considered analogous to the meaning of minOccurs within an XML Schema [4]. Note that Elements with minOccurs set to "1" that also have a default value declared are not required to be stored but are required to be interpretted, see the Section 1.11.1.6. |
| maxOccurs | No | A value to express the maximum number of occurrences that the EBML Element MAY occur within its Parent Element if its Parent Element is used. If the Element has no Parent Level (as is the case with Elements at Level 0), then maxOccurs refers to constaints on the Element's occurrence within the EBML Document. This value may be either a positive integer or the term "unbounded" to indicate there is no maximum number of occurrences or the term "identical" to indicate that the Element is an Section 1.11.1.3. If the maxOccurs attribute is not expressed for that Element then that Element shall be considered to have a maxOccurs value of 1. The semantic meaning of maxOccurs within an EBML Schema is considered analogous to the meaning of minOccurs within an XML Schema [5], with EBML Schema adding the concept of Identically Recurring Elements. |
| range | No | For Elements which are of numerical types (Unsigned Integer, Signed Integer, Float, and Date) a numerical range may be specified. If specified that the value of the EBML Element |

| | | |
|---|---|---|
| | | MUST be within the defined range inclusively. See the Section 1.11.1.4 for rules applied to expression of range values. |
| default | No | A default value may be provided. If an Element is mandatory but not written within its Parent EBML Element, then the parser of the EBML Document MUST insert the defined default value of the Element. EBML Elements that are Master-elements MUST NOT declare a default value. |
| type | Yes | As defined within the Section 1.8, 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'. |
| unknownsizeallowed | No | A boolean to express if an EBML Element MAY be used as an "Unknown-Sized Element" (having all VINT_DATA bits of Element Data Size set to 1). The "unknownsizeallowed" attribute only applies to Master-elements. If the "unknownsizeallowed" attribute is not used it is assumed that the element is not allowed to use an unknown Element Data Size. |
| recursive | No | A boolean to express if an EBML Element MAY be stored recursively. In this case the Element MAY be stored at levels greater that defined in the "level" attribute if the Element is a Child Element of a Parent Element with the same Element ID. The "recursive" attribute only applies to Master-elements. If the "recursive" attribute is not used it is assumed that the element is not allowed to be used recursively. |
| minver | No | The "minver" (minimum version) attribute stores a non-negative |

| | | |
|---|---|---|
| | | integer that represents the first version of the docType to support the element. If the "minver" attribute is not used it is assumed that the element has a minimum version of "1". |
| maxver | No | 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. If the "maxver" attribute is not used it is assumed that the element has a maximum version equal to the value stored in the "version" attribute of . |

The "<element>" nodes shall contain a description of the meaning and use of the EBML Element stored within one or many "<documentation>" sub-elements. The "<documentation>" sub-element may use a "lang" attribute which may be set to the RFC 5646 value of the language of the element's documentation. The "<documentation>" sub-element may use a "type" attribute to distinguish the meaning of the documentation. Recommended values for the "<documentation>" sub-element's "type" attribute include: "definition", "rationale", "usage notes", and "references".

The "<element>" nodes MUST be arranged hierarchically according to the permitted structure of the EBML Document Type. An "<element>" node that defines an EBML Element which is a Child Element of another Parent Element MUST be stored as an immediate sub-element of the "<element>" node that defines the Parent Element. "<element>" nodes that define Level 0 Elements and Global Elements should be sub-elements of "<EBMLSchema>".

#### 1.11.1.2. EBML Schema Example

```
<?xml version="1.0" encoding="utf-8"?>
<EBMLSchema docType="files-in-ebml-demo" version="1">
 <!-- Root Element-->
 <element name="Files" level="0" id="0x1946696C" type="master">
  <documentation lang="en" type="definition">Container of data and
  attributes representing one or many files.</documentation>
  <element name="File" level="1" id="0x6146" type="master" minOccurs="1"
  maxOccurs="unbounded">
   <documentation lang="en" type="definition">An attached file.
   </documentation>
   <element name="FileName" level="2" id="0x614E" type="utf-8"
   minOccurs="1">
    <documentation lang="en" type="definition">Filename of the attached
    file.</documentation>
   </element>
   <element name="MimeType" level="2" id="0x464D" type="string"
     minOccurs="1">
    <documentation lang="en" type="definition">MIME type of the
    file.</documentation>
   </element>
   <element name="ModificationTimestamp" level="2" id="0x4654"
     type="date" minOccurs="1">
    <documentation lang="en" type="definition">Modification timestamp of
    the file.</documentation>
   </element>
   <element name="Data" level="2" id="0x4664" type="binary"
     minOccurs="1">
    <documentation lang="en" type="definition">The data of the
    file.</documentation>
   </element>
  </element>
 </element>
</EBMLSchema>
```

#### 1.11.1.3. Identically Recurring Elements

An Identically Recurring Element is an 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. 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 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 Child Element or if CRC-32 Child Elements are present but none are valid then the first instance of the Identically Recurring Element should be used for interpretation.

#### 1.11.1.4. Expression of range

The "range" attribute MUST only be used with EBML Elements that are either "signed integer", "unsigned integer", or "float". The "range" attribute does not support date EBML Elements. The "range" expression may contain whitespace for readability but whitespace within a "range" expression MUST NOT convey meaning. The expression of the "range" MUST adhere to one of the following forms:

- o "x-y" where x and y are integers or floats and "y" must be greater than "x", meaning that the value must be greater than or equal to "x" and less than or equal to "y".

- o ">x" where "x" is an integer or float, meaning that the value MUST be greater than "x".

- o "x" where "x" is an integer or float, meaning that the value MUST be equal "x".

The "range" may use the prefix "not" to indicate that the expressed range is negated. Please also see the section on Section 1.11.1.5.

#### 1.11.1.5. 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 a Hexadecimal Floating-Point Constants as defined in the C11 standard ISO/IEC 9899:2011 [6] (see section 6.4.4.2 on Floating Constants). The following table provides examples of expressions of float ranges.

| as decimal | as Hexadecimal Floating-Point Constants |
|---|---|
| 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. Note that 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.

#### 1.11.1.6. 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 EBML Element's value is equal to the declared default value then that Element is not required to be present within the EBML Document if its Parent EBML Element is present. In this case, the default value of the Mandatory EBML Element may be assumed although the EBML Element is not present within its Parent EBML Element. Also in this case the parser of the EBML Document MUST insert the defined default value of the Element.

If a Mandatory EBML Element has no default value declared by an EBML Schema and its Parent EBML Element is present than 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 EBML Element is present and the EBML Element's value is NOT equal to the declared default value then the EBML Element MUST be used.

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 EBML Element is used.

| Is the default value declared? | Is the value equal to default? | Is the Parent Element used? | 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 |
| No | n/a | Yes | Yes |
| No | n/a | No | No |

#### 1.11.1.7. Note on the Use of default attributes to define non-Mandatory EBML Elements

If an EBML Element is not Mandatory, has a defined default value, and is an Empty EBML Element then the EBML Element MUST be interpreted as expressing the default value.

### 1.11.2. EBML Header Elements

This specification here contains definitions of all EBML Elements of the EBML Header.

| Name | EBML |
|---|---|
| Level | 0 |
| EBML ID | "0x1A45DFA3" |
| Mandatory | Yes |
| Multiple | No |
| Range | - |
| Default | - |
| Type | Master-element |
| Description | Set the EBML characteristics of the data to follow. Each EBML Document has to start with this. |

| Name | EBMLVersion |
| --- | --- |
| Level | 1 |
| EBML ID | "0x4286" |
| Mandatory | Yes |
| Multiple | No |
| Range | 1 |
| Default | 1 |
| Type | Unsigned Integer |
| Description | The version of EBML parser used to create the EBML Document. |

| Name | EBMLReadVersion |
| --- | --- |
| Level | 1 |
| EBML ID | "0x42F7" |
| Mandatory | Yes |
| Multiple | No |
| Range | 1 |
| Default | 1 |
| Type | Unsigned Integer |
| Description | The minimum EBML version a parser has to support to read this EBML Document. |

| Name | EBMLMaxIDLength |
| --- | --- |
| Level | 1 |
| EBML ID | "0x42F2" |
| Mandatory | Yes |
| Multiple | No |
| Range | >3 |
| Default | 4 |
| Type | Unsigned Integer |
| Description | The EBMLMaxIDLength is the maximum length in octets of the Element IDs to be found within the EBML Body. An EBMLMaxIDLength value of four is recommended, though larger values are allowed. |

| | |
|---|---|
| Name | EBMLMaxSizeLength |
| Level | 1 |
| EBML ID | "0x42F3" |
| Mandatory | Yes |
| Multiple | No |
| Range | >0 |
| Default | 8 |
| Type | Unsigned Integer |
| Description | The EBMLMaxSizeLength is the maximum length in octets of the expression of all Element Data Sizes to be found within the EBML Body. To be clear EBMLMaxSizeLength documents the maximum 'length' of all Element Data Size expressions within the EBML Body and not the maximum 'value' of all Element Data Size expressions within the EBML Body. Elements that have a Element Data Size expression which is larger in octets than what is expressed by EBMLMaxSizeLength SHALL be considered invalid. |

| | |
|---|---|
| Name | DocType |
| Level | 1 |
| EBML ID | "0x4282" |
| Mandatory | Yes |
| Multiple | No |
| Range | - |
| Default | matroska |
| Type | String |
| Description | A string that describes and identifies the content of the EBML Body that follows this EBML Header. |

| Name | DocTypeVersion |
| --- | --- |
| Level | 1 |
| EBML ID | "0x4287" |
| Mandatory | Yes |
| Multiple | No |
| Range | - |
| Default | 1 |
| Type | Unsigned Integer |
| Description | The version of DocType interpreter used to create the EBML Document. |

| Name | DocTypeReadVersion |
| --- | --- |
| Level | 1 |
| EBML ID | "0x4285" |
| Mandatory | Yes |
| Multiple | No |
| Range | - |
| Default | 1 |
| Type | Unsigned Integer |
| Description | The minimum DocType version an interpreter has to support to read this EBML Document. |

#### 1.11.3. Global elements (used everywhere in the format)

| | |
|---|---|
| Name | CRC-32 |
| Level | 1+ |
| Global | Yes |
| EBML ID | "0xBF" |
| Mandatory | No |
| Range | - |
| Default | - |
| 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 Element within its Parent Element for easier reading. All Top-Level Elements of an EBML Document 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 standard and in section 8.1.1.6.2 of ITU-T recommendation V.42, with initial value of "0xFFFFFFFF". The CRC value MUST be computed on a little endian bitstream and MUST use little endian storage. |

| | |
|---|---|
| Name | Void |
| Level | 0+ |
| Global | Yes |
| EBML ID | "0xEC" |
| Mandatory | No |
| Multiple | Yes |
| Range | - |
| Default | - |
| Type | Binary |
| Description | Used to void damaged data, to avoid unexpected behaviors when using damaged data. The content is discarded. Also used to reserve space in a sub-element for later use. |

## 2. References

### 2.1. Normative References

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

### 2.2. Informative References

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, DOI 10.17487/RFC2279, January 1998, <http://www.rfc-editor.org/info/rfc2279>.

### 2.3. URIs

[1] http://www.w3.org/XML/Schema#dev

[2] http://www.w3.org/TR/xml/

[3] http://www.w3.org/TR/1999/REC-xml-names-19990114/#ns-decl

[4] https://www.w3.org/TR/xmlschema-0/#ref6

[5] https://www.w3.org/TR/xmlschema-0/#ref6

[6] http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf



Authors' Addresses

Steve Lhomme

Dave Rice