cellar
Internet-Draft
Intended status: Standards Track
Expires: April 14, 2017

S. Lhomme

D. Rice

M. Bunkus

October 11, 2016

Extensible Binary Meta Language
draft-ietf-cellar-ebml-01

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 http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on April 14, 2017.

Copyright Notice

Table of Contents

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 (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 can 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.

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].

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

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

"EBML": Extensible Binary Meta Language

"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".

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

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

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

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

"EBML Document Type": An "EBML Document Type" is a name provided by an "EBML Schema" for a particular implementation of "EBML" for a data format (examples: matroska and webm).

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

"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.

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

"EBML Stream": An "EBML Stream" is a file that consists of one or many "EBML Documents" that are concatenated together.

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

"Element Name": The official 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 "Empty Element" is 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.

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

"Parent Element": A relative term to describe the "Master Element" which contains a specified element.

"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".

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

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

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

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

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

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

3. 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 (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" could 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 "EBML Element" beyond the scope of the "Parent Element"; possibly triggering access-out-of-bounds issues).
- Very high lengths in order to force out-of-memory situations resulting in a denial of service, access-out-of-bounds issues etc.
- Missing "EBML Elements" that are mandatory 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.

Side channel attacks could exploit:

- The semantic equivalence of the same string stored in a "String Element" or "UTF-8 Element" with and without zero-bit padding.
- The semantic equivalence of "VINT_DATA" within "Element Data Size" with to different lengths due to left-padding zero bits.
- Data contained within a "Master Element" which is not itself part of an "EBML Element".
- Extraneous copies of "Identically Recurring Element".
- Copies of "Identically Recurring Element" within a "Parent Element" that contains an invalid "CRC-32 Elements".
- Use of "Void Elements".

4. 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 many other "EBML Elements", or both.

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 left-pad the "VINT_DATA" value with zero bits so that the whole "Variable Size Integer" is octet-aligned. "Variable Size Integers" SHALL be referred to as "VINT" for shorthand.

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.

Within the "EBML Header" the "VINT_WIDTH" MUST NOT exceed three bits in length (meaning that the "Variable Size Integer" MUST NOT exceed four octets in length). Within the "EBML Body", when "VINTs" are 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 "VINTs" are 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".

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".

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 out of 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" SHOULD be zero-padded to the left to a size that fits. The "VINT_DATA" value MUST be expressed as a big-endian unsigned integer.

5.4. VINT Examples

This table shows examples of "Variable Size Integers" with lengths from 1 to 5 octets. The Size column refers to the size of the "VINT_DATA" in bits. 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	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

Data encoded as a "Variable Size Integer" MAY be rendered at octet lengths larger than needed to store the data. In this table a binary value of "0b10" is shown encoded as different "Variable Size Integers" with lengths 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 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

6. Element ID

The "Element ID" MUST be 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 are used if the octet length of the longest "Element ID" of the "EBML Document" is declared in the "EBMLMaxIDLength Element" of the "EBML Header" (see Section 11.2.4). 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. 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 "VINT" encoding is possible.

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	00000000000000	Invalid: VINT_DATA MUST NOT be set to all 0
	1	0000001	Valid
0	1	00000000000001	Invalid: A shorter VINT_DATA encoding is available.
	1	0111111	Valid
0	1	00000000111111	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	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 = 266,338,303

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, "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 "0b0111111", 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 8). 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 interpret the value of the "Empty Element" as defined as part of the definition of the corresponding "EBML Element Type" associated with the "Element ID".

An "Element Data Size" with all "VINT_DATA" bits set to one is reserved as an indicator that the size of the "EBML Element" is unknown. The only reserved value for the "VINT_DATA" of "Element Data Size" is all bits set to one. An "EBML Element" with an unknown "Element Data Size" is referred to as an "Unknown-Sized Element". Only "Master Elements" SHALL be "Unknown-Sized Elements". "Master Elements" MUST NOT use an unknown size unless the "unknownsizeallowed" attribute of their "EBML Schema" is set to true (see Section 11.1.4.10). 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 the beginning of the next element, defined by this document or the corresponding "EBML Schema", that is not a valid sub-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. For example, an "EBML 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)

8. EBML Element Types

"EBML Element" are defined by an "EBML Schema" which MUST declare one of the follow "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.

8.1. 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 MUST be 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 Ingeter Element" stores a number from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807.

8.2. Unsigned Integer Element

A "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.

8.3. Float Element

A "Float Element" MUST declare a length of either zero octets (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].

8.4. 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". Octets with all bits set to zero MAY follow the string value when needed, such as reducing the length of a stored string while maintaining the same "Element Data Size". A string with one or more octets with all bits set to zero and a string without one or more octets with all bits set to zero are semantically equal.

8.5. 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 [RFC2279]. Octets with all bits set to zero MAY follow the string value when needed, such as reducing the length of a stored UTF-8 data while maintaining the same "Element Data Size". A UTF-8 value with one or more octets with all bits set to zero and a UTF-8 value without one or more octets with all bits set to zero are semantically equal.

8.6. 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.

8.7. 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 7 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" equals to the "EBMLReferencePath" of the "Master Element" "Element Path" (see Section 11.1.4.2). "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". 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 the definition for that "Master Element". In this case, the "EBML Reader" should skip data until a valid "Element ID" of the same "EBMLParentPath" or the next upper level "Element Path" 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 within a "Master Element" that is not part of a "Child Element" MUST be ignored.

8.8. 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".

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" 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".

9.1. 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" consists of a single "Master Element" with an "Element Name" of "EBML" and "Element ID" of "0x1A45DFA3" (see Section 11.2.1). The "EBML Header" MUST only contain "EBML Elements" that are defined as part of this document.

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

9.2. 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 considered as whichever comes first: the beginning of a new "EBML Header" at the "Root Level" or the end of the file. The "EBML Body" MUST consist only of "EBML Elements" and MUST NOT contain any data that is not part of an "EBML Element". This document defines precisely what "EBML Elements" are to be used within the "EBML Header", but 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".

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 "EBML Header" at the "Root Level" marks the beginning of an "EBML Document".

11. Elements semantic

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 [W3C.REC-xmlschema-0-20010502] to an XML Document [W3C.REC-xml-20081126]. 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 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" SHOULD be unique and persistent.

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. For any specified "Master Element" the "Child Element" refers to the "EBML Element" that is immediately contained within that "Master Element". For any "EBML Element" that is not defined at "Root Level", the "Parent Element" is identified by the

"<element>" node which has the "EBMLReferencePath" equals to the "EBMLParentPath" of that "EBML Element" "Element Path". The only exception for this rule are "Global Elements" which MAY occur within any "Parent Element" within the restriction of the "level" declaration of "Global Element".

An "EBML Schema" MUST declare exactly one "EBML Element" at "Root Level" (referred to as the "Root Element") that MUST occur exactly once within an "EBML Document". The "Void Element" MAY also occur at "Root Level" but is not considered to be "Root Elements" (see Section 11.3.1).

"EBML 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 "EBML Header Elements" (see Section 11.2). The "EBML Schema" also does not document "Global Elements" that are defined by this document (namely the "Void Element" and the "CRC-32 Element").

11.1.1. Element

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

11.1.2. Attributes

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

11.1.2.1. 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.

11.1.2.2. version

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". "EBML Elements" may be introduced and deprecated by using the "minver" and "maxver" attributes of "<element>".

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

11.1.3. Element

Each "<element>" defines one "EBML Element" through the use of several attributes that are defined in Section 11.1.2. "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 many "<documentation>" sub-elements. All "<element>" nodes MUST be sub-elements of the "<EBMLSchema>".

11.1.4. Attributes

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

11.1.4.1. name

The "name" provides the official 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.

11.1.4.2. 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            = EBMLElementOccurence "(" EBMLReferencePath ")"
EBMLReferencePath       = [EBMLParentPath] EBMLElementPath
EBMLParentPath          = EBMLFixedParent EBMLLastParent
EBMLFixedParent         = *(EBMLPathAtom)
EBMLElementPath         = EBMLPathAtom / EBMLPathAtomRecursive
EBMLPathAtom            = PathDelimiter EBMLAtomName
EBMLPathAtomRecursive   = "1*(" EBMLPathAtom ")"
EBMLLastParent          = EBMLPathAtom / EBMLVariableParent
EBMLVariableParent      = "(" VariableParentOccurence "\)"
EBMLAtomName            = 1*(EBMLNameChar)
EBMLNameChar            = ALPHA / DIGIT / "-" / "."
PathDelimiter           = "\"
EBMLElementOccurence    = [EBMLMinOccurence] "*" [EBMLMaxOccurence]
EBMLMinOccurence        = 1*DIGIT
EBMLMaxOccurence        = 1*DIGIT
VariableParentOccurence = [PathMinOccurence] "*" [PathMaxOccurence]
PathMinOccurence        = 1*DIGIT
PathMaxOccurence        = 1*DIGIT
```

The ""*"", ""("" and "")"" symbols MUST be interpreted as they are defined in the ABNF.

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 "EBMLElementOccurence" 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 "Parent Element".

The "EBMLMinOccurence" represents the minimum 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 "EBMLMinOccurence" refers to constaints on the occurrence of the "EBML Element" within the "EBML Document". If "EBMLMinOccurence" is not present then that "EBML Element" is considered to have a "EBMLMinOccurence" value of 0. The semantic meaning of "EBMLMinOccurence" within an "EBML Schema" is considered analogous to the meaning of "minOccurs" within an "XML Schema". "EBML Elements" with "EBMLMinOccurence" set to "1" that also have a "default" value (see Section 11.1.4.8) declared are not REQUIRED to be stored but are REQUIRED to be interpreted, see Section 11.1.11. An "EBML Element" defined with a "EBMLMinOccurence" value greater than zero is called a "Mandatory EBML Element".

The "EBMLMaxOccurence" represents the maximum 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 "EBMLMaxOccurence" refers to constaints on the occurrence of the "EBML Element" within the "EBML Document". If "EBMLMaxOccurence" is not present then that "EBML Element" is considered to have an unbounded "EBMLMaxOccurence" value. The semantic meaning of "EBMLMaxOccurence" within an "EBML Schema path" is considered analogous to the meaning of "maxOccurs" within an "XML Schema".

The "VariableParentOccurence" 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 a "EBMLPathAtomRecursive" part, the "EBML Element" can occur within itself recursively (see the Section 11.1.4.11).

11.1.4.3. 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.

11.1.4.4. minOccurs

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

The "minOccurs" attribute is OPTIONAL. If the "minOccurs" attribute is not present then that "EBML Element" is considered to have a "minOccurs" value of 0.

11.1.4.5. maxOccurs

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

The "maxOccurs" attribute is OPTIONAL. If the "maxOccurs" attribute is not present then that "EBML Element" is considered to have a maxOccurs value of 1.

11.1.4.6. 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 11.1.9 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.

11.1.4.7. size

A value to express the valid length of the "Element Data" as written measured in octets. The "size" provides a constraint in addition to the Length value of the definition of the corresponding "EBML Element Type". This "size" MUST be expressed as either a non-negative integer or a range (see Section 11.1.9) that consists of only non-negative integers and valid operators.

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

11.1.4.8. default

If an Element is mandatory (has a "EBMLMinOccurence" 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.

The "default" attribute is OPTIONAL.

11.1.4.9. 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 8.

The "type" attribute is REQUIRED.

11.1.4.10. unknownsizeallowed

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). "EBML Elements" that are not "Master Elements" MUST NOT set "unknownsizeallowed" to true.

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".

11.1.4.11. recursive

A boolean to express if an "EBML Element" MAY be stored recursively. In this case the "EBML Element" MAY be stored at levels greater that defined in the "level" attribute if the "EBML Element" is a "Child Element" of a "Parent Element" with the same "Element ID". "EBML Elements" that are not "Master Elements" MUST NOT set "recursive" to true.

If the "path" contains a "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.

11.1.4.12. 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".

11.1.4.13. 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 .

11.1.5. Element

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

11.1.6. Attributes

11.1.6.1. lang

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

The "lang" attribute is OPTIONAL.

11.1.6.2. type

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

The "type" attribute is OPTIONAL.

11.1.7. EBML Schema Example

```
<?xml version="1.0" encoding="utf-8"?>
<EBMLSchema docType="files-in-ebml-demo" version="1">
 <!-- 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"
  maxOccurs="unbounded">
  <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>
```

11.1.8. 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.

11.1.9. Expression of range

The "range" attribute MUST only be used with EBML Elements that are either "signed integer", "unsigned integer", "float", or "date". 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:

- "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". "x" MUST be less than "y".
- ">x" where "x" is an integer or float, meaning that the value MUST be greater than "x".
- ">=x" where "x" is an integer or float, meaning that the value MUST be greater than or equal to "x".
- "<x" where "x" is an integer or float, meaning that the value MUST be less than "x".
- "<=x" where "x" is an integer or float, meaning that the value MUST be less than or equal to "x".
- "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 Section 11.1.10.

11.1.10. 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.

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. 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.

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

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: 1

default: 1

type: Unsigned Integer

description: The version of "EBML Writer" used to create the "EBML Document".

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 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 length in octets of the expression of all "Element Data Sizes" to be found within the "EBML Body". To be clear the "EBMLMaxSizeLength Element" 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". "EBML Elements" that have an "Element Data Size" expression which is larger in octets than what is expressed by "EBMLMaxSizeLength ELEMENT" SHALL be considered invalid.

11.2.6. DocType Element

name: "DocType"

path: "1*1(\EBML\DocType)"

id "0x4282"

minOccurs: 1

maxOccurs: 1

size: >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

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

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".

11.3. Global elements (used everywhere in the format)

name: CRC-32

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

id: "0xBF"

minOccurs: 0

maxOccurs: 1

size: 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" 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.

11.3.1. Void Element

name: Void

path: "*((*\)\Void)"

id: "0xEC"

minOccurs: 0

maxOccurs: unbounded

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.

12. References

12.1. 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, <http://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, <http://www.rfc-editor.org/info/rfc2119>.

[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>.

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

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://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, <http://www.rfc-editor.org/info/rfc5646>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <http://www.rfc-editor.org/info/rfc7405>.

[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>.

12.2. 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>.

Authors' Addresses

Steve Lhomme

Dave Rice

Moritz Bunkus