**COMMITTEE DRAFT****© ISO/IEC 2019 – All rights reserved****Text of ISO/IEC CD 23002-7****63****Part 7: Supplemental enhancement information messages for coded video bitstreams****Information technology — MPEG video technologies****Élément introductif — Élément central — Partie 2: Titre de la partie****Information technology — MPEG video technologies — Part 7: Supplemental enhancement information messages for coded video bitstreams****E****2019-07-12****(30) Committee****ISO/IEC****ISO/IEC J****202x****International Standard****202x****18699****ISO/IEC 23002‑****ISO/IEC 23002‑7****ISO/IEC 23002-7:202x****JISC****Coding of audio, picture, multimedia and hypermedia information****Information technology****11****29****1** **2****見出し 2****見出し 1****0****2****STD Version 2.1c2****40** **4** **ISO/IEC JTC 1/SC 29 /WG 11 N 18699**

Date: **2019-07-12**

**Text of ISO/IEC CD 23002-7**

ISO/IEC JTC 1/SC 29/WG 11

Secretariat:

**Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding, AMENDMENT 1: Additional supplemental enhancement information**

*Élément introductif — Élément central — Partie 2: Titre de la partie*

|  |
| --- |
| **Warning**  This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.  Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation |

**Copyright notice**

This ISO document is a working draft or committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

[Indicate the full address, telephone number, fax number, telex number, and electronic mail address, as appropriate, of the Copyright Manger of the ISO member body responsible for the secretariat of the TC or SC within the framework of which the working document has been prepared.]

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. In some areas of information technology which fall within the purview of ISO and IEC, the necessary standards are prepared on a collaborative basis with ITU-T.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 23002-7:202x was prepared by Joint Technical Committee ISO/IEC JTC 1, *Coding of audio, picture, multimedia and hypermedia information*, Subcommittee SC 29, *Coding of moving pictures and audio*, in a joint collaboration with ITU-T Study Group 16. Technically aligned twin text is developed as Recommendation ITU-T "H.SEI" [Final numbering to be determined by the ITU TSB.

Information technology — MPEG video technologies — Part 7: Supplemental enhancement information messages for coded video bitstreams

**CONTENTS**

*Page*

1 Scope 1

2 Normative references 1

2.1 Identical Recommendations | International Standards 1

2.2 Paired Recommendations | International Standards equivalent in technical content 1

2.3 Additional references 1

3 Definitions 1

4 Abbreviations 4

5 Conventions 4

5.1 General 4

5.2 Arithmetic operators 4

5.3 Logical operators 4

5.4 Relational operators 5

5.5 Bit-wise operators 5

5.6 Assignment operators 5

5.7 Range notation 5

5.8 Mathematical functions 6

5.9 Order of operation precedence 7

5.10 Variables, syntax elements and tables 8

5.11 Text description of logical operations 9

5.12 Processes 10

6 Syntax and semantics 11

6.1 Method of specifying syntax in tabular form 11

6.2 Specification of syntax functions and descriptors 12

7 SEI message payload specifications 13

7.1 General 13

7.2 SEI payload syntax 13

7.2.1 General 13

7.2.2 Decoded picture hash SEI message syntax 13

7.2.3 Dependent random access point indication SEI message syntax 13

7.2.4 Reserved SEI message syntax 14

7.3 SEI payload semantics 14

7.3.1 General 14

7.3.2 Decoded picture hash SEI message semantics 14

7.3.3 Dependent random access point indication SEI message semantics 16

7.3.4 Reserved SEI message semantics 16

INTERNATIONAL STANDARD

Draft ISO/IEC 23002-7

Draft Rec. ITU-T H.SEI

ITU-T RECOMMENDATION

# Scope

This Recommendation | International Standard specifies the syntax and semantics of supplemental enhancement information (SEI) messages. The SEI messages defined in this standard may be conveyed within coded video bitstreams in a manner specified in a video coding specification or may be conveyed by other means as determined by the specifications for systems that make use of such coded video bitstreams. This document is particularly intended for use with coded video bitstreams as specified by Rec. ITU-T H.VVC | ISO/IEC 23090-3, although it is drafted in a manner intended to be sufficiently generic that it may also be used with other types of coded video bitstreams.

This document is written in a manner such that it is intended to be referenced by other technical specifications. Such other technical specifications are to be written in a manner to specify certain necessary elements to enable the use of these SEI messages.

SEI messages can assist in processes related to decoding, display or other purposes. However, unless otherwise specified in a referencing specification, the interpretation and use of the SEI messages specified in this document is not a required functionality of a video decoder or receiving video system. Although semantics are specified for these SEI messages, decoders and receiving video systems may simply ignore the content of these SEI messages or may use them in some manner not specified in this document.

# Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

## Identical Recommendations | International Standards

– None

## Paired Recommendations | International Standards equivalent in technical content

– None

## Additional references

– IETF RFC 1321 (in force), *The MD5 Message-Digest Algorithm*.

# Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

* 1. **byte-aligned**: A position in a *coded video bitstream* or *syntax structure* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *coded video bitstream* or *syntax structure*, and a bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *coded video bitstream* or *syntax structure* is byte-aligned.
  2. **byte stream**: An encapsulation of a *NAL unit stream* containing *start code prefixes* and *NAL units*.
  3. **chroma**: An adjective, represented by the symbols Cb and Cr, specifying that a sample array or single sample is representing one of the two colour difference signals related to the primary colours.

NOTE – The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.

* 1. **coded picture**: A *coded representation* of a *picture* containing all *CTUs* of the *picture*.
  2. **coded video bitstream**: A sequence of bits that forms the representation of *coded pictures* and associated data forming one or more coded video sequences *(CVSs)*.
  3. **coded video sequence (CVS)**: A sequence of *access units* that consists, in *decoding order*, of an *IRAP access unit*, followed by zero or more *access* *units* that are not *IRAP access units*, including all subsequent *access units* up to but not including any subsequent *access unit* that is an *IRAP access unit*.
  4. **component**: An array or single sample from one of the three arrays (*luma* and two *chroma*) that compose a *picture* in 4:2:0, 4:2:2, or 4:4:4 colour format or the array or a single sample of the array that compose a *picture* in monochrome format.
  5. **decoded picture**: A *decoded picture* is derived by decoding a *coded picture*.
  6. **decoder**: An embodiment of a *decoding process*.
  7. **decoding order**: The order in which *syntax elements* are conveyed in the *coded video bitstream* and are processed by a *decoding process*.
  8. **decoding process**: The process that reads a *coded video bitstream* and derives *decoded* *pictures* from it.
  9. **emulation prevention byte**: A *byte* equal to 0x03 that is present within a *NAL unit* when the *syntax elements* of the *coded video bitstream* form certain patterns of *byte* values in a manner that ensures that no sequence of consecutive *byte-aligned* *bytes* in the *NAL unit* can contain a *start code prefix*.
  10. **encoder**: An embodiment of an *encoding process*.
  11. **encoding process**: A process that produces a *coded video bitstream*.
  12. **flag**: A variable or single-bit *syntax element* that can take one of the two possible values: 0 and 1.
  13. **informative**: A term used to refer to content provided in this Specification that does not establish any mandatory requirements for conformance to this Specification and thus is not considered an integral part of this Specification.
  14. **intra random access point (IRAP) access unit**: An *access unit* in which the *coded picture* is an *IRAP picture*.
  15. **intra random access point (IRAP) picture**: A *coded picture* that can be decoded without first decoding any other picture in the *coded video bitstream*.
  16. **luma**: An adjective, represented by the symbol or subscript Y or L, specifying that a sample array or single sample is representing the monochrome signal related to the primary colours.

NOTE – The term luma is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

* 1. **may**: A term that is used to refer to behaviour that is allowed, but not necessarily required*.*

NOTE – In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

* 1. **must**: A term that is used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Specification (used exclusively in an *informative* context).
  2. **network abstraction layer (NAL) unit**: A *syntax structure* containing an indication of the type of data to follow and *bytes* containing that data in the form of an *RBSP* interspersed as necessary with *emulation prevention bytes*.
  3. **network abstraction layer (NAL) unit stream**: A sequence of *NAL units*.
  4. **non-VCL NAL unit**: A *NAL unit* that is not a *VCL NAL unit*.
  5. **note**: A term that is used to prefix *informative* remarks (used exclusively in an *informative* context).
  6. **output order**: The order in which the *decoded* *pictures* are output from the *decoded picture buffer* (for the *decoded pictures* that are to be output from the *decoded picture buffer*).
  7. **parameter**: A *syntax element* of a *sequence parameter set (SPS)* or *picture parameter set (PPS)*, or the second word of the defined term *quantization parameter*.
  8. **partitioning**: The division of a set into subsets such that each element of the set is in exactly one of the subsets.
  9. **picture**: An array of *luma* samples in monochrome format or an array of *luma* samples and two corresponding arrays of *chroma* samples in 4:2:0, 4:2:2, and 4:4:4 colour format.

NOTE – A picture may be either a frame or a field. However, in one CVS, either all pictures are frames or all pictures are fields.

* 1. **picture parameter set (PPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *coded pictures* as determined by a *syntax element* found in each *slice header.*
  2. **picture order count (POC)**: A variable that is associated with each *picture*, uniquely identifies the associated *picture* among all *pictures* in the *CVS*, and, when the associated *picture* is to be output from the *decoded picture buffer*, indicates the position of the associated *picture* in *output order* relative to the *output order* positions of the other *pictures* in the same *CVS* that are to be output from the *decoded picture buffer*.
  3. **profile**: A specified subset of the syntax of this Specification.
  4. **random access**: The act of starting the decoding process for a *coded video bitstream* at a point other than the beginning of the stream.
  5. **raw byte sequence payload (RBSP)**: A *syntax structure* containing an integer number of *bytes* that is encapsulated in a *NAL unit* and that is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and zero or more subsequent bits equal to 0.
  6. **raw byte sequence payload (RBSP) stop bit**: A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*, for which the location of the end within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP.*
  7. **reference picture**: A *picture* that is a *short-term reference picture*.

NOTE – A reference picture contains samples that may be used for inter prediction in the decoding process of subsequent pictures in decoding order.

* 1. **reference picture list**: A list of *reference pictures* that is used for *inter prediction* of a *P* or *B slice.*

NOTE – For the decoding process of a P slice, there is one reference picture list – reference picture list 0. For the decoding process of a B slice, there are two reference picture lists – reference picture list 0 and reference picture list 1.

* 1. **reserved**: A term that may be used to specify that some values of a particular *syntax element* are for future use by ITU-T | ISO/IEC and shall not be used in *syntax structures* conforming to this version of this Specification, but may be used in *syntax structures* conforming to future extensions of this Specification by ITU‑T | ISO/IEC.
  2. **sequence parameter set (SPS)**: A *syntax structure* containing *syntax elements* that apply to zero or more entire *CVSs* as determined by the content of a *syntax element* found in the *PPS* referred to by a *syntax element* found in each *slice header.*
  3. **shall**: A term used to express mandatory requirements for conformance to this Specification.

NOTE – When used to express a mandatory constraint on the values of syntax elements or on the results obtained by operation of the specified decoding process, it is the responsibility of the encoder to ensure that the constraint is fulfilled. When used in reference to operations performed by the decoding process, any decoding process that produces identical cropped decoded pictures to those output from the decoding process described in this Specification conforms to the decoding process requirements of this Specification.

* 1. **short-term reference picture**: A *picture* that is marked as "used for short-term reference".
  2. **should**: A term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this Specification.
  3. **source**: A term used to describe the video material or some of its attributes before encoding.
  4. **start code prefix**: A unique sequence of three *bytes* equal to 0x000001 embedded in the *byte stream* as a prefix to each *NAL unit*.

NOTE – The location of a start code prefix can be used by a decoder to identify the beginning of a new NAL unit and the end of a previous NAL unit. Emulation of start code prefixes is prevented within NAL units by the inclusion of emulation prevention bytes.

* 1. **string of data bits (SODB)**: A sequence of some number of bits representing *syntax elements* present within a *raw byte sequence payload* prior to the *raw byte sequence payload stop bit*, where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.
  2. **syntax element**: An element of data represented in a *syntax structure*.
  3. **syntax structure**: Zero or more *syntax elements* that are present together in a specified order*.*
  4. **temporal sublayer identifier**: A number greater than or equal to 0 such that pictures of all temporal sublayers have a specified temporal output order relative to each other and pictures with a lower temporal identifier can be decoded without reference to pictures with a higher temporal identifier.
  5. **trailing picture**: A *coded picture* that follows an *IRAP* picture in both decoding order and output order.
  6. **unspecified**: A term that may be used to specify some values of a particular *syntax element* to indicate that the values have no specified meaning in this Specification and will not have a specified meaning in the future as an integral part of future versions of this Specification.
  7. **video coding layer (VCL) NAL unit**: A collective term for *coded slice NAL units* and the subset of *NAL units* that have *reserved* values of nal\_unit\_type that are classified as VCL NAL units in this Specification.

# Abbreviations

CRC Cyclic Redundancy Check

CVS Coded Video Sequence

DRAP Dependent Random Access Point

IDR Instantaneous Decoding Refresh

IRAP Intra Random Access Point

NAL Network Abstraction Layer

POC Picture Order Count

PPS Picture Parameter Set

RPS Reference Picture Set

SEI Supplemental Enhancement Information

SPS Sequence Parameter Set

VCL Video Coding Layer

VPS Video Parameter Set

# Conventions

## General

NOTE – The mathematical operators used in this Specification are similar to those used in the C programming language. However, the results of integer division and arithmetic shift operations are defined more precisely, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0, e.g., "the first" is equivalent to the 0-th, "the second" is equivalent to the 1-th, etc.

## Arithmetic operators

The following arithmetic operators are defined as follows:

|  |  |
| --- | --- |
| + | Addition |
| − | Subtraction (as a two-argument operator) or negation (as a unary prefix operator) |
| \* | Multiplication, including matrix multiplication |
| xy | Exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation. |
| / | Integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1. |
| ÷ | Used to denote division in mathematical equations where no truncation or rounding is intended. |
|  | Used to denote division in mathematical equations where no truncation or rounding is intended. |
|  | The summation of f( i ) with i taking all integer values from x up to and including y. |
| x % y | Modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0. |

## Logical operators

The following logical operators are defined as follows:

x && y Boolean logical "and" of x and y

x | | y Boolean logical "or" of x and y

! Boolean logical "not"

x ? y : z If x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the value of z.

## Relational operators

The following relational operators are defined as follows:

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

= = Equal to

!= Not equal to

When a relational operator is applied to a syntax element or variable that has been assigned the value "na" (not applicable), the value "na" is treated as a distinct value for the syntax element or variable. The value "na" is considered not to be equal to any other value.

## Bit-wise operators

The following bit-wise operators are defined as follows:

& Bit-wise "and". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

| Bit-wise "or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

^ Bit-wise "exclusive or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

x >> y Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.

x << y Arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

## Assignment operators

The following arithmetic operators are defined as follows:

= Assignment operator

+ + Increment, i.e., *x*+ + is equivalent to *x* = *x* + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation.

− − Decrement, i.e., *x*− − is equivalent to *x* = *x* − 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation.

+= Increment by amount specified, i.e., x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3).

−= Decrement by amount specified, i.e., x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3).

## Range notation

The following notation is used to specify a range of values:

x = y..z x takes on integer values starting from y to z, inclusive, with x, y, and z being integer numbers and z being greater than y.

## Mathematical functions

The following mathematical functions are defined:

Abs( x ) = (1)

Asin( x ) the trigonometric inverse sine function, operating on an argument x that is  
in the range of −1.0 to 1.0, inclusive, with an output value in the range of   
−π÷2 to π÷2, inclusive, in units of radians (2)

Atan( x ) the trigonometric inverse tangent function, operating on an argument x, with  
an output value in the range of −π÷2 to π÷2, inclusive, in units of radians (3)

Atan2( y, x ) = (4)

Ceil( x ) the smallest integer greater than or equal to x. (5)

Clip1Y( x ) = Clip3( 0, ( 1 << BitDepthY ) − 1, x ) (6)

Clip1C( x ) = Clip3( 0, ( 1 << BitDepthC ) − 1, x ) (7)

Clip3( x, y, z ) = (8)

Cos( x ) the trigonometric cosine function operating on an argument x in units of radians. (9)

Floor( x ) the largest integer less than or equal to x. (10)

GetCurrMsb( a, b, c, d ) = (11)

Ln( x ) the natural logarithm of x (the base-e logarithm, where e is the natural logarithm base constant 2.718 281 828...). (12)

Log2( x ) the base-2 logarithm of x. (13)

Log10( x ) the base-10 logarithm of x. (14)

Min( x, y ) = (15)

Max( x, y ) = (16)

Round( x ) = Sign( x ) \* Floor( Abs( x ) + 0.5 ) (17)

Sign( x ) = (18)

Sin( x ) the trigonometric sine function operating on an argument x in units of radians (19)

Sqrt( x ) the square root of x (20)

Swap( x, y ) equivalent to setting a temporary variable t equal to x, then x = y and y = t (21)

Tan( x ) the trigonometric tangent function operating on an argument x in units of radians (22)

## Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

– Operations of a higher precedence are evaluated before any operation of a lower precedence.

– Operations of the same precedence are evaluated sequentially from left to right.

Table 1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE – For those operators that are also used in the C programming language, the order of precedence used in this Specification is the same as used in the C programming language.

Table 1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)

|  |
| --- |
| **operations (with operands x, y, and z)** |
| "x++", "x− −" |
| "!x", "−x" (as a unary prefix operator) |
| xy |
| "x \* y", "x / y", "x ÷ y", "", "x % y" |
| "x + y", "x − y" (as a two-argument operator), "" |
| "x  <<  y", "x  >>  y" |
| "x < y", "x  <=  y", "x > y", "x  >=  y" |
| "x  = =  y", "x  !=  y" |
| "x & y" |
| "x | y" |
| "x  &&  y" |
| "x  | |  y" |
| "x ? y : z" |
| "x..y" |
| "x = y", "x  +=  y", "x  −=  y" |

## Variables, syntax elements and tables

Syntax elements in the syntax tables are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e., not bold) type.

In some cases the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the clause in which they are derived.

In some cases, "mnemonic" names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes "mnemonic" names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper case letter and may contain more upper case letters.

NOTE – The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions that specify properties of the current position in the SEI message payload data are referred to as syntax functions. These functions are specified in clause 6.2 and assume the existence of a pointer with an indication of the position of the next bit to be read by the decoding process from the payload data. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in clause 5.8) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The indexing order is reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as s[ x ][ y ] or as syx. A single column of a matrix may be referred to as a list and denoted by omission of the row index. Thus, the column of a matrix s at horizontal position x may be referred to as the list s[ x ].

A specification of values of the entries in rows and columns of an array may be denoted by { {...} {...} }, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix s equal to { { 1 6 } { 4 9 }} specifies that s[ 0 ][ 0 ] is set equal to 1, s[ 1 ][ 0 ] is set equal to 6, s[ 0 ][ 1 ] is set equal to 4, and s[ 1 ][ 1 ] is set equal to 9.

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value different from zero.

## Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0 )  
 statement 0  
else if( condition 1 )  
 statement 1  
...  
else /\* informative remark on remaining condition \*/  
 statement n

may be described in the following manner:

... as follows / ... the following applies:

– If condition 0, statement 0

– Otherwise, if condition 1, statement 1

– ...

– Otherwise (informative remark on remaining condition), statement n

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0a && condition 0b )  
 statement 0  
else if( condition 1a | | condition 1b )  
 statement 1  
...  
else  
 statement n

may be described in the following manner:

... as follows / ... the following applies:

– If all of the following conditions are true, statement 0:

– condition 0a

– condition 0b

– Otherwise, if one or more of the following conditions are true, statement 1:

– condition 1a

– condition 1b

– ...

– Otherwise, statement n

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0 )  
 statement 0  
if( condition 1 )  
 statement 1

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

## Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lower case variable explicitly specified as input. Each process specification has explicitly specified an output. The output is a variable that can either be an upper case variable or a lower case variable.

When invoking a process, the assignment of variables is specified as follows:

– If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification.

– Otherwise (the variables at the invoking and the process specification have the same name), assignment is implied.

# Syntax and semantics

## Method of specifying syntax in tabular form

Technical specifications that reference this document shall specify some means to carry the payload syntax of each specified SEI message, to identify which SEI message is conveyed, and to identify the length in bits of the SEI message payload data. The length of the SEI message payload data is referred to herein by the variable name payloadSize.

For example, each SEI message may be carried as a string of data bits (SODB) that is prefixed with an SEI message payload type indication derived as a payloadType variable within a NAL unit that may contain emulation prevention bytes as specified in Rec. ITU-T H.VVC | ISO/IEC 23090-3.

The syntax tables in this document specify a superset of the syntax of all allowed SEI message payloads. Additional constraints on the syntax may be specified, either directly or indirectly, in other clauses.

The following table lists examples of the syntax specification format. When **syntax\_element** appears, it specifies that a syntax element is parsed from the SEI message payload data and the data pointer is advanced to the next position beyond the syntax element in the syntax parsing process.

|  |  |
| --- | --- |
|  | Descriptor |
| /\* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples \*/ |  |
| **syntax\_element** | ue(v) |
| conditioning statement |  |
|  |  |
| /\* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. \*/ |  |
| { |  |
| statement |  |
| statement |  |
| ... |  |
| } |  |
|  |  |
| /\* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true \*/ |  |
| while( condition ) |  |
| statement |  |
|  |  |
| /\* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true \*/ |  |
| do |  |
| statement |  |
| while( condition ) |  |
|  |  |
| /\* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed \*/ |  |
| if( condition ) |  |
| primary statement |  |
| else |  |
| alternative statement |  |
|  |  |
| /\* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. \*/ |  |
| for( initial statement; condition; subsequent statement ) |  |
| primary statement |  |

## Specification of syntax functions and descriptors

The functions presented here are used in the syntactical description. These functions are expressed in terms of the value of an SEI message payload data pointer that indicates the position of the next bit to be read by the decoding process from the payload data.

byte\_aligned( ) is specified as follows:

– If the current position in the payload data is on a byte boundary, i.e., the next bit in the payload data is the first bit in a byte, the return value of byte\_aligned( ) is equal to TRUE.

– Otherwise, the return value of byte\_aligned( ) is equal to FALSE.

more\_data\_in\_payload( ) is specified as follows:

– If byte\_aligned( ) is equal to TRUE and the current position in the sei\_payload( ) syntax structure is 8 \* payloadSize bits from the beginning of the sei\_payload( ) syntax structure, the return value of more\_data\_in\_payload( ) is equal to FALSE.

– Otherwise, the return value of more\_data\_in\_payload( ) is equal to TRUE.

more\_rbsp\_data( ) is specified as follows:

– If there is no more data in the raw byte sequence payload (RBSP), the return value of more\_rbsp\_data( ) is equal to FALSE.

– Otherwise, the RBSP data are searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp\_stop\_one\_bit) of the rbsp\_trailing\_bits( ) syntax structure, the following applies:

– If there is more data in an RBSP before the rbsp\_trailing\_bits( ) syntax structure, the return value of more\_rbsp\_data( ) is equal to TRUE.

– Otherwise, the return value of more\_rbsp\_data( ) is equal to FALSE.

The method for enabling determination of whether there is more data in the RBSP is specified by the application.

more\_rbsp\_trailing\_data( ) is specified as follows:

– If there is more data in an RBSP, the return value of more\_rbsp\_trailing\_data( ) is equal to TRUE.

– Otherwise, the return value of more\_rbsp\_trailing\_data( ) is equal to FALSE.

read\_bits( n ) reads the next n bits from the payload data and advances the data pointer by n bit positions. When n is equal to 0, read\_bits( n ) is specified to return a value equal to 0 and to not advance the data pointer.

The following descriptors specify the parsing process of each syntax element:

– b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read\_bits( 8 ).

– f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ).

– i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a two's complement integer representation with most significant bit written first.

– st(v): null-terminated string encoded as universal coded character set (UCS) transmission format-8 (UTF-8) characters as specified in ISO/IEC 10646. The parsing process is specified as follows: st(v) begins at a byte-aligned position in the payload data and reads and returns a series of bytes from the payload data, beginning at the current position and continuing up to but not including the next byte-aligned byte that is equal to 0x00, and advances the data pointer by ( stringLength + 1 ) \* 8 bit positions, where stringLength is equal to the number of bytes returned.

NOTE – The st(v) syntax descriptor is only used in this Specification when the current position in the payload data is a byte-aligned position.

– u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a binary representation of an unsigned integer with most significant bit written first.

[Ed. (GJS): ue(v) and se(v) are not defined, since they are not used in the currently specified SEI messages. If additional SEI messages are defined that need them, their specification would need to be added.]

# SEI message payload specifications

## General

This clause specifies the syntax and semantics for SEI message payloads.

[Ed. (GJS): It might be nice to reorganize the subclauses such that for each SEI message there is one subclause, and the syntax and semantics are two subordinate subclauses of that, instead of grouping together the syntax of all SEI messages. This could be more friendly to readers by avoiding the need to use split screen viewing or to jump back and forth in the document when studying a particular SEI message.]

## SEI payload syntax

### General

The syntax of the container of the SEI messages as well as the method of identifying which SEI message is outside the scope of this document. The syntax specified in this subclause is for the payload data of each associated SEI message. For SEI messages for which the specified syntax structure is empty, such as the dependent random access point SEI message, there is no payload data and the mere indication that the SEI message is present (e.g., as indicated by a payload type indicator with no payload data) is sufficient to convey the associated information (e.g., by indicating that a set of specified constraints are fulfilled).

### Decoded picture hash SEI message syntax

|  |  |
| --- | --- |
| decoded\_picture\_hash( payloadSize ) { | Descriptor |
| **hash\_type** | u(8) |
| for( cIdx = 0; cIdx < ( chroma\_format\_idc = = 0 ? 1 : 3 ); cIdx++ ) |  |
| if( hash\_type = = 0 ) |  |
| for( i = 0; i < 16; i++) |  |
| **picture\_md5**[ cIdx ][ i ] | b(8) |
| else if( hash\_type = = 1 ) |  |
| **picture\_crc**[ cIdx ] | u(16) |
| else if( hash\_type = = 2 ) |  |
| **picture\_checksum**[ cIdx ] | u(32) |
| } |  |

### Dependent random access point indication SEI message syntax

|  |  |
| --- | --- |
| dependent\_rap\_indication( payloadSize ) { | Descriptor |
| } |  |

### Reserved SEI message syntax

|  |  |
| --- | --- |
| reserved\_sei\_message( payloadSize ) { | Descriptor |
| for( i = 0; i < payloadSize; i++ ) |  |
| **reserved\_sei\_message\_payload\_byte** | u(8) |
| } |  |

## SEI payload semantics

### General

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message.

NOTE – Persistence information for SEI messages is informatively summarized in Table 2 –

Table 2 – Persistence scope of SEI messages (informative)

|  |  |
| --- | --- |
| SEI message | Persistence scope |
| Decoded picture hash | The access unit containing the SEI message |
| DRAP indication | The access unit containing the SEI message |

### Decoded picture hash SEI message semantics

This message provides a hash for each colour component of the current decoded picture.

Use of this SEI message requires the definition of the following parameters:

– A picture width and picture height in units of luma samples, denoted herein by pic\_width\_in\_luma\_samples and pic\_height\_in\_luma\_samples.

– A chroma format indicator, denoted herein by chroma\_format\_idc, such that the value 0 indicates that the picture has only a luma component and other values indicate that the picture has three colour components that consist of a luma component and two associated chroma components, such that the width and height of each chroma component are pic\_width\_in\_luma\_samples / SubWidthC and pic\_height\_in\_luma\_samples / SubHeightC, respectively, where SubWidthC and SubHeightC are determined from chroma\_format\_idc as specified by Table 3.

– A bit depth for the samples of the luma component, denoted herein as BitDepthY, and when chroma\_format\_idc is not equal to 0, a bit depth for the samples of the two associated chroma components, denoted herein as BitDepthC.

– For each colour component cIdx, an array component[ cIdx ][ i ] in raster scan order of decoded sample values in two's complement representation

**Table 3 – SubWidthC and SubHeightC values derived from  
chroma\_format\_idc**

|  |  |  |  |
| --- | --- | --- | --- |
| **chroma\_format\_idc** | **Chroma format** | **SubWidthC** | **SubHeightC** |
| 0 | Monochrome | 1 | 1 |
| 1 | 4:2:0 | 2 | 2 |
| 2 | 4:2:2 | 2 | 1 |
| 3 | 4:4:4 | 1 | 1 |

Prior to computing the hash, the decoded picture data are arranged into one or three strings of bytes called pictureData[ cIdx ] of lengths dataLen[ cIdx ] as follows:

for( cIdx = 0; cIdx < ( chroma\_format\_idc = = 0 ) ? 1 : 3; cIdx++ ) {  
 if( cIdx = = 0 ) {  
 compWidth[ cIdx ] = pic\_width\_in\_luma\_samples  
 compHeight[ cIdx ] = pic\_height\_in\_luma\_samples  
 compDepth[ cIdx ] = BitDepthY  
 } else {  
 compWidth[ cIdx ] = pic\_width\_in\_luma\_samples / SubWidthC  
 compHeight[ cIdx ] = pic\_height\_in\_luma\_samples / SubHeightC  
 compDepth[ cIdx ] = BitDepthC (23)  
 }  
 iLen = 0  
 for( i = 0; i < compWidth[ cIdx ] \* compHeight[ cIdx ]; i++ ) {  
 pictureData[ cIdx ][ iLen++ ] = component[ cIdx ][ i ] & 0xFF  
 if( compDepth[ cIdx ] > 8 )  
 pictureData[ cIdx ][ iLen++ ] = component[ cIdx ][ i ] >> 8  
 }  
 dataLen[ cIdx ] = iLen  
}

where component[ cIdx ][ i ] is an array in raster scan of decoded sample values in two's complement representation.

**hash\_type** indicates the method used to calculate the checksum as specified in Table 7‑4. Values of hash\_type that are not listed in in Table 7‑4 are reserved for future use by ITU-T | ISO/IEC and shall not be present in payload data conforming to this version of this Specification. Decoders shall ignore decoded picture hash SEI messages that contain reserved values of hash\_type.

Table 7‑4 – Interpretation of hash\_type

|  |  |
| --- | --- |
| hash\_type | Method |
| 0 | MD5 (IETF RFC 1321) |
| 1 | CRC |
| 2 | Checksum |

**picture\_md5**[ cIdx ][ i ] is the 16-byte MD5 hash of the cIdx-th colour component of the decoded picture. The value of picture\_md5[ cIdx ][ i ] shall be equal to the value of digestVal[ cIdx ] obtained as follows, using the MD5 functions defined in IETF RFC 1321:

MD5Init( context )  
MD5Update( context, pictureData[ cIdx ], dataLen[ cIdx ] ) (24)  
MD5Final( digestVal[ cIdx ], context )

**picture\_crc**[ cIdx ] is the cyclic redundancy check (CRC) of the colour component cIdx of the decoded picture. The value of picture\_crc[ cIdx ] shall be equal to the value of crcVal[ cIdx ] obtained as follows:

crc = 0xFFFF  
pictureData[ cIdx ][  dataLen[ cIdx ] ] = 0  
pictureData[ cIdx ][  dataLen[ cIdx ] + 1 ] = 0  
for( bitIdx = 0; bitIdx < ( dataLen[ cIdx ]  + 2 ) \* 8; bitIdx++ ) { (25)  
 dataByte = pictureData[ cIdx ][ bitIdx >> 3 ]  
 crcMsb = ( crc >> 15 ) & 1  
 bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1  
 crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb \* 0x1021 )  
}  
crcVal[ cIdx ] = crc

NOTE – The same CRC specification is found in Rec. ITU-T H.271.

**picture\_checksum**[ cIdx ] is the checksum of the colour component cIdx of the decoded picture. The value of picture\_checksum[ cIdx ] shall be equal to the value of checksumVal[ cIdx ] obtained as follows:

sum = 0  
for( y = 0; y < compHeight[ cIdx ]; y++ )  
 for( x = 0; x < compWidth[ cIdx ]; x++ ) {  
 xorMask = ( x & 0xFF ) ^ ( y & 0xFF ) ^ ( x >> 8 ) ^ ( y >> 8 )  
 sum = ( sum + ( ( component[ cIdx ][ y \* compWidth[ cIdx ] + x ] & 0xFF ) ^ (26)  
 xorMask ) ) & 0xFFFFFFFF  
 if( compDepth[ cIdx ] > 8 )  
 sum = ( sum + ( ( component[ cIdx ][ y \* compWidth[ cIdx ] + x ] >> 8 ) ^  
 xorMask ) ) & 0xFFFFFFFF  
 }  
checksumVal[ cIdx ] = sum

### Dependent random access point indication SEI message semantics

The picture associated with a dependent random access point (DRAP) indication SEI message is referred to as a DRAP picture.

The presence of the DRAP indication SEI message indicates that the constraints on picture order and picture referencing specified in this subclause apply. These constraints can enable a decoder to properly decode the DRAP picture and the pictures that follow it in both decoding order and output order without needing to decode any other pictures except the associated IRAP picture.

The constraints indicated by the presence of the DRAP indication SEI message are as follows:

– The DRAP picture shall be a trailing picture.

– The DRAP picture shall have a temporal sublayer identifier equal to 0.

– The DRAP picture shall not include any pictures in the active entries of its reference picture lists except the preceding IRAP picture in decoding order.

– Any picture that follows the DRAP picture in both decoding order and output order shall not include, in the active entries of its reference picture lists, any picture that precedes the DRAP picture in decoding order or output order, with the exception of the preceding IRAP picture in decoding order.

### Reserved SEI message semantics

The reserved SEI message consists of data reserved for future backward-compatible use by ITU-T | ISO/IEC. Unless otherwise specified by a referencing specification, coded video bitstreams shall not contain reserved SEI messages and systems that make use of such coded video bitstreams shall not otherwise send reserved SEI messages until and unless the use of such messages has been specified by ITU-T | ISO/IEC. Decoders shall ignore reserved SEI messages.

**Bibliography**

1. Recommendation ITU-T H.VVC (in force), *Versatile video coding.*

ISO/IEC 23090-3 (in force), *Information technology* – *Coded representation of immersive media* – *Part 3: Versatile video coding.*

1. Recommendation ITU-T H.271 (in force), *Video back-channel messages for conveyance of status information and requests from a video receiver to a video sender*.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_