
**Information technology — Coding of
audio-visual objects —**

**Part 31:
Video coding for browsers**

*Technologies de l'information — Codage des objets audiovisuels —
Partie 31: Codage video pour navigateurs*



Reference number
ISO/IEC 14496-31:2017(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2017



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2017, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	v
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Abbreviated terms	5
5 Conventions	6
5.1 General	6
5.2 Arithmetic operators	6
5.3 Logical operators	6
5.4 Relational operators	6
5.5 Bitwise operators	7
5.6 Assignment operators	7
5.7 Range notation	7
5.8 Mathematical functions	7
5.9 Order of operation precedence	8
5.10 Method of describing bitstream syntax	8
5.11 Functions	10
5.12 Descriptors	10
5.12.1 General	10
5.12.2 $f(n)$	10
5.12.3 $B(p)$	10
5.12.4 $u(n)$	10
5.12.5 F	11
5.12.6 $P(7)$	11
5.12.7 $P(8)$	11
5.12.8 T	11
6 Bitstream syntax	11
6.1 General	11
6.2 Frame syntax	11
6.3 Uncompressed header syntax	12
6.4 Compressed header syntax	13
6.4.1 General	13
6.4.2 Segment based adjustments syntax	14
6.4.3 Loop filter adjustments syntax	15
6.4.4 Quantizer indices syntax	15
6.4.5 Token probability update syntax	16
6.4.6 MV probability update syntax	16
6.5 Macroblock data syntax	17
6.5.1 Macroblock header syntax	17
6.5.2 Prediction residual data syntax	18
6.5.3 Prediction residual block syntax	19
7 Bitstream semantics	20
7.1 General	20
7.2 Frame semantics	20
7.3 Uncompressed header semantics	20
7.4 Compressed header semantics	22
7.4.1 General	22
7.4.2 Segment-based adjustments semantics	24
7.4.3 Loop filter adjustments semantics	25
7.4.4 Quantizer indices semantics	26
7.4.5 Token probability update semantics	27
7.4.6 MV probability update semantics	27

7.5	Macroblock data semantics.....	27
7.5.1	Macroblock header semantics.....	27
7.5.2	Prediction residual block semantics.....	29
8	Decoding process.....	30
8.1	General.....	30
8.2	Header update process.....	31
8.3	Coding context update process.....	31
8.4	Macroblock decoding process.....	32
8.4.1	General.....	32
8.4.2	Intra prediction process.....	33
8.4.3	Inter prediction process.....	40
8.4.4	Prediction residual decoding process.....	43
8.4.5	Motion vector prediction process.....	48
8.4.6	Motion vector reconstruction process.....	50
8.5	Loop filter process.....	52
8.5.1	General.....	52
8.5.2	Normal macroblock deblocking process.....	53
8.5.3	Filter level process.....	53
8.5.4	Simple macroblock deblocking process.....	54
8.5.5	Macroblock edge deblocking process.....	55
8.5.6	Sub-block edge deblocking process.....	55
8.5.7	Simple edge deblocking process.....	55
8.5.8	Threshold process.....	56
8.5.9	Common filtering process.....	56
8.5.10	Macroblock edge filtering process.....	57
8.5.11	Loop filter threshold process.....	58
8.6	Output process.....	58
8.7	Reference frame update process.....	59
8.7.1	General.....	59
8.7.2	Copy frame process.....	60
9	Parsing process.....	60
9.1	Parsing process for $f(n)$	60
9.2	Parsing process for Boolean decoder.....	60
9.2.1	General.....	60
9.2.2	Initialization process for Boolean decoder.....	60
9.2.3	Boolean decoder selection process.....	61
9.2.4	Boolean decoding process.....	61
9.2.5	Parsing process for <code>read_literal</code>	62
9.3	Parsing process for tree encoded syntax elements.....	62
9.3.1	General.....	62
9.3.2	Tree selection process.....	62
9.3.3	Probability selection process.....	65
9.3.4	Tree decoding process.....	71
10	Additional probability tables.....	72
10.1	Probability update tables.....	72
10.2	Default probability tables.....	76
Annex A (normative) Profiles and levels.....		81
Annex B (informative) Reference encoder description.....		82

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 14496 series can be found on the ISO website.

Information technology — Coding of audio-visual objects —

Part 31: Video coding for browsers

1 Scope

This document specifies the video coding for browsers (VCB) syntax format and decoding process.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

3.1

4:2:0 colour format

colour format with two chroma arrays that each have half the height and half the width of the luma array

3.2

AC transform coefficient

transform coefficient (3.55) for which the *frequency index* (3.23) in at least one of the two dimensions is non-zero

3.3

bitstream

sequence of bits that forms the representation of *coded frames* (3.11)

3.4

bit string

ordered string with a limited number of bits, in which the left-most bit is the most significant bit (MSB) and the right-most bit is the least significant bit (LSB)

3.5

block

MxN (M-column by N-row) array of *sample values* (3.22) or MxN array of *transform coefficients* (3.55)

3.6

Boolean decoder

arithmetic decoder that processes one Boolean element at a time, wherein each Boolean element indicates how to refine the segmentation of a number line and hence specifies the decoded bits

3.7

Boolean decoding

process of decoding a *syntax element* ([3.54](#)) using a *Boolean decoder* ([3.6](#))

3.8

byte

8-bit *bit string* ([3.4](#))

3.9

byte aligned

position that is an integer multiple of eight, such that the first bit in the *bitstream* ([3.3](#)) has position 0 and the position of other bits is stated relative to this position

3.10

chroma

sample array or single *sample* ([3.51](#)), denoted by the symbols Cb and Cr, representing one of the two colour difference signals related to the primary colours

3.11

coded frame

set of *syntax elements* ([3.54](#)) that represents a *frame* ([3.22](#)) in the *bitstream* ([3.3](#))

3.12

component

array or single *sample* ([3.51](#)) from one of the three arrays [*luma* ([3.32](#)) and two *chroma* ([3.10](#))] that compose a picture in 4:2:0 colour format ([3.1](#))

3.13

control partition

partition containing the frame header and all of the macroblock headers for a *coded frame* ([3.11](#))

3.14

DC transform coefficient

transform coefficient ([3.55](#)) for which the *frequency index* ([3.23](#)) in both of the two dimensions is zero

3.15

decoded frame

frame ([3.22](#)) reconstructed from a *coded frame* ([3.11](#)) by a *decoder* ([3.16](#))

3.16

decoder

embodiment of the *decoding process* ([3.17](#))

3.17

decoding process

process that derives *decoded frames* ([3.15](#)) from *syntax elements* ([3.54](#))

3.18

dequantization

process in which *transform coefficients* ([3.55](#)) are obtained by scaling the *quantized transform coefficients* ([3.45](#))

3.19

encoder

embodiment of an *encoding process* ([3.20](#))

3.20**encoding process**

process that generates a *bitstream* ([3.3](#))

Note 1 to entry: Other than establishing criteria for conformance of bitstreams produced by an encoder, the encoding process is not specified in this document (although an example encoding process is informatively described in [Annex B](#)).

3.21**flag**

binary variable

3.22**frame**

array of luma samples (Y) and two corresponding arrays of chroma samples (Cb and Cr) in *4:2:0 colour format* ([3.1](#))

3.23**frequency index**

one-dimensional or two-dimensional index associated with a *transform coefficient* ([3.55](#)) prior to an *inverse transform* ([3.29](#)) part of the *decoding process* ([3.17](#))

3.24**inter coding**

macroblock ([3.33](#)) or *frame* ([3.22](#)) coded using *inter prediction* ([3.25](#))

3.25**inter prediction**

process of deriving the *prediction* ([3.37](#)) for the current *frame* ([3.22](#)) using previously *decoded frames* ([3.15](#))

3.26**intra coding**

macroblock ([3.33](#)) or *frame* ([3.22](#)) coded using *intra prediction* ([3.28](#))

3.27**I frame**

frame ([3.22](#)) that may be decoded using only *intra prediction* ([3.28](#))

3.28**intra prediction**

process of deriving the *prediction value* ([3.42](#)) for the current *sample* ([3.51](#)) using previously decoded *sample values* ([3.52](#)) in the same *decoded frame* ([3.15](#))

3.29**inverse transform**

part of the *decoding process* ([3.17](#)) by which a set of *transform coefficients* ([3.55](#)) are converted into spatial-domain values

3.30**key frame**

frame ([3.22](#)) where the *decoding process* ([3.17](#)) is reset, such that a key frame and all following frames are always decodable without access to preceding frames and such that a frame is a key frame if and only if it is an *I frame* ([3.27](#))

3.31**level**

specified set of constraints on the permissible values for the *syntax elements* ([3.54](#)) and other characteristics of the *bitstreams* ([3.3](#)) produced by conforming *encoders* ([3.19](#)) established in combination with defined *profiles* ([3.43](#)) for ensuring interoperability of the bitstreams produced by conforming encoders with the *decoding process* ([3.17](#)) specified by conforming *decoders* ([3.16](#))

3.32

luma

sample array or *single sample* (3.51), represented by the symbol Y, that ordinarily represents the brightness signal related to the primary colours

3.33

macroblock

16x16 luma sample value *block* (3.5) and its two corresponding 8x8 chroma sample value blocks

3.34

motion vector

two-dimensional vector used for *inter prediction* (3.25) that provides an offset from the coordinates in the *decoded frame* (3.15) to the coordinates in a *reference frame* (3.49)

3.35

parse

process of extracting a *syntax element* (3.54) from a *bitstream* (3.3)

3.36

partitioning

process of dividing a set into subsets such that each element in the set belongs to only one subset

3.37

prediction

prediction process consisting of either *inter* (3.25) or *intra prediction* (3.28)

3.38

prediction frame

P frame

frame (3.22) decoded by referencing another frame

3.40

prediction process

process of estimating the decoded *sample value* (3.52) or data element using a predictor

3.41

prediction residual

difference between the reconstructed *samples* (3.51) and the corresponding *prediction values* (3.42)

3.42

prediction value

combination of the previously decoded *sample values* (3.52) or data elements, used in the *prediction process* (3.40) of the next sample value or data element

3.43

profile

specified subset of syntax, semantics and processes established in combination with defined *levels* (3.31) for ensuring interoperability of the *bitstreams* (3.3) produced by conforming *encoders* (3.19) with the *decoding process* (3.17) specified for conforming *decoders* (3.16)

3.44

quantization parameter

variable used for scaling the *quantized coefficients* (3.45) in the *decoding process* (3.17)

3.45

quantized coefficient

transform coefficient (3.55) before *dequantization* (3.18)

3.46**random access point**

point, other than the beginning of the *bitstream* (3.3), for starting the *decoding process* (3.17) for the remainder of a bitstream

3.47**raster scan**

mapping of a two-dimensional pattern to a one-dimensional pattern such that the first entries in the one-dimensional pattern are from the top row of the two-dimensional pattern scanned from left to right, followed similarly by the second, third, etc. rows of the pattern (going down) each scanned from left to right

3.48**reconstruction**

addition of the decoded *prediction residual* (3.41) and the corresponding *prediction values* (3.42)

3.49**reference frame**

previously *decoded frame* (3.15) used during *inter prediction* (3.25)

3.50**reserved**

syntax element (3.54) value which may be used to extend this specification in the future

Note 1 to entry: Such values are not allowed to be present in bitstreams conforming to this document.

3.51**sample**

basic element of the array of the *decoded frame* (3.15)

3.52**sample value**

value of a *sample* (3.51), which is an integer in the range of 0 to 255, inclusive

3.53**sub-block**

4x4 *block* (3.5)

3.54**syntax element**

element of data represented in the *bitstream* (3.3)

3.55**transform coefficient**

scalar quantity, considered to be in a frequency domain, that is associated with a particular one-dimensional or two-dimensional *frequency index* (3.23) in an *inverse transform* (3.29) part of the *decoding process* (3.17)

4 Abbreviated terms

DCT	Discrete cosine transform
LSB	Least significant bit
MB	Macroblock
MSB	Most significant bit

SB	Sub-block
VCB	Video coding for browsers
WHT	Walsh hadamard transform

5 Conventions

5.1 General

The mathematical operators and their precedence rules used in this document are similar to those used in the C programming language. However, the operation of integer division with truncation is specifically defined.

In addition, an array with two elements used to hold a motion vector (such as Mv) can be accessed using either normal array notation (e.g. $Mv[0]$ and $Mv[1]$), or by name (i.e. Mv). The only operations defined when using the name are assignment and equality testing. The assignment of an array is represented using the normal notation $A = B$ and is specified to mean the same as doing both the individual assignments $A[0] = B[0]$ and $A[1] = B[1]$. Equality testing of two motion vectors is represented using the notation $A == B$ and is specified to mean the same as $(A[0] == B[0] \&\& A[1] == B[1])$.

5.2 Arithmetic operators

+	Addition
−	Subtraction (as a binary operator) or negation (as a unary prefix operator)
*	Multiplication
/	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.
$a \% b$	Remainder of a divided by b , defined only for $a \geq 0$ and $b > 0$.

5.3 Logical operators

$a \&\& b$	Logical AND operation between a and b
$a b$	Logical OR operation between a and b
!	Logical NOT operation

5.4 Relational operators

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

5.5 Bitwise operators

&	AND operation
	OR operation
~	Negation operation
a >> b	Shift a in 2's complement binary integer representation format to the right by b bit positions. This operator is only used with b being a non-negative integer. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of a prior to the shift operation.
a << b	Shift a in 2's complement binary integer representation format to the left by b bit positions. This operator is only used with b being a non-negative integer. Bits shifted into the LSBs as a result of the left shift have a value equal to 0.

5.6 Assignment operators

=	Assignment operator
++	Increment, x++ is equivalent to x = x + 1. When this operator is used for an array index, the variable value is obtained before the increment operation.
--	Decrement, i.e. x-- is equivalent to x = x - 1. When this operator is used for an array index, the variable value is obtained before the decrement operation.
+=	Addition assignment operator, for example: $x += 3 \text{ corresponds to } x = x + 3, \text{ and,}$ $x += (-3) \text{ is equivalent to } x = x + (-3).$
-=	Subtraction assignment operator, for example: $x -= 3 \text{ corresponds to } x = x - 3, \text{ and,}$ $x -= (-3) \text{ is equivalent to } x = x - (-3).$

5.7 Range notation

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.
----------	--

5.8 Mathematical functions

The following mathematical functions are defined as follows:

$$\text{Abs}(x) = \begin{cases} x; & x \geq 0 \\ -x; & x < 0 \end{cases}$$

$$\text{Clip1}(x) = \text{Clip3}(0, 255, x)$$

$$\text{Clip3}(x, y, z) = \begin{cases} x; & z < x \\ y; & z > y \\ z; & \text{otherwise} \end{cases}$$

$$\text{Min}(x, y) = \begin{cases} x; & x \leq y \\ y; & x > y \end{cases}$$

$$\text{Max}(x, y) = \begin{cases} x; & x \geq y \\ y; & x < y \end{cases}$$

5.9 Order of operation precedence

When order of precedence in an expression is not indicated explicitly by the 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 document is the same as used in the C programming language.

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

operations (with operands x, y, and z)
"x++", "x--"
"!x", "-x" (as a unary prefix operator)
"xy"
"x * y", "x / y", "x ÷ y", " $\frac{x}{y}$ ", "x % y"
"x + y", "x - y" (as a two argument operator), " $\sum_{i=x}^y f(i)$ "
"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"

5.10 Method of describing bitstream syntax

The bitstream syntax is specified in [Clause 6](#) in a style similar to the C programming language. A syntax element is completely described by its name together with either one or two descriptors for its method of coded representation. Syntax element names consist of descriptive groups of lowercase letters separated by an underscore character. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements.

A syntax element is represented in **bold** type in two places; where it is read from the bitstream (in [Clause 6](#)) and where it is defined (in [Clause 7](#)). Otherwise, syntax elements are represented in non-bold type throughout.

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 lowercase and uppercase letters without any underscore characters. Variables starting with an uppercase letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an uppercase 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 lowercase letter are only used within the subclause from which they are derived.

The association of values and names is specified in the text. In some cases, “mnemonic” names for syntax elements or variables are used interchangeably with their numerical values. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an uppercase letter and may contain more uppercase letters.

Hexadecimal notation, indicated by prefixing the hexadecimal number by “0x”, may be used when the number of bits is an integer multiple of 4. For example, “0x1a” represents a bit-string “0001 1010”.

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

[Table 2](#) lists examples of the syntax specification format. When **syntax_element** appears (with bold face font), it specifies that a syntax element is parsed from a bitstream.

Table 2 — Examples of syntax specification formats

syntax	type
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify its existence, type and value, as in the following examples */	
syntax_element	ue(v)
conditioning statement	
/* A group of statements enclosed in brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A “while” structure specifies that the statement is to be evaluated repeatedly while the condition remains true. */	
while (condition)	
statement	
/* A “do ... while” structure executes the statement once and then tests the condition. It repeatedly evaluates the statement while the condition remains true. */	
do	
statement	
while (condition)	

Table 2 (continued)

syntax	type
/* An “if ... else” structure tests the condition first. If it is true, the primary statement is evaluated. Otherwise, the alternative statement is evaluated. If the alternative statement is unnecessary to be evaluated, the “else” and corresponding alternative statement can be omitted. */	
if (condition)	
primary statement	
else	
alternative statement	
/* A “for” structure evaluates the initial statement at the beginning, then tests the condition. If it is true, the primary and subsequent statements are evaluated until the condition becomes false. */	
for (initial statement; condition; subsequent statement)	
primary statement	

5.11 Functions

Functions used for syntax specification are defined in this subclause.

The specification of these functions makes use of a bitstream position indicator. This bitstream position indicator locates the position of the bit that is going to be read next.

get_position(): Return the value of a bitstream position indicator.

init_bool(b, sz): Initialize the decoding process for Boolean decoder b with a size of sz as specified in [9.2.2](#).

set_bool(b): Change to using Boolean decoder b as specified in [9.2.3](#).

5.12 Descriptors

5.12.1 General

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

5.12.2 f(n)

An unsigned n-bit number appearing directly in a bitstream. The bits are read from high to low order. The parsing process specified in [9.1](#) is invoked and the syntax element is set equal to the return value.

5.12.3 B(p)

A single Boolean encoded bit with estimated probability $p \div 256$ of being 0. The syntax element is set equal to the return value of read_bool(p) (see [9.2.4](#) for a specification of this process).

NOTE p might be equal to 0, but there is no special treatment of this case.

5.12.4 u(n)

An unsigned Boolean encoded n-bit number encoded as n flags (a "literal"). The bits are read from high to low order. The syntax element is set equal to the return value of read_literal(n) (see [9.2.5](#) for a specification of this process).

5.12.5 F

A single Boolean encoded bit with estimated probability 50% of being 0. The syntax element is set equal to the return value of `read_bool( 128 )`.

NOTE A flag is the same as `B(128)` or `u(1)`.

5.12.6 P(7)

A Boolean encoded 7-bit specification of an 8-bit estimated probability. Coded as an `u(7)` number x ; the resulting 8-bit estimated probability is $x \gg 1$.

5.12.7 P(8)

A Boolean encoded 8-bit estimated probability. This is the same as `u(8)`, but this notation is used to emphasize that an estimated probability is being coded.

5.12.8 T

A Boolean tree-encoded value from an alphabet. Such values represent the leaves of a binary tree. The non-leaf nodes of the tree have associated probabilities p and are represented by `B(p)`. A zero represents choosing the left branch below the current node and a one represents choosing the right branch. Each element of this type has an associated table of estimated node probability values defined in this document. Reference is made to those tables when required.

Every leaf value whose tree depth is x is decoded using x `B(p)` values.

There are many ways that a given alphabet can be so represented. The choice of tree has little impact on data rate but does affect decoder performance.

6 Bitstream syntax**6.1 General**

This clause specifies the bitstream syntax in a tabular form. The meaning of each of the syntax elements is presented in [Clause 7](#).

6.2 Frame syntax

A bitstream consists of a sequence of one or more coded frames ordered in decoding order. Methods of framing the coded frames in a container format are outside the scope of this document. Each coded frame in turn is given to the decoding process as a bit string together with a variable, named “sz” in [Table 3](#), that specifies the total number of bytes in the coded frame.

Table 3 — Frame syntax

<code>frame(sz) {</code>	Type
<code>uncompressed_header()</code>	
<code>sz -= ControlPartitionSize + HeaderSize</code>	
<code>init_bool(0, ControlPartitionSize)</code>	
<code>set_bool(0)</code>	
<code>compressed_header()</code>	
<code>for (i = 0; i < NbrPartitions - 1; i++) {</code>	
<code>partition_size_low</code>	<code>f(8)</code>
<code>partition_size_mid</code>	<code>f(8)</code>

Table 3 (continued)

partition_size_high	f(8)
sz -= partitionSize[i] + 3	
}	
partitionSize[NbrPartitions - 1] = sz	
for (b = 0; b < NbrPartitions; b++)	
init_bool(b + 1, partitionSize[b])	
for (col = 0; col < MbCols; col++) {	
for (i = 0; i < 9; i++)	
AboveToken[col][i] = 0	
MbRefFrame[-1][col] = 0	
MbSplit[-1][col] = 0	
}	
MbRefFrame[-1][-1] = 0	
MbSplit[-1][-1] = 0	
for (row = 0; row < MbRows; row++) {	
for (i = 0; i < 9; i++)	
LeftToken[i] = 0	
MbRefFrame[row][-1] = 0	
MbSplit[row][-1] = 0	
for (col = 0; col < MbCols; col++) {	
set_bool(0)	
mb_header(row, col)	
set_bool(1 + (row % NbrPartitions))	
prediction_residual_data(row, col)	
}	
}	
}	

6.3 Uncompressed header syntax

uncompressed_header() {	Type
control_partition_size_low	f(3)
show_frame	f(1)
version_number	f(3)
frame_type	f(1)
control_partition_size_mid	f(8)
control_partition_size_high	f(8)
if (KeyFrame) {	
sync_byte_0	f(8)
sync_byte_1	f(8)
sync_byte_2	f(8)
frame_width_low	f(8)
horizontal_scale	f(2)
frame_width_high	f(6)
frame_height_low	f(8)

Table 3 (continued)

vertical_scale	f(2)
frame_height_high	f(6)
HeaderSize = 10	
} else	
HeaderSize = 3	
}	

6.4 Compressed header syntax

6.4.1 General

compressed_header() {	Type
if (KeyFrame) {	
colour_space	u(1)
clamping_type	u(1)
}	
segmentation_enabled	u(1)
if (segmentation_enabled)	
update_segmentation()	
filter_type	u(1)
loop_filter_level	u(6)
sharpness_level	u(3)
mb_lf_adjustments()	
log2_nbr_of_dct_partitions	u(2)
quant_indices()	
if (KeyFrame)	
refresh_entropy_probs	u(1)
else {	
refresh_ref1_frame	u(1)
refresh_ref2_frame	u(1)
if (!refresh_ref1_frame)	
copy_buffer_to_ref1	u(2)
if (!refresh_ref2_frame)	
copy_buffer_to_ref2	u(2)
sign_bias_ref1	u(1)
sign_bias_ref2	u(1)
refresh_entropy_probs	u(1)
refresh_ref0	u(1)
}	
token_prob_update()	
mb_has_skip_coeff	u(1)
if (mb_has_skip_coeff)	
prob_skip_false	P(8)
if (!KeyFrame) {	

prob_intra	P(8)
prob_ref0	P(8)
prob_ref1	P(8)
intra_16x16_prob_update_flag	u(1)
if (intra_16x16_prob_update_flag)	
for (i = 0; i < 4; i++)	
intra_16x16_prob[i]	P(8)
intra_chroma_prob_update_flag	u(1)
if (intra_chroma_prob_update_flag)	
for (i = 0; i < 3; i++)	
intra_chroma_prob[i]	P(8)
mv_prob_update()	
}	
}	

6.4.2 Segment based adjustments syntax

update_segmentation() {	Type
update_mb_segmentation_map	u(1)
update_segment_feature_data	u(1)
if (update_segment_feature_data) {	
segment_feature_mode	u(1)
for (i = 0; i < 4; i++) {	
quantizer_update	u(1)
if (quantizer_update) {	
quantizer_update_value	u(7)
quantizer_update_sign	u(1)
}	
}	
for (i = 0; i < 4; i++) {	
loop_filter_update	u(1)
if (loop_filter_update) {	
lf_update_value	u(6)
lf_update_sign	u(1)
}	
}	
}	
if (update_mb_segmentation_map)	
for (i = 0; i < 3; i++) {	
segment_prob_update	u(1)
if (segment_prob_update)	
segment_prob[i]	P(8)
else	

segment_prob[i] = 255	
}	
}	

6.4.3 Loop filter adjustments syntax

mb_lf_adjustments() {	Type
loop_filter_adj_enable	u(1)
if (loop_filter_adj_enable) {	
mode_ref_lf_delta_update	u(1)
if (mode_ref_lf_delta_update) {	
for (i = 0; i < 4; i++) {	
ref_frame_delta_update_flag	u(1)
if (ref_frame_delta_update_flag) {	
delta_magnitude	u(6)
delta_sign	u(1)
}	
}	
for (i = 0; i < 4; i++) {	
mb_mode_delta_update_flag	u(1)
if (mb_mode_delta_update_flag) {	
mode_delta_magnitude	u(6)
mode_delta_sign	u(1)
}	
}	
}	
}	
}	

6.4.4 Quantizer indices syntax

quant_indices() {	Type
y_ac_qi	u(7)
y_dc_delta_present	u(1)
if (y_dc_delta_present) {	
y_dc_delta_magnitude	u(4)
y_dc_delta_sign	u(1)
}	
y2_dc_delta_present	u(1)
if (y2_dc_delta_present) {	
y2_dc_delta_magnitude	u(4)
y2_dc_delta_sign	u(1)
}	
y2_ac_delta_present	u(1)
if (y2_ac_delta_present) {	

y2_ac_delta_magnitude	u(4)
y2_ac_delta_sign	u(1)
}	
uv_dc_delta_present	u(1)
if (uv_dc_delta_present) {	
uv_dc_delta_magnitude	u(4)
uv_dc_delta_sign	u(1)
}	
uv_ac_delta_present	u(1)
if (uv_ac_delta_present) {	
uv_ac_delta_magnitude	u(4)
uv_ac_delta_sign	u(1)
}	
}	

6.4.5 Token probability update syntax

token_prob_update() {	Type
for (i = 0; i < 4; i++)	
for (j = 0; j < 8; j++)	
for (k = 0; k < 3; k++)	
for (t = 0; t < 11; t++) {	
coeff_prob_update_flag	B(CoeffUpdateProbs[i][j][k][t])
if (coeff_prob_update_flag)	
coeff_prob [i][j][k][t]	P(8)
}	
}	

The contents of CoeffUpdateProbs are specified in [Table 35](#).

6.4.6 MV probability update syntax

mv_prob_update() {	Type
for (i = 0; i < 2; i++)	
for (j = 0; j < 19; j++) {	
mv_prob_update_flag	B(MvUpdateProbs[i][j])
if (mv_prob_update_flag)	
mv_prob [i][j]	P(7)
}	
}	

The contents of MvUpdateProbs are specified in [Table 36](#).

6.5 Macroblock data syntax

6.5.1 Macroblock header syntax

mb_header(row, col) {	Type
if (update_mb_segmentation_map)	
segment_id	T
if (mb_has_skip_coeff)	
mb_skip_coeff	B(prob_skip_false)
if (!KeyFrame)	
is_inter_mb	B(prob_intra)
if (is_inter_mb) {	
mb_ref_frame_sel1	B(prob_ref0)
if (mb_ref_frame_sel1)	
mb_ref_frame_sel2	B(prob_ref1)
mv_mode	T
if (mv_mode == MV_SPLIT) {	
mv_split_mode	T
for (i = 0; i < numMvs; i++) {	
sub_mv_mode	T
if (sub_mv_mode == NEW4X4) {	
SubMvdy = mv_component(0)	
SubMvdx = mv_component(1)	
}	
}	
} else if (mv_mode == MV_NEW) {	
Mvdy = mv_component(0)	
Mvdx = mv_component(1)	
}	
} else { /* intra mb */	
intra_y_mode	T
if (intra_y_mode == B_PRED)	
for (block = 0; block < 16; block++)	
intra_b_mode [block]	T
intra_uv_mode	T
}	
}	

6.5.1.1 MV component syntax

mv_component(cp) {	Type
mv_long	B(mv_prob[cp][0])
if (mv_long) {	
c = 0	
for (i = 0; i < 3; i++) {	

mv_bit	B(mv_prob[cp][9 + i])
c += mv_bit << i	
}	
for (i = 9; i > 3; i--) {	
mv_bit	B(mv_prob[cp][9 + i])
c += mv_bit << i	
}	
if (c & 0xfff0) {	
mv_bit	B(mv_prob[cp][9 + 3])
c += mv_bit << 3	
} else	
c += 8	
} else {	
mv_short	T
c = mv_short	
}	
if (c) {	
mv_sign	B(mv_prob[cp][1])
if (mv_sign)	
c = -c	
}	
return c	
}	

6.5.2 Prediction residual data syntax

prediction_residual_data(row, col) {	Type
HasNonZero = 0	
if ((is_inter_mb && mv_mode != MV_SPLIT)	
(!is_inter_mb && intra_y_mode != B_PRED))	
HasY2 = 1	
else	
HasY2 = 0	
if (mb_skip_coeff == 0) {	
if (HasY2) {	
prediction_residual_block(24, 0) /* Y2 */	
LeftToken[8] = AboveToken[col][8] = NonZero[24]	
for (i = 0; i < 16; i++)	
prediction_residual_block(i, 1) /* Y */	
} else	
for (i = 0; i < 16; i++)	
prediction_residual_block(i, 0) /* Y */	
for (i = 0; i < 8; i++)	
prediction_residual_block(16 + i, 0) /* 4 Cb, 4 Cr */	
for (i = 0; i < 4; i++) {	

LeftToken[i] = NonZero[i * 4 + 3]	
AboveToken[col][i] = NonZero[12 + i]	
}	
for (i = 0; i < 2; i++) {	
LeftToken[4 + i] = NonZero[16 + i * 2 + 1]	
AboveToken[col][4 + i] = NonZero[16 + 2 + i]	
LeftToken[6 + i] = NonZero[16 + 4 + i * 2 + 1]	
AboveToken[col][6 + i] = NonZero[16 + 4 + 2 + i]	
}	
} else	
for (i = 0; i < (HasY2 ? 9 : 8); i++) {	
LeftToken[i] = 0	
AboveToken[col][i] = 0	
}	
}	

6.5.3 Prediction residual block syntax

prediction_residual_block(block, firstCoeff) {	Type
lastCoeff = 1	
breakLoop = 0	
for (i = firstCoeff; i < 16 && !breakLoop; i++) {	
coeff_token	T
if (coeff_token == EOB)	
breakLoop = 1	
else {	
if (coeff_token >= DCT_CAT1) {	
v = 0	
for (j = 0; j < catNum; j++) {	
extra_bit	B(catProbs[coeff_token][j])
v += extra_bit	
}	
coeffVal = catBase + v	
} else	
coeffVal = coeff_token	
if (coeffVal != 0) {	
coef_sign	u(1)
if (coef_sign)	
coeffVal = -coeffVal	
}	
lastCoeff = coeffVal	
y = zigzag[i] >> 2	
x = zigzag[i] - y * 4	
TransCoeff[block][x][y] = coeffVal	
}	

}	
if (i == firstCoeff)	
NonZero[block] = 0	
else {	
NonZero[block] = 1	
HasNonZero = 1	
}	
for (; i < 16; i++) {	
y = zigzag[i] >> 2	
x = zigzag[i] - y * 4	
TransCoeff[block][x][y] = 0	
}	
}	

The zigzag scan is specified as follows:

zigzag[16] = {
0, 1, 4, 8, 5, 2, 3, 6, 9, 12, 13, 10, 7, 11, 14, 15
}

7 Bitstream semantics

7.1 General

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in this clause. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this document. This clause specifies the meaning of the syntax elements.

7.2 Frame semantics

A bitstream consists of a sequence of one or more coded frames. The first frame in a bitstream shall be a key frame.

partition_size_low, **partition_size_mid**, and **partition_size_high** combine to give the number of bytes in the i^{th} partition of the prediction residual data. The variable `partitionSize[ i ]` is specified as:

`partitionSize[ i ] = partition_size_low + (partition_size_mid << 8) + (partition_size_high << 16)`

7.3 Uncompressed header semantics

frame_type specifies whether the current frame is a P frame. The variable `KeyFrame` is specified as follows:

- If `frame_type` is equal to 0, `KeyFrame` is set equal to 1.
- Otherwise (`frame_type` is equal to 1), `KeyFrame` is set equal to 0.

A frame is a key frame if and only if `KeyFrame` is equal to 1.

When `KeyFrame` is equal to 1, the saved probability tables are reset to their default values as follows:

```

for ( i = 0; i < 4; i++ )
    SavedIntra16x16Prob[ i ] = DefaultIntra16x16Prob[ i ]
for ( i = 0; i < 3; i++ )
    SavedIntraChromaProb[ i ] = DefaultIntraChromaProb[ i ]
for ( i = 0; i < 4; i++ )
    for ( j = 0; j < 8; j++ )
        for ( k = 0; k < 3; k++ )
            for ( t = 0; t < 11; t++ )
                SavedCoeffProb[ i ][ j ][ k ][ t ] = DefaultCoeffProb[ i ][ j ][ k ][ t ]
for ( i = 0; i < 2; i++ )
    for ( j = 0; j < 19; j++ )
        SavedMvProb[ i ][ j ] = DefaultMvProb[ i ][ j ]

```

The contents of the arrays DefaultIntra16x16Prob and DefaultIntraChromaProb are specified in [Table 37](#).

The contents of DefaultCoeffProb are specified in [Table 38](#).

The contents of DefaultMvProb are specified in [Table 39](#).

The saved probability tables persist across frames.

The first coded frame shall have KeyFrame equal to 1.

version_number specifies the filters used for reconstruction and loop filtering as shown in [Table 4](#).

Table 4 — Convolution filter and loop filter types according to version_number

version_number	Convolution Filter	Loop Filter
0	Bicubic	Normal
1	Bilinear	Simple
2	Bilinear	None
3	None	None
4, 5, 6, 7	Reserved for future use	

The version_number specifies the worst case loop filter that may be present. A bitstream may have version_number equal to 0 and filter_type equal to 1 indicating the simple loop filter is to be used.

show_frame specifies whether the current frame is meant to be displayed.

control_partition_size_low, **control_partition_size_mid**, **control_partition_size_high** specify the size of the first partition, called the control partition, in bytes, excluding the uncompressed data chunk. The variable ControlPartitionSize is set equal to control_partition_size_low + (control_partition_size_mid << 3) + (control_partition_size_high << 11).

sync_byte_0 shall be 0x9d.

sync_byte_1 shall be 0x01.

sync_byte_2 shall be 0x2a.

horizontal_scale and **vertical_scale** specify the frame scaling ratio for each dimension as shown in [Table 5](#).

Table 5 — Frame width and frame height scaling ratio specifications

Value	Up-scaling ratio
0	1 : 1
1	5 : 4
2	5 : 3
3	2 : 1

NOTE 1 Up-scaling does not affect the reconstruction buffer, which is maintained at the encoded resolution.

frame_width_low, **frame_width_high** and **frame_height_low**, **frame_height_high** specify the dimensions of the key frame and the following P frames. The variables **FrameWidth** and **FrameHeight** are specified as follows.

- $\text{FrameWidth} = (\text{frame_width_high} \ll 8) + \text{frame_width_low}$
- $\text{FrameHeight} = (\text{frame_height_high} \ll 8) + \text{frame_height_low}$

Both **FrameWidth** and **FrameHeight** shall be greater than 0. The variables **MbRows** and **MbCols** are defined as follows.

- $\text{MbCols} = (\text{FrameWidth} + 15) \gg 4$
- $\text{MbRows} = (\text{FrameHeight} + 15) \gg 4$

The decoding process operates on complete macroblocks. In a decoder implementation, the current frame and reference frame buffers shall have enough storage to hold all the sample values for a frame that is **MbCols** macroblocks wide and **MbRows** macroblocks high.

NOTE 2 The frame dimensions are not necessarily evenly divisible by 16. The right-most $\text{MbCols} * 16 - \text{FrameWidth}$ columns and the upper-most $\text{MbRows} * 16 - \text{FrameHeight}$ rows are not actually part of the decoded frame and are discarded from the final output. However, these "excess samples" are maintained in the internal reconstruction buffer that is used to predict subsequent frames.

7.4 Compressed header semantics

7.4.1 General

colour_space shall be 0 for bitstreams conforming to this document. All other values are reserved for future use.

clamping_type specifies whether clamping can be ignored in the reconstruction process for this frame and the following P frames.

The sample value clamping type bit is interpreted as follows.

- 0: Reconstructed sample values need to be clamped between 0 and 255 (inclusive).
- 1: Reconstructed sample values are guaranteed to be between 0 and 255; no clamping is necessary.

When **clamping_type** is not present, it is inferred to be equal to the value in the previous frame.

segmentation_enabled specifies that segmentation is used for the current frame.

filter_type specifies the loop filter type. Two types are defined in this document: the "normal" filter type (corresponding to a value of 0) and the "simple" loop filter type (corresponding to a value of 1).

If **version_number** is equal to 1, then **filter_type** shall be equal to 1.

loop_filter_level specifies one of the control parameters of the deblocking filter.

sharpness_level specifies one of the control parameters of the deblocking filter.

log2_nbr_of_dct_partitions determines the number of separate partitions containing the DCT coefficients of the macroblocks. The variable NbrPartitions is set equal to $1 \ll \log2_nbr_of_dct_partitions$.

refresh_entropy_probs specifies whether the updated token probabilities are also used for subsequent frames.

refresh_ref1_frame specifies whether the current decoded frame refreshes the Ref1 reference frame. When not present, it is inferred equal to 1.

refresh_ref2_frame specifies whether the current decoded frame refreshes the Ref2 reference frame. When not present, it is inferred equal to 1.

copy_buffer_to_ref1 specifies how to update the Ref1 reference frame. When not present, it is inferred equal to 0. The value is interpreted as follows.

- 0: no change is made to the Ref1 frame.
- 1: Ref0 is to be copied to the Ref1 frame.
- 2: Ref2 is to be copied to the Ref1 frame.

copy_buffer_to_ref1 shall not be equal to 3.

copy_buffer_to_ref2 specifies how to update the Ref2 frame. When not present, it is inferred equal to 0. The value is interpreted as follows.

- 0: no change is made to the Ref2 frame.
- 1: Ref0 is to be copied to the Ref2 frame.
- 2: Ref1 frame is to be copied to the Ref2 frame.

copy_buffer_to_ref2 shall not be equal to 3.

sign_bias_ref1 controls the sign of motion vectors when the Ref1 frame is referenced.

sign_bias_ref2 controls the sign of motion vectors when the Ref2 frame is referenced.

The array entries of RefFrameSignBias are set as follows.

- RefFrameSignBias[1] = 0
- RefFrameSignBias[2] = sign_bias_ref1
- RefFrameSignBias[3] = sign_bias_ref2

refresh_ref0 specifies whether the current decoded frame refreshes the Ref0 reference buffer. When refresh_ref0 is not present, it is inferred equal to 1.

mb_has_skip_coeff specifies whether mb_skip_coeff is present in the macroblock headers.

prob_skip_false specifies the probability that the macroblock is not skipped.

prob_intra specifies the probability that the macroblock uses intra coding.

prob_ref0 specifies the probability that the Ref0 reference frame is used for inter prediction.

prob_ref1 specifies the probability that the Ref1 reference frame is used for inter prediction.

intra_16x16_prob_update_flag specifies whether the branch probabilities used in the decoding of the luma intra prediction mode are updated.