SMPTE ST 2117-1:2023
Revision of SMPTE ST 2117-1:2020

SMPTE STANDARD

VC-6 Multiplanar Picture Format – Part 1: Elementary Bitstream

Table of Contents

Approved December 30, 2022

List of Tables

List of Figures

Foreword

SMPTE (the Society of Motion Picture and Television Engineers) is an internationally-recognized standards developing organization. Headquartered and incorporated in the United States of America, SMPTE has members in over 80 countries on six continents. SMPTE's Engineering Documents, including Standards, Recommended Practices, and Engineering Guidelines, are prepared by SMPTE's Technology Committees. Participation in these Committees is open to all with a bona fide interest in their work. SMPTE cooperates closely with other standards-developing organizations, including ISO, IEC and ITU.

SMPTE Engineering Documents are drafted in accordance with the rules given in its Standards Operations Manual. This SMPTE Engineering Document was prepared by Technology Committee 10E.

Intellectual Property

At the time of publication no notice had been received by SMPTE claiming patent rights essential to the implementation of this Engineering Document. However, attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. SMPTE shall not be held responsible for identifying any or all such patent rights.

Introduction

This clause is entirely informative and does not form an integral part of this Engineering Document.

VC-6 is a flexible, multi-resolution, intra-only bitstream format, capable of compressing any ordered set of integer element grids, each of independent size, within limits specified in this standard, but is also designed for picture compression. It employs data agnostic techniques for compression, and is capable of compressing low or high bit-depth pictures. The bitstream's headers can contain a variety of metadata about the picture. The codec can operate in both lossless and lossy mode.

Example applications include contribution links that can be decoded on a variety of devices, where the maximal resolution of the decoded picture is determined at the encoder and the displayed resolution from the decoder can be chosen based on its configuration. This standard can be used for mastering and content exchange of TV and film imagery of all genres.

This standard's features and benefits include:

- Bitstream
    - Support for up to 255 components, e.g., colors and alpha, represented in individual component planes, each with any resolution smaller than $65{,}536$ by $65{,}536$. Component plane sizes are independent of each other.
    - Prioritization of particular locations in a picture.
    - Simple, repetitive bitstream payload structure with a shared header and limited co-dependency.
    - Extensive support for injection of metadata.
- Encoder
    - Potential for different encoder implementations to employ different methods to produce bitstreams that conform to this standard, as this standard only specifies a bitstream format and generic decoding processes.
    - Ability of a conformant bitstream to specify any valid combinations of reduced and extended syntax options, so as to keep the choice of optimal bitstream format entirely an encoder's decision.

- Decoder
    - Ability of decoders to independently select a resolution according to processor and display capability.
    - Elimination of blocking artifacts.
    - Power consumption/quality trade-off at decoder end.
- Codec
    - Lossless quality, assuming a sufficient bitrate.
    - Nonlinear upsampling, configurable in both encoder and decoder.
    - Efficient, multi-resolution format.

There are two significant differences between this standard's hierarchical coding techniques and those of discrete wavelet transform-based formats:

- This standard can use more elements in its decomposition scheme than in the actual picture. It is unnecessary to send all decomposition elements explicitly; instead, residual data is sent to correct already-calculated predictions. Residual data takes up fewer bits when encoded, setting this standard apart from wavelet transforms.
- This standard uses upsampling techniques that might not be suitable for use in wavelet transforms.

This standard also employs additional hierarchical coding techniques that are distinct from those used in predictive coding, sub-band coding-based codecs and discrete wavelet transform-based formats. These techniques involve data structures called s-trees. S-trees have several features similar to, and several features dissimilar to, those commonly associated with quadtrees, although s-trees contain implicit/explicit information not usually found in a quadtree. For example, in an s-tree:

- An internal, delimited tier structure maps an s-tree to a bitstream payload.
- The grid to which an s-tree is mapped is not necessarily square; it can be rectangular.
- Nodes have a variety of roles, and each node will or will not carry a label with role-specific meanings.
- Some nodes in s-trees cannot have children, due to the specific dimensions of the grids.

Applications include contribution links that can be decoded on a variety of devices where the maximal resolution of the decoded picture is determined at the encoder and the displayed resolution from the decoder can be chosen based on its configuration.

Precise definitions of s-tree-related terms are provided in Clause 4.

1 Scope

This standard specifies:

- The generic syntax and semantics of the VC-6 bitstream for the interchange of high quality compressed planar images.
- The generic processes that can be used to calculate grids of component values at specified resolutions from a VC-6 bitstream.

Interplanar processing, bitstream processing, performance considerations, and container formats are out of scope.

2 Conformance Notation

Normative text is text that describes elements of the design that are indispensable or contains the conformance language keywords: "shall", "should", or "may". Informative text is text that is potentially helpful to the user, but not indispensable, and can be removed, changed, or added editorially without affecting interoperability. Informative text does not contain any conformance keywords.

All text in this document is, by default, normative, except: the Introduction, any clause explicitly labeled as "Informative" or individual paragraphs that start with "NOTE" or “NOTE” followed by a number. Within a clause, multiple notes are numbered.

The keywords "shall" and "shall not" indicate requirements strictly to be followed in order to conform to the document and from which no deviation is permitted.

The keywords, "should" and "should not" indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

The keywords "may" and "need not" indicate courses of action permissible within the limits of the document.

The keyword “reserved” indicates a provision that is not defined at this time, shall not be used, and may be defined in the future. The keyword “forbidden” indicates “reserved” and in addition indicates that the provision will never be defined in the future.

A conformant implementation according to this document is one that includes all mandatory provisions ("shall") and, if implemented, then all recommended provisions ("should") as described. A conformant implementation need not implement optional provisions ("may") and need not implement them as described.

Unless otherwise specified, the order of precedence of the types of normative information in this document shall be as follows: Normative prose shall be the authoritative definition; tables shall be next; then formal languages; then figures; and then any other language forms.

3 Normative References

The following standards and related documents contain provisions that, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the standards indicated as follows:

- Institute of Electrical and Electronics Engineers (IEEE), IEEE Std 754-2008, “Standard for Floating-Point Arithmetic”

4 Terms and Definitions

For the purposes of this document, the terms and definitions in Clause 4.1 through the end of Clause 4 shall 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

The terms and definitions are presented in two formats: an alphabetical list with hyperlinks to the respective definitions of the terms, and a hierarchical list that shows the relationship between various terms. The alphabetical list of terms is specified in Clause 4.1. The hierarchical list comprises the remainder of Clause 4.

4.1 Alphabetical List of Terms

Terms used in this document and links to their corresponding definitions are listed in Table 1.

Table 1 — Alphabetical List of Terms

Term	Clause
active volume	4.2.7
altitude	4.3.3
bitstream	4.6.2
component value	4.8.6.1
composition	4.8.2
core-echelon	4.7.8.1
depth	4.3.2
depth-first pre-order traversal	4.9.3
desparsification	4.9.2
dequantization	4.9.5
dequantization offset	4.10.5.2
direction	4.7.5.1
distance	4.3.6
echelon	4.7.8
echelon index	4.7.9
entropy decoding	4.9.1
EPS	4.6.3
EPS shortcut	4.6.5
expanded picture structure	4.6.3
expanded picture structure shortcut	4.6.5
final s-tree	4.2.5
graft	4.2.8
grandchild factor	4.4.2.1
GrandchildFactorRMFCatalog	4.10.6.1.1
graph	4.2.1
grid	4.7.1

Term	Clause
index-difference	4.4.2.4
IndexDifferenceRMFCatalog	4.10.6.1.4
junction-tableau	4.5.2.2
junction-tier	4.5.1.3
junction-tableau-set	4.10.7.1
knot	4.10.4.1
label	4.4.1
layer	4.3.4
line-segment	4.10.4
mini-tree	4.5.4
multiplanar picture format	4.7.4
node	4.2.6
path	4.3.1
perfect counterpart	4.2.3
picture	4.7.3
plane	4.7.7
plane creation	4.8.1
plane creation cycle	4.8.1.1
plane stack	4.8.3
predicted-plane	4.8.5
quadrant	4.7.2.1
quadrant array	4.7.2.2
quantization	4.9.4
quantization parameters tuple	4.10.5
quantization parameters tuple catalog	4.10.6.2
range interval	4.10.1
range mapping function	4.10.2
range mapping threshold	4.10.3
RMF	4.10.2
RMT	4.10.3
resid-val	4.4.2.2
ResidualRMFCatalog	4.10.6.1.2
residual-grid	4.7.5
resultant-plane	4.8.6
rise	4.3.5
root-tableau	4.5.2.3
root tableau-layer	4.5.3.1
root-tier	4.5.1.2
s-tree	4.2.4
s-tree assembly	4.8.1.2
sparse	4.2.9

Term	Clause
sparsified area	4.7.6
sparsity	4.2.9.1
stepwidth	4.10.5.1
stratum	4.8.7
stream	4.6.6
streamlength-difference	4.4.2.3
StreamlengthDifferenceRMFCatalog	4.10.6.1.3
summit	4.7.2
symbol	4.4.2
syntax term	4.6.4
tableau	4.5.2
tableau-layer	4.5.3
tableau metadata AUX catalog	4.10.6.3
tableau metadata catalog	4.10.6
tableau metadata RMF catalog	4.10.6.1
tableau-set-index	4.10.7
terminal-tableau	4.5.2.1
terminal-tier	4.5.1.4
terminal-tableau-set	4.10.7.2
tier	4.5.1
tier index	4.5.1.1
tree	4.2.2
upsampler	4.8.4
VC-6	4.6.1

4.2 Tree Structure

4.2.1
graph
abstract data type (ADT) comprising a set of nodes (vertices) and a set of links (edges) where each link is a two-element subset of the set of nodes

4.2.2
tree
graph, in which an arbitrary node has been designated the root node and every pair of nodes is connected by one set of links

4.2.3
perfect counterpart
s-tree, where every node except at altitude 0 has 4 child nodes, i.e., all leaf nodes are at maximum altitude

4.2.4
s-tree
representation of a grid in terms of values of grid width and of grid height and an ordered full 4-tree, where specific types of nodes have specific types of labels

4.2.5
final s-tree
representation of a residual-grid by an s-tree of rise 8, 12 or 16, with resid-vals as labels

4.2.6
node
mathematical, featureless and indivisible object

Note 1 to entry: Hierarchical relationships between nodes are described using commonly accepted terms, including: ancestor, descendant, grandparent, parent, child, grandchild, and sibling.

4.2.7
active volume
subset of nodes in every layer of a perfect counterpart, comprising all nodes or ancestors of nodes within a prescribed square or rectangle

4.2.8
graft
function to replace a specified node in tableau-layer 0 of an s-tree with an s-tree (i.e., a tableau) of rise 4

4.2.9
sparse
characteristic of an s-tree or tableau, representing explicitly only part of its data, the remaining data being zeroes that are represented implicitly

4.2.9.1
sparsity
implicit and/or explicit presence of zeroes

4.3 S-tree Measurements

4.3.1
path
set of consecutive links that connects a pair of nodes

4.3.2
depth
number of links in the path between the root node of a tree and a given node

4.3.3
altitude
depth of the node in a given s-tree minus the rise of the given s-tree, the range of which is an integer from -16 to 0 inclusive

4.3.4
layer
set of all nodes, that are at the same depth, in an s-tree, tableau or mini-tree

4.3.5
rise
one less than the number of layers in a given s-tree, and limited to values 4, 8, 12 and 16.

4.3.6
distance
number of links in the path between two nodes

4.4 Labels

4.4.1
label
data, associated with a node, consisting of one or more symbols

4.4.2
symbol
member of a particular ordered collection of values having no duplicates within a single collection, i.e., member of any available alphabet

4.4.2.1
grandchild factor
4-bit code associated with a node in tableau-layer -2, -3 or -4, that specifies the occurrences of grandchild nodes of that node

4.4.2.2
resid-val
value of the label of a node, representing a residual-grid element explicitly

4.4.2.3
streamlength-difference
difference between two streamoffsets

4.4.2.4
index-difference
difference between two tableau-set-indices

4.5 Tiers and Tableaux

4.5.1
tier
set of tableaux, grafted during s-tree assembly, that occupy the same layers, and that consists of metadata or data for constructing an s-tree

4.5.1.1
tier index
zero minus the number of subsequent tiers until a given terminal-tier is reached

EXAMPLE 1
In a 4-tier s-tree:

- The root-tier has tier index -3.
- The following two tiers have tier indices -2 and -1, respectively.
- The terminal-tier has tier index 0.

4.5.1.2
root-tier
junction-tier with the lowest tier index t, of all the junction-tiers of an s-tree

4.5.1.3
junction-tier
tier with tier index less than 0

Note 1 to entry: The number of junction-tiers depends on the grid width and grid height of a residual-grid.

4.5.1.4
terminal-tier
tier with tier index 0

4.5.2
tableau (pl. tableaux)
5-layer quadtree within an s-tree

4.5.2.1
terminal-tableau
tableau located in tier with tier index 0, and whose tableau-layer 0 node labels in the active volume contain resid-vals

4.5.2.2
junction-tableau
tableau located in tier with tier index -1, -2, or -3, and whose tableau-layer 0 node labels in the active volume contain streamlength-differences

4.5.2.3
root-tableau
sole junction-tableau of a root-tier

4.5.3
tableau-layer
hierarchical element of a tableau, the nodes within which can contain metadata or other data, depending upon the nature of the node, and the position and nature of the tableau-layer within the tableau

Note 1 to entry: Tableau-layer index is specified immediately following the tableau-layer term, as in tableau-layer x, where x is an integer, and where $-4 \leq x \leq 0$.

4.5.3.1
root tableau-layer
tableau-layer -4, which contains the root node of a tableau

4.5.4
mini-tree
tree that is a condensed representation of a tableau

4.6 Bitstream

4.6.1
VC-6
SMPTE video codec 6

4.6.2
bitstream
contiguous sequence of data bits that represents an encoded picture and consists of primary, secondary, and tertiary headers, followed by the bitstream payload

4.6.3
expanded picture structure
EPS
range of multiplanar picture formats, where sets of features are fully, partly or not exploited, depending upon EPS shortcuts

4.6.4
syntax term
name of a value represented by a segment in a specific bitstream's primary, secondary or tertiary header

4.6.5
expanded picture structure shortcut
EPS shortcut
permitted transformation made to the bitstream structure to enable lighter-weight syntax

4.6.6
stream
variable-length section of a bitstream payload, from which an entire tableau, belonging to a tier, can be obtained

4.7 Grids, Planes, and Echelons

4.7.1
grid
rectangular 2D array of elements without empty elements

4.7.2
summit
grid, whose elements map to the maximum number of altitude 0 nodes in an s-tree of a given rise, or whose elements map to the maximum number of tableau-layer 0 nodes in a tableau

4.7.2.1
quadrant
one of four square subsets of a summit

4.7.2.2
quadrant array
list of four square subsets of a summit, arranged in the sibling order of the four child nodes of each node in an s-tree.

4.7.3
picture
all plane stacks represented in a bitstream

4.7.4
multiplanar picture format
bitstream format that represents multiple grids of integers, the grids of which can have differing grid widths and differing grid heights

4.7.5
residual-grid
grid of resid-vals represented by an s-tree's terminal-tier

4.7.5.1
direction
average (A), horizontal (H), vertical (V) or diagonal (D) nature of an echelon, residual-grid or resid-val

4.7.6
sparsified area
area, within a residual-grid, whose values are implicit zeroes, and therefore not encoded in the expanded picture structure

4.7.7
plane
representation of a grid of integer value elements

4.7.8
echelon
dequantized residual-grid for a direction, and the tiers and tableaux that specify this grid

Note 1 to entry: The four echelons representing the four directions, A, H, V, and D, are inputs to the plane creation cycle that produces the output resultant-plane.

4.7.8.1
core-echelon
any of four echelons (directions A, H, V, D), each of whose echelon index is $1 - n$, where n, the number of echelons, per direction, in a given plane stack, is specified in the secondary header in the bitstream

4.7.9
echelon index
integer, ech, associated with each echelon in a plane stack, increasing incrementally by 1 with each resultant-plane of increasing size

Note 1 to entry: An echelon index is specified immediately following the echelon term, as in echelon ech, where N is the number of echelons, per direction, and: $-14 \leq -N < ech \leq 0$

Note 2 to entry: When reference is made to an echelon index, it refers collectively to all directions (A, H, V, D), i.e., 4 echelons. When referring to an echelon of a specific direction, the echelon index is followed by the appropriate subscript letter specifying direction (A, H, V, D).

4.8 Plane Creation

4.8.1
plane creation
process of creating a series of resultant-planes starting from the core-echelon and using cycles fed by new resid-vals to derive higher echelons up to a selected echelon index

4.8.1.1
plane creation cycle
step in resultant-plane creation, identified by an echelon index, that upsamples the output from a possible previous step and prepares input for a possible following step

4.8.1.2
s-tree assembly
increase in rise of an s-tree from 4, in increments of 4, to its maximum, by adding tableaux to a root-tableau, tier by tier, in the plane creation cycle

4.8.2
composition
function that converts four residual-grids, of four corresponding echelons of the same echelon index, into a composed residual-grid

4.8.3
plane stack
all resultant-planes for the same component

4.8.4
upsampler
interpolation technique to calculate values in a higher-resolution grid, based on values in a lower-resolution grid

4.8.5
predicted-plane
plane resulting from the application of an upsampler to a resultant-plane

4.8.6
resultant-plane
grid of component values, the result of a plane creation cycle with four inputs that are echelons representing four directions, A, H, V, and D

4.8.6.1
component value
numerical value defined at every element of a resultant-plane

Note 1 to entry: A component value can have any purpose. For pictures or moving pictures, a component value can carry some property of a picture such as a chrominance component, a luminance component, or an alpha component.

4.8.7
stratum (*pl. strata*)
sequence of decoding computational processes

Note 1 to entry: Strata are as follows:

- stratum 0 process sequence:
  - desparsification
  - entropy decoding

  Stratum 0 results in residual-grids.
- stratum 1 process sequence:
  - dequantization
  - composition, the result of which is a composed residual-grid
- stratum 2 process sequence:
  - upsampling
  - summation of a composed residual-grid and predicted-plane, the result of which is a resultant-plane.

4.9 Processes

4.9.1
entropy decoding
decoding of symbols from a bitstream where, in general, some symbols are represented more compactly in that bitstream, and where the alphabet can change implicitly between symbols

4.9.2
desparsification
parsing a stream of the payload to identify locations not defaulting to value 0 in the residual-grid, controlling an entropy decoder, extracting resid-vals from node labels, and writing resid-vals to the residual-grid in those locations

Note 1 to entry: See Clause D.3 for a detailed description of assembly of grids, also referred to as desparsification.

4.9.3
depth-first pre-order traversal
process of traversing an s-tree from least to greatest altitude

4.9.4
quantization
reduction of the resolution in a quantity, using a stepwidth

4.9.5
dequantization
mapping a quantity from a set of fewer values into a larger set with greater resolution, using a stepwidth and an associated offset

4.10 Tableau Metadata

4.10.1
range interval
range of numerical values bounded by a lower and upper threshold used for resid-val decoding

4.10.2
range mapping function
RMF
function that maps any integer n to the upper boundary, v, of the range interval that corresponds to the nth symbol, where $0 \leq v \leq 1$, and where $1 \leq n \leq N$

Note 1 to entry: The symbols are elements of a fixed or configurable list of N symbols, where N is determined by symbol type.

Note 2 to entry: An empty range interval corresponds to a symbol that has no representation.

4.10.3
range mapping threshold
RMT
range mapping function output value for a given integer n

Note 1 to entry: The RMT value for the m^{th} symbol, where $m > 1$ is not less than the RMT value for the $(m - 1)^{\text{th}}$ symbol.

Note 2 to entry: Each value in the range interval $g \leq v < h$ represents the m^{th} symbol, where $m > 1$, and where g and h are respective RMT values for the $(m - 1)^{\text{th}}$ and m^{th} symbols.

Where $g = h$, the m^{th} symbol has no representation.

Note 3 to entry: Each value in the range interval $0 \leq v < f$ represents the first symbol, where f is the RMT value for the first symbol.

Where $f = 0$, the first symbol has no representation.

4.10.4
line-segment
part of a compact representation of an RMF, used for RMFs in tableau metadata catalogs other than GrandchildFactorRMFCatalog

4.10.4.1
knot
each end of a line-segment that is part of a compact representation of an RMF

4.10.5
quantization parameters tuple
tuple consisting of a stepwidth and a dequantization offset

4.10.5.1
stepwidth
minimum quantization or dequantization difference of resid-vals ignoring offsets

4.10.5.2
dequantization offset
offset for dequantization of resid-vals

4.10.6
tableau metadata catalog
list of metadata, where the metadata is one of the following:

- all RMFs for a specific type of symbol potentially needed for tableau retrieval

  Note 1 to entry: See Clause 4.10.6.1.1 through Clause 4.10.6.1.4.

- all quantization parameter tuples potentially needed for tableau retrieval

  Note 2 to entry: See Clause 4.10.6.2.

- a specific metadata item type that is an unsigned integer, with no role in tableau retrieval

  Note 3 to entry: See Clause 4.10.6.3.

4.10.6.1
tableau metadata RMF catalog
tableau metadata catalog of every RMF that can be used to entropy decode a specific symbol

Note 1 to entry: Each tableau metadata RMF catalog is named using camel case, in order to distinguish it from other tableau metadata catalogs that do not contain RMFs.

4.10.6.1.1
GrandchildFactorRMFCatalog
tableau metadata RMF catalog containing every RMF that can be used to decode grandchild factors in junction-tableaux and terminal-tableaux

4.10.6.1.2
ResidualRMFCatalog
tableau metadata RMF catalog containing every RMF that can be used to decode resid-vals in terminal-tableaux

4.10.6.1.3
StreamlengthDifferenceRMFCatalog
tableau metadata RMF catalog containing every RMF that can be used to decode streamlength-difference symbols in junction-tableaux

4.10.6.1.4
IndexDifferenceRMFCatalog
tableau metadata RMF catalog containing an RMF that can be used to decode tableau-set-indices in junction-tableaux

Note 1 to entry: There are two IndexDifferenceRMFCatalogs.

4.10.6.2
quantization parameters tuple catalog
tableau metadata catalog of every quantization parameters tuple that can be used to dequantize resid-vals

4.10.6.3
tableau metadata AUX catalog
tableau metadata catalog, whose metadata is not of named type, and that allows junction- or terminal-tableaux to have user-defined data whose semantics are not defined in this standard, but whose syntax and value are determined by the bitstream.

4.10.7
tableau-set-index (pl. tableau-set-indices)
index of a tuple specifying items of metadata associated with a tableau

Note 1 to entry: The tuple does not specify all items needed to retrieve a tableau, because no IndexDifferenceRMFCatalog item is specified by the tuple.

4.10.7.1
junction-tableau-set
set of all junction-tableaux in the bitstream having identical tableau-set-indices

4.10.7.2
terminal-tableau-set
set of all terminal-tableaux in the bitstream having identical tableau-set-indices

5 Symbols and Abbreviated Terms

Abbreviations used in this document are as specified in Table 2.

Table 2 — Symbols and Abbreviations Used in This Document

Abbreviation/More Frequently Used Name	Expanded or Alternate Name
A	echelon direction: average
ADT	abstract data type
AUX	auxiliary
D	echelon direction: diagonal
EPS	expanded picture structure
EPS shortcut	expanded picture structure shortcut
H	echelon direction: horizontal
RMF	range mapping function
RMT	range mapping threshold
V	echelon direction: vertical

6 Conventions, Notation, Variables, and Grid Operations

6.1 Operation Precision and Use of Operators

The operators and functions described in Clause 6, except where noted otherwise, operate on and return numerical values with unlimited precision and perform internal calculations with unlimited precision.

Operators are used with integer or fractional operands, unless otherwise specified.

6.2 Numbers

6.2.1 Index

An index (*pl. indices*) is an integer constant or variable.

6.2.2 Hexadecimal Numbers

Hexadecimal numbers are prefixed with "$0x$".

Hexadecimal number formats used in this document are as specified in Table 3.

Table 3 — Hexadecimal Number Examples

Hexadecimal Value	Description	Example
$[:xdigit:]$	Digits 0 to 9 or A to F	7
$0x[:xdigit:]+$	Hexadecimal value without fractional part, i.e., integer	$0xA7$
$0x[:xdigit:]+\backslash.[:xdigit:]$	Hexadecimal value, with one fractional digit	$0xB8.8$

6.3 Arithmetic Operators

Arithmetic operators used in this document are as specified in Table 4.

Table 4 — Arithmetic Operators

Operator	Type	Example	Description
$-$	Unary	$-x$	Minus symbol, specifying the negative of the expression to which it is applied.
$+$	Binary	$x + y$	Signed addition (exact).
$-$	Binary	$x - y$	Signed subtraction (exact).
$*$	Binary	$x * y$	Signed multiplication (exact).
$//$	Binary	$x // y$	Integer division with truncation of the result towards negative infinity. EXAMPLE $3 // 4$ results to 0, and $-3 // 4$ results to -1.
$/$	Binary	x / y	Division with a result that includes any fractional part. If any type of rounding needs to be applied to the result, then the type of rounding is specified where the operator is used in the document.
superscript	Binary	x^y	Raising the integer base to the power of the integer exponent.
$\lvert \ldots \rvert$	Unary	$\lvert x \rvert$	Absolute value.
$\%$	Binary	$x \% y$	Modulo, returning the remainder of the result of a division, where x and y are integers.

6.4 Increment and Decrement Operators

Increment and decrement operators used in this document are as specified in Table 5.

Table 5 — Increment and Decrement Operators

Operator	Type	Example	Description
$++$	Unary	$x++$ (postfix)	Increment an operand, with the following variants, based on the position of the operator: • prefix : returns the incremented value of the operand • postfix: returns the original value of the operand
$--$	Unary	$x--$ (postfix)	Decrement an operand, with the following variants, based on the position of the operator: • prefix : returns the decremented value of the operand • postfix: returns the original value of the operand

6.5 Relational Operators

Relational operators evaluate to a Boolean value. Relational operators used in this document are as specified in Table 6.

Table 6 — Relational Operators

Operator	Type	Example	Description
$>$	Binary	$x > y$	True if x is greater than y; otherwise, false
$<$	Binary	$x < y$	True if x is less than y; otherwise, false
$>=, \geq$	Binary	$x >= y$	True if x is greater than or equal to y; otherwise, false
$<=, \leq$	Binary	$x <= y$	True if x is less than or equal to y; otherwise, false
$==$	Binary	$x == y$	True if x is equal to y; otherwise, false

6.6 Summation Operator

The summation operator used in this document is as specified in Table 7.

Table 7 — Summation Operator

Operator	Type	Example	Description
$\sum$	Binary	$\sum_{i=a}^{b} f(i)$	Summation operator, specifying the sum of function values, $f(i)$, with $a \leq i \leq b$, where i, a and b are integers.

6.7 Logical Operators

The result of evaluating a variable or expression used as an argument of a logical operator is of Boolean type with the value:

- True if the variable or expression evaluates to a non-zero value.
- False if the variable or expression evaluates to a zero value.

Logical operators used in this document are as specified in Table 8.

Table 8 — Logical Operators

Operator	Type	Example	Description
||	Binary	$x \,\|\|\, y$	True if either x or y is true, otherwise false
&&	Binary	$x \,\&\&\, y$	True if both x and y are true, otherwise false

6.8 Assignment Operator

The assignment operator used in this document is as specified in Table 9.

Table 9 — Assignment Operator

Operator	Type	Examples	Description
$=$	Binary	$x = y + 1$ $x = f(y)$	Assignment operator, that assigns the expression or function result on the right to the variable on the left.

6.9 Bitwise Operator

The bitwise operator used in this document is as specified in Table 10.

Table 10 — Bitwise Operator

Operator	Type	Example	Description
$\gg$	Binary	$x \gg n$	Arithmetic right shift of a number, x by n bits. Most Significant Bit (MSB) of x, using two's complement, fills the bit positions that have been vacated by the shift operation (sign-preserving bit shift).

6.10 Grid and Tuple Notation

The grid and tuple notation used in this document is as specified in Table 11.

Table 11 — Grid and Tuple Notation

Operator	Description
$A[i, j]$	The element of a grid A, at row i and column j, where: • $A[0,0]$ is the first element. • Integer $i \geq 0$ • Integer $j \geq 0$
(w, h) or plain text w by h	Dimensions of grids where, for example, w and h are the grid width and grid height.
$(x1, x2, \ldots., xn)$	n-tuple.

6.11 Parentheses and Brackets Operators

The parentheses and brackets operators used in this document are as specified in Table 12.

Table 12 — Parentheses and Brackets Operators and Notation

Notation	Type	Example	Description
(…)	Grouping	$x * (y + z)$	Parentheses operator, used to group arithmetic elements.
(…)	Call	$function(x)$	Parentheses operator, used to call a function.
[…]	Indexing	$A[i,j]$	Brackets operator, used to index.
{	Grouping	$t = \begin{cases} 4, \ if \ x \leq 0 \ AND \ y > 0 \\ 0, \ otherwise \end{cases}$	Groups multiple statements

6.12 Floor, Ceiling, Log, Max, and Min Operators

Floor, ceiling, log, max, and min operators used in this document are as specified in Table 13.

Table 13 — Floor, Ceiling, Log, Max, and Min Operators

Notation	Type	Example	Description
⌊…⌋	Unary	$\lfloor x \rfloor$	Specifies the flooring operation, mapping x to the greatest integer number that is less than or equal to x.
⌈…⌉	Unary	$\lceil x \rceil$	Denotes the ceiling operation, mapping x to the least integer number that is greater than or equal to x.
$\lceil log2(\ldots) \rceil$	Unary	$\lceil log2(x) \rceil$	The smallest non-negative, integer, n, satisfying $x \leq 2^n$, where x is a positive integer.
$max(\ldots)$	NA	$max(x1, x2, \ldots . xn)$	Denotes the largest of the numbers x1, x2 … xn.
$min(\ldots)$	NA	$min(x1, x2, \ldots . xn)$	Denotes the smallest of the numbers x1, x2 … xn.

6.13 Variables

The variables used in this document are specified in Table 14.

Table 14 — Variables Used in This Document

Variable(s)	Description
a	altitude
$ComposedResidualGrid$	See D.5.3.2
D_A	grid element value for echelon direction average (A)
D_D	grid element value for echelon direction diagonal (D)
D_H	grid element value for echelon direction horizontal (H)
D_V	grid element value for echelon direction vertical (V)
ech	echelon Index
H	height of a residual-grid (number of rows)
h	height of a resultant-plane (number of rows)
i	various
j	various
k	various
L	number of layers in an s-tree

Variable(s)	Description
m	various
$ModifiedPredictedAverageGrid$	See D.7.5
n	various
ofs	dequantization offset
$PredictedPlane$	See D.7.2
q	quantized resid-val
R	rise of an s-tree
$ResultantPlane$	See D.7.3
$S[\,]$	shortcut, where the number within the brackets specifies the shortcut
s-tree$[L, (W, H), \mathrm{labeltype}]$	s-tree, where number of layers is L, dimension tuple of residual-grid associated with active volume is (W, H), and $\mathrm{labeltype}$ is a set of label values
sw	stepwidth
t	tier index, where t is an integer, and $-3 \leq t \leq 0$
tableau$[(W, H), \mathrm{labeltype}]$	tableau, where number of layers is 5, dimension tuple of residual-grid associated with active volume is (W, H), and $\mathrm{labeltype}$ is as defined in 9.12 for junction-tableaux and terminal-tableaux
W	width of a residual-grid (number of columns)
w	width of a resultant-plane (number of columns)
d, e, p	See Clause D.2.3.
K, L, ndS, p, q, S, S_type, T, T_type, U, U_type, uq, xr, yr, xs, ys, xt, yt, xu, yu	See Clause D.3.

6.14 Grid/Array Operations

6.14.1 Convolution

Convolution is the result of applying a 3-dimensional filter to a 3-dimensional array.

6.14.2 Morton map

A Morton map is a space-filling curve function that for each $(m * n)$ bits returns n integers corresponding to a subdivision in n-space. Morton code, or Z-order, is the inverse of a Morton map.

The full number of subdivisions along any of the n axes in n-space is 2^m, which is the length of each axis. Each subdivision is a hypercube in n-space.

The $(m = 2)$ Morton map applied recursively will calculate a Morton map for larger values of m, i.e., finer subdivisions in n-space.

The Morton map is used herein with $n = 2$ and $m = 1$ to map children of nodes in s-tree to four subdivisions in a grid. If a parent's corresponding subdivision overhangs the grid, then one or more of its child subdivisions overhang the grid. The child subdivisions are quadrant array members. Their parent subdivision is the union of the child subdivisions.

7 VC-6 Process Overview (Informative)

For easier viewing, the process of decoding the VC-6 bitstream headers and payload is illustrated in a graphic that is split into three segments, Figure 1, Figure 2 and Figure 3. This process could be repeated, in which case, each frame would comprise its own bitstream (headers and payload), which could be combined in an unspecified manner (e.g., a container).

Steps 1, 2 and 3 of the process overview are depicted in Figure 1:

1. After commencement of decoding, the bitstream configuration that determines decoder behavior is read from the headers. The primary header specifies the number of plane stacks, the number of echelon indices (resolutions) to be used by each plane stack, and the grid dimensions for the highest echelons of each plane stack, etc.
2. Each individual plane stack that is specified in the header is represented in the payload, using the specified number of resolutions.
3. The decoder optionally recovers user data that was written in step 1.

Step 4 of the process overview is depicted in Figure 2:

4. A given echelon index of the plane stack is chosen for finishing the decoding process. A plane stack can be decoded to provide resultant-planes of multiple sizes. The lowest-ranked resultant-plane is a prerequisite for decoding the resultant-plane with the next higher echelon index, and the following resultant-plane is a prerequisite for decoding the next resultant-plane, and so on, until a final resultant-plane, as determined by the decoder, is decoded.

   In more detail, the process of decoding a resultant-plane involves three strata, and the behavior in each stratum is determined by the bitstream configuration specified by the headers. For example, there are several upsamplers that could be required.

   - In decoder stratum 0, tableaux are decoded from the bitstream payload, tier by tier, to produce quantized contributions to four directions (A, H, V, D), except where the configuration indicates that all contributions to a direction are zero. Simultaneously, when decoding a tableau, if the presence of auxiliary metadata is specified in the bitstream header, then auxiliary metadata associated with the tableau can be recovered.
   - In decoder stratum 1, dequantization (using a quantization parameters tuple specified by the configuration) and composition result in a composed residual-grid for a given echelon index.
   - In decoder stratum 2, a resultant-plane of higher echelon index is produced after a resultant-plane of lower echelon index is upsampled by a factor of 2 and corrected to give the next higher echelon index' resultant-plane using the resid-vals of that next higher echelon index.

Step 5 of the process is depicted in Figure 3.

5. The resulting plane stack is combined with other plane stacks resulting from other, parallel, plane reconstruction processes. Each plane stack provides data for a single component. For example, if there are 4 plane stacks, then each can map to Y, U, V or α, respectively.

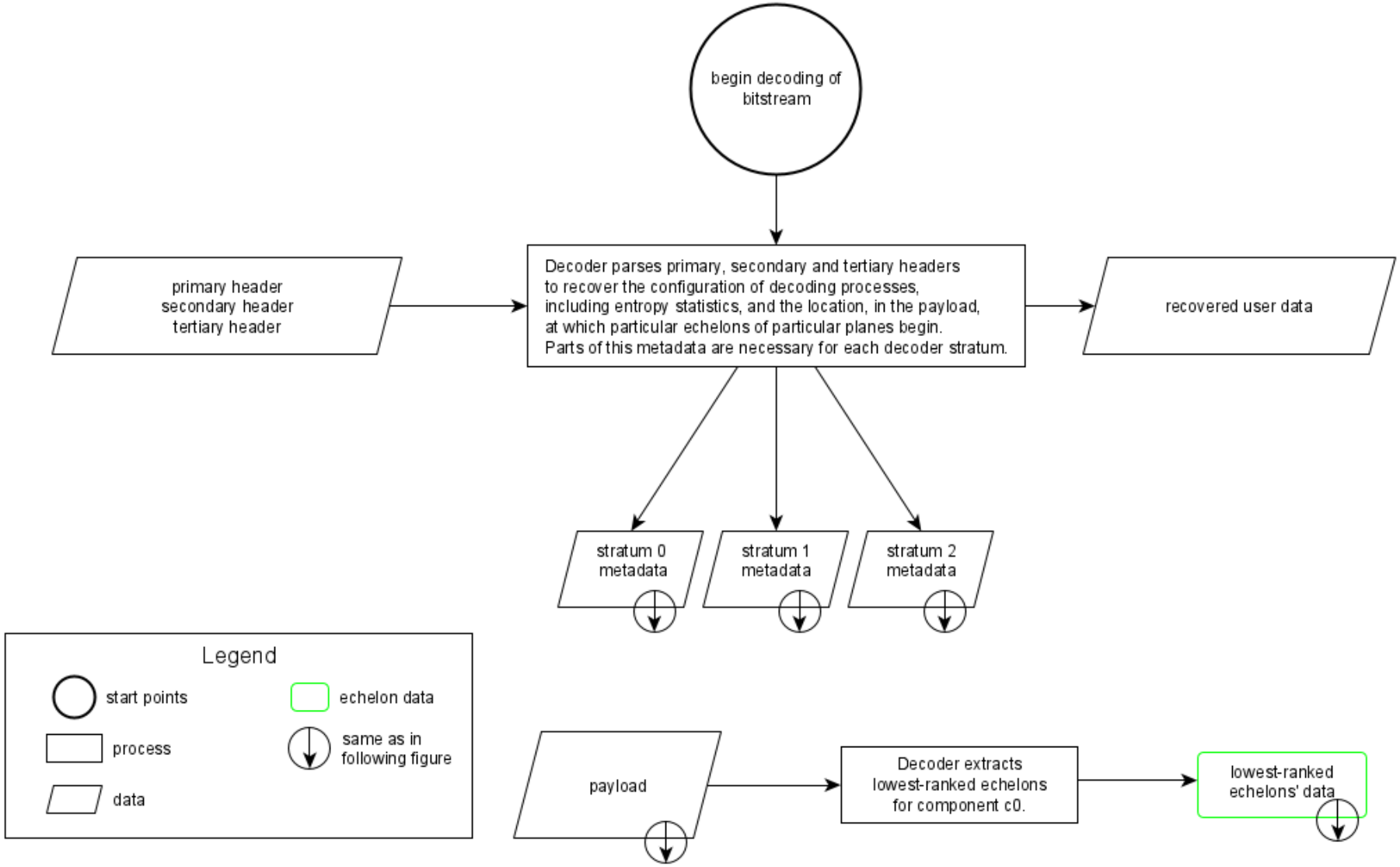


Figure 1 — VC-6 decoding process overview, steps 1, 2 and 3

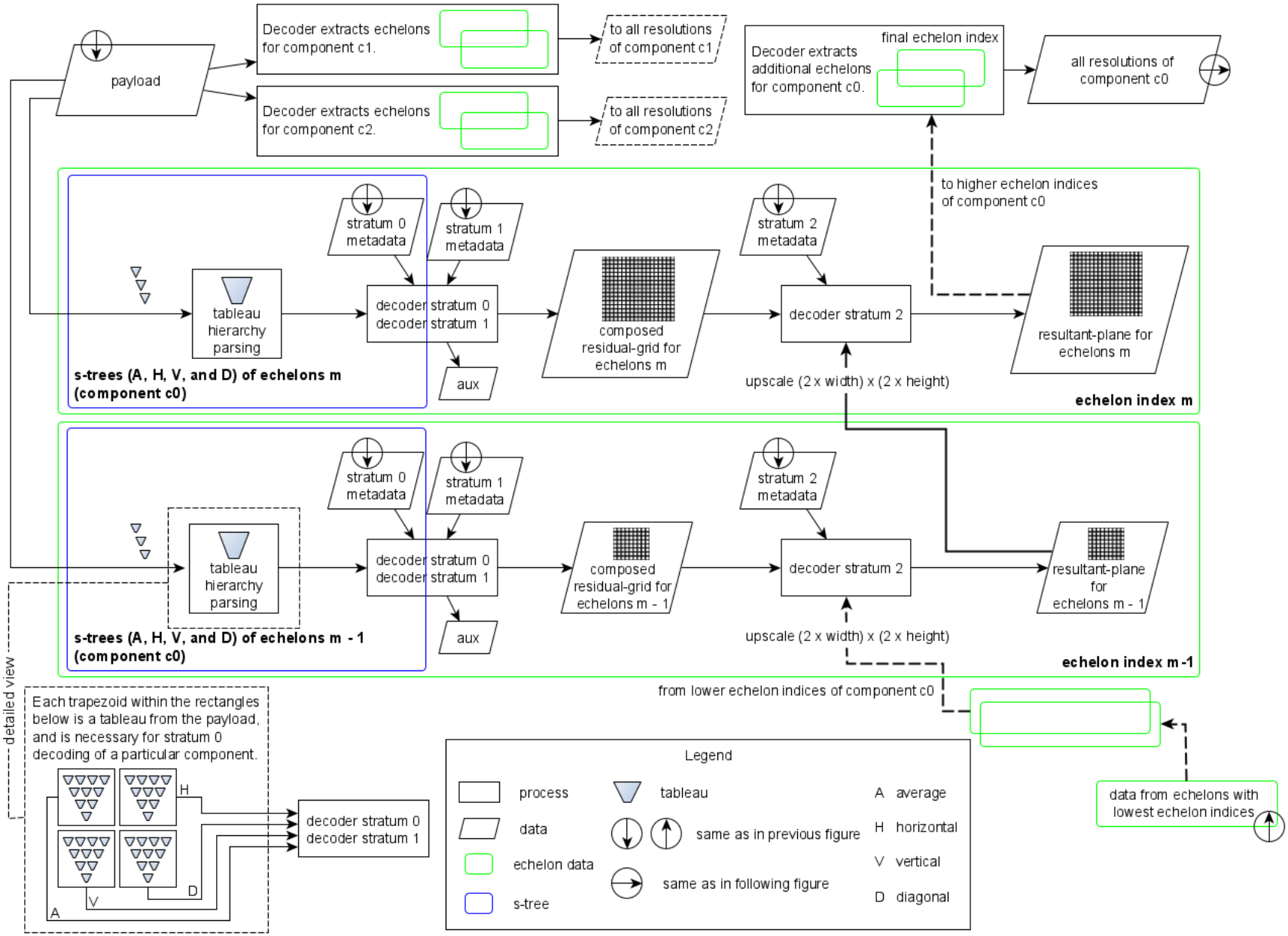


Figure 2 — VC-6 decoding process overview, step 4

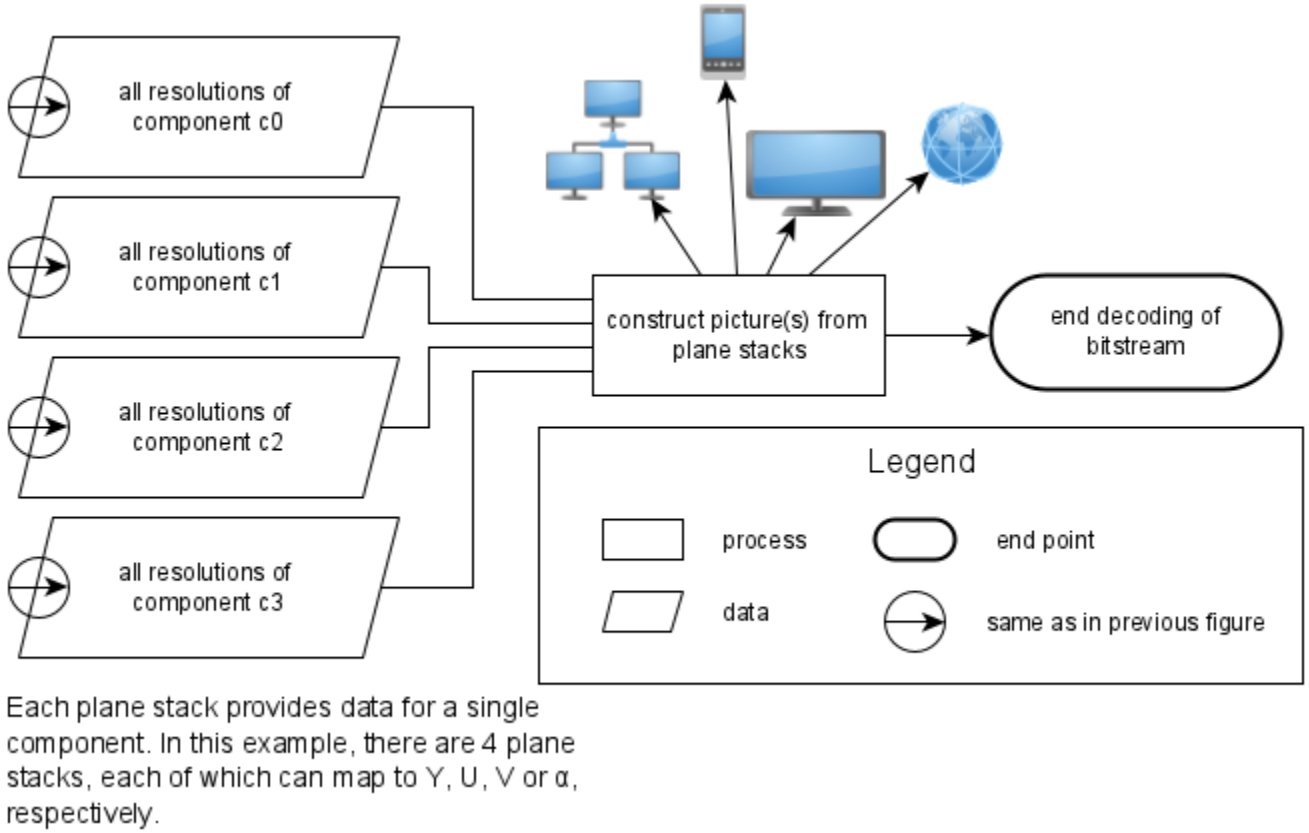


Figure 3 — VC-6 decoding process overview, step 5

See Annex A for additional information about plane reconstruction.

8 Bitstream Syntax and Semantics

8.1 Bitstream Syntax

8.1.1 Bitstream Overview

This standard specifies a bitstream and generic decoding processes. An implementation of the decoding processes is application-specific, and needs access to relevant bits of the bitstream.

NOTE The bitstream might be containerized in one or more pieces (e.g., for transport, processing, storage, etc.), but this is out of scope of this standard and transparent with respect to this standard.

8.1.2 Bitstream Components

The expanded picture structure (EPS) bitstream is comprised of a whole number of bytes.

The bitstream elements and their order shall be as specified in Table 15.

Table 15 — Bitstream Elements and Order

Order	Element	Quantity	Size
1	header comprised of:	1	Multiple of 8 bits.
1a	- primary header	1	Multiple of 8 bits, fixed length of 168 bits.
1b	- secondary header	1	Multiple of 8 bits, variable length.
1c	- tertiary header	1	Multiple of 8 bits, variable length, specifying locations of data in the payload.
2	payload	1	Multiple of 8 bits.

NOTE In this standard, since entropy decoding is performed using a range decoder, there is not a static correspondence between values in mini-trees and bit sequences in the bitstream payload, as there might be in other entropy deciding systems (e.g., in Huffman code formats).

The bitstream structure is illustrated in Figure 4. The bitstream header and structured-type function specifications are specified in 8.4, 8.5, 8.6 and 8.3.3. The processes employed to reconstruct a resultant-plane from a payload are specified in Annex D. Symbol decoding is specified in Annex F.

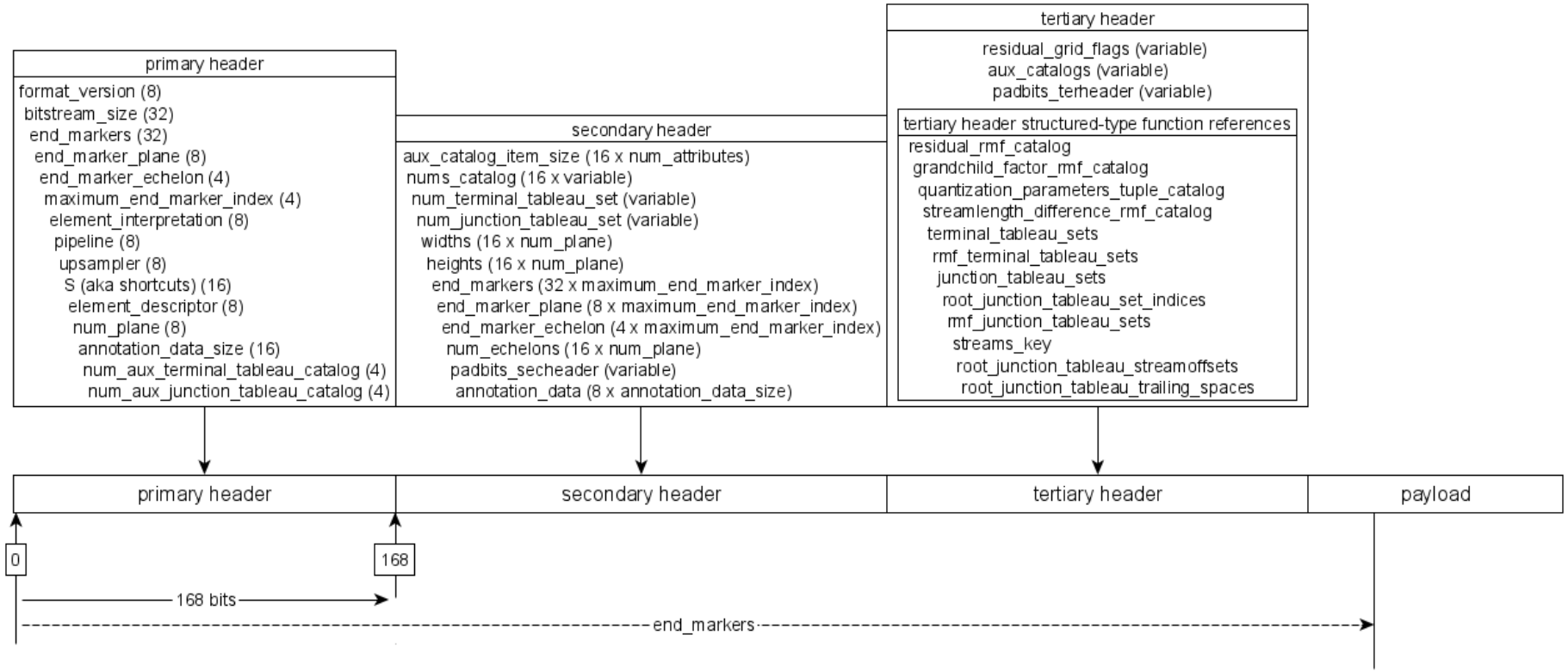


Figure 4 — Bitstream structure

8.2 Syntax Table Conventions for Headers

8.2.1 Syntax Terms

Syntax terms are used in header syntax, as specified in Table 19, Table 21, Table 23, and Table 17.

A syntax term has the following characteristics:

- Every syntax term appears once on the left side of a syntax term assignment statement. In subsequent appearances, the syntax term can appear in an expression on the right side of an assignment statement.

  In the first appearance of a specific syntax term in a header syntax specification in this document, the syntax term name is shown in bold type.

- A syntax term can include trailing indices, each enclosed in square brackets.

  In this document, the square brackets and indices are shown in plain type.

The order of syntax terms in a bitstream header is specified by the order of the assignments of bitstream read function results to syntax terms in Table 19,

Table 21, and Table 23, and the functions in Table 17. See 8.3.2 for additional details pertaining to bitstream read functions used in these tables.

8.2.2 Function Arguments

Structured-type functions have no arguments.

Bitstream read functions and the logarithm function each have an argument, specified within parentheses. The value of the argument is used as the actual value of the symbol `n`, which appears as a formal value in the description of the function.

8.2.3 Variables Used for Intermediate Calculations

Variables used for intermediate calculations in the syntax tables shall be signed integers, and such calculations shall be exact, with no restriction on the size of the integers.

8.2.4 Syntax Term Assignment Statements

A syntax term assignment statement assigns a value to a syntax term, using either of the following formats:

- `syntax_term = read_function(integer_expression)`
  where `read_function()` specifies any one of the functions listed in 8.3.2.

- `syntax_term = structured_type_function()`
  where `structured_type_function()` specifies any one of the functions listed in 8.3.3.

8.2.5 Sequence of Statements

A sequence of indented statements sharing the same level of indentation is a compound statement, and is treated functionally as a single statement.

8.2.6 “For” Structure

A “`for`” structure specifies processing of an initial statement, followed by a specified test of a condition, followed by a subsequent primary statement, which can be a compound statement.

If the condition is `1`, then the “for” structure specifies processing of primary and subsequent statements until the condition is `0`.

An example of the syntax is as follows:

EXAMPLE

```
for (initial_statement; condition; subsequent_statement)

    primary_statement
```

8.2.7 “If…Else” Structure

An “`if … else`” structure specifies a test of a condition, and if the condition is `1`, then specifies processing of a primary statement or otherwise specifies processing of an alternative statement. The “`else`” part and the alternative statement is omitted if there is no alternative processing.

An example of the syntax is as follows:

EXAMPLE

```
if (condition)

      primary_statement

else

      alternative_statement
```

8.3 Functions Used by Syntax Table for Headers

8.3.1 Logarithm Function

The ⌈ log2(n) ⌉ logarithm function is defined in Table 13.

The purpose of this function is to calculate the number of bits used by the bitstream read functions in 8.3.2, when the value of the input arguments of bitstream read functions is not fixed.

8.3.2 Bitstream Read Functions

8.3.2.1 Bitstream Read Functions Overview

The purpose of the bitstream read functions read_bits_unsigned**,** read_bits and read_bits_fractional, is to specify how the bitstream is organized, by indicating how it might be read. Return values of these functions define the values of all syntax terms in the syntax specification of headers and structured-type functions. The functions all take an integer as an input parameter. The position of the next bit to be read from the bitstream is referred to as the read pointer.

Given a sequence of n bits in bitstream order, each of the bitstream read functions partitions this sequence into a list of $(n + 7) // 8$ subsequences of bits—specifically, $n // 8$ octets, followed, if $n \% 8$ is non-zero, by a subsequence of $n \% 8$ remaining bits.

Subsequences are ordered from least-significant to most-significant subsequence. Bits within each subsequence are ordered from most-significant bit to least-significant bit.

NOTE Among Figure 5, Figure 6, Figure 7, Figure 8, and Figure 9, matching color indicates correspondence between the same sets of bits.

A 31-bit portion of a bitstream containing three sequences of lengths 8, 11 and 12 bits, is illustrated in Figure 5.

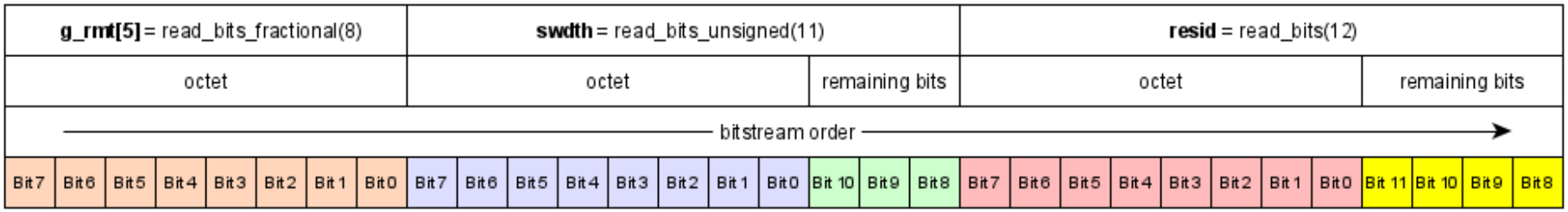


Figure 5 — Example of three successive syntax terms in a tertiary header: g_rmt[5], followed by swdth, followed by resid

8.3.2.2 read_bits_unsigned(n)

The `read_bits_unsigned(n)` bitstream read function is specified as follows:

- The sequence of `n` bits starting at the read pointer shall be converted to the integer result, using unsigned binary notation, by interpreting the bits in the order specified in 8.3.2.1.
- The value of `n` passed to `read_bits_unsigned(n)` is non-zero, positive.
- The read pointer shall advance `n` bits.

The result of the example function `swdth = read_bits_unsigned(11)` from Figure 5 is shown in Figure 6.

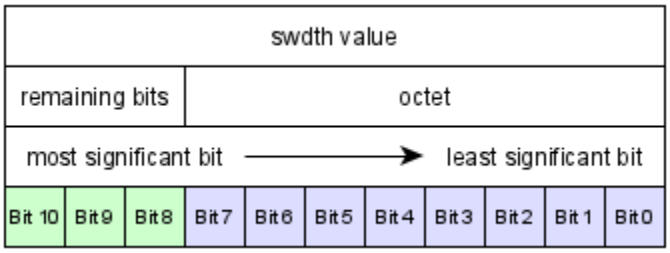


Figure 6 — Example of returned value for swdth from read_bits_unsigned(11)

8.3.2.3 read_bits(n)

The `read_bits(n)` bitstream read function is specified as follows:

- The sequence of `n` bits starting at the read pointer shall be converted to the integer result, using two's complement binary notation, by interpreting the bits in the order specified in 8.3.2.1.
- The value of `n` passed to `read_bits(n)` is non-zero, positive.
- The read pointer shall advance `n` bits.

The result of the example function `resid = read_bits(12)` from Figure 5 is shown in Figure 7.

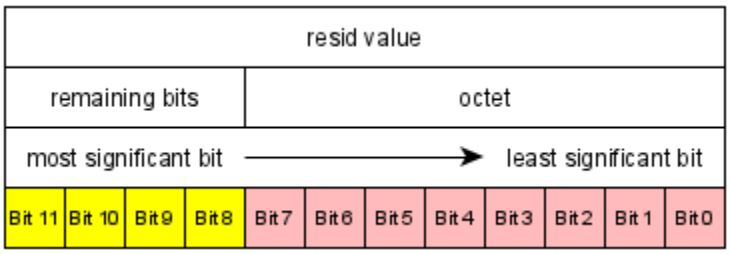


Figure 7 — Example of returned value for resid from read_bits(12)

8.3.2.4 read_bits_fractional(n)

The `read_bits_fractional(n)` bitstream read function is specified as follows:

- The sequence of `n` bits starting at the read pointer shall be converted to a non-negative binary fraction between 0 and $1 - (0.5)^n$, inclusive, using unsigned binary notation for the fractional part, by interpreting the bits in the order specified in 8.3.2.1.
- The value of `n` passed to `read_bits_fractional(n)` is non-zero, positive.
- The read pointer shall advance `n` bits.
- The function returns a value of a non-negative binary fraction referred to as `fractional8 type or fractional16` type, for 8 or 16 fractional bits, respectively.

The result of the example function `g_rmt[5] = read_bits_fractional(8)` from Figure 5 is shown in Figure 8.

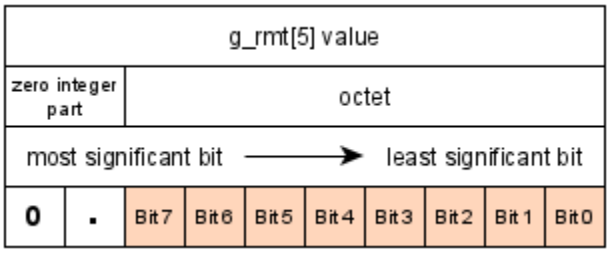


Figure 8 — Example of returned value for g_rmt[5] from read_bits_fractional(8)

A further example of the `read_bits_fractional(n)` function, where two octets are involved, is illustrated in Figure 9.

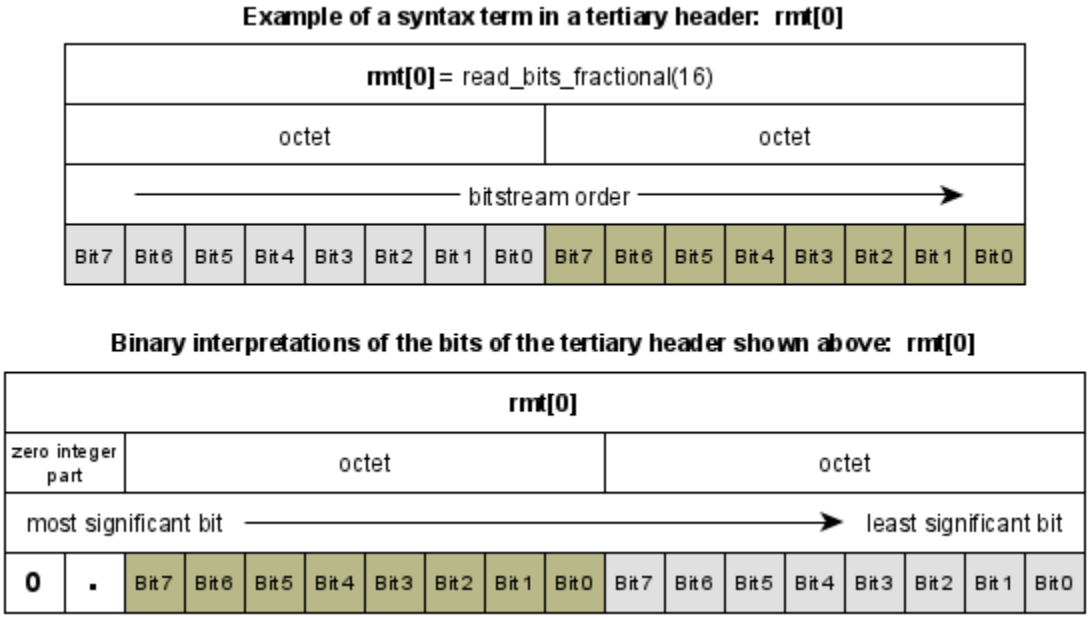


Figure 9 — 16-bit bitstream fragment and binary interpretation from read_bits_fractional(16)

8.3.3 Structured-type Functions

A structured-type function returns an object having one or more fields. Structured-type function names end with "`_t,`" where "`_t`" stands for "`_type.`" These functions do not take an input.

Structured-type functions and their syntax are as specified in Table 16 and Table 17, respectively, and are used for parsing a tertiary header.

Table 16 — Structured-type Functions

Structured-type Function Name	Descriptive Name	Description

`residual_line_segments_rmf15_t`
 	
`streamlength_line_segments_rmf15_t`
 	
`terminal_tableau_set_index_line_segments_rmf15_t`
 	
`junction_tableau_set_index_line_segments_rmf15_t`
	line-segment types	RMF types with 16 points on 15 line-segments. They comprise a list of RMT values and a list of knots.
		• Where the RMT value changes along the line-segment, knots are strictly increasing.
		• Where the RMT value does not change along the line segment, knots are non-decreasing.

`residual_t`	residual type	A signed integer type.
`residual_rmf_t`	ResidualRMF type	An RMF in a ResidualRMFCatalog of potential RMFs for use by terminal-tableau-sets with their tableaux' resid-vals.
`grandchild_factor_rmf_t`	GrandchildFactorRMF type	An RMF in a GrandchildFactorRMFCatalog of potential RMFs for use by terminal-tableau-sets and junction-tableau-sets with their tableaux' grandchild factors.
`stepwidth_t`	stepwidth type	A numerical type for a stepwidth.
`quantization_parameters_tuple_t`	quantization parameters tuple type	A quantization parameters tuple in a quantization parameters tuple catalog of potential quantization parameters tuples for use by terminal-tableau-sets.
`streamlength_difference_rmf_t`	StreamlengthDifferenceRMF type	An RMF in a StreamlengthDifferenceRMFCatalog of potential RMFs for use by junction-tableau-sets with their tableaux' streamlength-differences.
`catalog_item_index_t`	catalog index type	A non-negative integer type capable of indexing the items (e.g., RMFs) in the one or more catalog(s) that contain the highest number of items. A catalog index type is used by terminal-tableau-sets and junction-tableau-sets.
`junction_tableau_set_index_t`	junction-tableau-set-index type	A type capable of indexing each junction-tableau-set.

Structured-type Function Name	Descriptive Name	Description
permutation_descriptior_t	permutation-descriptor type	A type able to specify the significance, from integers 0 (lowest) to 3 (highest), of plane stacks, echelons, directions and tiers in a bitstream payload, in any order.
root_junction_tableau_streamoffsets_t	root-junction-tableau-streamoffsets type	A type capable of holding an offset, measured in bytes, from the end of the tertiary header to the root-tableau of any s-tree.
streamlength_t	streamlength type	A difference in offsets measured in bytes.
terminal_tableau_set_index_t	terminal-tableau-set-index type	A type capable of indexing each terminal-tableau-set.

Table 17 — Structured-type Function Syntax

Specification	Line
residual_line_segments_rmf15_t()	1
for (i = 0; i < 14; i++)	2
object.**rmt**[i] = read_bits_fractional(16)	3
for (i = 0; i < 16; i++)	4
object.**knot**[i] = residual_t()	5
return(object)	6
streamlength_line_segments_rmf15_t()	7
for (i = 0; i < 14; i++)	8
object.**rmt**[i] = read_bits_fractional(16)	9
for (i = 0; i < 16; i++)	10
object.**knot**[i] = streamlength_t()	11
return(object)	12
terminal_tableau_set_index_line_segments_rmf15_t()	13
for (i = 0; i < 14; i++)	14
object.**rmt**[i] = read_bits_fractional(16)	15
for (i = 0; i < 16; i++)	16
object.**knot**[i] = terminal_tableau_set_index_t()	17
return(object)	18

Specification	Line
junction_tableau_set_index_line_segments_rmf15_t()	19
for (i = 0; i < 14; i++)	20
object.**rmt**[i] = read_bits_fractional(16)	21
for (i = 0; i < 16; i++)	22
object.**knot**[i] = junction_tableau_set_index_t()	23
return(object)	24
residual_t()	25
object.**resid** = read_bits(1 + element_descriptor[0] + element_descriptor[1] + 2)	26
return(object)	27
residual_rmf_t()	28
object.**resrmf** = residual_line_segments_rmf15_t()	29
return(object)	30
grandchild_factor_rmf_t()	31
for (i = 0; i < 6; i++)	32
object.**g_rmt**[i] = read_bits_fractional(8)	33
return(object)	34
stepwidth_t()	35
object.**swdth** = read_bits_unsigned(element_descriptor[0] + element_descriptor[1] + 2)	36
return(object)	37
quantization_parameters_tuple_t()	38
object.**stepwidth** = stepwidth_t()	39
object.**dq_shift** = residual_t()	40
return(object)	41
streamlength_difference_rmf_t()	42
object.**streamrmf** = streamlength_line_segments_rmf15_t()	43
return(object)	44
catalog_item_index_t()	45
object.**itemidx** = read_bits_unsigned(⌈ log2(max_num_catalog) ⌉)	46
return(object)	47
junction_tableau_set_index_t()	48
object.**junction_tableau_index** = read_bits(1 + ⌈ log2(num_junction_tableau_set) ⌉)	49
return(object)	50

Specification	Line
permutation_descriptor_t()	51
/* return d[0],d[1],d[2],d[3], of values 0,1,2,3 for code 0 and 3,2,1,0 for code 23, etc.*/	52
object.**code** = read_bits_unsigned(5)	53
object.**permutation_usage_flag** = read_bits_unsigned(3)	54
if n < 24	55
d[0] = code // 6	56
r = (code – 6 * d[0]) // 2	57
s = 0	58
t = code – 6 * d[0] – 2 * r	59
if r >= d[0]	60
r = r + 1	61
lo = min(d[0],r)	62
hi = max(d[0],r)	63
if (s == lo)	64
s = s + 1	65
if (s == hi)	66
s = s + 1	67
if (s == lo – t)	68
s = s + t	69
if (s == hi – t)	70
s = s + t	71
d[1] = r	72
d[2] = s + t	73
d[3] = 6 – d[0] – d[1] – d[2]	74
object.d = d	75
return(object)	76
root_junction_tableau_streamoffsets_t()	77
object.**rootof** = read_bits(⌈ log2(bitstream_size) ⌉)	78
return(object)	79

Specification	Line
streamlength_t()	80
sz_terminal_tableau = 210 + 256 * (16 + 1 + element_descriptor[1] + element_descriptor[2] + 2)	81
big_streamoffset = sz_terminal_tableau	82
test = 16 + 1 + ⌈ log2(1 + (sz_terminal_tableau + 7) // 8) ⌉	83
if (S[2] == 0)	84
test = test + 16 + 1 + ⌈ log2(num_terminal_tableau_sets) ⌉	85
test = 210 + 256 * test	86
if test > big_streamoffset	87
big_streamoffset = test	88
test = 16 + 1 + ⌈ log2((test + 7) // 8) ⌉	89
if (S[7] == 0)	90
test = test + 16 + 1 + ⌈ log2(num_junction_tableau_sets) ⌉	91
test = 210 + 256 * test	92
if test > big_streamoffset	93
big_streamoffset = test	94
sz = 1 + ⌈ log2(1 + (big_streamoffset + 7) // 8) ⌉	95
object.**strof** = read_bits(sz)	96
return(object)	97
terminal_tableau_set_index_t()	98
object.**terminal_tableau_index** = read_bits(1 + ⌈ log2(num_terminal_tableau_set) ⌉)	99
return(object)	100

8.4 Primary Header

The primary header:

- Contains fixed-size configuration data, such as the number of plane stacks and bitvectors needed to configure a decoder, and contains no arrays, apart from the array associated with the pipeline bitvector.
- Includes the size, in bytes, of optional, annotation data to be embedded as the last item in the secondary header, but whose value has no relevance as far as this standard is concerned.
- Includes the number of tableau metadata AUX catalogs whose items can be associated with a tableau, despite the items having no further relevance to this standard.
- Specifies bits, `S[k]`, of the shortcut bitvector, pertaining to an entire bitstream payload, where `k` ranges from `1` to `16`. The subset of these variables and their effects listed in Clause 8.8.4 and Annex B define the syntax of the secondary and tertiary headers. These bits control the structure of mini-trees, for example, enabling or disabling simplified interpretations of mini-tree labels, and influencing compression efficiency.

The primary header and its bitstream syntax are specified in Table 18 and Table 19, respectively.

Table 18 — Primary Header

Field Name	Starting Offset	Description/Constraints
`format_version`	`0x00`	The version of this VC-6 bitstream. This value is `0x10`.
`bitstream_size`	`0x01`	The size of the bitstream in bytes.
`end_markers[0]`	`0x05`	A first size in bytes that can be used for implementing scalable decoding. It needs to be at least as big as sum of header sizes and not exceed `bitstream_size`. (A decoder implementation interprets this field as per D.2.4, if the implementation uses it.)
`end_marker_plane[0]`	`0x09`	The plane stack relating to `end_marker[0]`.
`end_marker_echelon[0]`	`0x0A`	The echelon relating to `end_marker[0]`
`maximum_end_marker_index`	`0x0A.8`	Number of `end_marker` elements in secondary header. This can be `0`.
`element_interpretation`	`0x0B`	Interpretation of component values, as defined in 8.8.1. Value `0` means that the interpretation of elements is not specified.
`pipeline`	`0x0C`	Bitvector, as defined in 8.8.2.
`upsampler`	`0x0D`	Bitvector, as defined in 8.8.3 and Annex G.
`S`	`0x0E`	Shortcuts bitvector, as defined in 8.8.4 and Annex B.
`element_descriptor`	`0x10`	The two elements that represent the numbers of integer and fractional bits respectively used in resultant-planes.
`num_plane`	`0x11`	The number of plane stacks in the picture.
`annotation_data_size`	`0x12`	The size of the user-defined data, in bytes.

Field Name	Starting Offset	Description/Constraints
num_aux_terminal_tableau_catalog	0x14	The number of tableau metadata AUX catalogs for terminal-tableaux, identified by index from 0 to num_aux_terminal_tableau_catalog.
num_aux_junction_tableau_catalog	0x14.8	The number of tableau metadata AUX catalogs for junction-tableaux, identified by index from 0 to num_aux_junction_tableau_catalog.

Table 19 — Bitstream Syntax, Primary Header

Primary Header Syntax	Line
format_version = read_bits_unsigned(8)	1
bitstream_size = read_bits_unsigned(32)	2
end_markers[0] = read_bits_unsigned(32)	3
end_marker_plane[0] = read_bits_unsigned(8)	4
end_marker_echelon[0] = read_bits_unsigned(4)	5
maximum_end_marker_index = read_bits_unsigned(4)	6
element_interpretation = read_bits_unsigned(8)	7
for (i = 1; i < 9; i++)	8
pipeline[i] = read_bits_unsigned(1)	9
upsampler = read_bits_unsigned(8)	10
for (i = 9; i < 17; i++)	11
S[i] = read_bits_unsigned(1)	12
for (i = 1; i < 9; i++)	13
S[i] = read_bits_unsigned(1)	14
element_descriptor[0] = read_bits_unsigned(5)	15
element_descriptor[1] = read_bits_unsigned(3)	16
num_plane = read_bits_unsigned(8)	17
annotation_data_size = read_bits_unsigned(16)	18
num_aux_terminal_tableau_catalog = read_bits_unsigned(4)	19
num_aux_junction_tableau_catalog = read_bits_unsigned(4)	20

8.5 Secondary Header

The secondary header contains configuration data whose size and/or number of occurrences is determined by data in the primary header. For example, the secondary header includes 1D arrays of integers specifying the width and height of each plane stack's resultant-plane, where num_plane from the primary header is the variable 1D array size used for these arrays in the secondary header.

The secondary header and its bitstream syntax are specified in Table 20 and Table 21, respectively.

Table 20 — Secondary Header

Field Name	Description/Constraints
`aux_catalog_item_size`	The size of each tableau metadata AUX catalog item in bits.
`nums_catalog`	The number of items in each tableau metadata catalog.
`num_terminal_tableau_set`	The number of terminal-tableau-sets.
`num_junction_tableau_set`	The number of junction-tableau-sets.
`widths`	The width of each plane stack's resultant-plane at echelon `0`.
`heights`	The height of each plane stack's resultant-plane at echelon `0`.
`end_markers`	Tail of an array whose elements need to be at least as big as sum of header sizes and not exceed `bitstream_size`; Its head, `end_markers[0]`, is in primary header.
`end_marker_plane`	Tail of an array; Its head, `end_marker_plane[0]`, is in primary header.
`end_marker_echelon`	Tail of an array; Its head, `end_marker_echelon[0]`, is in primary header.
`num_echelons`	The number of echelons of each plane stack.
`padbits_secheader`	Arbitrary bits directly preceding the annotation data, if present, or directly preceding the tertiary header.
`annotation_data`	User data.

Table 21 — Bitstream Syntax, Secondary Header

Secondary Header Syntax	Line
Li = num_aux_terminal_tableau_catalog + num_aux_junction_tableau_catalog	21
for (i = 0; i < Li; i++)	22
`      `**aux_catalog_item_size**[i] = read_bits_unsigned(16)	23
Li = 0	24
if (S[1] == 0)	25
`      `Li = Li + 3 – S[3] – S[4] + num_aux_terminal_tableau_catalog	26
if (S[6] == 0)	27
`      `Li = Li + 1 – S[1] – S[4] + num_aux_junction_tableau_catalog	28
Lmain = Li – (1 – S[1]) * num_aux_terminal_tableau_catalog – (1 – S[6]) * num_aux_junction_tableau_catalog	29
max_num_catalog = 0	30
for (i = 0; i < Li; i++)	31
`      `if ((i < Lmain) || (aux_catalog_item_size[i – Lmain] > 0))	32
`            `**nums_catalog**[i] = read_bits_unsigned(16)	33
`            `if (nums_catalog[i] > max_num_catalog)	34
`                  `max_num_catalog = nums_catalog[i]	35
if (S[2] == 0)	36
`      `**num_terminal_tableau_set** = read_bits_unsigned(16)	37
if (S[7] == 0)	38
`      `**num_junction_tableau_set** = read_bits_unsigned(16)	39
for (i = 0; i < num_plane; i++)	40
`      `**widths**[i] = read_bits_unsigned(16)	41
for (i = 0; i < num_plane; i++)	42
`      `**heights**[i] = read_bits_unsigned(16)	43
Li = maximum_end_marker_index + 1	44
for (i = 1; i < Li; i++)	45
`      `**end_markers**[i] = read_bits_unsigned(32)	46
for (i = 1; i < Li; i++)	47
`      `**end_marker_plane**[i] = read_bits_unsigned(8)	48
for (i = 1; i < Li; i++)	49
`      `**end_marker_echelon**[i] = read_bits_unsigned(4)	50

Secondary Header Syntax	Line
echelons_max = 0	51
for (i = 0; i < num_plane; i++)	52
num_echelons[i] = read_bits_unsigned(4)	53
if (num_echelons[i] > echelons_max)	54
echelons_max = num_echelons[i]	55
/* Integer num_padbits specifies the smallest number of bits required to ensure that the size of the secondary header is a multiple of 8 bits. */	
padbits_secheader = read_bits(num_padbits)	56
for (i = 0; i < annotation_data_size; i++)	57
annotation_data[i] = read_bits_unsigned(8)	58

8.6 Tertiary Header

The tertiary header consists of arrays whose sizes are dependent upon information from the first two headers, such as arrays including, at minimum, the RMFs that are required for decoding the remainder of the bitstream.

The tertiary header and its bitstream syntax are specified in Table 22 and Table 23, respectively. Structured-type functions listed in 8.3.3 are referenced by name in the syntax of the tertiary header. RMT values in histograms are as specified in Annex E.

Table 22 — Tertiary Header

Field Name	Description/Constraints
residual_grid_flags	Each element is a bit where value 1 indicates that a residual-grid contains resid-vals represented in the bitstream payload using an s-tree, and value 0 indicates that the residual-grid will contain only resid-vals of 0.
residual_rmf_catalog	Set of RMFs of resid-vals from which all terminal-tableau-sets draw one of their RMFs.
grandchild_factor_rmf_catalog	Set of RMFs of grandchild factors from which terminal-tableau-sets and junction-tableau-sets can draw one of their RMFs.
quantization_parameters_tuple_catalog	Set of parameters from which terminal-tableau-sets can draw one of their quantization parameters tuples.
streamlength_difference_rmf_catalog	Set of RMFs of streamlength-differences from which all junction-tableau-sets draw one of their RMFs.
aux_catalogs	List of additional sets of integers from each of which a terminal-tableau-set or junction-tableau-set (but not both) can draw one of its tableau metadata AUX catalog items.
terminal_tableau_sets	Tuples of catalog item indices.
rmf_terminal_tableau_sets	An RMF of index-differences of all terminal-tableau-sets.
junction_tableau_sets	Tuples of catalog item indices.
root_junction_tableau_set_indices	Non-negative integers which specify the junction-tableau-set to be used for retrieving each root-tableau.

Field Name	Description/Constraints
rmf_junction_tableau_sets	An RMF of index-differences of all junction-tableau-sets.
streams_key	Specification of a permutation of plane stacks, echelons, directions and tiers in a bitstream payload, where the significance of each ranges from integers 0 (lowest) to 3 (highest), in any order.
root_junction_tableau_streamoffsets	The offset from the end of this header to the root-tableau of each s-tree, in bytes.
root_junction_tableau_trailing_spaces	The number of bytes that is added to the number of range encoded bytes in a root tableau to calculate a streamoffset from the start of that tableau root.
padbits_terheader	Arbitrary bits directly preceding the position corresponding to a root_streamoffsets element of 0.

Table 23 — Bitstream Syntax, Tertiary Header

Tertiary Header Syntax	Line
num_live_grids = 0	59
num_live_junction_tiers = 0	60
for (i = echelons_max – 1; i > – 1; i––)	61
for (j = 0; j < num_plane; j++)	62
if (j < num_echelons[i])	63
for (k = 0; k < 4; k++)	64
residual_grid_flags[i][j][k] = read_bits_unsigned(1)	65
if (residual_grid_flags[i][j][k] == 1)	66
num_live_grids = num_live_grids + 1	67
num_live_junction_tiers = num_live_junction_tiers + 1	68
width_echelon = widths[j]	69
height_echelon = heights[j]	70
for (m = 0; m < (i + 1); m++)	71
width_echelon = (width_echelon + 1) // 2	72
height_echelon = (height_echelon + 1) // 2	73
if ((width_echelon > 256) || (height_echelon > 256))	74
num_live_junction_tiers = num_live_junction_tiers + 1	75
if ((width_echelon > 4096) || (height_echelon > 4096))	76
num_live_junction_tiers = num_live_junction_tiers + 1	77

Tertiary Header Syntax	Line
Li = 0	78
if ((S[1] == 0) && (S[2] == 0))	79
Li = Li + nums_catalog[0]	80
if (S[2] == 0)	81
Li = Li + num_terminal_tableau_set	82
elseif (S[1] == 1)	83
Li = Li + num_live_grids	84
for (i = 0; i < Li; i++)	85
residual_rmf_catalog[i] = residual_rmf_t()	86
Li = S[1] * S[2] * num_live_grids + S[6] * S[7] * num_live_junction_tiers	87
if (S[4] == 0)	88
Li = Li + (1 – S[1] * S[6]) * nums_catalog[1 – S[1]]	89
if ((S[1] == 1) && (S[2] == 0))	90
Li = Li + num_terminal_tableau_set	91
if ((S[6] == 1) && (S[7] == 0))	92
Li = Li + num_junction_tableau_set	93
for (i = 0; i < Li; i++)	94
grandchild_factor_rmf_catalog[i] = grandchild_factor_rmf_t()	95
Li = 0	96
if (S[3] == 0)	97
Li = Li + S[1] * S[2] * num_live_grids	98
Li = Li + S[1] * (1 – S[2]) * num_terminal_tableau_set	99
Li = Li + (1 – S[1]) * nums_catalog[2 – S[1] – S[4]]	100
for (i = 0; i < Li; i++)	101
quantization_parameters_tuple_catalog[i] = quantization_parameters_tuple_t()	102
Li = (1 – S[6]) * nums_catalog[3 – S[1] – S[4] – S[3]]	103
Li = Li + S[6] * (1 – S[7]) * num_junction_tableau_set	104
Li = Li + S[6] * S[7] * num_live_junction_tiers	105
for (i = 0; i < Li; i++)	106
streamlength_difference_rmf_catalog[i] = streamlength_difference_rmf_t()	107

Tertiary Header Syntax	Line
Li = num_aux_terminal_tableau_catalog + num_aux_junction_tableau_catalog	108
for (i = 0; i < Li; i++)	109
Lj = 0	110
if (((S[1] == 1) && (S[2] == 0)) || ((S[6] == 1) && (S[7] == 0)))	111
if (i < num_aux_terminal_tableau_catalog)	112
Lj = Lj + num_terminal_tableau_set	113
else	114
Lj = Lj + num_junction_tableau_set	115
if ((S[1] == 0) || (S[6] == 0))	116
Lj = Lj + nums_catalog[4 – S[1] – S[4] – S[3] + i]	117
if (i < num_aux_terminal_tableau_catalog)	118
if (S[2] = 1)	119
Lj = Lj + num_live_grids	120
else	121
if (S[7] = 1)	122
Lj = Lj + num_live_junction_tiers	123
for (j = 0; j < Lj; j++)	124
if (aux_catalog_item_size[i] > 0)	125
aux_catalogs[i][j] = read_bits_unsigned(aux_catalog_item_size[i])	126
if (S[1] == 0)	127
Li = 3 – S[4] – S[3] + num_aux_terminal_tableau_catalog	128
Lmain = Li – num_aux_terminal_tableau_catalog	129
if (S2 == 0)	130
Lj = num_terminal_tableau_set	131
else	132
Lj = num_live_grids	133
for (i = 0; i < Li; i++)	134
if ((i < Lmain) || (aux_catalog_item_size[i – Lmain] > 0))	135
for (j = 0; j < Lj; j++)	136
terminal_tableau_sets[i][j] = catalog_item_index_t()	137
if (S[2] == 0)	138
rmf_terminal_tableau_sets = terminal_tableau_set_index_line_segments_rmf15_t()	139

Tertiary Header Syntax	Line
if (S[6] == 0)	140
Li = 2 – S[4] + num_aux_junction_tableau_catalog	141
Lmain = Li – num_aux_junction_tableau_catalog	142
if (S[7] == 0)	143
Lj = num_junction_tableau_set	144
else	145
Lj = num_live_junction_tiers	146
for (i = 0; i < Li; i++)	147
if ((i < Lmain) || (aux_catalog_item_size[i – Lmain + num_aux_terminal_tableau_catalog] > 0))	148
for (j = 0; j < Lj; j++)	149
junction_tableau_sets[i][j] = catalog_item_index_t()	150
if (S[7] == 0)	151
for (i = 0; i < num_live_grids; i++)	152
root_junction_tableau_set_indices[i] = junction_tableau_set_index_t()	153
if (S[7] == 0)	154
rmf_junction_tableau_sets = junction_tableau_set_index_line_segments_rmf15_t()	155
streams_key = permutation_descriptor_t()	156
for (i = 0; i < num_live_grids; i++)	157
root_junction_tableau_streamoffsets[i] = root_junction_tableau_streamoffsets_t()	158
for (i = 0; i < num_live_grids; i++)	159
root_junction_tableau_trailing_spaces[i] = streamlength_t()	160
/* Integer num_padbits specifies the smallest number of bits required to ensure that the size of the tertiary header is a multiple of 8 bits */	
padbits_terheader = read_bits(num_padbits)	161
/* Mini-trees follow after this tertiary header (Payload) */	

8.7 Payload

8.7.1 Mini-trees in Payload

Mini-trees are in the payload. The start positions of mini-trees in the payload are specified by either:

- The start position of the mini-tree in the root tier of an s-tree is specified in the bitstream header.
- The start position of a mini-tree, in a tier other than the root tier of an s-tree, is specified by its ancestor mini-tree(s).

Mini-tree start positions are specified in D.2.2. For an illustrated example, see Figure D.3.

8.7.2 Bytes in Mini-trees in Payload

Bytes defining a mini-tree in a bitstream payload are ordered from most-significant byte to least-significant byte (big-endian).

NOTE 1 A mini-tree contains an integer number of bytes.

NOTE 2 Specification of the payload makes no use of bitstream read functions.

8.7.3 Bits in Bytes in Payload

Bits in a byte in a mini-tree in a bitstream payload are ordered from most-significant bit to least-significant bit.

8.8 Settings that Influence the Decoding Process

8.8.1 Element Interpretation

The `element_interpretation` syntax term, from the primary header, specifies the interpretation of the component values, according to their plane stack, as specified in Table 24.

Table 24 — Element Interpretation

element_interpretation Value (in Decimal)	Element Interpretation
`0`	The interpretation of elements is not specified.
`1 to 127`	Reserved
`128 to 255`	Forbidden

8.8.2 Pipeline Configuration

The pipeline bitvector, `pipeline`, from the primary header specifies the choice of composition transform to be used, from the variants specified in D.5.2. Pipeline bitvector settings and corresponding configurations are specified in Table 25.

Table 25 — Pipeline Configuration

Bit Number	pipeline bitvector Setting	Pipeline Configuration
`1`	If = `1`, then:	The integer composition transform variant is used.
`1`	If ≠ `1`, then:	The standard composition transform variant is used.
`2 to 8`	`0`	Reserved, set to `0`.

8.8.3 Upsampling Configuration

The `upsampler` syntax term, from the primary header, specifies the choice of upsampler to be used in upsampling operations specified in D.7.2. `Upsampler` values are specified in Table 26.

Table 26 — Upsampler Variants

upsampler Value (in Decimal)	Upsampler Description
0	Nearest Neighbor (see Clause G.2)
1	Sharp (see G.1.3)
2	Bicubic (see G.1.2)
3 to 31	Reserved
32	Standard Nonlinear 9-Transformer Upsamplers (see Clause G.3)
33 to 127	Reserved
128 to 255	Forbidden

8.8.4 Use of Shortcuts to Simplify the EPS Bitstream

The use of a shortcut upon the EPS transforms the encoded format of every residual-grid in the bitstream.

The bits of the shortcut bitvector (`S`) of the bitstream have the semantic meanings and syntactical constraints specified in Table 27 and the clauses therein referenced.

See Annex B for further details pertaining to shortcuts.

Table 27 — Shortcuts That Affect Secondary and Tertiary Headers

Variable	Description	Clause Reference
S[1]	terminal-tableau-set bypassing	Clause B.2
S[2]	static terminal-tableau-set allocation	Clause B.3
S[3]	quantization disabled	Clause B.4
S[4]	perfect quadtree	Clause B.5
S[5]	reserved	Clause B.6
S[6]	junction-tableau-set bypassing	Clause B.7
S[7]	static junction-tableau-set allocation	Clause B.8
S[8], S[9], …, S[16]	reserved	Clause B.9

9 Multiplanar Picture Format Structure

9.1 Multiplanar Picture Format Structure Overview

9.1.1 Multiplanar Picture Format Data Structures

The multiplanar picture format specifies a structure for a bitstream, using the hierarchy of data structures.

9.1.2 Multiplanar Picture

A multiplanar picture is expressed as a list of plane stacks within the multiplanar picture format. The dimensions of the resultant-planes within each plane stack in a multiplanar picture are configurable. The resultant-planes in a multiplanar picture can be square or rectangular.

9.1.3 Plane Stack

A plane stack contains resultant-planes at a single or multiple resolutions, for a single component. Each resultant-plane is derived from the surfaces, echelons, tiers and tableaux specifying that resultant-plane at its specific resolution. A resultant-plane is the result of one or more cycles of plane reconstruction, and provide values for one component (e.g., Y, Cb, Cr, α) of a multiplanar picture. Plane reconstruction is described in Annex A.

9.1.4 Echelon

A resultant-plane can depend upon a resultant-plane from an earlier plane reconstruction cycle and depends on payload data. The payload data relevant to a specific plane reconstruction cycle consists of 4 elements known as echelons.

9.1.5 Quadtree

For the purposes of this document and standard, a quadtree shall have the following properties:

- Each node contains either 0 or 4 children.
- The children of a node are ordered.

9.1.6 S-tree

The data structure that comprises the echelon is a quadtree with specific properties, and provides data, in the form of a grid, for one cycle of plane reconstruction.

9.1.7 Tier

A tier is a 5-layer segment of an s-tree, and includes all nodes in those layers.

9.1.8 Tableau

A tableau is a 5-layer quadtree that is wholly contained within a tier.

9.1.9 Node

A tableau contains multiple node types, which are based on function and location in a tableau and in an s-tree.

9.1.10 Symbol

A node of a specific type can have one or more symbols associated with it. Symbols can be contained in a label.

9.1.11 Grid

A component value resides in a grid. A grid element is either 0, or a value mapped from a node in a tableau or an s-tree, where one or more symbols are defined at that node.

9.1.12 Perfect Counterpart

The perfect counterpart of a specified s-tree is an s-tree with the same depth as the specified s-tree.

Every node in the perfect counterpart of an s-tree has 4 children, with the exception of the nodes at altitude 0.

9.1.13 Active Volume

The active volume is a subset of the nodes of a perfect counterpart, where each node in the subset is connected by a simple path to a node within a grid of given grid width and grid height.

9.1.14 Summit

A summit is a grid derived from the perfect counterpart of an s-tree, representing the tableau-layer 0 nodes of highest tier index of the perfect counterpart of the s-tree.

9.1.15 Mini-Tree

A mini-tree is a condensed representation of a tableau in the bitstream.

9.2 Data Structure Hierarchy

The data structure is a hierarchy comprised of the following elements:

- predicted-plane
- resultant-plane
- echelon
- s-tree
- tableau
- mini-tree

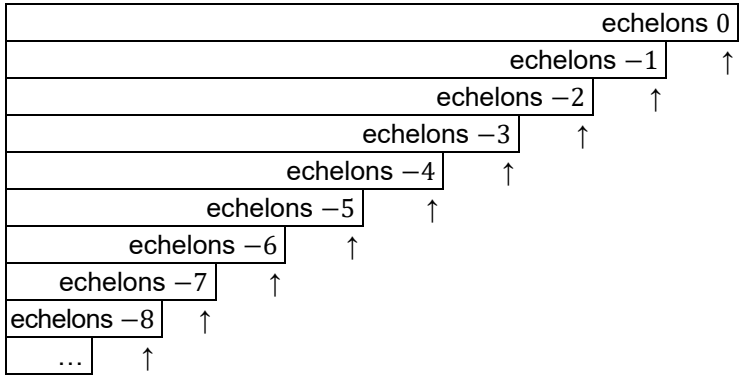
The structure hierarchy is further detailed in Table 28.

NOTE Table 28 is presented from the highest-level structure to the lowest-level structure. It is instructive, however, to read the table from the bottom up, as well.

For the purpose of brevity in Table 28:

- The left-top quadrant of a grid is referred to as Quadrant 1.
- The right-top quadrant of a grid is referred to as Quadrant 2.
- The left-bottom quadrant of a grid is referred to as Quadrant 3.
- The right-bottom quadrant of a grid is referred to as Quadrant 4.

Table 28 — Top-Down Structure Hierarchy

Structural Element(s)	Description	Resulting Element(s)
pixels	Pixels are calculated from the component values provided by resultant-planes of selected plane stacks. This row is informative, because the use of calculated component values is out of scope in this bitstream definition document.	picture comprised of pixels.
(component) x (number of plane stacks)	Component value is calculated from each plane stack. This row is informative, because pixel generation is out of scope in this bitstream definition document.	pixel generated from n component values, where n is the number of plane stacks.
component	Value of picture component.	Depends upon usage. Out of scope in this bitstream definition document. EXAMPLE Y, Cb, Cr, or α: $0 \leq \text{component} \leq 255$
plane stack	Echelons from a maximum of 15 echelon indices and 4 directions, for each echelon index, are processed. The results of processing an echelon of echelon index ech act as input to echelon $ech + 1$. Within a plane stack, an echelon of smaller echelon index has lower resolution than one of higher echelon index.	resultant-planes at desired resolutions 
resultant-plane of current echelon-index	The composed residual-grid and the predicted-plane are added together. Plane sizes are as specified in the secondary header of the bitstream.	resultant-plane

<table>
<tr><th>Structural Element(s)</th><th>Description</th><th>Resulting Element(s)</th></tr>
<tr><td>resultant-plane of previous echelon-index</td><td>The resultant-plane of the previous echelon is upsampled.</td><td>predicted-plane</td></tr>
<tr><td>composed-residual-grid of current echelon-index, one for four s-trees of directions:
A, H, V, and D</td><td>Residual-grids from A, H, V, and D directions of s-trees are dequantized and composed.</td><td><table><tr><td>quadrant 1
composed
residual-grid</td><td>quadrant 2
composed
residual-grid</td></tr><tr><td>quadrant 3
composed
residual-grid</td><td>quadrant 4
composed
residual-grid</td></tr></table></td></tr>
<tr><td>s-tree x 4, one for each direction:
A, H, V, and D</td><td>S-trees are assembled for all directions.</td><td><table><tr><td>s-tree A
quadrant 1</td><td>s-tree A
quadrant 2</td></tr><tr><td>s-tree A
quadrant 3</td><td>s-tree A
quadrant 4</td></tr></table> <table><tr><td>s-tree H
quadrant 1</td><td>s-tree H
quadrant 1</td></tr><tr><td>s-tree H
quadrant 1</td><td>s-tree H
quadrant 1</td></tr></table>
<table><tr><td>s-tree V
quadrant 1</td><td>s-tree V
quadrant 2</td></tr><tr><td>s-tree V
quadrant 4</td><td>s-tree V
quadrant 4</td></tr></table> <table><tr><td>s-tree D
quadrant 1</td><td>s-tree D
quadrant 2</td></tr><tr><td>s-tree D
quadrant 4</td><td>s-tree D
quadrant 3</td></tr></table></td></tr>
<tr><td>s-tree, single direction:
A, H , V, or D</td><td>The s-tree root node is mapped to the entire summit of the s-tree.

Each of the four child nodes of the s-tree root node is mapped to a quadrant of the summit of the s-tree in the following Z-order:
top-left quadrant
top-right quadrant
bottom-left quadrant
bottom-right quadrant
The descendant nodes of each child node map to a quadrant in the child node’s respective quadrant in Z-order. Z-order mapping continues until all nodes in the s-tree have been mapped to the summit of the s-tree.</td><td>Desparsification and entropy decoding is applied to the s-tree, resulting in a residuals-grid that contains resid-vals and additional zeroes not resulting from these processes.

resid-vals grid<table><tr><td>child/
quadrant 1</td><td>child/
quadrant 2</td></tr><tr><td>child/
quadrant 3</td><td>child/
quadrant 4</td></tr></table></td></tr>
</table>

Structural Element(s)	Description	Resulting Element(s)
tableau	The tableau root node is mapped to the entire summit of the tableau. Each of the four child nodes of the tableau root node is mapped to a quadrant of the summit of the tableau in the following Z-order: • top-left quadrant • top-right quadrant • bottom-left quadrant • bottom-right quadrant The descendant nodes of each child node map to a quadrant in the child node's respective quadrant in Z-order. Z-order mapping continues until all nodes in the tableau have been mapped to the summit of the tableau.	tableau quadrant array child/ quadrant 1 | child/ quadrant 2 child/ quadrant 3 | child/ quadrant 4
mini-tree	A mini-tree's labels are used to retrieve a single tableau. Multiple mini-trees in the bitstream retrieve the tableaux that comprise an entire s-tree.	tableau
echelon	Decoding of an echelon begins.	Desparsification and entropy decoding are applied to the payload, resulting in tableaux via mini-trees.
Bitstream	The bitstream is read.	The active volume specifications, mini-trees and other elements are parsed from the bitstream.

9.3 S-Trees, Tiers, Tableaux and Nodes

9.3.1 Tiers and Tier Types

An s-tree shall be comprised of two, three, or four tiers.

Two adjacent tiers in an s-tree have an overlap of one layer.

There shall be three types of tier, as specified in Table 29:

Table 29 — Tier Types

tier type
terminal-tier
junction-tier
root-tier, a type of junction tier

A tier shall be comprised of one or more tableaux.

With the exception of the tier that contains the root node of an s-tree, a tier can contain multiple tableaux.

9.3.2 Maximum and Minimum Numbers of Tableaux, by Tier

The maximum and minimum numbers of tableaux, by tier, shall be as specified in Table 30.

Table 30 — Number of Tableaux in a Tier

tier	Maximum number of tableaux	Minimum number of tableaux
0	16^6	1
-1	16^4	1
-2	16^2	1
-3	1	1

9.3.3 Tableau Types

There shall be three types of tableaux, as listed in Table 31.

Table 31 — Tableau Types

tableau-type
terminal-tableau
junction-tableau
root-tableau, a type of junction-tableau

9.3.4 Maximum and Minimum Numbers of Nodes, by Tableau Layer

A tableau is comprised of from 8 to 341 nodes, as specified in Table 32.

Table 32 — Number of Nodes in a Tableau-layer

tableau-layer	Maximum number of nodes	Minimum number of nodes
0	256	1
-1	64	1
-2	16	1
-3	4	4
-4	1	1
Total number of tableau nodes per tableau	**341**	**8**

9.3.5 Tableau Type by Tier

The location of the root node of the tableau shall determine its type and the tier of which the tableau is a member. The tableau root node altitude/tier member relationships for 4-tier, 3-tier and 2-tier s-trees are specified in Table 33, Table 34, and Table 35.

Table 33 — Tableau Nodes: 4-tier S-tree

tableau root node altitude	tableau type	tier index	tier type
-4	terminal	0	terminal
-8	junction	-1	junction
-12	junction	-2	junction
-16	root/junction	-3	root/junction

Table 34 — Tableau Nodes: 3-tier S-tree

tableau root node altitude	tableau type	tier index	tier type
-4	terminal	0	terminal
-8	junction	-1	junction
-12	root/junction	-2	root/junction

Table 35 — Tableau Nodes: 2-tier S-tree

tableau root node altitude	tableau type	tier index	tier type
-4	terminal	0	terminal
-8	root/junction	-1	root/junction

9.3.6 Node Types

Nodes shall be of types that are specific to tier, tableau type and tableau-layer, with characteristics specified in Table 36.

Table 36 — Node Types

node type	description	found in tableau type	tableau-layer	tier type	has/does not have children?
s-tree root node (also a tableau root node)	root node of the s-tree	root	-4	root	has
tableau root node	root node of a tableau in the s-tree	root/ junction	-4	all	has
junction node	root node of a tableau or node in tableau-layer 0 of a tableau to which a tableau is grafted	all	$0, -4$	all, except tableau-layer 0 in tier 0 (terminal tier)	has
subterminal leaf node	node is tableau-layer 0 that does not become the root of a grafted tableau	junction	0	junction/root	does not have
cargo node	node that does not carry information about its grandchild nodes	all	-1	all	has
transitional node	node that carries information about its grandchild nodes	all	$-2, -3$	all	has
simple leaf node	node that has no child nodes	all	$-1, -2, -3$	all	does not have
terminal leaf node	node at altitude 0 of the s-tree that carries information mapped to a grid as input into the plane reconstruction cycle	terminal	0	terminal	does not have
supplementary node	node that is added to an s-tree so that all nodes in the s-tree, except those at altitude 0, have 4 children.	All	all, except tableau-layers -3 and -4 in the root-tableau	all	• tableau-layer 0: does not have • all other tableau layers: has

Key:
The colors correspond to the node colors defined in the legend specified in Figure 12.

NOTE A junction node in tableau-layer 0 in a tier with tier index t acts as the tableau root node of a tableau in a tier with tier index $t + 1$.

9.3.7 Child Nodes by Tableau-layer

Depending upon the tableau-layer containing a node, the nodes shall, shall not, or may have children, as specified in Table 37.

Table 37 — Child Nodes by Tableau Layer

tableau-layer	shall/shall not/may have children?
0	• terminal-tableau: shall not • junction-tableau: may, if node in tableau-layer 0 in tier t becomes root of tableau in tier $t + 1$
-1	may
-2	may
-3	may
-4	shall

9.3.8 Symbol Types

One or more symbols may be associated with a node. Symbol types are listed in Table 38.

Table 38 — Symbol Types

symbol	description
resid-val	symbol contributing an explicit value in a residual-grid, and represented in the label of a terminal leaf node mapped to the residual-grid
streamlength-difference	the difference between two streamoffsets
index-difference	the difference between two tableau-set-indices
grandchild factor	4-bit code associated with a non-leaf node that is in tableau-layer -2, -3 or -4, that specifies the occurrences of grandchild nodes of that node

9.3.9 Symbols by Tableau-layer

Each tableau is represented in the bitstream by up to 533 symbols, one or more of which is associated with a node. The maximum number (maximum #) of symbols that can appear in a tableau-layer is specified in Table 39. Unless otherwise specified, all values apply to terminal-tableaux and junction-tableaux.

Table 39 — Number of Symbols per Tableau-layer

tableau-layer	Maximum # grandchild factor symbols	Maximum # streamlength-difference symbols	Maximum # index-difference symbols	Maximum # resid-val symbols	Maximum total # symbols per tableau-layer	
0: terminal-tableau	0	0	0	256	256	
0: junction-tableau: junction nodes	0	256	256	0	512*	Total # per tableau-layer 0: 512
0: junction-tableau: subterminal leaf nodes	0	0	256	0	256	
-1	0	0	0	0	0	
-2	16	0	0	0	16	
-3	4	0	0	0	4	
-4	1	0	0	0	1	

* Expressed as a tuple, (streamlength-difference, index-difference).

Total labels per tableau: **terminal-tableau: 277**
junction-tableau: 533

Up to 256 of the possible 277 symbols of each terminal-tableau are associated with its terminal leaf nodes. These symbols are resid-vals.

Up to 512 of the possible 533 symbols of each junction-tableau are associated with its junction nodes and subterminal leaf nodes. These symbols are streamlength-differences (for junction nodes and subterminal leaf nodes) and index-differences (for junction nodes only).

9.3.10 Tabular Representation of S-trees

9.3.10.1 Overview, Tableau-layer/Tier Overlap

Characteristics of 4-tier, 3-tier, and 2-tier s-trees are specified in Table 40, Table 41, and Table 42. Except for table header rows, shaded rows specify tier and tableau-layer overlap.

NOTE When an s-tree is assembled, some labels are discarded. Table 40, Table 41, and Table 42 specify the labels that are associated with nodes in all layers of an s-tree, as it is assembled. Once an s-tree is completely assembled via a process called grafting, then only labels at altitude 0 are retained. See 9.11 and D.3.

9.3.10.2 Tabular Representation of 4-Tier S-tree

Characteristics of a 4-tier s-tree shall be as specified in Table 40.

Table 40 — Tabular Representation of a 4-Tier S-tree

tier	altitude	tableau-layer				# of child nodes allowed per node in specified layer				node type: label (symbol present in the payload) applied to node in specified layer: resid-val sd: streamlength-difference gf: grandchild factor id: index-difference				• layer maximum dimensions: (grid width x grid height), • layer maximum # of nodes
0	0				0				0				terminal leaf: resid-val	65,536 x 65,536, 2^{32}
0	−1				−1				4				cargo: none simple leaf: none	32,768 x 32,768, 2^{30}
0	−2				−2				0 or 4				transitional: gf simple leaf: none	16,384 x 16,384, 2^{28}
0	−3				−3				0 or 4				transitional: gf simple leaf: none	8,192 x 8,192, 2^{26}
0	−4				−4				4				tableau root: gf	4,096 x 4,096, 2^{24}
−1	−4			0				0				junction: sd, id subterminal leaf: sd		4,096 x 4,096, 2^{24}
−1	−5			−1				4				cargo: none simple leaf: none		2,048 x 2,048, 2^{22}
−1	−6			−2				0 or 4				transitional: gf simple leaf: none		1,024 x 1,024, 2^{20}
−1	−7			−3				0 or 4				transitional: gf simple leaf: none		512 x 512, 2^{18}
−1	−8			−4				4				tableau root: gf		256 x 256, 2^{16}
−2	−8		0				0				junction: sd, id subterminal leaf: sd			256 x 256, 2^{16}
−2	−9		−1				4				cargo: none simple leaf: none			128 x 128, 2^{14}

tier	altitude	tableau-layer	# of child nodes allowed per node in specified layer	node type: label (symbol present in the payload) applied to node in specified layer: resid-val sd: streamlength-difference gf: grandchild factor id: index-difference	• layer maximum dimensions: (grid width x grid height), • layer maximum # of nodes
−2	−10	−2	0 or 4	transitional: gf simple leaf: none	64 x 64, 2^{12}
−2	−11	−3	0 or 4	transitional: gf simple leaf: none	32 x 32, 2^{10}
−2	−12	−4	4	tableau root: gf	16 x 16, 2^{8}
−3	−12	0	0	junction: sd, id subterminal leaf: sd	16 x 16, 2^{8}
−3	−13	−1	4	cargo: none simple leaf: none	8 x 8, 2^{6}
−3	−14	−2	0 or 4	transitional: gf simple leaf: none	4 x 4, 2^{4}
−3	−15	−3	0 or 4	transitional: gf simple leaf: none	2 x 2, 2^{2}
−3	−16	−4	4	s-tree/tableau root: gf	1 x 1, 2^{0}

9.3.10.3 Tabular Representation of 3-Tier S-tree

Characteristics of a 3-tier s-tree shall be as specified in Table 41.

Table 41 — Tabular Representation of a 3-Tier S-tree

tier	altitude	tableau-layer			# of child nodes allowed per node in specified layer			node type: label (symbol present in the payload) applied to node in specified layer: resid-val gf: grandchild factor sd: streamlength-difference id: index-difference			• layer maximum dimensions: (grid width x grid height), • layer maximum # of nodes
0	0			0			0			terminal leaf: resid-val	4,096 x 4,096, 2^{24}
0	−1			−1			4			cargo: none simple leaf: none	2,048 x 2,048, 2^{22}
0	−2			−2			0 or 4			transitional: gf simple leaf: none	1,024 x 1,024, 2^{20}
0	−3			−3			0 or 4			transitional: gf simple leaf: none	512 x 512, 2^{18}
0	−4			−4			4			tableau root: gf	256 x 256, 2^{16}
−1	−4		0			0			junction: sd, id subterminal leaf: sd		256 x 256, 2^{16}
−1	−5		−1			4			cargo: none simple leaf: none		128 x 128, 2^{14}
−1	−6		−2			0 or 4			transitional: gf simple leaf: none		64 x 64, 2^{12}
−1	−7		−3			0 or 4			transitional: gf simple leaf: none		32 x 32, 2^{10}
−1	−8		−4			4			tableau root: gf		16 x 16, 2^{8}
−2	−8	0			0			junction: sd, id subterminal leaf: sd			16 x 16, 2^{8}
−2	−9	−1			4			cargo: none simple leaf: none			8 x 8, 2^{6}
−2	−10	−2			0 or 4			transitional: gf simple leaf: none			4 x 4, 2^{4}
−2	−11	−3			0 or 4			transitional: gf simple leaf: none			2 x 2, 2^{2}
−2	−12	−4			4			s-tree/tableau root: gf			1 x 1, 2^{0}

9.3.10.4 Tabular Representation of 2-Tier S-tree

Characteristics of a 2-tier s-tree shall be as specified in Table 42.

Table 42 — Tabular Representation of a 2-Tier S-tree

tier	altitude	tableau -layer		# of child nodes allowed per node in specified layer		node type: label (symbol present in the payload) applied to node in specified layer: resid-val gf: grandchild factor	sd: streamlength-difference id: index-difference	• layer maximum dimensions: (grid width x grid height), • layer maximum # of nodes
0	0		0		0		terminal leaf: resid-val	256 x 256, 2^{16}
0	−1		−1		4		cargo: none simple leaf: none	128 x 128, 2^{14}
0	−2		−2		0 or 4		transitional: gf simple leaf: none	64 x 64, 2^{12}
0	−3		−3		0 or 4		transitional: gf simple leaf: none	32 x 32, 2^{10}
0	−4		−4		4		tableau root: gf	16 x 16, 2^{8}
−1	−4	0		0		junction: sd, id subterminal leaf: sd		16 x 16, 2^{8}
−1	−5	−1		4		cargo: none simple leaf: none		8 x 8, 2^{6}
−1	−6	−2		0 or 4		transitional: gf simple leaf: none		4 x 4, 2^{4}
−1	−7	−3		0 or 4		transitional: gf simple leaf: none		2 x 2, 2^{2}
−1	−8	−4		4		s-tree/tableau root: gf		1 x 1, 2^{0}

9.3.11 Diagrammatic Representation of an S-tree (Informative)

An s-tree and its corresponding mini-trees are depicted in diagrammatic form in Figure 11. The legend in Figure 10 applies to Figure 11

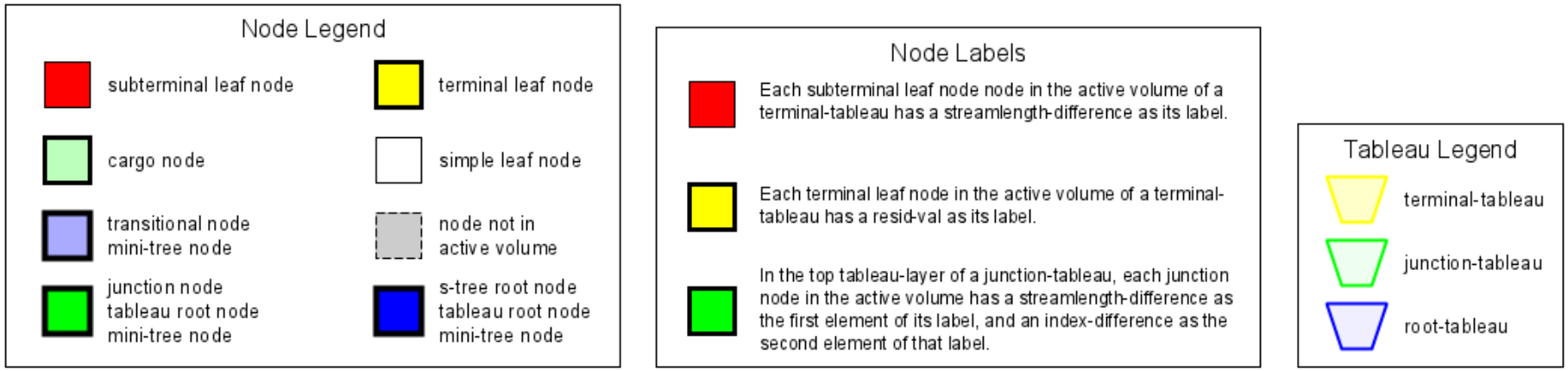


Figure 10 — Legend for Figure 11

The first, second, third and fourth child nodes of a node are shown from left to right. Mini-tree nodes and links are shown in heavy bold. See 9.13 for details about mini-tree representation of tableaux in the bitstream.

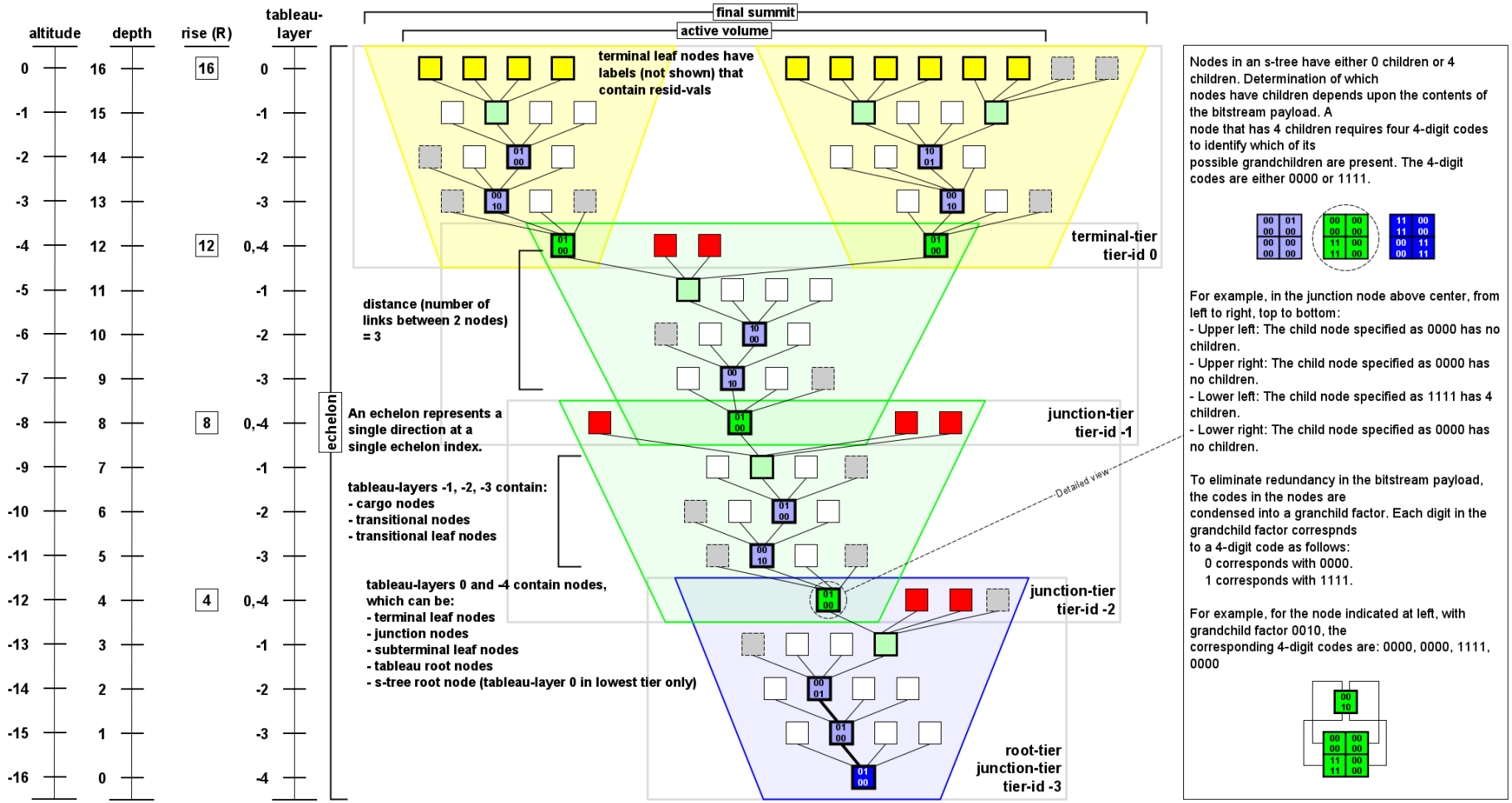


Figure 11 — Diagrammatic Representation of an S-tree

9.4 S-tree Specifications

9.4.1 Overview (Informative)

This standard is designed to exploit sparsity in data sets. Sparsity is achieved by using space-efficient s-tree data structures. The purpose of an s-tree is to provide a format with which to define a residual-grid. The residual-grid of an s-tree has sparsified areas that are represented implicitly, without transmitting encoded data. The hierarchical data structure of an s-tree contains metadata in its lower layers, as well as encoded (non-sparsified) data in it highest layer, i.e., at altitude 0.

9.4.2 S-tree Specifications

An s-tree, *s-tree*$[L, (W, H), \mathrm{labeltype}]$, shall be constrained as follows:

- L, the number of layers in the s-tree, is equal to $R + 1$, where R is the rise of the s-tree.
- R, a multiple of 4, is limited to value 8, 12 or 16, except for an s-tree of rise 4, which is a tableau.
- Dimensions tuple (W, H) specifies residual-grid width and residual-grid height, respectively, which together with L, define the active volume of the s-tree. See 9.7 for detailed information about active volume.
- W and H are each not greater than 2^R, where R is rise. 2^R by 2^R is the area of the s-tree's corresponding summit. See 9.7.1 for detailed information about summit.
- $\mathrm{labeltype}$ is associated with nodes that require labels. Labels in an s-tree can have different values, but have the same label type.

The tiers of an s-tree shall be constrained as follows:

- In the case of s-trees with residual type (see `residual_t` in Table 16) as $\mathrm{labeltype}$, tiers in s-trees of rise 8, 12 or 16 each comprise one or more tableaux, with the exception of the root tier, which comprises only one tableau.
- A single tableau, being an s-tree of rise 4, is not a tier.

The nodes in an s-tree shall be constrained as follows:

- Each layer in an s-tree contains one or more nodes that are, with the exception of the s-tree root node, each connected by a link, to its immediate ancestor node.
- Each node has either 0 or 4 child nodes.
- The four child nodes of a parent node are ordered from first child node to fourth child node.
- Nodes at altitude k of an s-tree, where $|k| \,\%\, 4 \neq 1$, and that have children, also have grandchildren.
- Nodes in the same layer of an s-tree are not connected to each other within the layer.
- Nodes that are outside of the active volume have no children.
- The nodes represent specific regions in the 2D grids that contain data that are used for plane reconstruction.

Labels in an s-tree shall be constrained as follows:

- Each node that is at the maximum altitude of an s-tree, and that is inside the active volume, has a label.
- Nodes that are inside the active volume, and that are not cargo nodes or simple leaf nodes, have a label.
- $\mathrm{labeltype}$ specifies the type(s) of symbol(s) associated with labels of nodes at the maximum altitude of an s-tree.
- It is possible to recursively dismantle an s-tree with more than 5 layers, until all the resulting s-trees are tableaux, and belong to tiers. When an s-tree is dismantled, some additional labels are provided.
- It is possible to assemble an s-tree from s-trees with fewer layers. When an s-tree is assembled, some existing labels are discarded. The final assembled s-tree includes resid-vals that contribute to the reconstruction of a resultant-plane.

9.5 Tableau Specifications

A tableau is a special case of an s-tree, which has analogous characteristics. A tableau, *tableau*$[(W, H), \mathrm{labeltype}]$, shall be constrained as follows:

- The number of tableau-layers is 5, and the tableau's rise R is 4.
- Dimensions tuple (W, H) specifies tableau grid width and tableau grid height, respectively, which define the active volume of the tableau. See 9.7 for detailed information about active volume.
- W and H are each not greater than 2^4. The area of the tableau's corresponding summit is 2^4 by 2^4. See 9.7.1 for detailed information about summit.
- $\mathrm{labeltype}$ is associated with nodes that require labels as defined in 9.12. Labels in a tableau can have different values, but can have the same label type.
- A tableau is either a junction-tableau or a terminal-tableau, depending upon its tier index in an s-tree. A terminal-tableau has labels that contain resid-vals, and is present only in a final s-tree.

A tableau-layer shall be constrained as follows:

- The root tableau-layer contains only one node, which is the tableau root node of the tableau. The root tableau-layer is specified as tableau-layer -4, and has a label with a grandchild-factor symbol.
- Three layers, specified as -3, -2, and -1, determine the sparsity of the tableau.
- Tableau-layer 0 nodes have a label specific to either a junction-tableau or a terminal-tableau as follows:
    - In a junction-tableau, all junction nodes in tableau-layer 0 have labels with streamlength-differences and index-differences. The metadata describe how to retrieve other tableaux and how sparsity affects the next higher tier of the residual-grid's s-tree.
    - In a junction-tableau, all subterminal leaf nodes in tableau-layer 0 have labels with streamlength-differences. The metadata describe the absence of tableaux in the next higher tier.
    - In a terminal-tableau, all terminal leaf nodes in tableau-layer 0, which coincides with the layer at altitude 0 of the s-tree, each carry one or more resid-vals.

The nodes in a tableau shall be constrained as follows:

- Each tableau-layer contains one or more nodes that are, with the exception of the root node of a root tableau of an s-tree, each connected by a link, to its immediate ancestor node.
- Each node has either 0 or 4 child nodes.
- The four child nodes of a parent node are ordered from first child node to fourth child node.
- Nodes in the same layer of a tableau are not connected to each other within the layer.
- Nodes that are outside of the active volume have no children.
- The nodes represent specific regions in the 2D grids that contain data that are used in plane reconstruction.

Labels in a tableau shall be constrained as follows:

- Nodes that are inside the active volume, and that are not cargo nodes or simple leaf nodes, have a label.
- labeltype specifies the type(s) of symbol(s) associated with labels of nodes at the maximum tableau-layer 0 of a tableau.

9.6 Perfect Counterpart

9.6.1 Perfect Counterpart Specification

Every s-tree has an s-tree as its perfect counterpart, where all nodes except those at altitude 0 of the perfect counterpart each have 4 children, and all nodes at altitude 0 have 0 children.

There are 4 possible perfect counterparts, based on the number of tiers in an s-tree, as defined in Table 43.

Table 43 — Perfect Counterparts to S-trees, Based on Number of Tiers

# of tiers	s-tree rise	grid dimensions, expressed as: • grid width x grid height • 2^p x 2^p • 16^p x 16^p	total # of nodes in perfect counterpart of the s-tree of specified rise (sum of nodes of all layers of the perfect counterpart)	# of links (total # of nodes in the perfect counterpart minus 1)
4	16	65,536 x 65,536 2^{16} x 2^{16} 16^4 x 16^4	22,369,621 + 2^{13} + 2^{14} + 2^{15}+ 2^{16} = 5,726,623,061	5,726,623,060
3	12	4,096 x 4,096 2^{12} x 2^{12} 16^3 x 16^3	87,381 + 2^9 + 2^{10} + 2^{11}+ 2^{12} = 22,369,621	22,369,620
2	8	256 x 256 2^8 x 2^8 16^2 x 16^2	341 + 2^5 + 2^6 + 2^7+ 2^8 = 87,381	87,380
1 (single tableau)	4	16 x 16 2^4 x 2^4 16^1 x 16^1	2^0 + 2^1 + 2^2 + 2^3+ 2^4 = 341	340

9.6.2 Perfect Counterpart Example (Informative)

The legend shown in Figure 12 applies to Figure 13, Figure 14, Figure 18, and Figure 19, as well to Table 36, Table 49, and Table 50.

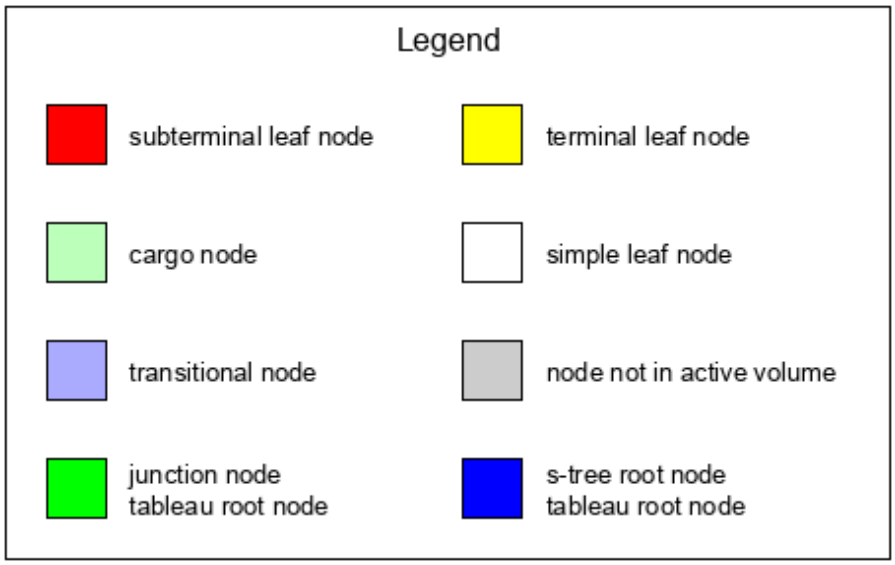


Figure 12 — Legend for Figure 13, Figure 14, Figure 18, Figure 19, Table 36, Table 49, and Table 50

Figure 13 depicts a root-tableau that might appear in an s-tree.

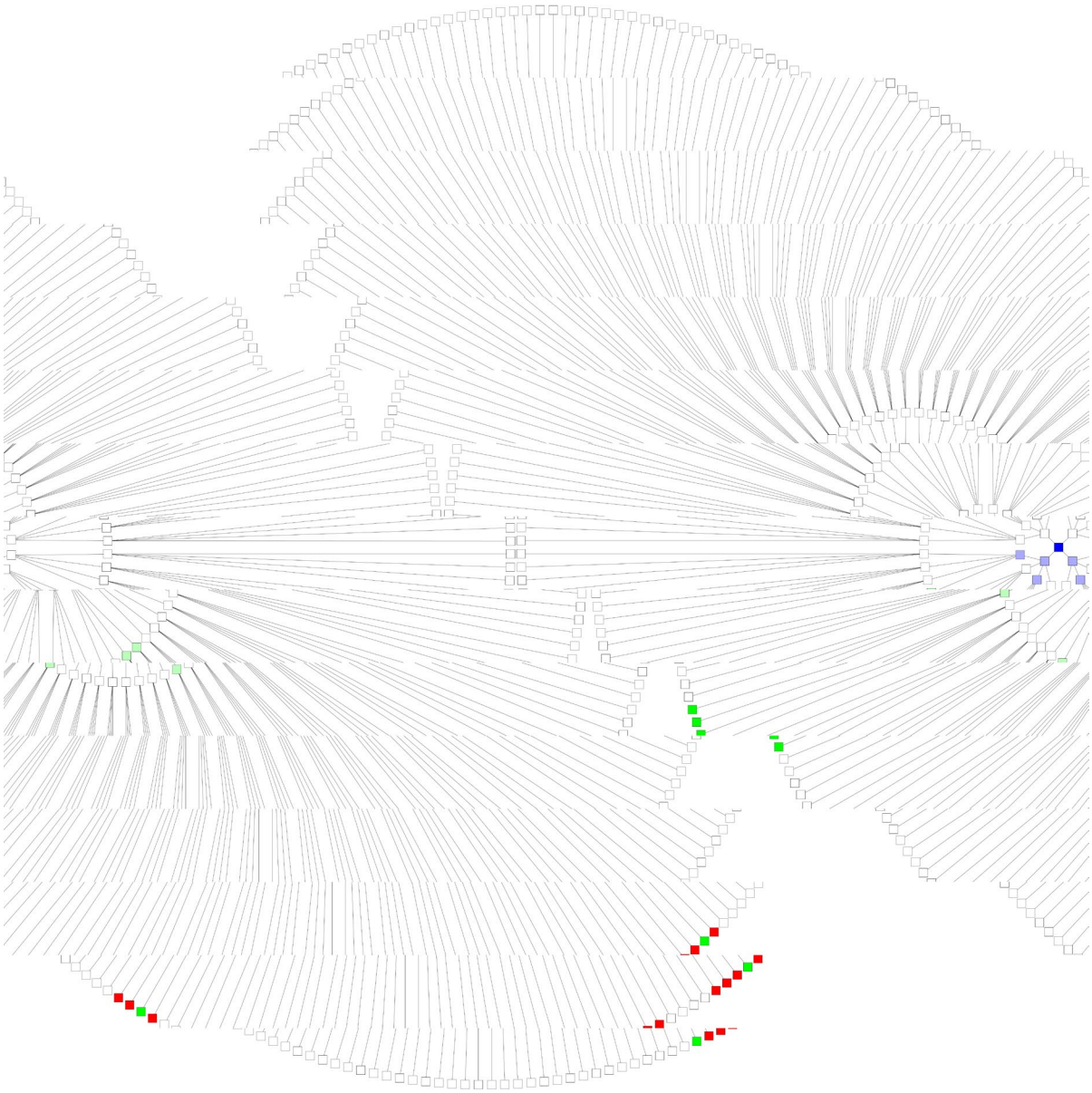


Figure 13 — Example of a Root Tableau (S-tree of rise 4)

Figure 14 shows the perfect counterpart of the tableau shown in Figure 13, including all 341 possible nodes and all 340 possible links.

The grid representing the summit of an s-tree includes all of the nodes at altitude 0 of the perfect counterpart of an s-tree. The summit, therefore, always is of dimensions 2^R by 2^R, for a given rise R of the s-tree, which in Figure 14, is 4.

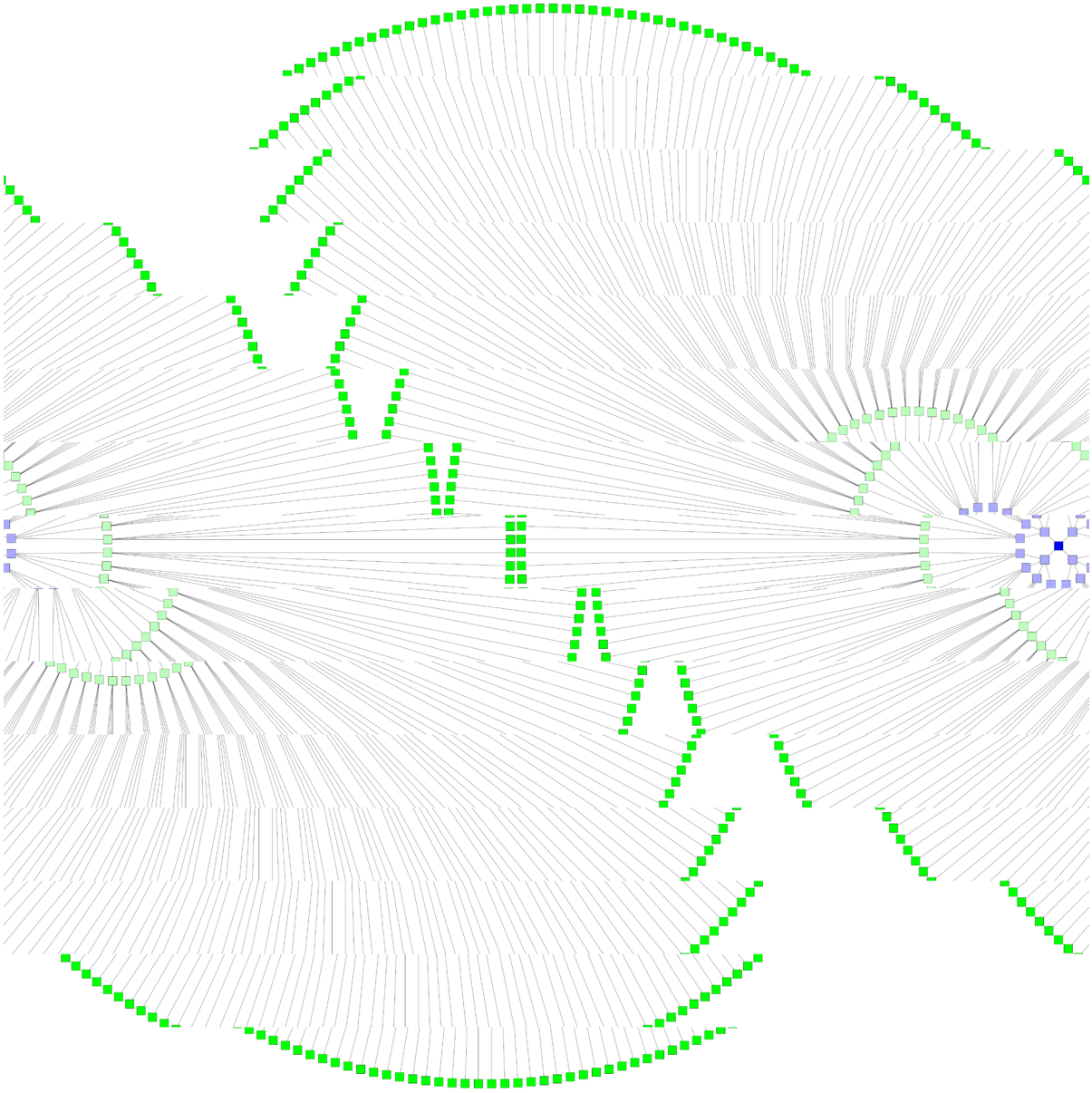


Figure 14 — Perfect counterpart of tableau in Figure 13

9.7 Summit, Active Volume, and Residual-grid

9.7.1 Summit Specifications

The summit of an s-tree represents all of the nodes at altitude 0 of the perfect counterpart of the s-tree, and is defined as a grid $(2^R, 2^R)$ of the s-tree's perfect counterpart, where R is the rise of the s-tree. Summit grid types and their corresponding dimensions are as specified in Table 44.

Table 44 — Summit Specifications

Grid Type	Description	Dimensions, # of Nodes
4-tier s-tree summit	maximum number of s-tree nodes that can be mapped to a residual-grid.	65,536 x 65,536, 2^{32}
3-tier s-tree summit		4,096 x 4,096, 2^{24}
2-tier s-tree summit		256 x 256, 2^{16}
tableau summit	maximum number of tableau nodes that can be mapped to a grid representing the tableau.	16 x 16, 2^4

9.7.2 Active Volume Specifications

The active volume shall be specified as follows:

- Active volumes of ***s-tree***$[L, (W, H), \text{labeltype}]$ and ***tableau***$[(W, H), \text{labeltype}]$ are limited by their parameters, and have members according to the following rules. L, W and H are defined in Table 14. The relationship between active volume and number of layers in an s-tree and tableau, respectively, is specified in Table 46 and Table 47.
    - The widths and heights specified in the secondary header are applied in Clause D.8, to determine the size and resolution of the residual-grid.
    - The active volume includes all nodes within a summit having grid coordinates $x \leq W$ and $y \leq H$, where x is a column index, y is a row index and (W, H) is the dimensions tuple of the grid. All other nodes at altitude 0 of the perfect counterpart are excluded.
    - The active volume includes every node in lower layers of the perfect counterpart, where each node is an ancestor of a node within the summit, where that node within the summit is included in the active volume. All other nodes in lower layers are excluded.
- The active volume of an s-tree or individual tableau is the union of a subset of the summit of the s-tree or the individual tableau, respectively, and the set of all nodes that have associated data that determine a component value.
- The set of all active volumes of the tableaux in an s-tree is equal to the active volume of the s-tree.
- The active volume of an s-tree or tableau is square or rectangular.
- A larger active volume corresponds with a higher-resolution residual-grid, and vice versa.
- Any nodes or data from any nodes of an s-tree or tableau that are located outside of the active volume are ignored when calculating any grid values.

Active volume types are specified in Table 45.

Table 45 — Active Volume Types

Active volume type	Description
s-tree populated active volume	Subset, of the s-tree summit, that contains all terminal leaf nodes and their ancestors.
s-tree unpopulated active volume	All grid elements that are mapped to the active volume of the s-tree, but that are not part of the populated active volume of the s-tree. These grid values are implicitly of value 0.
tableau populated active volume	Subset, of the tableau summit, that contains all junction nodes in tableau-layer 0 and their ancestors.
tableau unpopulated active volume.	All grid elements that are mapped to the active volume of the tableau, but that are not part of the populated active volume of the tableau. These grid values are implicitly of value 0.

Active volume characteristics for s-trees and tableaux, respectively, are specified in Table 46 and Table 47.

Table 46 — S-tree Active Volume Specifications

Grid Type	Number of s-tree layers, L	Maximum dimensions of the active volume, equal to the summit	Maximum number of nodes in active volume	Minimum dimensions of the active volume	Minimum number of nodes in the populated active volume to include altitude 0
4-tier s-tree active volume	17	65,536 x 65,536	2^{32}	4,097 x 1, or 1 x 4097	17
3-tier s-tree active volume	13	4,096 x 4,096	2^{24}	257 x 1, or 1 x 257	13
2-tier s-tree active volume	9	256 x 256	2^{16}	1 x 1	9

Table 47 — Tableau Active Volume Specifications

Grid Type	Number of tableau-layers	Maximum dimensions of the active volume, equal to the summit	Maximum number of nodes in active volume	Minimum dimensions of the active volume	Minimum number of nodes in the populated active volume to include altitude 0
tableau active volume	5	16 x 16	2^{8}	1 x 1	5

Membership in the active volume shall be a test applicable to any node of an s-tree. Since the active volume is defined in terms of a perfect counterpart:

- The s-tree need not have nodes corresponding to the entire active volume.
- There may be nodes in the s-tree that are absent from the active volume, although these nodes can have one or more sibling nodes that are included in the active volume.

9.7.3 Residual-Grid

The residual-grid of the s-tree is represented by the terminal-tier's portion of the active volume.

Each terminal leaf node in the active volume has a resid-val as its label, and is mapped to the residual-grid.

Each node in tableau-layer 0 not in the active volume does not have a label.

This residual-grid and its tableaux are contained within the same echelon.

There may be zeros that are explicitly present in the bitstream.

9.8 Sparsity and Unpopulated Active Volumes

Unpopulated active volumes are part or parts of a tableau that are absent from a bitstream payload due to sparsified areas, and are subsets of the active volume.

A residual-grid can contain sparsified areas. All resid-vals in sparsified areas are of value 0, and shall not be explicitly encoded in an s-tree. Information about the sparsified areas is contained only in lower layers of the s-tree, thus resulting in unpopulated active volumes in the s-tree.

A tableau that has one or more simple leaf nodes in tableau-layers -1, -2 and/or -3, and that are, by definition, in the active volume, contains an unpopulated active volume associated with each simple leaf node.

A subterminal leaf node in tableau-layer 0 in a tier with tier index t does not act as the root node of a tableau in a tier with tier index $t + 1$, and has no descendants, because a subterminal leaf node represents a tableau that has entirely unpopulated active volume.

There can be multiple unpopulated active volumes in an s-tree, each associated with either a simple leaf node or a subterminal leaf node. Each unpopulated active volume extends to the tier or tiers of tier index greater than the tier containing the simple leaf node or subterminal leaf node.

A tableau shall be absent from the bitstream payload if it is entirely an unpopulated active volume.

Figure 15 shows the representation of the residual-grid of an s-tree.

Figure 16 depicts an s-tree with unpopulated active volumes, and includes tableaux, some of which have unpopulated active volumes.

NOTE The gray areas in Figure 15 represent explicit, encoded data in the bitstream. The three white regions represent sparsified areas and are implicit.

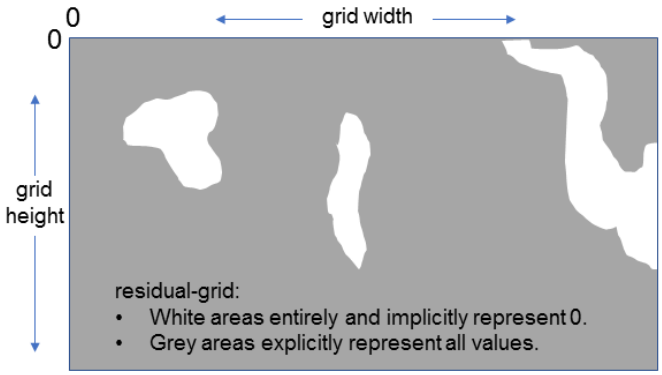


Figure 15 — Residual-grid of an s-tree

9.9 Diagrammatic Representations of Active Volume and Grids

Figure 16 depicts an s-tree comprised of four tiers, each containing one or more tableaux. In the area outside of the active volume, there is no explicit or implicit data. If this standard is used to compress a picture, then the light gray areas represent areas of a picture that are not used, due to the dimensions of the residual-grid specified in the bitstream header. The smaller the light gray area, the larger the active volume, thus the higher the resolution of the picture, and vice versa.

NOTE 1 In Figure 16, each tableau is shown in two dimensions, by not distinguishing between grid width and grid height. Tableaux of a specific tier are shown along a single axis (rather than two axes), in order to highlight key tableau features.

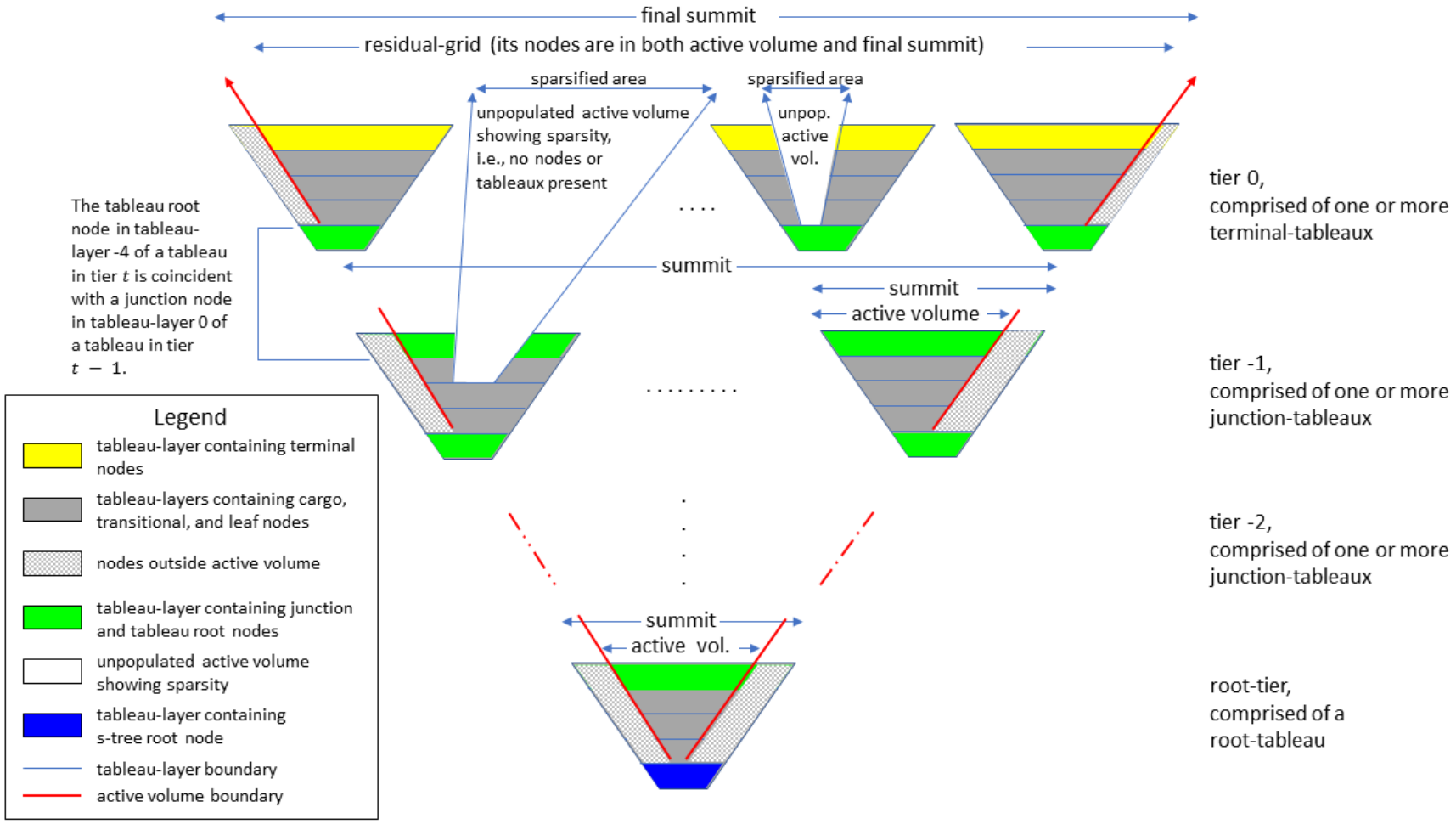


Figure 16 — Summits, grids, active volume, represented by four tiers of tableaux

Figure 17 depicts the nodes in a tableau and the corresponding 2D grid of each tableau-layer.

NOTE 2 In Figure 17, the grids of nodes on the right show, three-dimensionally, the 5 tableau-layers. The grids, reading from bottom to top, have dimensions 1 by 1, 2 by 2, 4 by 4, 8 by 8, and 16 by 16. The nodes in these grids include all of the supplementary nodes, which are not shown in the tableau-layers at the left of Figure 17.

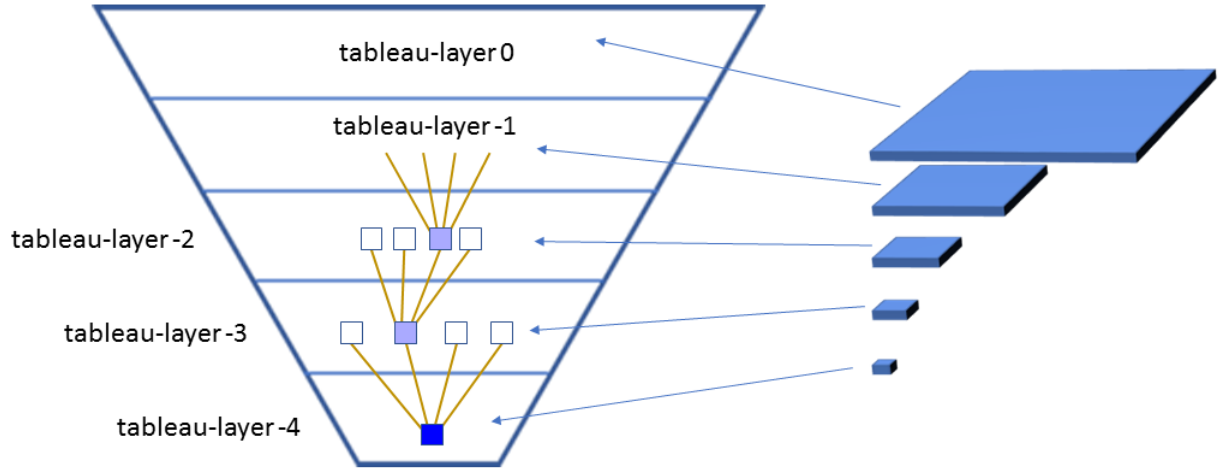


Figure 17 — Schematic showing tableau-layers of an s-tree and their corresponding grids

9.10 Mapping an S-tree Node to a Grid

The quadrant array is the ordering of quadrants, specified by mapping the first, second, third and fourth child nodes of a node, to the top-left, top-right, bottom-left and bottom-right quarters, respectively, of a square subset of the summit of an s-tree.

The four child nodes linked to a parent s-tree node shall be associated with the elements of the quadrant array. The child associated with the first quadrant array element is reached by a depth-first pre-order traversal prior to the second child, etc.

The s-tree root node shall be associated with the square grid area that is the entire summit of the s-tree. The area associated with the s-tree root node is 2^R by 2^R; therefore, the area associated with a node at layer k is $2^{|k|}$ by $2^{|k|}$.

Each node of an s-tree shall be associated with a square grid that is a subset of the summit of the s-tree. A resid-val symbol is associated with the 1 by 1 area of the summit associated with a terminal leaf node.

NOTE Layer k in the active volume, where $k \leq 0$, can be mapped to the top-left portion of a grid of maximum dimensions $2^{R-|k|}$ nodes by $2^{R-|k|}$ nodes, where R is rise, $0 \leq |k| \leq R$, and:

- The actual dimensions of the layer at altitude 0 are the s-tree dimensions tuple (W, H).
- The actual dimensions in layer $k-1$, where $0 \leq |k| < R$, are $(x_k + 1)\ //\ 2$ by $(y_k + 1)\ //\ 2$ where (x_k, y_k) represents the actual dimensions in layer k and $0 \leq |k| < R$.

EXAMPLE
Rise R of an s-tree $= 8$. The area associated with the s-tree root node is specified, as follows, in Formula (1):

$$2^8 \times 2^8 = 65{,}536 \quad (1)$$

The area of a node at layer $k = -3$ is specified, as follows, in Formula (2):

$$2^{|-3|} \times 2^{|-3|} = 64 \quad (2)$$

9.11 S-tree Assembly

9.11.1 S-tree Assembly Overview (Informative)

An s-tree can be assembled from a group of smaller s-trees. The grafting process described in 9.11.2 is one method of assembly. Other methods provably yielding the same final s-tree can be employed, but are not described in this document.

9.11.2 S-tree Grafting

For the purpose of simplicity in this clause, the following nomenclature is used to refer to s-trees, as specified in Table 48.

Table 48 — S-tree Nomenclature for 9.11.2

S-tree Nomenclature	Description
target s-tree	s-tree that is assembled from one or more iterations of grafting
primary s-tree	s-tree to which other s-trees are grafted, and which contains the root node of the target s-tree
secondary s-tree	s-tree that is to be grafted to a primary s-tree. A secondary s-tree is a tableau, and has 5 layers and a label type that depends upon the tier of the s-tree that is grafted.

A target s-tree can be assembled from one primary and one or more secondary s-trees, via a process called grafting.

Grafting shall replace a junction node in the primary s-tree with a secondary s-tree. The labels of the primary s-tree are discarded, after providing all the necessary metadata pertaining to the secondary s-trees.

When all secondary s-trees in a tier have been grafted to the primary s-tree, the result is an assembled s-tree that is the target tree.

The target s-tree can become the primary s-tree for grafting an additional tier of secondary s-trees.

The grafting process can be repeated until the all of the tableaux in all of the tiers of the s-tree have been grafted.

NOTE 1 The series of temporary “mixed” s-trees, which are no longer like the primary s-tree, and not yet like the target s-tree, have no single label type. Label types are independently applicable to all of the primary, secondary and target s-trees.

Figure 18 shows the grafting of s-trees of rise 4 (i.e., tableaux) to the root-tableau shown in Figure 13. A root node of an s-tree that is grafted to the root-tableau in tier -1 becomes the root node of a tableau in the tier 0.

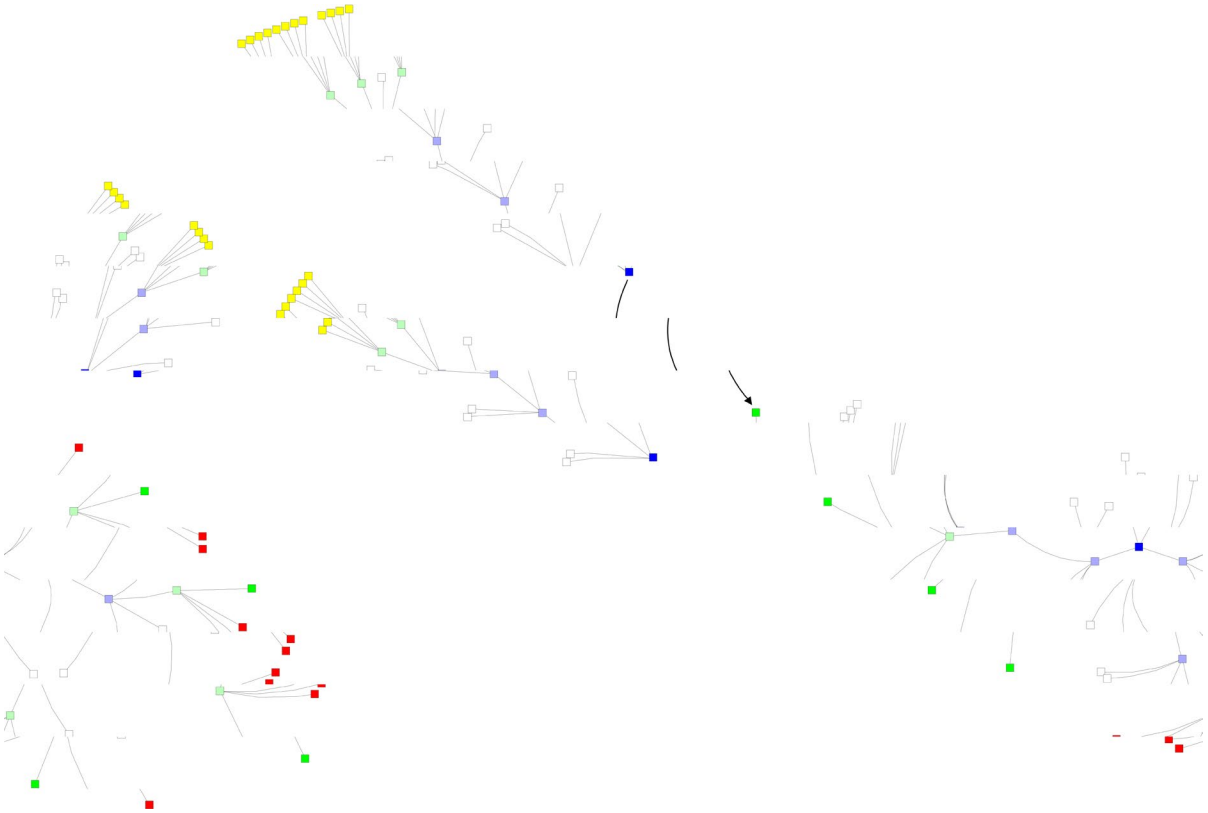


Figure 18 — Tableaux grafted to assemble the s-tree shown in Figure 19

In Figure 18, the arrow schematically represents the grafting of a tier 0 tableau to a tier -1 tableau at one of the original junction nodes. If all eight tier 0 tableaux from Figure 19 are grafted to the central tableau, then the s-tree in Figure 19 is assembled.

NOTE 2 The tableau, as well as being an abstract type, is also parametric, because it can hold altitude 0 nodes with labels of any type. This will be useful when labels on one tableau include the streamlengths of other tableaux. In Figure 18, the central tableau would encode the streamlengths of the 8 outer tableaux of Figure 19. This means that the streamlengths and other properties of tableaux in one tier are not available until the lower tiers have been decoded.

Figure 19 depicts an s-tree of rise 8, representing nine tableaux: one root-tableau and eight terminal-tableaux.

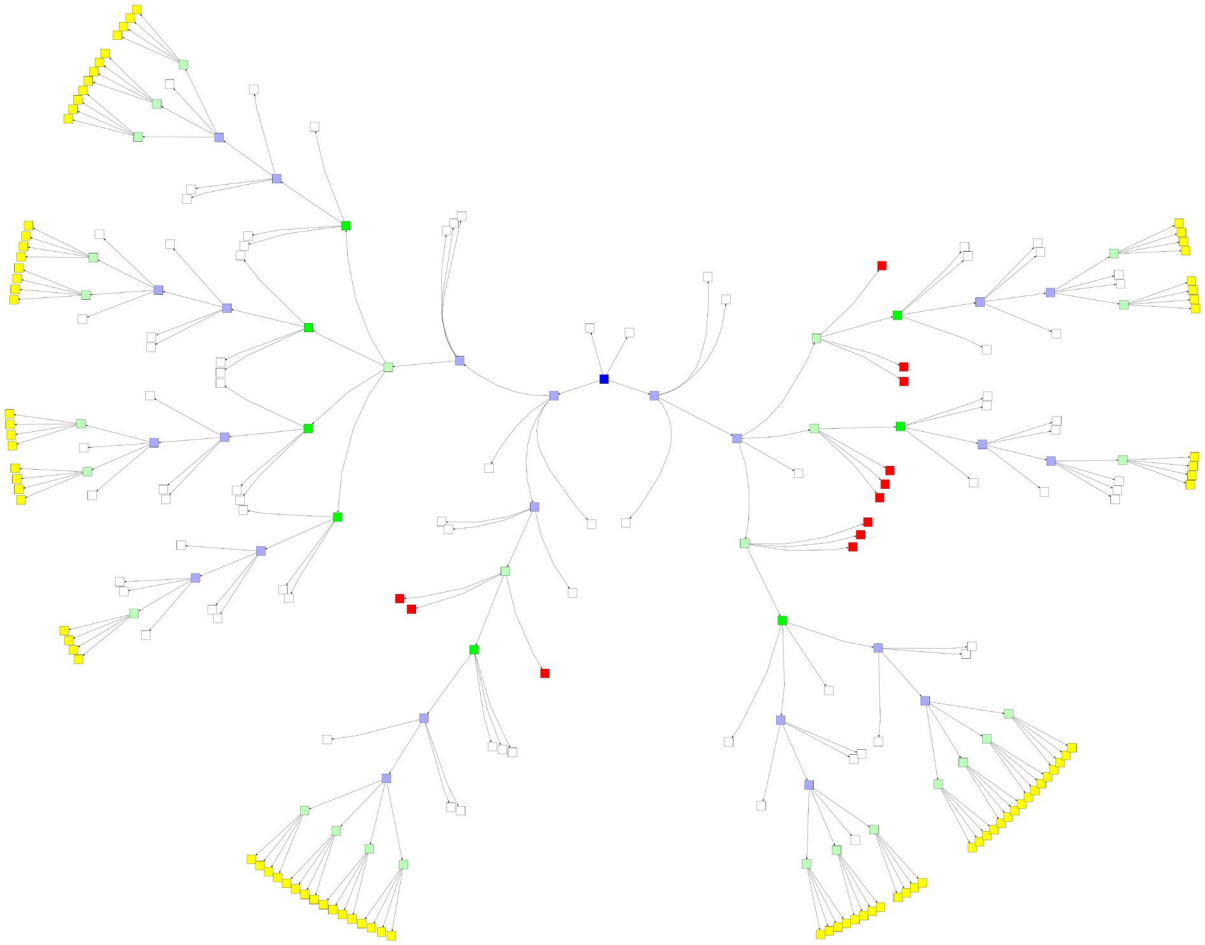


Figure 19 — S-tree of rise 8

In the s-tree shown in Figure 19:

- The number of layers, L, is 9.
- There are 88 nodes at altitude 0.
- The summit is of area 2^8 by 2^8.
- The altitude of the s-tree root node is -8.

NOTE 3 As labels are not shown in Figure 19, the diagram can represent an s-tree having any $\mathrm{labeltype}$. As active volume is also not shown, the diagram can represent an s-tree having dimensions tuple of $(256, 256)$ or smaller, depending upon which nodes at altitude 0 lack labels in the actual s-tree. By not defining sibling node order, the diagram does not distinguish between s-trees with topologies that differ only due to node order.

EXAMPLE
A single, full-resolution, non-sub-sampled resultant-plane of size $1{,}080$ by $2{,}650$, requires a larger s-tree, of 13 layers, for each 540 by $1{,}325$ direction (i.e., A, D, H, V). In the payload of a bitstream, the streams associated with such an s-tree have one root-tableau in root-tier -2, up to 18 junction-tableaux in junction-tableaux-tier -1 and up to 2,822 terminal-tableaux in terminal-tier 0.

9.12 Labels in Tableaux and Mini-trees

9.12.1 Label Overview

S-trees and tableaux are abstract data structures in which a node can have a label.

Labels help decoding both the structure of an s-tree and the data values represented by the s-tree.

The type or representation of labels is generally not specified by the abstract data structure that employs the labels.

Labels shall contain symbols.

9.12.2 Junction-tableau Labels

Junction-tableaux labels shall specify the occurrence and positioning of streams in the bitstream that carry further junction-tableaux or terminal-tableaux. In junction-tableaux, the following shall apply:

- Every tableau-layer 0 junction node label is a tuple of streamlength-difference and index-difference
- Every subterminal leaf node label is a streamlength-difference.
- No other nodes in the tableau have a label once the tableau is complete.

9.12.3 Terminal-tableau Labels

Terminal-tableaux labels shall specify resid-vals at altitude 0 of a fully assembled s-tree. In terminal-tableaux, the following shall apply:

- Every terminal leaf node label is a resid-val.
- No other nodes in the tableau have a label once the tableau is complete.

9.12.4 Mini-tree Labels

A mini-tree is the representation of a tableau in the bitstream, and may be:

- Derived from any tableau; and,
- Read from the bitstream to retrieve a tableau.

A mini-tree shall have the following characteristics:

- A mini-tree has fewer nodes and links than a tableau, and has labels on every node.
- Mini-tree labels contain a grandchild factor associated with a node of the tableau, followed, in some cases, by values of one or more labels from the tableau-layer 0 of the tableau.
- Grandchild factors can appear only in the label of a mini-tree node, and not in the label of an s-tree node.
- Each node of a mini-tree has a label that includes a grandchild factor, and for a node that is in mini-tree layer 0, can also include values, as specified in Table 49. Mini-tree nodes in layers -1 and -2 each have a label consisting only of a grandchild factor.

Table 49 — Mini-tree Layer 0 Label(s) by Associated Tableau-layer 0 Nodes

number of nodes from a tableau that are associated with a single layer 0 node of the corresponding mini-tree		first part of label in layer 0 of mini-tree	tableau type	type of tableau-layer 0 node	next part(s) of label in layer 0 of mini-tree
cargo nodes	tableau-layer 0 nodes				
1, 2, 3, or 4	4, 8, 12, or 16	grandchild-factor	terminal	terminal leaf	resid-val
			junction	junction	tuple, containing a streamlength-difference and an index-difference
				subterminal leaf	streamlength-difference
			junction, terminal	not in active volume	none

Key:
The colors correspond to the node colors defined in the legend specified in Figure 12.

9.13 Representing a Tableau as a Mini-tree

9.13.1 Overview (Informative)

The representation of a tableau by a mini-tree reduces the amount of data required to carry information necessary for decoding, and illustrates the efficiency of this standard.

9.13.2 Tableau/Mini-tree Relationship

The building blocks of an s-tree consist of several mini-trees, each of which is a condensed tableau, and each of which is represented in the bitstream by a depth-first pre-order traversal of the labels of its corresponding mini-tree.

The following shall apply to a mini-tree and its nodes:

- Mini-trees have 3 layers, whereas s-tree tableaux have 5 layers.
- Nodes that are outside of the active volume, though they appear in tableaux, are not parts of the mini-trees specified in the bitstream.
- The mini-tree has a root node.
- Every node of a mini-tree has a label.
- The mini-tree has the transitional nodes of the tableau that are to be decoded from the bitstream, as the descendants of the root node. It is implicit when starting to decode a mini-tree from the bitstream, that the root node of the mini-tree exists.
- The mini-tree root node has at least one child node, and the number of children is the number of bits of value 1 in the grandchild factor of the mini-tree root node, not counting any bits of value 1 associated with nodes not in the active volume of the tableau that is to be decoded from the bitstream.
- The first part of the label of a grandchild node of the mini-tree root node is the grandchild factor of the grandchild node of the root node of the tableau that is to be decoded from the bitstream.
- The second part of the label of a grandchild node of the mini-tree root node is:
    - If the tableau that is to be decoded from the bitstream is a junction-tableau, then the streamlength-differences of the junction nodes and the subterminal leaf nodes of the tableau.
    - If the tableau that is to be decoded from the bitstream is a terminal-tableau, then resid-vals of the terminal leaf nodes of the tableau.
- The label of a child node of the mini-tree root node is the grandchild factor of the child node of the root node of the tableau that is to be decoded from the bitstream.
- The transitional nodes and the mini-tree root node each display grandchild factors that are 4-bit codes, and that determine if the children of the transitional nodes of the tableau that is to be decoded from the bitstream will have children.
- The mini-tree leaf nodes each have a label that contains a grandchild factor, followed by up to 16 $\mathrm{labeltype}$-type values, depending upon the node's specific 4-bit code and upon the active volume of the tableau that is to be decoded from the bitstream. The order of the letters and dashes (-) reflect the order of the tableau-layer 0 nodes, in the tableau that is to be decoded from the bitstream, from depth-first pre-order traversal.

Labels appearing in depth-first pre-order traversal order in the bitstream allow the location, and number, of nodes to be read. The number of children (i.e., nodes in layer 0 or layer -1 of a mini-tree) of a parent node (i.e., root node or transitional node in layer -1 of a mini-tree) is specified by the label of the parent node and by the active volume of the tableau that is to be decoded from the bitstream.

The label of a grandchild node of the root node of a mini-tree is a concatenation of two elements, as shown in Figure C.3.

- The first element of this concatenation is a grandchild factor.
- The second element of the label is a list of up to 16 values. This list consists of copies of the values of all labels associated, in the tableau to be decoded from the bitstream, with grandchildren of the corresponding tableau-layer -2 transitional node of the original tableau.
- The number of values in the second element of a label of a mini-tree leaf node in tableau-layer 0 of a mini-tree is specified by the first element of the label and by the active volume of the tableau.

The correspondence between the mini-tree layers and the s-tree layers shall be as specified in Table 50.

Table 50 — Correspondence between Tableau and Mini-Tree Layers

s-tree tableau-layer	tableau node type		symbol type(s) carried by tableau node label	mini-tree layer	mini-tree node type	how represented in label in mini-tree	symbol type(s) carried by mini-tree node label
0	terminal-tableau only	terminal leaf	resid-val	0	-	represented by a label containing a resid-val carried by its mini-tree layer 0 ancestor node	-
	junction-tableau only	junction	streamlength-difference, index-difference		-	represented by a label containing a (streamlength-difference, index difference) tuple carried by its mini-tree layer 0 ancestor node	-
		subterminal leaf	streamlength-difference		-	represented by a label containing a streamlength-difference carried by its mini-tree layer 0 ancestor node	-
−1	cargo		none		-	represented by a label containing a grandchild factor bit of value 1, carried by its mini-tree layer 0 ancestor node	-
	simple leaf		none		-	represented implicitly by a label containing a grandchild factor bit of value 0, carried by its mini-tree layer 0 ancestor node	-
−2	transitional		grandchild factor		mini-tree leaf	also represented by a label containing a grandchild factor bit of value 1, carried by its mini-tree layer −1 ancestor node	grandchild factor streamlength-difference index-difference
	simple leaf		none		-	represented implicitly by a label containing a grandchild factor bit of value 0, carried by its mini-tree layer −1 ancestor node	-

s-tree tableau-layer	tableau node type	symbol type(s) carried by tableau node label	mini-tree layer	mini-tree node type	how represented in label in mini-tree	symbol type(s) carried by mini-tree node label
-3	transitional	grandchild factor	-1	transitional	also represented by a label containing a grandchild factor bit of value 1, carried by the mini-tree root node	grandchild factor
	simple leaf	none		-	represented implicitly by a label containing a grandchild factor bit of value 0, carried by the mini-tree root node	-
-4	root	grandchild factor	-2	root	not applicable	grandchild factor

Key:
The colors in this table correspond to the node colors defined in the legend specified in Figure 12.

The maximum and minimum number of nodes in a mini-tree shall be as specified in Table 51.

Table 51 — Number of Nodes per Mini-tree Layer

mini-tree layer	Maximum	Minimum
0	16	1
-1	4	1
-2	1	1
Total # of nodes	21	3

Annex C describes an example of how tableau and mini-tree nodes and labels correspond to one another.

Annex A
(Normative)

Plane Reconstruction Cycles Overview

A resultant-plane (see D.7.3) may be created at each echelon-index, via a recursive process called plane reconstruction. The plane reconstruction process encompasses a series of cyclical processes, called plane reconstruction cycles. Within each plane reconstruction cycle, a resultant-plane shall be reconstructed using the resultant-plane immediately preceding echelon index, where

- The resultant-plane of a higher echelon index is of higher resolution than that of a lower echelon index.
- The decoding process can omit the decoding of echelons of an echelon index higher than a user-defined threshold.

To reconstruct a resultant-plane at a given resolution, the plane reconstruction cycle is executed. To initiate the process, the core-echelons, i.e., the echelons for all four directions at echelon index $1 - m$, where m is the number of echelons, shall be decoded first.

The resultant-plane for the successor of echelon index n, i.e., echelon index $n + 1$, is reconstructed:

- From the result of applying composition to the additional data in residual-grids decoded from the bitstream, which can recover the composed residual-grid from the four residual-grids (A, H, V, and D) of their respective echelons, and,
- Where applicable, from the result of an upsampler, acting on the resultant-plane of echelon index n, i.e., the predicted-plane.

The upsampler is selected by the bitstream header.

A resid-val is an input to the process of dequantization, followed by composition, yielding difference values between an intermediate, predicted-plane, and a subsequent resultant-plane. For a core-echelon, the predicted-plane contains only zeroes.

The plane reconstruction cycles at the echelon index of the core-echelon and at echelon index 0 shall differ from the plane reconstruction cycles at other echelon indices as follows:

- For an echelon index ech that is not a core-echelon, the predicted-plane is upsampled from the resultant-plane of echelon index $(ech - 1)$.
- The initial plane reconstruction cycle, which involves the core-echelons, does not add a predicted-plane.
- A plane reconstruction cycle with the resultant-plane of echelon index 0 as its output does not have following plane reconstruction cycles.

Figure A.1 shows four plane reconstruction cycles for the core-echelons' echelon index, and three higher, successor echelon indices -2, -1, and 0, respectively. An echelon is present in the bitstream in the form of tiers of compact data, from which a correction called a residual-grid is calculated. Resultant-planes of higher echelons are predicted from resultant-planes of lower echelons, and are then corrected using residual-grids.

In the three higher cycles shown in Figure A.1, resid-vals are subject to dequantization and composition, prior to adding a predicted-plane, to result in a resultant-plane. The predicted-plane passes the resultant-plane from the previous echelon to an upsampler. The use of tiers of junction-tableaux and terminal-tableaux to decode residual-grids is not explicitly shown, but is represented by the desparsification and entropy decoding processes.

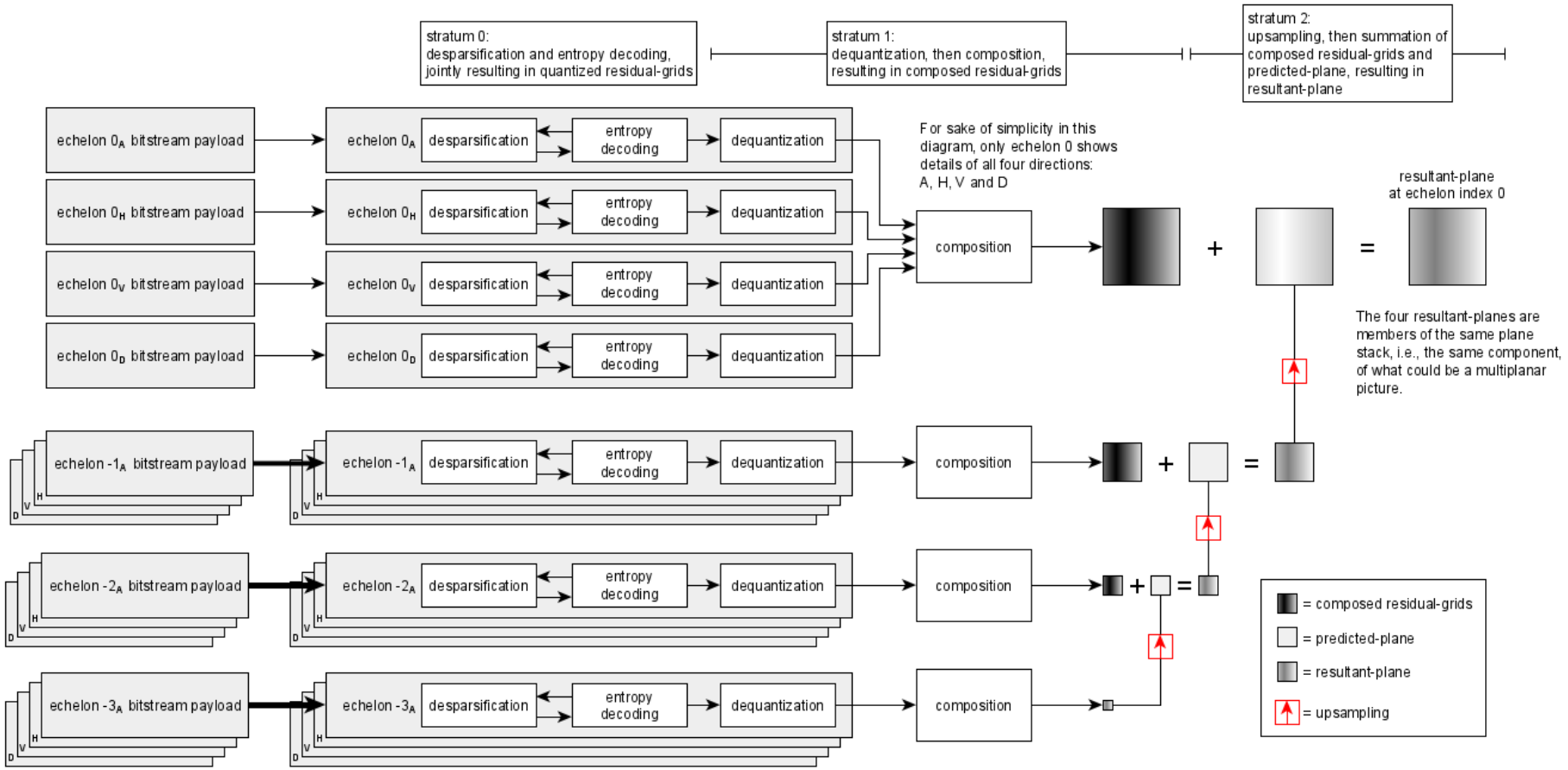


Figure A.1 — Four cycles of plane reconstruction

Annex B
(Normative)

Shortcut Types and Effects on the EPS

B.1 Overview (Informative)

Shortcuts can be used on the EPS to transform the coded format of every residual-grid in the bitstream. The following Clauses B.2 through B.10 describe each of the shortcuts and their respective effect on the bitstream.

B.2 Terminal-tableau-set Bypassing, Shortcut Bit S[1]

If, and only if, `S[1]` is set to `1`, then terminal-tableau-set bypassing shall be enabled, wherein terminal-tableau-sets shall implicitly have every tableau metadata catalog item index the same as that of the terminal-tableau-set-index, and thus, `terminal_tableau_sets` is absent from the tertiary header.

If, and only if, `S[1]` is set to `0`, then the terminal-tableau-set-index of a terminal-tableau specifies a terminal-tableau-set for use by that terminal-tableau.

B.3 Static Terminal-tableau-set Allocation, Shortcut Bit S[2]

If `S[1]` is set to `0`, then `S[2]` shall be set to `0`.

If `S[1]` is set to `1`, then `S[2]` shall be set to `0` or `1`.

If, and only if, `S[2]` is set to `1`, then static terminal-tableau-set allocation is enabled, wherein terminal-tableau-set-indices shall be omitted from terminal-tableaux, and there shall be one terminal-tableau-set-index associated with each live grid.

If, and only if, `S[2]` is set to `0`, then a terminal-tableau-set-index is associated with each terminal-tableau.

B.4 Quantization Disabled, Shortcut Bit S[3]

If, and only if, `S[3]` is set to `1`, then quantization shall be disabled, wherein resid-vals shall not be quantized, and integer composition shall be used instead of standard composition.

If, and only if, `S[3]` is set to `0`, then quantization of resid-vals shall be enabled, and standard composition shall be used.

B.5 Perfect Quadtree, Shortcut Bit S[4]

If, and only if, `S[4]` is set to `1`, then perfect quadtree shall be enabled, wherein sparse tree optimizations shall not be enabled, and grandchild factors shall default to `1111,` eliminating the need for explicit grandchild factors on mini-tree labels.

If, and only if, `S[4]` is set to `0`, then all tableaux use grandchild factors to specify sparsity.

B.6 Shortcut Bit S[5]

Bit `S[5]` is reserved and should be set to `0`.

B.7 Junction-tableau-set Bypassing, Shortcut Bit S[6]

If, and only if, `S[6]` is set to `1`, then junction-tableau-set bypassing shall be enabled, wherein junction-tableau-sets shall implicitly have every tableau metadata catalog item index the same as that of the junction-tableau-set-index, and thus, `junction_tableau_sets` is absent from the tertiary header.

If, and only if, `S[6]` is set to `0`, then the junction-tableau-set-index of a junction-tableau specifies a junction-tableau-set for use by that junction-tableau.

B.8 Static Junction-tableau-set Allocation, Shortcut Bit S[7]

If `S[6]` is set to `0`, then `S[7]` shall be set to `0`.

If `S[6]` is set to `1`, then `S[7]` shall be set to `0` or `1`.

If, and only if, `S[7]` is set to `1`, then static junction-tableau-set allocation is enabled, wherein junction-tableau-set-indices shall be omitted from junction-tableaux, and there shall be one junction-tableau-set-index associated with each junction-tier of each live grid.

If, and only if, `S[7]` is set to `0`, then a junction-tableau-set-index is associated with each junction-tableau.

B.9 Shortcut Bits S[8]…S[16]

`S[8]` through `S[16]` are reserved, and each should be set to `0`.

B.10 Shared Catalogs

In the cases of static terminal-tableau-set or static junction-tableau-set allocation for catalogs shared between junction-tableaux and terminal-tableaux, catalog items related to terminal-tableaux shall precede catalog items related to junction-tableaux.

GrandchildFactorRMFCatalog is the only such shared tableau metadata catalog in this document.

Annex C
(Informative)

Mini-tree-Tableau Correspondence Example

The following example in this clause shows how the nodes and labels of a tableau and mini-tree correspond to one another.

NOTE 1 The legend shown in Figure C.1 applies to Figure C.2, Figure C.3, Figure C.4, Figure C.5, and Figure C.6. Index-difference tuple elements are not shown in these diagrams. Figure C.2 shows a junction-tableau (which is also a root-tableau) that depicts the tableau structure, including node altitudes, child node order and sparsity, of the central s-tree shown in Figure 18.

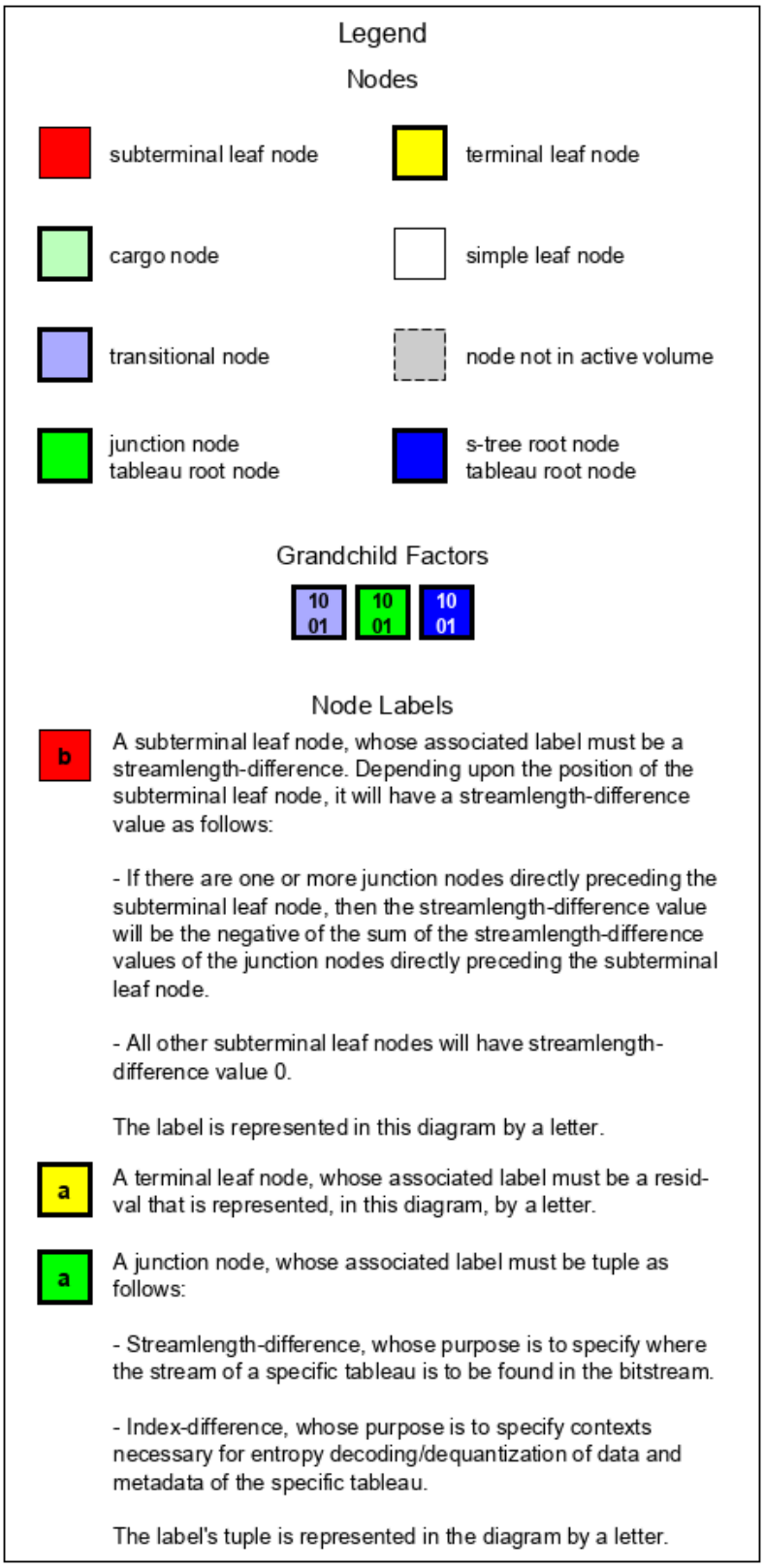


Figure C.1 — Legend for Figure C.2, Figure C.3, Figure C.4, Figure C.5, and Figure C.6

The tableau structure shown Figure C.2 is as follows:

- Tableau-layer 0 contains nodes, all of which have labels, with the exception of nodes that are not within the active volume. Letters “a” through “s” correspond to the values of labels of the first 19 tableau-layer 0 nodes from left to right. The 20th node (rightmost) in tableau-layer 0 is not in the active volume.

  The four child nodes of a parent node are ordered from left to right. The grandchild factors associated with the tableau root node (including s-tree root node), and the transitional nodes of the tableau are ordered from left-to-right, top to bottom.

  Tableau-layer 0 contains the following nodes:

  - 8 junction-/tableau root nodes, identified by the letters a, h, l, m, n, o, p, and q. Each junction node has a non-zero streamlength-difference value.
  - 11 subterminal leaf nodes, identified by the letters b, c, d, e, f, g, i, j, k, r and s. Depending upon the position of a subterminal leaf node, it has a streamlength-difference value as follows:
    - If there are one or more junction nodes directly preceding the subterminal leaf node, (i.e., directly to the left of that subterminal leaf node in Figure C.2 and Figure C.3) then the streamlength-difference value is the negative of the sum of the streamlength-difference values of the junction nodes directly preceding the subterminal leaf node.
    - All other subterminal leaf nodes have streamlength-difference value 0.
  - 1 node not in the active volume

- Tableau-layer -1 contains the following nodes:
  - 5 cargo nodes
  - 6 simple leaf nodes
  - 1 node not in the active volume

- Tableau-layer -2 contains the following nodes:
  - 3 transitional nodes. Each node with children is specified in its parent node as a 1, which represents the 4-digit code 1111.
  - 5 simple leaf nodes. Each simple leaf node is specified in its parent node as a 0, which represents the 4-digit code 0000.

- Tableau-layer -3 contains the following nodes:
  - 2 transitional nodes. Each node with children is specified in its parent node as a 1, which represents the 4-digit code 1111..
  - 2 simple leaf nodes Each simple leaf node is specified in its parent node as a 0, which represents the 4-digit code 0000.

- Tableau-layer -4 contains the s-tree root node. The s-tree root node has four children.

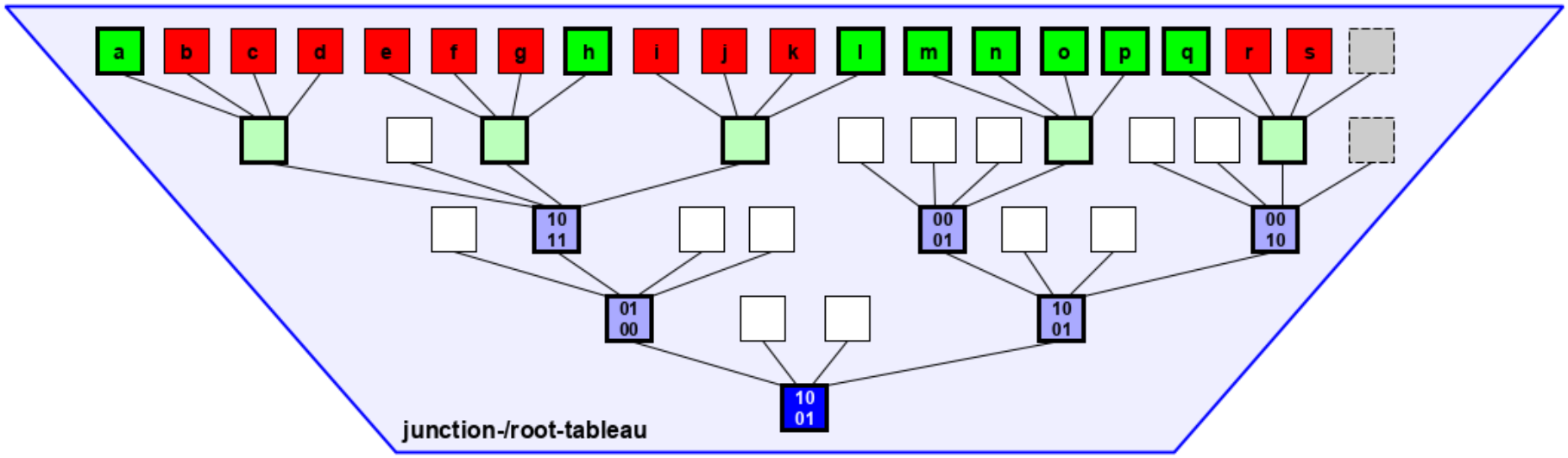


Figure C.2 — Junction-/root-tableau

The entire sparsity structure of a tableau is coded using transitional nodes and their associated grandchild factors. The sparsity structure, as described later in this clause, is defined in its own, simpler, mini-tree with only the original tableau's s-tree root node and the original tableau's transitional nodes used as nodes, and with grandchild factors used as labels.

Figure C.3 shows how a tableau is condensed into a mini-tree. Tableau-layers 0 and -1 of the tableau are represented by labels in the mini-tree, condensing the tableau into 3 layers, and therefore requiring less space in the bitstream.

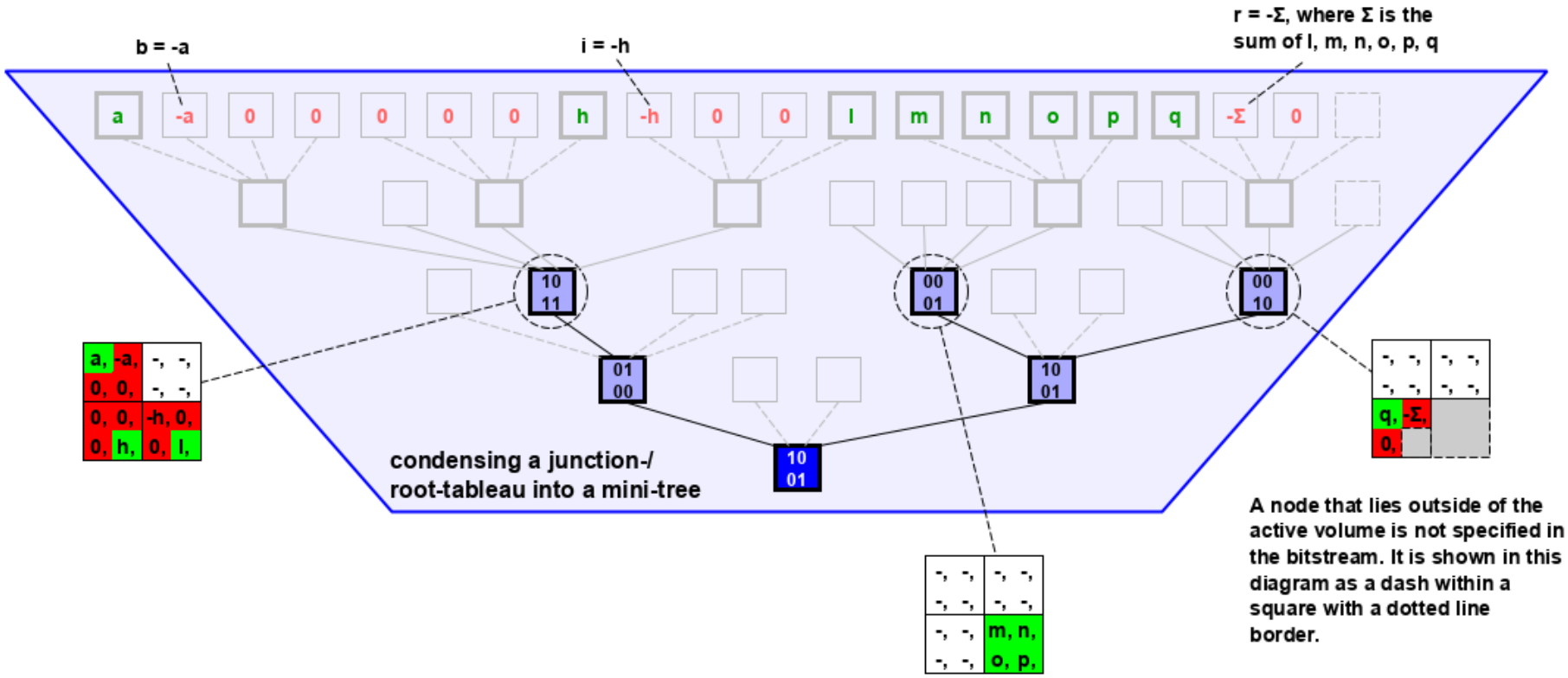


Figure C.3 — Condensing a junction-/root-tableau into mini-tree

The grandchild nodes of each transitional node in tableau-layer -2 of the tableau are specified, in the mini-tree, in a label associated with each transitional node.

For example, in Figure C.3, the label associated with the leftmost transitional node in tableau-layer -2 has four child nodes:

- The leftmost child node has, as children, one junction node with streamlength-difference a, one subterminal leaf node with streamlength-difference −a, and two subterminal leaf nodes with streamlength-difference 0.

  The streamlength-difference −a of the subterminal leaf node whose streamlength-difference was originally represented as b, is calculated as the negative of the sum of the streamlength-differences of the immediately preceding junction nodes. As there is only one preceding junction node, and its streamlength-difference is a, then the streamlength-difference of the subterminal leaf node, originally represented as b, is −a. As such, the sum of the streamlength-differences a and b is 0.

  In another example from Figure C.3, the streamlength-difference of the subterminal leaf node whose streamlength-difference was originally represented as r, is the negative of the sum of the streamlength-differences of the immediately preceding junction nodes m, n, o, p and q, and is therefore(−m + n + o + p + q). As such, the sum of the streamlength-differences m, n, o, p, q and r is 0.

- The next child node to the right has no child nodes; therefore, its associated label contains no values. In the label of this child node, the leaf nodes are shown in white.

- The next child node to the right has, as children, 3 subterminal leaf nodes and one junction node with streamlength-difference h.

- The next child node to the right has, as children, 3 subterminal leaf nodes and one junction node with streamlength-difference l.

The resulting mini-tree is shown in Figure C.4.

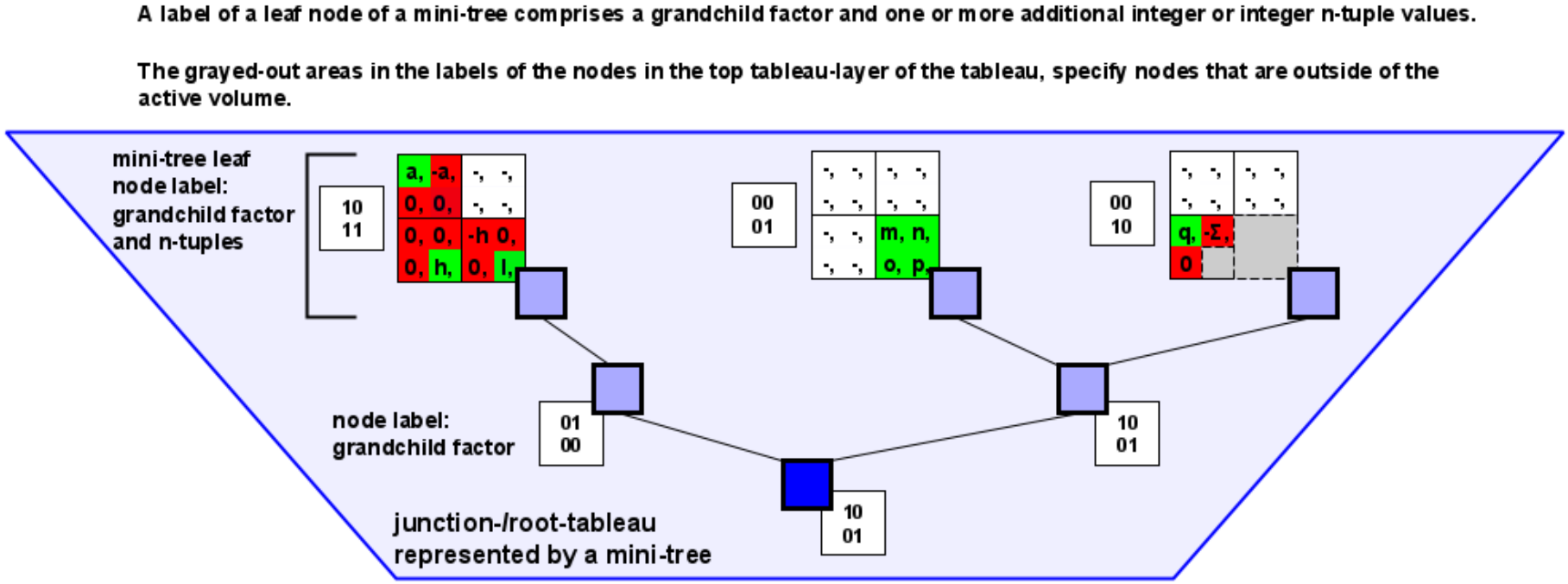


Figure C.4 — Junction-/root-tableau represented by a mini-tree

As shown in Figure C.5 and Figure C.6, a terminal-tableau is condensed into a mini-tree in a similar manner as a junction-tableau, except that the nodes in the tableau-layer 0, which is also altitude 0 of the s-tree, are terminal leaf nodes, and therefore have resid-vals.

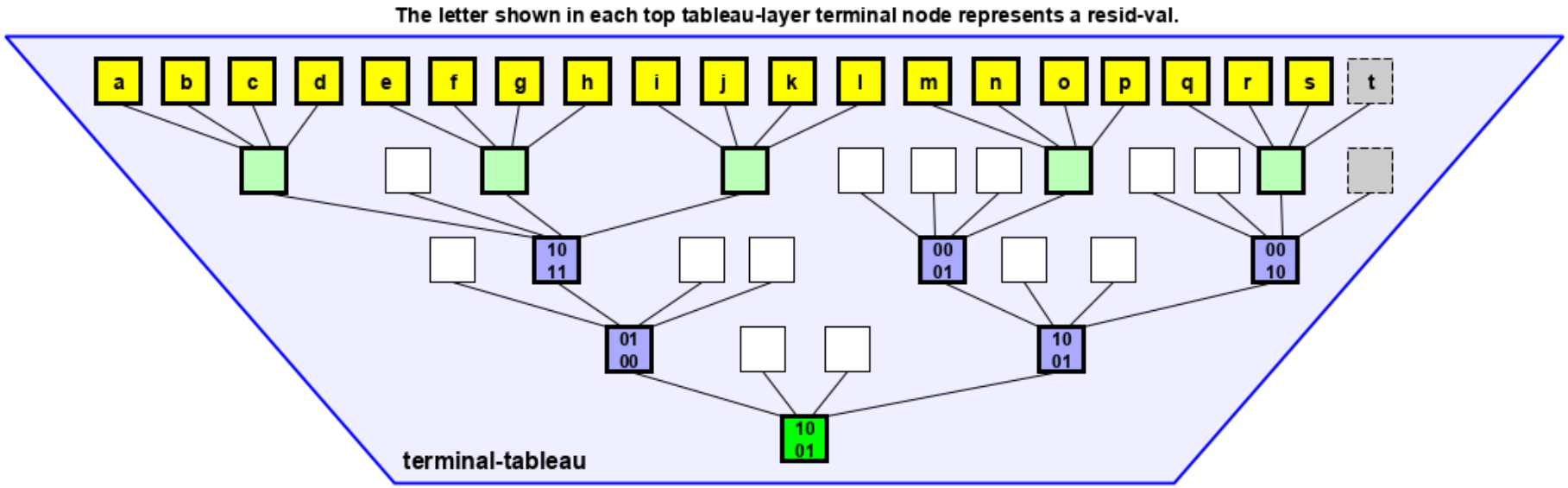


Figure C.5 — Terminal-tableau

The resulting mini-tree, as shown in Figure C.6, has leaf nodes, each of which have grandchild factors and resid-vals.

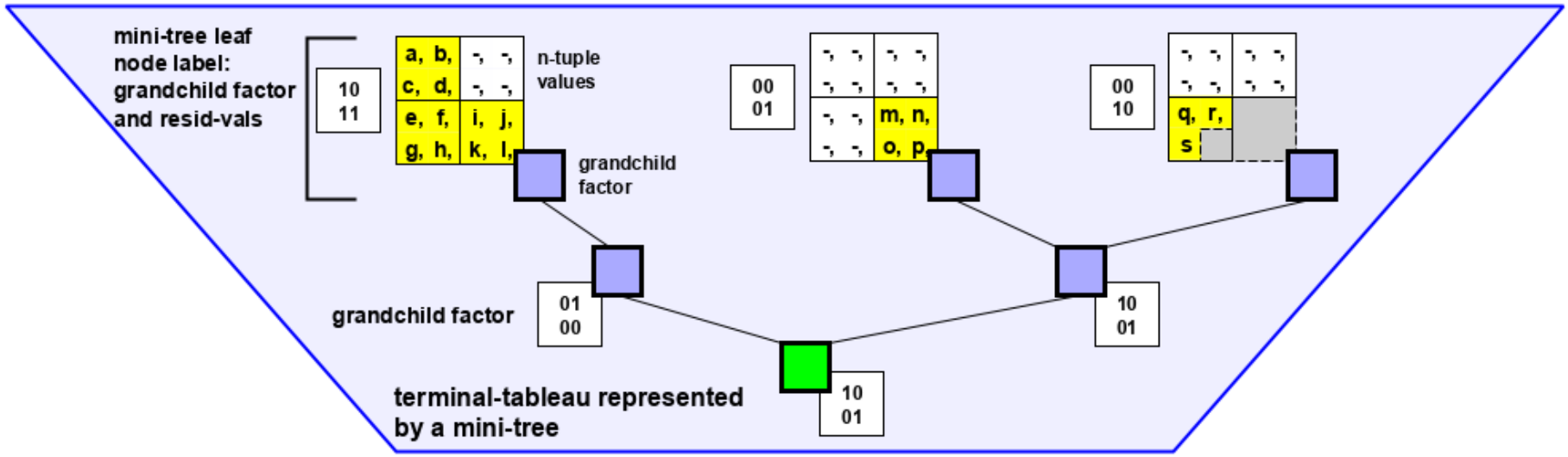


Figure C.6 — Terminal-tableau represented by a mini-tree

Annex D
(Normative)

Processes to Reconstruct a Resultant-Plane from a Payload

D.1 Use of Tableau Metadata Catalogs and Tableau-set-indices

D.1.1 Tableau Retrieval

Tableaux are retrieved during the assembly of larger s-trees. The s-tree assembly process shall be specified as follows:

- An s-tree is assembled tier by tier if its rise exceeds 4.
- When retrieving a tableau within any tier, the tableau retrieval process uses depth-first pre-order traversal.
- The tableau retrieval process starts at the tableau root node of the tableau, and proceeds towards tableau-layer 0 of the tableau. If a simple leaf node is reached, then the tableau retrieval process backtracks through ancestors of that simple leaf node and then retreats to an already visited tableau-layer of the tableau. The tableau retrieval process then proceeds again towards tableau-layer 0 of the tableau.
- The tableau retrieval process repeatedly retreats and proceeds in the same manner in that tableau, until all of the nodes and all of the labels of that tableau have been retrieved.
- Each root node and transitional node in the tableau is associated with a grandchild factor. By using the grandchild factors and the depth-first pre-order traversal, the tableau retrieval process is able to calculate all data in the tableau grid.
- Simple leaf nodes are located based on grandchild factors decoded during the depth-first pre-order traversal.
- A simple leaf node specifies an unpopulated active volume that includes the simple leaf node and its missing descendants, which corresponds to implicit zero values in the tableau grid.
- The tableau retrieval process is capable of explicitly decoding zeros outside of the sparsified areas, that are explicitly present in the bitstream.
- Where zero values occur together with non-zero values in the area of a grid associated with a single cargo node, those zero values are not sparsified in the tableau. Other zero values can be sparsified (i.e., implicit) although they might not be sparsified (i.e., explicit) depending on the sparsified areas.

A cargo node is associated with a 2 by 2 area of the tableau grid, except where the cargo node has child nodes not in the active volume. For example, if a cargo node has one child node in the active volume, then the associated area is 1 by 1.

- Any tableau whose entire active volume is an unpopulated active volume is completely absent from the bitstream payload.
- The final number of tiers depends upon the grid width and grid height of the residual-grid, but is at least 2.
- At each stage of s-tree assembly, retrieved tableaux of a new tier of the s-tree are grafted onto an existing version of the s-tree.

Tableau metadata catalog items might be required for retrieval of tableaux, as specified in Table D.1.

Table D.1 — Tableau Metadata Catalogs

Metadata Catalog	Purpose of Metadata Catalog Items
GrandchildFactorRMFCatalog	Required for retrieving tableaux, unless the perfect quadtree shortcut is specified.
StreamlengthDifferenceRMFCatalog	Required for retrieving junction-tableaux
IndexDifferenceRMFCatalog	Required for retrieving all junction-tableaux, unless: • the static junction-tableau-set allocation shortcut is specified, where a tableau is in tier -3 or -2, and/or; • the static terminal-tableau-set allocation shortcut is specified, where a tableau is in tier -1.
ResidualRMFCatalog	Required for retrieving terminal-tableaux
quantization parameters tuple catalog	Required for retrieving terminal-tableaux unless the quantization disabled shortcut is specified
tableau metadata AUX catalog	Associated with a tableau, but does not have a role in retrieving the tableau. Such metadata is produced by the retrieval process, so that processes external to standard can process and output such metadata in any way

NOTE S-trees are formed recursively in terms of tableau abstract data types (ADTs). Tableaux can be simplified to mini-tree ADTs. Concrete definitions, such as the functions `residual_t()` and `streamlength_t()` and/or `terminal_tableau_set_index_t()` and/or `junction_tableau_set_index_t()` (as described in Table 16), enable this standard to realize the ADTs in the encoding or decoding process.

The permitted values of metadata in tableau metadata catalogs, other than IndexDifferenceRMFCatalog, are provided by the arrays specified in Table D.2.

Table D.2 — Arrays of Metadata in Tableau Metadata Catalogs Other than IndexDifferenceRMFCatalogs

Metadata Array	Catalog(s)	Presence in Tertiary Header of the Bitstream
`grandchild_factor_rmf_catalog`	GrandchildFactorRMFCatalog	present in bitstream, unless the perfect quadtree shortcut is specified
`streamlength_difference_rmf_catalog`	StreamlengthDifferenceRMFCatalog	present in bitstream
`residual_rmf_catalog`	ResidualRMFCatalog	present in bitstream
`quantization_parameters_tuple_catalog`	quantization parameters tuple catalog	present in bitstream unless the quantization disabled shortcut is specified
`aux_catalogs`	tableau metadata AUX catalog(s)	present in the bitstream unless `num_aux_terminal_tableau_catalog` and `num_aux_junction_tableau_catalog` are both `0`

The single permitted value of metadata in each IndexDifferenceRMFCatalog is provided by the arrays specified in Table D.3.

Table D.3 — Metadata Values in IndexDifferenceRMFCatalogs

Metadata Value	Catalog	Presence in Tertiary Header of the Bitstream
`rmf_junction_tableau_sets`	IndexDifferenceRMFCatalog	present in bitstream, unless either the static junction-tableau-set allocation shortcut is specified, or there is only one junction-tier
`rmf_terminal_tableau_sets`	IndexDifferenceRMFCatalog	present in bitstream, unless the static terminal-tableau-set allocation shortcut is specified

D.1.2 Obtaining Tableau Metadata for Sets of Terminal-tableaux and Junction-tableaux

The specific value of a required item of metadata for each tableau to be retrieved is obtained from an element of the corresponding array listed in D.1.1. This element index is calculated from the tableau's tableau-set-index (see D.1.6).

NOTE 1 A tableau-set-index is an index of a tuple specifying metadata needed to retrieve a tableau. The tuple does not specify all metadata needed to retrieve a tableau, because no IndexDifferenceRMFCatalog item is specified by the tuple.

NOTE 2 A terminal-tableau-set is the set of all terminal-tableaux in the bitstream having identical tableau-set-indices.

NOTE 3 A junction-tableau-set is the set of all junction-tableaux in the bitstream having identical tableau-set-indices.

The following shall apply to tableau-sets:

- A terminal-tableau-set contains only terminal-tableaux.
    - Not all terminal-tableaux in an echelon need be included in the same terminal-tableau-set.
    - Terminal-tableaux from any echelons of any directions of any plane stacks can be included in the same terminal-tableau-set.
- A junction-tableau-set contains only junction-tableaux.
    - Not all junction-tableaux in an echelon need be included in the same junction-tableau-set.
    - Junction-tableaux from any echelons of any directions of any plane stacks can be included in the same junction-tableau-set.

The following shall apply to tableau-set-indices:

- The root-tableau of a residual-grid has, as tableau-set-index, an element of the `root_junction_tableau_set_indices` array present in the tertiary header, unless the static junction-tableau-set allocation shortcut is specified.
- Excluding the root-tableau, any tableau belonging to tier t takes a tableau-set-index from the label of some junction node in tableau-layer 0 of some tableau in tier $t - 1$, i.e., the tier below tier t, unless either of the following two conditions applies:
    - $t = 0$ and the static terminal-tableau-set allocation shortcut is specified.
    - $t < 0$, and the static junction-tableau-set allocation shortcut is specified.
- Tableau-set-indices are carried only by labels in a junction-tier.

D.1.3 Terminal-tableau-set Tableau Metadata Catalogs

D.1.3.1 ResidualRMFCatalog

An item in ResidualRMFCatalog is any value that is capable of being returned by the `residual_rmf_t()` function, representing an RMF for resid-vals.

This function and the associated `residual_line_segments_rmf15_t` function are defined in Table 16. The syntax is defined in Table 17.

D.1.3.2 GrandchildFactorRMFCatalog

An item in GrandchildFactorRMFCatalog is any array that is capable of being returned by the `grandchild_factor_rmf_t()` function, as follows:

```
type grandchild_factor_rmf_t : fractional8[6]
```

The array contains fractional RMT values, compactly representing an RMF for grandchild factors, and where:

- 7 bins shall exist, where the last RMT value (1.0) shall be implicit.
- The RMT values shall be non-decreasing, where an RMT of 0.0 that follows a strictly positive RMT shall be replaced by an RMT of 1.0.

NOTE Within each bin, the probability of each grandchild factor is assumed to be equal. The bins are specified in Clause E.4.

The `grandchild_factor_rmf_catalog` is shared between junction-tableaux and terminal-tableaux.

D.1.3.3 Quantization Parameters Tuple Catalog

A quantization parameters tuple in a quantization parameters tuple catalog is returned by the `quantization_parameters_tuple_t()` function, as follows:

```
type quantization_parameters_tuple_t : tuple(stepwidth_t, residual_t)
```

This tuple contains the stepwidth and the dequantization offset. The dequantization offset shall be used for precise dequantization.

D.1.4 Junction-tableau-set Tableau Metadata Catalogs

D.1.4.1 StreamlengthDifferenceRMFCatalog

A value of StreamlengthDifferenceRMFCatalog is any value capable of being returned by the `streamlength_difference_rmf_t()` function, representing an RMF for differences of streamlength values.

This function and the associated `streamlength_line_segments_rmf15_t` function are defined in Table 16. The syntax is defined in Table 17.

D.1.4.2 GrandchildFactorRMFCatalog

The junction-tableau has the same type of GrandchildFactorRMFCatalog as that of the terminal-tableau-set, as described in D.1.3.2. The `grandchild_factor_rmf_catalog` array is shared between junction-tableaux and terminal-tableaux.

D.1.5 Line-Segments RMF Tuple Type

A line-segment is part of a compact representation of an RMF. In a line-segment, the ends are known as knots. The right-hand knot of a segment has an RMT value that represents all sample values from minus infinity (−∞) up to the end of that segment.

The conditional probabilities of the different symbols within a specific line-segment are not individually signaled, as they can be deduced by interpolation between 0.0 and 1.0 over the line-segment to an accuracy of 16 fractional bits.

The tuple-valued data, `tuple(rmt[14], knot[16])`, can represent 15 line segments, as follows:

- The tuple shall contain a first array of fractional RMT values and a second array of knots of line-segments.
- The last RMT value (1.0) shall be implicit.

This tuple type is returned by each of the following functions:

- `residual_line_segments_rmf15_t()`
- `streamlength_line_segments_rmf15_t()`
- `terminal_tableau_set_index_line_segments_rmf15_t()`
- `junction_tableau_set_index_line_segments_rmf15_t()`

RMT elements are of the type returned by `read_bits_fractional(16)` and knot elements depend upon the specific function. See Table 17 for specifications.

NOTE This choice of tuple type enables the same functional block to be used for retrieving both junction-tableaux and terminal-tableaux, with some parametrization.

D.1.6 Junction-tableau-set and Terminal-tableau-set Specifications

NOTE 1 The fixed-size primary header and the variable-sized secondary and tertiary headers are defined in 8.4, 8.5 and 8.6, respectively. The secondary header can end with user-defined data, which the decoding process ignores. This clause pertains to the data in the tertiary header.

Tableau metadata items (see D.1.3 and D.1.4) apply to 0 or more terminal-tableau-sets or to 0 or more junction-tableau-sets, based on the elements of arrays that are included in the tertiary header.

The tableau-set-index of a tableau shall determine the values of metadata items associated with a tableau.

NOTE 2 The metadata items, when applied to mini-trees, permit decoding of each junction-tableau or terminal-tableau in the next tier due to be grafted.

Where a tableau metadata AUX catalog has an `aux_catalog_item_size[i]` value of `0`, then no catalog item value shall be provided from that metadata catalog to any tableau.

Other tableau metadata catalogs present in the header shall provide values for every tableau, or only for every terminal-tableau, or only for every junction-tableau.

The following shall apply to the use of metadata catalogs by tableau-sets:

- Terminal-tableau-sets:
    - ResidualRMFCatalog is a tableau metadata catalog of all terminal-tableau-sets.
    - If the GrandchildFactorRMFCatalog tableau metadata catalog is present, then it is a tableau metadata catalog of all terminal-tableau-sets.
    - If the quantization parameters tuple catalog is present, then it is a tableau metadata catalog of all terminal-tableau-sets.
    - If any tableau metadata AUX catalogs for terminal-tableaux are present, then they are the only other tableau metadata catalogs of a terminal-tableau-set.
- Junction-tableau-sets:
    - StreamlengthDifferenceRMFCatalog is a tableau metadata catalog of all junction-tableau-sets.
    - If the GrandchildFactorRMFCatalog tableau metadata catalog is present, then it is a tableau metadata catalog of all junction-tableau-sets.
    - If any tableau metadata AUX catalogs for junction-tableaux are present, then they are the only other tableau metadata catalogs of a junction-tableau-set.

For a given grid, `grid_no` shall be the number of live grids preceding that grid in the calculation in Table 23 of `num_live_grids`.

The following shall apply to terminal-tableau metadata catalog items:

- If the static terminal-tableau-set allocation shortcut is specified, then the tableau-set-index of all terminal-tableaux in a residual-grid is the `grid_no` of the residual-grid.

- The terminal-tableau-set bypassing shortcut applies to metadata catalog items as follows:

    - If the terminal-tableau-set bypassing shortcut is specified, then any terminal-tableau with tableau-set-index value equal to `j` has the following metadata catalog items:

        - `residual_rmf_catalog[j]`,
        - `grandchild_factor_rmf_catalog[j]`,
        - `quantization_parameters_tuple_catalog[j]`, and
        - `aux_catalogs[i][j]`

      where `i` ranges from `0` to `(num_aux_terminal_tableau_catalog – 1)`.

    - Otherwise, it has the following metadata catalog items:

        - `residual_rmf_catalog[terminal_tableau_sets[0][j]]`,
        - `grandchild_factor_rmf_catalog[terminal_tableau_sets[1][j]]`,
        - `quantization_parameters_tuple_catalog[terminal_tableau_sets[2 – S[4]][j]]`, and
        - `aux_catalogs[i][terminal_tableau_sets[i + 3 – S[3] – S[4]][j]]`

      where `i` ranges from `0` to `(num_aux_terminal_tableau_catalog – 1)`.

For a junction-tier that is the lowest tier of a residual-grid, `global_tab_tier_no` is defined as the number of live tiers in all live grids preceding that grid in the calculation in Table 23 of `num_live_grids`. The value of `global_tab_tier_no` then increases by 1 for each higher tier until tier -1 of the same residual-grid.

The following shall apply to junction-tableau metadata catalog items:

- If the static junction-tableau-set allocation shortcut is specified, then the tableau-set-index of all junction-tableaux in a tier of a residual-grid is the `global_tab_tier_no` of the tier.

- The terminal-tableau-set bypassing shortcut applies to metadata catalog items as follows:

    - If the junction-tableau-set bypassing shortcut is specified, then any junction-tableau with tableau-set-index value equal to j has the following metadata catalog items:

        - `streamlength_difference_rmf_catalog[j]`,
        - `grandchild_factor_rmf_catalog[offset + j]`, and
        - `aux_catalogs[num_aux_terminal_tableau_catalog + i][j]`

      where `i` ranges from `0` to `num_aux_junction_tableau_catalog` minus `1`, and where offset is `0` if neither static terminal-tableau-set allocation shortcut nor static junction-tableau-set allocation shortcut is specified, and otherwise is `num_aux_terminal_tableau_catalog` if static terminal-tableau-set allocation shortcut is not specified and otherwise is `num_live_grids`.

- Otherwise, it has the following metadata catalog items:
  - streamlength_difference_rmf_catalog[junction_tableau_sets[0][j]],
  - grandchild_factor_rmf_catalog[offset + junction_tableau_sets[1][j]], and
  - aux_catalogs[num_aux_terminal_tableau_catalog + i][junction_tableau_sets[i + 2 – S[4]][j]]

  where i ranges from 0 to num_aux_junction_tableau_catalog minus 1 and where offset is 0 if static terminal-tableau-set allocation shortcut is not specified and otherwise is num_live_grids.

If the static terminal-tableau-set allocation shortcut is not specified, then the value of tableau-set-index, j, as used throughout D.1.6, in finding metadata catalog items of a terminal-tableau, will always be an integer greater than or equal to zero and consistent with the following constraints:

- the index applied to residual_rmf_catalog[] is less than nums_catalog[0]
- the index applied to grandchild_factor_rmf_catalog[] is less than nums_catalog[1]
- the index applied to quantization_parameters_tuple_catalog[] is less than nums_catalog[2 – S[1] – S[4]]
- the index applied to aux_catalogs[i][] is less than nums_catalog[4 – S[1] – S[4] – S[3] + i]

If the static junction-tableau-set allocation shortcut is not specified, then the value of tableau-set-index, j, as used throughout D.1.6, in finding metadata catalog items of a junction-tableau, will always be an integer greater than or equal to zero and consistent with the following constraints:

- the index applied to streamlength_difference_rmf_catalog[] is less than nums_catalog[3 – S[1] – S[4] – S[3]]
- the index applied to grandchild_factor_rmf_catalog[] is less than nums_catalog[1]
- the index applied to aux_catalogs[i][] is less than nums_catalog[4 – S[1] – S[4] – S[3] + i]

Unless S[2] shortcut bit is set to 1, tableau-set-indices, used in decoding the root-tier of a residual-grid, are obtained from a root_junction_tableau_set_indices array in the tertiary header (see 8.6).

Unless S[8] shortcut bit is set to 1, tableau-set-indices, used in decoding tiers of a residual-grid, other than the root-tier, are obtained by decoding index-differences from labels of junction-tableaux (see D.2.8.3).

When decoding index-differences from labels of junction-tableaux, the labels at tableau-layer 0 of the root-tableau shall be decoded to provide tuples, (streamlength-difference, index-difference), as described in 9.12. Each resulting tableau-set-index is the tableau-set-index for decoding a tableau in the next (higher) tier. The same method of determining and using tableau-set-indices is used with tableau-set-indices decoded from tableau-layer 0 of all tableaux, until tableau-set-indices decoded from the tableau-layer 0 of tier -1 have been used to decode the terminal-tableaux.

To decode the tableau-set-index from a label, the decoding process shall use the RMF from one of the two IndexDifferenceRMFCatalogs, specified as follows:

- `rmf_terminal_tableau_sets` is used to decode index-differences from all junction-tableaux in junction-tier -1, in the given bitstream.
- `rmf_junction_tableau_sets` is used to decode index-differences from all junction-tableaux in junction-tiers lower than junction-tier -1, in the given bitstream.

IndexDifferenceRMFCatalog RMF shall refer to the following RMFs, depending upon which junction-tier is being decoded:

- For junction-tier -1, `rmf_junction_tableau_sets` is used.
- For lower junction-tiers, `rmf_terminal_tableau_sets` is used.

EXAMPLE
It can be recognized, from earlier in this clause, D.1.6, that when no shortcuts are specified, and where `tsinda` and `tsindb` are valid tableau-sets:

- `quantization_parameters_tuple_catalog[terminal_tableau_sets[2][tsinda]]` provides a particular choice of quantization parameters tuple catalog metadata item, and
- `streamlength_difference_rmf_catalog[junction_tableau_sets[0][tsindb]]` provides a particular choice of StreamlengthDifferenceRMFCatalog metadata item.

D.2 Retrieving Junction-tableaux and Terminal-tableaux Using RMFs

D.2.1 Overview (Informative)

Tableaux retrieval uses multiple RMFs (see Annex E) to decode multiple types of symbols in a tableau, as follows:

- In a terminal-tableau, the symbol types requiring RMFs are:
  - grandchild factors
  - resid-vals
- In a junction-tableau, the symbol types requiring RMFs are:
  - grandchild factors
  - index-differences
  - streamlength-differences

The colors in Figure D.1 and Figure D.2 correspond to the node colors defined in the legend specified in Figure 12.

Figure D.1 shows an example during decoding of a terminal-tableau, where a label containing a grandchild factor and 12 resid-vals is in the process of being decoded. In Figure D.1, only the first 8 resid-vals of that label have been decoded, unlike in Figure D.2.

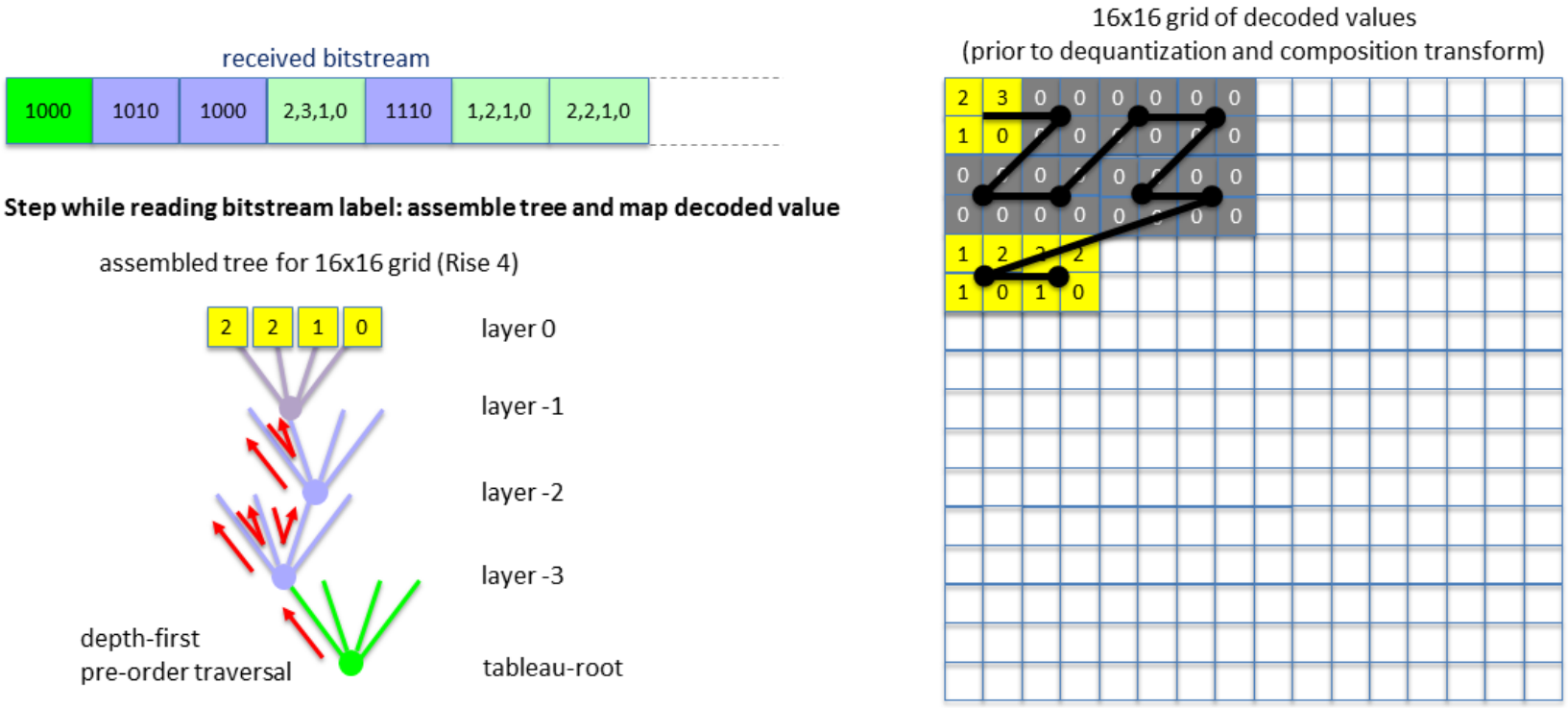


Figure D.1 — A step in the decoding of a terminal-tableau

In the example grid pictured in Figure D.1, the top-right quarter of the bottom-left quarter of the top-left quarter of the terminal-tableau receives the four resid-vals 2, 2, 1 and 0. The values were decoded from a label starting with grandchild factor 1110 and continuing with resid-vals 1,2,1,0,2,2,1,0,0,2,0,1 as pictured in the bitstream.

Figure D.2 shows an example during decoding of the same terminal-tableau where 64 resid-vals would not be decoded at any stage in the decoding. This figure shows a simulated dense portion of the sparse s-tree, receiving resid-vals of value 0. In practice, none of these values are received from the bitstream because the value of these resid-vals is implicit.

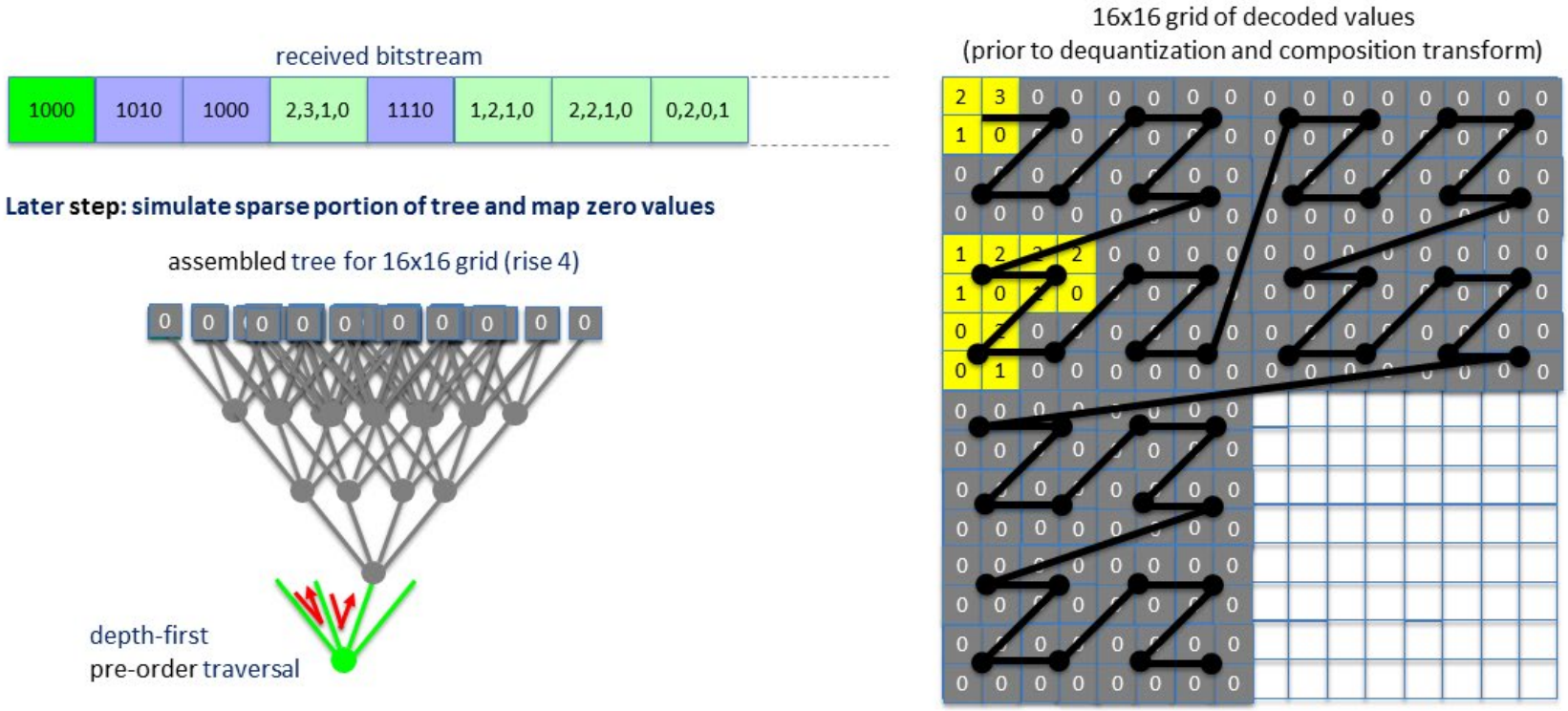


Figure D.2 — A later step in the decoding of the same terminal-tableau as in Figure D.1

In the example grid pictured in Figure D.2, the bottom-left quarter of the terminal-tableau receives resid-vals all of value 0. The values were not decoded from a label. The very first grandchild factor 1000 is pictured in the bitstream. The tableau-layer -2 symbol signaled the 64 implicit zeroes by means of its third digit and was the only value decoded (earlier) from the very first (i.e., tableau-layer -1) label.

Ensuring that the correct RMFs are used to decode a given junction-tableau or terminal-tableau involves some preparation by the decoding process, which is described in D.2.6.

D.2.2 Rules for Junction-tableaux and Terminal-tableaux Ordering

The s-tree depicted in Figure 19 would be transmitted by a bitstream using one junction-tableau and 8 terminal-tableaux.

The root-tableau is the one and only tableau in the root-tier of an s-tree.

There are $n + 1$ tiers for a particular s-tree, where n is the number of junction-tiers.

The number of junction-tiers, n, shall be the total contribution to the value of `num_live_junction_tiers` from lines 68, 75 and 77 in Table 23 accumulated while the values of indices `i`, `j` and `k` of line 66 are the index values of the associated residual-grid.

NOTE 1 Tier $-n$, the root-tier, is a junction-tier and is the tier with lowest index.

NOTE 2 Tier 0, the terminal-tier, is the tier with highest index.

Figure D.3 shows an extract of a tertiary header (at the left side of the diagram), where `root_junction_tableau_streamoffset` indicates an element of `root_junction_tableau_streamoffsets` and `root_junction_tableau_trailing_spaces` indicates an element of `root_junction_tableau_trailing_spaces`. These two indicated offsets constitute the only metadata from the tertiary header that affect the layout of the associated s-tree. These two offsets control the locations of two of the tableaux of the associated s-tree, but do not directly affect the locations of the other tableaux of that s-tree. These other locations are discussed later in the following paragraphs of the current clause.

Figure D.3 shows a hypothetical s-tree's bitstream layout. The root-tableau is in tier -2 and follows 221 bytes after the end of the tertiary header since the `root_junction_tableau_streamoffsets` element of the s-tree has a value of 221. The root-tableau has two children, which are both in tier -1. It also has three grandchildren, namely two children of its first child and one child of its second child. These grandchildren are all in terminal-tier 0.

NOTE 3 The terms "child" and "grandchild" are used in the above paragraph and the following paragraph in the context of a tree, all of whose nodes are themselves tableaux, as in Figure D.4, rendering it not an example of any s-tree. It is a useful higher level of abstraction than an s-tree, and is not used anywhere else.

The decoding process of the root-tableau, once completed, establishes the length of the root-tableau in the s-tree's bitstream layout.

The offset from the start of the root-tableau in Figure D.3 to the start of its first child is formed by adding 3 to the length in bytes of the root-tableau since the `root_junction_tableau_trailing_spaces` element of the s-tree has a value of 3.

The offset from the start of this first child to the start of the second child has a value of 30 bytes and the offset from the start of the second child to the start of the first grandchild has a value of $(30 - 5) = 25$ bytes.

These children are tableaux: the first child defines offsets of 19 bytes from the start of the first grandchild to the start of the second grandchild, and of $(19 - 2) = 17$ bytes from the start of the second grandchild to the start of the third grandchild; the second one defines a spare offset, of 17 bytes from the start of the third grandchild.

The spare offset is the last of the offsets and, though it has a value, might not lead to any specific element of any specific s-tree.

NOTE 4 `Root_junction_tableau_streamoffsets` elements are positive or zero. `Root_junction_tableau_trailing_spaces` elements, though positive and non-zero in the case described previously in the current clause, are in general positive, negative or zero. The other offsets are also positive, negative, or zero, but the value zero corresponds to an offset without a target, i.e., one of the possibly many tableaux absent from the bitstream payload, and therefore, absent from Figure D.3.

Offsets in bytes for a junction-tableau or terminal-tableau, counted from the end of the tertiary header, shall be as follows, where k is an integer satisfying $1 \leq k \leq n$:

- Offset to a root-tableau, counted from the end of the tertiary header, is an element of `root_junction_tableau_streamoffsets`.
- Offset to the start of a junction-tableau (or terminal-tableau) in tier $-n + k$, counted from the end of the tertiary header is the offset to the root-tableau, plus all of the following:
    - streamlength of the root-tableau;
    - a total sum of other streamlengths, where one streamlength is associated with each label of the tableaux in all tiers below tier $-n + k - 1$; and,
    - a partial sum of other streamlengths, where one streamlength is associated with each label of the tableaux of tier $-n + k - 1$, and where the partial sum concludes with the last streamlength before the streamlength of the given tableau of tier $-n + k$.

NOTE 5 As an example, offset to the start of a junction-tableau (or terminal-tableau) in tier $-n + 1$ will be the offset to the root-tableau, plus the streamlength of the root-tableau, plus zero total sum, plus a partial sum of streamlengths, where one streamlength is associated with each label of the root-tableau.

NOTE 6 The bitstream contains the streamlength-difference instead of encoding the streamlength element of labels directly. This is covered in D.2.8.3 and is not needed here as the discussion is of the labels not the bitstream.

Figure D.3 and the text following this figure illustrate these concepts, for $n = 2$.

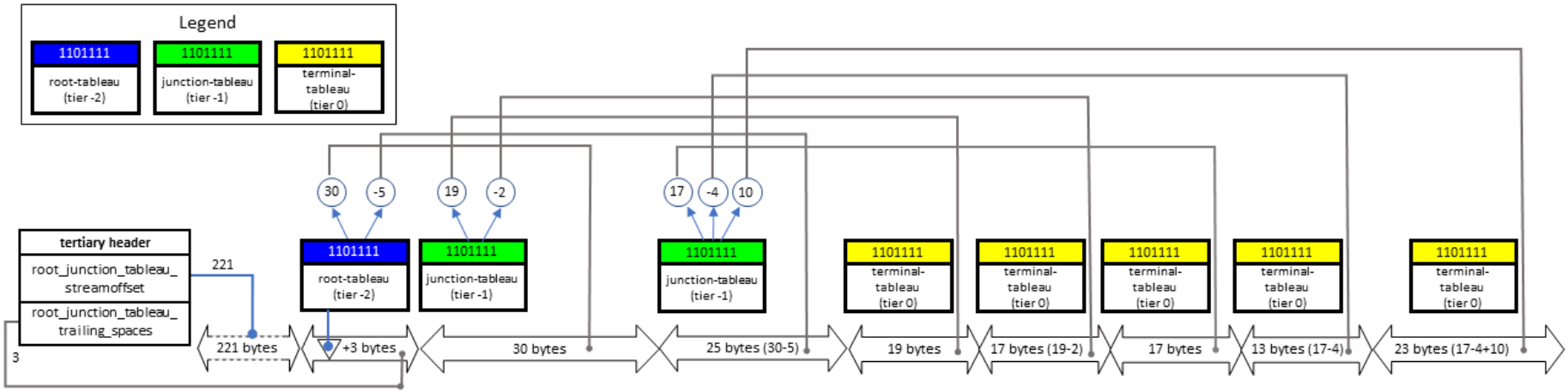


Figure D.3 — An s-tree's bitstream layout after the tertiary header, as 8 streams within two junction-tiers and a terminal-tier

Figure D.3 shows the first seven encoded bits in each of 6 tableaux belonging to one residual-grid, using incorrect bit values from a dummy bitstream. Links from the root-tableau lead to tier -1 tableaux (or terminal-tableaux, for smaller grids, unlike in the current figure) and links from tier -1 junction-tableaux lead to terminal-tableaux. Figure D.3 is independent of the encoded bits in the terminal-tableaux, but not the junction-tableaux.

Per the tableau ordering specifications, offsets to tableaux depend on the data in previous junction-tableaux. For example, without the data of the root-tableau of a residual-grid, there is no way to access the data in the subsequent tiers of that residual-grid.

Tier 0 contains only one or more terminal-tableaux. Other tiers contain only one or more junction-tableaux.

NOTE 7 Once the root-tableau has been decoded, the total number of tier $-n + 1$ junction-tableaux is the total number of non-zero streamlengths which are calculated by the additional decoding process (see D.2.8.3) during the traversal stage (see D.2.8).

After decoding all the tier $k - 1$ junction-tableaux, the decoding process can select the head and total number, of the junction-tableaux (for $k < 0$) and terminal-tableaux (for $k = 0$), in tier k.

In the case of tier $-n$, the offset of the root-tableau shall come from the header.

NOTE 8 From the sources and targets of the links connected to all the junction-tableaux and offsets in Figure D.3, it can be noticed that the decoded tableaux might be combined into one larger s-tree as represented schematically in Figure D.4 (by the processes specified in this document).

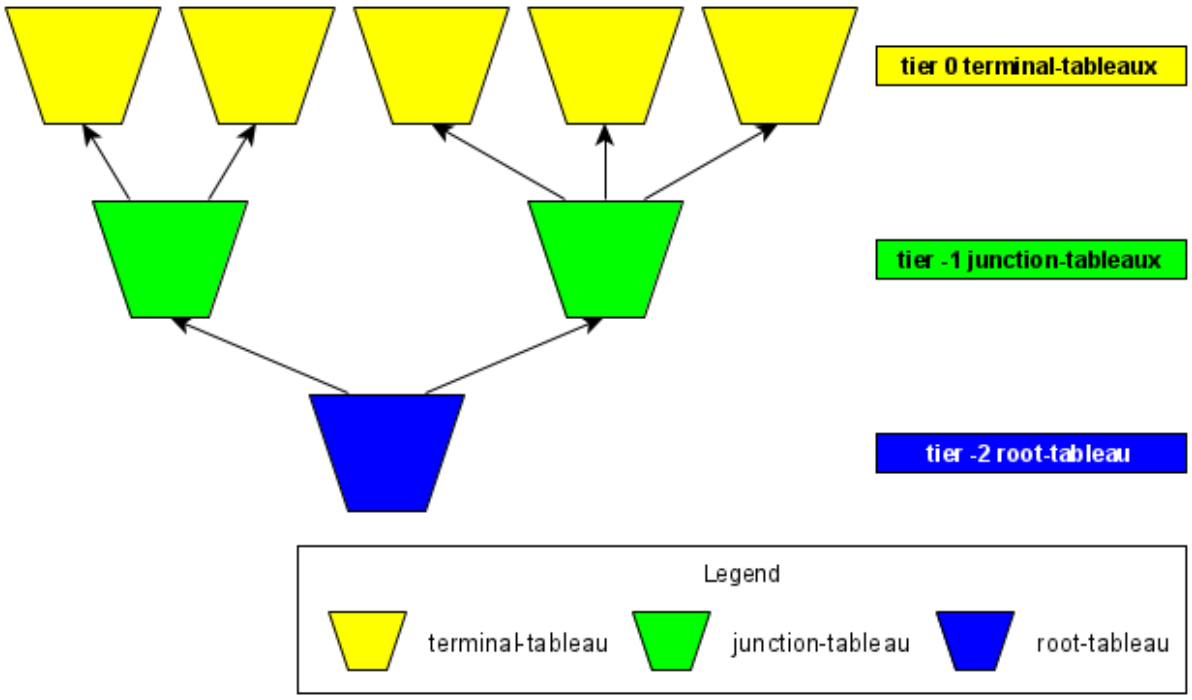


Figure D.4 — Anticipated structure of the s-tree determined by the links of Figure D.3

Streams exist only for tableaux that have a populated active volume.

The offsets (e.g., as shown at the bottom of Figure D.3) shall be signed and not be zero, except that the `root_junction_tableau_streamoffsets` element shall be positive or zero.

NOTE 9 This means that the start of a stream is located on any byte boundary within the bitstream except that, in order of depth-first pre-order traversal of the labels of a given junction-tableau, the start of a tableau that is represented in the bitstream by a mini-tree differs from the start of the next tableau that is represented in the bitstream by a mini-tree.

The streamlength value of zero is used for denoting a tableau that is not represented in the bitstream by a mini-tree. The streamlength value can appear within a label but is not used for location of a stream.

In a valid bitstream, streams shall not overlap with the headers.

Figure D.3 shows 7 streamlength-differences, from which streamlengths are simply obtained, and shows the use of the streamlengths as offsets.

NOTE 10 The start of the bitstream, and the primary and secondary headers are not included in Figure D.3.

The offset in bytes from the start of the bitstream to the start of the root-tableau shall be the sum of an element of `root_junction_tableau_streamoffsets` and the lengths of the headers.

The offset in bytes from the start of the root-tableau to the first tableau of the tier immediately above the root-tableau shall be the sum of an element of `root_junction_tableau_trailing_spaces` and the number of bytes used in decoding the root-tableau.

The 5 remaining offsets from streams in Figure D.3 are the streamlengths from labels on the root-tableau, and from the tier -1 junction-tableaux present in this example. These are retrieved from streamlength-differences, as shown, by adding them to the previous streamlength. The notional previous streamlength shall be initialized to 0 at the start of each tableau.

NOTE 11 For example, in Figure D.3, the following tableau's offset from the start of the root-tableau shown is 3 bytes greater than the length of the root-tableau.

NOTE 12 In Figure D.3, subterminal leaf nodes have streamlength-differences leading to offsets of 0 (tableaux that have entire active volumes that are unpopulated active volumes), but are not shown. All others are shown. In Figure D.3, it is assumed that two non-zero streamlength-differences on a tableau have no intervening streamlength-difference.

NOTE 13 The location reached by the final offset of junction-tier with tier index -1 has no specific role; for example, the last offset of 17 bytes in Figure D.3 might not lead to the start of a stream of any tableau in any residual-grid.

The targets of offsets are the starts of certain bytes. In a valid bitstream the starts are no more than `bitstream_size` bytes after the start of the bitstream.

The number of bytes used in decoding a tableau from a valid bitstream does not exceed the number of bytes from the start of the tableaux to the byte whose number is `bitstream_size`.

D.2.3 Permuted Bitstream Payloads

In a permuted bitstream payload, the tiers from an echelon are not contiguous in order of significance. Permuted bitstreams can be of different types, depending upon the `streams_key` syntax term specified in the bitstream tertiary header.

NOTE For example, in some permuted bitstreams, a tier that belongs to one echelon of a plane stack, can be sequenced between two tiers of a different echelon of the same or a different plane stack.

Let "$pedt$" indicate that plane stack index p is more significant than echelon index, echelon index e is more significant than direction index d, and tier index t is least significant. Other combinations have analogous meanings.

A bitstream has a permuted bitstream payload if tier index is not the least significant index. Thus, there are 6 of 24 cases where the bitstream payload is organized as in Figure D.3 without being permuted.

In the other cases, there is a permutation. The logic of Figure D.3 still applies, except that, the final offset in a tier, which starts before the final tableau in the tier, ends at the start of a tier of a different echelon instead of at the start of the following tier of the same echelon.

The permuted tiers are ordered from lowest to highest in p, e and d, and from highest to lowest in t. The order's significance for each index p, e, d and t varies from 3 (most significant) to 0 (least significant). The first (i.e., least significant) tier, in a first live echelon, is a root tier. Any non-root-tier, that is a member of a live echelon, has an immediately preceding tier, not necessarily a member of the same live echelon, and has a predecessor, not necessarily immediate, that is the previous tier of the same live echelon.

If a given tier is a root-tier, then the given root-tier starts `root_junction_tableau_streamoffsets` element bytes after the start of the bitstream payload, as already discussed.

If a given tier is not a root-tier:

- Then, if the immediately preceding tier is a root-tier, then the offset from the start of the root-tier to the start of the first tableau of the given tier shall be the coded length of that root-tableau, rounded up to the nearest octet, plus `root_junction_tableau_trailing_spaces` bytes.
- Then, if the immediately preceding tier is not a root-tier:
    - Then the offset from the start of the final tableau of that previous tier, to the start of the first tableau of the given tier, shall be the final one of the streamlengths derived from the tableaux of the previous tier.
    - Then the offset from the start of any tableau in the given tier, to the start of the next tableau in that given tier, shall be the corresponding streamlength derived from the tableaux of the predecessor that is the previous tier of the same live echelon.
    - Then the offset from the start of any tableau in the given tier, to the start of the next tableau in that given tier, shall be the corresponding streamlength derived from the root-tableau of the same echelon.

D.2.4 End Markers

End markers are metadata in headers that denote locations in the payload. There shall be at least one and not more than 17 end markers in a bitstream.

An element or elements of the `end_markers` grid are in the bitstream as defined in 8.4 and 8.5. They are specified in the primary header and, if there is more than one element in the grid, then in the secondary header.

A byte is classified as relevant for `end_marker_echelon[i]` if, for at least one non-positive integer `m` where `m ≤ −end_marker_echelon[i]`, this document requires, for decoding symbols of at least one tableau of plane stack `end_marker_plane[i]` and of echelon index m, this byte to be read from the bitstream.

The value of `end_markers[i]`, where i is a non-negative integer less than or equal to `maximum_end_marker_index`, shall be the number of bytes of the bitstream before and including those bytes classified as relevant for `end_marker_echelon[i]`.

NOTE Valid values of `end_markers[i]` cannot alter the interpretation of any junction-tableau or terminal-tableau of the payload.

D.2.5 RMF Catalogs

This standard supports the use of different RMFs for decoding the same type of symbol, depending upon the individual junction-tableau or terminal-tableau. There are four tableau metadata RMF catalogs, each of which contains RMFs that decode their respective symbol. The catalogs and the maximum number of RMFs in each catalog are as specified in Table D.4.

Table D.4 — RMF Catalogs

RMF Catalog	Symbol that Is Decoded	Maximum Number of RMFs
GrandchildFactorRMFCatalog	grandchild factor for junction-tableaux	32,768
	grandchild factor for terminal-tableaux	32,768
StreamlengthDifferenceRMFCatalog	streamlength-difference	65,536
IndexDifferenceRMFCatalog	index-difference for junction-tableaux	1
IndexDifferenceRMFCatalog	index-difference for terminal-tableaux	1
ResidualRMFCatalog	residual	65,536

The RMF Selection Stage of D.2.6 shall determine which RMFs from which tableau metadata RMF catalogs shall be involved in decoding a specific tableau. In any RMF, RMT values shall be constrained as follows:

- All RMT values are non-negative and non-decreasing, and ≤ 1.0. An RMT value of 0.0 can follow a strictly positive RMT value, but all 0.0 values following a strictly positive RMT value are interpreted as 1.0.
- An RMT value of 1.0 is not represented explicitly.
- The last RMT value is implicitly 1.0 (and is also non-decreasing). The last RMT value is the 7th RMT in the case of GrandchildFactorRMFCatalog, and the 15th RMT in the other tableau metadata RMF catalogs.

The GrandchildFactorRMFCatalog RMF shall not have 16 individual RMT values for the 16 possible grandchild factor values. Each grandchild factor value shall be allocated to one of seven groups, and each group shall have an RMT value, as specified in Table D.5.

Table D.5 — GrandchildFactorRMFCatalog Groups

Group	Grandchild Factor Values	RMT Explicit/Implicit?
1	7, 11, 13, 14	Explicit
2	1, 2, 4, 8	Explicit
3	5, 10	Explicit
4	3, 12	Explicit
5	6, 9	Explicit
6	0	Explicit
7	15	Implicit, with RMT value of 1.0

The type of value shall be as specified in D.1.3.2.

IndexDifferenceRMFCatalog, ResidualRMFCatalog, and StreamlengthDifferenceRMFCatalog RMFs shall have 15 segments, as specified in Table D.6.

Table D.6 — 15-Segment RMFs

Segment	RMT Explicit/Implicit?
1 to 14	Explicit
15	Implicit, with RMT value of 1.0

IndexDifferenceRMFCatalog, ResidualRMFCatalog, and StreamlengthDifferenceRMFCatalog RMFs shall be specified as follows:

- They have minimum and maximum symbol values, which are integers.
- They have RMTs up to and including certain knots.
- There are 16 knot values.
- Each segment represents 0 or more symbols, starting with one knot value, and ending with the predecessor of the following knot value.
- The number of segments that can potentially be present in the payload ranges from 1 to 15, inclusive. A segment is present in the payload, only if there are symbols present in that segment; however, the presence of symbols in a segment does not imply that the segment is present in the payload with any or all of these symbols.

The RMF Selection Stage determines, from the tier for reached in s-tree assembly, as described in Clause D.3, whether the next stream to be processed is a junction-tableau or a terminal-tableau.

NOTE It might not be necessary to have two entirely distinct functional units for processing the two cases.

The RMF Selection Stage also calculates the location of the start of the stream in the bitstream by summing the streamlengths of previous tableaux, and determines the relevant tableau-set-index as governed by the s-tree assembly process.

D.2.6 RMF Selection Stage

This clause shall be skipped if the tableau's streamlength is 0.

The decoding process shall use the value of tableau-set-index to obtain the GrandchildFactorRMFCatalog RMF from the secondary header as explained in D.1.6.

NOTE 1 An RMF from a GrandchildFactorRMFCatalog is used in D.2.8.1 to decode the 4-bit grandchild factors of tableau root node and transitional node labels.

NOTE 2 Further RMFs, discussed in the current clause, allow the tail of the label of a transitional node in tableau-layer -2 to be decoded.

The decoding process shall use the value of tableau-set-index to obtain another RMF from the tertiary header, as described in D.1.6 .

In particular:

- The decoding process shall use the ResidualRMFCatalog for this tableau, if this tableau is a terminal-tableau.
- The decoding process shall use the StreamlengthDifferenceRMFCatalog for this tableau if this tableau is a junction-tableau.

The decoding process shall obtain the IndexDifferenceRMFCatalog RMF from the tertiary header (see 8.6) as follows:

- The decoding process shall use `rmf_terminal_tableau_sets` as the IndexDifferenceRMFCatalog for this tableau if this tableau is a terminal-tableau.
- The decoding process shall use `rmf_junction_tableau_sets` as the IndexDifferenceRMFCatalog for this tableau if this tableau is a junction-tableau.

D.2.7 Initialization Stage

RMF shall be set to the GrandchildFactorRMFCatalog.

OldIndex and old-streamlength shall be set to 0 in the case of a junction-tableau. Otherwise, they shall not be used.

If the bitstream is incomplete (i.e., ends before the end of the byte whose start is `bitstream_size` bytes after the start of the bitstream), then decoding of symbols in tableaux shall be possible until *truncation* is activated, see D.2.8.6, when simulation shall take over. After simulation takes over, all symbol decodings that would otherwise happen in the next stages shall be simulated as follows:

- Where decoding part of a stream to find a streamlength-difference is described (see D.2.8.3), but truncation is active, there shall be no such decoding. The Traversal Stage shall then set streamlength-difference to minus old-streamlength.
- Where decoding part of a stream to find a resid-val is described (see D.2.8.3), but truncation is active, there shall be no such decoding. The Traversal Stage shall then set resid-val to 0.
- Where decoding part of a stream to find a 4-bit code is described (see D.2.8.1), but truncation is active, there shall be no such decoding. The Traversal Stage shall then set the 4-bit code to 15 in decimal.
- All other processes for decoding the tableau (see D.2.8.1 and D.2.8.2) shall proceed as normal.

If the start of a junction-tableau is contained within an incomplete bitstream, but while decoding that junction-tableau, the range decoding process runs out of bytes required for decoding that junction-tableau, then symbols—including those already decoded—shall be simulated.

NOTE This ensures, when a junction-tableau is decoded, that either all or none of the streamlengths that are required can be decoded, thus allowing the start and end of all streams present in the tier above the tier of the junction-tableau to be calculated. (This is also vital for calculation of the start of the next tier but one, where such a tier exists.)

Simulation of this tableau shall take over immediately if the tableau's streamlength is 0, and shall cease when the tableau's traversal stage (see D.2.8) is complete.

D.2.8 Traversal Stage

D.2.8.1 Traversing Nodes According to Labels

The decoding process shall, initially or after catalog reset, decode the new tableau's 4-bit codes in depth-first pre-order traversal order, until it has found the 4-bit code at the head of the label of a tableau-layer -2 transitional node. Catalog reset is specified in D.2.8.4.

NOTE In Figure C.4 all the grandchild factors of the bitstream of a tableau are shown in a mini-tree. The first node to be traversed (tableau root node) has grandchild factor 1001 as mini-tree label. The second node to be traversed (a tableau-layer -3 transitional node) has 0100 as label. The third node to be traversed (a tableau-layer -2 transitional node) has 1011 as the head of its label and values symbolized by a, h and l following the head.

D.2.8.2 Traversing the Tail of a Tableau-Layer −2 Transitional Node Label

Up to 4 additional label decoding operations shall be performed, each one as in D.2.8.3. These shall be followed by a catalog reset.

NOTE In Figure C.4 after decoding the symbol 1011 as the head of the third label, three additional label decoding operations are performed. The first decodes the value denoted in red by letter a, the second h and the third l. After decoding 0001 at the head of the fifth label, an additional label decoding operation is performed, which decodes m, n, o and p. After decoding 0010 at the head of the sixth label, an additional label decoding operation is performed, which decodes q.

D.2.8.3 Additional Symbol Decoding Process

What follows depends upon whether the tableau is a junction-tableau or a terminal-tableau:

- In a junction-tableau, the decoding process shall repeat the following steps 4 times:

  1. Switch RMF to StreamlengthDifferenceRMFCatalog and decode a streamlength-difference.

  2. Add streamlength-difference to `old_streamlength` to result in a streamlength belonging to the tier immediately above.

  3. If streamlength > 0, then switch RMF again to IndexDifferenceRMFCatalog and decode an index-difference.

     If streamlength = 0, then skip to step 6.

     NOTE 1 Streamlength cannot be negative.

  4. Add index-difference to `old_index` to result in a tableau-set-index (relating to a tableau root node in the summit of this tableau).

  5. Update `old_index` to the calculated tableau-set-index.

6. Update `old_streamlength` to the calculated streamlength.
7. If streamlength ≠ 0, then update the element in this tableau's summit associated with the tableau-set-index and streamlength.

   If streamlength = 0, then update the element associated with the streamlength.

   NOTE 2 If the current junction-tableau is in a tier below tier -1, then this tableau-set-index will enable decoding of junction-tableaux in the tier immediately above (the tableau-set is, in this instance, a junction-tableau-set). If the current tier is tier -1, then the tableau-set is a terminal-tableau-set.

- In a terminal-tableau, the decoding process shall perform the following steps:
  1. Switch RMF to ResidualRMFCatalog and decode 4 resid-vals.
  2. Use resid-vals to update their associated elements in the tableau's summit.

D.2.8.4 Catalog Reset

Following completion of the (up to 4) additional label decoding operations (yielding either up to 16 streamlengths and up to 16 junction node tableau-set-indices, or up to 16 resid-vals) the decoding process shall reset tableau metadata RMF catalog to GrandchildFactorRMFCatalog, whatever the type of tableau. If the tableau is not yet complete, then the decoding process shall resume the traversal stage as described in D.2.8.1.

D.2.8.5 Active Area Test

A tableau shall not contain data outside its active volume, whether streamlengths or resid-vals.

NOTE 1 This is why the s-tree was defined in 9.7 to have an active volume related to the area represented by a residual-grid (as well as the hypothetical summit).

Any step described anywhere in this clause is ignored and shall not be performed where the step references outside the active volume.

NOTE 2 A consequence of this is that on occasion some (but not all) bits of a particular 4-bit code encountered by the decoding process might not affect the end result of decoding in any way.

D.2.8.6 Buffer Refresh

During the traversal stage, the symbol decoding process shall refresh by reading a new range decoding process input word (see Clause F.3) from the bitstream.

NOTE 1 Refresh is only triggered intermittently by the symbol decoding process.

If the bitstream is incomplete, (see D.2.7) and a tableau symbol is being decoded, and the decoding process needs to refresh, and this requires a byte (or a fraction of an octet) after the end of that incomplete bitstream, then truncation shall become active for that symbol and the following symbols in the same tableau.

NOTE 2 There could be the following optional feature for handling broken input streams: If the end of a terminal-tableau is after the end of the bitstream, then just before decoding each symbol of the tableau there is a check that there remain a number of bytes before the end of the bitstream.

This is called `truncation_number`. Truncation is activated the first time that the optional check fails (with the optional feature of repeatedly checking in a tableau). D.2.8.7 specifies the required response to truncation.

D.2.8.7 Response to Truncation

If truncation is activated, then the decoding process employs 0-valued pad bits, as described in Clause F.3, when the end of the bitstream (now not the end of the tableau) is encountered during decoder refresh. The fast path in Figure F.1 shall not be used once truncation has been activated. For all successors of this symbol in the same tableau, decoding shall be simulated as described in D.2.7.

D.3 Assembly of Grids from Junction-tableaux and Terminal-tableaux via Desparsification (Stratum 0)

The assembly of an s-tree of a live residual-grid shall be carried out in order of tier, from lowest to highest.

The absence of tableaux in an s-tree in the bitstream payload is specified in the tertiary header (see Table 22, `residual_grid_flags`) for a particular echelon of a particular plane stack. In this case, the residual-grid is a grid of zeroes, and the remainder of D.3 shall not apply.

The grafting function $graft(S, T, ndS, (xr, yr))$, which replaces a tableau-layer 0 node, ndS, of a primary s-tree, with nodes of a secondary s-tree that is a tableau, is defined, as follows, in Formula (D.1):

$$U = graft(S, T, ndS, (xu, yu)) \tag{D.1}$$

where (defining the output U)

U is the s-tree of structure *s-tree*$[L, (xu, yu), U\_type]$, obtained by replacing ndS with the tableau root node of T and replicating all the links in T and all the descendants of the tableau root node of ;

where (defining the inputs S, T, ndS and (xr, yr))

S is an s-tree of structure *s-tree*$[K, (xs, ys), S\_type]$;

T is a tableau of structure *tableau*$[(xt, yt), T\_type]$;

ndS is a tableau-layer 0 node of S that has a label;

(xu, yu) is the dimensions tuple of U;

where

K is the number of layers in S;

L equals K, unless ndS is the final tableau-layer 0 node, in which case $L = K + 4$;

(xs, ys) is the dimensions tuple of S;

(xt, yt) is the dimensions tuple of T, which shall be $(16,16)$ unless ndS is at the right or bottom edge of the residual-grid of S, where it shall be reduced, but only as much as is necessary to have all nodes of U within the residual-grid of U;

S_type is the label type of S;

T_type is the label type of T;

U_type is T_type for a label in tableau-layer 0 of U and is S_type for a label in tableau-layer -4 of U.

Graft is a one-way function because the label on ndS is discarded.

Once an s-tree has been assembled a number of times to comprise tiers of tableaux, from root-tableau up to and including junction-tier t junction-tableaux, the tableaux of tier $t + 1$ shall be decoded. When all the relevant tableaux have been decoded, they are together grafted to the s-tree to add 4 layers, hence assembling it again.

Labels in an assembled s-tree of rise $K + 4$, replacing the primary s-tree of rise K, shall be associated only with the nodes at altitude 0 of the s-tree of rise $K + 4$.

The active volume and rise R of the intended final s-tree (root-tableau to terminal-tier 0) are calculated prior to the retrieval of any tableaux or tiers, as they are fixed properties of a residual-grid.

Prior to decoding the tier $t + 1$ tableaux of an s-tree of rise R, a complete s-tree of rise $R - (4 * |t|)$ shall already have been calculated.

NOTE 1 For example, prior to decoding terminal-tier 0, a complete s-tree of rise $R - 4$ would already have been calculated.

The active volume of the s-tree of rise R is known as the final active volume.

The summit of the s-tree of rise R is known as the final summit. This summit is a grid of dimensions 2^R by 2^R, where the elements of this grid shall be resid-vals.

NOTE 2 The active volume is a subset of the summit, and can be equal to the summit. The final active volume is a subset of the final summit, and can be equal to the final summit.

The same summit may be represented in a coarser grid of dimensions $2^{(R-(4*|t|))}$ by $2^{(R-(4*|t|))}$, where the elements of the coarser grid shall themselves be grids of dimensions $2^{(R-(4*|t|))}$ by $2^{(R-(4*|t|))}$, where the elements shall be resid-vals.

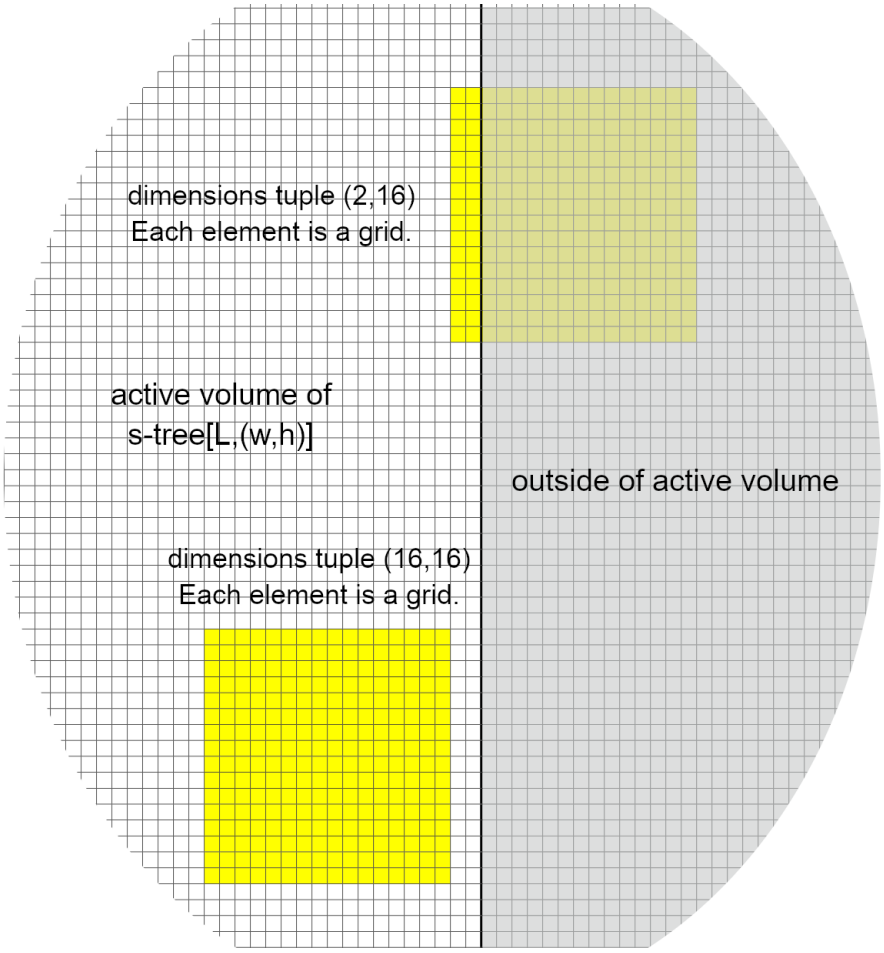


Figure D.5 — Two elements of a coarser grid superimposed on an original grid

Figure D.5 shows part of an original grid containing two elements, each shown in yellow, of a coarser grid,. Each of the two elements of the coarser grid contains 256 elements representing a square region of the original grid. The coarser grid need not be represented explicitly in a decoding process, but is a useful aid in describing the decoding process.

NOTE 3 Only the relevant area of the original grid is shown in Figure D.5. The elements of the coarser grid are themselves grids. The highlighted black line marks the rightmost column that is still within the width W of the residual-grid represented by *s-tree*$[L, (W,H),\text{labeltype}]$.

The dimensions tuple of a tableau associated with the leftmost yellow square shall be $(16,16)$, because the whole of the square is to the left of the highlighted vertical black line. The dimensions tuple of a tableau associated with the rightmost of the two yellow squares shall be $(2,16)$.

A final grid is that with W as its grid width, H as its grid height, and dimensions no greater than 2^R by 2^R.

The layer at altitude 0 of the final active volume is mapped into a 2D grid of elements that are rectangular subsets of a final grid, each of maximum dimensions $2^{(R-(4*|t|))}$ by $2^{(R-(4*|t|))}$.

The definition of ordered full 4-tree (quadtree, as specified in this document) ensures that the appropriate mapping is a recursive application of the Morton map. The Morton map simply maps “top-left” to a copy of a top-left rectangle of the original grid, etc.

Each row or column of the grid is kept as short as is possible, by ignoring square elements that are entirely outside the final active volume. Elements in the bottommost and rightmost columns have actual dimensions reduced, where necessary, in order that they end at the corresponding edges of the layer at altitude 0 of the final active volume.

Each tableau in tier $t + 1$ shall correspond to one of these grid elements. The correspondence is made by:

- associating the root of the existing s-tree with the summit of the final s-tree, still of undetermined sparsity structure.
- recursively identifying quarters of the residual-grid, by means of the Morton map during a depth-first pre-order traversal of the existing s-tree.

NOTE 4 If a tableau-layer 0 node becomes associated with an area that overlaps with one of the elements of the grid, then the next tableau in the bitstream will be associated with that element.

For example, prior to decoding terminal-tier 0, the nominal element dimension will be $2^{(R-4)}$ by $2^{(R-4)}$.

The active volume of such a tableau is determined by the dimensions of the corresponding grid elements. If an element has actual dimensions p by q, then the tableau active volume at its tableau-layer 0 is, as follows, in Formula (D.2):

$$(tableau\ width, tableau\ height) = (\lceil p * 16^{(t+1)} \rceil, \lceil q * 16^{(t+1)} \rceil) \quad \text{(D.2)}$$

where the modified square brackets denote ceiling.

For example, taking $t = -1$, the active volume at tableau-layer 0 of a terminal-tableau will simply be p by q.

The grids involved in the s-tree assembly process are calculated as follows:

Given an input s-tree of rise $R - (4 * |t|)$, where

- R is the rise specified for the final s-tree with all tiers grafted
- $|t|$ is the number of tiers yet to be grafted
- t is the tier index of the latest tier already grafted,

a final summit, of the s-tree of rise R, can be partitioned into grids of dimensions $2^{(4*|t|)}$ by $2^{(4*|t|)}$, that are the elements of a containing grid of dimensions $2^{(R-(4*|t|))}$ by $2^{(R-(4*|t|))}$.

As detailed in this clause, there is a three-way correspondence between:

- each junction node in tableau-layer 0 of the input s-tree
- a corresponding rectangular subset of the final grid
- a corresponding tableau in the new tier

From this correspondence, the dimensions tuple (at most, 16×16) of each tableau in the new tier, corresponding to a junction node and to a rectangular subset of the final grid, can be calculated from Formula (D.2) as:

$$\lceil p * 16^{(|t+1|)} \rceil \; by \; \lceil q * 16^{(|t+1|)} \rceil$$

where

p by q are the actual dimensions of the corresponding rectangular subset of the final grid.

The s-tree assembly process adds the new tier as follows:

Calculate the dimension tuples of the tableaux corresponding to each junction node in tableau-layer 0 of the input s-tree.

Get tableau-set-indices of tableaux in the tier $t + 1$: i.e., the labels of the junction nodes that are in tableau-layer 0 of the input s-tree.

Decode new tier into a queue of tableaux. Ensure that the decoding process always uses the appropriate tableau-set-indices.

For each tableau-layer 0 node of the s-tree:

If it is not marked as a subterminal leaf node, then:

Get next tableau.

Replace node with a strict subtree, namely the tree represented by that tableau (i.e., graft a tableau).

The junction node of the tableau becomes a tableau root node in the s-tree built by the s-tree assembly process.

End with s-tree of rise $R + (4 * |(t + 1)|)$.

Marking a tableau-layer 0 node of a tableau as a subterminal leaf node shall occur during decoding of a junction-tableau, for one of the following reasons:

- Either the node was outside the active volume of the tableau; or,
- The node label was decoded to show a streamlength of 0.

The same s-tree assembly process shall be repeated until there is an s-tree with rise R. This s-tree's $\mathrm{labeltype}$ parameter shall be resid-val, whereas previous s-trees shall have other $\mathrm{labeltype}$ parameters (which could differ between s-trees of rise $R - 4$ and s-trees of rise less than $R - 4$).

From the final s-tree of rise R, the residual-grid shall be extracted by initializing every resid-val to 0 and then, for those nodes at altitude 0 that are within the final active volume, the decoding process shall copy the resid-val (that was decoded from its label when terminal-tableaux were decoded) to the correct 1 by 1 element of the residual-grid.

For all resid-val values copied to a row and column of the residual-grid, the tableau-set-index used in decoding the resid-val (see the s-tree assembly process earlier in this clause, D.3) shall be copied to the same row and column of the terminal-tableau-set-index-grid of Clause D.4.

The values in the terminal-tableau-set-index-grid need not be initialized.

D.4 Endpoint of Stratum 0 and Input for Stratum 1

At the endpoint of stratum 0, a decoding process shall meet the following conditions:

- At the endpoint of stratum 0, a fully populated non-sparse residual-grid (quantized directional component), and a fully populated non-sparse terminal-tableau-set-index-grid exist for each echelon (i.e., for A, H, V, and D directions) of each plane stack.
- At the endpoint of stratum 0, those residual-grids without any streams in the bitstream are also available, though filled with zeros.

D.5 Residual-Grid Transformations (Stratum 1)

D.5.1 Dequantization

D.5.1.1 Quantization Parameters Tuple

A quantization parameters tuple is represented as a 2-tuple, containing the stepwidth and the dequantization offset (see D.1.3.3).

The decoding process shall obtain a value of tableau-set-index for each resid-val in a residual-grid by reading it from the corresponding position in an associated TerminalTableausetIndexGrid.

The decoding process shall use this tableau-set-index to obtain the quantization parameters tuple from the tertiary header (see 8.6).

The decoding process uses the `quantization_parameters_tuple` catalog item specified in D.1.6 as the quantization parameters tuple for a resid-val belonging to a terminal-tableau of a given terminal-tableau-set.

NOTE The TerminalTableausetIndexGrid maintains the tableau-set-indices from Stratum 0 for use in Stratum 1.

D.5.1.2 Precise Dequantization Process

This process shall receive as input the quantized resid-val, q, the stepwidth, sw, and the dequantization offset, ofs (see D.5.1.1). It shall output the dequantized resid-val, uq, computed, as follows, in Formula (D.3):

$$uq = q * sw - sgn(q) * ofs \tag{D.3}$$

where $sgn(q)$ denotes the sign of q, defined, as follows, in Formula (D.4):

$$sgn(q) = \begin{cases} 1, \text{if } q > 0 \\ 0, \text{if } q = 0 \\ -1, \text{if } q < 0 \end{cases} \tag{D.4}$$

D.5.2 Composition Transform

D.5.2.1 Composition Transform Structure and Variants

The composition transform shall receive as input the values allocated to four directions, as specified in Table D.7.

Table D.7 — Inputs to Composition Transform

Direction	Input Value
average	D_A
horizontal	D_H
vertical	D_V
diagonal	D_D

The transform shall output the recovered resid-vals, specified as $x00$, $x01$, $x10$, $x11$.

There are two composition transform variants, as described in D.5.2.2 and D.5.2.3, the choice of which is specified in B.4.

D.5.2.2 Standard Composition Transform

Standard composition transform is defined, as follows in Formulae (D.5), (D.6), (D.7), and (D.8):

$$x00 = (D_A + D_H + D_V + D_D + 2) \gg 2 \quad \text{(D.5)}$$

$$x01 = (D_A - D_H + D_V - D_D + 2) \gg 2 \quad \text{(D.6)}$$

$$x10 = (D_A + D_H - D_V - D_D + 2) \gg 2 \quad \text{(D.7)}$$

$$x11 = (D_A - D_H - D_V + D_D + 2) \gg 2 \quad \text{(D.8)}$$

D.5.2.3 Integer Composition Transform

Integer composition transform consists of two steps, as described as follows:

Step 1

This step processes the input values, D_A, D_H, D_V, and D_D (see D.5.2.1), which are resid-vals. This step distinguishes between the four inputs from the core-echelons versus those from higher echelons, as follows:

For the core-echelons:

The composition transform shall modify the four input value, as follows, in Formulae (D.9), (D.10), (D.11), and (D.12):

$$D'_D = D_D \quad \text{(D.9)}$$

$$D'_V = 2 * D_V + (D_D \,\%\, 2) \quad \text{(D.10)}$$

$$D'_H = 2 * D_H + (D_D \,\%\, 2) \quad \text{(D.11)}$$

$$D'_A = 4 * D_A + (D'_H + D'_V - D'_D) \,\%\, 4 \quad \text{(D.12)}$$

For echelons other than the core-echelons:

The transform shall modify the four input values, as follows, in Formulae (D.13), (D.14), (D.15), and (D.16).

$$D'_D = D_D \tag{D.13}$$

$$D'_V = 2 * D_V + (D_D \,\%\, 2) \tag{D.14}$$

$$D'_H = 2 * D_H + (D_D \,\%\, 2) \tag{D.15}$$

$$D'_A = 4 * D_A + rem - t \tag{D.16}$$

where, as specified in Formula (D.17):

$$rem = (D'_H + D'_V - D'_D) \,\%\, 4 \tag{D.17}$$

and, as specified in Formula (D.18):

$$t = \begin{cases} 4, \; if \; A \leq 0 \; AND \; rem > 0 \\ 0, \; otherwise \end{cases} \tag{D.18}$$

Step 2
In this step the transform recovers the resid-vals using the modified values, D'_A, D'_H, D'_V, and D'_D, as follows, in Formulae (D.19), (D.20), (D.21), and (D.22):

$$x00 = (D'_A + D'_H + D'_V + D'_D) \gg 2 \tag{D.19}$$

$$x01 = (D'_A - D'_H + D'_V - D'_D) \gg 2 \tag{D.20}$$

$$x10 = (D'_A + D'_H - D'_V - D'_D) \gg 2 \tag{D.21}$$

$$x11 = (D'_A - D'_H - D'_V + D'_D) \gg 2 \tag{D.22}$$

D.5.3 Dequantized Residual-grid Composition Process

D.5.3.1 Dequantized Residual-grid Composition Process Overview

The four decoded quantized residual-grids, corresponding to the directions, A, H, V, and D, shall be dequantized (see D.5.1), and the corresponding unquantized grids, $uqGrid_A$, $uqGrid_H$, $uqGrid_V$, and $uqGrid_D$, shall be produced, for use as inputs to the composition process.

The output of the process shall be a composed residual-grid.

Except when composing the core-echelons, the composition process inputs also include a $ModifiedPredictedAverageGrid$ of the same echelon index as the other inputs.

Computation of $ModifiedPredictedAverageGrid$ is not the responsibility of Stratum 1. It shall be chosen by the system that invokes Stratum 1 and the resulting data will be available to use in Stratum 1 when required (see D.5.3.2).

NOTE The system invoking Stratum 1 is Stratum 2 if the decoding process is conformant at Stratum 2. Stratum 2 invokes the (Stratum 1) Composition process (of this D.5.3) for one echelon at a time. Each invocation occurs after choosing a $ModifiedPredictedAverageGrid$ according to D.7.5.

The composition process is a composition transform applied multiple times. The composition process distinguishes between core-echelons and other echelons, as described in D.5.3.3 and D.5.3.2.

D.5.3.2 Composition Process for Echelons Other than the Core-echelons

1. Each decoded quantized residual-grid, and each corresponding unquantized grid, shall be a grid of dimensions (W^{ech}, H^{ech}), where ech is echelon index.
2. A correction, of dimensions (W^{ech}, H^{ech}), shall be applied to grid, $uqGrid_A$, as follows, in Formula (D.23):

$$uqCorrectedGrid_A[i,j] = uqGrid_A[i,j] + ModifiedPredictedAverageGrid[i,j] \quad \text{(D.23)}$$

3. The composition transform shall be called (see D.5.2), operating on all 4-tuples, T_{ij}, where as specified in Formula (D.24):

$$Tij = (uqCorrectedGrid_A[i,j], uqGrid_H[i,j], uqGrid_V[i,j], uqDGrid_D[i,j]) \quad \text{(D.24)}$$

where

$$i = 0, 1, \ldots, H^{ech} - 1$$

$$j = 0, 1, \ldots, W^{ech} - 1$$

and shall output all 4-tuples C_{ij}, as defined, as follows, in Formula (D.25):

$$C_{ij} = (C00[i,j], C01[i,j], C10[i,j], C11[i,j]) \quad \text{(D.25)}$$

This way, four grids of dimensions (H^{ech}, W^{ech}), namely $C00$, $C01$, $C10$, and $C11$, are computed.

4. The composed grid, $ComposedResidualGrid^{ech}$, shall be a grid of dimensions $(2 * W^{ech}, 2 * H^{ech})$, which is created by assembling $C00$, $C01$, $C10$, and $C11$ into a single grid as follows, in Formulae (D.26), (D.27) (D.28), and (D.29):

$$ComposedResidualGrid^{ech}[2 * i, 2 * j] = C00[i,j] \quad \text{(D.26)}$$

$$ComposedResidualGrid^{ech}[2 * i, (2 * j) + 1] = C01[i,j] \quad \text{(D.27)}$$

$$ComposedResidualGrid^{ech}[(2 * i) + 1, 2 * j] = C10[i,j] \quad \text{(D.28)}$$

$$ComposedResidualGrid^{ech}[2 * i + 1, (2 * j) + 1] = C11[i,j] \quad \text{(D.29)}$$

where

$$i = 0, 1, \ldots, H^{ech} - 1$$

$$j = 0, 1, \ldots, W^{ech} - 1$$

D.5.3.3 Composition Process for the Core-echelons

The composition process for the core-echelons shall include only the steps 1, 3 and 4 of the process followed for higher echelons (see D.5.3.2).

D.6 Endpoint of Stratum 1 and Input for Stratum 2

At the end of stratum 1 in the decoding process, there are now fewer ComposedResidualGrids than residual-grids. Each ComposedResidualGrid has been reconstituted from residual-grids of directions, A, H, V, and D, present in the original bitstream, with the exception of any direction that is entirely void of terminal-tableaux.

NOTE At this stage in the decoding process, dequantization has already been completed.

Values have 2 additional fractional bits relative to the fractional bits specified in the element descriptor, assuming standard decoding mode is in use.

D.7 Plane Reconstruction Process (Stratum 2)

D.7.1 Plane Stack Reconstruction (Stratum 2) Overview

Each plane stack shall be reconstructed from core-echelon to echelon 0. Stratum 2 is responsible for invoking the composition processes of Stratum 1 for each echelon index in turn, once the necessary input data is available.

The composed residual-grid shall be subtracted from the predicted-plane (see D.7.2) to create the resultant-plane at that echelon index.

NOTE The composed residual-grid, predicted-plane, and the resultant-plane all belong to the same echelon index, within a specific plane stack.

Before a resultant-plane is exported by the decoding process, but not when it is used as input to other calculations within Stratum 2, the fractional bits shall be set to zero. Furthermore, the integer part shall be rounded towards minus infinity and shall be clamped to be representable by the integer part of the element descriptor.

D.7.2 Plane Prediction via Upsampling

A predicted-plane, $PredictedPlane^{ech}$, of dimensions $(2 * W^{ech}, 2 * H^{ech})$, shall be computed by upsampling (see Annex G) the resultant-plane of echelon index $ech - 1$, $ResultantPlane^{ech-1}$ (see D.7.3), where ech is echelon index of the predicted-plane.

$ResultantPlane^{ech-1}$ shall be of dimensions $(2 * W^{ech-1}, 2 * H^{ech-1})$,

where, as specified, in Formulae (D.30) and (D.31):

$$W^{ech} = 2 * W^{ech-1} \tag{D.30}$$

$$H^{ech} = 2 * H^{ech-1} \tag{D.31}$$

The type of upsampling shall be determined from the primary header (see 8.4 and 8.8.3).

In the case of the core-echelon, $PredictedPlane$ shall be defined as a grid of zeros, as follows, in Formula (D.32):

$$PredictedPlane[i,j] = 0 \tag{D.32}$$

where

$i = 0, 1, \ldots, (2 * H^{ech-1})$

$j = 0, 1, \ldots., (2 * W^{ech-1})$

D.7.3 Resultant-Plane Reconstruction at an Echelon Index

The resultant-plane of each echelon index, $ResultantPlane^{ech}$, is a grid of dimensions $(2 * H^{ech}, 2 * W^{ech})$, computed, as follows, in Formula (D.33):

$$ResultantPlane^{ech}[i,j] = PredictedPlane^{ech}[i,j] + ComposedResidualGrid^{ech}[i,j] \tag{D.33}$$

(see D.5.3 for the definition of $ComposedResidualGrid^{ech}$ and D.7.2 for $PredictedPlane^{ech}$.)

The resultant-plane shall also have its integer part clamped to be representable by the integer part of the element descriptor.

D.7.4 PredictedPlane Decomposition

The predicted-plane PredictedPlane^{ech} shall be decomposed to four grids of dimensions (W^{ech}, H^{ech}), namely $Pr00$, $Pr01$, $Pr10$ and $Pr11$, as follows, in Formulae (D.34), (D.35), (D.36), and (D.37):

$$Pr00[i,j] = PredictedPlane^{ech}\,[2 * i, 2 * j] \tag{D.34}$$

$$Pr01[i,j] = PredictedPlane^{ech}\,[2 * i, (2 * j) + 1] \tag{D.35}$$

$$Pr10[i,j] = PredictedPlane^{ech}\,[2 * i + 1, 2 * j] \tag{D.36}$$

$$Pr11[i,j] = PredictedPlane^{ech}\,[2 * i + 1, (2 * j) + 1] \tag{D.37}$$

where

$i = 0, 1, \ldots, H^{ech-1}$

$j = 0, 1, \ldots, W^{ech-1}$

D.7.5 ModifiedPredictedAverageGrid

A correction grid of dimensions (W^{ech}, H^{ech}), $ModifiedPredictedAverageGrid$, shall be formed, as follows, in Formula (D.38):

$$ModifiedPredictedAverageGrid\,[i,j] = CA[i,j] - CB[i,j]) \tag{D.38}$$

where, as specified in Formulae (D.39) and (D.40):

$$CA[i,j] = Pr00[i,j] + Pr01[i,j] + Pr10[i,j] + Pr11[i,j] \tag{D.39}$$

$$CB[i,j] = 4 * ResultantPlane^{ech-1}[i,j] \tag{D.40}$$

D.8 Grid Dimensions

D.8.1 Dimensions of Resultant-Planes

A resultant-plane, of specific echelon index, in a specific plane stack, shall be a grid of dimensions (w^{ech}, h^{ech}).

Dimensions shall be defined recursively as follows, in Formulae (D.41) and (D.42):

$$w^{ech-1} = (w^{ech} + 1)\,//\,2 \tag{D.41}$$

$$h^{ech-1} = (h^{ech} + 1)\,//\,2 \tag{D.42}$$

with w^0, h^0 being available in the secondary header (see 8.5).

The plane reconstruction process (see D.7.3) shall ignore any columns/rows exceeding the echelon resultant-plane dimensions defined in D.8.1.

NOTE Exceeding is possible due to ceiling operations included in the calculations of residual-grid dimensions (see D.8.2).

D.8.2 Residual-grid Dimensions

A residual-grid of a single echelon, of given direction, shall be a grid of dimensions (W^{ech}, H^{ech}), with, as specified in the following Formulae (D.43) and (D.44):

$$W^{ech} = (w^{ech} + 1)\,//\,2 \tag{D.43}$$

$$H^{ech} = (h^{ech} + 1)\,//\,2 \tag{D.44}$$

D.9 Endpoint of Stratum 2

At this stage in the decoding process, a resultant-plane has been exported by the decoding process.

The decoding process is suitable for reconstructing and exporting resultant-planes in all selected plane stacks at all echelons up to a selected echelon.

NOTE For a selected echelon, a raw file containing one or more resultant-planes, from all plane stacks, in order, to the bit-depth of the integer portion of the element descriptor, could, in principle, be produced after this stage. For example, with 3 components, this standard neither requires nor prevents output in a planar format, and optionally padding an 8-bit or 10-bit value, for example, to 1 or 2 bytes.

Annex E
(Normative)

Histogram Formats

E.1 Constraints on Permitted Histograms

Knots in the bitstreams shall comply with the following constraints:

- A histogram includes 16 knots, with monotonically increasing values, representing 15 line-segments.
- Each histogram bin contains all integers from the value of a knot to one less than the value of the succeeding knot.
- At least one knot in a histogram is not zero.
- Any knot of value zero, immediately or otherwise succeeding a non-zero knot, is replaced by 1 + the max-representable-integer-in-the-header.
- Other knots of value zero are retained at value zero.

RMT values shall comply with the following constraints:

- A stage 1 histogram includes 5 (in the case of GrandchildFactorRMFCatalog) or 14 (in the case of IndexDifferenceRMFCatalog, StreamlengthDifferenceRMFCatalog, or ResidualRMFCatalog) explicit RMT values, preceded by an implicit zero and followed by an implicit zero.
- At least one RMT value in a histogram is not zero.
- Any RMT value of zero, immediately or otherwise succeeding a non-zero RMT value, is replaced by 1.0, i.e., 100%.
- Other RMT values of zero are retained at value zero.

If stage 1 decodes to a specific histogram bin, then that bin shall contain at least one integer.

E.2 StreamlengthDifferenceRMFCatalog Histograms

NOTE 1 From D.1.6, these Histograms are not used in the terminal-tier.

Two streamlengths histograms are used in the decoding of junction-tableaux using StreamlengthDifferenceRMFCatalog in Clause D.2.

This histogram format contains data necessary for the range decoding process to decode a streamlength-difference in two stages.

The first stage shall use StreamlengthDifferenceRMFCatalog to derive the RMTs of 15 ordered symbols. The first 14 RMT values shall come from the fractional16[14] array (see D.1.5) and the final shall be 1.0. The range decoding process, based on these 15 RMT values shall return the position p, satisfying $0 \leq p < 15$, of one of the ordered symbols.

NOTE 2 Identification by name of these 15 individual symbols is left to the user. They are not grandchild factors, but are bins of the first decoding stage, described in Clause E.4, where each bin represents several ordered symbols of the second, streamlength-differences decoding stage.

The knot[16] array derived from the StreamlengthDifferenceRMFCatalog is referred to in the next paragraph as x.

In the second decoding stage, a new uniform RMF shall be derived from x[p] and x[p + 1], whose symbols shall be the actual streamlength-differences ranging from x[p] to x[p + 1] − 1. The number of RMT values needed for range decoding to select one of these symbols shall be m = x[p+1] − x[p]. The RMT values shall be 1.0 / m, 2.0 / m, ….. (m − 1) / m, and 1.0, where the fractions shall be rounded to an accuracy of 16 fractional bits.

The range decoding process shall further select the position, p2, of the symbol in the histogram. The final decoded streamlength-difference value returned by the process is therefore x[p] + p2.

E.3 IndexDifferenceRMFCatalog Histograms

NOTE From D.1.6, these histograms are not used in the terminal-tier.

The format shall be common to Clause E.2, except that the words StreamlengthDifferenceRMFCatalog and streamlength-difference are replaced by IndexDifferenceRMFCatalog and index-difference, respectively.

Depending upon the tier, the decoded tableau-set-index can index a terminal-tableau-set or a junction-tableau-set.

E.4 GrandchildFactorRMFCatalog Histograms

NOTE From D.1.6, these histograms are used in all tiers.

This histogram format shall be used for GrandchildFactorRMFCatalog. It enables the range decoding process to decode the grandchild factors in labels of mini-tree nodes in either 1 or 2 stages. The result of the first stage shall determine whether the second stage is applicable.

The first stage shall use GrandchildFactorRMFCatalog to derive the RMTs of 7 unnamed and ordered symbols. The first 6 RMT values shall come from the fractional8[6] array (see D.1.3.2) and the final value shall be 1.0. The range decoding process, based on these 7 RMTs shall return the position p where 0 ≤ p < 7, of one of the unnamed symbols.

The 7 unnamed symbols narrow the grandchild factor, in decimal, down to a member of one of the following 7 groups of integers, according to the unnamed symbol's position, p: [7, 11, 13, 14], [1, 2, 4, 8], [5, 10], [3, 12], [6, 9], [0] and [15], in this order, from D.2.5 The integers represent 4-bit unsigned binary numbers, as shown in Table E.1.

EXAMPLE

If the value of p is returned as 4, then the integer will be 6 or 9 in decimal, representing that the grandchild factor in binary will be 0110 or 1001.

If the value of p is returned as 6, then the integer will be 15 in decimal, representing that the grandchild factor in binary is 1111.

Table E.1 — Grandchild Factor RMF Bins

Bin #	# of Distinct Grandchild Factors Contributing to Bin	# of Bits that are 1 in Each Grandchild Factor Contributing to Bin	Grandchild Factors Included in this Bin	Decimal Value of Grandchild Factors
0	4	3	0111 1011 1101 1110	7 11 13 14
1	4	1	0001 0010 0100 1000	1 2 4 8
2	2	2	0101 1010	5 10
3	2	2	0011 1100	3 12
4	2	2	0110 1001	6 9
5	1	0	0000	0
6	1	4	1111	15

If either member of the final pair of the above sets is selected, then there is no need for a second stage, and 0 or 15 shall be the final range decoded integer. The integer 0 shall be reserved, and is not therefore seen in this standard's bitstreams.

If either member of the first pair of the above sets is selected, then there shall be a second stage, using RMT values representing 0.25, 0.5, 0.75 and 1.0. In this case, the range decoding process shall return the further position p2 where $0 \leq p2 < 4$, in this histogram. The final decoded grandchild factor shall return s[p2] where s is the selected set of integers.

If either member of the second pair of the above sets is selected, then there shall be a second stage, using RMT values representing 0.5 and 1.0. The range decoding process shall return the further position, p3 where $0 \leq p3 < 2$, in this histogram. The final decoded grandchild factor shall be returned as s[p3] where s is the selected set of integers.

E.5 ResidualRMFCatalog Histograms

NOTE From D.1.6, these histograms are not used in junction-tiers.

The format shall be the same as in Clause E.2, except that the words "StreamlengthDifferenceRMFCatalog" and "streamlength-difference" are replaced by "ResidualRMFCatalog" and "resid-val" respectively.

Annex F
(Normative)

Symbol Decoding

F.1 Range Decoding Algorithm

The processes shown in Figure F.1 are specified in Clauses F.2 through F.7.

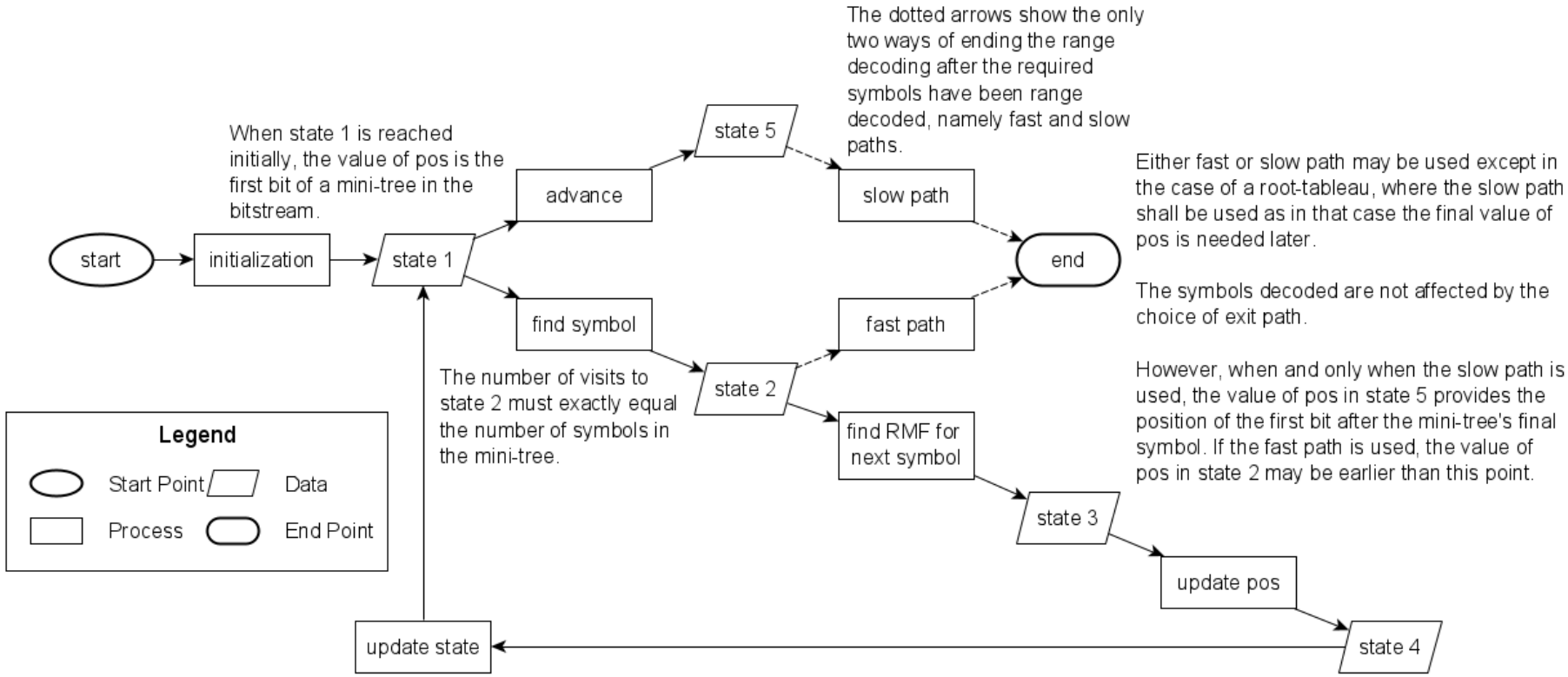


Figure F.1 — Range decoder states and processes

F.2 Initialization

The initial RMF is the range mapping function for the first symbol in a particular stream.

RANGE_SCALE_FACTOR is a constant integer with value 2^{16}.

The initial value of the variable, pos, shall be the bit position within the bitstream, of the first bit in a particular stream. The first bit is the most significant bit of a stream.

The initial value of low shall be 0.

The initial value of range shall be $2^{32} - 1$.

NOTE In this standard, each tableau that is present in the bitstream payload is represented by a mini-tree in the form of a stream of bits.

F.3 Find Symbol

The find symbol process shall involve the following sequence of steps:

1. Given the length of the bitstream, S, and the position pos, the min(32, S − pos) bits beginning at position pos in the bitstream are selected, followed by sufficient 0-valued bits to make a total of 32 bits, representing a 32-bit unsigned binary value, which is allocated to the integer variable denoted by code.

2. An encoded_numerator in the range from 0 to $2^{16} - 1$ is calculated as follows: encoded_numerator = (code − low) // (range // RANGE_SCALE_FACTOR).

3. A symbol, and associated RMT value, selected_rmt, are found, by selecting, from the current RMF, the only RMT value with (selected_rmt * RANGE_SCALE_FACTOR) less than or equal to encoded_numerator and (succeeding_rmt * RANGE_SCALE_FACTOR) greater than encoded_numerator. The symbol having that RMT is the decoded symbol, and the succeeding_rmt is the RMT of the succeeding symbol in the same RMF.

F.4 Find RMF for Next Symbol

If the decoded symbol is not the final symbol in the stream, then the RMF shall be set to the RMF for the still undetermined next symbol in the stream. Otherwise, the RMF shall become undefined.

NOTE In this standard, whether there is a next symbol and, if so, the type of the next symbol, depend on the already decoded symbols of the stream. As well as depending upon the type of the next symbol, the next RMF also depends on the properties of the tableau that a stream is representing. For example, a specific terminal-tableau-set can specify properties of a terminal-tableau.

F.5 Update Pos

The update pos process shall involve the following sequence of steps:

1. The current value of range is divided by RANGE_SCALE_FACTOR

2. Using selected_rmt and succeeding_rmt, variables are further altered as follows:

   low is first increased by range * selected_rmt

   range is then increased by range * (succeeding_rmt − selected_rmt)

3. (low + range) % (2^{32}) is calculated and written to an unsigned binary representation, padded to 32 bits, REP1.

4. Low is written to an unsigned binary representation, padded to 32 bits, REP2.

5. The number of leading bits that match in REP1 and REP2 is allocated to an integer variable, N.

6. Range is written to REP3, where REP3 is an unsigned binary representation, padded to 32 bits, REP3. The bits of REP3 are shifted to the left by N bits, inserting 0-valued bits on the right and dropping overflows to the left, to give REP4. The bits of REP4 are shifted to the right by 16 bits, inserting 0-valued bits on the left and dropping underflows to the right, to give REP5.

7. If REP5 represents an integer different from zero:

        pos is increased by N

    Otherwise

        pos is increased by N + 16

F.6 Update State

The update state process shall be as follows:

- If REP5 represents an integer different from zero:
    - The bits of REP2 are shifted to the left by N bits, inserting 0-valued bits on the right and dropping overflows to the left. The result as an integer is the new value of low.
    - The bits of REP3 are shifted to the left by N bits, inserting 0-valued bits on the right and dropping overflows to the left. The result as an integer is the new value of range.
- If REP5 represents zero:
    - The bits of REP2 are shifted to the left by (N + 16) bits, inserting 0-valued bits on the right and dropping overflows to the left. The result as an integer is the new value of low.
    - The result $(2^{32} - \text{low}) \% (2^{32})$ becomes the new value of range.

As a consequence of the update state process, the most significant bit of REP2 would be 0, if N is less than 32.

F.7 Advance

If N is 32, then the range decoder process shall move directly to its next state, without following the steps specified in the rest of Clause F.7.

If N is less than 32, then the N least significant bits of REP2 shall be removed, to leave a sequence of 32 − N bits, and, as a consequence of the update state process, the most significant bit of the sequence is 0.

If there is a second 0 in the sequence, then pos shall now be increased by the inclusive number of bits from the first 0 to the second 0. Otherwise, pos shall increase by 1.

If there is a further 1 bit immediately after a second 0 bit in the sequence, then pos shall further increase by 1.

NOTE The annotations slow path and fast path in Figure F.1 are without effect, except to indicate that either of these two options can be taken, depending upon whether an accurate pos is needed after the end of the range decoding process.

Annex G
(Normative)

Standard Upsamplers

G.1 Standard Linear Upsamplers

G.1.1 Standard Linear Sampler Overview

A standard linear upsampler is characterized by an even-sized, $s \times s$ kernel coefficient array $K[y, x]$, and its origin o. An upsampler is described herein as linear when the interpolated value at a position is a multilinear function of original discrete values. This property still allows nonlinear dependence of the interpolated value on its position.

Kernel coefficients shall be fixed point, and as in kernel-based picture filtering the value of component $p_l[y, x]$ in the upsampled grid shall be computed, as follows, in Formula (G.1):

$$p_l[y,x] = \sum_{j=0}^{s-1} \sum_{i=0}^{s-1} p_{l+1}\left[(y \,//\, 2) + d(y) - o + j, (x \,//\, 2) + d(x) - o + i\right] * K[f(d(y), j, s), f(d(x), i, s)] \tag{G.1}$$

where:

s is the size of the kernel;

$o_x = o_y = o = (s \,//\, 2)$ is the origin;

$d(x)$ is 0 if x is even and is 1 otherwise;

$f(dx, i, s)$ is i if dx is 0 and is $s - 1 - i$ if dx is 1;

l is the echelon index, therefore p_l is the upper and target grid, and p_{l+1} is the lower grid;

K is the 2D kernel array;

where

the results are rounded towards the nearest representable number, with numbers that are half-way between representable numbers rounded towards minus infinity;

and where

when reading outside of the valid frame of the lower grid, the values of the outermost valid column or row are used.

The standard linear upsamplers used in this standard are as follows:

- Bicubic upsampler (see G.1.2)
- Sharp upsampler (see G.1.3)

Each has kernel size $s \times s = 4 \times 4$ and origin $[o_x, o_y] = [2,2]$.

G.1.2 Bicubic Upsampler

Bicubic coefficients, normalized to 256, shall be as specified in Table G.1, where c is the horizontal dimension, and r the vertical dimension:

Table G.1 — Bicubic Coefficients (normalized to 256)

$K[r,c]$		c			
		0	1	2	3
r	0	0	−2	−6	1
	1	−2	15	54	−5
	2	−6	54	193	−19
	3	1	−5	−19	2

G.1.3 Sharp Upsampler

Sharp coefficients, normalized to 256, shall be as specified in Table G.2, where c is the horizontal dimension, and r the vertical dimension:

Table G.2 — Sharp Coefficients (normalized to 256)

$K[r,c]$		c			
		0	1	2	3
r	0	2	−4	−11	3
	1	−4	18	46	−6
	2	−11	46	250	−39
	3	3	−6	−39	−8

G.2 Standard Nearest Neighbor Upsampler

A standard nearest-neighbor (NN) upsampler shall produce the output as a Kronecker Product of the lower-resolution array by a 2 x 2 unit array. A Kronecker Product is an operation on two arrays, of arbitrary dimensions (p, q) and (m, n), which results in an array of dimensions (pm, qn).

To produce the Kronecker product of an array, X, with an $m\,x\,n$ unit array Y (an array, each of whose elements is 1), each element of X is replaced by an $m \times n$ block of copies of the element.

The value of one element in the upsampled output array is the value of the reference element in the lower-resolution array, replicated twice along each axis.

EXAMPLE
Table G.3 shows an array X:

Table G.3 — Array X

	Column 0	Column 1	Column 2
Row 0	23	100	−68
Row 1	−2	255	0

Table G.4 shows an array Y:

Table G.4 — Unit Array Y

	Column 0	Column 1
Row 0	1	1
Row 1	1	1

The Kronecker Product of X and Y results in the array shown in Table G.5:

Table G.5 — Kronecker Product of the Example Arrays X and Y

	Column 0	Column 1	Column 2	Column 3	Column 4	Column 5
Row 0	23	23	100	100	−68	−68
Row 1	23	23	100	100	−68	−68
Row 2	−2	−2	255	255	0	0
Row 3	−2	−2	255	255	0	0

G.3 Standard Nonlinear 9-Transformer Upsamplers

G.3.1 Names and Roles

This clause specifies the names and permitted roles (input and output echelon indices) of standard nonlinear 9-transformer 16-8-3 upsamplers for resultant-planes with specific core-echelon indices.

NOTE 16-8-3 is a shortened version of 16-8-16-8-16-8, numbers connected with T5 to T0 in G.3.2.

Standard nonlinear 9-transformer 16-8-3 upsampler is abbreviated as nonlinear upsampler in the current clause.

The names of the nonlinear upsamplers are upsampler 5, upsampler 4, upsampler 3, upsampler 2, upsampler 1 and upsampler 0. The generic specification of nonlinear upsampler in G.3.2 applies to each one.

Let $m5$ be the greater of core-echelon index and the value -5.

Let q be an input echelon index and $q + 1$ be an output echelon index, where q is less than 0 and not less than core-echelon index.

If core-echelon index is equal to 0, then the use of an upsampler does not apply.

If core-echelon index is less than 0, then upsampler 5 should be used with output echelon index $q + 1$ and input echelon index q, where q is any value less than $m5$ and not less than core-echelon index. These uses are the only roles of upsampler 5.

If core-echelon index is less than -1, then upsampler 4 should be used with output echelon index $q + 1$ and input echelon index q, where q is equal to $m5$. This use is the only role of upsampler 4.

If core-echelon index is less than -2, then upsampler 3 should be used with output echelon index $q + 1$ and input echelon index q, where q is equal to the sum of $m5$ and the value 1. This use is the only role of upsampler 3.

If core-echelon index is less than -3, then upsampler 2 should be used with output echelon index $q + 1$ and input echelon index q, where q is equal to the sum of $m5$ and the value 2. This use is the only role of upsampler 2.

If core-echelon index is less than -4, then upsampler 1 should be used with output echelon index $q + 1$ and input echelon index q, where q is equal to the sum of $m5$ and the value 3. This use is the only role of upsampler 1.

If core-echelon index is less than -5, then upsampler 0 should be used with output echelon index $q + 1$ and input echelon index q, where q is equal to the sum of $m5$ and the value 4. This use is the only role of upsampler 0.

G.3.2 Picture Transformers

A standard nonlinear 9-transformer 16-8-3 upsampler, abbreviated in this clause (G.3.2) to nonlinear upsampler, comprises 9 picture transformers, referred to as R0, S0, S1, T0, T1, T2, T3, T4 and T5, except that S0 and S1 are not present in the case of upsampler 0.

Figure G.1 illustrates the use of transformers of a particular nonlinear upsampler.

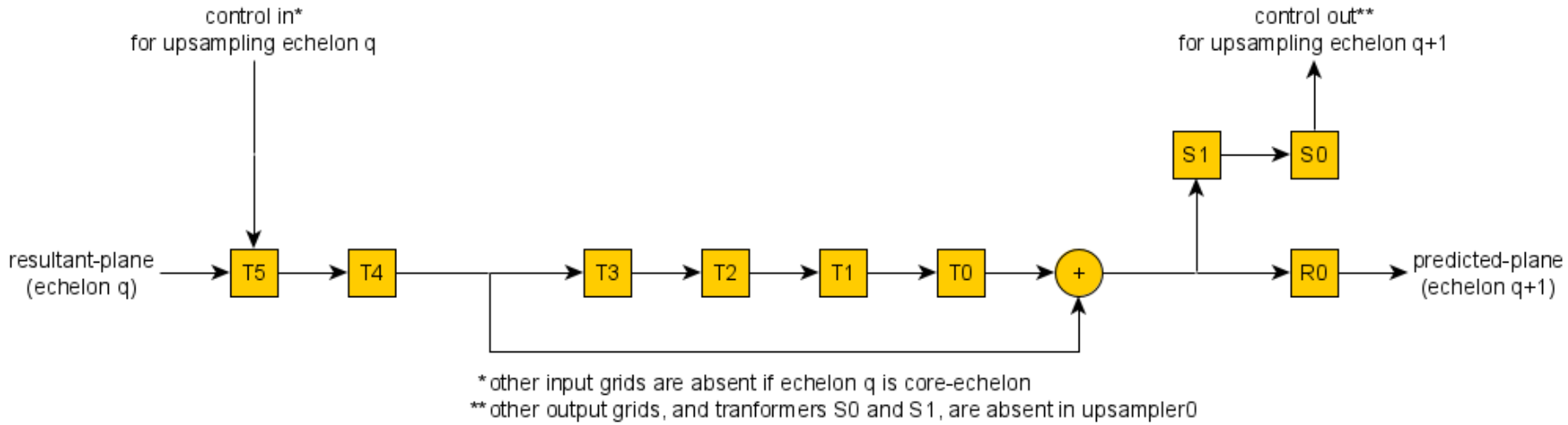


Figure G.1 — Flow of data through the picture transformers in a nonlinear upsampler

Picture transformers operate on grids, which are either in picture format or feature format:

- Feature format is expressed in 32-bit floating point type, as specified by IEEE Std 754-2008 “Standard for Floating-Point Arithmetic”.
- Picture format is expressed in decimal fixed-point format, with three decimal places.

All output grids described in this clause (G.3.2) shall be in feature format, except that the output from R0 is in picture format.

All output grids described in this clause (G.3.2) shall be of grid width and grid height corresponding to the input echelon index, except that the outputs from R0, S0 and S1 are of grid width and grid height corresponding to the output echelon index.

The input or inputs to T5 shall be a grid or grids with echelon index equal to input echelon index.

If the input echelon index is the core-echelon index, then T5 shall have 1 grid as input, obtained from a resultant-plane, which is in picture format, at the input echelon index, by converting all elements to floating point and dividing all resulting elements by 1000.

If the input echelon index is not the core-echelon index, then T5 shall have 1 converted grid and 7 previously calculated grids as inputs. The 1 converted grid is obtained from the resultant-plane, which is in picture format, at the input echelon index, by converting all elements to feature format and dividing all resulting elements by 1000. The 7 previously calculated grids are the outputs of S0 from a nonlinear upsampler in the role where the input echelon index and output echelon index were lower by 1.

Outputs shall be specified as follows:

- The outputs of T5, T3 or T1 are 16 grids.
- The outputs of T4, T2 or T0 are 8 grids.
- The outputs of S1 are 16 grids, all of grid width and grid height corresponding to the output echelon index.
- The outputs of S0 are 7 grids, all of grid width and grid height corresponding to the output echelon index.
- The output from R0 is 1 grid in picture format, of grid width and grid height corresponding to the output echelon index, representing a predicted-plane.

Inputs shall be specified as follows:

- The inputs of T4 are the outputs of T5.
- The inputs of T3 are the outputs of T4.
- The inputs of T2 are the outputs of T3.
- The inputs of T1 are the outputs of T2.
- The inputs of T0 are the outputs of T1.
- The inputs of S1 are the respective outputs of T0 added to the respective outputs of T4.
- The inputs of S0 are the outputs of S1.
- The inputs of R0 are the respective outputs of T0 added to respective outputs of T4.

G.3.3 Standard Picture Transformers

A sharp upsampler is a standard linear upsampler using coefficients from the sharp upsampler table.

Any picture transformer of a standard nonlinear 9-transformer 16-8-3 upsampler has a set of convolution operators, each convolution operator characterized as in G.3.4.

Except in the case of R0, the outputs of a picture transformer are the outputs of the convolution operators in its set of convolution operators.

The oob rule for any convolution operator in a set associated with T0, T1, T2, T3, T4 or T5 is the mirror function, as follows, in Formula (G.2):

$$mirror(x, W) = \begin{cases} x, & 0 \leq x < W \\ -x, & 0 \\ 2 * W - x - 2, & x \geq W \end{cases} \quad \text{(G.2)}$$

The oob rule for any convolution operator in a set associated with R0, S0 or S1 is the zero function, as follows, in Formula (G.3):

$$zero(x, W) = \begin{cases} x, & 0 \leq x < W \\ 0, & otherwise \end{cases} \quad \text{(G.3)}$$

The size of a set of convolution operators is the number of outputs of the corresponding picture transformer of G.3.2.

The number of input channels of a convolution operator is the number of inputs of the corresponding picture transformer of G.3.2.

The kernel size is 1 for a convolution operator corresponding to S0, T0, T2 or T4.

The kernel size is 3 for a convolution operator corresponding to T1, T3 or T5.

The kernel size is 16 for a convolution operator corresponding to S1. Note that only 16 of 256 coefficients are distinct in each channel of S1.

The kernel size is 16 for a convolution operator corresponding to R0. Note that only 16 of 256 coefficients are distinct in each channel of R0.

The step shall be 2 for a convolution operator corresponding to S1 or R0.

The step shall be 1 for a convolution operator corresponding to S0, T0, T1, T2, T3, T4 and T5.

R0 shall convert all elements of the feature format output of its convolution operator to picture format by multiplying them by 1000 and rounding to fixed-point. Finally, each R0 output element shall be the sum of a corresponding converted element and corresponding output element from a sharp upsampler applied to the resultant-plane of core-echelon index.

NOTE Thus the output of R0 is in picture format and is at output echelon index in accordance with G.3.2.

There is one bias factor for each convolution operator and one multiplication factor for each picture transformer.

G.3.4 Convolution Operator

G.3.4.1 Convolution Operator Parameters

NOTE 1 A convolution operator belongs to a set of convolution operators for a specific picture transformer of a specific nonlinear upsampler (see G.3.3).

A convolution operator takes input parameter **I** and computes output parameter **O**. The relationship between **I** and **O** is given in G.3.4.2.

NOTE 2 **I** and **O** correspond respectively to one input and one output from the inputs and outputs of G.3.2.

A convolution operator is determined using G.3.4.2 and the following constant parameters described in G.3.3:

- Bias factor b
- 3D coefficients array k
- Multiplication factor a
- A rule for out-of-bound reading (oob rule)
- A step that is either 1 or 2

G.3.4.2 Convolution Operator Input/Output Relationship

Array indexing starts at 0 in this subclause (G.3.4.2).

The out-of-bound reading rule is a method of choosing values implicitly outside the domain of an array, instead of having to read them.

All convolution operators have an out-of-bound reading rule (oob rule), which applies when the convolution operator needs a value that would fall outside of the area of its input grid or grids.

Floating point numbers and computations shall conform to the standard for floating-point arithmetic specified in IEEE Std 754-2008.

The relationship between **I** and **O** of a convolution operator shall be defined, as follows, by Formula (G.4):

$$O[y,x] = max\,(R[y,x], a * R[y,x]) \tag{G.4}$$

where $R[y,x]$ is defined, as follows, by Formula (G.5):

$$R[y,x] = b + \sum_{c=0}^{C-1} \sum_{j=0}^{siz-1} \sum_{i=0}^{siz-1} S[c, y-o+j, x-o+i] * k[c,j,i] \tag{G.5}$$

where

C is the number of input channels;

siz is the kernel size;

$O[y,x]$ is the output grid, in feature format. First index y is the vertical dimension, second index x is the horizontal dimension;

$k[c,j,i]$ is the kernel, where i is the horizontal index and j is the vertical index;

o is $siz\ //\ 2$;

W_c is the horizontal size of input channel c;

H_c is the vertical size of input channel c;

where the out-of-bound reading rule is applied on $I[c,n,m]$ arrays as follows:

If step is 1, then Formula (G.6) applies:

$$S[c,y,x] = I[c,\mathrm{oob}(y),\mathrm{oob}(x)] \tag{G.6}$$

If step is 2, then, then Formula (G.7) applies:

$$S[c,y,x] = \begin{cases} I\left[c, oob\left((y-1)\ //\ 2, H_c\right), oob\left((x-1)\ //\ 2, W_c\right)\right], for\ \ both\ x\ and\ y\ odd \\ 0,\ otherwise \end{cases} \tag{G.7}$$

where

$I[c,n,m]$ is the 3D input array (in feature format). First index c is the channel index, second index m is the vertical dimension, and the third index m is the horizontal dimension.

G.4 Organization of Coefficients Sets for Standard Nonlinear Upsamplers

Values of all factors and coefficients described in Clause G.3, and required for use by upsamplers 0 to 5, are provided in SMPTE-ST-2117-1a.

An informative JSON schema for the nonlinear coefficient data is provided in element SMPTE ST-2117-1b. It is provided to assist software tools parsing the JSON data.

In the JSON file, object names "upsampler_0", "upsampler_1", "upsampler_2", "upsampler_3", "upsampler_4" and "upsampler_5" refer to the nonlinear upsamplers specified to be used in the roles given in G.3.1.

Within any such object:

- The names "Set_0", "Set_1", "Set_2", "Set_3", "Set_4" and "Set_5" correspond to picture transformers T0 to T5 referred to in G.3.2;
- The name "reference" corresponds to R0; and,
- The name "featureDeconvolution" corresponds to S0 and S1.

The method of encoding the relevant floating point data into strings in JSON arrays is detailed in the "ReadMe" object of the JSON file.

See Annex H for list of additional elements.

Annex H
(Normative)

Additional Elements

Table H.1 lists non-prose elements of this document.

Table H.1 — Non-prose element/Description

Non-prose element	Description
SMPTE-ST-2117-1a	Provides values of all factors and coefficients described in Clause G.3, and required for use by upsamplers 0 to 5
SMPTE-ST-2117-1b	An informative JSON schema for the nonlinear coefficient data

The XML Schemas to SMPTE ST 2117-1 are provided as companion elements to this document in two GitHub repositories at the following URL:

https://github.com/SMPTE/st2117-1a

https://github.com/SMPTE/st2117-1b

Bibliography (Informative)

Belyaev E., Veselov A., Turlikov A., Kai L. (2011) Complexity Analysis of Adaptive Binary Arithmetic Coding Software Implementations. In: Balandin S., Koucheryavy Y., Hu H. (eds) Smart Spaces and Next Generation Wired/Wireless Networking. ruSMART 2011, NEW2AN 2011. Lecture Notes in Computer Science, vol 6869, 598-607. Springer, Berlin, Heidelberg

Gillespie R. and Davis W. (1981) Tree Data Structures for Graphics and Image Processing. In: Tanner P. and Swail E. (eds) Proceedings of the 7th Canadian Man-Computer Communications Conference. CMCCC 1981, 155-162. National Research Council of Canada

Gray R. and Neuhoff D. (1998) Quantization. IEEE Transactions on Information Theory, vol 44, 2325-2383

Rabbani M. and Jones P. (1991) Digital Image Compression Techniques. SPIE Tutorial Text, vol TT07. The Society of Photo-Optical Instrumentation Engineers, Bellingham, Washington

Shusterman E. and Feder M. (1994) Image Compression via Improved Quadtree Decomposition Algorithms. IEEE Transactions on Image Processing, vol 3, 207-215

Internet Engineering Task Force (IETF), RFC 8259, “The JavaScript Object Notation (JSON) Data Interchange Format”, 2017-12, Available at: https://www.tools.ietf.org/html/rfc8259