WebVTT: The Web Video Text Tracks Format

W3C Candidate Recommendation 4 April 2019

This version:
https://www.w3.org/TR/2019/CR-webvtt1-20190404/

Latest published version:
https://www.w3.org/TR/webvtt1/

Editor's Draft:
https://w3c.github.io/webvtt/

Previous Versions:
https://www.w3.org/TR/2018/CR-webvtt1-20180510/

Test Suite:
https://github.com/web-platform-tests/wpt/tree/master/webvtt

Editor:
Silvia Pfeiffer (CSIRO)

Former Editors:
Simon Pieters (Opera Software AS)
Silvia Pfeiffer (NICTA)
Philip Jägenstedt (Opera Software ASA)
Ian Hickson (Google)

Participate:
GitHub w3c/webvtt (new issue, open issues, legacy open bugs)

Commits:
GitHub w3c/webvtt/commits
@webvtt

Abstract

This specification defines WebVTT, the Web Video Text Tracks format. Its main use is for marking up external text track resources in connection with the HTML <track> element. WebVTT files provide captions or subtitles for video content, and also text video descriptions [MAUR], chapters for content

navigation, and more generally any form of metadata that is time-aligned with audio or video content.

This specification is based on the Draft Community Group Report of the Web Media Text Tracks Community Group.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at https://www.w3.org/TR/.

This document was produced by the W3C Timed Text Working Group as a Candidate Recommendation. This document is intended to become a W3C Recommendation. If you wish to make comments regarding this document, please send them to public-tt@w3.org (subscribe, archives) with `[webvtt]` at the start of your email's subject. All comments are welcome. W3C publishes a Candidate Recommendation to indicate that the document is believed to be stable and to encourage implementation by the developer community. This document will remain a Candidate Recommendation at least until 2 May 2019 in order to ensure the opportunity for wide review.

Please see the Working Group's Implementation Report.

For this specification to exit the CR stage, at least 2 independent implementations of every feature defined in this specification need to be documented in the implementation report. The implementation report is based on implementer-provided test results for the test suite. The Working Group does not require that implementations are publicly available but encourages them to be so.

The following features are at-risk, and may be dropped during the CR period:

- collision avoidance with snap-to-lines false
- ::cue-region pseudo-element
- :past and :future pseudo-classes

A cumulative summary of all changes applied to this version since the WebVTT First Public Working Draft was published is available at Changes from FPWD WebVTT.

For convenience, a complete diff between this version and the WebVTT previous Working Draft was published is found at Diff from previous Working Draft WebVTT.

Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It

is inappropriate to cite this document as other than work in progress.

This document was produced by a group operating under the W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document is governed by the 1 March 2019 W3C Process Document.

Table of Contents

§ 1. Introduction

This section is non-normative.

The ***WebVTT*** (Web Video Text Tracks) format is intended for marking up external text track resources in connection with the HTML <track> element.

WebVTT files provide captions or subtitles for video content, and also text video descriptions [MAUR], chapters for content navigation, and more generally any form of metadata that is time-aligned with audio or video content.

The majority of the current version of this specification is dedicated to describing how to use WebVTT files for captioning or subtitling. There is minimal information about chapters and time-

aligned metadata and nothing about video descriptions at this stage.

In this section we provide some example WebVTT files as an introduction.

§ 1.1. A simple caption file

This section is non-normative.

The main use for WebVTT files is captioning or subtitling video content. Here is a sample file that captions an interview:

```
WEBVTT

00:11.000 --> 00:13.000
<v Roger Bingham>We are in New York City

00:13.000 --> 00:16.000
<v Roger Bingham>We’re actually at the Lucern Hotel, just down the street

00:16.000 --> 00:18.000
<v Roger Bingham>from the American Museum of Natural History

00:18.000 --> 00:20.000
<v Roger Bingham>And with me is Neil deGrasse Tyson

00:20.000 --> 00:22.000
<v Roger Bingham>Astrophysicist, Director of the Hayden Planetarium

00:22.000 --> 00:24.000
<v Roger Bingham>at the AMNH.

00:24.000 --> 00:26.000
<v Roger Bingham>Thank you for walking down here.

00:27.000 --> 00:30.000
<v Roger Bingham>And I want to do a follow-up on the last conversation we did.

00:30.000 --> 00:31.500 align:right size:50%
<v Roger Bingham>When we e-mailed—

00:30.500 --> 00:32.500 align:left size:50%
<v Neil deGrasse Tyson>Didn’t we talk about enough in that conversation?

00:32.000 --> 00:35.500 align:right size:50%
<v Roger Bingham>No! No no no no; 'cos 'cos obviously 'cos

00:32.500 --> 00:33.500 align:left size:50%
<v Neil deGrasse Tyson><i>Laughs</i>

00:35.500 --> 00:38.000
<v Roger Bingham>You know I’m so excited my glasses are falling off here.
```

You can see that a WebVTT file in general consists of a sequence of text segments associated with a time-interval, called a cue (definition). Beyond captioning and subtitling, WebVTT can be used for

time-aligned metadata, typically in use for delivering name-value pairs in cues. WebVTT can also be used for delivering chapters, which helps with contextual navigation around an audio/video file. Finally, WebVTT can be used for the delivery of text video descriptions, which is text that describes the visual content of time-intervals and can be synthesized to speech to help vision-impaired users understand context.

> This version of WebVTT focuses on solving the captioning and subtitling use cases. More specification work is possible for the other use cases. A decision on what type of use case a WebVTT file is being used for is made by the software that is using the file. For example, if in use with a HTML file through a <track> element, the kind attribute defines how the WebVTT file is to be interpreted.

The following subsections provide an overview of some of the key features of the WebVTT file format, particularly when in use for captioning and subtitling.

§ 1.2. Caption cues with multiple lines

This section is non-normative.

Line breaks in cues are honored. User agents will also insert extra line breaks if necessary to fit the cue in the cue's width. In general, therefore, authors are encouraged to write cues all on one line except when a line break is definitely necessary.

EXAMPLE 1

These captions on a public service announcement video demonstrate line breaking:

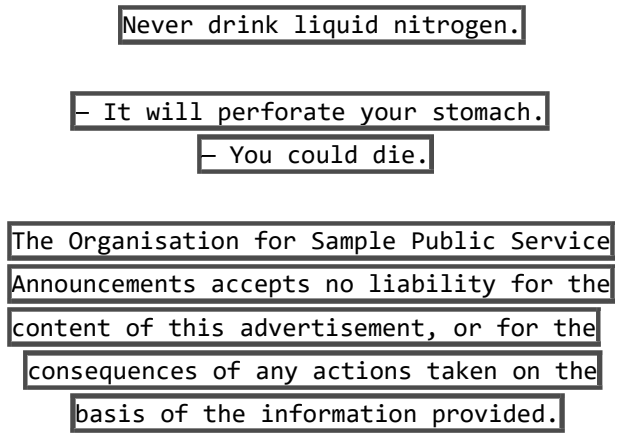
```
WEBVTT

00:01.000 --> 00:04.000
Never drink liquid nitrogen.

00:05.000 --> 00:09.000
– It will perforate your stomach.
– You could die.

00:10.000 --> 00:14.000
The Organisation for Sample Public Service Announcements accepts no liability fo
```

The first cue is simple, it will probably just display on one line. The second will take two lines, one for each speaker. The third will wrap to fit the width of the video, possibly taking multiple lines. For example, the three cues could look like this:



If the width of the cues is smaller, the first two cues could wrap as well, as in the following example. Note how the second cue's explicit line break is still honored, however:

```
Never drink
liquid nitrogen.

– It will perforate
your stomach.
– You could die.

The Organisation for
Sample Public Service
Announcements accepts
no liability for the
content of this
advertisement, or for
the consequences of
any actions taken on
the basis of the
information provided.
```

Also notice how the wrapping is done so as to keep the line lengths balanced.

§ 1.3. Styling captions

This section is non-normative.

CSS style sheets that apply to an HTML page that contains a `<video>` element can target WebVTT cues and regions in the video using the '::cue', '::cue()', '::cue-region' and '::cue-region()' pseudo-elements.

EXAMPLE 2

In this example, an HTML page has a CSS style sheet in a `<style>` element that styles all cues in the video with a gradient background and a text color, as well as changing the text color for all WebVTT Bold Objects in cues in the video.

```
<!doctype html>
<html>
 <head>
  <title>Styling WebVTT cues</title>
  <style>
   video::cue {
     background-image: linear-gradient(to bottom, dimgray, lightgray);
     color: papayawhip;
   }
   video::cue(b) {
     color: peachpuff;
   }
  </style>
 </head>
 <body>
  <video controls autoplay src="video.webm">
   <track default src="track.vtt">
  </video>
 </body>
</html>
```

CSS style sheets can also be embedded in WebVTT files themselves.

Style blocks are placed after any headers but before the first cue, and start with the line "STYLE". Comment blocks can be interleaved with style blocks.

Blank lines cannot appear in the style sheet. They can be removed or be filled with a space or a CSS comment (e.g. `/**/`).

The string "`-->`" cannot be used in the style sheet. If the style sheet is wrapped in "`<!--`" and "`-->`", then those strings can just be removed. If "`-->`" appears inside a CSS string, then it can use CSS escaping e.g. "`--\>`".

EXAMPLE 3

This example shows how cues can be styled with style blocks in WebVTT.

```
WEBVTT

STYLE
::cue {
  background-image: linear-gradient(to bottom, dimgray, lightgray);
  color: papayawhip;
}
/* Style blocks cannot use blank lines nor "dash dash greater than" */

NOTE comment blocks can be used between style blocks.

STYLE
::cue(b) {
  color: peachpuff;
}

hello
00:00:00.000 --> 00:00:10.000
Hello <b>world</b>.

NOTE style blocks cannot appear after the first cue.
```

§ 1.4. Other caption and subtitling features

This section is non-normative.

WebVTT also supports some less-often used features.

EXAMPLE 4

In this example, the cues have an identifier:

```
WEBVTT

test
00:00.000 --> 00:02.000
This is a test.

123
00:00.000 --> 00:02.000
That’s an, an, that’s an L!

crédit de transcription
00:04.000 --> 00:05.000
Transcrit par Célestes™
```

This allows a style sheet to specifically target the cues.

```
/* style for cue: test */
::cue(#test) { color: lime; }
```

Due to the syntax rules of CSS, some characters need to be escaped with CSS character escape sequences. For example, an ID that starts with a number 0-9 needs to be escaped. The ID `123` can be represented as "\31 23" (31 refers to the Unicode code point for "1"). See Using character escapes in markup and CSS for more information on CSS escapes.

```
/* style for cue: 123 */
::cue(#\31 23) { color: lime; }
/* style for cue: crédit de transcription */
::cue(#crédit\ de\ transcription) { color: red; }
```

EXAMPLE 5

This example shows how classes can be used on elements, which can be helpful for localization or maintainability of styling, and also how to indicate a language change in the cue text.

```
WEBVTT

04:02.500 --> 04:05.000
J’ai commencé le basket à l'âge de 13, 14 ans

04:05.001 --> 04:07.800
Sur les <i.foreignphrase><lang en>playground</lang></i>, ici à Montpellier
```

EXAMPLE 6

In this example, each cue says who is talking using voice spans. In the first cue, the span specifying the speaker is also annotated with two classes, "first" and "loud". In the third cue, there is also some italics text (not associated with a specific speaker). The last cue is annotated with just the class "loud".

```
WEBVTT

00:00.000 --> 00:02.000
<v.first.loud Esme>It’s a blue apple tree!

00:02.000 --> 00:04.000
<v Mary>No way!

00:04.000 --> 00:06.000
<v Esme>Hee!</v> <i>laughter</i>

00:06.000 --> 00:08.000
<v.loud Mary>That’s awesome!
```

Notice that as a special exception, the voice spans don’t have to be closed if they cover the entire cue text.

Style sheets can style these spans:

```
::cue(v[voice="Esme"]) { color: cyan }
::cue(v[voice="Mary"]) { color: lime }
::cue(i) { font-style: italic }
::cue(.loud) { font-size: 2em }
```

EXAMPLE 7

This example shows how to position cues at explicit positions in the video viewport.

```
WEBVTT

00:00:00.000 --> 00:00:04.000 position:10%,line-left align:left size:35%
Where did he go?

00:00:03.000 --> 00:00:06.500 position:90% align:right size:35%
I think he went down this lane.

00:00:04.000 --> 00:00:06.500 position:45%,line-right align:center size:35%
What are you waiting for?
```

Since the cues in these examples are horizontal, the "position" setting refers to a percentage of the width of the video viewpoint. If the text were vertical, the "position" setting would refer to the height of the video viewport.

The "line-left" or "line-right" only refers to the physical side of the box to which the "position" setting applies, in a way which is agnostic regarding the horizontal or vertical direction of the cue. It does not affect or relate to the direction or position of the text itself within the box.

The cues cover only 35% of the video viewport's width - that's the cue box's "size" for all three cues.

The first cue has its cue box positioned at the 10% mark. The "line-left" and "line-right" within the "position" setting indicates which side of the cue box the position refers to. Since in this case the text is horizontal, "line-left" refers to the left side of the box, and the cue box is thus positioned between the 10% and the 45% mark of the video viewport's width, probably underneath a speaker on the left of the video image. If the cue was vertical, "line-left" positioning would be from the top of the video viewport's height and the cue box would cover 35% of the video viewport's height.

The text within the first cue's cue box is aligned using the "align" cue setting. For left-to-right rendered text, "start" alignment is the left of that box, for right-to-left rendered text the right of the box. So, independent of the directionality of the text, it will stay underneath that speaker. Note that "center" position alignment of the cue box is the default for start aligned text, in order to avoid having the box move when the base direction of the text changes (from left-to-right to right-to-left or vice versa) as a result of translation.

The second cue has its cue box right aligned at the 90% mark of the video viewport width ("right" aligned text right aligns the box). The same effect can be achieved with "position:55%,line-left",

which explicitly positions the cue box. The third cue has center aligned text within the same positioned cue box as the first cue.

EXAMPLE 8

This example shows two regions containing rollup captions for two different speakers. Fred's cues scroll up in a region in the left half of the video, Bill's cues scroll up in a region on the right half of the video. Fred's first cue disappears at 12.5sec even though it is defined until 20sec because its region is limited to 3 lines and at 12.5sec a fourth cue appears:

```
WEBVTT

REGION
id:fred
width:40%
lines:3
regionanchor:0%,100%
viewportanchor:10%,90%
scroll:up

REGION
id:bill
width:40%
lines:3
regionanchor:100%,100%
viewportanchor:90%,90%
scroll:up

00:00:00.000 --> 00:00:20.000 region:fred align:left
<v Fred>Hi, my name is Fred

00:00:02.500 --> 00:00:22.500 region:bill align:right
<v Bill>Hi, I'm Bill

00:00:05.000 --> 00:00:25.000 region:fred align:left
<v Fred>Would you like to get a coffee?

00:00:07.500 --> 00:00:27.500 region:bill align:right
<v Bill>Sure! I've only had one today.

00:00:10.000 --> 00:00:30.000 region:fred align:left
<v Fred>This is my fourth!

00:00:12.500 --> 00:00:32.500 region:fred align:left
<v Fred>OK, let's go.
```

Note that regions are only defined for horizontal cues.

§ 1.5. Comments in WebVTT

This section is non-normative.

Comments can be included in WebVTT files.

Comments are just blocks that are preceded by a blank line, start with the word "`NOTE`" (followed by a space or newline), and end at the first blank line.

EXAMPLE 9

Here, a one-line comment is used to note a possible problem with a cue.

```
WEBVTT

00:01.000 --> 00:04.000
Never drink liquid nitrogen.

NOTE I’m not sure the timing is right on the following cue.

00:05.000 --> 00:09.000
– It will perforate your stomach.
– You could die.
```

EXAMPLE 10

In this example, the author has written many comments.

```
WEBVTT

NOTE
This file was written by Jill. I hope
you enjoy reading it. Some things to
bear in mind:
- I was lip-reading, so the cues may
not be 100% accurate
- I didn't pay too close attention to
when the cues should start or end.

00:01.000 --> 00:04.000
Never drink liquid nitrogen.

NOTE check next cue

00:05.000 --> 00:09.000
– It will perforate your stomach.
– You could die.

NOTE end of file
```

§ 1.6. Chapters example

This section is non-normative.

A WebVTT file can consist of chapters, which are navigation markers for the video.

Chapters are plain text, typically just a single line.

EXAMPLE 11

In this example, a talk is split into each slide being a chapter.

```
WEBVTT

NOTE
This is from a talk Silvia gave about WebVTT.

Slide 1
00:00:00.000 --> 00:00:10.700
Title Slide

Slide 2
00:00:10.700 --> 00:00:47.600
Introduction by Naomi Black

Slide 3
00:00:47.600 --> 00:01:50.100
Impact of Captions on the Web

Slide 4
00:01:50.100 --> 00:03:33.000
Requirements of a Video text format
```

§ 1.7. Metadata example

This section is non-normative.

A WebVTT file can consist of time-aligned metadata.

Metadata can be any string and is often provided as a JSON construct.

Note that you cannot provide blank lines inside a metadata block, because the blank line signifies the end of the WebVTT cue.

EXAMPLE 12

In this example, a talk is split into each slide being a chapter.

```
WEBVTT

NOTE
Thanks to http://output.jsbin.com/mugibo

1
00:00:00.100 --> 00:00:07.342
{
 "type": "WikipediaPage",
 "url": "https://en.wikipedia.org/wiki/Samurai_Pizza_Cats"
}

2
00:07.810 --> 00:09.221
{
 "type": "WikipediaPage",
 "url" :"http://samuraipizzacats.wikia.com/wiki/Samurai_Pizza_Cats_Wiki"
}

3
00:11.441 --> 00:14.441
{
 "type": "LongLat",
 "lat" : "36.198269",
 "long": "137.2315355"
}
```

§ 2. Conformance

All diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC2119. The key word "OPTIONALLY" in the normative parts of this document is to be interpreted with the same normative meaning as "MAY" and "OPTIONAL". For readability, these words do not appear in all uppercase let-

ters in this specification. [RFC2119]

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps may be implemented in any manner, so long as the end result is equivalent. (In particular, the algorithms defined in this specification are intended to be easy to follow, and not intended to be performant.)

§ 2.1. Conformance classes

This specification describes the conformance criteria for user agents (relevant to implementors) and WebVTT files (relevant to authors and authoring tool implementors).

§4 Syntax defines what consists of a valid WebVTT file. Authors need to follow the requirements therein, and are encouraged to use a conformance checker. §6 Parsing defines how user agents are to interpret a file labelled as text/vtt, for both valid and invalid WebVTT files. The parsing rules are more tolerant to author errors than the syntax allows, in order to provide for extensibility and to still render cues that have some syntax errors.

EXAMPLE 13

For example, the parser will create two cues even if the blank line between them is skipped. This is clearly a mistake, so a conformance checker will flag it as an error, but it is still useful to render the cues to the user.

User agents fall into several (possibly overlapping) categories with different conformance requirements.

User agents that support scripting

All processing requirements in this specification apply. The user agent must also be conforming implementations of the IDL fragments in this specification, as described in the Web IDL specification. [WEBIDL-1]

User agents with no scripting support

All processing requirements in this specification apply, except those in §6.5 WebVTT cue text DOM construction rules and §9 API.

User agents that do not support CSS

All processing requirements in this specification apply, except parts of §6 Parsing that relate to

stylesheets and CSS, and all of §7 Rendering and §8 CSS extensions. The user agent must instead only render the text inside WebVTT caption or subtitle cue text in an appropriate manner and specifically support the color classes defined in §5 Default classes for WebVTT Caption or Subtitle Cue Components. Any other styling instructions are optional.

User agents that do not support a full HTML CSS engine

All processing requirements in this specification apply, including the color classes defined in §5 Default classes for WebVTT Caption or Subtitle Cue Components. However, the user agent will need to apply the CSS related features in §6 Parsing, §7 Rendering and §8 CSS extensions in such a way that the rendered results are equivalent to what a full CSS supporting renderer produces.

User agents that support a full HTML CSS engine

All processing requirements in this specification apply. However, only a limited set of CSS styles is allowed because user agents that do not support a full HTML CSS engine will need to implement CSS functionality equivalents. User agents that support a full CSS engine must therefore limit the CSS styles they apply for WebVTT so as to enable identical rendering without bleeding in extra CSS styles that are beyond the WebVTT specification.

Conformance checkers

Conformance checkers must verify that a WebVTT file conforms to the applicable conformance criteria described in this specification. The term "validator" is equivalent to conformance checker for the purpose of this specification.

Authoring tools

Authoring tools must generate conforming WebVTT files. Tools that convert other formats to WebVTT are also considered to be authoring tools.

When an authoring tool is used to edit a non-conforming WebVTT file, it may preserve the conformance errors in sections of the file that were not edited during the editing session (i.e. an editing tool is allowed to round-trip erroneous content). However, an authoring tool must not claim that the output is conformant if errors have been so preserved.

§ 2.2. Unicode normalization

Implementations of this specification must not normalize Unicode text during processing.

EXAMPLE 14

For example, a cue with an identifier consisting of the characters U+0041 LATIN CAPITAL LETTER A followed by U+030A COMBINING RING ABOVE (a decomposed character sequence), or the character U+212B ANGSTROM SIGN (a compatibility character), will not match a selector targeting a cue with an ID consisting of the character U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE (a precomposed character).

§ 3. Data model

The box model of WebVTT consists of three key elements: the video viewport, cues, and regions. The video viewport is the rendering area into which cues and regions are rendered. Cues are boxes consisting of a set of cue lines. Regions are subareas of the video viewport that are used to group cues together. Cues are positioned either inside the video viewport directly or inside a region, which is positioned inside the video viewport.

The position of a cue inside the video viewport is defined by a set of cue settings. The position of a region inside the video viewport is defined by a set of region settings. Cues that are inside regions can only use a limited set of their cue settings. Specifically, if the cue has a "vertical", "line" or "size" setting, the cue drops out of the region. Otherwise, the cue's width is calculated to be relative to the region width rather than the viewport.

§ 3.1. Overview

This section is non-normative.

The WebVTT file is a container file for chunks of data that are time-aligned with a video or audio resource. It can therefore be regarded as a serialisation format for time-aligned data.

A WebVTT file starts with a header and then contains a series of data blocks. If a data block has a start and end time, it is called a WebVTT cue. A comment is another kind of data block.

Different kinds of data can be carried in WebVTT files. The HTML specification identifies captions, subtitles, chapters, audio descriptions and metadata as data kinds and specifies which one is being used in the text track kind attribute of the text track element [HTML51].

A WebVTT file must only contain data of one kind, never a mix of different kinds of data. The data kind of a WebVTT file is externally specified, such as in a HTML file's text track element. The envi-

ronment is responsible for interpreting the data correctly.

WebVTT caption or subtitle cues are rendered as overlays on top of a video viewport or into a region, which is a subarea of the video viewport.

§ 3.2. WebVTT cues

A ***WebVTT cue*** is a text track cue [HTML51] that additionally consist of the following:

A cue text
: The raw text of the cue, and rules for its interpretation.

§ 3.3. WebVTT caption or subtitle cues

A ***WebVTT caption or subtitle cue*** is a WebVTT cue that has the following additional properties allowing the cue text to be rendered and converted to a DOM fragment:

A cue box
: The cue box of a WebVTT cue is a box within which the text of all lines of the cue is to be rendered. It is either rendered into the video's viewport or a region inside the viewport if the cue is part of a region.

> The position of the cue box within the video viewport's or region's dimensions depends on the value of the WebVTT cue position and the WebVTT cue line.

> Lines are wrapped within the cue box's size if lines' lengths make this necessary.

A writing direction
: A writing direction, either

- ***horizontal*** (a line extends horizontally and is offset vertically from the video viewport's top edge, with consecutive lines displayed below each other),
- ***vertical growing left*** (a line extends vertically and is offset horizontally from the video viewport's right edge, with consecutive lines displayed to the left of each other), or
- ***vertical growing right*** (a line extends vertically and is offset horizontally from the video viewport's left edge, with consecutive lines displayed to the right of each other).

> The writing direction affects the interpretation of the line, position, and size cue settings to be interpreted with respect to either the width or height of the video.

By default, the writing direction is set to to horizontal.

> The vertical growing left writing direction could be used for vertical Chinese, Japanese, and Korean, and the vertical growing right writing direction could be used for vertical Mongolian.

A snap-to-lines flag

A boolean indicating whether the line is an integer number of lines (using the line dimensions of the first line of the cue), or whether it is a percentage of the dimension of the video. The flag is set to true when lines are counted, and false otherwise.

Cues where the flag is false will be offset as requested modulo overlap avoidance if multiple cues are in the same place.

By default, the snap-to-lines flag is set to true.

A line

The line defines positioning of the cue box.

The line offsets the cue box from the top, the right or left of the video viewport as defined by the writing direction, the snap-to-lines flag, or the lines occupied by any other showing tracks.

The line is set either as a number of lines, a percentage of the video viewport height or width, or as the special value ***auto***, which means the offset is to depend on the other showing tracks.

By default, the line is set to auto.

If the writing direction is horizontal, then the line percentages are relative to the height of the video, otherwise to the width of the video.

A WebVTT cue has a ***computed line*** whose value is that returned by the following algorithm, which is defined in terms of the other aspects of the cue:

1. If the line is numeric, the WebVTT cue snap-to-lines flag of the WebVTT cue is false, and the line is negative or greater than 100, then return 100 and abort these steps.

   > Although the WebVTT parser will not set the line to a number outside the range 0..100 and also set the WebVTT cue snap-to-lines flag to false, this can happen when using the DOM API's `snapToLines` and `line` attributes.

2. If the line is numeric, return the value of the WebVTT cue line and abort these steps. (Either the WebVTT cue snap-to-lines flag is true, so any value, not just those in the range 0..100, is valid, or the value is in the range 0..100 and is thus valid regardless of the value of that flag.)
3. If the WebVTT cue snap-to-lines flag of the WebVTT cue is false, return the value 100 and abort these steps. (The line is the special value auto.)
4. Let *cue* be the WebVTT cue.
5. If *cue* is not in a list of cues of a text track, or if that text track is not in the list of text tracks of a media element, return −1 and abort these steps.
6. Let *track* be the text track whose list of cues the *cue* is in.
7. Let *n* be the number of text tracks whose text track mode is showing and that are in the media element's list of text tracks before *track*.
8. Increment *n* by one.
9. Negate *n*.
10. Return *n*.

> EXAMPLE 15
>
> For example, if two text tracks are showing at the same time in one media element, and each text track currently has an active WebVTT cue whose line are both auto, then the first text track's cue's computed line will be −1 and the second will be −2.

A line alignment

An alignment for the cue box's line, one of:

Start alignment

The cue box's top side (for horizontal cues), left side (for vertical growing right), or right side (for vertical growing left) is aligned at the line.

Center alignment

The cue box is centered at the line.

End alignment

The cue box's bottom side (for horizontal cues), right side (for vertical growing right), or left side (for vertical growing left) is aligned at the line.

By default, the line alignment is set to start.

> The line alignment is separate from the text alignment — right-to-left vs. left-to-right cue text does not affect the line alignment.

A position

The position defines the indent of the cue box in the direction defined by the writing direction.

The position is either a number giving the position of the cue box as a percentage value or the special value ***auto***, which means the position is to depend on the text alignment of the cue.

If the cue is not within a region, the percentage value is to be interpreted as a percentage of the video dimensions, otherwise as a percentage of the region dimensions.

By default, the position is set to auto.

If the writing direction is horizontal, then the position percentages are relative to the width of the video, otherwise to the height of the video.

A WebVTT cue has a ***computed position*** whose value is that returned by the following algorithm, which is defined in terms of the other aspects of the cue:

1. If the position is numeric between 0 and 100, then return the value of the position and abort these steps. (Otherwise, the position is the special value auto.)
2. If the cue text alignment is left, return 0 and abort these steps.
3. If the cue text alignment is right, return 100 and abort these steps.
4. Otherwise, return 50 and abort these steps.

> Since the default value of the WebVTT cue position alignment is center, if there is no WebVTT cue text alignment setting for a cue, the WebVTT cue position defaults to 50%.

> Even for horizontal cues with right-to-left cue text, the cue box is positioned from the left edge of the video viewport. This allows defining a rendering space template which can be filled with either left-to-right or right-to-left cue text, or both.

For WebVTT cues that have a size other than 100%, and a text alignment of start or end, authors must not use the default auto position.

> When the text alignment is start or end, the auto position is 50%. This is different from left and right aligned text, where the auto position is 0% and 100%, respectively. The above requirement is present because it can be surprising that automatic positioning doesn't work for start or end aligned text. Since cue text can consist of text with left-to-right base direction, or right-to-left base direction, or both (on different lines), such automatic positioning would have unexpected results.

A position alignment

An alignment for the cue box in the dimension of the writing direction, describing what the position is anchored to, one of:

Line-left alignment

The cue box's left side (for horizontal cues) or top side (otherwise) is aligned at the position.

Center alignment

The cue box is centered at the position.

Line-right alignment

The cue box's right side (for horizontal cues) or bottom side (otherwise) is aligned at the position.

Auto alignment

The cue box's alignment depends on the value of the text alignment of the cue.

By default, the position alignment is set to auto.

A WebVTT cue has a ***computed position alignment*** whose value is that returned by the following algorithm, which is defined in terms of other aspects of the cue:

1. If the WebVTT cue position alignment is not auto, then return the value of the WebVTT cue position alignment and abort these steps.
2. If the WebVTT cue text alignment is left, return line-left and abort these steps.
3. If the WebVTT cue text alignment is right, return line-right and abort these steps.
4. If the WebVTT cue text alignment is start, return line-left if the base direction of the cue text is left-to-right, line-right otherwise.
5. If the WebVTT cue text alignment is end, return line-right if the base direction of the cue text is left-to-right, line-left otherwise.
6. Otherwise, return center.

> Since the position always measures from the left of the video (for horizontal cues) or the top (otherwise), the WebVTT cue position alignment line-left value varies between left and top for horizontal and vertical cues.

A size

A number giving the size of the cue box, to be interpreted as a percentage of the video, as defined by the writing direction.

By default, the WebVTT cue size is set to 100%.

If the writing direction is horizontal, then the size percentages are relative to the width of the video, otherwise to the height of the video.

A text alignment

An alignment for all lines of text within the cue box, in the dimension of the writing direction, one of:

Start alignment

The text of each line is individually aligned towards the start side of the box, where the start side for that line is determined by using the CSS rules for 'plaintext' value of the 'unicode-bidi' property. [CSS-WRITING-MODES-3]

Center alignment

The text is aligned centered between the box's start and end sides.

End alignment

The text of each line is individually aligned towards the end side of the box, where the end side for that line is determined by using the CSS rules for 'plaintext' value of the 'unicode-bidi' property. [CSS-WRITING-MODES-3]

Left alignment

The text is aligned to the box's left side (for horizontal cues) or top side (otherwise).

Right alignment

The text is aligned to the box's right side (for horizontal cues) or bottom side (otherwise).

By default, the text alignment is set to center.

> The base direction of each line in a cue (which is used by the Unicode Bidirectional Algorithm to determine the order in which to display the characters in the line) is determined by looking up the first strong directional character in each line, using the CSS 'plaintext' algorithm. In the occasional cases where the first strong character on a line would produce the wrong base direction for that line, the author can use an U+200E LEFT-TO-RIGHT MARK or U+200F RIGHT-TO-LEFT MARK character at the start of the line to correct it. [BIDI]

EXAMPLE 16

In this example, the second cue will have a right-to-left base direction, rendering as "`.I think ,عالي`". (Note that the text below shows all characters left-to-right; a text editor would not necessarily have the same rendering.)

```
WEBVTT

00:00:07.000 --> 00:00:09.000
What was his name again?

00:00:09.000 --> 00:00:11.000
يلاع, I think.
```

To change that line to left-to-right base direction, start the line with an U+200E LEFT-TO-RIGHT MARK character (it can be escaped as "`&lrm;`").

Where the base direction of some embedded text within a line needs to be different from the surrounding text on that line, this can be achieved by using the paired Unicode bidi formatting code characters.

EXAMPLE 17

In this example, assuming no bidi formatting code characters are used, the cue text is rendered as "`I’ve read the book 3 נוילנד times!`" (i.e. the "3" is on the wrong side of the book title) because of the effect of the Unicode Bidirection Algorithm. (Again, the text below shows all characters left-to-right.)

```
WEBVTT

00:00:04.000 --> 00:00:08.000
I’ve read the book דנלוונ 3 times!
```

If a U+2068 FIRST STRONG ISOLATE (FSI) character was placed before the book title and a U+2069 POP DIRECTIONAL ISOLATE (PDI) character after it, the rendering would be the intended "`I’ve read the book נוילנד 3 times!`". (Those characters can be escaped as "`&#x2068;`" and "`&#x2069;`", respectively.)

The default text alignment is center alignment regardless of the base direction of the cue text. To make the text alignment of each line match the base direction of the line (e.g. left for English, right for Hebrew), use start alignment, or end alignment for the opposite alignment.

EXAMPLE 18

In this example, start alignment is used. The first line is left-aligned because the base direction is left-to-right, and the second line is right-aligned because the base direction is right-to-left.

```
WEBVTT

00:00:00.000 --> 00:00:05.000 align:start
Hello!
שלום!
```

This would render as follows:

The left alignment and right alignment can be used to left-align or right-align the cue text regardless of its lines' base direction.

A region

An optional WebVTT region to which a cue belongs.

By default, the region is set to null.

The associated rules for updating the text track rendering of WebVTT cues are the rules for updating the display of WebVTT text tracks.

When a WebVTT cue whose active flag is set has its writing direction, snap-to-lines flag, line, line alignment, position, position alignment, size, text alignment, region, or text change value, then the user agent must empty the text track cue display state, and then immediately run the text track's rules for updating the display of WebVTT text tracks.

§ 3.4. WebVTT caption or subtitle regions

A ***WebVTT region*** represents a subpart of the video viewport and provides a limited rendering area for

WebVTT caption or subtitle cues.

> Regions provide a means to group caption or subtitle cues so the cues can be rendered together, which is particularly important when scrolling up.

Each WebVTT region consists of:

An identifier
: An arbitrary string of zero or more characters other than U+0020 SPACE or U+0009 CHARACTER TABULATION character. The string must not contain the substring "-->" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN). Defaults to the empty string.

A width
: A number giving the width of the box within which the text of each line of the containing cues is to be rendered, to be interpreted as a percentage of the video width. Defaults to 100.

A lines value
: A number giving the number of lines of the box within which the text of each line of the containing cues is to be rendered. Defaults to 3.

> Since a WebVTT region defines a fixed rendering area, a cue that has more lines than the region allows will be clipped. For scrolling regions, the clipping happens at the top, for non-scrolling regions it happens at the bottom.

A region anchor point
: Two numbers giving the x and y coordinates within the region which is anchored to the video viewport and does not change location even when the region does, e.g. because of font size changes. Defaults to (0,100), i.e. the bottom left corner of the region.

A region viewport anchor point
: Two numbers giving the x and y coordinates within the video viewport to which the region anchor point is anchored. Defaults to (0,100), i.e. the bottom left corner of the video viewport.

A scroll value
: One of the following:

  None
  : Indicates that the cues in the region are not to scroll and instead stay fixed at the location they were first painted in.

  Up
  : Indicates that the cues in the region will be added at the bottom of the region and push any

already displayed cues in the region up until all lines of the new cue are visible in the region.

The following diagram illustrates how anchoring of a region to a video viewport works. The black cross is the anchor, orange explains the anchor's offset within the region and green the anchor's offset within the video viewport. Think of it as sticking a pin through a note onto a board:

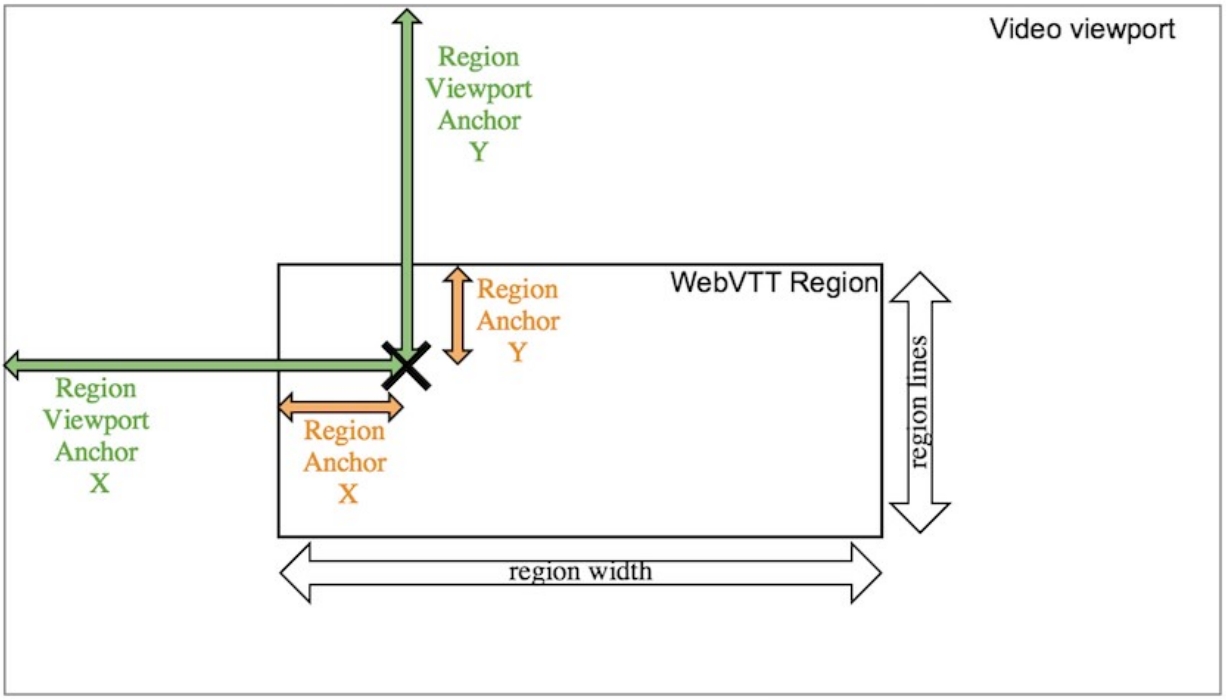


Figure 1 *Image description: Within the video viewport, there is a WebVTT region. Inside the region, there is an anchor point marked with a black cross. The vertical and horizontal distance from the video viewport's edges to the anchor is marked with green arrows, representing the region viewport anchor X and Y offsets. The vertical and horizontal distance from the region's edges to the anchor is marked with orange arrows, representing the region anchor X and Y offsets. The size of the region is represented by the region width for the horizontal axis, and region lines for the vertical axis.*

For parsing, we also need the following:

A text track list of regions
: A list of zero or more WebVTT regions.

§ 3.5. WebVTT chapter cues

A ***WebVTT chapter cue*** is a WebVTT cue whose cue text is interpreted as a chapter title that describes

the chapter as a navigation target.

Chapter cues mark up the timeline of a audio or video file in consecutive, non-overlapping intervals. It is further possible to subdivide these intervals into sub-chapters building a navigation tree.

§ 3.6. WebVTT metadata cues

A ***WebVTT metadata cue*** is a WebVTT cue whose cue text is interpreted as time-aligned metadata.

§ 4. Syntax

§ 4.1. WebVTT file structure

A ***WebVTT file*** must consist of a WebVTT file body encoded as UTF-8 and labeled with the MIME type `text/vtt`. [RFC3629]

A ***WebVTT file body*** consists of the following components, in the following order:

1. An optional U+FEFF BYTE ORDER MARK (BOM) character.
2. The string "`WEBVTT`".
3. Optionally, either a U+0020 SPACE character or a U+0009 CHARACTER TABULATION (tab) character followed by any number of characters that are not U+000A LINE FEED (LF) or U+000D CARRIAGE RETURN (CR) characters.
4. Two or more WebVTT line terminators to terminate the line with the file magic and separate it from the rest of the body.
5. Zero or more WebVTT region definition blocks, WebVTT style blocks and WebVTT comment blocks separated from each other by one or more WebVTT line terminators.
6. Zero or more WebVTT line terminators.
7. Zero or more WebVTT cue blocks and WebVTT comment blocks separated from each other by one or more WebVTT line terminators.
8. Zero or more WebVTT line terminators.

A ***WebVTT line terminator*** consists of one of the following:

- A U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair.

- A single U+000A LINE FEED (LF) character.
- A single U+000D CARRIAGE RETURN (CR) character.

A ***WebVTT region definition block*** consists of the following components, in the given order:

1. The string "`REGION`" (U+0052 LATIN CAPITAL LETTER R, U+0045 LATIN CAPITAL LETTER E, U+0047 LATIN CAPITAL LETTER G, U+0049 LATIN CAPITAL LETTER I, U+004F LATIN CAPITAL LETTER O, U+004E LATIN CAPITAL LETTER N).
2. Zero or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters.
3. A WebVTT line terminator.
4. A WebVTT region settings list.
5. A WebVTT line terminator.

A ***WebVTT style block*** consists of the following components, in the given order:

1. The string "`STYLE`" (U+0053 LATIN CAPITAL LETTER S, U+0054 LATIN CAPITAL LETTER T, U+0059 LATIN CAPITAL LETTER Y, U+004C LATIN CAPITAL LETTER L, U+0045 LATIN CAPITAL LETTER E).
2. Zero or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters.
3. A WebVTT line terminator.
4. Any sequence of zero or more characters other than U+000A LINE FEED (LF) characters and U+000D CARRIAGE RETURN (CR) characters, each optionally separated from the next by a WebVTT line terminator, except that the entire resulting string must not contain the substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN). The string represents a CSS style sheet; the requirements given in the relevant CSS specifications apply. [CSS22]
5. A WebVTT line terminator.

A ***WebVTT cue block*** consists of the following components, in the given order:

1. Optionally, a WebVTT cue identifier followed by a WebVTT line terminator.
2. WebVTT cue timings.
3. Optionally, one or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters followed by a WebVTT cue settings list.
4. A WebVTT line terminator.

5. The ***cue payload***: either WebVTT caption or subtitle cue text, WebVTT chapter title text, or WebVTT metadata text, but it must not contain the substring "-->" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).
6. A WebVTT line terminator.

A WebVTT cue block corresponds to one piece of time-aligned text or data in the WebVTT file, for example one subtitle. The cue payload is the text or data associated with the cue.

A ***WebVTT cue identifier*** is any sequence of one or more characters not containing the substring "-->" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN), nor containing any U+000A LINE FEED (LF) characters or U+000D CARRIAGE RETURN (CR) characters.

A WebVTT cue identifier must be unique amongst all the WebVTT cue identifiers of all WebVTT cues of a WebVTT file.

A WebVTT cue identifier can be used to reference a specific cue, for example from script or CSS.

The ***WebVTT cue timings*** part of a WebVTT cue block consists of the following components, in the given order:

1. A WebVTT timestamp representing the start time offset of the cue. The time represented by this WebVTT timestamp must be greater than or equal to the start time offsets of all previous cues in the file.
2. One or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters.
3. The string "-->" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).
4. One or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters.
5. A WebVTT timestamp representing the end time offset of the cue. The time represented by this WebVTT timestamp must be greater than the start time offset of the cue.

The WebVTT cue timings give the start and end offsets of the WebVTT cue block. Different cues can overlap. Cues are always listed ordered by their start time.

A ***WebVTT timestamp*** consists of the following components, in the given order:

1. Optionally (required if *hours* is non-zero):
    1. Two or more ASCII digits, representing the *hours* as a base ten integer.
    2. A U+003A COLON character (:)
2. Two ASCII digits, representing the *minutes* as a base ten integer in the range $0 \leq minutes \leq 59$.
3. A U+003A COLON character (:)
4. Two ASCII digits, representing the *seconds* as a base ten integer in the range $0 \leq seconds \leq 59$.
5. A U+002E FULL STOP character (.).
6. Three ASCII digits, representing the thousandths of a second *seconds-frac* as a base ten integer.

A WebVTT timestamp is always interpreted relative to the current playback position of the media data that the WebVTT file is to be synchronized with.

A ***WebVTT cue settings list*** consist of a sequence of zero or more ***WebVTT cue settings*** in any order, separated from each other by one or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters. Each setting consists of the following components, in the order given:

1. A WebVTT cue setting name.
2. An optional U+003A COLON (colon) character.
3. An optional WebVTT cue setting value.

A ***WebVTT cue setting name*** and a ***WebVTT cue setting value*** each consist of any sequence of one or more characters other than U+000A LINE FEED (LF) characters and - U+000D CARRIAGE RETURN (CR) characters except that the entire resulting string must not contain the substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).

A ***WebVTT percentage*** consists of the following components:

1. One or more ASCII digits.
2. Optionally:
    1. A U+002E DOT character (.).
    2. One or more ASCII digits.
3. A U+0025 PERCENT SIGN character (%).

When interpreted as a number, a WebVTT percentage must be in the range 0..100.

A ***WebVTT comment block*** consists of the following components, in the given order:

1. The string "`NOTE`".
2. Optionally, the following components, in the given order:
    1. Either:
        - A U+0020 SPACE character or U+0009 CHARACTER TABULATION (tab) character.
        - A WebVTT line terminator.
    2. Any sequence of zero or more characters other than U+000A LINE FEED (LF) characters and U+000D CARRIAGE RETURN (CR) characters, each optionally separated from the next by a WebVTT line terminator, except that the entire resulting string must not contain the substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).
3. A WebVTT line terminator.

A WebVTT comment block is ignored by the parser.

§ 4.2. Types of WebVTT cue payload

§ 4.2.1. WebVTT metadata text

WebVTT metadata text consists of any sequence of zero or more characters other than U+000A LINE FEED (LF) characters and U+000D CARRIAGE RETURN (CR) characters, each optionally separated from the next by a WebVTT line terminator. (In other words, any text that does not have two consecutive WebVTT line terminators and does not start or end with a WebVTT line terminator.)

WebVTT metadata text cues are only useful for scripted applications (e.g. using the `metadata` text track kind in a HTML text track).

§ 4.2.2. WebVTT caption or subtitle cue text

WebVTT caption or subtitle cue text is cue payload that consists of zero or more WebVTT caption or subtitle cue components, in any order, each optionally separated from the next by a WebVTT line terminator.

The ***WebVTT caption or subtitle cue components*** are:

- A WebVTT cue class span.
- A WebVTT cue italics span.

- A WebVTT cue bold span.
- A WebVTT cue underline span.
- A WebVTT cue ruby span.
- A WebVTT cue voice span.
- A WebVTT cue language span.
- A WebVTT cue timestamp.
- A WebVTT cue text span, representing the text of the cue.
- An HTML character reference, representing one or two Unicode code points, as defined in HTML, in the text of the cue. [HTML51]

All WebVTT caption or subtitle cue components bar the HTML character reference may have one or more ***cue component class names*** attached to it by separating the cue component class name from the cue component start tag using the period ('.') notation. The class name must immediately follow the "period" (.).

WebVTT cue internal text consists of an optional WebVTT line terminator, followed by zero or more WebVTT caption or subtitle cue components, in any order, each optionally followed by a WebVTT line terminator.

A ***WebVTT cue class span*** consists of a WebVTT cue span start tag "`c`" that disallows an annotation, WebVTT cue internal text representing cue text, and a WebVTT cue span end tag "`c`".

A ***WebVTT cue italics span*** consists of a WebVTT cue span start tag "`i`" that disallows an annotation, WebVTT cue internal text representing the italicized text, and a WebVTT cue span end tag "`i`".

A ***WebVTT cue bold span*** consists of a WebVTT cue span start tag "`b`" that disallows an annotation, WebVTT cue internal text representing the boldened text, and a WebVTT cue span end tag "`b`".

A ***WebVTT cue underline span*** consists of a WebVTT cue span start tag "`u`" that disallows an annotation, WebVTT cue internal text representing the underlined text, and a WebVTT cue span end tag "`u`".

A ***WebVTT cue ruby span*** consists of the following components, in the order given:

1. A WebVTT cue span start tag "`ruby`" that disallows an annotation.
2. One or more occurrences of the following group of components, in the order given:
    1. WebVTT cue internal text, representing the ruby base.
    2. A WebVTT cue span start tag "`rt`" that disallows an annotation.
    3. A ***WebVTT cue ruby text span***: WebVTT cue internal text, representing the ruby text com-

ponent of the ruby annotation.

4. A WebVTT cue span end tag "rt". If this is the last occurrence of this group of components in the WebVTT cue ruby span, then this last end tag string may be omitted.

3. If the last end tag string was not omitted: Optionally, a WebVTT line terminator.
4. If the last end tag string was not omitted: Zero or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters, each optionally followed by a WebVTT line terminator.
5. A WebVTT cue span end tag "ruby".

Cue positioning controls the positioning of the baseline text, not the ruby text.

Ruby in WebVTT is a subset of the ruby features in HTML. This might be extended in the future to also support an object for ruby base text as well as complex ruby, when these features are more mature in HTML and CSS. [HTML51] [CSS3-RUBY]

A ***WebVTT cue voice span*** consists of the following components, in the order given:

1. A WebVTT cue span start tag "v" that requires an annotation; the annotation represents the name of the voice.
2. WebVTT cue internal text.
3. A WebVTT cue span end tag "v". If this WebVTT cue voice span is the only component of its WebVTT caption or subtitle cue text sequence, then the end tag may be omitted for brevity.

A ***WebVTT cue language span*** consists of the following components, in the order given:

1. A WebVTT cue span start tag "lang" that requires an annotation; the annotation represents the language of the following component, and must be a valid BCP 47 language tag. [BCP47]
2. WebVTT cue internal text.
3. A WebVTT cue span end tag "lang".

The requirement above regarding valid BCP 47 language tag is an authoring requirement, so a conformance checker will do validity checking of the language tag, but other user agents will not.

A ***WebVTT cue span start tag*** has a *tag name* and either requires or disallows an annotation, and consists of the following components, in the order given:

1. A U+003C LESS-THAN SIGN character (<).

2. The *tag name*.
3. Zero or more occurrences of the following sequence:
    1. U+002E FULL STOP character (.)
    2. One or more characters other than U+0009 CHARACTER TABULATION (tab) characters, U+000A LINE FEED (LF) characters, U+000D CARRIAGE RETURN (CR) characters, U+0020 SPACE characters, U+0026 AMPERSAND characters (&), U+003C LESS-THAN SIGN characters (<), U+003E GREATER-THAN SIGN characters (>), and U+002E FULL STOP characters (.), representing a class that describes the cue span's significance.
4. If the start tag requires an annotation: a U+0020 SPACE character or a U+0009 CHARACTER TABULATION (tab) character, followed by one or more of the following components, the concatenation of their representations having a value that contains at least one character other than U+0020 SPACE and U+0009 CHARACTER TABULATION (tab) characters:
    - WebVTT cue span start tag annotation text, representing the text of the annotation.
    - An HTML character reference, representing one or two Unicode code points, as defined in HTML, in the text of the annotation. [HTML51]
5. A U+003E GREATER-THAN SIGN character (>).

A ***WebVTT cue span end tag*** has a *tag name* and consists of the following components, in the order given:

1. A U+003C LESS-THAN SIGN character (<).
2. U+002F SOLIDUS character (/).
3. The *tag name*.
4. A U+003E GREATER-THAN SIGN character (>).

A ***WebVTT cue timestamp*** consists of a U+003C LESS-THAN SIGN character (<), followed by a WebVTT timestamp representing the time that the given point in the cue becomes active, followed by a U+003E GREATER-THAN SIGN character (>). The time represented by the WebVTT timestamp must be greater than the times represented by any previous WebVTT cue timestamps in the cue, as well as greater than the cue's start time offset, and less than the cue's end time offset.

A ***WebVTT cue text span*** consists of one or more characters other than U+000A LINE FEED (LF) characters, U+000D CARRIAGE RETURN (CR) characters, U+0026 AMPERSAND characters (&), and U+003C LESS-THAN SIGN characters (<).

WebVTT cue span start tag annotation text consists of one or more characters other than U+000A LINE FEED (LF) characters, U+000D CARRIAGE RETURN (CR) characters, U+0026

AMPERSAND characters (&), and U+003E GREATER-THAN SIGN characters (>).

§ 4.2.3. WebVTT chapter title text

WebVTT chapter title text is cue text that makes use of zero or more of the following components, each optionally separated from the next by a WebVTT line terminator:

- WebVTT cue text span
- HTML character reference [HTML51]

§ 4.3. WebVTT region settings

A WebVTT cue settings list can contain a reference to a WebVTT region. To define a region, a WebVTT region definition block is specified.

The ***WebVTT region settings list*** consists of zero or more of the following components, in any order, separated from each other by one or more U+0020 SPACE characters, U+0009 CHARACTER TABULATION (tab) characters, or WebVTT line terminators, except that the string must not contain two consecutive WebVTT line terminators. Each component must not be included more than once per WebVTT region settings list string.

- A WebVTT region identifier setting.
- A WebVTT region width setting.
- A WebVTT region lines setting.
- A WebVTT region anchor setting.
- A WebVTT region viewport anchor setting.
- A WebVTT region scroll setting.

> The WebVTT region settings list gives configuration options regarding the dimensions, positioning and anchoring of the region. For example, it allows a group of cues within a region to be anchored in the center of the region and the center of the video viewport. In this example, when the font size grows, the region grows uniformly in all directions from the center.

A ***WebVTT region identifier setting*** consists of the following components, in the order given:

1. The string "`id`".

2. A U+003A COLON character (:).

3. An arbitrary string of one or more characters other than ASCII whitespace. The string must not contain the substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).

A WebVTT region identifier setting must be unique amongst all the WebVTT region identifier settings of all WebVTT regions of a WebVTT file.

A WebVTT region identifier setting must be present in each WebVTT cue settings list. Without an identifier, it is not possible to associate a WebVTT cue with a WebVTT region in the syntax.

> The WebVTT region identifier setting gives a name to the region so it can be referenced by the cues that belong to the region.

A ***WebVTT region width setting*** consists of the following components, in the order given:

1. The string "`width`".

2. A U+003A COLON character (:).

3. A WebVTT percentage.

> The WebVTT region width setting provides a fixed width as a percentage of the video width for the region into which cues are rendered and based on which alignment is calculated.

A ***WebVTT region lines setting*** consists of the following components, in the order given:

1. The string "`lines`".

2. A U+003A COLON character (:).

3. One or more ASCII digits.

> The WebVTT region lines setting provides a fixed height as a number of lines for the region into which cues are rendered. As such, it defines the height of the roll-up region if it is a scroll region.

A ***WebVTT region anchor setting*** consists of the following components, in the order given:

1. The string "`regionanchor`".

2. A U+003A COLON character (:).

3. A WebVTT percentage.

4. A U+002C COMMA character (,).

5. A WebVTT percentage.

The WebVTT region anchor setting provides a tuple of two percentages that specify the point within the region box that is fixed in location. The first percentage measures the x-dimension and the second percentage y-dimension from the top left corner of the region box. If no WebVTT region anchor setting is given, the anchor defaults to 0%, 100% (i.e. the bottom left corner).

A ***WebVTT region viewport anchor setting*** consists of the following components, in the order given:

1. The string "`viewportanchor`".

2. A U+003A COLON character (:).

3. A WebVTT percentage.

4. A U+002C COMMA character (,).

5. A WebVTT percentage.

The WebVTT region viewport anchor setting provides a tuple of two percentages that specify the point within the video viewport that the region anchor point is anchored to. The first percentage measures the x-dimension and the second percentage measures the y-dimension from the top left corner of the video viewport box. If no region viewport anchor is given, it defaults to 0%, 100% (i.e. the bottom left corner).

For browsers, the region maps to an absolute positioned CSS box relative to the video viewport, i.e. there is a relative positioned box that represents the video viewport relative to which the regions are absolutely positioned. Overflow is hidden.

A ***WebVTT region scroll setting*** consists of the following components, in the order given:

1. The string "`scroll`".

2. A U+003A COLON character (:).

3. The string "`up`".

> The WebVTT region scroll setting specifies whether cues rendered into the region are allowed to move out of their initial rendering place and roll up, i.e. move towards the top of the video viewport. If the scroll setting is omitted, cues do not move from their rendered position.

> Cues are added to a region one line at a time below existing cue lines. When an existing rendered cue line is removed, and it was above another already rendered cue line, that cue line moves into its space, thus scrolling in the given direction. If there is not enough space for a new cue line to be added to a region, the top-most cue line is pushed off the visible region (thus slowly becoming invisible as it moves into overflow:hidden). This eventually makes space for the new cue line and allows it to be added.

> When there is no scroll direction, cue lines are added in the empty line closest to the line in the bottom of the region. If no empty line is available, the oldest line is replaced.

§ 4.4. WebVTT cue settings

A WebVTT cue setting is part of a WebVTT cue settings list and provides configuration options regarding the position and alignment of the cue box and the cue text within.

> For example, a set of WebVTT cue settings may allow a cue box to be aligned to the left or positioned at the top right with the cue text within center aligned.

The current available WebVTT cue settings that may appear in a WebVTT cue settings list are:

- A WebVTT vertical text cue setting.
- A WebVTT line cue setting.
- A WebVTT position cue setting.
- A WebVTT size cue setting.
- A WebVTT alignment cue setting.
- A WebVTT region cue setting.

Each of these setting must not be included more than once per WebVTT cue settings list.

A ***WebVTT vertical text cue setting*** is a WebVTT cue setting that consists of the following components, in the order given:

1. The string "`vertical`" as the WebVTT cue setting name.
2. A U+003A COLON character (:).
3. One of the following strings as the WebVTT cue setting value: "`rl`", "`lr`".

A WebVTT vertical text cue setting configures the cue to use vertical text layout rather than horizontal text layout. Vertical text layout is sometimes used in Japanese, for example. The default is horizontal layout.

A ***WebVTT line cue setting*** consists of the following components, in the order given:

1. The string "`line`" as the WebVTT cue setting name.
2. A U+003A COLON character (:).
3. As the WebVTT cue setting value:
    1. an offset value, either:
       To represent a specific offset relative to the video viewport
       A WebVTT percentage.

       Or to represent a line number
        1. Optionally a U+002D HYPHEN-MINUS character (-).
        2. One or more ASCII digits.
    2. An optional alignment value consisting of the following components:
        1. A U+002C COMMA character (,).
        2. One of the following strings: "`start`", "`center`", "`end`"

A WebVTT line cue setting configures the offset of the cue box from the video viewport's edge in the direction orthogonal to the writing direction. For horizontal cues, this is the vertical offset from the top of the video viewport, for vertical cues, it's the horizontal offset. The offset is for the start, center, or end of the cue box, depending on the WebVTT cue line alignment value - start by default. The offset can be given either as a percentage of the relevant writing-mode dependent video viewport dimension or as a line number. Line numbers are based on the size of the first line of the cue. Positive line numbers count from the start of the video viewport (the first line is numbered 0), negative line numbers from the end of the video viewport (the last line is numbered −1).

A ***WebVTT position cue setting*** consists of the following components, in the order given:

1. The string "`position`" as the WebVTT cue setting name.

2. A U+003A COLON character (:).

3. As the WebVTT cue setting value:

    1. a position value consisting of: a WebVTT percentage.

    2. an optional alignment value consisting of:

        1. A U+002C COMMA character (,).

        2. One of the following strings: "`line-left`", "`center`", "`line-right`"

A WebVTT position cue setting configures the indent position of the cue box in the direction orthogonal to the WebVTT line cue setting. For horizontal cues, this is the horizontal position. The cue position is given as a percentage of the video viewport. The positioning is for the line-left, center, or line-right of the cue box, depending on the cue's computed position alignment, which is overridden by the WebVTT position cue setting.

A ***WebVTT size cue setting*** consists of the following components, in the order given:

1. The string "`size`" as the WebVTT cue setting name.

2. A U+003A COLON character (:).

3. As the WebVTT cue setting value: a WebVTT percentage.

A WebVTT size cue setting configures the size of the cue box in the same direction as the WebVTT position cue setting. For horizontal cues, this is the width of the cue box. It is given as a percentage of the width of the video viewport.

A ***WebVTT alignment cue setting*** consists of the following components, in the order given:

1. The string "`align`" as the WebVTT cue setting name.

2. A U+003A COLON character (:).

3. One of the following strings as the WebVTT cue setting value: "`start`", "`center`", "`end`", "`left`", "`right`"

A WebVTT alignment cue setting configures the alignment of the text within the cue. The "`start`" and "`end`" keywords are relative to the cue text's lines' base direction; for left-to-right English text, "`start`" means left-aligned.

A ***WebVTT region cue setting*** consists of the following components, in the order given:

1. The string "`region`" as the WebVTT cue setting name.

2. A U+003A COLON character (:).

3. As the WebVTT cue setting value: a WebVTT region identifier.

A WebVTT region cue setting configures a cue to become part of a region by referencing the region's identifier unless the cue has a "vertical", "line" or "size" cue setting. If a cue is part of a region, its cue settings for "position" and "align" are applied to the line boxes in the cue relative to the region box and the cue box width and height are calculated relative to the region dimensions rather than the viewport dimensions.

§ 4.5. Properties of cue sequences

§ 4.5.1. WebVTT file using only nested cues

A WebVTT file whose cues all follow the following rules is said to be a ***WebVTT file using only nested cues***:

given any two cues *cue1* and *cue2* with start and end time offsets (*x1*, *y1*) and (*x2*, *y2*) respectively,

- either *cue1* lies fully within *cue2*, i.e. *x1* >= *x2* and *y1* <= *y2*
- or *cue1* fully contains *cue2*, i.e. *x1* <= *x2* and *y1* >= *y2*.

EXAMPLE 19

The following example matches this definition:

```
WEBVTT

00:00.000 --> 01:24.000
Introduction

00:00.000 --> 00:44.000
Topics

00:44.000 --> 01:19.000
Presenters

01:24.000 --> 05:00.000
Scrolling Effects

01:35.000 --> 03:00.000
Achim’s Demo

03:00.000 --> 05:00.000
Timeline Panel
```

Notice how you can express the cues in this WebVTT file as a tree structure:

- WebVTT file
  - Introduction
    - Topics
    - Presenters
  - Scrolling Effects
    - Achim’s Demo
    - Timeline Panel

If the file has cues that can’t be expressed in this fashion, then they don’t match the definition of a WebVTT file using only nested cues. For example:

```
WEBVTT

00:00.000 --> 01:00.000
The First Minute

00:30.000 --> 01:30.000
The Final Minute
```

In this ninety-second example, the two cues partly overlap, with the first ending before the second ends and the second starting before the first ends. This therefore is not a WebVTT file using only nested cues.

§ 4.6. Types of WebVTT files

The syntax definition of WebVTT files allows authoring of a wide variety of WebVTT files with a mix of cues. However, only a small subset of WebVTT file types are typically authored.

Conformance checkers, when validating WebVTT files, may offer to restrict syntax checking for validating these types.

§ 4.6.1. WebVTT file using metadata content

A WebVTT file whose cues all have a cue payload that is WebVTT metadata text is said to be a ***WebVTT file using metadata content***.

§ 4.6.2. WebVTT file using chapter title text

A ***WebVTT file using chapter title text*** is a WebVTT file using only nested cues whose cues all have a cue payload that is WebVTT chapter title text.

§ 4.6.3. WebVTT file using caption or subtitle cue text

A WebVTT file whose cues all have a cue payload that is WebVTT caption or subtitle cue text is said to be a ***WebVTT file using caption or subtitle cue text***.

§ 5. Default classes for WebVTT Caption or Subtitle Cue Components

Many captioning formats have simple ways of specifying a limited subset of text colors and background colors for text. Therefore, the WebVTT spec makes available a set of default cue component class names for WebVTT caption or subtitle cue components that authors can use in a standard way to mark up colored text and text background.

> User agents that support CSS style sheets may implement this section through adding User Agent stylesheets.

§ 5.1. Default text colors

WebVTT caption or subtitle cue components that have one or more class names matching those in the first cell of a row in the table below must set their 'color' property as presentational hints to the value in the second cell of the row:

class names	'color' value
`white`	'rgba(255,255,255,1)'
`lime`	'rgba(0,255,0,1)'
`cyan`	'rgba(0,255,255,1)'
`red`	'rgba(255,0,0,1)'
`yellow`	'rgba(255,255,0,1)'
`magenta`	'rgba(255,0,255,1)'
`blue`	'rgba(0,0,255,1)'
`black`	'rgba(0,0,0,1)'

> If your background is captioning, don't get confused: The color for the class `lime` is what has traditionally been used in captioning under the name 'green' (e.g. 608/708).

Do not use the classes `blue` and `black` on the default dark background, since they result in unreadable text. In general, please refer to WCAG for guidance on color contrast [WCAG20] and make sure to take into account the text color, background color and also the video's color.

§ 5.2. Default text background colors

WebVTT caption or subtitle cue components that have one or more class names matching those in the first cell of a row in the table below must set their 'background-color' property as presentational hints to the value in the second cell of the row:

class names	'background' value
`bg_white`	'rgba(255,255,255,1)'
`bg_lime`	'rgba(0,255,0,1)'
`bg_cyan`	'rgba(0,255,255,1)'
`bg_red`	'rgba(255,0,0,1)'
`bg_yellow`	'rgba(255,255,0,1)'
`bg_magenta`	'rgba(255,0,255,1)'
`bg_blue`	'rgba(0,0,255,1)'
`bg_black`	'rgba(0,0,0,1)'

The color for the class `bg_lime` is what has traditionally been used in captioning under the name 'green' (e.g. 608/708).

For the purpose of determining the cascade of the color and background classes, the order of appearance determines the cascade of the classes.

EXAMPLE 20

This example shows how to use the classes.

```
WEBVTT

02:00.000 --> 02:05.000
<c.yellow.bg_blue>This is yellow text on a blue background</c>

04:00.000 --> 04:05.000
<c.yellow.bg_blue.magenta.bg_black>This is magenta text on a black background</c
```

Default classes can be changed by authors, e.g. ::cue(.yellow) {color:cyan} would change all .yellow classed text to cyan.

§ 6. Parsing

WebVTT file parsing is the same for all types of WebVTT files, including captions, subtitles, chapters, or metadata. Most of the steps will be skipped for chapters or metadata files.

§ 6.1. WebVTT file parsing

A ***WebVTT parser***, given an input byte stream, a text track list of cues *output*, and a collection of CSS style sheets *stylesheets*, must decode the byte stream using the UTF-8 decode algorithm, and then must parse the resulting string according to the WebVTT parser algorithm below. This results in WebVTT cues being added to *output*, and CSS style sheets being added to *stylesheets*. [RFC3629]

A WebVTT parser, specifically its conversion and parsing steps, is typically run asynchronously, with the input byte stream being updated incrementally as the resource is downloaded; this is called an ***incremental WebVTT parser***.

A WebVTT parser verifies a file signature before parsing the provided byte stream. If the stream lacks this WebVTT file signature, then the parser aborts.

The ***WebVTT parser algorithm*** is as follows:

1. Let *input* be the string being parsed, after conversion to Unicode, and with the following transformations applied:

- Replace all U+0000 NULL characters by U+FFFD REPLACEMENT CHARACTERs.
- Replace each U+000D CARRIAGE RETURN U+000A LINE FEED (CRLF) character pair by a single U+000A LINE FEED (LF) character.
- Replace all remaining U+000D CARRIAGE RETURN characters by U+000A LINE FEED (LF) characters.

2. Let *position* be a pointer into *input*, initially pointing at the start of the string. In an incremental WebVTT parser, when this algorithm (or further algorithms that it uses) moves the *position* pointer, the user agent must wait until appropriate further characters from the byte stream have been added to *input* before moving the pointer, so that the algorithm never reads past the end of the *input* string. Once the byte stream has ended, and all characters have been added to *input*, then the *position* pointer may, when so instructed by the algorithms, be moved past the end of *input*.
3. Let *seen cue* be false.
4. If *input* is less than six characters long, then abort these steps. The file does not start with the correct WebVTT file signature and was therefore not successfully processed.
5. If *input* is exactly six characters long but does not exactly equal "`WEBVTT`", then abort these steps. The file does not start with the correct WebVTT file signature and was therefore not successfully processed.
6. If *input* is more than six characters long but the first six characters do not exactly equal "`WEBVTT`", or the seventh character is not a U+0020 SPACE character, a U+0009 CHARACTER TABULATION (tab) character, or a U+000A LINE FEED (LF) character, then abort these steps. The file does not start with the correct WebVTT file signature and was therefore not successfully processed.
7. collect a sequence of code points that are *not* U+000A LINE FEED (LF) characters.
8. If *position* is past the end of *input*, then abort these steps. The file was successfully processed, but it contains no useful data and so no WebVTT cues were added to *output*.
9. The character indicated by *position* is a U+000A LINE FEED (LF) character. Advance *position* to the next character in *input*.
10. If *position* is past the end of *input*, then abort these steps. The file was successfully processed, but it contains no useful data and so no WebVTT cues were added to *output*.
11. *Header*: If the character indicated by *position* is not a U+000A LINE FEED (LF) character, then collect a WebVTT block with the *in header* flag set. Otherwise, advance *position* to the next char-

acter in *input*.

12. collect a sequence of code points that are U+000A LINE FEED (LF) characters.
13. Let *regions* be an empty text track list of regions.
14. *Block loop*: While *position* doesn't point past the end of *input*:

    1. Collect a WebVTT block, and let *block* be the returned value.
    2. If *block* is a WebVTT cue, add *block* to the text track list of cues *output*.
    3. Otherwise, if *block* is a CSS style sheet, add *block* to *stylesheets*.
    4. Otherwise, if *block* is a WebVTT region object, add *block* to *regions*.
    5. collect a sequence of code points that are U+000A LINE FEED (LF) characters.

15. *End*: The file has ended. Abort these steps. The WebVTT parser has finished. The file was successfully processed.

When the algorithm above says to ***collect a WebVTT block***, optionally with a flag *in header* set, the user agent must run the following steps:

1. Let *input*, *position*, *seen cue* and *regions* be the same variables as those of the same name in the algorithm that invoked these steps.
2. Let *line count* be zero.
3. Let *previous position* be *position*.
4. Let *line* be the empty string.
5. Let *buffer* be the empty string.
6. Let *seen EOF* be false.
7. Let *seen arrow* be false.
8. Let *cue* be null.
9. Let *stylesheet* be null.
10. Let *region* be null.

11. *Loop*: Run these substeps in a loop:

    1. collect a sequence of code points that are *not* U+000A LINE FEED (LF) characters. Let *line* be those characters, if any.

    2. Increment *line count* by 1.

    3. If *position* is past the end of *input*, let *seen EOF* be true. Otherwise, the character indicated by *position* is a U+000A LINE FEED (LF) character; advance *position* to the next character in *input*.

    4. If *line* contains the three-character substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN), then run these substeps:

        1. If *in header* is not set and at least one of the following conditions are true:

            - *line count* is 1
            - *line count* is 2 and *seen arrow* is false

        ...then run these substeps:

            1. Let *seen arrow* be true.

            2. Let *previous position* be *position*.

            3. *Cue creation*: Let *cue* be a new WebVTT cue and initialize it as follows:

                1. Let *cue*'s text track cue identifier be *buffer*.

                2. Let *cue*'s text track cue pause-on-exit flag be false.

                3. Let *cue*'s WebVTT cue region be null.

                4. Let *cue*'s WebVTT cue writing direction be horizontal.

                5. Let *cue*'s WebVTT cue snap-to-lines flag be true.

                6. Let *cue*'s WebVTT cue line be auto.

                7. Let *cue*'s WebVTT cue line alignment be start alignment.

                8. Let *cue*'s WebVTT cue position be auto.

                9. Let *cue*'s WebVTT cue position alignment be auto.

10. Let *cue*'s WebVTT cue size be 100.

11. Let *cue*'s WebVTT cue text alignment be center alignment.

12. Let *cue*'s cue text be the empty string.

4. Collect WebVTT cue timings and settings from *line* using *regions* for *cue*. If that fails, let *cue* be null. Otherwise, let *buffer* be the empty string and let *seen cue* be true.

Otherwise, let *position* be *previous position* and break out of *loop*.

5. Otherwise, if *line* is the empty string, break out of *loop*.

6. Otherwise, run these substeps:

1. If *in header* is not set and *line count* is 2, run these substeps:

1. If *seen cue* is false and *buffer* starts with the substring "`STYLE`" (U+0053 LATIN CAPITAL LETTER S, U+0054 LATIN CAPITAL LETTER T, U+0059 LATIN CAPITAL LETTER Y, U+004C LATIN CAPITAL LETTER L, U+0045 LATIN CAPITAL LETTER E), and the remaining characters in *buffer* (if any) are all ASCII whitespace, then run these substeps:

1. Let *stylesheet* be the result of creating a CSS style sheet, with the following properties: [CSSOM]

location
null

parent CSS style sheet
null

owner node
null

owner CSS rule
null

media
The empty string.

title
The empty string.

alternate flag
Unset.

origin-clean flag
Set.

2. Let *buffer* be the empty string.

2. Otherwise, if *seen cue* is false and *buffer* starts with the substring "`REGION`" (U+0052 LATIN CAPITAL LETTER R, U+0045 LATIN CAPITAL LETTER E, U+0047 LATIN CAPITAL LETTER G, U+0049 LATIN CAPITAL LETTER I, U+004F LATIN CAPITAL LETTER O, U+004E LATIN CAPITAL LETTER N), and the remaining characters in *buffer* (if any) are all ASCII whitespace, then run these substeps:

1. *Region creation*: Let *region* be a new WebVTT region.

2. Let *region*'s identifier be the empty string.

3. Let *region*'s width be 100.

4. Let *region*'s lines be 3.

5. Let *region*'s anchor point be (0,100).

6. Let *region*'s viewport anchor point be (0,100).

7. Let *region*'s scroll value be none.

8. Let *buffer* be the empty string.

2. If *buffer* is not the empty string, append a U+000A LINE FEED (LF) character to *buffer*.

3. Append *line* to *buffer*.

4. Let *previous position* be *position*.

7. If *seen EOF* is true, break out of *loop*.

12. If *cue* is not null, let the cue text of *cue* be *buffer*, and return *cue*.

13. Otherwise, if *stylesheet* is not null, then Parse a stylesheet from *buffer*. If it returned a list of rules, assign the list as *stylesheet*'s CSS rules; otherwise, set *stylesheet*'s CSS rules to an empty list. [CSSOM] [CSS-SYNTAX-3] Finally, return *stylesheet*.

14. Otherwise, if *region* is not null, then collect WebVTT region settings from *buffer* using *region* for the results. Construct a WebVTT Region Object from *region*, and return it.

15. Otherwise, return null.

§ 6.2. WebVTT region settings parsing

When the WebVTT parser algorithm says to ***collect WebVTT region settings*** from a string *input* for a text track, the user agent must run the following algorithm.

A ***WebVTT region object*** is a conceptual construct to represent a WebVTT region that is used as a root node for lists of WebVTT node objects. This algorithm returns a list of WebVTT Region Objects.

1. Let *settings* be the result of splitting *input* on spaces.
2. For each token *setting* in the list *settings*, run the following substeps:
    1. If *setting* does not contain a U+003A COLON character (:), or if the first U+003A COLON character (:) in *setting* is either the first or last character of *setting*, then jump to the step labeled *next setting*.
    2. Let *name* be the leading substring of *setting* up to and excluding the first U+003A COLON character (:) in that string.
    3. Let *value* be the trailing substring of *setting* starting from the character immediately after the first U+003A COLON character (:) in that string.
    4. Run the appropriate substeps that apply for the value of *name*, as follows:

        If *name* is a case-sensitive match for `"id"`

        Let *region*'s identifier be *value*.

        Otherwise if *name* is a case-sensitive match for `"width"`

        If parse a percentage string from *value* returns a *percentage*, let *region*'s WebVTT region width be *percentage*.

        Otherwise if *name* is a case-sensitive match for `"lines"`

        1. If *value* contains any characters other than ASCII digits, then jump to the step labeled *next setting*.
        2. Interpret *value* as an integer, and let *number* be that number.
        3. Let *region*'s WebVTT region lines be *number*.

Otherwise if *name* is a case-sensitive match for "regionanchor"

1. If *value* does not contain a U+002C COMMA character (,), then jump to the step labeled *next setting*.
2. Let *anchorX* be the leading substring of *value* up to and excluding the first U+002C COMMA character (,) in that string.
3. Let *anchorY* be the trailing substring of *value* starting from the character immediately after the first U+002C COMMA character (,) in that string.
4. If parse a percentage string from *anchorX* or parse a percentage string from *anchorY* don't return a *percentage*, then jump to the step labeled *next setting*.
5. Let *region*'s WebVTT region anchor point be the tuple of the *percentage* values calculated from *anchorX* and *anchorY*.

Otherwise if *name* is a case-sensitive match for "viewportanchor"

1. If *value* does not contain a U+002C COMMA character (,), then jump to the step labeled *next setting*.
2. Let *viewportanchorX* be the leading substring of *value* up to and excluding the first U+002C COMMA character (,) in that string.
3. Let *viewportanchorY* be the trailing substring of *value* starting from the character immediately after the first U+002C COMMA character (,) in that string.
4. If parse a percentage string from *viewportanchorX* or parse a percentage string from *viewportanchorY* don't return a *percentage*, then jump to the step labeled *next setting*.
5. Let *region*'s WebVTT region viewport anchor point be the tuple of the *percentage* values calculated from *viewportanchorX* and *viewportanchorY*.

Otherwise if *name* is a case-sensitive match for "scroll"

1. If *value* is a case-sensitive match for the string "up", then let *region*'s scroll value be up.

5. *Next setting*: Continue to the next setting, if any.

The rules to ***parse a percentage string*** are as follows. This will return either a number in the range 0..100, or nothing. If at any point the algorithm says that it "fails", this means that it is aborted at that point and returns nothing.

1. Let *input* be the string being parsed.
2. If *input* does not match the syntax for a WebVTT percentage, then fail.
3. Remove the last character from *input*.
4. Let *percentage* be the result of parsing *input* using the rules for parsing floating-point number values. [HTML51]
5. If *percentage* is an error, is less than 0, or is greater than 100, then fail.
6. Return *percentage*.

§ 6.3. WebVTT cue timings and settings parsing

When the algorithm above says to ***collect WebVTT cue timings and settings*** from a string *input* using a text track list of regions *regions* for a WebVTT cue *cue*, the user agent must run the following algorithm.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Skip whitespace.
4. Collect a WebVTT timestamp. If that algorithm fails, then abort these steps and return failure. Otherwise, let *cue*'s text track cue start time be the collected time.
5. Skip whitespace.
6. If the character at *position* is not a U+002D HYPHEN-MINUS character (-) then abort these steps and return failure. Otherwise, move *position* forwards one character.
7. If the character at *position* is not a U+002D HYPHEN-MINUS character (-) then abort these steps and return failure. Otherwise, move *position* forwards one character.
8. If the character at *position* is not a U+003E GREATER-THAN SIGN character (>) then abort these steps and return failure. Otherwise, move *position* forwards one character.
9. Skip whitespace.

10. Collect a WebVTT timestamp. If that algorithm fails, then abort these steps and return failure. Otherwise, let *cue*'s text track cue end time be the collected time.

11. Let *remainder* be the trailing substring of *input* starting at *position*.

12. Parse the WebVTT cue settings from *remainder* using *regions* for *cue*.

When the user agent is to ***parse the WebVTT cue settings*** from a string *input* using a text track list of regions *regions* for a text track cue *cue*, the user agent must run the following steps:

1. Let *settings* be the result of splitting *input* on spaces.

2. For each token *setting* in the list *settings*, run the following substeps:

    1. If *setting* does not contain a U+003A COLON character (:), or if the first U+003A COLON character (:) in *setting* is either the first or last character of *setting*, then jump to the step labeled *next setting*.

    2. Let *name* be the leading substring of *setting* up to and excluding the first U+003A COLON character (:) in that string.

    3. Let *value* be the trailing substring of *setting* starting from the character immediately after the first U+003A COLON character (:) in that string.

    4. Run the appropriate substeps that apply for the value of *name*, as follows:

        If *name* is a case-sensitive match for `"region"`

        1. Let *cue*'s WebVTT cue region be the last WebVTT region in *regions* whose WebVTT region identifier is *value*, if any, or null otherwise.

        If *name* is a case-sensitive match for `"vertical"`

        1. If *value* is a case-sensitive match for the string `"rl"`, then let *cue*'s WebVTT cue writing direction be vertical growing left.

        2. Otherwise, if *value* is a case-sensitive match for the string `"lr"`, then let *cue*'s WebVTT cue writing direction be vertical growing right.

        3. If *cue*'s WebVTT cue writing direction is not horizontal, let *cue*'s WebVTT cue region be null (there are no vertical regions).

        If *name* is a case-sensitive match for `"line"`

        1. If *value* contains a U+002C COMMA character (,), then let *linepos* be the lead-

ing substring of *value* up to and excluding the first U+002C COMMA character (,) in that string and let *linealign* be the trailing substring of *value* starting from the character immediately after the first U+002C COMMA character (,) in that string.

2. Otherwise let *linepos* be the full *value* string and *linealign* be null.

3. If *linepos* does not contain at least one ASCII digit, then jump to the step labeled *next setting*.

4. **If the last character in *linepos* is a U+0025 PERCENT SIGN character (%)**

   If parse a percentage string from *linepos* doesn't fail, let *number* be the returned *percentage*, otherwise jump to the step labeled *next setting*.

   Otherwise

   1. If *linepos* contains any characters other than U+002D HYPHEN-MINUS characters (-), ASCII digits, and U+002E DOT character (.), then jump to the step labeled *next setting*.

   2. If any character in *linepos* other than the first character is a U+002D HYPHEN-MINUS character (-), then jump to the step labeled *next setting*.

   3. If there are more than one U+002E DOT characters (.), then jump to the step labeled *next setting*.

   4. If there is a U+002E DOT character (.) and the character before or the character after is not an ASCII digit, or if the U+002E DOT character (.) is the first or the last character, then jump to the step labeled *next setting*.

   5. Let *number* be the result of parsing *linepos* using the rules for parsing floating-point number values. [HTML51]

   6. If *number* is an error, then jump to the step labeled *next setting*.

5. If *linealign* is a case-sensitive match for the string "`start`", then let *cue*'s WebVTT cue line alignment be start alignment.

6. Otherwise, if *linealign* is a case-sensitive match for the string "`center`", then let *cue*'s WebVTT cue line alignment be center alignment.

7. Otherwise, if *linealign* is a case-sensitive match for the string "`end`", then let *cue*'s WebVTT cue line alignment be end alignment.

8. Otherwise, if *linealign* is not null, then jump to the step labeled *next setting*.

9. Let *cue*'s WebVTT cue line be *number*.

10. If the last character in *linepos* is a U+0025 PERCENT SIGN character (%), then let *cue*'s WebVTT cue snap-to-lines flag be false. Otherwise, let it be true.

11. If *cue*'s WebVTT cue line is not auto, let *cue*'s WebVTT cue region be null (the cue has been explicitly positioned with a line offset and thus drops out of the region).

If *name* is a case-sensitive match for "`position`"

1. If *value* contains a U+002C COMMA character (,), then let *colpos* be the leading substring of *value* up to and excluding the first U+002C COMMA character (,) in that string and let *colalign* be the trailing substring of *value* starting from the character immediately after the first U+002C COMMA character (,) in that string.

2. Otherwise let *colpos* be the full *value* string and *colalign* be null.

3. If parse a percentage string from *colpos* doesn't fail, let *number* be the returned *percentage*, otherwise jump to the step labeled *next setting* (position's value remains the special value auto).

4. If *colalign* is a case-sensitive match for the string "`line-left`", then let *cue*'s WebVTT cue position alignment be line-left alignment.

5. Otherwise, if *colalign* is a case-sensitive match for the string "`center`", then let *cue*'s WebVTT cue position alignment be center alignment.

6. Otherwise, if *colalign* is a case-sensitive match for the string "`line-right`", then let *cue*'s WebVTT cue position alignment be line-right alignment.

7. Otherwise, if *colalign* is not null, then jump to the step labeled *next setting*.

8. Let *cue*'s position be *number*.

If *name* is a case-sensitive match for "`size`"

1. If parse a percentage string from *value* doesn't fail, let *number* be the returned *percentage*, otherwise jump to the step labeled *next setting*.

2. Let *cue*'s WebVTT cue size be *number*.

3. If *cue*'s WebVTT cue size is not 100, let *cue*'s WebVTT cue region be null (the cue has been explicitly sized and thus drops out of the region).

If *name* is a case-sensitive match for "`align`"

1. If *value* is a case-sensitive match for the string "`start`", then let *cue*'s WebVTT cue text alignment be start alignment.

2. If *value* is a case-sensitive match for the string "`center`", then let *cue*'s WebVTT cue text alignment be center alignment.

3. If *value* is a case-sensitive match for the string "`end`", then let *cue*'s WebVTT cue text alignment be end alignment.

4. If *value* is a case-sensitive match for the string "`left`", then let *cue*'s WebVTT cue text alignment be left alignment.

5. If *value* is a case-sensitive match for the string "`right`", then let *cue*'s WebVTT cue text alignment be right alignment.

5. *Next setting*: Continue to the next token, if any.

When this specification says that a user agent is to ***collect a WebVTT timestamp***, the user agent must run the following steps:

1. Let *input* and *position* be the same variables as those of the same name in the algorithm that invoked these steps.

2. Let *most significant units* be *minutes*.

3. If *position* is past the end of *input*, return an error and abort these steps.

4. If the character indicated by *position* is not an ASCII digit, then return an error and abort these steps.

5. Collect a sequence of code points that are ASCII digits, and let *string* be the collected substring.

6. Interpret *string* as a base-ten integer. Let $value_1$ be that integer.

7. If *string* is not exactly two characters in length, or if $value_1$ is greater than 59, let *most significant units* be *hours*.

8. If *position* is beyond the end of *input* or if the character at *position* is not a U+003A COLON character (:), then return an error and abort these steps. Otherwise, move *position* forwards one character.

9. collect a sequence of code points that are ASCII digits, and let *string* be the collected substring.

10. If *string* is not exactly two characters in length, return an error and abort these steps.

11. Interpret *string* as a base-ten integer. Let $value_2$ be that integer.

12. If *most significant units* is *hours*, or if *position* is not beyond the end of *input* and the character at *position* is a U+003A COLON character (:), run these substeps:

    1. If *position* is beyond the end of *input* or if the character at *position* is not a U+003A COLON character (:), then return an error and abort these steps. Otherwise, move *position* forwards one character.

    2. collect a sequence of code points that are ASCII digits, and let *string* be the collected substring.

    3. If *string* is not exactly two characters in length, return an error and abort these steps.

    4. Interpret *string* as a base-ten integer. Let $value_3$ be that integer.

    Otherwise (if *most significant units* is not *hours*, and either *position* is beyond the end of *input*, or the character at *position* is not a U+003A COLON character (:)), let $value_3$ have the value of $value_2$, then $value_2$ have the value of $value_1$, then let $value_1$ equal zero.

13. If *position* is beyond the end of *input* or if the character at *position* is not a U+002E FULL STOP character (.), then return an error and abort these steps. Otherwise, move *position* forwards one character.

14. collect a sequence of code points that are ASCII digits, and let *string* be the collected substring.

15. If *string* is not exactly three characters in length, return an error and abort these steps.

16. Interpret *string* as a base-ten integer. Let $value_4$ be that integer.

17. If $value_2$ is greater than 59 or if $value_3$ is greater than 59, return an error and abort these steps.

18. Let *result* be $value_1 \times 60 \times 60 + value_2 \times 60 + value_3 + value_4 / 1000$.

19. Return *result*.

§ 6.4. WebVTT cue text parsing rules

A ***WebVTT Node Object*** is a conceptual construct used to represent components of cue text so that its processing can be described without reference to the underlying syntax.

There are two broad classes of WebVTT Node Objects: WebVTT Internal Node Objects and WebVTT Leaf Node Objects.

WebVTT Internal Node Objects are those that can contain further WebVTT Node Objects. They are conceptually similar to elements in HTML or the DOM. WebVTT Internal Node Objects have an ordered list of child WebVTT Node Objects. The WebVTT Internal Node Object is said to be the *parent* of the children. Cycles do not occur; the parent-child relationships so constructed form a tree structure. WebVTT Internal Node Objects also have an ordered list of class names, known as their ***applicable classes***, and a language, known as their ***applicable language***, which is to be interpreted as a BCP 47 language tag. [BCP47]

> User agents will add a language tag as the applicable language even if it is not a valid or not even well-formed language tag. [BCP47]

There are several concrete classes of WebVTT Internal Node Objects:

Lists of WebVTT Node Objects
: These are used as root nodes for trees of WebVTT Node Objects.

WebVTT Class Objects
: These represent spans of text (a WebVTT cue class span) in cue text, and are used to annotate parts of the cue with applicable classes without implying further meaning (such as italics or bold).

WebVTT Italic Objects
: These represent spans of italic text (a WebVTT cue italics span) in WebVTT caption or subtitle cue text.

WebVTT Bold Objects
: These represent spans of bold text (a WebVTT cue bold span) in WebVTT caption or subtitle cue text.

WebVTT Underline Objects
: These represent spans of underline text (a WebVTT cue underline span) in WebVTT caption or subtitle cue text.

WebVTT Ruby Objects
: These represent spans of ruby (a WebVTT cue ruby span) in WebVTT caption or subtitle cue

text.

WebVTT Ruby Text Objects

These represent spans of ruby text (a WebVTT cue ruby text span) in WebVTT caption or subtitle cue text.

WebVTT Voice Objects

These represent spans of text associated with a specific voice (a WebVTT cue voice span) in WebVTT caption or subtitle cue text. A WebVTT Voice Object has a value, which is the name of the voice.

WebVTT Language Objects

These represent spans of text (a WebVTT cue language span) in WebVTT caption or subtitle cue text, and are used to annotate parts of the cue where the applicable language might be different than the surrounding text's, without implying further meaning (such as italics or bold).

WebVTT Leaf Node Objects are those that contain data, such as text, and cannot contain child WebVTT Node Objects.

There are two concrete classes of WebVTT Leaf Node Objects:

WebVTT Text Objects

A fragment of text. A WebVTT Text Object has a value, which is the text it represents.

WebVTT Timestamp Objects

A timestamp. A WebVTT Timestamp Object has a value, in seconds and fractions of a second, which is the time represented by the timestamp.

The ***WebVTT cue text parsing rules*** consist of the following algorithm. The input is a string *input* supposedly containing WebVTT caption or subtitle cue text, and optionally a fallback language *language*. This algorithm returns a list of WebVTT Node Objects.

1. Let *input* be the string being parsed.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Let *result* be a list of WebVTT Node Objects, initially empty.
4. Let *current* be the WebVTT Internal Node Object *result*.
5. Let *language stack* be a stack of language tags, initially empty.
6. If *language* is set, set *result*'s applicable language to *language*, and push *language* onto the *language stack*.

7. *Loop*: If *position* is past the end of *input*, return *result* and abort these steps.

8. Let *token* be the result of invoking the WebVTT cue text tokenizer.

9. Run the appropriate steps given the type of *token*:

If *token* is a string

1. Create a WebVTT Text Object whose value is the value of the string token *token*.

2. Append the newly created WebVTT Text Object to *current*.

If *token* is a start tag

How the start tag token *token* is processed depends on its tag name, as follows:

If the tag name is `"c"`

Attach a WebVTT Class Object.

If the tag name is `"i"`

Attach a WebVTT Italic Object.

If the tag name is `"b"`

Attach a WebVTT Bold Object.

If the tag name is `"u"`

Attach a WebVTT Underline Object.

If the tag name is `"ruby"`

Attach a WebVTT Ruby Object.

If the tag name is `"rt"`

If *current* is a WebVTT Ruby Object, then attach a WebVTT Ruby Text Object.

If the tag name is `"v"`

Attach a WebVTT Voice Object, and set its value to the token's annotation string, or the empty string if there is no annotation string.

If the tag name is `"lang"`

Push the value of the token's annotation string, or the empty string if there is no annotation string, onto the *language stack*; then attach a WebVTT Language Object.

Otherwise

Ignore the token.

When the steps above say to ***attach a WebVTT Internal Node Object*** of a particular concrete class, the user agent must run the following steps:

1. Create a new WebVTT Internal Node Object of the specified concrete class.
2. Set the new object's list of applicable classes to the list of classes in the token, excluding any classes that are the empty string.
3. Set the new object's applicable language to the top entry on the *language stack*, if the stack is not empty.
4. Append the newly created node object to *current*.
5. Let *current* be the newly created node object.

If *token* is an end tag

If any of the following conditions is true, then let *current* be the parent node of *current*.

- The tag name of the end tag token *token* is `"c"` and *current* is a WebVTT Class Object.
- The tag name of the end tag token *token* is `"i"` and *current* is a WebVTT Italic Object.
- The tag name of the end tag token *token* is `"b"` and *current* is a WebVTT Bold Object.
- The tag name of the end tag token *token* is `"u"` and *current* is a WebVTT Underline Object.
- The tag name of the end tag token *token* is `"ruby"` and *current* is a WebVTT Ruby Object.
- The tag name of the end tag token *token* is `"rt"` and *current* is a WebVTT Ruby Text Object.
- The tag name of the end tag token *token* is `"v"` and *current* is a WebVTT Voice Object.

Otherwise, if the tag name of the end tag token *token* is `"lang"` and *current* is a WebVTT Language Object, then let *current* be the parent node of *current*, and pop the top value from the *language stack*.

Otherwise, if the tag name of the end tag token *token* is `"ruby"` and *current* is a WebVTT Ruby Text Object, then let *current* be the parent node of the parent node of *current*.

Otherwise, ignore the token.

If *token* is a timestamp tag

1. Let *input* be the tag value.
2. Let *position* be a pointer into *input*, initially pointing at the start of the string.
3. Collect a WebVTT timestamp.

4. If that algorithm does not fail, and if *position* now points at the end of *input* (i.e. there are no trailing characters after the timestamp), then create a WebVTT Timestamp Object whose value is the collected time, then append it to *current*.

   Otherwise, ignore the token.

10. Jump to the step labeled *loop*.

The ***WebVTT cue text tokenizer*** is as follows. It emits a token, which is either a string (whose value is a sequence of characters), a start tag (with a tag name, a list of classes, and optionally an annotation), an end tag (with a tag name), or a timestamp tag (with a tag value).

1. Let *input* and *position* be the same variables as those of the same name in the algorithm that invoked these steps.
2. Let *tokenizer state* be WebVTT data state.
3. Let *result* be the empty string.
4. Let *classes* be an empty list.
5. *Loop*: If *position* is past the end of *input*, let *c* be an end-of-file marker. Otherwise, let *c* be the character in *input* pointed to by *position*.

   An end-of-file marker is not a Unicode character, it is used to end the tokenizer.

6. Jump to the state given by *tokenizer state*:

   WebVTT data state

   Jump to the entry that matches the value of *c*:

   U+0026 AMPERSAND (&)

   Set *tokenizer state* to the HTML character reference in data state, and jump to the step labeled *next*.

   U+003C LESS-THAN SIGN (<)

   If *result* is the empty string, then set *tokenizer state* to the WebVTT tag state and jump to the step labeled *next*.

   Otherwise, return a string token whose value is *result* and abort these steps.

   End-of-file marker

   Return a string token whose value is *result* and abort these steps.

Anything else

Append *c* to *result* and jump to the step labeled *next*.

HTML character reference in data state

Attempt to consume an HTML character reference, with no additional allowed character.

If nothing is returned, append a U+0026 AMPERSAND character (&) to *result*.

Otherwise, append the data of the character tokens that were returned to *result*.

Then, in any case, set *tokenizer state* to the WebVTT data state, and jump to the step labeled *next*.

WebVTT tag state

Jump to the entry that matches the value of *c*:

U+0009 CHARACTER TABULATION (tab) character
U+000A LINE FEED (LF) character
U+000C FORM FEED (FF) character
U+0020 SPACE character

Set *tokenizer state* to the WebVTT start tag annotation state, and jump to the step labeled *next*.

U+002E FULL STOP character (.)

Set *tokenizer state* to the WebVTT start tag class state, and jump to the step labeled *next*.

U+002F SOLIDUS character (/)

Set *tokenizer state* to the WebVTT end tag state, and jump to the step labeled *next*.

ASCII digits

Set *result* to *c*, set *tokenizer state* to the WebVTT timestamp tag state, and jump to the step labeled *next*.

U+003E GREATER-THAN SIGN character (>)

Advance *position* to the next character in *input*, then jump to the next "end-of-file marker" entry below.

End-of-file marker

Return a start tag whose tag name is the empty string, with no classes and no annotation, and abort these steps.

Anything else

Set *result* to *c*, set *tokenizer state* to the WebVTT start tag state, and jump to the step labeled *next*.

WebVTT start tag state

Jump to the entry that matches the value of *c*:

U+0009 CHARACTER TABULATION (tab) character
U+000C FORM FEED (FF) character
U+0020 SPACE character
Set *tokenizer state* to the WebVTT start tag annotation state, and jump to the step labeled *next*.

U+000A LINE FEED (LF) character
Set *buffer* to *c*, set *tokenizer state* to the WebVTT start tag annotation state, and jump to the step labeled *next*.

U+002E FULL STOP character (.)
Set *tokenizer state* to the WebVTT start tag class state, and jump to the step labeled *next*.

U+003E GREATER-THAN SIGN character (>)
Advance *position* to the next character in *input*, then jump to the next "end-of-file marker" entry below.

End-of-file marker
Return a start tag whose tag name is *result*, with no classes and no annotation, and abort these steps.

Anything else
Append *c* to *result* and jump to the step labeled *next*.

WebVTT start tag class state
Jump to the entry that matches the value of *c*:

U+0009 CHARACTER TABULATION (tab) character
U+000C FORM FEED (FF) character
U+0020 SPACE character
Append to *classes* an entry whose value is *buffer*, set *buffer* to the empty string, set *tokenizer state* to the WebVTT start tag annotation state, and jump to the step labeled *next*.

U+000A LINE FEED (LF) character
Append to *classes* an entry whose value is *buffer*, set *buffer* to *c*, set *tokenizer state* to the WebVTT start tag annotation state, and jump to the step labeled *next*.

U+002E FULL STOP character (.)
Append to *classes* an entry whose value is *buffer*, set *buffer* to the empty string, and jump to the step labeled *next*.

U+003E GREATER-THAN SIGN character (>)
Advance *position* to the next character in *input*, then jump to the next "end-of-file

marker" entry below.

End-of-file marker

Append to *classes* an entry whose value is *buffer*, then return a start tag whose tag name is *result*, with the classes given in *classes* but no annotation, and abort these steps.

Anything else

Append *c* to *buffer* and jump to the step labeled *next*.

WebVTT start tag annotation state

Jump to the entry that matches the value of *c*:

U+0026 AMPERSAND (&)

Set *tokenizer state* to the HTML character reference in annotation state, and jump to the step labeled *next*.

U+003E GREATER-THAN SIGN character (>)

Advance *position* to the next character in *input*, then jump to the next "end-of-file marker" entry below.

End-of-file marker

Remove any leading or trailing ASCII whitespace characters from *buffer*, and replace any sequence of one or more consecutive ASCII whitespace characters in *buffer* with a single U+0020 SPACE character; then, return a start tag whose tag name is *result*, with the classes given in *classes*, and with *buffer* as the annotation, and abort these steps.

Anything else

Append *c* to *buffer* and jump to the step labeled *next*.

HTML character reference in annotation state

Attempt to consume an HTML character reference, with the additional allowed character being U+003E GREATER-THAN SIGN (>).

If nothing is returned, append a U+0026 AMPERSAND character (&) to *buffer*.

Otherwise, append the data of the character tokens that were returned to *buffer*.

Then, in any case, set *tokenizer state* to the WebVTT start tag annotation state, and jump to the step labeled *next*.

WebVTT end tag state

Jump to the entry that matches the value of *c*:

U+003E GREATER-THAN SIGN character (>)

Advance *position* to the next character in *input*, then jump to the next "end-of-file

marker" entry below.

End-of-file marker
Return an end tag whose tag name is *result* and abort these steps.

Anything else
Append *c* to *result* and jump to the step labeled *next*.

WebVTT timestamp tag state
Jump to the entry that matches the value of *c*:

U+003E GREATER-THAN SIGN character (>)
Advance *position* to the next character in *input*, then jump to the next "end-of-file marker" entry below.

End-of-file marker
Return a timestamp tag whose tag name is *result* and abort these steps.

Anything else
Append *c* to *result* and jump to the step labeled *next*.

7. *Next*: Advance *position* to the next character in *input*.

8. Jump to the step labeled *loop*.

When the algorithm above says to attempt to ***consume an HTML character reference***, it means to attempt to consume a character reference as defined in HTML. [HTML51]

When the HTML specification says to consume a character, in this context, it means to advance *position* to the next character in *input*. When it says to unconsume a character, it means to move *position* back to the previous character in *input*. "EOF" is equivalent to the end-of-file marker in this specification. Finally, this context is *not* "as part of an attribute" (when it comes to handling a missing semicolon).

§ 6.5. *WebVTT cue text DOM construction rules*

For the purpose of retrieving a WebVTT cue's content via the `getCueAsHTML()` method of the `VTTCue` interface, it needs to be parsed to a `DocumentFragment`. This section describes how.

To convert a list of WebVTT Node Objects to a DOM tree for `Document` *owner*, user agents must create a tree of DOM nodes that is isomorphous to the tree of WebVTT Node Objects, with the following

mapping of WebVTT Node Objects to DOM nodes:

WebVTT Node Object	DOM node
List of WebVTT Node Objects	`DocumentFragment` node.
WebVTT Region Object	`DocumentFragment` node.
WebVTT Class Object	HTML `<span>` element.
WebVTT Italic Object	HTML `<i>` element.
WebVTT Bold Object	HTML `<b>` element.
WebVTT Underline Object	HTML `<u>` element.
WebVTT Ruby Object	HTML `<ruby>` element.
WebVTT Ruby Text Object	HTML `<rt>` element.
WebVTT Voice Object	HTML `<span>` element with a `title` attribute set to the WebVTT Voice Object's value.
WebVTT Language Object	HTML `<span>` element with a `lang` attribute set to the WebVTT Language Object's applicable language.
WebVTT Text Object	`Text` node whose `data` is the value of the WebVTT Text Object.
WebVTT Timestamp	`ProcessingInstruction` node whose `target` is "`timestamp`" and whose `data` is a WebVTT timestamp representing the value of the WebVTT

HTML elements created as part of the mapping described above must have their `namespaceURI` set to the HTML namespace, use the appropriate IDL interface as defined in the HTML specification, and, if the corresponding WebVTT Internal Node Object has any applicable classes, must have a `class` attribute set to the string obtained by concatenating all those classes, each separated from the next by a single U+0020 SPACE character.

The `ownerDocument` attribute of all nodes in the DOM tree must be set to the given document *owner*.

All characteristics of the DOM nodes that are not described above or dependent on characteristics defined above must be left at their initial values.

§ 6.6. WebVTT rules for extracting the chapter title

The ***WebVTT rules for extracting the chapter title*** for a WebVTT cue *cue* are as follows:

1. Let *nodes* be the list of WebVTT Node Objects obtained by applying the WebVTT cue text parsing rules to the *cue*'s cue text.
2. Return the concatenation of the values of each WebVTT Text Object in *nodes*, in a pre-order, depth-first traversal, excluding WebVTT Ruby Text Objects and their descendants.

§ 7. Rendering

This section describes in some detail how to visually render WebVTT caption or subtitle cues in a user agent. The processing model is quite tightly linked to media elements in HTML, where CSS is available. User agents that do not support CSS are expected to render plain text only, without styling and positioning features. User agents that do not support a full HTML CSS engine are expected to render an equivalent visual representation to what a user agent with a full CSS engine would render.

§ 7.1. Processing model

The ***rules for updating the display of WebVTT text tracks*** render the text tracks of a media element (specifically, a `<video>` element), or of another playback mechanism, by applying the steps below. All the text tracks that use these rules for a given media element, or other playback mechanism, are ren-

dered together, to avoid overlapping subtitles from multiple tracks. A fallback language *language* may be set when calling this algorithm.

> In HTML, audio elements don't have a visual rendering area and therefore, this algorithm will abort for audio elements. When authors do create WebVTT captions or subtitles for audio resources, they need to publish them in a video element for rendering by the user agent.

The output of the steps below is a set of CSS boxes that covers the rendering area of the media element or other playback mechanism, which user agents are expected to render in a manner suiting the user.

The rules are as follows:

1. If the media element is an `<audio>` element, or is another playback mechanism with no rendering area, abort these steps.
2. Let *video* be the media element or other playback mechanism.
3. Let *output* be an empty list of absolutely positioned CSS block boxes.
4. If the user agent is exposing a user interface for *video*, add to *output* one or more completely transparent positioned CSS block boxes that cover the same region as the user interface.
5. If the last time these rules were run, the user agent was not exposing a user interface for *video*, but now it is, optionally let *reset* be true. Otherwise, let *reset* be false.
6. Let *tracks* be the subset of *video*'s list of text tracks that have as their rules for updating the text track rendering these rules for updating the display of WebVTT text tracks, and whose text track mode is showing.
7. Let *cues* be an empty list of text track cues.
8. For each track *track* in *tracks*, append to *cues* all the cues from *track*'s list of cues that have their text track cue active flag set.
9. Let *regions* be an empty list of WebVTT regions.
10. For each track *track* in *tracks*, append to *regions* all the regions with an identifier from *track*'s list of regions.
11. If *reset* is false, then, for each WebVTT region *region* in *regions* let *regionNode* be a WebVTT region object.

12. Apply the following steps for each *regionNode*:

    1. Prepare some variables for the application of CSS properties to *regionNode* as follows:

        - Let *regionWidth* be the WebVTT region width. Let *width* be '*regionWidth* vw' ('vw' is a CSS unit). [CSS-VALUES]
        - Let *lineHeight* be '6vh' ('vh' is a CSS unit) [CSS-VALUES] and *regionHeight* be the WebVTT region lines. Let *lines* be *lineHeight* multiplied by *regionHeight*.
        - Let *viewportAnchorX* be the x dimension of the WebVTT region anchor and *regionAnchorX* be the x dimension of the WebVTT region anchor. Let *leftOffset* be *regionAnchorX* multiplied by *width* divided by 100.0. Let *left* be *leftOffset* subtracted from '*viewportAnchorX* vw'.
        - Let *viewportAnchorY* be the y dimension of the WebVTT region anchor and *regionAnchorY* be the y dimension of the WebVTT region anchor. Let *topOffset* be *regionAnchorY* multiplied by *lines* divided by 100.0. Let *top* be *topOffset* subtracted from '*viewportAnchorY* vh'.

    2. Apply the terms of the CSS specifications to *regionNode* within the following constraints, thus obtaining a CSS box *box* positioned relative to an initial containing block:

        1. No style sheets are associated with *regionNode*. (The regionNodes are subsequently restyled using style sheets after their boxes are generated, as described below.)
        2. Properties on *regionNode* have their values set as defined in the next section. (That section uses some of the variables whose values were calculated earlier in this algorithm.)
        3. The video viewport (and initial containing block) is video's rendering area.

    3. Add the CSS box *box* to *output*.

13. If *reset* is false, then, for each WebVTT cue *cue* in *cues*: if *cue*'s text track cue display state has a set of CSS boxes, then:

    - If *cue*'s WebVTT cue region is not null, add those boxes to that region's *box* and remove *cue* from *cues*.
    - Otherwise, add those boxes to *output* and remove *cue* from *cues*.

14. For each WebVTT cue *cue* in *cues* that has not yet had corresponding CSS boxes added to *output*,

in text track cue order, run the following substeps:

1. Let *nodes* be the list of WebVTT Node Objects obtained by applying the WebVTT cue text parsing rules, with the fallback language *language* if provided, to the *cue*'s cue text.
2. If *cue*'s WebVTT cue region is null, run the following substeps:
    1. Apply WebVTT cue settings to obtain CSS boxes *boxes* from *nodes*.
    2. Let *cue*'s text track cue display state have the CSS boxes in *boxes*.
    3. Add the CSS boxes in *boxes* to *output*.
3. Otherwise, run the following substeps:
    1. Let *region* be *cue*'s WebVTT cue region.
    2. If *region*'s WebVTT region scroll setting is up and *region* already has one child, set *region*'s 'transition-property' to 'top' and 'transition-duration' to '0.433s'.
    3. Let *offset* be *cue*'s computed position multiplied by *region*'s WebVTT region width and divided by 100 (i.e. interpret it as a percentage of the region width).
    4. Adjust *offset* using *cue*'s computed position alignment as follows:

        ↪ **If the computed position alignment is center alignment**
        Subtract half of *region*'s WebVTT region width from *offset*.

        ↪ **If the computed position alignment is line-right alignment**
        Subtract *region*'s WebVTT region width from *offset*.

    5. Let *left* be '*offset* %'. [CSS-VALUES]
    6. Obtain a set of CSS boxes *boxes* positioned relative to an initial containing block.
    7. If there are no line boxes in *boxes*, skip the remainder of these substeps for *cue*. The cue is ignored.
    8. Let *cue*'s text track cue display state have the CSS boxes in *boxes*.
    9. Add the CSS boxes in *boxes* to *region*.
    10. If the CSS boxes *boxes* together have a height less than the height of the *region* box, let *diff* be the absolute difference between the two height values. Increase *top* by *diff* and re-apply it to *regionNode*.

15. Return *output*.

User agents may allow the user to override the above algorithm's positioning of cues, e.g. by dragging them to another location on the `<video>`, or even off the `<video>` entirely.

§ 7.2. Processing cue settings

When the processing algorithm above requires that the user agent ***apply WebVTT cue settings*** to obtain CSS boxes from a list of WebVTT Node Objects *nodes*, the user agent must run the following algorithm.

1. If the WebVTT cue writing direction is horizontal, then let *writing-mode* be "horizontal-tb". Otherwise, if the WebVTT cue writing direction is vertical growing left, then let *writing-mode* be "vertical-rl". Otherwise, the WebVTT cue writing direction is vertical growing right; let *writing-mode* be "vertical-lr".

2. Determine the value of *maximum size* for *cue* as per the appropriate rules from the following list:

   ↪ **If the computed position alignment is line-left**
   Let *maximum size* be the computed position subtracted from 100.

   ↪ **If the computed position alignment is line-right**
   Let *maximum size* be the computed position.

   ↪ **If the computed position alignment is center, and the computed position is less than or equal to 50**
   Let *maximum size* be the computed position multiplied by two.

   ↪ **If the computed position alignment is center, and the computed position is greater than 50**
   Let *maximum size* be the result of subtracting computed position from 100 and then multiplying the result by two.

3. If the WebVTT cue size is less than *maximum size*, then let *size* be WebVTT cue size. Otherwise, let *size* be *maximum size*.

4. If the WebVTT cue writing direction is horizontal, then let *width* be '*size* vw' and *height* be 'auto'. Otherwise, let *width* be 'auto' and *height* be '*size* vh'. (These are CSS values used by the next section to set CSS properties for the rendering; 'vw' and 'vh' are CSS units.) [CSS-VALUES]

5. Determine the value of *x-position* or *y-position* for *cue* as per the appropriate rules from the fol-

lowing list:

↪ **If the WebVTT cue writing direction is horizontal**

↪ **If the computed position alignment is line-left alignment**
Let *x-position* be the computed position.

↪ **If the computed position alignment is center alignment**
Let *x-position* be the computed position minus half of *size*.

↪ **If the computed position alignment is line-right alignment**
Let *x-position* be the computed position minus *size*.

↪ **If the WebVTT cue writing direction is vertical growing left or vertical growing right**

↪ **If the computed position alignment is line-left alignment**
Let *y-position* be the computed position.

↪ **If the computed position alignment is center alignment**
Let *y-position* be the computed position minus half of *size*.

↪ **If the computed position alignment is line-right alignment**
Let *y-position* be the computed position minus *size*.

6. Determine the value of whichever of *x-position* or *y-position* is not yet calculated for *cue* as per the appropriate rules from the following list:

↪ **If the WebVTT cue snap-to-lines flag is false**

↪ **If the WebVTT cue writing direction is horizontal**
Let *y-position* be the computed line.

↪ **If the WebVTT cue writing direction is vertical growing left or vertical growing right**
Let *x-position* be the computed line.

↪ **If the WebVTT cue snap-to-lines flag is true**

↪ **If the WebVTT cue writing direction is horizontal**
Let *y-position* be 0.

↪ **If the WebVTT cue writing direction is vertical growing left or vertical growing right**
Let *x-position* be 0.

> These are not final positions, they are merely temporary positions used to calculate box dimensions below.

7. Let *left* be '*x-position* vw' and *top* be '*y-position* vh'. (These are CSS values used by the next section to set CSS properties for the rendering; 'vw' and 'vh' are CSS units.) [CSS-VALUES]

8. Obtain a set of CSS boxes *boxes* positioned relative to an initial containing block.

9. If there are no line boxes in *boxes*, skip the remainder of these substeps for *cue*. The cue is ignored.

10. Adjust the positions of *boxes* according to the appropriate steps from the following list:

↪ **If *cue*'s WebVTT cue snap-to-lines flag is true**

Many of the steps in this algorithm vary according to the WebVTT cue writing direction. Steps labeled "**Horizontal**" must be followed only when the WebVTT cue writing direction is horizontal, steps labeled "**Vertical**" must be followed when the WebVTT cue writing direction is either vertical growing left or vertical growing right, steps labeled "**Vertical Growing Left**" must be followed only when the WebVTT cue writing direction is vertical growing left, and steps labeled "**Vertical Growing Right**" must be followed only when the WebVTT cue writing direction is vertical growing right.

1. **Horizontal**: Let *full dimension* be the height of *video*'s rendering area.

   Vertical: Let *full dimension* be the width of *video*'s rendering area.

2. **Horizontal**: Let *step* be the height of the first line box in *boxes*.

   Vertical: Let *step* be the width of the first line box in *boxes*.

3. If *step* is zero, then jump to the step labeled *done positioning* below.

4. Let *line* be *cue*'s computed line.

5. Round *line* to an integer by adding 0.5 and then flooring it.

6. **Vertical Growing Left**: Add one to *line* then negate it.

7. Let *position* be the result of multiplying *step* and *line*.

8. **Vertical Growing Left**: Decrease *position* by the width of the bounding box of the boxes in *boxes*, then increase *position* by *step*.

9. If *line* is less than zero then increase *position* by *max dimension*, and negate *step*.

10. **Horizontal**: Move all the boxes in *boxes* down by the distance given by *position*.

    Vertical: Move all the boxes in *boxes* right by the distance given by *position*.

11. Remember the position of all the boxes in *boxes* as their *specified position*.
12. Let *title area* be a box that covers all of the *video*'s rendering area.
13. *Step loop*: If none of the boxes in *boxes* would overlap any of the boxes in *output*, and all of the boxes in *boxes* are entirely within the *title area* box, then jump to the step labeled *done positioning* below.
14. **Horizontal**: If *step* is negative and the top of the first line box in *boxes* is now above the top of the *title area*, or if *step* is positive and the bottom of the first line box in *boxes* is now below the bottom of the *title area*, jump to the step labeled *switch direction*.

    Vertical: If *step* is negative and the left edge of the first line box in *boxes* is now to the left of the left edge of the *title area*, or if *step* is positive and the right edge of the first line box in *boxes* is now to the right of the right edge of the *title area*, jump to the step labeled *switch direction*.
15. **Horizontal**: Move all the boxes in *boxes* down by the distance given by *step*. (If *step* is negative, then this will actually result in an upwards movement of the boxes in absolute terms.)

    Vertical: Move all the boxes in *boxes* right by the distance given by *step*. (If *step* is negative, then this will actually result in a leftwards movement of the boxes in absolute terms.)
16. Jump back to the step labeled *step loop*.
17. *Switch direction*: If *switched* is true, then remove all the boxes in *boxes*, and jump to the step labeled *done positioning* below.
18. Otherwise, move all the boxes in *boxes* back to their *specified position* as determined in the earlier step.
19. Negate *step*.
20. Set *switched* to true.
21. Jump back to the step labeled *step loop*.

↪ **If *cue*'s WebVTT cue snap-to-lines flag is false**

1. Let *bounding box* be the bounding box of the boxes in *boxes*.

2. Run the appropriate steps from the following list:

   ↪ **If the WebVTT cue writing direction is horizontal**

      ↪ **If the WebVTT cue line alignment is center alignment**
      Move all the boxes in *boxes* up by half of the height of *bounding box*.

      ↪ **If the WebVTT cue line alignment is end alignment**
      Move all the boxes in *boxes* up by the height of *bounding box*.

   ↪ **If the WebVTT cue writing direction is vertical growing left or vertical growing right**

      ↪ **If the WebVTT cue line alignment is center alignment**
      Move all the boxes in *boxes* left by half of the width of *bounding box*.

      ↪ **If the WebVTT cue line alignment is end alignment**
      Move all the boxes in *boxes* left by the width of *bounding box*.

3. If none of the boxes in *boxes* would overlap any of the boxes in *output*, and all the boxes in *boxes* are within the *video*'s rendering area, then jump to the step labeled *done positioning* below.

4. If there is a position to which the boxes in *boxes* can be moved while maintaining the relative positions of the boxes in *boxes* to each other such that none of the boxes in *boxes* would overlap any of the boxes in *output*, and all the boxes in *boxes* would be within the *video*'s rendering area, then move the boxes in *boxes* to the closest such position to their current position, and then jump to the step labeled *done positioning* below. If there are multiple such positions that are equidistant from their current position, use the highest one amongst them; if there are several at that height, then use the leftmost one amongst them.

5. Otherwise, jump to the step labeled *done positioning* below. (The boxes will unfortunately overlap.)

11. *Done positioning*: Return *boxes*.

§ 7.3. Obtaining CSS boxes

When the processing algorithm above requires that the user agent ***obtain a set of CSS boxes*** *boxes*,

then apply the terms of the CSS specifications to *nodes* within the following constraints: [CSS22]

- The *document tree* is the tree of WebVTT Node Objects rooted at *nodes*.
- For the purpose of selectors in STYLE blocks of a WebVTT file, the style sheet must apply to a hypothetical document that contains a single empty element with no explicit name, no namespace, no attributes, no classes, no IDs, and unknown primary language, that acts like the media element for the text tracks that were sourced from the given WebVTT file. The selectors must not match other text tracks for the same media element. In this hypothetical document, the element must not match any selector that would match the element itself.

  > This element exists only to be the originating element for the '::cue', '::cue()', '::cue-region' and '::cue-region()' pseudo-elements.

- For the purpose of determining the cascade of the declarations in STYLE blocks of a WebVTT file, the relative order of appearance of the style sheets must be the same order as they were added to the collection, and the order of appearance of the collection must be after any style sheets that apply to the associated `<video>` element's document.

EXAMPLE 21

For example, given the following (invalid) HTML document:

```
<!doctype html>
<title>Invalid cascade example</title>
<video controls autoplay src="video.webm">
 <track default src="track.vtt">
</video>
<style>
 ::cue { color:red }
</style>
```

...and the "track.vtt" file contains:

```
WEBVTT

STYLE
::cue { color:lime }

00:00:00.000 --> 00:00:25.000
Red or green?
```

The 'color:lime' declaration would win, because it is last in the cascade, even though the `<style>` element is after the `<video>` element in the document order.

- For the purpose of resolving URLs in STYLE blocks of a WebVTT file, or any URLs in resources referenced from STYLE blocks of a WebVTT file, if the URL's scheme is not "`data`", then the user agent must act as if the URL failed to resolve.

Supporting external resources with '@import' or 'background-image' would be a new ability for media elements and `<track>` elements to issue network requests as the user watches the video, which could be a privacy issue.

- For the purposes of processing by the CSS specification, WebVTT Internal Node Objects are equivalent to elements with the same contents.
- For the purposes of processing by the CSS specification, WebVTT Text Objects are equivalent to `Text` nodes.
- No style sheets are associated with *nodes*. (The nodes are subsequently restyled using style sheets after their boxes are generated, as described below.)

- The children of the *nodes* must be wrapped in an anonymous box whose 'display' property has the value 'inline'. This is the ***WebVTT cue background box***.
- Runs of children of WebVTT Ruby Objects that are not WebVTT Ruby Text Objects must be wrapped in anonymous boxes whose 'display' property has the value 'ruby-base'. [CSS3-RUBY]
- Properties on WebVTT Node Objects have their values set as defined in the next section. (That section uses some of the variables whose values were calculated earlier in this algorithm.)
- Text runs must be wrapped according to the CSS line-wrapping rules.
- The video viewport (and initial containing block) is *video*'s rendering area.

Let *boxes* be the boxes generated as descendants of the initial containing block, along with their positions.

§ 7.4. Applying CSS properties to WebVTT Node Objects

When following the rules for updating the display of WebVTT text tracks, user agents must set properties of WebVTT Node Objects at the CSS user agent cascade layer as defined in this section. [CSS22]

Initialize the (root) list of WebVTT Node Objects with the following CSS settings:

- the 'position' property must be set to 'absolute'
- the 'unicode-bidi' property must be set to 'plaintext'
- the 'writing-mode' property must be set to *writing-mode*
- the 'top' property must be set to *top*
- the 'left' property must be set to *left*
- the 'width' property must be set to *width*
- the 'height' property must be set to *height*
- the 'overflow-wrap' property must be set to 'break-word'
- the 'text-wrap' property must be set to 'balance' [CSS-TEXT-4]

The variables *writing-mode*, *top*, *left*, *width*, and *height* are the values with those names determined by the rules for updating the display of WebVTT text tracks for the WebVTT cue from whose text the list of WebVTT Node Objects was constructed.

The 'text-align' property on the (root) list of WebVTT Node Objects must be set to the value in the second cell of the row of the table below whose first cell is the value of the corresponding cue's WebVTT cue text alignment:

WebVTT cue text alignment	'text-align' value
Start alignment	'start'
Center alignment	'center'
End alignment	'end'
Left alignment	'left'
Right alignment	'right'

The 'font' shorthand property on the (root) list of WebVTT Node Objects must be set to '5vh sans-serif'. [CSS-VALUES]

The 'color' property on the (root) list of WebVTT Node Objects must be set to 'rgba(255,255,255,1)'. [CSS3-COLOR]

The 'background' shorthand property on the WebVTT cue background box and on WebVTT Ruby Text Objects must be set to 'rgba(0,0,0,0.8)'. [CSS3-COLOR]

The 'white-space' property on the (root) list of WebVTT Node Objects must be set to 'pre-line'. [CSS22]

The 'font-style' property on WebVTT Italic Objects must be set to 'italic'.

The 'font-weight' property on WebVTT Bold Objects must be set to 'bold'.

The 'text-decoration' property on WebVTT Underline Objects must be set to 'underline'.

The 'display' property on WebVTT Ruby Objects must be set to 'ruby'. [CSS3-RUBY]

The 'display' property on WebVTT Ruby Text Objects must be set to 'ruby-text'. [CSS3-RUBY]

Every WebVTT region object is initialized with the following CSS settings:

- the 'position' property must be set to 'absolute'
- the 'writing-mode' property must be set to 'horizontal-tb'
- the 'background' shorthand property must be set to 'rgba(0,0,0,0.8)'
- the 'overflow-wrap' property must be set to 'break-word'
- the 'font' shorthand property must be set to '5vh sans-serif'

- the ‘color’ property must be set to ‘rgba(255,255,255,1)’
- the ‘overflow’ property must be set to ‘hidden’
- the ‘width’ property must be set to *width*
- the ‘min-height’ property must be set to ‘0px’
- the ‘max-height’ property must be set to *height*
- the ‘left’ property must be set to *left*
- the ‘top’ property must be set to *top*
- the ‘display’ property must be set to ‘inline-flex’
- the ‘flex-flow’ property must be set to ‘column’
- the ‘justify-content’ property must be set to ‘flex-end’

The variables *width*, *height*, *top*, and *left* are the values with those names determined by the rules for updating the display of WebVTT text tracks for the WebVTT region from which the WebVTT region object was constructed.

The children of every WebVTT region object are further initialized with these CSS settings:

- the ‘position’ property must be set to ‘relative’
- the ‘unicode-bidi’ property must be set to ‘plaintext’
- the ‘width’ property must be set to ‘auto’
- the ‘height’ property must be set to *height*
- the ‘left’ property must be set to *left*
- the ‘text-align’ property must be set as described for the root List of WebVTT Node Objects not part of a region

All other non-inherited properties must be set to their initial values; inherited properties on the root list of WebVTT Node Objects must inherit their values from the media element for which the WebVTT cue is being rendered, if any. If there is no media element (i.e. if the text track is being rendered for another media playback mechanism), then inherited properties on the root list of WebVTT Node Objects and the WebVTT region objects must take their initial values.

If there are style sheets that apply to the media element or other playback mechanism, then they must be interpreted as defined in the next section.

§ 8. CSS extensions

> This section specifies some CSS pseudo-elements and pseudo-classes and how they apply to WebVTT. This section does not apply to user agents that do not support CSS.

§ 8.1. Introduction

This section is non-normative.

The '::cue' pseudo-element represents a cue.

The '::cue(*selector*)' pseudo-element represents a cue or element inside a cue that match the given selector.

The '::cue-region' pseudo-element represents a region.

The '::cue-region(*selector*)' pseudo-element represents a region or element inside a region that match the given selector.

> Similarly to all other pseudo-elements, these pseudo-elements are not directly present in the `<video>` element's document tree.

The ':past' and ':future' pseudo-classes can be used in '::cue(*selector*)' to match WebVTT Internal Node Objects based on the current playback position.

EXAMPLE 22

The following table shows examples of what can be selected with a given selector, together with WebVTT syntax to produce the relevant objects.

Selector (CSS syntax example)	Matches (WebVTT syntax example)

'::cue'
 	
`video::cue {`
`  color: yellow;`
`}`
	Any list of WebVTT Node Objects.
	
	`WEBVTT`
	
	`00:00:00.000 --> 00:00:08.000`
	`Yellow!`
	
	`00:00:08.000 --> 00:00:16.000`
	`Also yellow!`

ID selector in '::cue()'
 	
`video::cue(#cue1) {`
`  color: yellow;`
`}`
	Any list of WebVTT Node Objects with the cue's text identifier matching the given ID.
	
	`WEBVTT`
	
	`cue1`
	`00:00:00.000 --> 00:00:08.000`
	`Yellow!`

Selector (CSS syntax example)	Matches (WebVTT syntax example)

Type selector in '::cue()'
 	
`video::cue(c),`
`video::cue(i),`
`video::cue(b),`
`video::cue(u),`
`video::cue(ruby),`
`video::cue(rt),`
`video::cue(v),`
`video::cue(lang) {`
`  color: yellow;`
`}`
	WebVTT Internal Node Objects (except the root list c WebVTT Node Objects) with the given name.
	
	`WEBVTT`
	
	`00:00:00.000 --> 00:00:08.000`
	`<c>Yellow!</c>`
	`<i>Yellow!</i>`
	`<u>Yellow!</u>`
	`<b>Yellow!</b>`
	`<u>Yellow!</u>`
	`<ruby>Yellow! <rt>Yellow!</rt></ruby>`
	`<v Kathryn>Yellow!</v>`
	`<lang en>Yellow!</lang>`

Class selector in '::cue()'
 	
`video::cue(.loud) {`
`  color: yellow;`
`}`
	WebVTT Internal Node Objects (except the root list c WebVTT Node Objects) with the given applicable cla
	
	`WEBVTT`
	
	`00:00:00.000 --> 00:00:08.000`
	`<c.loud>Yellow!</c>`
	`<i.loud>Yellow!</i>`
	`<u.loud>Yellow!</u>`
	`<b.loud>Yellow!</b>`
	`<u.loud>Yellow!</u>`
	`<ruby.loud>Yellow! <rt.loud>Yellow!</rt></r`
	`<v.loud Kathryn>Yellow!</v>`
	`<lang.loud en>Yellow!</lang>`

Attribute selector in '::cue()'	For "lang", the root list of WebVTT Node Objects or Language Object with the given applicable language; "voice", the WebVTT Voice Object with the given vo

Selector (CSS syntax example)	Matches (WebVTT syntax example)

`video::cue([lang="en-US"]) {`
`  color: yellow;`
`}`
`video::cue(lang[lang="en-GB"]) {`
`  color: cyan;`
`}`
`video::cue(v[voice="Kathryn"] {`
`  color: lime;`
`}`
	`WEBVTT`
	
	`00:00:00.000 --> 00:00:08.000`
	`Yellow!`
	
	`00:00:08.000 --> 00:00:16.000`
	`<lang en-GB>Cyan!</lang>`
	
	`00:00:16.000 --> 00:00:24.000`
	`<v Kathryn>I like lime.</v>`
	
	The applicable language for the list of WebVTT Node can be set by the `srclang` attribute in HTML.
	
	`<video ...>`
	`<track src="example-attr.vtt"`
	`srclang="en-US" default>`
	`</video>`

':lang()' pseudo-class in '::cue()'
 	
`video::cue(:lang(en)) {`
`  color: yellow;`
`}`
`video::cue(:lang(en-GB)) {`
`  color: cyan;`
`}`
	WebVTT Internal Node Objects with an applicable la matching the given language range.
	
	`WEBVTT`
	
	`00:00:00.000 --> 00:00:08.000`
	`Yellow!`
	
	`00:00:08.000 --> 00:00:16.000`
	`<lang en-GB>Cyan!</lang>`
	
	As above, the applicable language for the list of Web Objects can be set by the `srclang` attribute in HTML

':past' and ':future' pseudo-classes in '::cue()'	In cues that have WebVTT Timestamp Objects, Web Internal Node Objects, depending on the current play position.

Selector (CSS syntax example)	Matches (WebVTT syntax example)

`video::cue(:past) {`
`  color: yellow;`
`}`
`video::cue(:future) {`
`  color: cyan;`
`}`
	`WEBVTT`
	
	`00:00:00.000 --> 00:00:08.000`
	`<c>No match (no timestamps)</c>`
	
	`00:00:08.000 --> 00:00:16.000`
	`No match <00:00:12.000> (no elements)`
	
	`00:00:16.000 --> 00:00:24.000`
	`<00:00:16.000> <c>This</c>`
	`<00:00:18.000> <c>can</c>`
	`<00:00:20.000> <c>match</c>`
	`<00:00:22.000> <c>:past/:future</c>`
	`<00:00:24.000>`

'::cue-region'
 	
`video::cue-region {`
`  color: yellow;`
`}`
	Any region (list of WebVTT region objects).
	
	`WEBVTT`
	
	`REGION`
	`id:editor-comments`
	`regionanchor:0%,0%`
	`viewportanchor:0%,0%`
	
	`00:00:00.000 --> 00:00:08.000`
	`No match (normal cue)`
	
	`00:00:08.000 --> 00:00:16.000 region:editor`
	`Yellow!`

ID selector in '::cue-region()'
 	
`video::cue-region(#scroll) {`
`  color: cyan;`
`}`
	Any region (list of WebVTT region objects) with a W region identifier matching the given ID.

Selector (CSS syntax example)	Matches (WebVTT syntax example)

	`WEBVTT`
	
	`REGION`
	`id:editor-comments`
	`width: 40%`
	`regionanchor:0%,100%`
	`viewportanchor:10%,90%`
	
	`REGION`
	`id:scroll`
	`width: 40%`
	`regionanchor:100%,100%`
	`viewportanchor:90%,90%`
	`scroll:up`
	
	`00:00:00.000 --> 00:00:08.000`
	`No match (normal cue)`
	
	`00:00:08.000 --> 00:00:16.000 region:editor`
	`Yellow!`
	
	`00:00:10.000 --> 00:00:16.000 region:scroll`
	`Over here it’s Cyan!`

§ 8.2. Processing model

When a user agent is rendering one or more WebVTT cues according to the rules for updating the display of WebVTT text tracks, WebVTT Node Objects in the list of WebVTT Node Objects used in the rendering can be matched by certain pseudo-selectors as defined below. These selectors can begin or stop matching individual WebVTT Node Objects while a cue is being rendered, even in between applications of the rules for updating the display of WebVTT text tracks (which are only run when the set of active cues changes). User agents that support the pseudo-element described below must dynamically update renderings accordingly. When either ‘white-space’ or one of the properties corresponding to the ‘font’ shorthand (including ‘line-height’) changes value, then the WebVTT cue’s text track cue display state must be emptied and the text track’s rules for updating the text track rendering must be immediately rerun.

Pseudo-elements apply to elements that are matched by selectors. For the purpose of this section, that element is the *matched element*. The pseudo-elements defined in the following sections affect the styling of parts of WebVTT cues that are being rendered for the *matched element*.

> If the *matched element* is not a `<video>` element, the pseudo-elements defined below won't have any effect according to this specification.

A CSS user agent that implements the text tracks model must implement the '::cue', '::cue(*selector*)', '::cue-region' and '::cue-region(*selector*)' pseudo-elements, and the ':past' and ':future' pseudo-classes.

§ 8.2.1. The '::cue' pseudo-element

The ***::cue*** pseudo-element (with no argument) matches any list of WebVTT Node Objects constructed for the *matched element*, with the exception that the properties corresponding to the 'background' shorthand must be applied to the WebVTT cue background box rather than the list of WebVTT Node Objects.

The following properties apply to the '::cue' pseudo-element with no argument; other properties set on the pseudo-element must be ignored:

- 'color'
- 'opacity'
- 'visibility'
- the properties corresponding to the 'text-decoration' shorthand
- 'text-shadow'
- the properties corresponding to the 'background' shorthand
- the properties corresponding to the 'outline' shorthand
- the properties corresponding to the 'font' shorthand, including 'line-height'
- 'white-space'
- 'text-combine-upright'
- 'ruby-position'

The ***::cue(*selector*)*** pseudo-element with an argument must have an argument that consists of a CSS selector [SELECTORS4]. It matches any WebVTT Internal Node Object constructed for the *matched element* that also matches the given CSS selector, with the nodes being treated as follows:

- The *document tree* against which the selectors are matched is the tree of WebVTT Node Objects rooted at the list of WebVTT Node Objects for the cue.
- WebVTT Internal Node Objects are elements in the tree.
- WebVTT Leaf Node Objects cannot be matched.
- For the purposes of element type selectors, the names of WebVTT Internal Node Objects are as given by the following table, where objects having the concrete class given in a cell in the first column have the name given by the second column of the same row:

Concrete class	Name
WebVTT Class Objects	`c`
WebVTT Italic Objects	`i`
WebVTT Bold Objects	`b`
WebVTT Underline Objects	`u`
WebVTT Ruby Objects	`ruby`
WebVTT Ruby Text Objects	`rt`
WebVTT Voice Objects	`v`
WebVTT Language Objects	`lang`
Other elements (specifically, lists of WebVTT Node Objects)	No explicit name.

- For the purposes of element type and universal selectors, WebVTT Internal Node Objects are considered as being in the namespace expressed as the empty string.
- For the purposes of attribute selector matching, WebVTT Internal Node Objects have no attributes, except for WebVTT Voice Objects, which have a single attribute named "`voice`" whose value is the value of the WebVTT Voice Object, WebVTT Language Objects, which have a single attribute named "`lang`" whose value is the object's applicable language, and lists of WebVTT Node Objects that have a non-empty applicable language, which have a single attribute named "`lang`" whose value is the object's applicable language.
- For the purposes of class selector matching, WebVTT Internal Node Objects have the classes de-

scribed as the WebVTT Node Object's applicable classes.

- For the purposes of the ':lang()' pseudo-class, WebVTT Internal Node Objects have the language described as the WebVTT Node Object's applicable language.
- For the purposes of ID selector matching, lists of WebVTT Node Objects have the ID given by the cue's text track cue identifier, if any.

The following properties apply to the '::cue()' pseudo-element with an argument:

- 'color'
- 'opacity'
- 'visibility'
- the properties corresponding to the 'text-decoration' shorthand
- 'text-shadow'
- the properties corresponding to the 'background' shorthand
- the properties corresponding to the 'outline' shorthand
- properties relating to the transition and animation features

In addition, the following properties apply to the '::cue()' pseudo-element with an argument when the selector does not contain the ':past' and ':future' pseudo-classes:

- the properties corresponding to the 'font' shorthand, including 'line-height'
- 'white-space'
- 'text-combine-upright'
- 'ruby-position'

Properties that do not apply must be ignored.

As a special exception, the properties corresponding to the 'background' shorthand, when they would have been applied to the list of WebVTT Node Objects, must instead be applied to the WebVTT cue background box.

§ 8.2.2. The ':past' and ':future' pseudo-classes

The ':past' and ':future' pseudo-classes sometimes match WebVTT Node Objects. [SELECTORS4]

The ***:past*** pseudo-class only matches WebVTT Node Objects that are *in the past*.

> A WebVTT Node Object *c* is ***in the past*** if, in a pre-order, depth-first traversal of the WebVTT cue's list of WebVTT Node Objects, there exists a WebVTT Timestamp Object whose value is less than the current playback position of the media element that is the *matched element*, entirely after the WebVTT Node Object *c*.

The ***:future*** pseudo-class only matches WebVTT Node Objects that are *in the future*.

> A WebVTT Node Object *c* is ***in the future*** if, in a pre-order, depth-first traversal of the WebVTT cue's list of WebVTT Node Objects, there exists a WebVTT Timestamp Object whose value is greater than the current playback position of the media element that is the *matched element*, entirely before the WebVTT Node Object *c*.

§ 8.2.3. The '::cue-region' pseudo-element

Pseudo-elements apply to elements that are matched by selectors. For the purpose of this section, that element is the matched element. The pseudo-element defined below affects the styling of text track regions that are being rendered for the matched element.

> If the matched element is not a video element, the pseudo-element defined below won't have any effect according to this specification.

The ***::cue-region*** pseudo-element (with no argument) matches any list of WebVTT region objects constructed for the *matched element*.

The ***::cue-region(selector)*** pseudo-element with an argument must have an argument that consists of a CSS selector [SELECTORS4]. It matches any list of WebVTT region objects constructed for the *matched element* that also matches the given CSS selector as follows:

- Any region (list of WebVTT region objects) with a WebVTT region identifier matching the given ID.

No other selector matching is defined for '::cue-region(*selector*)'.

The same properties that apply to '::cue' apply to the '::cue-region' pseudo-element; other properties set on the pseudo-element must be ignored.

When a user agent is rendering one or more text track regions according to the rules for updating the display of WebVTT text tracks, WebVTT region objects used in the rendering can be matched by the above pseudo-element. User agents that support the pseudo-element must dynamically update render-

ings accordingly. When either 'white-space' or one of the properties corresponding to the 'font' shorthand (including 'line-height') changes value, then the text track cue display state of all the WebVTT cues in the region must be emptied and the text track's rules for updating the text track rendering must be immediately rerun.

§ 9. API

§ 9.1. The VTTCue interface

The following interface is used to expose WebVTT cues in the DOM API:

```
enum AutoKeyword { "auto" };
typedef (double or AutoKeyword) LineAndPositionSetting;
enum DirectionSetting { "" /* horizontal */, "rl", "lr" };
enum LineAlignSetting { "start", "center", "end" };
enum PositionAlignSetting { "line-left", "center", "line-right", "auto" };
enum AlignSetting { "start", "center", "end", "left", "right" };
[Exposed=Window,
 Constructor(double startTime, double endTime, DOMString text)]
interface VTTCue : TextTrackCue {
  attribute VTTRegion? region;
  attribute DirectionSetting vertical;
  attribute boolean snapToLines;
  attribute LineAndPositionSetting line;
  attribute LineAlignSetting lineAlign;
  attribute LineAndPositionSetting position;
  attribute PositionAlignSetting positionAlign;
  attribute double size;
  attribute AlignSetting align;
  attribute DOMString text;
  DocumentFragment getCueAsHTML();
};
```

cue **= new VTTCue(** ***startTime*****,** ***endTime*****,** ***text*** **)**

Returns a new `VTTCue` object, for use with the `addCue()` method.

The *startTime* argument sets the text track cue start time.

The *endTime* argument sets the text track cue end time.

The *text* argument sets the cue text.

cue **.** **`region`**

Returns the `VTTRegion` object to which this cue belongs, if any, or null otherwise.

Can be set.

cue **.** **`vertical`** **[=** ***value*** **]**

Returns a string representing the WebVTT cue writing direction, as follows:

↪ **If it is horizontal**

The empty string.

↪ **If it is vertical growing left**

The string "`rl`".

↪ **If it is vertical growing right**

The string "`lr`".

Can be set.

cue **.** **`snapToLines`** **[=** ***value*** **]**

Returns true if the WebVTT cue snap-to-lines flag is true, false otherwise.

Can be set.

cue **.** **`line`** **[=** ***value*** **]**

Returns the WebVTT cue line. In the case of the value being auto, the string "`auto`" is returned.

Can be set.

cue **.** **`lineAlign`** **[=** ***value*** **]**

Returns a string representing the WebVTT cue line alignment, as follows:

↪ **If it is start alignment**

The string "`start`".

↪ **If it is center alignment**

The string "`center`".

↪ **If it is end alignment**
The string "`end`".

Can be set.

***cue* . `position` [= *value*]**
Returns the WebVTT cue position. In the case of the value being auto, the string "`auto`" is returned.

Can be set.

***cue* . `positionAlign` [= *value*]**
Returns a string representing the WebVTT cue position alignment, as follows:

↪ **If it is line-left alignment**
The string "`line-left`".

↪ **If it is center alignment**
The string "`center`".

↪ **If it is line-right alignment**
The string "`line-right`".

↪ **If it is automatic alignment**
The string "`auto`".

Can be set.

***cue* . `size` [= *value*]**
Returns the WebVTT cue size.

Can be set.

***cue* . `align` [= *value*]**
Returns a string representing the WebVTT cue text alignment, as follows:

↪ **If it is start alignment**
The string "`start`".

↪ **If it is center alignment**
The string "`center`".

↪ **If it is end alignment**
The string "`end`".

↪ **If it is left alignment**
The string "`left`".

↪ **If it is right alignment**
The string "`right`".

Can be set.

***cue* . `text` [= *value*]**
Returns the cue text in raw unparsed form.

Can be set.

***fragment* = *cue* . getCueAsHTML()**
Returns the cue text as a `DocumentFragment` of HTML elements and other DOM nodes.

The ***`VTTCue(startTime, endTime, text)`*** constructor, when invoked, must run the following steps:

1. Create a new WebVTT cue. Let *cue* be that WebVTT cue.
2. Let *cue*'s text track cue start time be the value of the *startTime* argument, interpreted as a time in seconds.
3. Let *cue*'s text track cue end time be the value of the *endTime* argument, interpreted as a time in seconds.
4. Let *cue*'s cue text be the value of the *text* argument, and let the rules for extracting the chapter title be the WebVTT rules for extracting the chapter title.
5. Let *cue*'s text track cue identifier be the empty string.
6. Let *cue*'s text track cue pause-on-exit flag be false.
7. Let *cue*'s WebVTT cue region be null.
8. Let *cue*'s WebVTT cue writing direction be horizontal.
9. Let *cue*'s WebVTT cue snap-to-lines flag be true.
10. Let *cue*'s WebVTT cue line be auto.
11. Let *cue*'s WebVTT cue line alignment be start alignment.
12. Let *cue*'s WebVTT cue position be auto.

13. Let *cue*'s WebVTT cue position alignment be auto.

14. Let *cue*'s WebVTT cue size be 100.

15. Let *cue*'s WebVTT cue text alignment be center alignment.

16. Return the `VTTCue` object representing *cue*.

The ***region*** attribute, on getting, must return the `VTTRegion` object representing the WebVTT cue region of the WebVTT cue that the `VTTCue` object represents, if any; or null otherwise. On setting, the WebVTT cue region must be set to the new value.

The ***vertical*** attribute, on getting, must return the string from the second cell of the row in the table below whose first cell is the WebVTT cue writing direction of the WebVTT cue that the `VTTCue` object represents:

WebVTT cue writing direction	`vertical` value
Horizontal	"" (the empty string)
Vertical growing left	`"rl"`
Vertical growing right	`"lr"`

On setting, the WebVTT cue writing direction must be set to the value given in the first cell of the row in the table above whose second cell is a case-sensitive match for the new value.

The ***snapToLines*** attribute, on getting, must return true if the WebVTT cue snap-to-lines flag of the WebVTT cue that the `VTTCue` object represents is true; or false otherwise. On setting, the WebVTT cue snap-to-lines flag must be set to the new value.

The ***line*** attribute, on getting, must return the WebVTT cue line of the WebVTT cue that the `VTTCue` object represents. The special value auto must be represented as the string `"auto"`. On setting, the WebVTT cue line must be set to the new value; if the new value is the string `"auto"`, then it must be interpreted as the special value auto.

> In order to be able to set the `snapToLines` and `line` attributes in any order, the API does not reject setting `snapToLines` to false when `line` has a value outside the range 0..100, or vice versa.

The ***lineAlign*** attribute, on getting, must return the string from the second cell of the row in the table

below whose first cell is the WebVTT cue line alignment of the WebVTT cue that the `VTTCue` object represents:

WebVTT cue line alignment	`lineAlign` value
Start alignment	`"start"`
Center alignment	`"center"`
End alignment	`"end"`

On setting, the WebVTT cue line alignment must be set to the value given in the first cell of the row in the table above whose second cell is a case-sensitive match for the new value.

The ***`position`*** attribute, on getting, must return the WebVTT cue position of the WebVTT cue that the `VTTCue` object represents. The special value auto must be represented as the string `"auto"`. On setting, if the new value is negative or greater than 100, then an `IndexSizeError` exception must be thrown. Otherwise, the WebVTT cue position must be set to the new value; if the new value is the string `"auto"`, then it must be interpreted as the special value auto.

The ***`positionAlign`*** attribute, on getting, must return the string from the second cell of the row in the table below whose first cell is the WebVTT cue position alignment of the WebVTT cue that the `VTTCue` object represents:

WebVTT cue position alignment	`positionAlign` value
Line-left alignment	`"line-left"`
Center alignment	`"center"`
Line-right alignment	`"line-right"`
Automatic alignment	`"auto"`

On setting, the WebVTT cue position alignment must be set to the value given in the first cell of the row in the table above whose second cell is a case-sensitive match for the new value.

The ***`size`*** attribute, on getting, must return the WebVTT cue size of the WebVTT cue that the `VTTCue` object represents. On setting, if the new value is negative or greater than 100, then an

`IndexSizeError` exception must be thrown. Otherwise, the WebVTT cue size must be set to the new value.

The ***`align`*** attribute, on getting, must return the string from the second cell of the row in the table below whose first cell is the WebVTT cue text alignment of the WebVTT cue that the `VTTCue` object represents:

WebVTT cue text alignment	`align` value
Start alignment	`"start"`
Center alignment	`"center"`
End alignment	`"end"`
Left alignment	`"left"`
Right alignment	`"right"`

On setting, the WebVTT cue text alignment must be set to the value given in the first cell of the row in the table above whose second cell is a case-sensitive match for the new value.

The ***`text`*** attribute, on getting, must return the raw cue text of the WebVTT cue that the `VTTCue` object represents. On setting, the cue text must be set to the new value.

The ***`getCueAsHTML()`*** method must convert the cue text to a `DocumentFragment` for the responsible document specified by the entry settings object by applying the WebVTT cue text DOM construction rules to the result of applying the WebVTT cue text parsing rules to the cue text.

> A fallback language is not provided for `getCueAsHTML()` since a `DocumentFragment` cannot expose language information.

§ 9.2. The `VTTRegion` interface

The following interface is used to expose WebVTT regions in the DOM API:

```
enum ScrollSetting { "" /* none */, "up" };
[Exposed=Window,
 Constructor]
interface VTTRegion {
  attribute DOMString id;
  attribute double width;
  attribute unsigned long lines;
  attribute double regionAnchorX;
  attribute double regionAnchorY;
  attribute double viewportAnchorX;
  attribute double viewportAnchorY;
  attribute ScrollSetting scroll;
};
```

region **= new VTTRegion()**
Returns a new `VTTRegion` object.

region **.** **`id`**
Returns the text track region identifier. Can be set.

region **.** **`width`**
Returns the WebVTT region width as a percentage of the video width. Can be set. Throws an `IndexSizeError` if the new value is not in the range 0..100.

region **.** **`lines`**
Returns the text track region height as a number of lines. Can be set. Throws an `IndexSizeError` if the new value is negative.

region **.** **`regionAnchorX`**
Returns the WebVTT region anchor X offset as a percentage of the region width. Can be set. Throws an `IndexSizeError` if the new value is not in the range 0..100.

region **.** **`regionAnchorY`**
Returns the WebVTT region anchor Y offset as a percentage of the region height. Can be set. Throws an `IndexSizeError` if the new value is not in the range 0..100.

region **.** **`viewportAnchorX`**
Returns the WebVTT region viewport anchor X offset as a percentage of the video width. Can be set. Throws an `IndexSizeError` if the new value is not in the range 0..100.

region **.** **`viewportAnchorY`**
Returns the WebVTT region viewport anchor Y offset as a percentage of the video height. Can be set. Throws an `IndexSizeError` if the new value is not in the range 0..100.

region **.** **`scroll`**
Returns a string representing the WebVTT region scroll as follows:

↪ **If it is unset**
The empty string.

↪ **If it is up**
The string "up".

Can be set.

The ***`VTTRegion()`*** constructor, when invoked, must run the following steps:

1. Create a new WebVTT region. Let *region* be that WebVTT region.

2. Let *region*'s WebVTT region identifier be the empty string.
3. Let *region*'s WebVTT region width be 100.
4. Let *region*'s WebVTT region lines be 3.
5. Let *region*'s text track region regionAnchorX be 0.
6. Let *region*'s text track region regionAnchorY be 100.
7. Let *region*'s text track region viewportAnchorX be 0.
8. Let *region*'s text track region viewportAnchorY be 100.
9. Let *region*'s WebVTT region scroll be the empty string.
10. Return the `VTTRegion` object representing *region*.

The ***id*** attribute, on getting, must return the WebVTT region identifier of the WebVTT region that the `VTTRegion` object represents. On setting, the WebVTT region identifier must be set to the new value.

The ***width*** attribute, on getting, must return the WebVTT region width of the WebVTT region that the `VTTRegion` object represents. On setting, if the new value is negative or greater than 100, then an `IndexSizeError` exception must be thrown. Otherwise, the WebVTT region width must be set to the new value.

The ***lines*** attribute, on getting, must return the WebVTT region lines of the WebVTT region that the `VTTRegion` object represents. On setting, the WebVTT region lines must be set to the new value.

The ***regionAnchorX*** attribute, on getting, must return the WebVTT region anchor X offset of the WebVTT region that the `VTTRegion` object represents. On setting, if the new value is negative or greater than 100, then an `IndexSizeError` exception must be thrown. Otherwise, the WebVTT region anchor X distance must be set to the new value.

The ***regionAnchorY*** attribute, on getting, must return the WebVTT region anchor Y offset of the WebVTT region that the `VTTRegion` object represents. On setting, if the new value is negative or greater than 100, then an `IndexSizeError` exception must be thrown. Otherwise, the WebVTT region anchor Y distance must be set to the new value.

The ***viewportAnchorX*** attribute, on getting, must return the WebVTT region viewport anchor X offset of the WebVTT region that the `VTTRegion` object represents. On setting, if the new value is negative or greater than 100, then an `IndexSizeError` exception must be thrown. Otherwise, the WebVTT

region viewport anchor X distance must be set to the new value.

The ***viewportAnchorY*** attribute, on getting, must return the WebVTT region viewport anchor Y offset of the WebVTT region that the `VTTRegion` object represents. On setting, if the new value is negative or greater than 100, then an `IndexSizeError` exception must be thrown. Otherwise, the WebVTT region viewport anchor Y distance must be set to the new value.

The ***scroll*** attribute, on getting, must return the string from the second cell of the row in the table below whose first cell is the WebVTT region scroll setting of the WebVTT region that the `VTTRegion` object represents:

WebVTT region scroll	`scroll` value
None	"" (the empty string)
Up	"up"

On setting, the WebVTT region scroll must be set to the value given on the first cell of the row in the table above whose second cell is a case-sensitive match for the new value.

§ 10. IANA considerations

§ 10.1. *text/vtt*

This registration is for community review and will be submitted to the IESG for review, approval, and registration with IANA.

Type name:
text

Subtype name:
vtt

Required parameters:
No parameters

Optional parameters:
No parameters

Encoding considerations:
8bit (always UTF-8)

Security considerations:

Text track files themselves pose no immediate risk unless sensitive information is included within the data. Implementations, however, are required to follow specific rules when processing text tracks, to ensure that certain origin-based restrictions are honored. Failure to correctly implement these rules can result in information leakage, cross-site scripting attacks, and the like.

Interoperability considerations:

Rules for processing both conforming and non-conforming content are defined in this specification.

Published specification:

This document is the relevant specification.

Applications that use this media type:

Web browsers and other video players.

Additional information:

Magic number(s):

WebVTT files all begin with one of the following byte sequences (where "EOF" means the end of the file):

- EF BB BF 57 45 42 56 54 54 0A
- EF BB BF 57 45 42 56 54 54 0D
- EF BB BF 57 45 42 56 54 54 20
- EF BB BF 57 45 42 56 54 54 09
- EF BB BF 57 45 42 56 54 54 EOF
- 57 45 42 56 54 54 0A
- 57 45 42 56 54 54 0D
- 57 45 42 56 54 54 20
- 57 45 42 56 54 54 09
- 57 45 42 56 54 54 EOF

> (An optional UTF-8 BOM, the ASCII string "`WEBVTT`", and finally a space, tab, line break, or the end of the file.)

File extension(s):

"`vtt`"

Macintosh file type code(s):

No specific Macintosh file type codes are recommended for this type.

Person & email address to contact for further information:

Silvia Pfeiffer <silviapfeiffer1@gmail.com>

Intended usage:
Common

Restrictions on usage:
No restrictions apply.

Authors:
Silvia Pfeiffer <silviapfeiffer1@gmail.com>, Simon Pieters <simonp@opera.com>, Philip Jägenstedt <philipj@opera.com>, Ian Hickson <ian@hixie.ch>

Change controller:
W3C

Fragment identifiers have no meaning with `text/vtt` resources.

§ Privacy and Security Considerations

§ Text-based format security

As with any text-based format, it is possible to construct malicious content that might cause buffer over-runs, value overflows (e.g. string representations of integers that overflow a given word length), and the like. Implementers should take care in implementing a parser that over-long lines, field values, or encoded values do not cause security problems.

§ Styling-related privacy and security

WebVTT can embed CSS style sheets, which will be applied in user agents that support CSS. Under these circumstances, the privacy and security considerations of CSS apply, with the following caveats.

Such style sheets cannot fetch any external resources, and it is important for privacy that user agents do not allow this. Otherwise, WebVTT files could be authored such that a third party is notified when the user watches a particular video, and even the current time in that video.

It is possible for a user agent to offer user style sheets, but their presence and nature will not be detectable by scripts running in the same user agent (e.g. browser) since the CSS object model for such style sheets is not exposed to script and there is no way to get the computed style for pseudo-elements other than '::before' and '::after' with the `getComputedStyle()` API. [CSSOM]

§ Scripting-related security

WebVTT does not include or enable scripting. It is important that user agents do not support a way to execute script embedded in a WebVTT file.

However, it is possible to construct and deliver a file that is designed not to present captions or subtitles, but instead to provide timed input ('triggers') to a script system. A poorly-written script or script system might then cause security, privacy or other problems; however, this consideration really applies to the script system. Since WebVTT supplies these triggers at their timestamps, a malicious file might present such triggers very rapidly, perhaps causing undue resource consumption.

§ Privacy of preference

A user agent that selects, and causes to download or interpret a WebVTT file, might indicate to the origin server that the user has a need for captions or subtitles, and also the language preference of the user for captions or subtitles. That is a (small) piece of information about the user. However, the offering of a caption file, and the choice whether to retrieve and consume it, are really characteristics of the format or protocol which does the offer (e.g. the HTML element), rather than of the caption format itself. [HTML51]

§ Acknowledgements

Thanks to the SubRip community, including in particular Zuggy and ai4spam, for their work on the SubRip software program whose SRT file format was used as the basis for the WebVTT text track file format.

Thanks to Ian Hickson and many others for their work on the HTML standard, where WebVTT was originally specified. [HTML51]

Thanks to Addison Phillips, Alastor Wu, Andreas Tai, Anna Cavender, Anne van Kesteren, Benjamin Schaaf, Brian Quass, Caitlin Potter, Courtney Kennedy, Cyril Concolato, Dae Kim, David Singer, Eric Carlson, fantasai, Frank Olivier, Fredrik Söderquist, Giuseppe Pascale, Glenn Adams, Glenn Maynard, John Foliot, Kyle Huey, Lawrence Forooghian, Loretta Guarino Reid, Ms2ger, Nigel Megitt, Ralph Giles, Richard Ishida, Rick Eyre, Ronny Mennerich, Theresa O'Connor, and Victor Cărbune for their useful comments.

§ Index

§ Terms defined by this specification

WebVTT Timestamp Object, in §6.4

WebVTT timestamp tag state, in §6.4

WebVTT Underline Object, in §6.4

WebVTT vertical text cue setting, in §4.4

WebVTT Voice Object, in §6.4

width, in §9.2

§ Terms defined by reference

[css-align-3] defines the following terms:

- justify-content

[css-backgrounds-3] defines the following terms:

- background
- background-color
- background-image

[css-cascade-4] defines the following terms:

- @import
- cascade

[css-color-4] defines the following terms:

- color
- green
- opacity

[css-display-3] defines the following terms:

- display
- inline

[css-flexbox-1] defines the following terms:

- flex-end
- flex-flow
- inline-flex

[css-fonts-3] defines the following terms:

- font
- font-style
- font-weight

[css-fonts-4] defines the following terms:

- bold
- italic

[css-overflow-3] defines the following terms:

- hidden
- overflow

[css-position-3] defines the following terms:

- absolute
- left
- position
- relative
- top

[css-sizing-3] defines the following terms:

- auto

[CSS-SYNTAX-3] defines the following terms:

- parse a stylesheet

[css-text-3] defines the following terms:

- break-word
- center
- end
- left
- overflow-wrap
- pre-line
- right
- start
- text-align
- white-space

[css-text-decor-3] defines the following terms:

- text-decoration
- text-shadow

[css-transitions-1] defines the following terms:

- transition-duration
- transition-property

[css-ui-4] defines the following terms:

- outline

[CSS-VALUES] defines the following terms:

- vh
- vw

[CSS-WRITING-MODES-3] defines the following terms:

- unicode-bidi

[css-writing-modes-4] defines the following terms:

- horizontal-tb
- plaintext
- text-combine-upright
- writing-mode

[CSS22] defines the following terms:

- height
- line-height
- max-height
- min-height
- visibility
- width

[CSS3-RUBY] defines the following terms:

- ruby
- ruby-base
- ruby-position
- ruby-text

[CSSOM] defines the following terms:

- alternate flag
- create a css style sheet
- css rules
- css style sheet
- getComputedStyle(elt, pseudoElt)
- location
- media
- origin-clean flag
- owner css rule
- owner node
- parent css style sheet
- title

[dom-20151119] defines the following terms:

- Document
- DocumentFragment
- ProcessingInstruction
- Text
- data
- namespaceURI
- ownerDocument
- target

[ENCODING-CR] defines the following terms:

- utf-8 decode

[HTML] defines the following terms:

- presentational hints

[selectors-3] defines the following terms:

- ::after
- ::before

[SELECTORS4] defines the following terms:

- :future
- :lang()
- :past
- attribute selector
- class selector
- id selector
- originating element
- type selector

[WebIDL] defines the following terms:

- Exposed
- unsigned long

[WEBIDL-1] defines the following terms:

- DOMString
- IndexSizeError
- boolean
- double

§ References

§ Normative References

[BCP47]
A. Phillips; M. Davis. Tags for Identifying Languages. September 2009. IETF Best Current Practice. URL: https://tools.ietf.org/html/bcp47

[BIDI]
Mark Davis; Aharon Lanin; Andrew Glass. Unicode Bidirectional Algorithm. 14 May 2017. Unicode Standard Annex #9. URL: https://www.unicode.org/reports/tr9/tr9-37.html

[CSS-ALIGN-3]
Elika Etemad; Tab Atkins Jr.. CSS Box Alignment Module Level 3. URL: https://www.w3.org/TR/css-align-3/

[CSS-BACKGROUNDS-3]
Bert Bos; Elika Etemad; Brad Kemper. CSS Backgrounds and Borders Module Level 3. URL: https://www.w3.org/TR/css-backgrounds-3/

[CSS-CASCADE-4]
Elika Etemad; Tab Atkins Jr.. CSS Cascading and Inheritance Level 4. URL: https://www.w3.org/TR/css-cascade-4/

[CSS-COLOR-4]
Tab Atkins Jr.; Chris Lilley. CSS Color Module Level 4. URL: https://www.w3.org/TR/css-color-4/

[CSS-DISPLAY-3]
Elika Etemad. CSS Display Module Level 3. URL: https://www.w3.org/TR/css-display-3/

[CSS-FLEXBOX-1]
Tab Atkins Jr.; Elika Etemad; Rossen Atanassov. CSS Flexible Box Layout Module Level 1.

URL: https://www.w3.org/TR/css-flexbox-1/

[CSS-FONTS-3]
John Daggett. CSS Fonts Module Level 3. URL: https://www.w3.org/TR/css-fonts-3/

[CSS-FONTS-4]
John Daggett; Myles Maxfield. CSS Fonts Module Level 4. URL: https://www.w3.org/TR/css-fonts-4/

[CSS-OVERFLOW-3]
David Baron; Florian Rivoal. CSS Overflow Module Level 3. URL: https://www.w3.org/TR/css-overflow-3/

[CSS-POSITION-3]
Rossen Atanassov; Arron Eicholz. CSS Positioned Layout Module Level 3. URL: https://www.w3.org/TR/css-position-3/

[CSS-SIZING-3]
Elika Etemad. CSS Intrinsic & Extrinsic Sizing Module Level 3. URL: https://www.w3.org/TR/css-sizing-3/

[CSS-SYNTAX-3]
Tab Atkins Jr.; Simon Sapin. CSS Syntax Module Level 3. URL: https://www.w3.org/TR/css-syntax-3/

[CSS-TEXT-3]
Elika Etemad; Koji Ishii. CSS Text Module Level 3. URL: https://www.w3.org/TR/css-text-3/

[CSS-TEXT-4]
Elika Etemad; Koji Ishii; Alan Stearns. CSS Text Module Level 4. URL: https://www.w3.org/TR/css-text-4/

[CSS-TEXT-DECOR-3]
Elika Etemad; Koji Ishii. CSS Text Decoration Module Level 3. URL: https://www.w3.org/TR/css-text-decor-3/

[CSS-TRANSITIONS-1]
David Baron; Dean Jackson; Brian Birtles. CSS Transitions. URL: https://www.w3.org/TR/css-transitions-1/

[CSS-UI-4]
Florian Rivoal. CSS Basic User Interface Module Level 4. URL: https://www.w3.org/TR/css-ui-4/

[CSS-VALUES]
Tab Atkins Jr.; Elika Etemad. CSS Values and Units Module Level 3. URL: https://www.w3.org/TR/css-values-3/

[CSS-WRITING-MODES-3]
Elika Etemad; Koji Ishii. CSS Writing Modes Level 3. URL: https://www.w3.org/TR/css-

writing-modes-3/

[CSS-WRITING-MODES-4]

Elika Etemad; Koji Ishii. CSS Writing Modes Level 4. URL: https://www.w3.org/TR/css-writing-modes-4/

[CSS22]

Bert Bos. Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification. URL: https://www.w3.org/TR/CSS22/

[CSS3-COLOR]

Tantek Çelik; Chris Lilley; David Baron. CSS Color Module Level 3. 5 December 2017. CR. URL: https://www.w3.org/TR/css-color-3/

[CSS3-RUBY]

Elika Etemad; Koji Ishii. CSS Ruby Layout Module Level 1. URL: https://www.w3.org/TR/css-ruby-1/

[CSSOM]

Simon Pieters; Glenn Adams. CSS Object Model (CSSOM). URL: https://www.w3.org/TR/cssom-1/

[DOM-20151119]

Anne van Kesteren; et al. W3C DOM4. 19 November 2015. REC. URL: https://www.w3.org/TR/2015/REC-dom-20151119/

[ENCODING-CR]

Anne van Kesteren; Joshua Bell; Addison Phillips. Encoding. 13 April 2017. CR. URL: https://www.w3.org/TR/2017/CR-encoding-20170413/

[HTML]

Anne van Kesteren; et al. HTML Standard. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[HTML51]

Steve Faulkner; et al. HTML 5.1 2nd Edition. URL: https://www.w3.org/TR/html51/

[RFC2119]

S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC3629]

F. Yergeau. UTF-8, a transformation format of ISO 10646. November 2003. Internet Standard. URL: https://tools.ietf.org/html/rfc3629

[SELECTORS-3]

Tantek Çelik; et al. Selectors Level 3. URL: https://www.w3.org/TR/selectors-3/

[SELECTORS4]

Elika Etemad; Tab Atkins Jr.. Selectors Level 4. URL: https://www.w3.org/TR/selectors-4/

[WebIDL]
Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. URL: https://heycam.github.io/webidl/

[WEBIDL-1]
Cameron McCormack. WebIDL Level 1. URL: https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/

§ Informative References

[MAUR]
Shane McCarron; Michael Cooper; Mark Sadecki. Media Accessibility User Requirements. WD. URL: http://www.w3.org/TR/media-accessibility-reqs/

[WCAG20]
Ben Caldwell; et al. Web Content Accessibility Guidelines (WCAG) 2.0. 11 December 2008. REC. URL: https://www.w3.org/TR/WCAG20/

§ IDL Index

```
enum AutoKeyword { "auto" };
typedef (double or AutoKeyword) LineAndPositionSetting;
enum DirectionSetting { "" /* horizontal */, "rl", "lr" };
enum LineAlignSetting { "start", "center", "end" };
enum PositionAlignSetting { "line-left", "center", "line-right", "auto" };
enum AlignSetting { "start", "center", "end", "left", "right" };
[Exposed=Window,
 Constructor(double startTime, double endTime, DOMString text)]
interface VTTCue : TextTrackCue {
  attribute VTTRegion? region;
  attribute DirectionSetting vertical;
  attribute boolean snapToLines;
  attribute LineAndPositionSetting line;
  attribute LineAlignSetting lineAlign;
  attribute LineAndPositionSetting position;
  attribute PositionAlignSetting positionAlign;
  attribute double size;
  attribute AlignSetting align;
  attribute DOMString text;
  DocumentFragment getCueAsHTML();
};

enum ScrollSetting { "" /* none */, "up" };
[Exposed=Window,
 Constructor]
interface VTTRegion {
  attribute DOMString id;
  attribute double width;
  attribute unsigned long lines;
  attribute double regionAnchorX;
  attribute double regionAnchorY;
  attribute double viewportAnchorX;
  attribute double viewportAnchorY;
  attribute ScrollSetting scroll;
};
```