**

See a problem?
Select text and file a bug!

WebVTT: The Web Video Text Tracks Format

W3C First Public Working Draft 13 November 2014

This version:
http://www.w3.org/TR/2014/WD-webvtt1-20141113/
Latest published version:
http://www.w3.org/TR/webvtt1/
Latest editor's draft:
http://dev.w3.org/html5/webvtt/
Test suite:
https://github.com/w3c/web-platform-tests/tree/master/webvtt
Bug tracker:
file a bug (open bugs)
Editors:
Silvia Pfeiffer, NICTA
Philip Jägenstedt, Opera Software ASA
Ian Hickson, Google (previous editor)
Version history:
https://github.com/w3c/webvtt/commits

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.

Status of This Document

This is a Recommendation Track snapshot. For the latest updates, possibly including important bug

This version is outdated!
For the latest version, please look at https://www.w3.org/TR/webvtt1/.

▲ expand

latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Work on this specification is being undertaken both in the W Community Group as well as in the W3C Timed Text Working Group. The latter group works towards a W3C Recommendation for reference purposes with interoperability requirements, while the former is a Draft Community Group Report that continues to evolve.

This document was published by the W3C Timed Text Working Group as a First Public Working Draft. 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.

Publication as a First Public Working Draft 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 5 February 2004 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 August 2014 W3C Process Document.

Table of Contents

1. Introduction
    1.1 Cues with multiple lines
    1.2 Comments
    1.3 Other features
2. Conformance
    2.1 Conformance for authors
    2.2 Document conformance
    2.3 Dependencies
3. Data model
    3.1 Text track cues
    3.2 Text track regions

4.3 Types of WebVTT cue payload
4.3.1 WebVTT metadata text
4.3.2 WebVTT cue text
4.4 WebVTT region definition
4.5 WebVTT cue settings
4.6 Properties of cue sequences
4.6.1 WebVTT file using only nested cues
4.7 Types of WebVTT files
4.7.1 WebVTT file using metadata content
4.7.2 WebVTT file using chapter title text
4.7.3 WebVTT file using cue text
5. Parsing
5.1 WebVTT file parsing
5.2 WebVTT region settings parsing
5.3 WebVTT cue timings and settings parsing
5.4 WebVTT cue text parsing rules
5.5 WebVTT cue text DOM construction rules
6. Rendering
6.1 Cues in isolation
6.2 Cues with video
6.2.1 Processing model
6.2.2 Applying CSS properties to WebVTT Node Objects
6.2.3 CSS extensions
6.2.3.1 The '::cue' pseudo-element
6.2.3.2 The ':past' and ':future' pseudo-classes
6.2.3.3 The '::cue-region' pseudo-element
7. API
7.1 The `VTTCue` interface
7.2 The `VTTRegion` interface
8. IANA considerations
8.1 `text/vtt`
9. Acknowledgements
A. References
A.1 Normative references
A.2 Informative references

1. Introduction

This section is non-normative.

The ***WebVTT*** (Web Video Text Tracks) format is intended for marking up external text

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:end size:50%
<v Roger Bingham>When we e-mailed—

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

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

00:32.500 --> 00:33.500 align:start 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.
```

1.1 Cues with multiple lines

This section is non-normative.

Line breaks in cues are honored. User agents will also insert extra line breaks if

These captions on a public service announcement video


```
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 for the content of this advertisement, or for the consequences
of any actions taken on the basis of the information provided.
```

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:

```
          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.
```

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
```

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

1.2 Comments

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.

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.
```

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
```

1.3 Other features

This section is non-normative.

WebVTT also supports some less-often used features.

In this example, the cues have an identifier:

```
WEBVTT

1
00:00.000 --> 00:02.000
That's an, an, that's an L!

transcript credit
00:04.000 --> 00:05.000
Transcribed by Celestials™
```

This allows a style sheet to specifically target the cues (notice the use of CSS character escape sequences):

In this example, each cue says who is talking using voice
span specifying the speaker is also annotated with two c
the third cue, there is also some italics text (not associat
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: blue }
::cue(v[voice="Mary"]) { color: green }
::cue(i) { font-style: italic }
::cue(.loud) { font-size: 2em }
```

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%,start align:start size:35%
Where did he go?

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

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

The cues cover only 35% of the video viewport's width. The first cue has its cue box left aligned at the 10% mark of the video viewport width and the text is left aligned

"position:55%,start", which explicitly positions the cue box. The third cue has middle aligned text within the same type of cue box as the first c

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.

2. Conformance

2.1 Conformance for authors

The Syntax section of this specification defines what consists a valid WebVTT document. Authors need to follow the Syntax specification and are encouraged to use a validator.

The Parsing section of this specification defines in some detail the required processing

specification are valid.

2.2 Document 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 letters 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.)

2.3 Dependencies

This specification relies on several other underlying specifications.

The following term is defined in the Encoding standard: [ENCODING]

- ***UTF-8 decode***

The following terms are defined in the DOM standard: [DOM]

- `Document` interface
- `DocumentFragment` interface
- `ProcessingInstruction` interface
- `Text` interface
- `data` attribute
- `localName` attribute
- `namespaceURI` attribute
- `ownerDocument` attribute
- `target` attribute

- `class` attribute
- `lang` attribute
- `title` attribute
- ***Responsible document***
- ***Entry settings object***
- ***MIME type***
- ***Case-sensitive***
- ***Collect a sequence of characters***
- ***ASCII digits***
- ***Alphanumeric ASCII characters***
- ***Space character***
- ***Skip whitespace***
- ***Split a string on spaces***
- ***HTML namespace***
- ***Media element***
- `audio` element
- `video` element
- ***Current playback position***
- ***Expose a user interface to the user***
- ***List of text tracks***
- ***Text track***
- ***Text track kind***
- ***Text track mode***
- ***Text track showing***
- ***Text track cue***
- ***Text track list of cues***
- ***Text track cue order***
- ***Text track cue identifier***
- ***Text track cue start time***
- ***Text track cue end time***
- ***Text track cue pause-on-exit flag***
- ***Text track cue text***
- ***Text track cue active flag***
- ***Text track cue display state***
- ***Rules for updating the text track rendering***
- ***Rules for rendering the cue in isolation***
- `TextTrackCue` interface
- `addCue()` method

The following term is defined in the Web IDL specification: [WEBIDL]

3.1 Text track cues

WebVTT cues are HTML text track cues that additionally c

A cue box

The cue box of a text track cue is a box within which the text of all lines of the cue is to be rendered.

NOTE

The position of the cue box within the video frame's dimensions depends on the value of the text track cue text position and the text track cue line position.

NOTE

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 positioned vertically, with consecutive lines displayed below each other), ***vertical growing left*** (a line extends vertically and is positioned horizontally, with consecutive lines displayed to the left of each other), or ***vertical growing right*** (a line extends vertically and is positioned horizontally, with consecutive lines displayed to the right of each other).

The writing direction is a property of the text inside the cue box which influences the interpretation of the positioning settings of the cue box.

If the writing direction is horizontal, then the line position percentages are relative to the height of the video, and text position and size percentages are relative to the width of the video.

Otherwise, line position percentages are relative to the width of the video, and text position and size percentages are relative to the height of the video.

The writing direction defaults to horizontal.

multiple of the line dimensions of the first line of the cue), or whether it is a percentage of the dimension of the video.

Cues whose text track cue snap-to-lines flag is set wi area on user agents that use overscan. Cues with the flag unset will be positioned 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 position

The line position defines positioning of the cue box.

A line position is either a number giving the position of the lines of the cue, to be interpreted as defined by the writing direction and snap-to-lines flag of the cue, or the special value ***auto***, which means the position is to depend on the other showing tracks.

A text track cue has a ***text track cue computed line 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 text track cue line position is numeric, the text track cue snap-to-lines flag of the text track cue is not set, and the text track cue line position is negative or greater than 100, then return 100 and abort these steps.

2. If the text track cue line position is numeric, return the value of the text track cue line position and abort these steps. (Either the text track cue snap-to-lines flag is set, 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 text track cue snap-to-lines flag of the text track cue is not set, return the value 100 and abort these steps. (The text track cue line position is the special value auto.)

4. Let *cue* be the text track 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

9. Negate *n*.
10. Return *n*.

A line alignment

An alignment for the cue box's line position, 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 position.

Middle alignment
: The cue box is centered at the line position.

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 position.

A text track cue has a default text track cue line alignment of start.

A text position

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

The text 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.

A text track cue has a ***text track cue computed text 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 text track cue text position is numeric, then return the value of the text track cue text position and abort these steps. (Otherwise, the text track cue text position is the special value auto.)
2. If the text track cue text alignment is start or left, return 0 and abort these steps.
3. If the text track cue text alignment is end or right, return 100 and abort these

NOTE

Since the default value of the text track cue text a
is no text track cue text alignment setting for a cu
position defaults to 50%.

NOTE

Even for horizontal cues with right-to-left *paragraph direction* text, the cue box is positioned from the left edge of the video frame. This allows defining a rendering space template which can be filled with either left-to-right or right-to-left *paragraph direction* text. If such a cue box template is created with start or end aligned text, it is best to also specify a size since otherwise the text may flip from one side of the video frame to the other.

A text position alignment

An alignment for the cue box in the dimension of the writing direction, describing which part of the cue box is aligned to the text position, one of:

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

Middle alignment
: The cue box is centered at the text position.

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

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

A text track cue has a ***text track cue computed text 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 text track cue text position alignment is not auto, then return the value of the text track cue text position alignment and abort these steps.

steps.

4. If the text track cue text alignment is middle, ret
steps.

NOTE

Since the text track cue text position always measures from the left of the video (for horizontal cues) or the top (otherwise), the text track cue text position alignment start value varies between left and top for horizontal and vertical cues, but not between left and right even for changing *paragraph direction*.

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 text track cue size is 100%.

A text alignment

An alignment for all lines of text within the cue box, in the dimension of the writing direction and the *paragraph direction* [BIDI], one of:

Start alignment
The text is aligned towards the *paragraph direction* start side of the cue box.
Middle alignment
The text is aligned centered between the box's start and end sides.
End alignment
The text is aligned towards the *paragraph direction* end side of the cue box.
Left alignment
The text is aligned to the box's left side.
Right alignment
The text is aligned to the box's right side.

By default, the value of the text track cue text alignment is middle aligned.

A region

An optional text track region to which a cue belongs.

lines flag, line position, text position, size, text alignment, region, or text change value. then the user agent must empty the text track cue display s the text track's rules for updating the display of WebVTT te

3.2 Text track regions

A ***text track region*** represents a subpart of the video viewport and provides a rendering area for text track cues.

Each text track region consists of:

An identifier

An arbitrary 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.

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 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

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

For parsing, we also need the following:

A text track list of regions

A list of zero or more text track regions.

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. Exactly one 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 metadata headers.
6. One or more WebVTT line terminators to terminate the header block and separate the cues from the file header.
7. Zero or more WebVTT cues and WebVTT comments 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 metadata header*** consists of the following components, in the given order:

1. A WebVTT metadata header name.
2. A U+003A COLON (colon) character.
3. A WebVTT metadata header value.

characters and U+000D CARRIAGE RETURN (CR) characters except that the entire resulting string must not contain the substring "-->" (U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN).

A ***WebVTT cue*** 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 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.

> NOTE
>
> A WebVTT cue 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.

> NOTE
>
> 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 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

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 (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.

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

A ***WebVTT timestamp*** can be either a WebVTT timestamp representing hours, minutes, seconds and thousandths of a second or a WebVTT timestamp representing a time in seconds and fractions of a second.

> NOTE
>
> 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, which always starts at 0.

A ***WebVTT timestamp representing hours hours, minutes minutes, seconds seconds, and thousandths of a second seconds-frac***, 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.

1. Let *seconds* be the integer part of the time.
2. Let *seconds-frac* be the fractional component of the t
   the decimal fraction given to three decimal digits.
3. If *seconds* is greater than 59, then let *minutes* be the integer component of *seconds* divided by sixty, and then let *seconds* be the remainder of dividing *seconds* divided by sixty. Otherwise, let *minutes* be zero.
4. If *minutes* is greater than 59, then let *hours* be the integer component of *minutes* divided by sixty, and then let *minutes* be the remainder of dividing *minutes* divided by sixty. Otherwise, let *hours* be zero.

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 zero 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 (%).

4.2 WebVTT comments

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

1. The string "`NOTE`".
2. Optionally, the following components, in the given order:
   1. Either:

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, but such 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.

> NOTE
>
> A WebVTT comment is ignored by the parser.

4.3 Types of WebVTT cue payload

4.3.1 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 (using the `metadata` text track kind).

4.3.2 WebVTT cue text

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

The ***WebVTT 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 gt escape, representing a ">" character in the text of the cue.
- A WebVTT cue lrm escape, representing a U+200E L Unicode bidirectional formatting character in the text
- A WebVTT cue rlm escape, representing a U+200F R Unicode bidirectional formatting character in the text of the cue.
- A WebVTT cue nbsp escape, representing a U+00A0 NO-BREAK SPACE character in the text of the cue.

WebVTT cue internal text consists of an optional WebVTT line terminator, followed by zero or more WebVTT 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 component 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

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 component of its WebVTT 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`".

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.
    - A WebVTT cue amp escape, representing a "&" character in the text of the annotation.
    - A WebVTT cue lt escape, representing a "<" character in the text of the annotation.
    - A WebVTT cue gt escape, representing a ">" character in the text of the

- A WebVTT cue rlm escape, representing a U+200F RIGHT-TO-LEFT MARK Unicode bidirectional formatting character in the
- A WebVTT cue nbsp escape, representing a U+ character in the text of the cue.

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 (>).

A ***WebVTT cue amp escape*** is the five character string "`&`".

A ***WebVTT cue lt escape*** is the four character string "`<`".

A ***WebVTT cue gt escape*** is the four character string "`>`".

A ***WebVTT cue lrm escape*** is the five character string "`&lrm;`".

A ***WebVTT cue rlm escape*** is the five character string "`&rlm;`".

A ***WebVTT cue nbsp escape*** is the six character string "` `".

4.4 WebVTT region definition

A ***WebVTT region metadata header*** is a special kind of WebVTT metadata header where both of the following apply:

- The WebVTT metadata header name is the string "Re
- The WebVTT metadata header value is a WebVTT region setting list.

A ***WebVTT region*** represents its WebVTT region settings.

The ***WebVTT region setting list*** of a WebVTT region metadata header consists of zero or more of the following components, in any order, separated from each other by one or more U+0020 SPACE characters or U+0009 CHARACTER TABULATION (tab) characters. Each component must not be included more than once per WebVTT region setting 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.

> NOTE
>
> The WebVTT region setting 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+003D EQUALS SIGN character (=).
3. An arbitrary string of one 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). The string is called the ***WebVTT region identifier***.

A WebVTT region identifier must be unique amongst all the WebVTT region identifiers of

NOTE

The WebVTT region identifier gives a name to the regi 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+003D EQUALS SIGN character (=).
3. A WebVTT percentage.

NOTE

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+003D EQUALS SIGN character (=).
3. One or more ASCII digits.

NOTE

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`".

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

5. A WebVTT percentage.

NOTE

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+003D EQUALS SIGN character (=).

3. A WebVTT percentage.

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

5. A WebVTT percentage.

NOTE

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 viewport anchor is given, it defaults to 0%, 100% (i.e. the bottom left corner).

NOTE

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.

given:

1. The string "`scroll`".
2. A U+003D EQUALS SIGN character (=).
3. The string "`up`".

NOTE

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.

NOTE

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.

NOTE

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.

4.5 WebVTT cue settings

A WebVTT cue settings list consists of zero or more of the following settings. Each setting must not be included more than once per WebVTT cue settings list.

- A WebVTT vertical text cue setting.
- A WebVTT line position cue setting.

- A WebVTT region cue setting.

NOTE

A WebVTT cue settings list gives configuration options regarding the position and alignment of the cue box and the cue text within. For example, it allows a cue box to be aligned to the left or positioned at the top right with the cue text within middle aligned.

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`".

NOTE

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 position 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. a position value, either:

       To represent a specific position relative to the video frame

       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:

NOTE

A WebVTT line position cue setting configures the posi direction opposite to the writing direction. For horizonta position. The positioning is calculated relative to the start, middle, or end of the cue box, depending on the text track cue line alignment value - start by default. The position can be given either as a percentage of the video 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 frame (the first line is numbered 0), negative line numbers from the end of the frame (the last line is numbered −1).

A ***WebVTT text 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: "`start`", "`middle`", "`end`"

NOTE

A WebVTT text position cue setting configures the position of the cue box in the direction orthogonal to the WebVTT line position cue setting. For horizontal cues, this is the horizontal position. The text position is given as a percentage of the video frame. The positioning is calculated relative to the start, middle, or end of the cue box, depending on the text track cue computed text position alignment value, which is overridden by the WebVTT text position cue setting alignment value.

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 text position cue setting. For l
width of the cue box. It is given as a percentage of the

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`", "`middle`", "`end`", "`left`", "`right`"

NOTE

A WebVTT alignment cue setting configures the alignment of the text within the cue. The keywords are relative to the text 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: an arbitrary string of one 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).

NOTE

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.

4.6 Properties of cue sequences

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 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*.

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
```

```
The Final Minute
```

In this ninety-second example, the two cues partly overla
the second ends and the second starting before the first
WebVTT file using only nested cues.

See a problem?
Select text and file a bug!

4.7 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.

4.7.1 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***.

4.7.2 WebVTT file using chapter title text

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

- WebVTT cue text span
- WebVTT cue amp escape
- WebVTT cue lt escape
- WebVTT cue gt escape
- WebVTT cue lrm escape
- WebVTT cue rlm escape
- WebVTT cue nbsp escape

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.

4.7.3 WebVTT file using cue text

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

A ***WebVTT parser***, given an input byte stream and a text track list of cues *output*. must decode the byte stream using the UTF-8 decode algorithm resulting string according to the WebVTT parser algorithm cues being added to *output*. [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 *line* be a string variable. Unset the *already collected line* flag.
4. Collect a sequence of characters that are *not* U+000A LINE FEED (LF) characters. Let *line* be those characters, if any.
5. If *line* 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.

7. If *line* is more than six characters long but the first six characters do not exactly equal "WEBVTT", or the seventh character is neither a U+0020 SPACE character nor a U+0009 CHARACTER TABULATION (tab) character, then the file does not start with the correct WebVTT file signature. The file was not successfully processed.

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 text track 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. *Header*: Collect a sequence of characters that are *not* U+000A LINE FEED (LF) characters. Let *line* be those characters, if any.

11. Let *regions* be a text track list of regions.

12. *Metadata header loop*: If *line* is not the empty string, run the following substeps:

    1. *Metadata header creation*: Let *metadata* be a new WebVTT metadata header.

    2. Let metadata's name be the empty string.

    3. Let metadata's value be the empty string.

    4. If *line* contains the character ":" (A U+003A COLON), then set metadata's name to the substring of *line* before the first ":" character and metadata's value to the substring after this character.

    5. If metadata's name equals "Region":

        1. *Region creation*: Let *region* be a new text track 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. Collect WebVTT region settings from metadata's value using *region* for the results.
        9. *Region processing*: Construct a WebVTT Region Object from *region*.
        10. Append *region* to the text track list of regions *regions*.

Advance *position* to the next character in *input*.

15. If *line* contains the three-character substring "-->" (U+002D HYPHEN-MINUS, U+003E GREATER-THA*collected line* flag and jump to the step labeled *cue loop*.

16. If *line* is not the empty string, then jump back to the step labeled *header*.
17. *Cue loop*: If the *already collected line* flag is set, then jump to the step labeled *cue creation*.
18. Collect a sequence of characters that are U+000A LINE FEED (LF) characters.
19. Collect a sequence of characters that are *not* U+000A LINE FEED (LF) characters. Let *line* be those characters, if any.
20. If *line* is the empty string, then jump to the step labeled *end*. (In such a case, *position* is also forcibly past the end of *input*.)
21. *Cue creation*: Let *cue* be a new text track cue and initialize it as follows:
    1. Let *cue*'s text track cue identifier be the empty string.
    2. Let *cue*'s text track cue pause-on-exit flag be false.
    3. Let *cue*'s text track cue region be null.
    4. Let *cue*'s text track cue writing direction be horizontal.
    5. Let *cue*'s text track cue snap-to-lines flag be true.
    6. Let *cue*'s text track cue line position be auto.
    7. Let *cue*'s text track cue line alignment be start alignment.
    8. Let *cue*'s text track cue text position be auto.
    9. Let *cue*'s text track cue text position alignment be auto.
    10. Let *cue*'s text track cue size be 100.
    11. Let *cue*'s text track cue text alignment be middle alignment.
    12. Let *cue*'s text track cue text be the empty string.
22. If *line* contains the three-character substring "-->" (U+002D HYPHEN-MINUS,

23. Let *cue*'s text track cue identifier be *line*.
24. If *position* is past the end of *input*, then discard *cue* a *end*.
25. If the character indicated by *position* is a U+000A LINE FEED (LF) character, advance *position* to the next character in *input*.
26. Collect a sequence of characters that are *not* U+000A LINE FEED (LF) characters. Let *line* be those characters, if any.
27. If *line* is the empty string, then discard *cue* and jump to the step labeled *cue loop*.
28. *Timings*: Unset the *already collected line* flag.
29. Collect WebVTT cue timings and settings from *line* using *regions* for *cue*. If that fails, jump to the step labeled *bad cue*.
30. Let *cue text* be the empty string.
31. *Cue text loop*: If *position* is past the end of *input*, then jump to the step labeled *cue text processing*.
32. If the character indicated by *position* is a U+000A LINE FEED (LF) character, advance *position* to the next character in *input*.
33. Collect a sequence of characters that are *not* U+000A LINE FEED (LF) characters. Let *line* be those characters, if any.
34. If *line* is the empty string, then jump to the step labeled *cue text processing*.
35. If *line* contains the three-character substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN), then set the *already collected line* flag and jump to the step labeled *cue text processing*.
36. If *cue text* is not empty, append a U+000A LINE FEED (LF) character to *cue text*.
37. Let *cue text* be the concatenation of *cue text* and *line*.
38. Return to the step labeled *cue text loop*.
39. *Cue text processing*: Let the text track cue text of *cue* be *cue text*, and let the rules for rendering the cue in isolation be the rules for interpreting WebVTT cue text.
40. Add *cue* to the text track list of cues *output*.

43. *Bad cue loop*: If *position* is past the end of *input*, then jump to the step labeled *end*.

44. If the character indicated by *position* is a U+000A LIN
advance *position* to the next character in *input*.

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

46. If *line* contains the three-character substring "`-->`" (U+002D HYPHEN-MINUS, U+002D HYPHEN-MINUS, U+003E GREATER-THAN SIGN), then set the *already collected line* flag and jump to the step labeled *cue loop*.

47. If *line* is the empty string, then jump to the step labeled *cue loop*.

48. Otherwise, jump to the step labeled *bad cue loop*.

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

5.2 WebVTT region settings parsing

When the WebVTT parser requires that the user agent ***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+003D EQUALS SIGN character (=), or if the first U+003D EQUALS SIGN 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+003D EQUALS SIGN character (=) in that string.

    3. Let *value* be the trailing substring of *setting* starting from the character immediately after the first U+003D EQUALS SIGN 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 "..."

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

If parse a percentage string from *value* re
text track 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 text track 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 text track 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.

5. Let *region*'s text track region viewport anchor point be the tuple of the *percentage* values calculated fro
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 "scroll 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* contains any characters other than U+0025 PERCENT SIGN characters (%), U+002E DOT characters (.) and ASCII digits, then fail.
3. If *input* does not contain at least one ASCII digit, then fail.
4. If *input* contains more than one U+002E DOT character (.), then fail.
5. If any character in *input* other than the last character is a U+0025 PERCENT SIGN character (%), then fail.
6. If the last character in *input* is not a U+0025 PERCENT SIGN character (%), then fail.
7. Ignoring the trailing percent sign, interpret *input* as a real number. Let that number be the *percentage*.
8. If *percentage* is outside the range 0..100, then fail.
9. Return *percentage*.

5.3 WebVTT cue timings and settings parsing

When the algorithm above requires that the user agent ***collect WebVTT cue timings and 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 algorithm.

1. Let *input* be the string being parsed.

4. Collect a WebVTT timestamp. If that algorithm fails, then abort these steps and return failure. Otherwise, let *cue*'s text track cue start
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:

1. Let *cue*'s text track cue region be the last text track region in *regions* whose text track region iden
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 text track cue writing direction be vertical growing left.

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

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

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

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 (-) and ASCII digits, 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*.

6. If the last character in *linepos* is a U+0025 PERCENT SIGN character (%), then let *cue*'s text tra false. Otherwise, let it be true.

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

8. If *linealign* is a case-sensitive match for the string "`middle`", then let *cue*'s text track cue line alignment be middle alignment.

9. If *linealign* is a case-sensitive match for the string "`end`", then let *cue*'s text track cue line alignment be end alignment.

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 the empty string.

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* (text track cue text position's value remains the special value auto).

4. Let *cue*'s text track cue text position be *number*.

5. If *colalign* is a case-sensitive match for the string "`start`", then let *cue*'s text track cue text position alignment be start alignment.

6. If *colalign* is a case-sensitive match for the string "`middle`", then let *cue*'s text track cue text position alignment be middle alignment.

7. If *colalign* is a case-sensitive match for the string "`end`", then let *cue*'s text track cue text position alignment be end alignment.

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*

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

1. If *value* is a case-sensitive match fo
   cue's text track cue text alignment b

2. If *value* is a case-sensitive match for the string "`middle`", then let *cue*'s text track cue text alignment be middle alignment.
3. If *value* is a case-sensitive match for the string "`end`", then let *cue*'s text track cue text alignment be end alignment.
4. If *value* is a case-sensitive match for the string "`left`", then let *cue*'s text track cue text alignment be left alignment.
5. If *value* is a case-sensitive match for the string "`right`", then let *cue*'s text track 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 characters 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 characters that are ASCII digits, and let *string* be the

steps.

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

12. If *most significant units* is *hours*, or if *position* is not be[illegible] 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 characters 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 characters 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*.

5.4 WebVTT cue text parsing rules

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 code. [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 WebVTT 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 cue text.

WebVTT Bold Objects

These represent spans of bold text (a WebVTT cue bold span) in WebVTT cue text.

WebVTT Underline Objects

These represent spans of underline text (a WebVTT cue underline span) in WebVTT cue text.

WebVTT Ruby Objects

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

WebVTT Ruby Text Objects

WebVTT Voice Objects

These represent spans of text associated with a spec
voice span) in WebVTT cue text. A WebVTT Voice O
name of the voice.

WebVTT Language Objects

These represent spans of text (a WebVTT cue language span) in WebVTT 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.

To parse a string *input* supposedly containing WebVTT cue text, user agents must use the following algorithm. 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 codes, initially empty.
6. *Loop*: If *position* is past the end of *input*, return *result* and abort these steps.
7. Let *token* be the result of invoking the WebVTT cue text tokenizer.

1. Create a WebVTT Text Object whose value is the value of the string token *token*.
2. Append the newly created WebVTT Text (

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.

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

1. Create a new WebVTT Internal Node Obj 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*.

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.

9. 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 *buffer* be the empty string.
5. Let *classes* be an empty list.
6. *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*.

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

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

WebVTT data state

Set *buffer* to *c*, set *tokenizer state* to the WebVTT escape 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*.

WebVTT escape state

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

U+0026 AMPERSAND (&)

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

Alphanumeric ASCII characters

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

U+003B SEMICOLON character (;)

First, examine the value of *buffer*:

If *buffer* is the string "`&amp`", then append a U+0026 AMPERSAND character (&) to *result*.

If *buffer* is the string "`&lt`", then append a U+003C LESS-THAN SIGN character (<) to *result*.

If *buffer* is the string "`&gt`", then append a U+003E GREATER-THAN SIGN character (>) to *result*.

If *buffer* is the string "`&lrm`", then append a U+200E LEFT-TO-RIGHT

MARK character to *result*.

If *buffer* is the string "` `", then append SPACE character to *result*.

Otherwise, append *buffer* followed by a U+003B SEMICOLON character (;) to *result*.

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

U+003C LESS-THAN SIGN (<)
End-of-file marker

Append *buffer* to *result*, return a string token whose value is *result*, and abort these steps.

Anything else

Append *buffer* to *result*, append *c* to *result*, 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

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

Advance *position* to the next character in "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.

WebVTT start tag class state

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

U+0009 CHARACTER TABULATION (tab) cha
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+003E GREATER-THAN SIGN character (>)

Advance *position* to the next character in *input*, then jump to the next

Remove any leading or trailing space characters from *buffer*. and replace any sequence of one or more con
buffer with a single U+0020 SPACE chara
whose tag name is *result*, with the classes
buffer as the annotation, and abort these steps.

Anything else

Append *c* to *buffer* 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*.

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

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 th Objects, with the following mapping of WebVTT Node Obje

WebVTT Node Object	DOM node
List of WebVTT Node Objects	`DocumentFragment` node
WebVTT Region Object	`DocumentFragment` node
WebVTT Class Object	`HTMLElement` element node with `localName` "`span`".
WebVTT Italic Object	`HTMLElement` element node with `localName` "`i`".
WebVTT Bold Object	`HTMLElement` element node with `localName` "`b`".
WebVTT Underline Object	`HTMLElement` element node with `localName` "`u`".
WebVTT Ruby Object	`HTMLElement` element node with `localName` "`ruby`".
WebVTT Ruby Text Object	`HTMLElement` element node with `localName` "`rt`".
WebVTT Voice Object	`HTMLElement` element node with `localName` "`span`", and a `title` attribute set to the WebVTT Voice Object's value.
WebVTT Language Object	`HTMLElement` element node with `localName` "`span`", and 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 Object	`ProcessingInstruction` node whose `target` is "`timestamp`" and whose `data` is a WebVTT timestamp representing the value of the WebVTT Timestamp Object, with all optional components included, with one leading zero if the *hours* component is less than ten, and with no leading zeros otherwise.

Node Object has any applicable classes, must have a `class` attribute set to the string obtained by concatenating all those classes, each separated by a 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. Rendering

6.1 Cues in isolation

The ***rules for interpreting WebVTT cue text*** (e.g. for use as chapter titles) 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 text track cue text.
2. ...

6.2 Cues with video

6.2.1 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 rendered together, to avoid overlapping subtitles from multiple tracks.

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. There is nothing to render.
2. Let *video* be the media element or other playback mechanism.

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 w… 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 text track regions.

10. For each track *track* in *tracks*, append to *regions* all the regions from *track*'s list of regions.

11. If *reset* is false, then, for each text track 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 text track region width. Let *width* be '*regionWidth* vw' ('vw' is a CSS unit). [CSSVALUES]

        - Let *lineHeight* be '5.33vh' ('vh' is a CSS unit) [CSSVALUES] and *regionHeight* be the text track region lines. Let *lines* be '*lineHeight* multiplied by *regionHeight*.

        - Let *viewportAnchorX* be the x dimension of the text track region viewport anchor and *regionAnchorX* be the x dimension of the text track 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 text track region viewport anchor and *regionAnchorY* be the y dimension of the text track region anchor. Let *topOffset* be *regionAnchorY* multiplied by *lines* divided by

containing block:

1. No style sheets are associated with *regio* subsequently restyled using style sheets a 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 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 text track cue *cue* in *cues*: if *cue*'s text track cue display state has a set of CSS boxes, then:

    - If *cue*'s text track 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 text track 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 to the *cue*'s text track cue text.

    2. If *cue*'s text track 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 text track cue region.

        2. If *region*'s text track 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. Apply the Unicode Bidirectional Algorithm's Paragraph Level steps to the

the first Unicode paragraph of the cue. [BIDI]

NOTE

Within a cue, paragraph boundaries are only denoted by Type B characters, such as U+000A LINE FEED (LF), U+0085 NEXT LINE (NEL), and U+2029 PARAGRAPH SEPARATOR. (This means each line of the cue is reordered as if it was a separate paragraph.)

4. If the *paragraph embedding level* determined in the previous step is even (the *paragraph direction* is left-to-right), let *direction* be 'ltr', otherwise, let it be 'rtl'.

5. Let *offset* be the text track cue computed text position multiplied by *region*'s text track region width and divided by 100 (i.e. interpret it as a percentage of the region width).

6. Adjust *offset* using the text track cue computed text position alignment as follows:

   If the text track cue computed text position alignment is middle alignment

   Subtract half of *region*'s text track region width from *offset*.

   If the text track cue computed text position alignment is end alignment

   Subtract *region*'s text track region width from *offset*.

7. Let *left* be '*offset* %'. ('%' is a CSS unit.) [CSSVALUES]

8. Apply the terms of the CSS specifications to *nodes* with the same constraints that are used when they are applied to *nodes* of a *cue* that is not part of a region.

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

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

12. If the CSS boxes *boxes* together have a height less than the height of the *region* box, let *diff* be the absolute diff values. Increase *top* by *diff* and re-apply it

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.

When the 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. Apply the Unicode Bidirectional Algorithm's Paragraph Level steps to 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, to determine the *paragraph embedding level* of the first Unicode paragraph of the cue. [BIDI]

   NOTE

   Within a cue, paragraph boundaries are only denoted by Type B characters, such as U+000A LINE FEED (LF), U+0085 NEXT LINE (NEL), and U+2029 PARAGRAPH SEPARATOR. (This means each line of the cue is reordered as if it was a separate paragraph.)

2. If the *paragraph embedding level* determined in the previous step is even (the *paragraph direction* is left-to-right), let *direction* be 'ltr', otherwise, let it be 'rtl'.

3. If the text track cue writing direction is horizontal, then let *writing-mode* be 'horizontal-tb'. Otherwise, if the text track cue writing direction is vertical growing left, then let *writing-mode* be 'vertical-rl'. Otherwise, the text track cue writing direction is vertical growing right; let *writing-mode* be 'vertical-lr'.

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

   If the text track cue computed text position alignment is start

   Let *maximum size* be the text track cue computed text position subtracted from 100.

If the text track cue computed text position alignment is middle. and the text track cue computed text position is less than or e

Let *maximum size* be the text track cue comput two.

If the text track cue computed text position alignment is middle, and the text track cue computed text position is greater than 50

Let *maximum size* be the result of subtracting text track cue computed text position from 100 and then multiplying the result by two.

5. If the text track cue size is less than *maximum size*, then let *size* be text track cue size. Otherwise, let *size* be *maximum size*.

6. If the text track 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.) [CSSVALUES]

7. Determine the value of *x-position* or *y-position* for *cue* as per the appropriate rules from the following list:

If the text track cue writing direction is horizontal

If the text track cue computed text position alignment is start alignment

Let *x-position* be the text track cue computed text position.

If the text track cue computed text position alignment is middle alignment

Let *x-position* be the text track cue computed text position minus half of *size*.

If the text track cue computed text position alignment is end alignment

Let *x-position* be the text track cue computed text position minus *size*.

If the text track cue writing direction is vertical growing left or vertical growing right

If the text track cue computed text position alignment is start alignment

Let *y-position* be the text track cue computed text position.

If the text track cue computed text position alignment is middle

size.

If the text track cue computed text position a

Let *y-position* be the text track cue compu

8. 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 text track cue snap-to-lines flag is not set
If the text track cue writing direction is horizontal

Let *y-position* be the text track cue computed line position.

If the text track cue writing direction is vertical growing left or vertical growing right

Let *x-position* be the text track cue computed line position.

If the text track cue snap-to-lines flag is set
If the text track cue writing direction is horizontal

Let *y-position* be 0.

If the text track cue writing direction is vertical growing left or vertical growing right

Let *x-position* be 0.

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

9. If the text track cue snap-to-lines flag is set, then run the appropriate steps from the following list:

If the text track cue writing direction is horizontal

1. Let *edge margin* be a user-agent-defined horizontal length, expressed as a percentage of the width of the *video*'s rendering area, which will be used to define a margin at the left and right edges of the video into which this cue will not be placed. In situations with overscan, this margin

which can be ugly).

2. If *x-position* is less than *edge margin* and the sum of *x-position* and *size* is more than *edge margin*, then increase *x-position* by *edge margin* and decrease *size* by the same amount.

3. Let *right margin edge* be 100 minus *edge margin*.

4. If *x-position* is less than *right margin edge*, and the sum of *x-position* and *size* is more than *right margin edge*, then decrease *size* by *edge margin*.

If the text track cue writing direction is vertical growing left or vertical growing right

1. Let *edge margin* be a user-agent-defined vertical length, expressed as a percentage of the height of the *video*'s rendering area, which will be used to define a margin at the top and bottom edges of the video into which this cue will not be placed. In situations with overscan, this margin should be sufficient to place the cue within the title-safe area. In the absence of overscan, this value should be picked for aesthetics (to avoid text being aligned precisely on the top or bottom edge of the video, which can be ugly).

2. If *y-position* is less than *edge margin* and the sum of *y-position* and *size* is more than *edge margin*, then increase *y-position* by *edge margin* and decrease *size* by the same amount.

3. Let *bottom margin edge* be 100 minus *edge margin*.

4. If *y-position* is less than *bottom margin edge*, and the sum of *y-position* and *size* is more than *right margin edge*, then decrease *size* by *edge margin*.

10. 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.) [CSSVALUES]

11. Apply the terms of the CSS specifications to *nodes* within the following constraints, thus obtaining a set of CSS boxes positioned relative to an initial containing block: [CSS]

    - The *document tree* is the tree of WebVTT Node Objects rooted at *nodes*.

- 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 restyled using style sheets after their boxes are 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'. [CSSRUBY]
- 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, with the following additional constraints:
  - Regardless of the value of the 'white-space' property, lines must be wrapped at the edge of their containing blocks, even if doing so requires splitting a word where there is no line breaking opportunity. (Thus, normally text wraps as needed, but if there is a particularly long word, it does not overflow as it normally would in CSS, it is instead forcibly wrapped at the box's edge.)
  - Regardless of the value of the 'white-space' property, any line breaks inserted by the user agent for the purposes of line wrapping must be placed so as to minimize Δ across each run of consecutive lines between preserved newlines in the source. Δ for a set of lines is defined as the sum over each line of the absolute of the difference between the line's length and the mean line length of the set.
- The 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.

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

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

    If *cue*'s text track cue snap-to-lines flag is set

    Many of the steps in this algorithm vary according to the text track cue writing

or vertical growing right, steps labeled "**Vertical Growing Left**" must be followed only when the text track cue writing dir
and steps labeled "**Vertical Growing Right**" mu
text track cue writing direction is vertical growin

1. **Horizontal**: Let *margin* be a user-agent-defined vertical length which will be used to define a margin at the top and bottom edges of the video into which cues will not be placed. In situations with overscan, this margin should be sufficient to place all cues within the title-safe area. In the absence of overscan, this value should be picked for aesthetics (to avoid text being aligned precisely on the bottom edge of the video, which can be ugly).

   Vertical: Let *margin* be a user-agent-defined horizontal length which will be used to define a margin at the left and right edges of the video into which cues will not be placed. In situations with overscan, this margin should be sufficient to place all cues within the title-safe area. In the absence of overscan, this value should be picked for aesthetics (to avoid text being aligned precisely on the left or right edges of the video, which can be ugly).

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

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

   These dimensions must not be adjusted for overscan. (The algorithm does that separately.)

3. Let *max dimension* be *full dimension* - (2 × *margin*).

4. **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*.

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

6. Let *line position* be the text track cue computed line position.

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

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

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

dimension, and negate *step*.

Otherwise, increase *position* by *margin*.

12. **Horizontal**: Move all the boxes in *boxes* d
position.

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

13. Remember the position of all the boxes in *boxes* as their *specified position*.

14. Let *best position* be null. It will hold a position for *boxes*, much like *specified position* in the previous step.

15. Let *best position score* be null.

16. Let *switched* be false.

17. **Horizontal**: Let *title area* be a box that covers all of the *video*'s rendering area except for a height of *margin* at the top of the rendering area and a height of *margin* at the bottom of the rendering area.

    Vertical: Let *title area* be a box that covers all of the *video*'s rendering area except for a width of *margin* at the left of the rendering area and a width of *margin* at the right of the rendering area.

18. *Step loop*: If none of the boxes in *boxes* would overlap any of the boxes in *output*, and all of the boxes in *output* are entirely within the *title area* box, then jump to the step labeled *done positioning* below.

19. Let *current position score* be the percentage of the area of the bounding box of the boxes in *boxes* that is outside the *title area* box.

20. If *best position* is null (i.e. this is the first run through this loop, *switched* is still false, the boxes in *boxes* are at their *specified position*, and *best position score* is still null), or if *current position score* is a lower percentage than that in *best position score*, then remember the position of all the boxes in *boxes* as their *best position*, and set *best position score* to *current position score*.

21. **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

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 lab

22. **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.)

23. Jump back to the step labeled *step loop*.

24. *Switch direction*: If *switched* is true, then move all the boxes in *boxes* back to their *best position*, and jump to the step labeled *done positioning* below.

25. Otherwise, move all the boxes in *boxes* back to their *specified position* as determined in the earlier step.

26. Negate *step*.

27. Set *switched* to true.

28. Jump back to the step labeled *step loop*.

If *cue*'s text track cue snap-to-lines flag is not set

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

2. Run the appropriate steps from the following list:

   If the text track cue writing direction is horizontal

   If the text track cue line alignment is middle alignment

   Move all the boxes in *boxes* up by half of the height of *bounding box*.

   If the text track cue line alignment is end alignment

   Move all the boxes in *boxes* up by the height of *bounding box*.

   If the text track cue writing direction is vertical growing left or

Move all the boxes in *boxes* left by half of the width of *bounding box*.

If the text track cue line 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 *output* 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 *output* 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.)

14. *Done positioning*: If there are any line boxes in the (possibly now repositioned) *boxes* that do not completely fit inside *video*'s rendering area, remove those offending line boxes from *boxes*.

15. Return *boxes*.

6.2.2 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. [CSS]

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 'direction' property must be set to *direction*
- the 'writing-mode' property must be set to *writing-mode*

- the 'height' property must be set to *height*

The variables *direction*, *writing-mode*, *top*, *left*, *width*, and *h* those names determined by the rules for updating the displ the text track 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 text track cue text alignment:

Text track cue text alignment	'text-align' value
Start alignment	'start'
Middle 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'. [CSSRUBY] [CSSVALUES]

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

The 'background' shorthand property on the WebVTT cue background box must be set to 'rgba(0,0,0,0.8)'. [CSSCOLOR]

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

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'. [CSSRUBY]

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

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

- the 'word-wrap' property must be set to 'break-word'
- the 'overflow-wrap' property must be set to 'break-wo
- the 'font' shorthand property must be set to 'calc(5.33
- the 'line-height' shorthand property must be set to '5.3
- 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 text track 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 text track 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.

6.2.3 CSS extensions

defined below. These selectors can begin or stop matching individual WebVTT Node Objects while a cue is being rendered, even in between ap updating the display of WebVTT text tracks (which are only cues changes). User agents that support the pseudo-elem 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 text track 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 text track cues that are being rendered for the *matched element*.

> NOTE
>
> 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' and '::cue(*selector*)' pseudo-elements, and the ':past' and ':future' pseudo-classes.

6.2.3.1 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'
- 'text-decoration'
- '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'

constructed for the *matched element* that also matches the given group of selectors, with the nodes being treated as follows:

- The *document tree* against which the selectors are m
  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, and WebVTT Language Objects, 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 described as the WebVTT Node Object's applicable classes.
- For the purposes of the `:lang()` pseudo-class, WebVTT Internal Node Objects have

The following properties apply to the '::cue()' pseudo-element with an argument:

- 'color'
- 'opacity'
- 'visibility'
- 'text-decoration'
- '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'

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.

6.2.3.2 The ':past' and ':future' pseudo-classes

The ':past' and ':future' pseudo-classes sometimes match WebVTT Node Objects. [SELECTORS]

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 text track 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 text track 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*.

6.2.3.3 The 'cue-region' pseudo-element

affects the styling of text track regions that are being rendered for the matched element.

NOTE

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 same properties that apply to '::cue' apply to the '::cue-region' pseudo-element with no argument; 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 renderings 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 text track cues in the region must be emptied and the text track's rules for updating the text track rendering must be immediately rerun.

A CSS user agent that implements the text tracks model must implement the '::cue-region' pseudo-element.

7. API

7.1 The `VTTCue` interface

The following interface is used to expose WebVTT cues in the DOM API:

```
enum AutoKeyword { "auto" };
enum DirectionSetting { "" /* horizontal */, "rl", "lr" };
enum LineAlignSetting { "start", "middle", "end" };
enum PositionAlignSetting { "start", "middle", "end",
"auto" };
enum AlignSetting { "start", "middle", "end", "left",
"right" };
[Constructor(double startTime, double endTime, DOMString
text)]
```

```
    attribute boolean snapToLines;
    attribute (double or AutoKeyword) li
    attribute LineAlignSetting lineAlign
    attribute (double or AutoKeyword) po
    attribute PositionAlignSetting positionAlign;
    attribute double size;
    attribute AlignSetting align;
    attribute DOMString text;
    DocumentFragment getCueAsHTML();
};
```

See a problem?
Select text and file a bug!

This box is non-normative. Implementation requirements are given below this box.

***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 text track 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 text track 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`".

Returns true if the text track cue snap-to-lines flag is set. false otherwise.

Can be set.

***cue* . line [= *value*]**

Returns the text track cue line position. In the case of the value being auto, the string "`auto`" is returned.

Can be set.

***cue* . lineAlign [= *value*]**

Returns a string representing the text track cue line alignment, as follows:

If it is start alignment

The string "`start`".

If it is middle alignment

The string "`middle`".

If it is end alignment

The string "`end`".

Can be set.

***cue* . position [= *value*]**

Returns the text track cue text 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 text track cue text position alignment, as follows:

If it is start alignment

The string "`start`".

If it is middle alignment

The string "`end`".

If it is automatic alignment

The string "`auto`".

Can be set.

***cue* . size [= *value*]**

Returns the text track cue size.

Can be set.

***cue* . align [= *value*]**

Returns a string representing the text track cue text alignment, as follows:

If it is start alignment

The string "`start`".

If it is middle alignment

The string "`middle`".

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 text track cue text in raw unparsed form.

Can be set.

***fragment* = *cue* . getCueAsHTML()**

The ***VTTCue(startTime, endTime, text)*** constructor, when invoked. must run the following steps:

1. Create a new text track cue. Let *cue* be that text track
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 text track cue text be the value of the *text* argument, and let the rules for rendering the cue in isolation be the rules for interpreting WebVTT cue text.
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 text track cue region be null.
8. Let *cue*'s text track cue writing direction be horizontal.
9. Let *cue*'s text track cue snap-to-lines flag be true.
10. Let *cue*'s text track cue line position be auto.
11. Let *cue*'s text track cue line alignment be start alignment.
12. Let *cue*'s text track cue text position be auto.
13. Let *cue*'s text track cue text position alignment be auto.
14. Let *cue*'s text track cue size be 100.
15. Let *cue*'s text track cue text alignment be middle alignment.
16. Return the `VTTCue` object representing *cue*.

The ***region*** attribute, on getting, must return the `VTTRegion` object representing the text track cue region of the text track cue that the `VTTCue` object represents, if any; or null otherwise. On setting, the text track 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 text track cue writing direction of the text track cue that the `VTTCue` object represents:

Text track cue writing direction	`vertical` value
Vertical growing left	"`rl`"
Vertical growing right	"`lr`"

On setting, the text track 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 text track cue snap-to-lines flag of the text track cue that the `VTTCue` object represents is set; or false otherwise. On setting, the text track cue snap-to-lines flag must be set if the new value is true, and must be unset otherwise.

The ***`line`*** attribute, on getting, must return the text track cue line position of the text track cue that the `VTTCue` object represents. The special value auto must be represented as the string "`auto`". On setting, the text track cue line 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 ***`lineAlign`*** attribute, on getting, must return the string from the second cell of the row in the table below whose first cell is the text track cue line alignment of the text track cue that the `VTTCue` object represents:

Text track cue line alignment	`lineAlign` value
Start alignment	"`start`"
Middle alignment	"`middle`"
End alignment	"`end`"

On setting, the text track 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 text track cue text position of the text track 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 text track cue text 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 text track cue text position alignment of the

Text track cue text position alignment	`positionAlign` value
Start alignment	"start"
Middle alignment	"middle"
End alignment	"end"
Automatic alignment	"auto"

See a problem?
Select text and file a bug!

On setting, the text track cue text 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 text track cue size of the text track 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 text track 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 text track cue text alignment of the text track cue that the `VTTCue` object represents:

Text track cue text alignment	`align` value
Start alignment	"start"
Middle alignment	"middle"
End alignment	"end"
Left alignment	"left"
Right alignment	"right"

On setting, the text track 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 text track cue text of the text track cue that the `VTTCue` object represents. On setting, the text track cue text must be set to the new value.

The ***getCueAsHTML()*** method must convert the text track 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 text track cue text.

```
enum ScrollSetting { "" /* none */, "u
[Constructor]
interface VTTRegion {
  attribute double width;
  attribute long lines;
  attribute double regionAnchorX;
  attribute double regionAnchorY;
  attribute double viewportAnchorX;
  attribute double viewportAnchorY;
  attribute ScrollSetting scroll;
};
```

See a problem?
Select text and file a bug!

This box is non-normative. Implementation requirements are given below this box.

***region* = new VTTRegion()**

Returns a new VTTRegion object.

***region* . width**

Returns the text track 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.

***region* . regionAnchorX**

Returns the text track 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* . regionAnchorX**

Returns the text track 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 text track 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

Returns the text track region viewport anchor Y offset as a percentage of the video height. Can be set. Throws an `IndexSizeEr` the range 0..100.

***region* . scroll**

Returns a string representing the text track 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 text track region. Let *region* be that text track region.
2. Let *region*'s text track region identifier be the empty string.
3. Let *region*'s text track region width be 100.
4. Let *region*'s text track 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 text track region scroll be the empty string.
10. Return the `VTTRegion` object representing region.

The ***`width`*** attribute, on getting, must return the text track region width of the text track region that the `VTTRegion` object represents, in percent of video width. On setting, the text track region width must be set to the new value, interpreted as a percentage.

The ***regionAnchorX*** attribute, on getting, must return the text track region anchor X offset of the text track region that the `VTTRegion` object represents, in percent of region width. On setting, the text track region anchor X distance must be set to the new value, interpreted as a percentage.

The ***regionAnchorY*** attribute, on getting, must return the text track region anchor Y offset of the text track region that the `VTTRegion` object represents, in percent of region height. On setting, the text track region anchor Y distance must be set to the new value, interpreted as a percentage.

The ***viewportAnchorX*** attribute, on getting, must return the text track region viewport anchor X offset of the text track region that the `VTTRegion` object represents, in percent of video width. On setting, the text track region viewport anchor X distance must be set to the new value, interpreted as a percentage.

The ***viewportAnchorY*** attribute, on getting, must return the text track region viewport anchor Y offset of the text track region that the `VTTRegion` object represents, in percent of video height. On setting, the text track region viewport anchor Y distance must be set to the new value, interpreted as a percentage.

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 text track region scroll setting of the text track region that the `VTTRegion` object represents:

Text track region scroll	`scroll` value
None	"" (the empty string)
Up	"`up`"

On setting, the text track 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.

8. IANA considerations

8.1 *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:

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

NOTE

(An optional UTF-8 BOM, the ASCII string "`WEBVTT`", and finally a space, tab, line break, or the end of the file.)

Macintosh file type code(s):
No specific Macintosh file type codes are recom

Person & email address to contact for further informat
Silvia Pfeiffer <silviapfeiffer1@gmail.com>

Intended usage:
Common

Restrictions on usage:
No restrictions apply.

Authors:
Silvia Pfeiffer <silviapfeiffer1@gmail.com>, Philip Jägenstedt <philipj@opera.com>, Ian Hickson <ian@hixie.ch>

Change controller:
W3C

Fragment identifiers have no meaning with `text/vtt` resources.

9. 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 the many contributors to the HTML standard, where WebVTT was originally specified. [HTML]

Thanks to the following active contributors to this spec: Glenn Adams, Victor Cărbune, Eric Carlson, Anna Cavender, Cyril Concolato, Rick Eyre, fantasai, John Foliot, Lawrence Forooghian, Ralph Giles, Loretta Guarino Reid, Kyle Huey, Anne van Kesteren, Glenn Maynard, Ronny Mennerich, Ms2ger, Frank Olivier, Giuseppe Pascale, Simon Pieters, Caitlin Potter, David Singer.

A. References

A.1 Normative references

[BCP47]
A. Phillips; M. Davis. *Tags for Identifying Languages*. September 2009. IETF Best Current Practice. URL: http://tools.ietf.org/html/bcp47

[BIDI]
Mark Davis; Aharon Lanin; Andrew Glass. *Unicode Bidirectional Algorithm*. 5 June 2014. Unicode Standard Annex #9. URL: http://www.unicode.org/reports/tr9/

[CSS]

[CSSCOLOR]
Tantek Çelik; Chris Lilley; David Baron. *CSS Color M*
W3C Recommendation. URL: http://www.w3.org/TR/
[CSSRUBY]
Elika Etemad; Koji Ishii. *CSS Ruby Layout Module Level 1*. 5 August 2014. W3C Working Draft. URL: http://www.w3.org/TR/css-ruby-1/
[CSSVALUES]
Håkon Wium Lie; Tab Atkins Jr.; Elika Etemad. *CSS Values and Units Module Level 3*. 30 July 2013. W3C Candidate Recommendation. URL: http://www.w3.org/TR/css3-values/
[DOM]
Anne van Kesteren; Aryeh Gregor; Ms2ger; Alex Russell; Robin Berjon. *W3C DOM4*. 10 July 2014. W3C Last Call Working Draft. URL: http://www.w3.org/TR/dom/
[ENCODING]
Anne van Kesteren; Joshua Bell; Addison Phillips. *Encoding*. 16 September 2014. W3C Candidate Recommendation. URL: http://www.w3.org/TR/encoding/
[HTML]
Robin Berjon; Steve Faulkner; Travis Leithead; Erika Doyle Navara; Edward O'Connor; Silvia Pfeiffer. *HTML5*. 16 September 2014. W3C Proposed Recommendation. URL: http://www.w3.org/TR/html5/
[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: http://www.ietf.org/rfc/rfc2119.txt
[RFC3629]
F. Yergeau. *UTF-8, a transformation format of ISO 10646*. November 2003. Internet Standard. URL: http://www.ietf.org/rfc/rfc3629.txt
[SELECTORS]
Elika Etemad; Tab Atkins Jr.. *Selectors Level 4*. 2 May 2013. W3C Working Draft. URL: http://www.w3.org/TR/selectors4/
[WEBIDL]
Cameron McCormack. *Web IDL*. 19 April 2012. W3C Candidate Recommendation. URL: http://www.w3.org/TR/WebIDL/

A.2 Informative references

[MAUR]
Shane McCarron; Michael Cooper; Mark Sadecki. *Media Accessibility User Requirements*. 14 August 2014. W3C Working Draft. URL: http://www.w3.org/TR/media-accessibility-reqs/