Embedding WebVTT In WebM

2012-01-25

by: Matthew Heaney and Frank Galligan

Objective

The purpose of this document is to specify a mechanism for embedding WebVTT in a WebM file.

Background

WebVTT is a standard for subtitles, captions, and related metadata. A web video text track comprises a set of *cues*, each of which has a timestamp, settings, and the actual payload text. The cues are listed in a dedicated WebVTT file (having the .vtt file extension by convention) that is associated with a web video using the *src* and *kind* attributes of the HTML5 *track* element.

WebM is a media standard for web video. Its container format is based on Matroska and there are separate tracks for video and audio. There is interest in embedding the contents of a WebVTT file inside a WebM file, so that the video text track does not have to be carried out-of-band, separate from the video itself.

Design Ideas

Our goal is to embed the contents of a WebVTT file in a WebM file, that preserves the information from each cue, and without too much disruption to the container standard.

A WebVTT cue is a set of lines comprising an identifier, a timestamp and optional settings, followed by the payload. The payload is the text of the subtitle or caption, chapter title, or metadata.

This format is actually very similar to the SubRip file format [SRT]. Matroska already supports embedded SRT subtitles as a track (see [MKVSRT]), by embedding just the SRT payload as the data portion of a block. However, this approach might not be suitable for WebVTT because:

- Timestamps are (optionally) annotated with additional settings, such as the position and orientation of the text.
- The cue identifier is optional, but it contains potentially useful information (it does not have to be merely number text, as is the case for SRT), so it would

need to be stored somehow. (The cue identifier in SRT is required.)

The contents of the WebVTT file would be stored as its own WebM track. The information that would appear as attributes of the HTML5 track tag can be embedded in WebM Track element as follows:

- The TrackType sub-element value is 0x11 (generic "subtitle" track type).
- The label attribute is stored as the Name sub-element.
- The srclang attribute is stored as the Language sub-element.

Per the convention (see [MKVCODECID]) used for flavors of a particular video or audio codec, the CodecID for a WebVTT track is "S_TEXT/VTT/*kind*", where *kind* is one of CAPTIONS, SUBTITLES, DESCRIPTIONS, CHAPTERS, or METADATA.

There are two places where WebVTT file content can be stored. The first place is in the CodecPrivate element of the track. This is useful for situations that require all of cues to be together and immediately available, such as for chapter cues (used to navigate to a particular section of the media). This storage location could also be used for the file-wide metadata that precedes the actual WebVTT cues.

If a muxer chooses to embed WebVTT content in the CodecPrivate area of the Track header, then it should also include the "WEBVTT" keyword.

The second place to store WebVTT cues would be in the track proper. The simplest way to represent WebVTT in the stream is to store the entire cue (not just the payload part) as the data portion of a Block element. The Block would need to be part of a BlockGroup element (not a SimpleBlock) in order to use a BlockDuration element, which is necessary to fully specify the original timestamp of the cue. This is different from how SRT is handled, because the the header lines of a WebVTT cue carry essential information.

The advantage of this approach (simply embedding the entire cue as the data portion of a block) is that it simplifies muxing of a WebVTT file. When writing the file, the muxer does not have to do anything special except parse the timestamp of the cue (in order to synthesize the block time and block duration). A symmetric benefit applies to the WebM reader, which only needs to read the block, and then feed its contents *in toto* to the WebVTT parser.

The disadvantage is that the WebVTT timestamp text is duplicated as the value of the timestamp part of the Block Group element (the start time) and the Duration element (whose value is synthesized from the start and stop times). The timestamp is small compared to the total cue size, so perhaps this does not matter.

The timestamps for WebVTT cues can overlap in time. This is how roll-up captions work: multiple cues are rendered simultaneously, and when the top cue expires, the other cues move up and a new cue appears at the bottom. The WebM block timestamps must therefore be allowed to be monotonically increasing (a requirement already needed for the WebM container to support VP8 alt-ref frames), and the duration for a block must be allowed to overlap the start time of the next block.

Note that it is not an either-or decision about where to store WebVTT file content. Some of the content from the WebVTT file can go in the CodecPrivate header of the WebM, and the remainder can go in the track. In some situations, all of the data would go in the CodecPrivate (the typical case for chapter-style cues), and in other cases, all of the data would go in the track (the canonical representation for non-chapter cues).

Alternatives Considered

SRT-style Embedding of Cue Payload
An alternative is for the muxer to fully parse each WebVTT cue, embed the payload the same as for SRT (the cue payload *only* as the data part of the Block element), and use the BlockAdditions element to store the cue identifier and cue settings. In this case, there would be no need to store the WebVTT timestamp text except in its translated form, as the values of the Block timestamp and BlockDuration element.

The advantage of this approach is that the information associated with a cue would already be in binary form, so in principle this would make it simpler for parsers or other downstream clients that must also parse the WebVTT cues. (But then again, they might also use the text as is, so perhaps this is not much of an advantage.) There might be a storage penalty however, because Matroska elements do have a certain amount of overhead. The disadvantage is that this ties WebM more closely to WebVTT, since any changes to the WebVTT standard would have to be matched with concomitant changes to the WebM standard; blob-style embedding avoids this.

Storage Optimized Representation
If the duplication of the timestamp in both the block payload (in raw, text form) and block timestamp fields (in binary form) is an issue, one possibility is to strip out the actual timestamp from the cue text during the mux phase, and then synthesize it back in during the demux phase. This is possible because the conversion in either direction is lossless.

The advantage is that the original WebVTT cue could be stored more compactly in

the WebM stream. However, this does complicate muxing and demuxing, for not much benefit, because the actual timestamp is a relatively small portion of the overall cue. The muxer must do some nominal parsing anyway to determine timestamp values, so it's not that much extra work to strip out the timestamp when writing the cue. The problem is much more severe for the demuxer, though, since it must actually parse the cue, and then reconstitute the timestamp line from the information in the block; none of this is necessary using the canonical representation.

Chapters

Chapter cues are used for navigation (they are used as a kind of index), so they should probably be co-located, hence our decision to use the CodecPrivate region to store (chapter) cues. But there are other places where chapter cues could be stored, such as in the track itself, or in different level-1 elements altogether.

Chapter cues are formatted the same as other cues, but there are relatively few of them, because they are used for navigation across (relatively) large spans of time. In Matroska, block timestamps must be monotonically increasing across tracks, so it is not possible to simply put all chapter cues together in the file (as timestamped blocks) without violating the container standard.

One possibility is to create a special WebM element similar to a Cues element in Matroska (its version of a keyframe index), where the chapter cues can all go together. A similar approach would be to embed all of the chapter cues in a single block, say at the beginning of the track; this will also work for live chapters.

Yet another possibility is to simply convert the chapter cues into Matroska chapter elements (see [MKVCHAP]) and embed them that way. The issues with this approach include: not treating all streams identically, having to convert between WebVTT and Matroska formats, tying WebM too closely to future changes to the WebVTT specification, and not supporting timestamped chapter cues for live streams.

Outstanding Issues

File-Wide Metadata

It has been proposed that file-wide metadata (see [DEV] or [CHANGE]) be stored at the top of the WebVTT file, and formatted as UNIX-style name-value pairs:

```
WEBVTT
Language=zh
```

```
Kind=Caption
Version=V1_ABC
License=CC-BY-SA

1
00:00:15.000 --> 00:00:17.950
first cue
```

File-wide metadata does not have a timestamp, so all the text (up to and excluding the linefeed separator that demarcates the file-wide metadata and the first cue) could be stored in the CodecPrivate sub-element of the Track element.

We have made the (tentative) decision that the CodecPrivate region can be used to store some portion of the original WebVTT file, so that scheme will accommodate file-wide metadata already.

A metadata cue [META] is the same as other WebVTT cues, with the difference that the text has no particular interpretation, except as generic text.

Default Cue Settings

The cue settings are attached to the timestamp line, and it has been suggested (see [DEV] or [CHANGE]) that the syntax be modified to allow for default cue settings to be specified. The timestamp line retains the distinguished arrow symbol ("-->"), but the actual timestamps are omitted:

```
WEBVTT

DEFAULTS --> D:vertical A:end

00:00.000 --> 00:02.000
This is vertical and end-aligned.
```

Block elements must have a timestamp value, so it's not clear how this cue should be embedded in the WebM stream. One idea is to use the start time of following cue, and embed the cue as a Block either within a BlockGroup that omits the BlockDuration element, or simply use a SimpleBlock element instead (possible here because no BlockDuration need be present).

Another idea is to embed both the default settings cue in the same block as the following (normal) cue. This might complicate the demuxer, though, since there is no longer a one-to-one correspondence between a WebVTT cue and a WebM block.

Inline CSS and Comments could be handled similarly.

The presence of default cue settings implies that the state of subtitle rendering system depends on everything that has come before (at least, what default cue settings have come before). One problem is that seeking into the middle of the stream will break the rendering, because the seek will skip over cues that potentially specify new default settings.

Assuming that seeking is desired, there are at least a couple of ways to handle this. One way is to simply not write any default settings cues into the WebM track. Instead, write the settings explicitly, in the normal way, on the same line as the timestamp. If a cue overrides the default, then that value would be preserved; otherwise, write the current default value.

Another way is to write a default settings cue that is the union of all current defaults, whenever you write a block that is the target of a WebM Cue (typically a video keyframe). A muxer will have to make similar arrangements anyway, to ensure that the WebVTT cues associated with a video frame are placed on the same cluster as the frame itself (similar to the same rule we have for audio).

In both cases, the muxer will have to a be WebVTT interpreter too, since each cue will need to be parsed to determine whether it specifies a default settings cue, and then actual default settings will need to be parsed.

Yet another possibility is that the default settings cues could be embedded separately, in a different part of the file, say in the CodecPrivate area. During the demuxing phase, the original stream of cues could be reconstituted from the set of settings cues and the normal cues. This still means more work for the muxer, however, because it must parse enough of the cue to determine what kind of cue it is, and then parse the settings too. (No matter what, the muxer will need to parse the timestamp of the cue, in order to synthesize the timestamp of the block, so perhaps the extra parsing is not too much of a greater burden.)

Metadata Format

WebVTT has explicit support for a metadata track, but it's not clear exactly what WebVTT metadata looks like. Is it name/value pairs? In any event, WebM will probably have to standardize a few of the names, no matter how it is formatted.

There might also be interest in supporting XMP (see [XMP1CORE] and [XMP2PROP]), RDF (see for example [RDF]), JSON, or some other stylized form of metadata the payload of a metadata cue. For example, it might be possible to embed, say, the GPS coordinates of the video track using XMP as follows:

```
00:42:00 -> 00:42:00
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="XMP Core 5.1.2">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
     <rdf:Description rdf:about=""
           xmlns:exif="http://ns.adobe.com/exif/1.0/">
        <exif:GPSLatitude>73/1,45/1,2272/100</exif:GPSLatitude>
        <exif:GPSLongitude>42/1,50/1,5427/100</exif:GPSLongitude>
        <exif:GPSLatitudeRef>W</exif:GPSLatitudeRef>
        <exif:GPSLongitudeRef>N</exif:GPSLongitudeRef>
     </rdf:Description>
  </rdf:RDF>
</x:xmpmeta>
```

(In this example, it's not clear whether the markup needs to escaped using the standard ampersand sequence.)

References

[WEBVTT] WebVTT Living Standard (accessed 12 Jan 2012)
http://dev.w3.org/html5/webvtt/

[CHANGE]
http://www.w3.org/WAI/PF/HTML/wiki/Media_WebVTT_Changes

[DEV]
http://blog.gingertech.net/2011/06/27/recent-developments-around-webvtt/

[META]
http://dev.w3.org/html5/webvtt/#webvtt-metadata-text

[MKV]
http://matroska.org/technical/specs/index.html

[MKVSRT] SRT Subtitles
http://www.matroska.org/technical/specs/subtitles/srt.html

[MKVCODECID] Matroska Codec Specs
http://matroska.org/technical/specs/codecid/index.html

[MKVCHAP]
http://matroska.org/technical/specs/chapters/index.html

[SRT] SubRip
http://en.wikipedia.org/wiki/SubRip

[WEBM]

http://www.webmproject.org/code/specs/container/

[XMP1CORE] XMP Specification, Part 1, Data Model, Serialization, and Core Properties

http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/XMPSpecificationPart1.pdf

[XMP2PROP] XMP Specification, Part 2, Additional Properties

http://wwwimages.adobe.com/www.adobe.com/content/dam/Adobe/en/devnet/xmp/pdfs/XMPSpecificationPart2.pdf

[RDF] Embedding RDF in WebVTT

http://ninsuna.elis.ugent.be/node/39