Timed Text Tracks and TV Services

Eric Winkelman – CableLabs, August 15, 2011

1. Abstract

HTML5 provides a new mechanism, Timed Text Tracks[1], for synchronizing video with accessibility features such as Captions and Subtitles. This mechanism includes a scripting interface that can be used to deliver other TV Services, including ad insertion and bound applications. This paper describes our prototype implementation of this facility, and identifies areas where additional work is required to support TV Services.

2. Timed Text Tracks

2.1 Overview

Timed Text Tracks are associated with video elements using <track> tags. Each track tag has a *kind* attribute indicating what the track supports (*subtitles, captions, metadata,* etc.) and an optional *src* attribute identifying a file containing cue information.

Each track tag is associated with a track element that contains the track's cue data. If a *src* attribute is specified, the cue data is read from the specified file, otherwise cues are created by the media player from in-band data or through the use of a user provided script.

The cues contain timing information indicating, in video stream time, when the cue-enter and cue-exit events should be fired, as well as additional data that is used to process each cue's events. A subtitle, for example, will have the stream time for starting and stopping the display of the subtitle, and the subtitle's text as the cue's data.

2.2 Implementation

When we started the proof-of-concept Timed Text Tracks were not supported in off-the-shelf browsers, so our investigation began by adding this feature to an HTML5 browser. By design, web browsers are very tolerant of unknown markup, and allow user applications to modify the Document Object Model (DOM) which is used for the internal representation of the web page. This flexibility allowed us to use JavaScript to add Timed Text Track support directly to a web page displaying the video.

When the browser encounters the unknown <track> tags, it inserts them into the DOM as an element without any special properties. Once the page is loaded, our JavaScript code locates these <track> tags and adds the Timed Text Track

functionality directly to the DOM element. The resulting track element behaves identically with the proposed built-in functionality.

2.3 Creating Cues

There are three methods for acquiring the cues: out-of-band cues can be created using a *src* file, in-band cues can be created by the media player, or cues can be created through a user provided script.

To create cues through a file, the cues need to be specified in the WebVTT[2] format. Once the web page is loaded, the cue file is automatically read, and parsed. The resulting cues are then associated with the track element.

The user agent can create cues from in-band metadata by using the interface defined by the specification. , We implemented this functionality in GStreamer[3]-based players through a custom plugin. We're currently working on adding this functionality to Chromium[4].

There is also an interface allowing scripts to create tracks and cues. This approach can be used to load cues from non-WebVTT files, or through AJAX.

While the format is specified for most of the kinds of cues, metadata cues can contain data of any type/format.

2.4 Script Interface

The scripting interface to Timed Text Tracks is through event handlers. While scripts can access any of the kinds of tracks, *metadata* tracks are processed solely by scripts. The browser monitors the video's play time, and fires two events for each cue. The first event, onenter, specifies when the cue becomes active. The second event, onexit, specifies when the cue becomes inactive. The registered event handlers for each of these events are then invoked on the fired cue.

Once the event handler is invoked it can access the cue's data to perform the desired action. We've chosen to format the cue's data as XML, as it is both flexible and easy to work with using JavaScript.

3. Ad Insertion

3.1 Signaling

For client-side ad insertion, we write an XML representation of an SCTE 35[5] Segmentation descriptor to the cue's data field. The key information in this signal is the start and end times, and the Segmentation Descriptor Id.

SCTE 35 signals are sent several times before the actual insertion point, with the first signal coming four seconds before the insertion point. When an onenter event

fires before the insertion time, that cue is interpreted as a “load” command, and the advertisement is loaded by the client. When an onenter fires at the insertion time, that signal is used as a “play” command. The ad is stopped when an onexit event is received at the time the ad is supposed to end.

3.2 Display

Client-side ads are displayed in web page using an <iframe>. This frame is positioned directly over the programming video, and hides it completely. The ad frame is hidden during the program and is only made visible when the ad is playing. This approach has an advantage in that the ad is loaded while the frame is hidden, and will buffer up before it is displayed. When the ad is played, it is made visible and starts playback immediately. When the ad is finished, playback is stopped, and the frame is hidden again.

It is important to note that the program stream continues to play during the advertisement, so there are two videos playing at one time. The viewer will be completely unaware of this, however, as the program video is hidden during the advertisement.

3.3 Ad Server

The ad to insert is determined by the URL of the program video and the Segmentation Descriptor Id contained in the cue’s data. This implementation requires this combination to be unique.

When the ad is loaded, this information is sent to an ad server, along with any cookies, and it redirects the frame to the appropriate advertisement. The mechanism used by the ad server to select this ad is outside the scope of this paper, but could be based on SCTE 130[6], or other mechanisms.

4. ETV

Enhanced TV[7] is a CableLabs’ standard for synchronizing the execution of an application with video playback. This is used for interactive content. Our proof-of-concept uses the ETV[7] signaling data (EISS)[8] to synchronize the execution of Web content with video playback; the CableLabs’ standard signaling messages are carried in Timed Text metadata tracks.

4.1 Signaling

For bound applications, we write an XML representation of an EISS signal to the cue’s data field. The key information in this signal is the EISS command (PREPARE/AUTOSTART/DESTROY) and the URL of the associated EBIF[7] application.

When an onenter fires with a PREPARE command, the associated ETV application is loaded by the client. When an onenter fires with an AUTOSTART command, the ETV

application is started. When an onenter fires with a DESTROY command, the ETV application is stopped. The onexit events are ignored in all cases.

4.2 Display

ETV applications are displayed in a web page using an <iframe>. This frame is positioned directly over the programming video. The contents of the frame determine whether the underlying video is visible or not. The application frame is hidden while the application is not running, and is only made visible when the application is started. This approach has an advantage in that the application is loaded while the frame is hidden, and will load before it is displayed. When the application is started, it is made visible and displays immediately. When the application is finished, it is stopped and the frame is hidden again.

It is important to note that the program stream continues to play while the ETV application is active, whether it is visible or not.

In order to control the application, we require it to implement a standard interface that can be used to start and stop the application. This approach is similar to how applets are designed.

4.3 Application Server

The ETV application that is run on the browser is determined by the URL of the associated EBIF application. When the application is loaded this information, along with any cookies, is sent to an application server that redirects the frame to the appropriate application. The mechanism for selecting this application is outside the scope of this paper, as are any issues in how the application is created.

5. Open Issues

There are a number of open issues that need to be resolved for wide-spread application of this technology:

- How is the type of application data in the in-band metadata track made available to script?
- How is the information in standard signals (ETV, Ad Insertion, Access Restrictions) mapped into cues?
- How does script become aware of new metadata tracks that may appear over time in continuous media streams, broadcast channels for example?

We've submitted an HTML spec bug (13359)[9] identifying the need for a type attribute for the track element to resolve the first issue. The second issue can be addressed by creation of specifications that define the mapping process. This topic

has been raised in the Web and TV IG Media Pipeline task force. The third issue has also been identified by an HTML spec bug (13358)[10].

6. References

[1] http://dev.w3.org/html5/spec/Overview.html#timed-text-tracks
[2] http://www.whatwg.org/specs/web-apps/current-work/multipage/the-video-element.html#webvtt-0
[3] http://gstreamer.freedesktop.org
[4] http://code.google.com/chromium/
[5] http://www.scte.org/documents/pdf/standards/ANSI_SCTE 35 2007 Digital Program Insertion Cueing Message for Cable.pdf
[6] http://www.scte.org/documents/pdf/Standards/ANSI_SCTE 130-1 2008.pdf
[7] http://www.cablelabs.com/specifications/OC-SP-ETV-BIF1.0-I05-091125.pdf
[8] http://www.cablelabs.com/specifications/OC-SP-ETV-AM1.0-I06-110128.pdf
[9] http://lists.w3.org/Archives/Public/public-html-bugzilla/2011Jul/0920.html
[10] http://lists.w3.org/Archives/Public/public-html-bugzilla/2011Jul/0919.html