HTML 5 – Part III
Audio & video

Laura Farinetti
Dipartimento di Automatica e Informatica
Politecnico di Torino
laura.farinetti@polito.it

HTML5 media elements

- Inserting a video or an audio should be as easy as inserting an image
  - Browsers should have built-in support for playing video
  - No third-party plugins should be required
  - Standard formats should exist that are supported by all browsers
- HTML5 defines a standard way to embed video or audio in a web page, using a <video> or <audio> element
- New HTML5 media elements

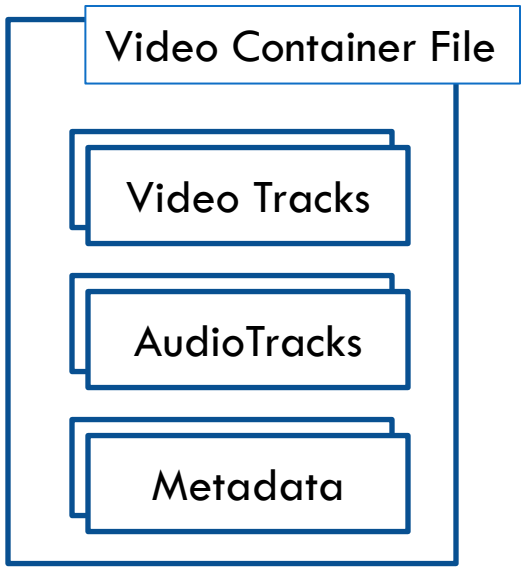
Tag	Description
<audio>	For multimedia content, sounds, music or other audio streams
<video>	For video content, such as a movie clip or other video streams
<source>	For media resources for media elements, defined inside video or audio elements
<embed>	For embedded content, such as a plug-in

HTML5 media elements

- The new audio and video tags make multimedia no longer a second-class citizen on the web
  - No separate download or enabled/disabled issues
  - No separate rendering (problems with HTML elements overlap)
  - Keyboard accessibility, styling with CSS, combining video and canvas
  - http://www.craftymind.com/factory/html5video/CanvasVideo.html
  - http://www.craftymind.com/factory/html5video/CanvasVideo3D.html
- The media elements expose a common, integrated, and scriptable API to the document
  - You can design and program your own multimedia controls (e.g., play, seek, etc.)

Audio and video files

- An audio or video file is just a container file, similar to a ZIP archive file that contains a number of files (audio tracks, video tracks, and additional metadata)
  - The audio and video tracks are combined at runtime to play the video
  - Metadata contains information about the video such as cover art, title and subtitle, captioning information, …



Audio and video codecs

- Algorithms are used to encode and decode a particular audio or video stream so that they can be played back
- A codec is able to understand a specific container format and decodes the audio and video tracks that it contains
- Examples of audio codecs
  - MPEG-3: MPEG-1 or MPEG-2 Audio Layer III
  - AAC (Advanced Audio Coding): designed to be the successor of the MP3 format and included in the MPEG-4 specification, AAC generally achieves better sound quality than MP3 at similar bit rates
  - Ogg Vorbis: open, patent-free, professional audio encoding and streaming technology from the Xiph.org Foundation
- Examples of video codecs
  - H.264: currently one of the most commonly used formats for the recording, compression, and distribution of high definition video
  - VP8: open video compression format released by Google
  - Ogg Theora: free and open video compression format from the Xiph.org Foundation

Audio and video codecs

- Some of the codecs are patented, while others are freely available
  - For example, the Vorbis audio codec and the Theora video codec are freely available, while the use of the MPEG-4 and H.264 codecs are subject to license fees
- Originally, the HTML5 specification was going to require that certain codecs were supported
  - Unfortunately, there does not appear to be a single codec that all browser vendors are willing to implement
- For now, the codec requirement has been dropped from the specification
  - This decision might be revisited in the future…

Video formats

- Currently, there are 3 supported video formats for the video element
  - Ogg = Ogg files with Theora video codec and Vorbis audio codec
  - MPEG4 = MPEG 4 files with H.264 video codec and AAC audio codec
  - WebM = WebM files with VP8 video codec and Vorbis audio codec

http://www.videojs.com/html5-video-support/

Declaring a media element

```
<!DOCTYPE html>
<html>
<title>HTML5 Audio </title>
<audio controls
   src="johann_sebastian_bach_air.ogg">
   An audio clip from Johann Sebastian Bach.
</audio>
</html>
```

audio.html

- The controls attribute tells the browser to display common user controls for starting, stopping, and seeking in the media clip, as well as volume control

Declaring a media element

- Leaving out the controls attribute hides them, and leaves the clip with no way for the user to start playing
  - It will not show anything at all in the case of audio files, as the only visual representation of an audio element is its controls
- By including the autoplay attribute, the media file will play as soon as it is loaded, without any user interaction

```
<audio autoplay>
  <source src="johann_sebastian_bach_air.ogg"
     type="audio/ogg; codecs=vorbis">
  An audio clip from Johann Sebastian Bach.
</audio>
```

audio2.html

Multiple sources

- But what if the browser in question does not support that container or codec?
- An alternate declaration can be used that includes multiple sources from which the browser can choose

```
<audio controls>
  <source src="johann_sebastian_bach_air.ogg">
  <source src="johann_sebastian_bach_air.mp3">
  An audio clip from Johann Sebastian Bach.
</audio>
```

- Sources are processed in order, so a browser that can play multiple listed source types will use the first one it encounters
- The beauty of this declaration model is that as you write code to interact with the media file, it doesn't matter to you which container or codec was actually used: the browser provides a unified interface for you to manipulate the media

JS APIs for media control

Function	Behavior
`load()`	Loads the media file and prepares it for playback. Normally does not need to be called unless the element itself is dynamically created. Useful for loading in advance of actual playback.
`play()`	Loads (if necessary) and plays the media file. Plays from the beginning unless the media is already paused at another position.
`pause()`	Pauses playback if currently active.
`canPlayType(type)`	Tests to see whether the `video` element can play a hypothetical file of the given MIME type.

Media attributes

Read-only attribute	Value
`duration`	The duration of the full media clip, in seconds. If the full duration is not known, `NaN` is returned.
`paused`	Returns `true` if the media clip is currently paused. Defaults to `true` if the clip has not started playing.
`ended`	Returns `true` if the media clip has finished playing.
`startTime`	Returns the earliest possible value for playback start time. This will usually be 0.0 unless the media clip is streamed and earlier content has left the buffer.
`error`	An error code, if an error has occurred.
`currentSrc`	Returns the string representing the file that is currently being displayed or loaded. This will match the source element selected by the browser.

Media attributes

Attribute	Value
`autoplay`	Sets the media clip to play upon creation or query whether it is set to `autoplay`.
`loop`	Returns `true` if the clip will restart upon ending or sets the clip to loop (or not loop).
`currentTime`	Returns the current time in seconds that has elapsed since the beginning of the playback. Sets `currentTime` to seek to a specific position in the clip playback.
`controls`	Shows or hides the user controls, or queries whether they are currently visible.
`volume`	Sets the audio volume to a relative value between 0.0 and 1.0, or queries the value of the same.
`muted`	Mutes or unmutes the audio, or determines the current mute state.
`autobuffer`	Tells the player whether or not to attempt to load the media file before playback is initiated. If the media is set for auto-playback, this attribute is ignored.

Example (audio)

Play

```
<audio id="clickSound">
  <source src="johann_sebastian_bach_air.ogg">
  <source src="johann_sebastian_bach_air.mp3">
</audio>
<button id="toggle" onclick="toggleSound()">Play</button>

<script type="text/javascript">
function toggleSound() {
  var music = document.getElementById("clickSound");
  var toggle = document.getElementById("toggle");
  if (music.paused) {
    music.play();
    toggle.innerHTML = "Pause";
  }
  else {
    music.pause();
    toggle.innerHTML ="Play";
  }
}
</script>
```

audioCue.html

Additional video attributes

Attribute	Value
`poster`	The URL of an image file used to represent the video content before it has loaded. Think "movie poster." This attribute can be read or altered to change the poster.
`width, height`	Read or set the visual display size. This may cause centering, letterboxing, or pillaring if the set width does not match the size of the video itself.
`videoWidth, videoHeight`	Return the intrinsic or natural width and height of the video. They cannot be set.

Example: mouseover video playback

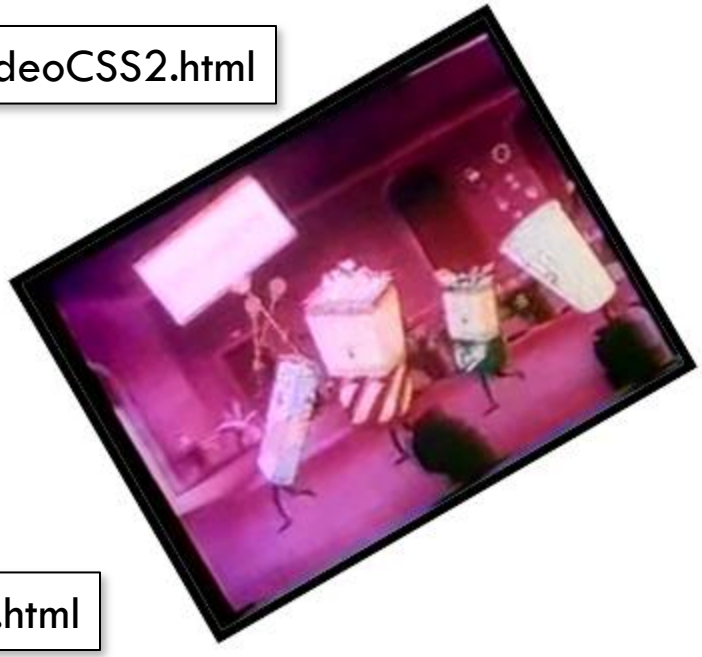
```
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="styles.css">
  <title>Mouseover Video</title>
  <video id="movies" onmouseover="this.play()"
    onmouseout="this.pause()" autobuffer="true"
    width="400px" height="300px">
    <source src="Intermission-Walk-in.ogv"
      type='video/ogg; codecs="theora, vorbis"'>
    <source src="Intermission-Walk-in_512kb.mp4"
      type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  </video>
</html>
```

mouseoverVideo.html

HTML5 video + CSS

- The video tag can be styled using traditional CSS (e.g. border, opacity, etc) since it is a first-class citizen in the DOM
  - You can also style it with the latest CSS3 properties like reections, masks, gradients, transforms, transitions and animations
- Examples

videoCSS2.html



videoCSS1.html

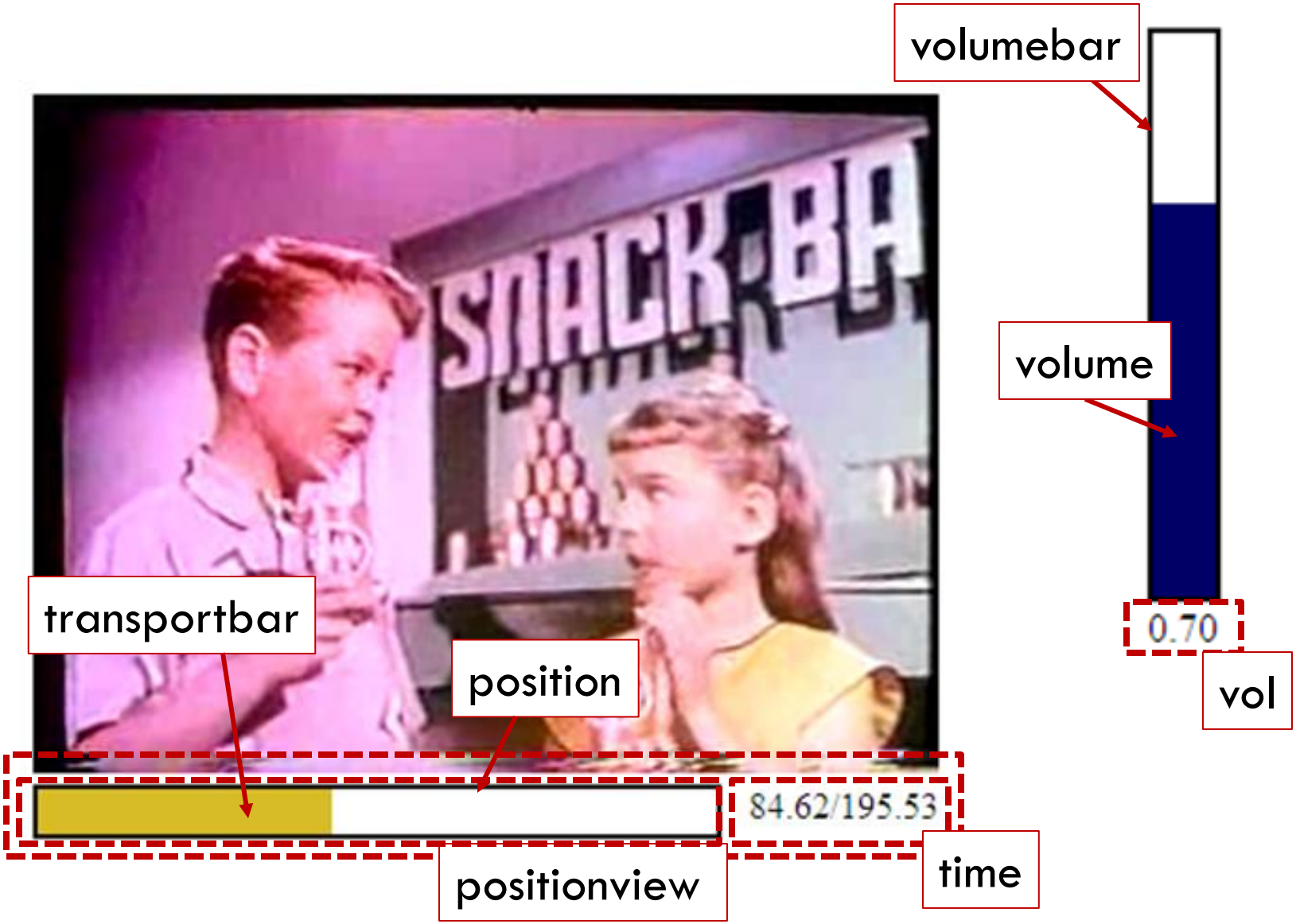
HTML5 video + JavaScript

- Example

videoJS.html

HTML5 video + JavaScript

HTML5 video + JavaScript



Example: video timeline viewer

videoTimeline.html

Example: video timeline viewer

```
<video id="movies" autoplay oncanplay="startVideo()"
       onended="stopTimeline()" autobuffer="true"
       width="400px" height="300px">
  <source src="Intermission-Walk-in.ogv"
       type='video/ogg; codecs="theora, vorbis"'>
  <source src="Intermission-Walk-in_512kb.mp4"
       type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
</video>
```

- Autoplay attribute: the video starts as soon as the page loads
- Two additional event handler functions, oncanplay (when the video is loaded and ready to begin play) and onended (when the video ends)

```
<canvas id="timeline" width="400px" height="300px">
```

- Canvas called timeline into which we will draw frames of video at regular intervals

Example: video timeline viewer

- Variables declaration

```
// # of milliseconds between timeline frame updates (5sec)
var updateInterval = 5000;
// size of the timeline frames
var frameWidth = 100;
var frameHeight = 75;
// number of timeline frames
var frameRows = 4;
var frameColumns = 4;
var frameGrid = frameRows * frameColumns;
// current frame
var frameCount = 0;
// to cancel the timer at end of play
var intervalId;

var videoStarted = false;
```

Example: video timeline viewer

- Function updateFrame: grabs a video frame and draws it onto the canvas

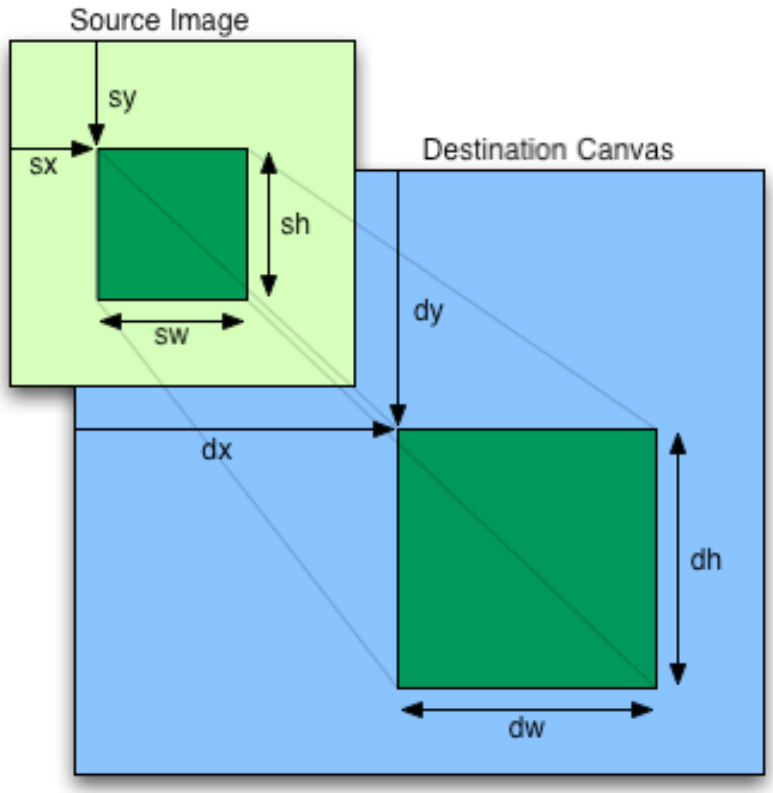
```
// paints a representation of the video frame into canvas
function updateFrame() {
  var video = document.getElementById("movies");
  var timeline = document.getElementById("timeline");
  var ctx = timeline.getContext("2d");
  // calculate out the current position based on frame
  // count, then draw the image there using the video
  // as a source
  var framePosition = frameCount % frameGrid;
  var frameX = (framePosition % frameColumns) * frameWidth;
  var frameY = (Math.floor(framePosition / frameRows)) *
               frameHeight;
  ctx.drawImage(video, 0, 0, 400, 300, frameX, frameY,
  frameWidth, frameHeight);
  frameCount++;
}
```

Example: video timeline viewer


```
frameCount = 25
framePosition = 25 % 16 = 9
frameX = (9 % 4) * 100 = 100
frameY = (Math.floor(9 / 4)) * 75 = 150
ctx.drawImage(video, 0, 0, 400, 300, 100, 150, 100, 75)
```

Canvas: drawImage

```
cxt.drawImage(image, dx, dy)
cxt.drawImage(image, dx, dy, dw, dh)
cxt.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```



- The first argument can be an image, a canvas or a video element
- When a canvas uses a video as an input source, it draws only the currently displayed video frame
  - Canvas displays will not dynamically update as the video plays
  - If you want the canvas content to update, you must redraw your images as the video is playing

Example: video timeline viewer

- Function startVideo: updates the timeline frames regularly
  - The startVideo() function is triggered as soon as the video has loaded enough to begin playing

```
function startVideo() {
  // only set up the timer the first time the video starts
  if (videoStarted) return;
  videoStarted = true;
  // calculate an initial frame, then create
  // additional frames on a regular timer
  updateFrame();
  intervalId = setInterval(updateFrame, updateInterval);
  ...
```

- setInterval: calls a function repeatedly, with a fixed time delay between each call to that function

```
var intervalID = window.setInterval(func, delay);
```

Example: video timeline viewer

- Function startVideo: handles user clicks on the individual timeline frames

```
// set up a handler to seek the video when a frame
// is clicked
var timeline = document.getElementById("timeline");
timeline.onclick = function(evt) {
  var offX = evt.layerX - timeline.offsetLeft;
  var offY = evt.layerY - timeline.offsetTop;
```

- offsetLeft: returns the number of pixels that the upper left corner of the current element is offset to the left within the parent node
- offsetTop: returns the distance of the current element relative to the top of the parent node
- layerX: returns the horizontal coordinate of the event relative to the current layer
- layerY: returns the vertical coordinate of the event relative to the current layer

Example: video timeline viewer

```
// calculate which frame in the grid was clicked
// from a zero-based index
var clickedFrame = Math.floor(offY/frameHeight)* frameRows;
clickedFrame += Math.floor(offX/frameWidth);
// find the actual frame since the video started
var seekedFrame = (((Math.floor(frameCount/frameGrid))*
  frameGrid) + clickedFrame);
```

- The clicked frame should be only one of the most recent video frames, so seekedFrame determines the most recent frame that corresponds to that grid index

Example: video timeline viewer


```
offX= 120
offY= 60
clickedFrame = Math.floor(60/75)* 4 = 0
clickedFrame += Math.floor(120/100)= 1
seekedFrame = (((Math.floor(25/16))* 16) + 1 = 17
```

Example: video timeline viewer

- Function startVideo: handles user clicks on the individual timeline frames

```
    // if the user clicked ahead of the current frame
    // then assume it was the last round of frames
    if (clickedFrame > (frameCount%16))
      seekedFrame -= frameGrid;
    // can't seek before the video
    if (seekedFrame < 0) return;
    // seek the video to that frame (in seconds)
    var video = document.getElementById("movies");
    video.currentTime = seekedFrame * updateInterval / 1000;
    // then set the frame count to our destination
    frameCount = seekedFrame;
  }
}
```

Example: video timeline viewer

- Function stopTimeline: stops capturing frames when the video finishes playing
  - The stopTimeline handler is be called when the "onended" video handler is triggered, i.e. by the completion of video playback.

```
 // stop gathering the timeline frames
function stopTimeline() {
  clearInterval(intervalId);
}
```

- clearInterval: cancels repeated action which was set up using setInterval()

Event listener

- addEventListener(eventType, listener, useCapture)
  - method that associates a function with a particular event and binds the event to the current node
- 3 parameters
  - eventType: a string representing the event to bind, without the "on" prefix (e.g. "click", "mousedown", …)
  - listener: the object or function to fire when the event fires. The actual parameter entered should be a reference to the function or object (ie: "dothis" instead of "dothis()")
  - useCapture: a Boolean value. If true, the node listens for the event type only while the event propagates toward the target node (in event capture node). If false, the node listens only when the event bubbles outward from the event target. If the current node is the target of the event, either Boolean value may be used
- The advantage of using the DOM to bind an event is that you can assign multiple functions to a node for the same event (ie: window.onload) without running into event handler conflicts

Synchronization

- There is currently no good API for synchronizing things with the timeline of a video (captions, infoboxes)
- Until something is standardized, for now
  - We can use a timer and read currentTime
  - We can listen for timeupdate and read currentTime
- Notes:
  - timeupdate is red with 15 to 250 ms interval while the video is playing, unless the previous event handler for timeupdate is still running
  - you cannot rely on the interval being the same over time or between browsers or devices
  - note that the setInterval, unless cleared, also runs when the video is not playing

Video with JavaScript synchronised captions

videoCaption.html

Video with JavaScript synchronised captions

```
<div id="transcript">
<h3>Transcript</h3>
<p>
<span data-begin=1 data-end=6>And now, before the next show
   starts, let's enjoy an intermission!</span>
<span data-begin=6 data-end=10>You'll find our snack bar chopped
   full of good things to eat and drink.</span>
<span data-begin=10 data-end=11.5>Tasty, tempting hotdogs, </span>
<span data-begin=12 data-end=14>thirst quenching soft drinks,
   </span>
<span data-begin=14 data-end=16>fresh crunchy popcorn, </span>
<span data-begin=16 data-end=19>a complete assortment of delicious
   candy,</span>
<span data-begin=19 data-end=21>and a full lot of cigarettes.
   </span>
...
</p>
</div>
```

Video with JavaScript synchronised captions

- In order for the script to know when to display each span, each of them is time-stamped
  - New feature of HTML5 that allows any element to have custom data attributes to pass data to scripts

```
<span data-begin=14 data-end=16>fresh crunchy popcorn, </span>
```

- The script hides the div that contains the plain transcript (JavaScript to write a CSS rule that set it to display:none)

```
document.write('<style>#transcript{display:none}</style>');
```

Custom attributes in HTML 5

- Formal support of custom attributes inside HTML elements
- Technically it is always possible to insert arbitrary attributes into an element and parse them using JavaScript getAttribute() method, but it is not valid HTML:

```
<div id="mydiv" brand="toyota" model="prius">
  John is very happy with his Toyota Prius, because he
  saves on gas.</div>
```

- In HTML 5, you can define custom attributes, but the attribute name must be prefixed with "data-" in order to validate:

```
<div id="mydiv" data-brand="toyota" data-model="prius">
  John is very happy with his Toyota Prius, because he
  saves on gas.</div>
```

Video with JavaScript synchronised captions

- The script grabs each span from the hidden div and positions them on top of the video at the correct time
- Overlaying the text is easy: it positions another div (with an id of caption) over the top of the video (with a text-shadow to improve legibility)
- To determine when to overlay each span, the script uses the ontimeupdate event to interrogate the video API and find out how long it has been playing
  - The video fires the ontimeupdate event every about 250ms in Opera
- Then it loops around the span elements in the transcript until it finds one with a data-begin and data-end time that encompasses the current time

Multilingual synchronized captions

Language Switcher

- English
- Italiano

videoCaption-lang.html

Other synchronization examples

- http://chirls.com/2011/01/13/what-im-working-on-synchronized-videos-in-html5-featuring-ok-go/

Video events

- The Video Events Test Page demonstrates the new HTML5 video element, its media API, and the media events
  - Play, pause, and seek in the entire video, change the volume, mute, change the playback rate (including going into negative values)
  - See the effect on the video and on the underlying events and properties
  - The Media Events table contains the number of times each media event has been received
  - The Media Properties table contains the value of the media properties

Video events

load() play() pause() currentTime+=10 currentTime-=10 currentTime=50
playbackRate++ playbackRate-- playbackRate+=0.1 playbackRate-=0.1
volume+=0.1 volume-=0.1 muted=true muted=false
Sintel teaser Bunny trailer Bunny movie Test movie

Media Events

loadstart	1	progress	20	suspend	2	abort	0	error	0		
emptied	0	stalled	0	play	1	pause	0	loadedmetadata	1		
loadeddata	1	waiting	5	playing	5	canplay	5	canplaythrough	5		
seeking	5	seeked	5	timeupdate	51	ended	0	ratechange	0		
durationchange	1	volumechange	0								

Media Properties

error		src		currentSrc	http://media.w3.org/2010/05/sintel/trailer.webm
networkState	1	preload	none	buffered	[object TimeRanges]
readyState	4	seeking	false	currentTime	24.75
initialTime	undefined	duration	52.663	startOffsetTime	undefined
paused	false	defaultPlaybackRate	undefined	playbackRate	undefined
played	undefined	seekable	undefined	ended	false
autoplay	false	loop	undefined	controls	true
volume	1	muted	false		

http://www.w3.org/2010/05/video/mediaevents.html

Licenza d'uso

- Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo 2.5 Italia (CC BY-NC-SA 2.5)"
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera
- Alle seguenti condizioni:
  - **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - **Non commerciale** — Non puoi usare quest'opera per fini commerciali.
  - **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- http://creativecommons.org/licenses/by-nc-sa/2.5/it/