TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

A Comparative Evaluation of Current HTML5 Web Video Implementations

Martin Hoernig, Andreas Bigontina, Bernd Radig

TUM-I1411

A Comparative Evaluation of Current HTML5 Web Video Implementations

Martin Hoernig, Andreas Bigontina, Bernd Radig
Image Understanding and Knowledge-Based Systems
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
{hoernig,bigontia,radig}@in.tum.de

June 20, 2014

Abstract

HTML5 video is the upcoming standard for playing videos on the World Wide Web. Although its specification has not been fully adapted yet, all major browsers provide the HTML5 video element and web developers already rely on its functionality. But there are differences between implementations and inaccuracies that trouble the web developer community. To help to improve the current situation we draw a comparison between the most important web browsers. We focus on the event mechanism, since it is essential for interacting with the video element. Furthermore we compare the seeking accuracy, which is relevant for more specialized applications. Our tests reveal varieties of differences between browser interfaces and show that even simple software solutions may still need third-party plugins in today's browsers.

Keywords: HTML5 video, events, seeking, user agents, browser, evaluation

1 Introduction

HTML5 [BFL+14] introduces a standard for the integration of videos or movies into web pages: HTML5 video. Before HTML5, browser plugins (like Adobe Flash or Microsoft Silverlight) were usually used. Now, given a modern web browser, a web developer should be able to achieve comparable or even better results with native techniques. To examine the situation, we focus our experiments to the most popular browsers and operating systems which have a market share of over 95% (see, for instance, [Net14] for a browser usage statistic). Hence, Microsoft Internet Explorer, Mozilla Firefox and Google Chrome are tested on Microsoft Windows and Apple Safari on Apple OS X. The HTML5 standard does not specify a video format for user agents. Therefore, the supported video formats vary. We decided to perform our experiments with MP4 video (h264 AVC and AAC encoded data in MP4 containers), which is the only video format supported by all user agents within our test set. Furthermore, it is a widely used industry standard for applications like broadcast TV or video on mobile devices.

To test the browser integrations, we perform a comparison of two major features, an usual application has to deal with: events and seeking. In Section 2 the defined flow of events for an example scenario is shown based on the HTML5 specification [BFL+14]. This flow is then compared with outputs the test browsers produced. Section 3 addresses the seeking capabilities. In particular, we check whether the shown frame changes appropriately, when the *seeked* event occurs, by examining the frame number and measuring any deviation from the expected one.

2 Events

Events are an important link between HTML and JavaScript and are part of the HTML specification. They are used to communicate all kinds of user input and user agent (respectively browser) occurrences to the running web application. In the case of the video element, events are used to treat incidents occurring during the playback of a video. A collection of events important in this section and their appearance is shown in Table 1.

To check the different browsers against a wrong event behavior, we created a semi-automatic test setup consisting of a video playback with buffer underrun and seeking. Because we had to force the buffer underrun via a reduction of the network bandwidth with an external application, we tested this part manually. Our test procedure contains the following commands (1, 2, 5) and conditions (3, 4):

1. **load**: Load a test video, assure that the network bitrate is high enough for interruption-free playback. Note that *autoplay* is disabled.
2. **start**: Start playback (via JavaScript) and play the first seconds.
3. **underrun**: Force a buffer underrun by slowing down the connection while the user-agent tries to receive more data.
4. **reset**: Reestablish a connection with sufficient bandwidth for interruption-free playback. (The playback should start automatically.)
5. **seek**: Seek forward to an unbuffered position. (The playback should start as data is forthcoming.)
6. **end**: The video reaches the end.

A video element processed in a hypothetical standard conform user agent would fire an event sequence according to Figure 1. This is not the only valid sequence as some options exist, e.g. a user agent can suspend the loading step after it has loaded the meta data to reduce bandwidth and fire the *suspend* event. A *stall* event could also be fired if the buffer underrun (4) did last too long (user agent specific timeout) and "data is unexpectedly not forthcoming." [BFL+14, 4.7.10.16 Event summary] However, the pretended buffer underrun (4) is defined to be short enough not to evoke a *stall* event. Besides such exceptions, the events are mandatory and have to occur in the given sequence. Since we know the network rate and the video bit rate, we can take the *canplaythrough* event as mandatory as well.

The behavior of the test browsers according to the points (1) to (6) is evaluated with the event sequence a [BFL+14]-conform user agent evokes. A compilation of all events is shown in Figure 1. Our primary test video is "Big Buck Bunny" from The Peach Open Movie Project [Sun08], which is also used as a test video at W3C.

Event name	Fired when...
progress	The user agent is fetching media data.
suspend	The user agent is intentionally not currently fetching media data.
stalled	The user agent is trying to fetch media data, but data is unexpectedly not forthcoming.
loadedmetadata	The user agent has just determined the duration and dimensions of the media resource and the text tracks are ready.
loadeddata	The user agent can render the media data at the current playback position for the first time.
canplay	The user agent can resume playback of the media data, but estimates that if playback were to be started now, the media resource could not be rendered at the current playback rate up to its end without having to stop for further buffering of content.
canplaythrough	The user agent estimates that if playback were to be started now, the media resource could be rendered at the current playback rate all the way to its end without having to stop for further buffering.
playing	Playback is ready to start after having been paused or delayed due to lack of media data.
waiting	Playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course.
seeking	The seeking IDL attribute changed to true, and the user agent has started seeking to a new position.
seeked	The seeking IDL attribute changed to false after the current playback position was changed.
ended	Playback has stopped because the end of the media resource was reached.
timeupdate	The current playback position changed as part of normal playback or in an especially interesting way, for example discontinuously.
resize	One or both of the videoWidth and videoHeight attributes have just been updated.
Further events:	*loadstart*, *abort*, *error*, *emptied*, *durationchange*, *play*, *pause*, *ratechange*, *volumechange*

Table 1: Summary of events, cited from [BFL$^+$14, 4.7.10.16 Event summary]

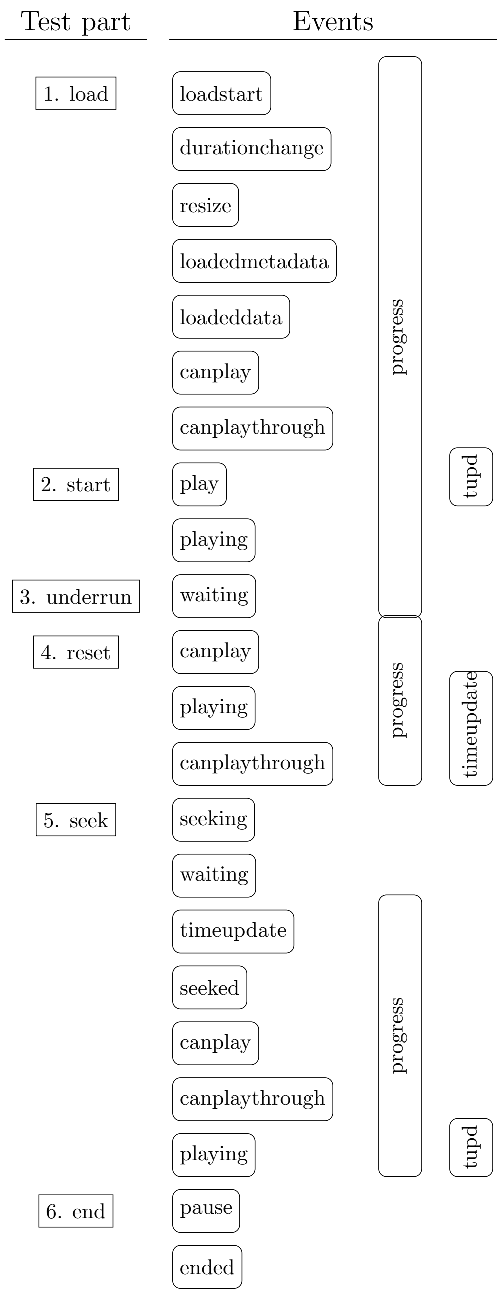

Figure 1: Event sequence a standard conform user agent fires during loading (1), playing (2), buffer underrun start (3) and end (4), seeking (5), and video end (6). Elements with rounded corners illustrate the events fired during this process. Horizontally shown events are fired once in the given order, vertical entries represent events fired repeatedly.

User agent, version, OS	1. load	2. start	3. underrun	4. reset	5. seek	6. end
Internet Explorer 11.0.9600.16521 Windows	m: *resize*	✓	m: *waiting*	m: *canplay*, *playing*, *canplaythrough*	m: *canplay*, *canplaythrough*	m: *pause*
Firefox 29.0 Windows	m: *resize*	✓	m: *waiting*	✓	wrong order; wf: *waiting*	✓
Chrome 34.0.1847.131 Windows	wf: *timeupdate*	✓	m: *waiting*	m: *canplay*, *playing*, *canplaythrough*	✓	✓
Safari 7.0 (9537.71) OS X	m: *resize*	✓	um: *waiting*	um: *canplay* *playing*	m: *waiting*, *canplay*, *canplaythrough* *playing*	✓

Table 2: Event test results. The user agents undergo our tests with some discrepancies. The candidates showed the absence of expected events (missing m, unpredictably missing um) and unexpected events (wrongly fired wf).

Unfortunately, not a single browser passes our test setup (see Table 2). Only Chrome fires *resize* on load as required, but also sends a *timeupdate* event which should not be present. While the video start is handled correctly, only Safari fires sometimes the *waiting* event if the playback has stopped. We denoted such behavior as unpredictable missing (um) as we have not discovered the pattern underneath. The playback resume on the other hand is done right by Firefox only. Then again seeking is handled right on Chrome, but the most important events in this context, *seeking* and *seeked*, are present all across the test set. In Internet Explorer a *pause* event is missing when the video ends.

Remarks:

- Internet Explorer fires *timeupdate* events in waiting state (when no update has to be committed). These events must be fired every 15 to 250 ms during the “time marches on” steps [BFL+14, 4.7.10.8 Playing the media resource], which apply if the current playback position changes, what is not the case in waiting state.
- Firefox unpredictably fires the *canplaythrough* event multiple times after seeking.
- Chrome uses an optional substep within the resource fetching algorithm [BFL+14, 4.7.10.5 Loading the media resource] which is designed for a conservative download strategy, if the user does not request the resource actively (e.g. in a preload step). This is not true in our case. The resource fetching fires a lot of *suspend* events to the video element, which is in this sense a wrong behavior.
- Safari shows some unpredictable behavior if buffer underruns occur, for example:

– No audio is played after the playback starts again.
– The specification states that a user agent must show the last rendered frame in the waiting state [BFL+14, 4.7.6 The video element], but Safari unpredictably shows a loop of the last frames.
– The *canplaythrough* event is fired together with a *waiting* event, but *canplaythrough* should only be fired if the new internal ready state is $HAVE\_ENOUGH\_DATA$ [BFL+14, 4.7.10.7 Ready states]. It is a strange behavior if the ready state alternates between $HAVE\_ENOUGH\_DATA$ and something else without reason, particularly within a buffer underrun.

Considering today's conditions, a web application cannot rely on a unified user agent behavior. While video loading and end detection are no noteworthy problems, the treatment of buffer underruns paints a different picture. Every tested browser showed problems with the handling of the *waiting* event to signal that "playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course." [BFL+14, 4.8.10.15 Event summary] In this situation, a proper buffer underrun handling is not possible. In addition *resize* is handled correctly only in Chrome .

Nevertheless the events *seeking* and *seeked* were present across all tested user agents. In the next section, we are going to examine how reliable they are.

3 Seeking

In this section we will examine the seeking capabilities of the selected user agents. An implementation compatible with the standard will set the *seeking* attribute to true and fire a *seeking* event when the *currentTime* attribute is changed. When the video data at the requested time is available, *seeking* has to be set to false and a *seeked* event must be fired.

In a first series of tests this basic behavior is examined. We use a video of 10 minutes length and jump to 1000 positions in that video. We repeat that procedure 10 times with different positions. These positions were chosen at random at a preparation step, but are the same for all user agents. Results are shown in Table 3.

User agent, version	Operating system	*seeking* event	*seeked* event
Internet Explorer 11.0.9600.16521	Windows	100%	100%
Firefox 28.0	Windows	100%	100%
Chrome 33.0.1750.154	Windows	100%	100%
Safari 7.0 (9537.71)	OS X	100%	100%

Table 3: *seeking* and *seeked* events are fired appropriately in all tested user agents.

These events are crucial for seeking in videos, and indeed, as mentioned above, all tested user agents pass this test. Though, a different question is if firing these events correlates with the changes of the displayed segment of the video. Is the requested frame already available when the *seeked* event is fired?

A different aspect is the accuracy of the seeking implementation. In some scenarios the frame rate is known to the application and it intents to step through the video in a frame-by-frame manner. Another application might store the position in a video to return to it later. Will the user be able to see exactly the same frame again, or will there be a deviation?

To answer those questions, we created a video that stores the frame number in each image. This is simply done by a binary coding of this number and drawing it as black and white blocks onto the image. Some additional checksum bits ensure the correctness of the encoding. When playing this video with a user agent, we can retrieve the frame number by drawing the video onto a canvas element and examining the pixel data.

As before we seek to 1000 positions in the video and check the frame number when the *seeked* event is fired. This procedure is repeated ten times with different positions. Various issues have been observed, depending on the user agent. The results are summarized in Table 4.

User agent, version	Operating system	*seeked* event fired without change of the image? (e.g. too early)
Internet Explorer 11.0.9600.16521	Windows	Happens in 86.17% of all cases.
Firefox 28.0	Windows	Happens when seeking to the last frame of the video or beyond.
Chrome 33.0.1750.154	Windows	Never
Safari 7.0 (9537.71)	OS X	Happens when seeking as reaction to a *loadedmetadata* event.

Table 4: We examine whether a *seeked* event actually indicates a change of the displayed frame and observe several issues.

Using Interent Explorer, in 86.17 % of all cases it was not possible to retrieve the expected frame number from a video when the *seeked* event was fired. However, when checking again later a change in the frame number could be observed (the latencies differed depending on video and test system). It is thereby likely that the *seeked* event is fired while the video element is still decoding or rendering the requested frame.

When trying to seek to the last frame or any frame beyond, Firefox does not seek at all. Nevertheless, it does fire the *seeked* event. Note that [BFL+14, 4.8.10.9 Seeking] says "If the new playback position is later than the end of the media resource, then let it be the end of the media resource instead."

With Safari we observe a too early fired *seeked* event, when seeking as callback to the *loadedmetadata* event. From the statement of the specification that says about the *HAVE_METADATA* ready state that "The API will no longer throw an exception when seeking." [BFL+14, 4.7.10.7 Ready states] we conclude that seeking must be possible with the occurrence of the *loadedmetadata* event.

User agent, version	Operating system	Wrong frames	Average absolute deviation	Maximal absolute deviation
Internet Explorer 11.0.9600.16521	Windows	100%	2	2
Firefox 28.0	Windows	0%	0	0
Chrome 33.0.1750.154	Windows	1.45%	1.8966	3
Safari 7.0 (9537.71)	OS X	0%	0	0

Table 5: The seeking accuracy is tested by measuring any deviation from the expected frame number.

Except for these special cases Firefox and Safari fire the *seeked* event as required. Chrome also performs these tests without problems. Next, we analyze the accuracy of the seeking implementation. We give every user agent enough time to actually finish seeking before we check the frame number. The results can be found in Table 5.

We find that Internet Explorer has a constant deviation from the expected frame by two frames. Looking at this problem in detail we can see that the first frame of the video is displayed three times as long as the others, which leads to the observed offset of two frames. Chrome fails to seek to the correct frame in 1.45% of all cases. When the wrong frame is shown, it is one, two or three frames off in our experiments. This might be close enough for some applications, but might trouble others. Finally, Firefox and Safari both seek to the expected frames without deviations.

Results

The experiments show that the *seeked* event is not very reliable. What is the benefit of a *seeked* event, if we cannot be sure that a new frame is shown when it is fired? Furthermore, we have shown small deviations from the expected frame. These are usually not important, and most users won't even take notice of this issue. However, it prevents the emergence of application that rely on frame-precise seeking.

4 Improvements to HTML5 video

As we are interested in a powerful HTML5 video platform to supersede third party plugins as the leading internet video technology, the portfolio of possible applications should not suffer from a transition to HTML5. Examples for possible media-oriented tasks are:

- video post-processing (edit the pixel data manually, e.g. to change the contrast),
- audio equalization, and
- audio spectrum visualization.

Unfortunately, these applications are currently not possible using pure HTML5. Though the video buffer can be obtained and drawn to a canvas (timer-based), it is not possible to ensure this for every frame, since the frame rate is unknown. The audio data cannot be accessed anyway. We suggest the following improvements to HTML5:

- Querying detailed meta data information (e.g. frame rate, bit rate, etc.)
- Frame-wise stepping, querying frame number
- Audio buffer access and manipulation
- Manipulating pixel data of the video (e.g. introducing a new *newframe* event to control the manipulation on a canvas)

5 Conclusion

HTML5 video is an important step towards the goal of having a unified way of displaying videos on the web without requiring third party plugins. However, when trying to interact with the element via JavaScript and create a user agent independent solution, several issues become apparent. We took a close look at the event system of the browsers and compared their behavior. The comparison between user agents and the specification showed missing events and differences of implementations. As a consequence among other issues, there is no reliable way of handling an underrun.

The analysis of seeking implementations reveals further problems. The *seeked* event that is fired too early in Internet Explorer prohibits web applications, for instance, too seek to a position and take a snapshot of the video. The inaccuracies in seeking in Chrome prevent meaningful frame by frame stepping through the video. In Firefox it is not possible to jump to the end of the video. Those and other bugs force web developers to find browser-dependent workarounds. As this is a contradiction to the paradigm of feature-detection, HTML5 video needs a more consistent application interface in today's browsers. If special applications like web video cutting are planned, we have to recommend (based on today's level of knowledge) the usage of third party plugins as seeking accuracy and completion are in general no trustworthy data.

We are offering user agent tests, test cases and test data to developers to evaluate and improve new implementation. We hope the next versions of the tested browsers stand our tests. With a growing distribution of an unified and standard conform video interface, we will see more advanced online multimedia applications and user experiences.

References

[BFL+14] Robin Berjon, Steve Faulkner, Travis Leithead, Erika Doyle Navara, Edward O'Connor, Silvia Pfeiffer, and Ian Hickson. HTML5. Candidate Recommendation 29 April 2014, W3C, April 2014. http://www.w3.org/TR/html5/.

[Net14] NetApplications. `http://www.netmarketshare.com/report.aspx?qprid=0&qptimeframe=M&qpsp=184&qpcustomd=0`, 2014. Accessed: 2014-06-10.

[Sun08] Sun Microsystems. Sun's Network.com Renders Computer-Animated Movie "Big Buck Bunny". Press Release, June 2008.