m3u-Playlist player in Vanilla Javascript

Dr. Arne Babenhauserheide

<2021-01-16 Sa>

Websites could be an awesome tool for simple radio, if only playlists in video- or audio-tags would **just work**. Here I'm creating a script that when imported turns every media-tag which points to an m3u-file into a player that allows playing the playlist.

Update 2021-03: process tags on browsers that "maybe" support m3u (i.e. mobile browsers).

I created this as a starting point for streaming via Freenet, which turned out to require some more hacks to reach real-time due to different latencies in a fully decentralized and anonymizing environment, but I'm happy that I still managed to keep the complete decentralized streaming-solution below 280 lines.

Back to the version on this site :-)

Here's the script: `m3u-player.js`

Here's an example playlist:
`m3u-player-example-playlist.m3u`

```
# Songs of Arne Babenhauserheide; Lieder im Zeitenwind
https://www.draketo.de/files/2017-10-11-new-horizons-for-science-im-IMK-ASF-KIT-fi
http://www.draketo.de/files/Infinite-Hands-2011-Remastered-by-Sascha.ogg
https://www.draketo.de/files/Auf-einem-Baum-warn-Demon-strant.mp3
https://www.draketo.de/musik/wind-to-thy-wings2.mp3
http://draketo.de/files/Fern%20doch%20nah.mp3
http://www.draketo.de/files/gentoo_for_me-v0.3.ogg
http://www.draketo.de/files/Happy_Birthday_to_GNU.ogg
https://www.draketo.de/files/Pond-erosa-puff.mp3
https://www.draketo.de/files/with-python-from-the-shadows-own-melody.ogg
http://infinite-hands.draketo.de/musik/Infinite-Hands--free-software.mp3
```

```
http://draketo.de/musik/broken-apple-heart_en.mp3
https://www.draketo.de/files/rabenanwaelte-und-abmahnkraehen.mp3
https://www.draketo.de/files/frankfurt-haie-otakus.mp3
https://www.draketo.de/files/seiken-densetsu-3-bardstale_0.ogg
https://www.draketo.de/files/Traeume-fuer-die-Welt-nur-lied.ogg
# evil recursive call for theoretically infinite playback that the parser must lim
https://www.draketo.de/software/m3u-player-example-playlist.m3u
```

Here's how to use it:

```
<script src="m3u-player.js" defer="defer"></script>
<audio src="m3u-player-example-playlist.m3u" controls="controls">
not supported?
</audio>
```

Have fun!
The license is GPLv2 or later.

The script

```
// @license magnet:?xt=urn:btih:cf05388f2679ee054f2beb29a391d25f4e673ac3&dn=gpl-2.
const nodes = document.querySelectorAll("audio,video");
const playlists = {};
const prefetchedTracks = new Map(); // use a map for insertion order, so we can ju
// maximum prefetched blobs that are kept.
const MAX_PREFETCH_KEEP = 10;
// maximum allowed number of entries in a playlist to prevent OOM attacks against
const MAX_PLAYLIST_LENGTH = 1000;
const PLAYLIST_MIME_TYPES = ["audio/x-mpegurl", "audio/mpegurl", "application/vnd.
function stripUrlParameters(link) {
  const url = new URL(link, window.location);
  url.search = "";
  url.hash = "";
  return url.href;
}
function isPlaylist(link) {
  const linkHref = stripUrlParameters(link);
  return linkHref.endsWith(".m3u") || linkHref.endsWith(".m3u8");
```

```
}
function isBlob(link) {
  return new URL(link, window.location).protocol == 'blob';
}
function parsePlaylist(textContent) {
  return textContent.match(/^(?!#)(?!\s).*$/mg)
    .filter(s => s); // filter removes empty strings
}
/**
 * Download the given playlist, parse it, and store the tracks in the
 * global playlists object using the url as key.
 *
 * Runs callback once the playlist downloaded successfully.
 */
function fetchPlaylist(url, onload, onerror) {
  const playlistFetcher = new XMLHttpRequest();
  playlistFetcher.open("GET", url, true);
  playlistFetcher.responseType = "blob"; // to get a mime type
  playlistFetcher.onload = () => {
    if (PLAYLIST_MIME_TYPES.includes(playlistFetcher.response.type)) { // security
      const reader = new FileReader();
      const load = onload; // propagate to inner scope
      reader.addEventListener("loadend", e => {
playlists[url] = parsePlaylist(reader.result);
onload();
      });
      reader.readAsText(playlistFetcher.response);
    } else {
      console.error("playlist must have one of the playlist MIME type '" + PLAYLIS
      onerror();
    }
  };
  playlistFetcher.onerror = onerror;
  playlistFetcher.abort = onerror;
  playlistFetcher.send();
}
function servedPartialDataAndCanRequestAll (xhr) {
  if (xhr.status === 206) {
```

```
    if (xhr.getResponseHeader("content-range").includes("/")) {
      if (!xhr.getResponseHeader("content-range").includes("/*")) {
return true;
      }
    }
  }
  return false;
}
function prefetchTrack(url, onload) {
  if (prefetchedTracks.has(url)) {
    return;
  }
  // first cleanup: kill the oldest entries until we're back at the allowed size
  while (prefetchedTracks.size > MAX_PREFETCH_KEEP) {
    const key = prefetchedTracks.keys().next().value;
    const track = prefetchedTracks.get(key);
    prefetchedTracks.delete(key);
  }
  // first set the prefetched to the url so we will never request twice
  prefetchedTracks.set(url, url);
  // now start replacing it with a blob
  const xhr = new XMLHttpRequest();
  xhr.open("GET", url, true);
  xhr.responseType = "blob";
  xhr.onload = () => {
    if (servedPartialDataAndCanRequestAll(xhr)) {
      const endRange = Number(xhr.getResponseHeader("content-range").split("/")[1]
      const rangeXhr = new XMLHttpRequest();
      rangeXhr.open("GET", url, true);
      rangeXhr.responseType = "blob";
      rangeXhr.setRequestHeader("range", "bytes=0-" + endRange);
      rangeXhr.onload = () => {
prefetchedTracks.set(url, rangeXhr.response);
if (onload) {
  onload();
}
      };
      rangeXhr.send();
```

```
    } else {
      prefetchedTracks.set(url, xhr.response);
      if (onload) {
onload();
      }
    }
  };
  xhr.send();
}
function updateSrc(mediaTag, callback) {
  const playlistUrl = mediaTag.getAttribute("playlist");
  const trackIndex =  mediaTag.getAttribute("track-index");
  // deepcopy playlists to avoid shared mutation
  let playlist = [...playlists[playlistUrl]];
  let trackUrl = playlist[trackIndex];
  // download and splice in playlists as needed
  if (isPlaylist(trackUrl)) {
    if (playlist.length >= MAX_PLAYLIST_LENGTH) {
      // skip playlist if we already have too many tracks
      changeTrack(mediaTag, +1);
    } else {
      // do not use the cached playlist here, though it is tempting: it might genu
      fetchPlaylist(
trackUrl,
() => {
  playlist.splice(trackIndex, 1, ...playlists[trackUrl]);
  playlists[playlistUrl] = playlist;
  updateSrc(mediaTag, callback);
},
() => callback());
    }
  } else {
    let url = prefetchedTracks.has(trackUrl)
? prefetchedTracks.get(trackUrl) instanceof Blob
? URL.createObjectURL(prefetchedTracks.get(trackUrl))
: trackUrl : trackUrl;
    const oldUrl = mediaTag.getAttribute("src");
    // prevent size flickering by setting height before src change
```

```
    const canvas = document.createElement("canvas");
    if (!isNaN(mediaTag.duration)) { // already loaded a valid file so the size sh
      // fix height to the height of the current video. Re-run after setting the s
      mediaTag.height = (mediaTag.clientWidth * mediaTag.videoHeight) / mediaTag.v
      // take screenshot of video and overlay it to mask short-term flicker.
      canvas.width = mediaTag.clientWidth;
      canvas.height = mediaTag.clientHeight;
      const context = canvas.getContext("2d");
      context.scale(mediaTag.clientWidth / mediaTag.videoWidth, mediaTag.clientHei
      context.drawImage(mediaTag, 0, 0);
      canvas.hidden = true;
      mediaTag.parentElement.appendChild(canvas);
      canvas.style.position = "absolute";
      canvas.style.marginLeft = "-" + mediaTag.clientWidth + "px";
      canvas.hidden = false;
    }
    mediaTag.setAttribute("src", url);
    mediaTag.oncanplaythrough = () => {
      if (!isNaN(mediaTag.duration)) { // already loaded a valid file
// fix height to the height of the current video. Re-run after setting the source.
mediaTag.height = (mediaTag.clientWidth * mediaTag.videoHeight) / mediaTag.videoWi
      }
      // remove overlay
      canvas.hidden = true;
      canvas.remove(); // to allow garbage collection
    };
    setTimeout(() => canvas.remove(), 300); // fallback
    // replace the url when done, because a blob from an xhr request
    // is more reliable in the media tag;
    // the normal URL caused jumping prematurely to the next track.
    if (url == trackUrl) {
      prefetchTrack(trackUrl, () => {
if (mediaTag.paused) {
  if (url == mediaTag.getAttribute("src")) {
    if (mediaTag.currentTime === 0) {
      mediaTag.setAttribute("src", URL.createObjectURL(
prefetchedTracks.get(url)));
    }
```

```
  }
}
      });
    }
    // allow releasing memory
    if (isBlob(oldUrl)) {
      URL.revokeObjectURL(oldUrl);
    }
    // update title
    mediaTag.parentElement.querySelector(".m3u-player--title").title = trackUrl;
    mediaTag.parentElement.querySelector(".m3u-player--title").textContent = track
    // start prefetching the next three tracks.
    for (const i of [1, 2, 3]) {
      if (playlist.length > Number(trackIndex) + i) {
prefetchTrack(playlist[Number(trackIndex) + i]);
      }
    }
    callback();
  }
}
function changeTrack(mediaTag, diff) {
  const currentTrackIndex = Number(mediaTag.getAttribute("track-index"));
  const nextTrackIndex = currentTrackIndex + diff;
  const tracks = playlists[mediaTag.getAttribute("playlist")];
  if (nextTrackIndex >= 0) { // do not collapse the if clauses with double-and, th
    if (tracks.length > nextTrackIndex) {
    mediaTag.setAttribute("track-index", nextTrackIndex);
      updateSrc(mediaTag, () => mediaTag.play());
    }
  }
}

/**
 * Turn a media tag into playlist player.
 */
function initPlayer(mediaTag) {
  mediaTag.setAttribute("playlist", mediaTag.getAttribute("src"));
  mediaTag.setAttribute("track-index", 0);
```

```
const url = mediaTag.getAttribute("playlist");
const wrapper = mediaTag.parentElement.insertBefore(document.createElement("div'
const controls = document.createElement("div");
const left = document.createElement("span");
const title = document.createElement("span");
const right = document.createElement("span");
controls.appendChild(left);
controls.appendChild(title);
controls.appendChild(right);
left.classList.add("m3u-player--left");
right.classList.add("m3u-player--right");
title.classList.add("m3u-player--title");
title.style.overflow = "hidden";
title.style.textOverflow = "ellipsis";
title.style.whiteSpace = "nowrap";
title.style.opacity = "0.3";
title.style.direction = "rtl"; // for truncation on the left
title.style.paddingLeft = "0.5em";
title.style.paddingRight = "0.5em";
controls.style.display = "flex";
controls.style.justifyContent = "space-between";
const styleTag = document.createElement("style");
styleTag.innerHTML = ".m3u-player--left:hover, .m3u-player--right:hover {color:
wrapper.appendChild(styleTag);
wrapper.appendChild(controls);
controls.style.width = mediaTag.getBoundingClientRect().width.toString() + "px";
// appending the media tag to the wrapper removes it from the outer scope but ke
wrapper.appendChild(mediaTag);
left.innerHTML = "&lt;"; // not textContent, because we MUST escape
 // the tag here and textContent shows the
 // escaped version
left.onclick = () => changeTrack(mediaTag, -1);
right.innerHTML = "&gt;";
right.onclick = () => changeTrack(mediaTag, +1);
fetchPlaylist(
  url,
  () => {
    updateSrc(mediaTag, () => null);
```

```
      mediaTag.addEventListener("ended", event => {
if (mediaTag.currentTime >= mediaTag.duration) {
  changeTrack(mediaTag, +1);
}
      });
    },
    () => null);
  // keep the controls aligned to the media tag
  mediaTag.resizeObserver = new ResizeObserver(entries => {
    controls.style.width = entries[0].contentRect.width.toString() + "px";
  });
  mediaTag.resizeObserver.observe(mediaTag);
}
function processTag(mediaTag) {
  const canPlayClaim = mediaTag.canPlayType('audio/x-mpegurl');
  let supportsPlaylists = !!canPlayClaim;
  if (canPlayClaim == 'maybe') { // yes, seriously: specced as you only know when
    supportsPlaylists = false;
  }
  if (!supportsPlaylists) {
    if (isPlaylist(mediaTag.getAttribute("src"))) {
      initPlayer(mediaTag);
    }
  }
}
document.addEventListener('DOMContentLoaded', () => {
  const nodes = document.querySelectorAll("audio,video");
  nodes.forEach(processTag);
});
// @license-end
```