Customize Media Notifications and Handle Playlists

By François Beaufort
Dives into Chromium source code

With the brand new Media Session API, you can now **customize media notifications** by providing metadata for the media your web app is playing. It also allows you to **handle media related events** such as seeking or track changing which may come from notifications or media keys. Excited? Try out the official Media Session samples.

The Media Session API is supported in Chrome 57 (beta in February 2017, stable in March 2017).

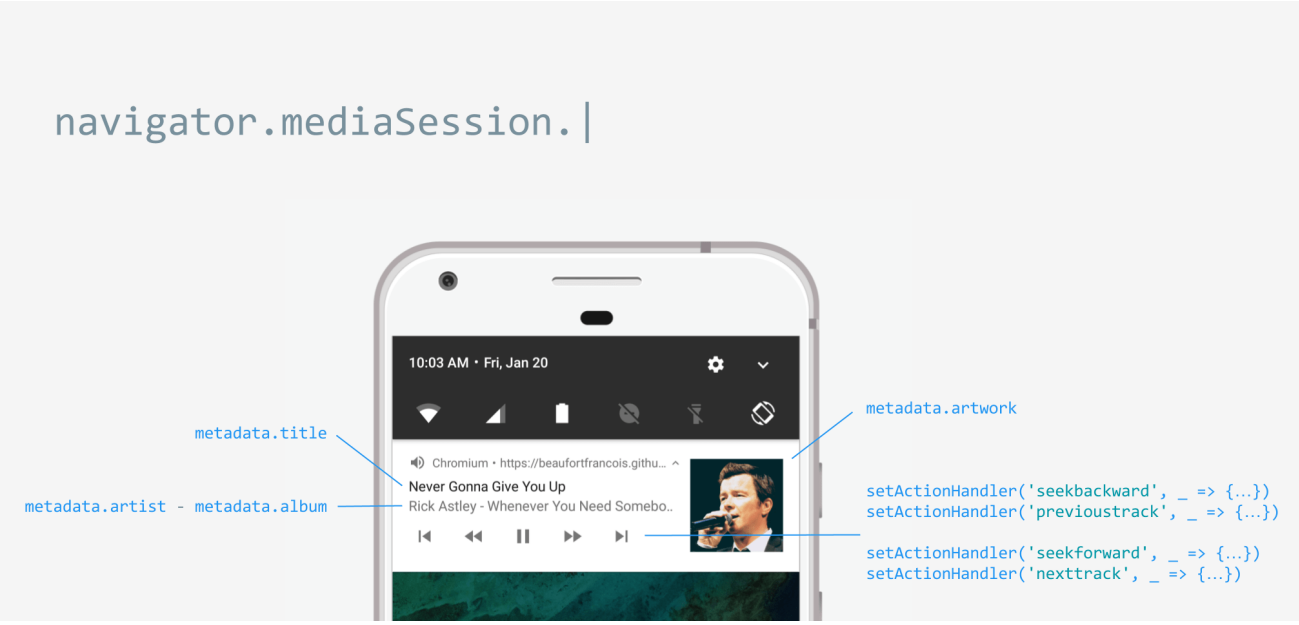


Photo by Michael Alø-Nielsen / CC BY 2.0

Gimme what I want

You already know about the Media Session API and are simply coming back to copy and paste with no shame some boilerplate code? So here it is.

```
if ('mediaSession' in navigator) {

  navigator.mediaSession.metadata = new MediaMetadata({
```

```
    title: 'Never Gonna Give You Up',
    artist: 'Rick Astley',
    album: 'Whenever You Need Somebody',
    artwork: [
      { src: 'https://dummyimage.com/96x96',   sizes: '96x96',   type: 'image/png
      { src: 'https://dummyimage.com/128x128', sizes: '128x128', type: 'image/png
      { src: 'https://dummyimage.com/192x192', sizes: '192x192', type: 'image/png
      { src: 'https://dummyimage.com/256x256', sizes: '256x256', type: 'image/png
      { src: 'https://dummyimage.com/384x384', sizes: '384x384', type: 'image/png
      { src: 'https://dummyimage.com/512x512', sizes: '512x512', type: 'image/png
    ]
  });

  navigator.mediaSession.setActionHandler('play', function() {});
  navigator.mediaSession.setActionHandler('pause', function() {});
  navigator.mediaSession.setActionHandler('seekbackward', function() {});
  navigator.mediaSession.setActionHandler('seekforward', function() {});
  navigator.mediaSession.setActionHandler('previoustrack', function() {});
  navigator.mediaSession.setActionHandler('nexttrack', function() {});
}
```

Get into the code

Let's play 🎷

Add a simple **`<audio>`** element to your web page and assign several media sources so that the browser can choose which one works best.

```
<audio controls>
  <source src="audio.mp3" type="audio/mp3"/>
  <source src="audio.ogg" type="audio/ogg"/>
</audio>
```

Note: I could have used a **`<video>`** element instead to showcase the Media Session API.

As you may know, **`autoplay`** is disabled for audio elements on Chrome for Android which means we have to use the **`play()`** method of the audio element. This method must be triggered by a user gesture such as a touch or a mouse click. That means listening to **`pointerup`**, **`click`**, and **`touchend`** events. In other words, the user must click a button before your web app can actually make noise.

```
playButton.addEventListener('pointerup', function(event) {
  let audio = document.querySelector('audio');

  // User interacted with the page. Let's play audio...
  audio.play()
  .then(_ => { /* Set up media session... */ })
  .catch(error => { console.log(error) });
});
```

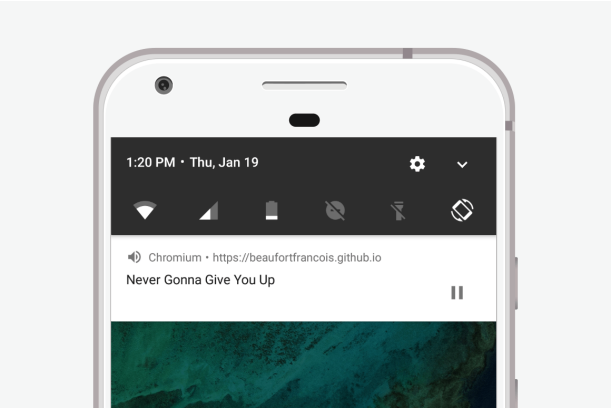
Note: If the `<audio>` element has the `controls` attribute, you can simply set up the media session in the audio `play` event listener instead which occurs when user taps the "play" audio control.

If you don't want to play audio right after the first interaction, I recommend you use the `load()` method of the audio element. This is one way for the browser to keep track of whether the user interacted with the element. Note that it may also help smooth the playback because the content will already be loaded.

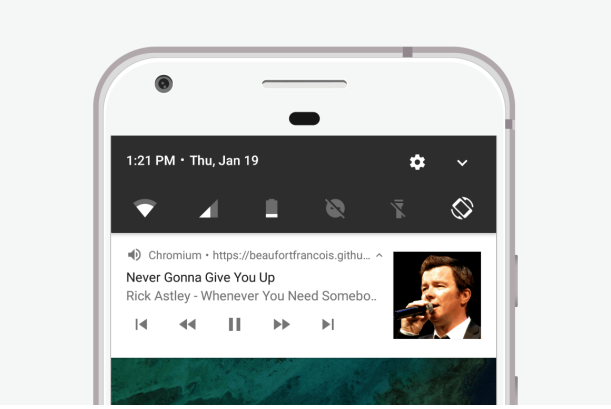
```
let audio = document.querySelector('audio');

welcomeButton.addEventListener('pointerup', function(event) {
  // User interacted with the page. Let's load audio...
  audio.load()
  .then(_ => { /* Show play button for instance... */ })
  .catch(error => { console.log(error) });
});

// Later...
playButton.addEventListener('pointerup', function(event) {
  audio.play()
  .then(_ => { /* Set up media session... */ })
  .catch(error => { console.log(error) });
});
```

Customize the notification

When your web app is playing audio, you can already see a media notification sitting in the notification tray. On Android, Chrome does its best to show appropriate information by using the document's title and the largest icon image it can find.



Without media session



With media session

Set metadata

Let's see how to customize this media notification by setting some media session metadata such as the title, artist, album name, and artwork with the Media Session API.

```
// When audio starts playing...
if ('mediaSession' in navigator) {

  navigator.mediaSession.metadata = new MediaMetadata({
    title: 'Never Gonna Give You Up',
    artist: 'Rick Astley',
    album: 'Whenever You Need Somebody',
    artwork: [
      { src: 'https://dummyimage.com/96x96',   sizes: '96x96',   type: 'image/png
      { src: 'https://dummyimage.com/128x128', sizes: '128x128', type: 'image/png
      { src: 'https://dummyimage.com/192x192', sizes: '192x192', type: 'image/png
      { src: 'https://dummyimage.com/256x256', sizes: '256x256', type: 'image/png
      { src: 'https://dummyimage.com/384x384', sizes: '384x384', type: 'image/png
      { src: 'https://dummyimage.com/512x512', sizes: '512x512', type: 'image/png
    ]
```

```
  });
}
```

Once playback is done, you don't have to "release" the media session as the notification will automatically disappear. Keep in mind that current **`navigator.mediaSession.metadata`** will be used when any playback starts. This is why you need to update it to make sure you're always showing relevant information in the media notification.

Previous track / next track

If your web app provides a playlist, you may want to allow the user to navigate through your playlist directly from the media notification with some "Previous Track" and "Next Track" icons.

```
let audio = document.createElement('audio');

let playlist = ['audio1.mp3', 'audio2.mp3', 'audio3.mp3'];
let index = 0;

navigator.mediaSession.setActionHandler('previoustrack', function() {
  // User clicked "Previous Track" media notification icon.
  index = (index - 1 + playlist.length) % playlist.length;
  playAudio();
});

navigator.mediaSession.setActionHandler('nexttrack', function() {
  // User clicked "Next Track" media notification icon.
  index = (index + 1) % playlist.length;
  playAudio();
});

function playAudio() {
  audio.src = playlist[index];
  audio.play()
  .then(_ => { /* Set up media session... */ })
  .catch(error => { console.log(error); });
}

playButton.addEventListener('pointerup', function(event) {
  playAudio();
});
```

Note that media action handlers will persist. This is very similar to the event listener pattern except that handling an event means that the browser stops doing any default behaviour

and uses this as a signal that your web app supports the media action. Hence, media action controls won't be shown unless you set the proper action handler.

By the way, unsetting a media action handler is as easy as assigning it to `null`.

Seek backward / seek forward

The Media Session API allows you to show "Seek Backward" and "Seek Forward" media notification icons if you want to control the amount of time skipped.

```
let skipTime = 10; // Time to skip in seconds

navigator.mediaSession.setActionHandler('seekbackward', function() {
  // User clicked "Seek Backward" media notification icon.
  audio.currentTime = Math.max(audio.currentTime - skipTime, 0);
});

navigator.mediaSession.setActionHandler('seekforward', function() {
  // User clicked "Seek Forward" media notification icon.
  audio.currentTime = Math.min(audio.currentTime + skipTime, audio.duration);
});
```

Play / pause

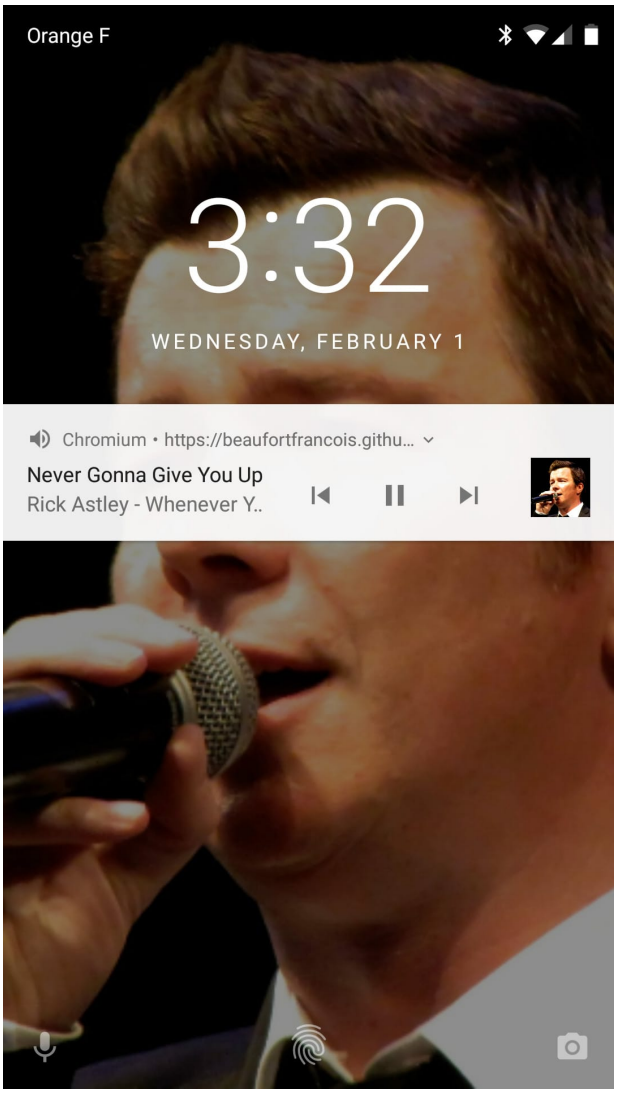
The "Play/Pause" icon is always shown in the media notification and the related events are handled automatically by the browser. If for some reason the default behaviour doesn't work out, you can still handle "Play" and "Pause" media events.

```
navigator.mediaSession.setActionHandler('play', function() {
  // User clicked "Play" media notification icon.
  // Do something more than just playing current audio...
});

navigator.mediaSession.setActionHandler('pause', function() {
  // User clicked "Pause" media notification icon.
  // Do something more than just pausing current audio...
});
```

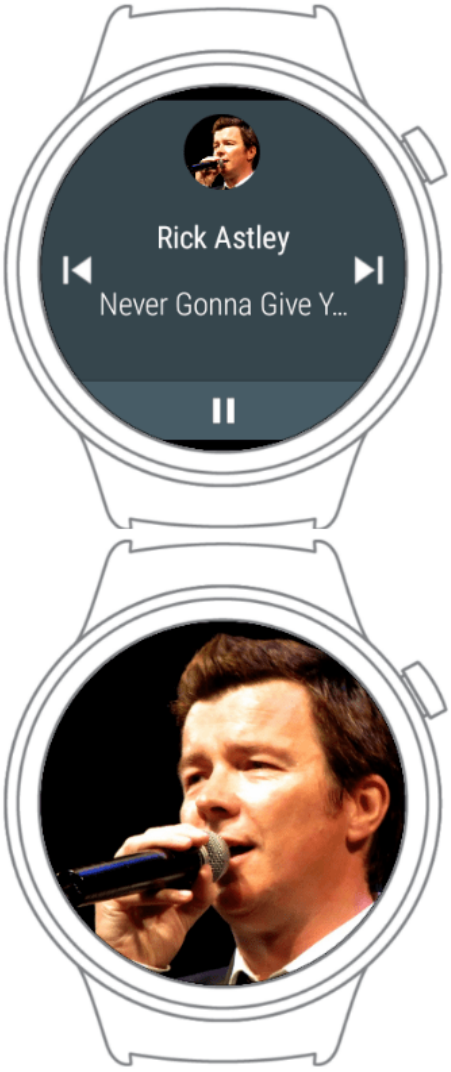
Note: The browser may consider that the web app is not playing media when files are seeking or loading. You can override this behaviour by setting `navigator.mediaSession.playbackState` to `"playing"` or `"paused"`. This comes in handy when you want to make sure your web app UI stays in sync with the media notification controls.

Notifications everywhere

The cool thing about the Media Session API is that the notification tray is not the only place where media metadata and controls are visible. The media notification is synced automagically to any paired wearable device. And it also shows up on lock screens.



Lock Screen - Photo by Michael Alø-Nielsen / CC BY 2.0



Wear Notification

Make it play nice offline

I know what you're thinking now... *service worker to the rescue!*

True but first and foremost, you want to make sure **all items in this checklist are checked**:

- All media and artwork files are served with the appropriate `Cache-Control` HTTP header. This will allow the browser to cache and reuse previously fetched resources. See the Caching checklist.
- Make sure all media and artwork files are served with the `Allow-Control-Allow-Origin: *` HTTP header. This will allow third-party web apps to fetch and consume HTTP responses from your web server.

The service worker caching strategy

Regarding media files, I recommend a simple "Cache, falling back to network" strategy as illustrated by Jake Archibald.

For artwork though, I'd be a little bit more specific and choose the approach below:

- **If** artwork is already in the cache, serve it from the cache
- **Else** fetch artwork from the network
    - **If** fetch is successful, add network artwork to the cache and serve it
    - **Else** serve the fallback artwork from the cache

That way, media notifications will always have a nice artwork icon even when browser can't fetch them. Here's how you could implement this:

```
const FALLBACK_ARTWORK_URL = 'fallbackArtwork.png';

addEventListener('install', event => {
  self.skipWaiting();
  event.waitUntil(initArtworkCache());
});

function initArtworkCache() {
  caches.open('artwork-cache-v1')
  .then(cache => cache.add(FALLBACK_ARTWORK_URL));
}

addEventListener('fetch', event => {
  if (/artwork-[0-9]+\.png$/.test(event.request.url)) {
    event.respondWith(handleFetchArtwork(event.request));
  }
});

function handleFetchArtwork(request) {
  // Return cache request if it's in the cache already, otherwise fetch
  // network artwork.
  return getCacheArtwork(request)
  .then(cacheResponse => cacheResponse || getNetworkArtwork(request));
}

function getCacheArtwork(request) {
  return caches.open('artwork-cache-v1')
  .then(cache => cache.match(request));
}
```

```
function getNetworkArtwork(request) {
  // Fetch network artwork.
  return fetch(request)
  .then(networkResponse => {
    if (networkResponse.status !== 200) {
      return Promise.reject('Network artwork response is not valid');
    }
    // Add artwork to the cache for later use and return network response.
    addArtworkToCache(request, networkResponse.clone())
    return networkResponse;
  })
  .catch(error => {
    // Return cached fallback artwork.
    return getCacheArtwork(new Request(FALLBACK_ARTWORK_URL))
  });
}

function addArtworkToCache(request, response) {
  return caches.open('artwork-cache-v1')
  .then(cache => cache.put(request, response));
}
```

Caution: If you want your service worker to be able to intercept artwork network requests on the very first page load, you may want to call `clients.claim()` within your service worker once it's activated.

Let user control cache

As the user consumes content from your web app, media and artwork files may take a lot of space on their device. It is **your responsibility to show how much cache is used and give users the ability to clear it**. Thankfully for us, doing so is pretty easy with the Cache API.

```
// Here's how I'd compute how much cache is used by artwork files...
caches.open('artwork-cache-v1')
.then(cache => cache.matchAll())
.then(responses => {
  let cacheSize = 0;
  let blobQueue = Promise.resolve();

   responses.forEach(response => {
    let responseSize = response.headers.get('content-length');
    if (responseSize) {
      // Use content-length HTTP header when possible.
      cacheSize += Number(responseSize);
    } else {
      // Otherwise, use the uncompressed blob size.
```

```
      blobQueue = blobQueue.then(_ => response.blob())
          .then(blob => { cacheSize += blob.size; blob.close(); });
    }
  });

  return blobQueue.then(_ => {
    console.log('Artwork cache is about ' + cacheSize + ' Bytes.');
  });
})
.catch(error => { console.log(error); });

// And here's how to delete some artwork files...
const artworkFilesToDelete = ['artwork1.png', 'artwork2.png', 'artwork3.png'];

caches.open('artwork-cache-v1')
.then(cache => Promise.all(artworkFilesToDelete.map(artwork => cache.delete(artwo
.catch(error => { console.log(error); });
```

Implementation notes

- Chrome for Android requests "full" audio focus to show media notifications only when the media file duration is at least 5 seconds.
- Notification artwork support blob URLs and data URLs.
- If no artwork is defined and there is an icon image at a desirable size, media notifications will use it.
- Notification artwork size in Chrome for Android is **512x512**. For low-end devices, it is **256x256**.
- Dismiss media notifications with **`audio.src = ''`**.
- As the Web Audio API doesn't request Android Audio Focus for historical reasons, the only way to make it work with the Media Session API is to hook up an **`<audio>`** element as the input source to the Web Audio API. Hopefully, the proposed Web AudioFocus API will improve the situation in the near future.
- Media Mession calls will affect media notifications only if they come from the same frame as the media resource. See the snippet below.

```
<iframe id="iframe">
  <audio>...</audio>
</iframe>
<script>
  iframe.contentWindow.navigator.mediaSession.metadata = new MediaMetadata({
    title: 'Never Gonna Give You Up',
```

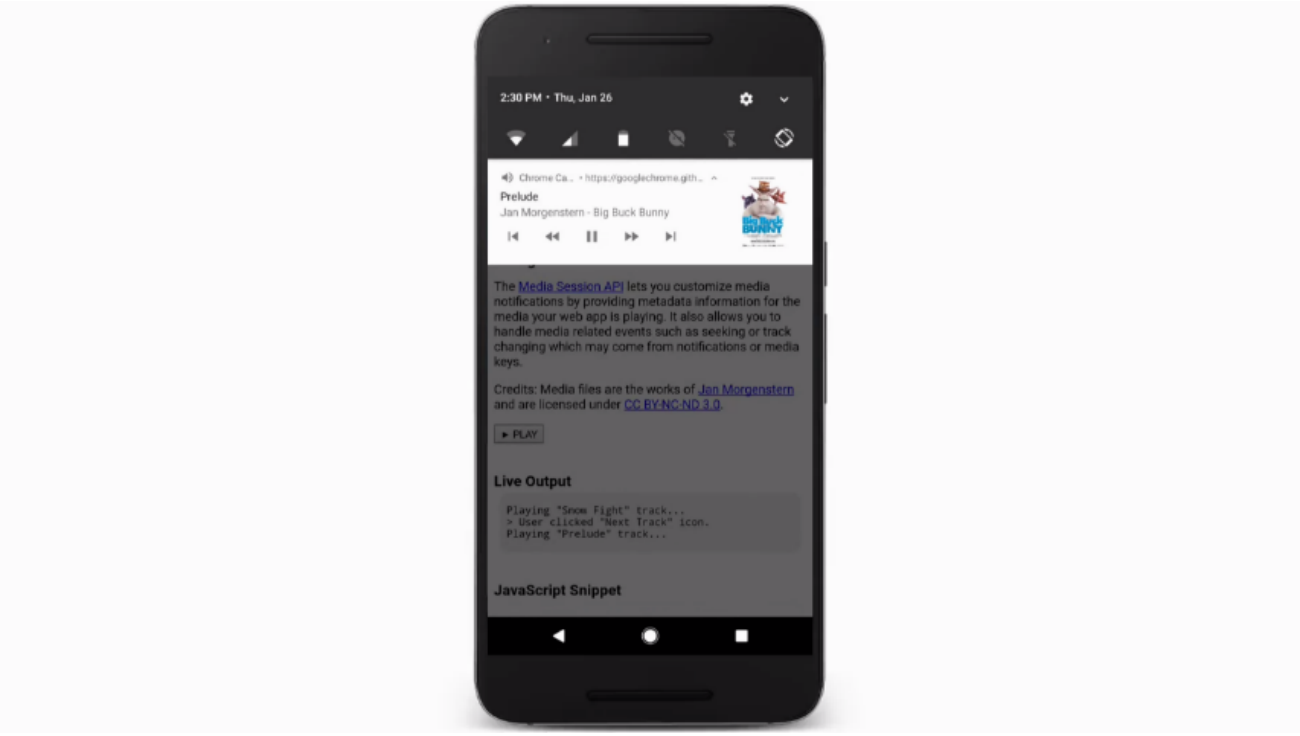
```
    ...
  });
</script>
```

Support

At the time of writing, Chrome for Android is the only platform that supports the Media Session API. More up-to-date information on browser implementation status can be found on Chrome Platform Status.

Samples & demos

Check out our official Chrome Media Session samples featuring Blender Foundation and Jan Morgenstern's work.



Resources

Media Session Spec: wicg.github.io/mediasession

Spec Issues: github.com/WICG/mediasession/issues

Chrome Bugs: crbug.com

Last updated July 2, 2018.