# Audio/Video Updates in Chrome 63/64

**By** François Beaufort
Dives into Chromium source code

- Web developers can now predict whether playback will be smooth and power efficient.
- Chrome now supports HDR video playback on Windows 10.
- Offline playback with persistent licenses are now supported on Windows and Mac.
- The default preload value for `<video>` and `<audio>` elements is now "metadata".
- An error is now thrown when media playbackRate is unsupported.
- Chrome now pauses all background video-only media.
- Audio is not muted anymore for extreme playbackRate.

## Media Capabilities - Decoding Info API

Today, web developers rely on `isTypeSupported()` or `canPlayType()` to vaguely know if some media can be decoded or not. The real question though should be: “How well it would perform on this device?”

This is exactly one of the things the proposed Media Capabilities wants to solve: An API to query the browser about the decoding abilities of the device based on information such as the codecs, profile, resolution, bitrates, etc. It would expose information such as whether the playback should be smooth and power efficient based on previous playback statistics recorded by the browser.

In a nutshell, here’s how the Decoding Info API works for now. Check out the official sample.

```
const mediaConfig = {
  type: 'media-source', // or 'file'
  audio: {
    contentType: 'audio/webm; codecs=opus',
    channels: '2', // audio channels used by the track
    bitrate: 132266, // number of bits used to encode a second of audio
    samplerate: 48000 // number of samples of audio carried per second
  },
  video: {
    contentType: 'video/webm; codecs="vp09.00.10.08"',
    width: 1920,
    height: 1080,
```

```
    bitrate: 2646242, // number of bits used to encode a second of video
    framerate: '25' // number of frames used in one second
  }
};

navigator.mediaCapabilities.decodingInfo(mediaConfig).then(result => {
  console.log('This configuration is' +
      (result.supported ? '' : ' NOT') + ' supported,' +
      (result.smooth ? '' : ' NOT') + ' smooth and' +
      (result.powerEfficient ? '' : ' NOT') + ' power efficient.');
});
```

You can try different media configurations until you find the best one (`smooth` and `powerEfficient`) and use it to play the appropriate media stream. By the way, Chrome's current implementation is based on previously recorded playback information. It defines `smooth` as true when the percentage of dropped frames is less than 10% while `powerEfficient` is true when more than 50% of frames are decoded by the hardware. Small frames are always considered power efficient.

**Note:** The result returned from `navigator.mediaCapabilities.decodingInfo` will always be reported as smooth and power-efficient if the media configuration is supported and playback stats have not been recorded yet by the browser.

I recommend using a snippet similar to the one below to detect availability and fallback to your current implementation for browsers that don't support this API.

```
function isMediaConfigSupported(mediaConfig) {

  const promise = new Promise((resolve, reject) => {
    if (!('mediaCapabilities' in navigator)) {
      return reject('MediaCapabilities API not available');
    }
    if (!('decodingInfo' in navigator.mediaCapabilities)) {
      return reject('Decoding Info not available');
    }
    return resolve(navigator.mediaCapabilities.decodingInfo(mediaConfig));
  });

  return promise.catch(_ => {
    let fallbackResult = {
      supported: false,
      smooth: false, // always false
      powerEfficient: false // always false
    };
    if ('video' in mediaConfig) {
      fallbackResult.supported = MediaSource.isTypeSupported(mediaConfig.video.conte
      if (!fallbackResult.supported) {
```

```
        return fallbackResult;
      }
    }
    if ('audio' in mediaConfig) {
      fallbackResult.supported = MediaSource.isTypeSupported(mediaConfig.audio.conte
    }
    return fallbackResult;
  });
}
```

**Caution:** The snippet above must use `canPlayType()` instead of `isTypeSupported()` if the media configuration type is `"file"`.

## Available for Origin Trials

In order to get as much feedback as possible from developers using the Decoding Info API (part of Media Capabilities) in the field, we've previously added this feature in Chrome 64 as an origin trial.

The trial has successfully ended in April 2018.

**Note:** The Decoding Info API is now enabled by default in Chrome 66.

Intent to Experiment | Intent to Ship | Chromestatus Tracker | Chromium Bug

## HDR video playback on Windows 10

High Dynamic Range (HDR) videos have higher contrast, revealing precise, detailed shadows and stunning highlights with more clarity than ever. Moreover support for wide color gamut means colors are more vibrant.

**Figure 1.** Simulated SDR vs HDR comparison (seeing true HDR requires an HDR display)

As VP9 Profile 2 10-bit playback is now supported in Chrome for Windows 10 Fall Creator Update, Chrome additionally supports HDR video playback when Windows 10 is in HDR mode. On a technical note, Chrome 64 now supports the scRGB color profile which in turn allows media to play back in HDR.

You can give it a try by watching The World in HDR in 4K (ULTRA HD) on YouTube and check that it plays HDR by looking at the YouTube player quality setting.

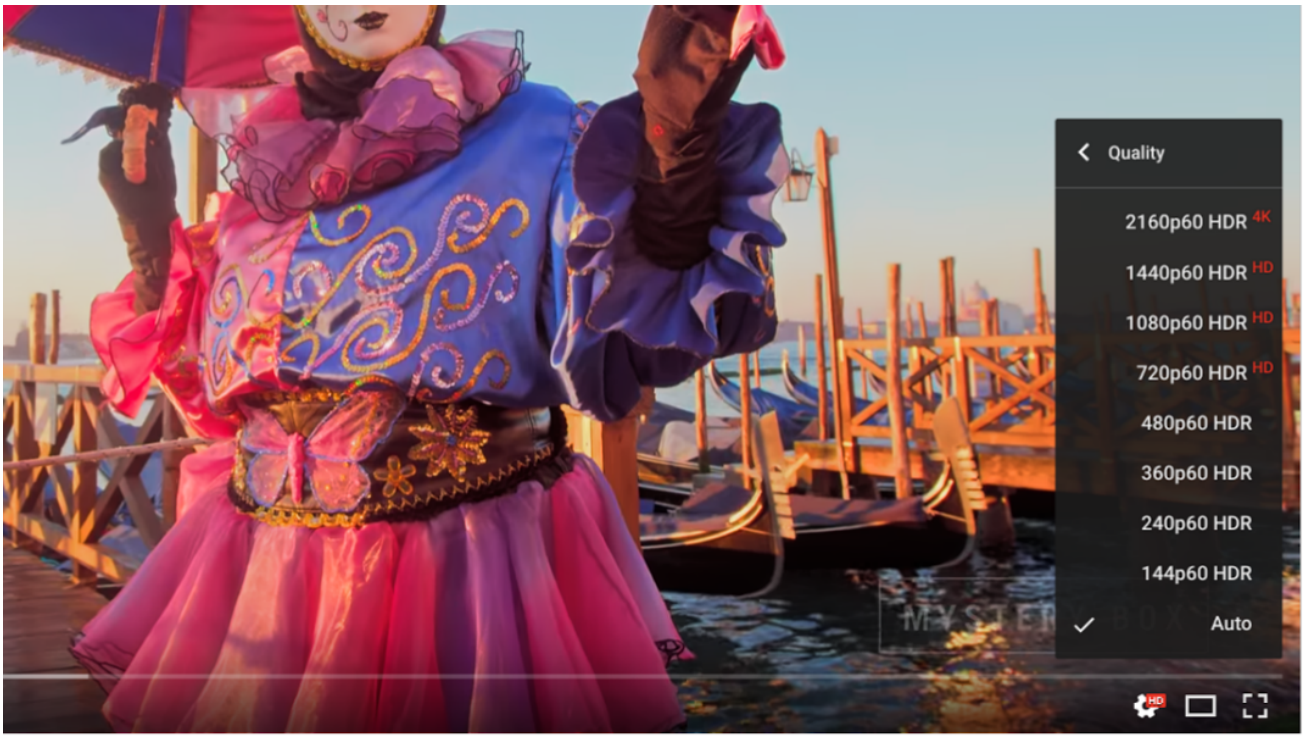


**Figure 2.** YouTube player quality setting featuring HDR

All you need for now is Windows 10 Fall Creator Update, an HDR-compatible graphics card and display (e.g. NVIDIA 10-series card, LG HDR TV or monitor), and turn on HDR mode in Windows display settings.

Web developers can detect the approximate color gamut supported by the output device with the recent color-gamut media query and the number of bits used to display a color on the screen with screen.colorDepth. Here's one way of using those to detect if VP9 HDR is supported for instance:

```
// Detect if display is in HDR mode and if browser supports VP9 HDR.
function canPlayVp9Hdr() {

  // TODO: Adjust VP9 codec string based on your video encoding properties.
  return (window.matchMedia('(color-gamut: p3)').matches &&
      screen.colorDepth >= 48 &&
      MediaSource.isTypeSupported('video/webm; codecs="vp09.02.10.10.01.09.16.09.01"
}
```

The VP9 codec string with Profile 2 passed to `isTypeSupported()` in the example above needs to be updated based on your video encoding properties.

For info, here are your current configuration's results inline:

✕ Screen supports approximately the gamut specified by the DCI P3 Color Space or more.

✕ Color depth is 48 bytes or more.

✔ Browser supports VP9 Profile 2, Level 1, 10-bit YUV content.

Note that it is not possible yet to define HDR colors in CSS, canvas, images and protected content. The Chrome team is working on it. Stay tuned!

## Persistent licenses for Windows and Mac

Persistent license in Encrypted Media Extensions (EME) means the license can be persisted on the device so that applications can load the license into memory without sending another license request to the server. This is how offline playback is supported in EME.

Until now, Chrome OS and Android were the only platforms to support persistent licenses. It is not true anymore. Playing protected content through EME while the device is offline is now possible in Chrome 64 on Windows and Mac as well.

```
const config = [{
  sessionTypes: ['persistent-license'],
  videoCapabilities: [{
    contentType: 'video/webm; codecs="vp09.00.10.08"',
    robustness: 'SW_SECURE_DECODE' // Widevine L3
  }]
}];

navigator.requestMediaKeySystemAccess('com.widevine.alpha', config)
.then(access => {
  // User will be able to watch encrypted content while being offline when
  // license is stored locally on device and loaded later.
})
.catch(error => {
  // Persistent licenses are not supported on this platform yet.
});
```

You can try persistent licenses yourself by checking out the Sample Media PWA and following these steps:

1. Go to https://biograf-155113.appspot.com/ttt/episode-2/

2. Click "Make available offline" and wait for the video to be downloaded.
3. Turn off your internet connection.
4. Click the "Play" button and enjoy the video!

## Media preload defaults to "metadata"

Matching other browsers' implementations, Chrome desktop now sets the default preload value for `<video>` and `<audio>` elements to "metadata" in order to reduce bandwidth and resource usage. This new behaviour only applies in Chrome 64 to cases where no preload value is set. Note that the preload attribute's hint is discarded when a `MediaSource` is attached to the media element as the web site handles its own preload.

In other words, `<video>` preload value is now "metadata" while `<video preload="auto">` preload value stays "auto". Give a try to the official sample.

Intent to Ship | Chromestatus Tracker | Chromium Bug

## Unsupported playbackRate raises an exception

Following an HTML specification change, when media elements' `playbackRate` is set to a value not supported by Chrome (e.g. a negative value), a "NotSupportedError" DOMException is thrown in Chrome 63.

```
const audio = document.querySelector('audio');
try {
  audio.playbackRate = -1;
} catch(error) {
  console.log(error.message); // Failed to set the playbackRate property
}
```

By the way, Chrome's current implementation raises this exception when `playbackRate` is either negative, less than 0.0625, or more than 16. Give a try to the official sample to see this in action.

Intent to Ship | Chromestatus Tracker | Chromium Bug

## Background video track optimizations

The chrome team is always trying to find new ways to improve battery life and Chrome 63 was no exception.

If the video doesn't contain any audio tracks, the video will be automatically paused when played in the background (e.g., in a non-visible tab) in Chrome desktop (Windows, Mac, Linux, and Chrome OS). This is a follow-up from a similar change that was only applying to MSE videos in Chrome 62.

Chromium Bug

## Remove muting for extreme playbackRates

Before Chrome 64, sound was muted when `playbackRate` was below 0.5 or above 4 as the quality degraded significantly. As Chrome has switched to a Waveform-Similarity-Overlap-Add (WSOLA) approach for quality degrading, sound doesn't need to be muted anymore. It means you can play sound crazy slow and crazy fast now.

Chromium Bug

---



*Last updated July 2, 2018.*