

MDN Plus now available in [your](#) country! Support MDN [and](#) make it your own. [Learn more](#) 🌟

https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API/Using_the_MediaStream_Recording_API

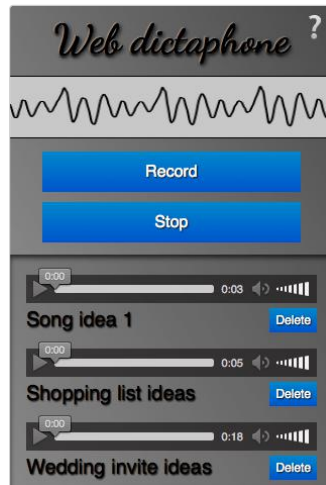
<https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia>
[MediaDevices.getUserMedia\(\)](#)

Using the MediaStream Recording API

The [MediaStream Recording API](#) makes it easy to record audio and/or video streams. When used with [navigator.mediaDevices.getUserMedia\(\)](#), it provides an easy way to record from the user's input devices and instantly use the result in web apps.

Both audio and video may be recorded, separately or together. This article aims to provide a basic guide on how to use the MediaRecorder interface, which provides this API.

A sample application: Web Dictaphone



To demonstrate basic usage of the MediaRecorder API, we have built a web-based dictaphone. It allows you to record snippets of audio and then play them back. It even gives you a visualization of your device's sound input, using the Web Audio API. We'll concentrate on the recording and playback functionality for this article.

You can see this [demo running live](#), or [grab the source code](#) on GitHub.

CSS goodies

The HTML is pretty simple in this app, so we won't go through it here; there are a couple of slightly more interesting bits of CSS worth mentioning, however, so we'll discuss them below. If you are not interested in CSS and want to get straight to the JavaScript, skip to the [Basic app setup](#) section.

Keeping the interface constrained to the viewport, regardless of device height, with calc()

The [calc\(\)](#) function is one of those useful little utility features that's cropped up in CSS that doesn't look like much initially, but soon starts to make you think "Wow, why didn't we have this before? Why was CSS2 layout so awkward?" It allows you to do a calculation to determine the computed value of a CSS unit, mixing different units in the process.

For example, in Web Dictaphone we have three main UI areas, stacked vertically. We wanted to give the first two (the header and the controls) fixed heights:

```
header {
  height: 70px;
}

.main-controls {
  padding-bottom: 0.7rem;
  height: 170px;
}
```

However, we wanted to make the third area (which contains the recorded samples you can play back) take up whatever space is left, regardless of the device height. Flexbox could be the answer here, but it's a bit overkill for such a simple layout. Instead, the problem was solved by making the third container's height equal to 100% of the parent height, minus the heights and padding of the other two:

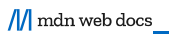
```
.sound-clips {
  box-shadow: inset 0 3px 4px rgba(0, 0, 0, 0.7);
  background-color: rgba(0, 0, 0, 0.1);
  height: calc(100% - 240px - 0.7rem);
  overflow: scroll;
}
```

Note: `calc()` has good support across modern browsers too, even going back to Internet Explorer 9.

Checkbox hack for showing/hiding

This is fairly well documented already, but we thought we'd give a mention to the checkbox hack, which abuses the fact that you can click on the `<label>` of a checkbox to toggle it checked/unchecked. In Web Dictaphone this powers the Information screen, which is shown/hidden by clicking the question mark icon in the top right-hand corner. First of all, we style the `<label>` how we want it, making sure that it has enough z-index to always sit above the other elements and therefore be focusable/clickable:

```
label {
  font-family: "NotoColorEmoji";
  font-size: 3rem;
  position: absolute;
  top: 2px;
  right: 3px;
}
```



Then we hide the actual checkbox, because we don't want it cluttering up our UI:

```
input[type="checkbox"] {
  position: absolute;
  top: -100px;
}
```

Next, we style the Information screen (wrapped in an `<aside>` element) how we want it, give it fixed position so that it doesn't appear in the layout flow and affect the main UI, transform it to the position we want it to sit in by default, and give it a transition for smooth showing/hiding:

```
aside {
  position: fixed;
  top: 0;
  left: 0;
  text-shadow: 1px 1px 1px black;
  width: 100%;
  height: 100%;
  transform: translateX(100%);
  transition: 0.6s all;
  background-color: #999;
  background-image: linear-gradient(
    to top right,
    rgba(0, 0, 0, 0),
    rgba(0, 0, 0, 0.5)
  );
}
```

Last, we write a rule to say that when the checkbox is checked (when we click/focus the label), the adjacent `<aside>` element will have its horizontal translation value changed and transition smoothly into view:

```
input[type="checkbox"]:checked ~ aside {
  transform: translateX(0);
}
```

Basic app setup

To grab the media stream we want to capture, we use `getUserMedia()`. We then use the `MediaRecorder` API to record the stream, and output each recorded snippet into the source of a generated `<audio>` element so it can be played back.

We'll declare some variables for the record and stop buttons, and the `<article>` that will contain the generated audio players:

```
const record = document.querySelector(".record");
const stop = document.querySelector(".stop");
const soundClips = document.querySelector(".sound-clips");
```

Finally for this section, we set up the basic `getUserMedia` structure:

```
if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
  console.log("getUserMedia supported.");
  navigator.mediaDevices
    .getUserMedia(
      // constraints - only audio needed for this app
    )
```

```

    {
      audio: true,
    }
  )

  // Success callback
  .then((stream) => {})

  // Error callback
  .catch((err) => {
    console.error(`The following getUserMedia error occurred: ${err}`);
  });
} else {
  console.log("getUserMedia not supported on your browser!");
}

```

The whole thing is wrapped in a test that checks whether `getUserMedia` is supported before running anything else. Next, we call `getUserMedia()` and inside it define:

- The constraints: Only audio is to be captured for our dictaphone.
- The success callback: This code is run once the `getUserMedia` call has been completed successfully.
- The error/failure callback: The code is run if the `getUserMedia` call fails for whatever reason.

Note: All of the code below is placed inside the `getUserMedia` success callback.

Capturing the media stream

Once `getUserMedia` has created a media stream successfully, you create a new `MediaRecorder` instance with the `MediaRecorder()` constructor and pass it the stream directly. This is your entry point into using the `MediaRecorder` API — the stream is now ready to be captured into a [Blob](#), in the default encoding format of your browser.

```
const mediaRecorder = new MediaRecorder(stream);
```

There are a series of methods available in the [MediaRecorder](#) interface that allow you to control recording of the media stream; in Web Dictaphone we just make use of two, and listen to some events. First of all, [MediaRecorder.start\(\)](#) is used to start recording the stream once the record button is pressed:

```

record.onclick = () => {
  mediaRecorder.start();
  console.log(mediaRecorder.state);
  console.log("recorder started");
  record.style.background = "red";
  record.style.color = "black";
};

```

When the `MediaRecorder` is recording, the [MediaRecorder.state](#) property will return a value of "recording".

As recording progresses, we need to collect the audio data. We register an event handler to do this using [ondataavailable](#):

```

let chunks = [];

mediaRecorder.ondataavailable = (e) => {
  chunks.push(e.data);
};

```

Note: The browser will fire `dataavailable` events as needed, but if you want to intervene you can also include a timeslice when invoking the `start()` method — for example `start(10000)` — to control this interval, or call [MediaRecorder.requestData\(\)](#) to trigger an event when you need it.

Lastly, we use the [MediaRecorder.stop\(\)](#) method to stop the recording when the stop button is pressed, and finalize the [Blob](#) ready for use somewhere else in our application.

```

stop.onclick = () => {
  mediaRecorder.stop();
  console.log(mediaRecorder.state);
  console.log("recorder stopped");
  record.style.background = "";
  record.style.color = "";
};

```

Note that the recording may also stop naturally if the media stream ends (e.g. if you were grabbing a song track and the track ended, or the user stopped sharing their microphone).

Grabbing and using the blob

When recording has stopped, the `state` property returns a value of "inactive", and a stop event is fired. We register an event handler for this using [onstop](#), and finalize our blob there from all the chunks we have received:

```
mediaRecorder.onstop = (e) => {
  console.log("recorder stopped");

  const clipName = prompt("Enter a name for your sound clip");

  const clipContainer = document.createElement("article");
  const clipLabel = document.createElement("p");
  const audio = document.createElement("audio");
  const deleteButton = document.createElement("button");

  clipContainer.classList.add("clip");
  audio.setAttribute("controls", "");
  deleteButton.innerHTML = "Delete";
  clipLabel.innerHTML = clipName;

  clipContainer.appendChild(audio);
  clipContainer.appendChild(clipLabel);
  clipContainer.appendChild(deleteButton);
  soundClips.appendChild(clipContainer);

  const blob = new Blob(chunks, { type: "audio/ogg; codecs=opus" });
  chunks = [];
  const audioURL = window.URL.createObjectURL(blob);
  audio.src = audioURL;

  deleteButton.onclick = (e) => {
    let evtTgt = e.target;
    evtTgt.parentNode.removeChild(evtTgt);
  };
};
```

Let's go through the above code and look at what's happening.

First, we display a prompt asking the user to name their clip.

Next, we create an HTML structure like the following, inserting it into our clip container, which is an [<article>](#) element.

```
<article class="clip">
  <audio controls></audio>
  <p>your clip name</p>
  <button>Delete</button>
</article>
```

After that, we create a combined [Blob](#) out of the recorded audio chunks, and create an object URL pointing to it, using `window.URL.createObjectURL(blob)`. We then set the value of the [<audio>](#) element's `src` attribute to the object URL, so that when the play button is pressed on the audio player, it will play the [Blob](#).

Finally, we set an `onclick` handler on the delete button to be a function that deletes the whole clip HTML structure.

Specifications

Specification
MediaStream Recording # mediarecorder-api

Browser compatibility

[Report problems with this compatibility data on GitHub](#)

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	
MediaRecorder	Chrome 47	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 47	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
MediaRecorder() constructor	Chrome 47	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 47	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
options parameter	Chrome 49	Edge 79	Firefox 43	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 43	Opera 36 Android	Safari 14 on iOS	Safari on iOS
audioBitrateMode	Chrome 89	Edge 89	Firefox No	Opera 75	Safari No	Chrome Android 89	Firefox for Android No	Opera 63 Android	Safari No on iOS	Safari on iOS
audioBitsPerSecond	Chrome 49	Edge 79	Firefox 71	Opera 36	Safari 14.1	Chrome Android 49	Firefox for Android 79	Opera 36 Android	Safari 14.5 on iOS	Safari on iOS
dataavailable event	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
error event	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
isTypeSupported	Chrome 47	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 47	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
mimeType	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
pause	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14.1	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14.5 on iOS	Safari on iOS
pause event	Chrome 49	Edge 79	Firefox 65	Opera 36	Safari 14.1	Chrome Android 49	Firefox for Android 65	Opera 36 Android	Safari 14.5 on iOS	Safari on iOS
requestData	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
resume	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14.1	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14.5 on iOS	Safari on iOS
resume event	Chrome 49	Edge 79	Firefox 65	Opera 36	Safari 14.1	Chrome Android 49	Firefox for Android 65	Opera 36 Android	Safari 14.5 on iOS	Safari on iOS
start	Chrome 47	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 47	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
start event	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
state	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
stop	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
stop event	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
stream	Chrome 49	Edge 79	Firefox 25	Opera 36	Safari 14	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari 14 on iOS	Safari on iOS
videoBitsPerSecond	Chrome 49	Edge 79	Firefox 71	Opera 36	Safari 14.1	Chrome Android 49	Firefox for Android 79	Opera 36 Android	Safari 14.5 on iOS	Safari on iOS
warning event	Chrome 49	Edge 79	Firefox 25–71	Opera 36	Safari No	Chrome Android 49	Firefox for Android 25	Opera 36 Android	Safari No on iOS	Safari on iOS

Tip: you can click/tap on a cell for more information.

Full support

Partial support

No support

Experimental. Expect behavior to change in the future.

Deprecated. Not for use in new websites.

See implementation notes.

Has more compatibility info.

See also

- MediaRecorder API landing page
- Navigator.getUserMedia()
- MediaRecorder API now supported by 65% of your website users

Last modified: Aug 9, 2022, [by MDN contributors](#)