BY PHIL NASH • 2020-02-10

TWITTER FACEBOOK LINKEDIN

Speech to text in the browser with the Web Speech API

The Web Speech API has two functions, speech synthesis, otherwise known as text to speech, and speech recognition, or speech to text. We previously investigated text to speech so let's take a look at how browsers handle recognising and transcribing speech with the `SpeechRecognition` API.

Being able to take voice commands from users means you can create more immersive interfaces and users like using their voice. In 2018, Google reported that 27% of the global online population is using voice search on mobile. With speech recognition in the browser you can enable users to speak to your site across everything from a voice search to creating an interactive bot as part of the application.

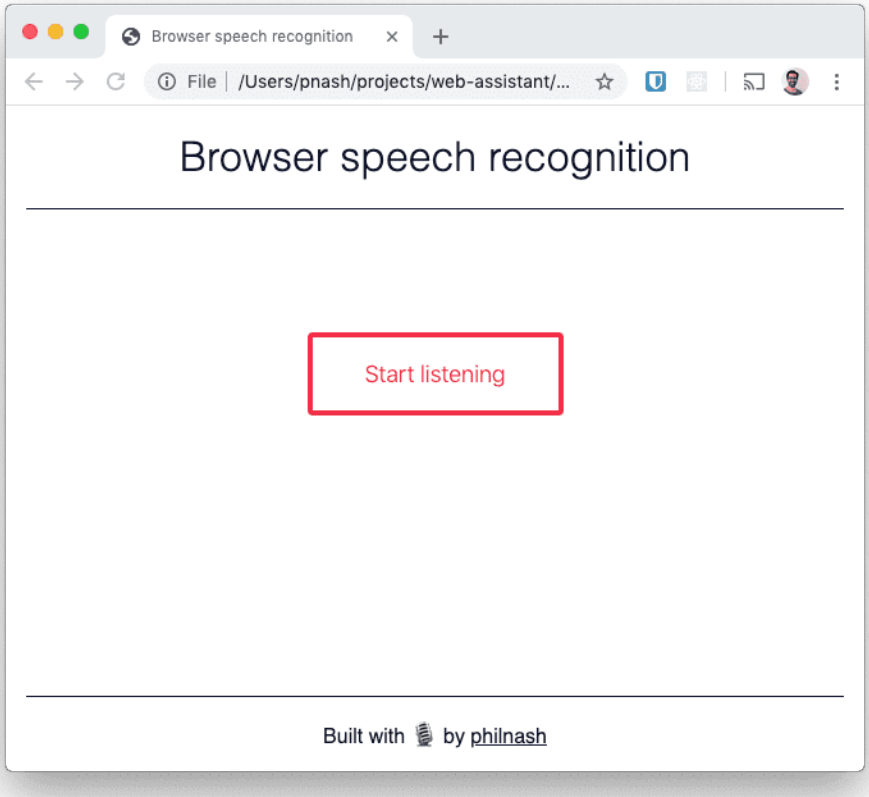
Let's see how the API works and what we can build with it.

What you'll need

We're going to build an example app to experience the API, if you want to build along you will need:

- Google Chrome
- A text editor

And that's it, we can do this with plain HTML, CSS and JavaScript. Once you have those prepared, create a new directory to work in and save this starter HTML and CSS to that directory. Make sure the files are in the same directory and then open the HTML file in the browser. It should look like this:



With that in place, let's see how to get the browser to listen to and understand us.

The SpeechRecognition API

Before we build speech recognition into our example application, let's get a feel for it in the browser dev tools. In Chrome open up your dev tools. Enter the following in the console:

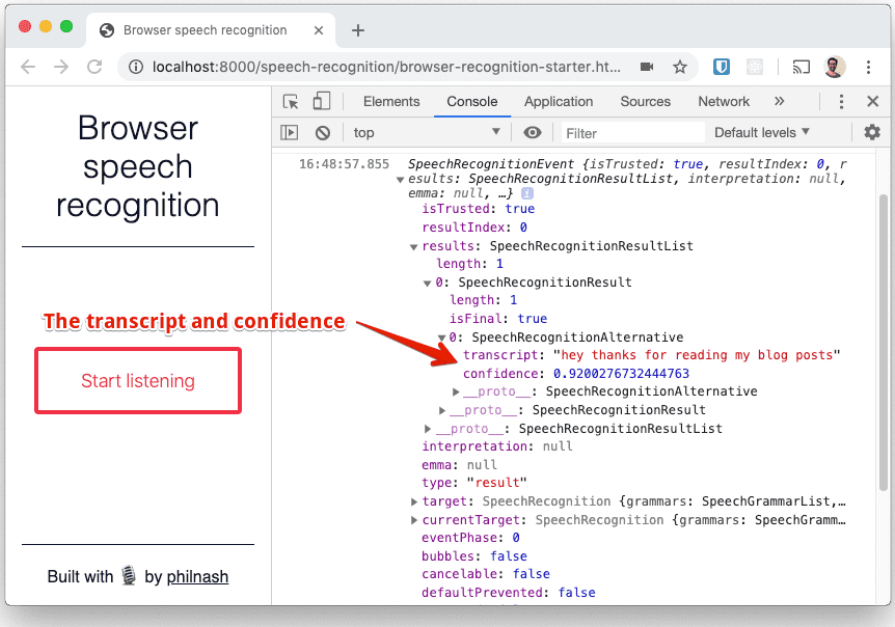
```
speechRecognition = new webkitSpeechRecognition();
speechRecognition.onresult = console.log;
speechRecognition.start();
```

When you run that code Chrome will ask for permission to use your microphone and then, if your page is being served on a web server, remember your choice. Run the code and, once you've given the permission, say something into your microphone. Once you stop speaking you should see a `SpeechRecognitionEvent` posted in the console.

There is a lot going on in these 3 lines of code. We created an instance of the `SpeechRecognition` API (vendor prefixed in this case with "webkit"), we told it to log any result it received from the speech to text service and we told it to start listening.

There are some default settings at work here too. Once the object receives a result it will stop listening. To continue transcription you need to call `start` again. Also, you only receive the final result from the speech recognition service. There are settings we'll see later that allow continuous transcription and interim results as you speak.

Let's dig into the `SpeechRecognitionEvent` object. The most important property is `results` which is a list of `SpeechRecognitionResult` objects. Well, there is one result object as we only said one thing before it stopped listening. Inspecting that result shows a list of `SpeechRecognitionAlternative` objects and the first one includes the transcript of what you said and a confidence value between 0 and 1. The default is to only return one alternative, but you can opt to receive more alternatives from the recognition service, which can be useful if you are letting your users select the option closest to what they said.



How it works

Calling this feature speech recognition in the browser is not exactly accurate. Chrome currently takes the audio and sends it to Google's servers to perform the transcription. This is why speech recognition is currently only supported in Chrome and some Chromium based browsers.

Mozilla has built support for speech recognition into Firefox, it is behind a flag in Firefox Nightly while they negotiate to also use the Google Cloud Speech API. Mozilla are working on their own DeepSpeech engine, but want to get support into browsers sooner so opted to use Google's service too.

So, since SpeechRecognition uses a server side API, your users will have to be online to use it. Hopefully we will see local, offline speech recognition abilities down the line, but for now this is a limitation.

Let's take the starter code we downloaded earlier and the code from dev tools and turn this into a small application where we live transcribe a user's speech.

Speech Recognition in a web application

Open the HTML you downloaded earlier and between the `<script>` tags at the bottom we'll start by listening for the `DOMContentLoaded` event and then grabbing references to some elements we'll use.

```
<script>
window.addEventListener("DOMContentLoaded", () => {
  const button = document.getElementById("button");
  const result = document.getElementById("result");
  const main = document.getElementsByTagName("main")[0];
});
</script>
```

We'll test to see if the browser supports the `SpeechRecognition` or `webkitSpeechRecognition` object and if it doesn't we'll show a message as we can't carry on.

```
<script>
window.addEventListener("DOMContentLoaded", () => {
  const button = document.getElementById("button");
  const result = document.getElementById("result");
  const main = document.getElementsByTagName("main")[0];
  const SpeechRecognition = window.SpeechRecognition || window.webkitSp
  if (typeof SpeechRecognition === "undefined") {
    button.remove();
    const message = document.getElementById("message");
    message.removeAttribute("hidden");
    message.setAttribute("aria-hidden", "false");
  } else {
    // good stuff to come here
  }
});
</script>
```

If we do have access to `SpeechRecognition` then we can prepare to use it. We'll define a variable to show whether we are currently listening for speech, instantiate the speech recognition object, and three functions to start, stop and respond to new results from the recogniser:

```
} else {
  let listening = false;
  const recognition = new SpeechRecognition();
  const start = () => {};
  const stop = () => {};
  const onResult = event => {};
}
```

For the start function, we want to start the speech recogniser and change the button text. We'll also add a class to the main element which will start an animation that shows the page is listening. For the stop function we'll do the opposite.

```
  const start = () => {
    recognition.start();
    button.textContent = "Stop listening";
    main.classList.add("speaking");
  };
  const stop = () => {
    recognition.stop();
    button.textContent = "Start listening";
    main.classList.remove("speaking");
  };
```

When we receive a result we will use it to render all results to the page. In this example we'll do so with straight DOM manipulation. We'll take the `SpeechRecognitionResult` objects we saw earlier and add them as paragraphs into the result `<div>`. To show the difference between final and interim results, we'll add a class to any results that are marked as final.

```
    const onResult = event => {
      result.innerHTML = "";
      for (const res of event.results) {
        const text = document.createTextNode(res[0].transcript);
        const p = document.createElement("p");
        if (res.isFinal) {
          p.classList.add("final");
        }
        p.appendChild(text);
        result.appendChild(p);
      }
    };
```

Before we run the speech recogniser we need to prepare it with the settings we'll use in this app. For this version we will continuously record the results instead of finishing after it detects the end of speech, this way we can keep transcribing it to the page until we press the stop button. We will also ask for interim results which will show us what the recogniser comes up with as we speak (much like you can do with speech to text during a Twilio phone call with `<Gather>` and `partialResultCallback`). We'll also add the result listener:

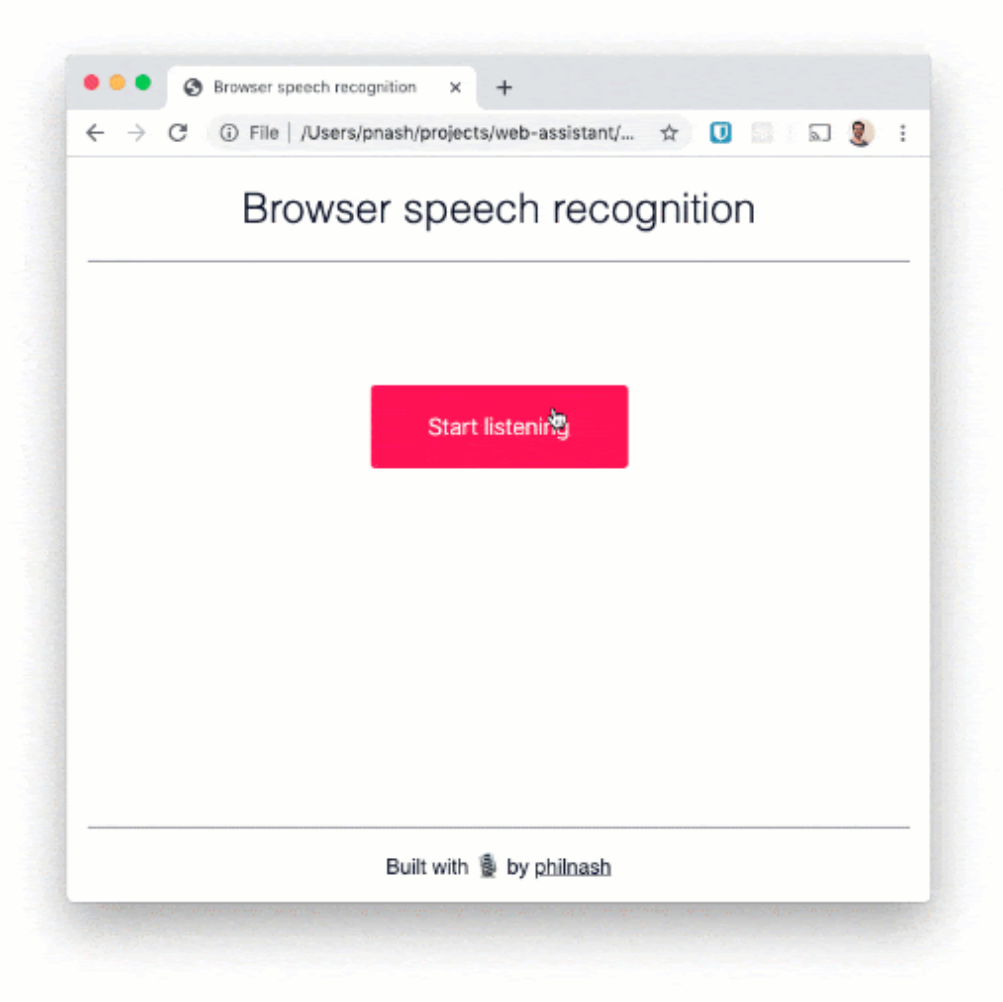
```
        const onResult = event => {
          // onResult code
        }
        recognition.continuous = true;
        recognition.interimResults = true;
        recognition.addEventListener("result", onResult);
      }
```

Finally, we'll add a listener to the button to start and stop recognition.

```
    const onResult = event => {
      // onResult code
    }
    recognition.continuous = true;
    recognition.interimResults = true;
    recognition.addEventListener("result", onResult);

    button.addEventListener("click", () => {
      listening ? stop() : start();
      listening = !listening;
    });
  }
```

Reload the browser and try it out.



You can now say several sentences and see them written to the page. The recogniser is pretty good at words, but less so at punctuation. There'd be a bit more work to do here if we wanted to turn this into dictation, for example.

Now we can talk to the browser

In this post you've seen how we can talk to the browser and have it understand us. In a previous post we also saw how the browser can speak to us. Putting these together along with a Twilio Autopilot powered assistant could make for a very interesting project.

If you want to play with the example from this post you can check it out on Glitch here. And if you want the source code, it's available in my web-assistant repo on GitHub.

There are all sorts of opportunities for interesting user interfaces using speech. I recently saw a great example of a voice based game in the browser. Let me know if you are working on something interesting with speech recognition in browsers either in the comments below or on Twitter at @philnash.

RATE THIS POST ★★★★★

AUTHORS  Phil Nash

Stories From The Road

The Owl's Nest: Inside Twilio

TWITTER FACEBOOK

Developer stories to your inbox.

Subscribe to the Developer Digest, a monthly dose of all things code.

Enter your email...

You may unsubscribe at any time using the unsubscribe link in the digest email. See our privacy policy for more information.

NEW!

Tutorials

Sample applications that cover common use cases in a variety of languages. Download, test drive, and tweak them yourself.

Get started

SIGN UP AND START BUILDING

Not ready yet? Talk to an expert.

ABOUT

LEGAL