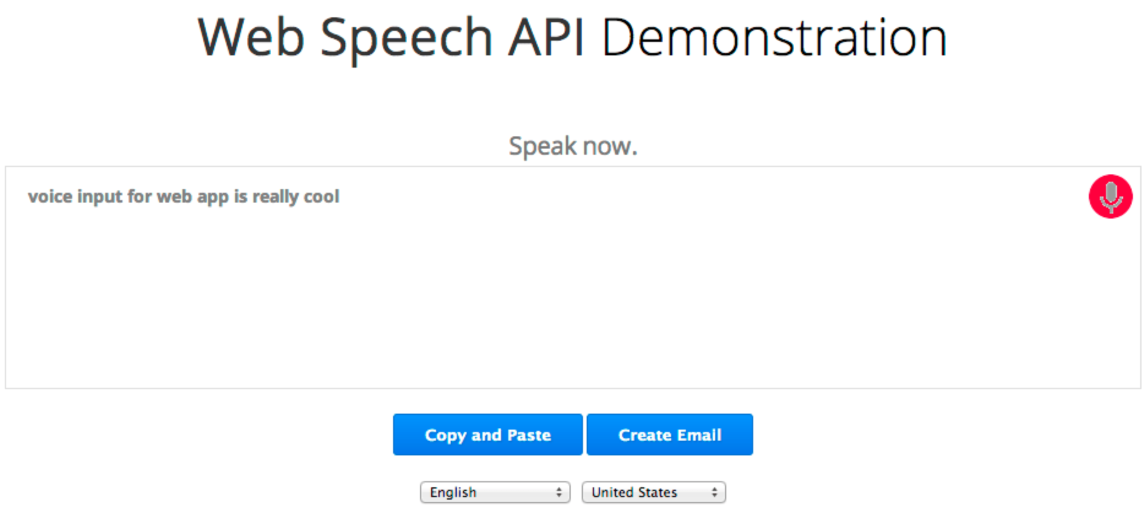
Voice Driven Web Apps: Introduction to the Web Speech API

By Glen Shires

Glen is a contributor to Web**Fundamentals**

The new JavaScript Web Speech API makes it easy to add speech recognition to your web pages. This API allows fine control and flexibility over the speech recognition capabilities in Chrome version 25 and later. Here's an example with the recognized text appearing almost immediately while speaking.



Web Speech API Demonstration

Speak now.

voice input for web app is really cool

DEMO / SOURCE

Let’s take a look under the hood. First we check to see if the browser supports the Web Speech API by checking if the `webkitSpeechRecognition` object exists. If not, we suggest the user upgrades their browser. (Since the API is still experimental, it's currently vendor prefixed.) Lastly, we create the `webkitSpeechRecognition` object which provides the speech interface, and set some of its attributes and event handlers.

```
if (!('webkitSpeechRecognition' in window)) {
  upgrade();
} else {
  var recognition = new webkitSpeechRecognition();
  recognition.continuous = true;
  recognition.interimResults = true;

  recognition.onstart = function() { ... }
  recognition.onresult = function(event) { ... }
  recognition.onerror = function(event) { ... }
  recognition.onend = function() { ... }
  ...
```

The default value for `continuous` is false, meaning that when the user stops talking, speech recognition will end. This mode is great for simple text like short input fields. In this demo, we set it to true, so that recognition will continue even if the user pauses while speaking.

The default value for `interimResults` is false, meaning that the only results returned by the recognizer are final and will not change. The demo sets it to true so we get early, interim results that may change. Watch the demo carefully, the grey text is the text that is interim and does sometimes change, whereas the black text are responses from the recognizer that are marked final and will not change.

To get started, the user clicks on the microphone button, which triggers this code:

```
function startButton(event) {
  ...
  final_transcript = '';
  recognition.lang = select_dialect.value;
  recognition.start();
```

We set the spoken language for the speech recognizer "lang" to the BCP-47 value that the user has selected via the selection drop-down list, for example “en-US” for English-United States. If this is not set, it defaults to the lang of the HTML document root element and hierarchy. Chrome speech recognition supports numerous languages (see the “`langs`” table in the demo source), as well as some right-to-left languages that are not included in this demo, such as he-IL and ar-EG.

After setting the language, we call `recognition.start()` to activate the speech recognizer. Once it begins capturing audio, it calls the `onstart` event handler, and then for each new set of results, it calls the `onresult` event handler.

```
  recognition.onresult = function(event) {
    var interim_transcript = '';

    for (var i = event.resultIndex; i < event.results.length; ++i) {
      if (event.results[i].isFinal) {
        final_transcript += event.results[i][0].transcript;
      } else {
        interim_transcript += event.results[i][0].transcript;
      }
    }
    final_transcript = capitalize(final_transcript);
    final_span.innerHTML = linebreak(final_transcript);
    interim_span.innerHTML = linebreak(interim_transcript);
  };
}
```

This handler concatenates all the results received so far into two strings: `final_transcript` and `interim_transcript`. The resulting strings may include "\n", such as when the user speaks “new paragraph”, so we use the `linebreak` function to convert these to HTML tags `<br>` or `<p>`. Finally it sets these strings as the innerHTML of their corresponding `<span>` elements: `final_span` which is styled with black text, and `interim_span` which is styled with gray text.

`interim_transcript` is a local variable, and is completely rebuilt each time this event is called because it’s possible that all interim results have changed since the last `onresult` event. We could do the same for `final_transcript` simply by starting the for loop at 0. However, because final text never changes, we’ve made the code here a bit more efficient by making `final_transcript` a global, so that this event can start the for loop at `event.resultIndex` and only append any new final text.

That’s it! The rest of the code is there just to make everything look pretty. It maintains state, shows the user some informative messages, and swaps the GIF image on the microphone button between the static microphone, the mic-slash image, and mic-animate with the pulsating red dot.

The mic-slash image is shown when `recognition.start()` is called, and then replaced with mic-animate when `onstart` fires. Typically this happens so quickly that the slash is not noticeable, but the first time speech recognition is used, Chrome needs to ask the user for permission to use the microphone, in which case `onstart` only fires when and if the user allows permission. **Pages hosted on HTTPS do not need to ask repeatedly for permission, whereas HTTP hosted pages do.**

So make your web pages come alive by enabling them to listen to your users!

We'd love to hear your feedback...

- For comments on the W3C Web Speech API specification: email, mailing archive, community group
- For comments on Chrome's implementation of this spec: email, mailing archive

Please refer to the Chrome Privacy Whitepaper to learn how Google is handling voice data from this API.