Hands-Free Coding

How I develop software using dictation and eye-tracking

Earlier this year, I developed Cubital Tunnel Syndrome, a repetitive-strain injury, in both of my elbows. As a result, I pretty much can't use a mouse or keyboard; after a few minutes, I get a burning pain shooting down my arms. Even if I try to limit my computer usage to 60-second bursts, I wind up inadvertently making the situation worse.

As you might imagine, this was a pretty big deal; as a software developer, my entire career is based on being able to use a keyboard!

After many failed attempts at solving the problem with physiotherapy, ergonomics, braces, diet and supplements, prescription medications, supplements, mindbody soul-searching, and a bunch of other stuff, I've found a solution that allows me to be productive without risking further nerve damage. I now work almost exclusively using a microphone and an eye-tracker.

In this article, I'll show you what that workflow looks like, and how I've optimized it to fit my needs!

Update, December 9th: I am thrilled to report that my injury has gotten much better! I'm back to using a keyboard and mouse as my primary

input mechanism. It's a relief, but I also feel confident that I would have managed just fine either way.

A quick demo

To give you a quick sense of what this looks like, here's a short video of me writing a React component:

We'll get into how this all works, don't worry if it doesn't make a ton of sense yet! I mostly wanted to showcase this upfront to show how *feasible* this process can be.

Writing code with Talon

Dictation software has been around for a long time, but it's usually used purely to transcribe speech, often in the legal and medical industries. Writing code is a different beast, since there's a lot of syntax and conventions and non-dictionary words.

Fortunately, specialized software exists! I currently use Talon Voice, a tool built specifically to help software developers work without using

their hands.

Talon has a free public version, but the exciting stuff happens in the paid private beta. You can gain access by supporting the creator on Patreon.

Let's dive into how this software works.

Alphabet

The first thing you learn as a new Talon user is how to dictate individual letters.

Generally, you won't be dictating one letter at a time, but it comes in handy now and then, like specifying CSS units (px, rem, etc).

English is an annoying language when it comes to phonetics. So many of our letters sound the same. There's a reason telephone operators say things like "M like Mary", "T as in Thomas".

The United Nations solved this problem with the NATO phonetic alphabet—you know, the Whiskey-Tango-Foxtrot thing. But these words tend to be multi-syllable, and nobody has time for that jazz. So Talon includes its own phonetic mappings of (mostly) single-syllable words:

- → `a` - air
- → `b` - bat
- → `c` - cap
- → `d` - drum

When I say “drum” into the microphone, the letter `d` is written as if I had pressed that key on the keyboard.

You can capitalize letters by prefixing them with “ship”. “ship drum” will output `D` instead of `d`.

Numbers are spoken normally, from 0 through 9. If I wanted to output `1024`, I would speak “one zero two four”.

Hotkeys and ordinals

Talon has intuitive mappings for most special characters. `command cap`, for example, will hold `command` while pressing the `C` key, to copy to the clipboard. `control command space` will open the Emoji drawer on MacOS, since that's the OS-level mapping.

Certain keys are mapped to shorter/cuter terms. Instead of "backspace", I say "junk". "Delete" becomes "dell". If you're unhappy with any of the mappings, by the way, everything is editable in Talon.

Arrow keys are prefixed with the word "go". If I want to move the cursor left, I say `go left`.

This would be really tedious if not for one awesome addition: ***ordinals***.

In English, an ordinal number is one used to describe order, like "fifth" or "ninth" or "three hundredth". In Talon, they're used to repeat commands. If I wanted to go left by 9 spaces, I would say `go left ninth`.

The phrasing is a little strange. Surely, "go left nine" would be more intuitive, right? But `nine` is already taken; it outputs the literal number `9`.

This works for all commands. If I wanted to write the number `1000`, I would say `one zero third`, to repeat the `0` character 3 times.

Formatters

The conventional way to write JavaScript uses camelCase for variables. In fact, there are lots of conventions when it comes to variable names! Talon has a solution for this: **formatters.**

A formatter is a command which will transform the text spoken afterwards. When I say “camel hello world”, for example, the software outputs `helloWorld`. Conversely, “snake hello world” produces `hello_world`.

If you want to output text without transforming it, the command is `say`. “say hello world” will output `hello world`.

Formatters can be composed. For example, I'm a fan of UPPER_SNAKE for JavaScript constants:

JS

```
const DARK_COLORS = {
  primary:                          ,
  // ...and so on
};
```

To output `DARK_COLORS`, I can combine the `snake` and `allcaps` formatters. “allcaps snake dark colors” outputs `DARK_COLORS`.

Command mode

While Talon does have a "dictation mode", the default mode is command-based. Commands can be thought of as functions. Everything we've seen so far is command-based.

For example, when I say `focus chrome`, it's like I'm calling the `focus` function, and passing `chrome` as an argument. `focus` is a command that focuses the specified application, so this would be equivalent to using Spotlight to select Chrome.

`focus` isn't some black-box native thing built into Talon, though; it's part of a community package of commands. I can access and edit the source, which is written in Python:

PY

```
class Actions:
  def switcher_focus(name: str):

    for app in ui.apps():
      if name in app.name and not app.background:
        app.focus()
        break
```

The real power of Talon is being able to ***create your own commands***. It offers a bunch of APIs for interacting with the operating system and outputting characters. I've created a dozen handy utilities for front-end development, and I expect I'll add many more as I keep using it.

You can add simple "say X to produce Y" commands using a YAML-like syntax:

YAML

```
react: insert("import React from 'react';")
```

When I speak "react", the software outputs `import React from 'react';`.

For more complex commands, you can write Python functions. As an example, here's what happens when I say “styled button fancy button”:

JS

```
const FancyButton = styled.button
  | <-- Cursor placed here
 ;
```

The second word, `button`, is matched against a known set of HTML elements. The subsequent words, `fancy button`, are UpperCamelCased and used for the component name. It adds some whitespace, and moves the cursor to the appropriate spot.

Here's the Python source for the command:

PY

```
@ctx.capture(rule=                                    )
def create_styled_component(m):
  component_name = actions.user.formatted_text(
    m.text,

  )

  return          {component_name}          {m.html_elements}
```

And here's the Talon mapping:

```
<user.create_styled_component>:
  insert(create_styled_component)
  key('left enter enter up tab')
```

Programming Talon commands is beyond the scope of this article. If you're interested, check out the unofficial Talon docs. You can also learn a ton by reading how existing commands are implemented.

You can also check out my fork of the commands, which includes all the React stuff I've added—be warned, though, it's messy, incomplete, and poorly documented.

Homophones

No matter how good speech recognition gets, there will always be ambiguities that will be difficult to resolve.

For example, if I say "check out my site", do I mean `site` or `sight`? Or possibly `cite`??

To resolve these ambiguities, Talon includes a `phones` command:

I learned about this trick from Emily Shea's amazing conference talk, Perl Out Loud.

> **Menu UI!**
>
> In order to select the correct homophone, Talon pops open a little menu. Because everything in Talon is made available through APIs, I imagine we can use the same UI for other things!
>
> I'm excited to experiment more with this idea.

Eye-tracking as a mouse replacement

By far the most sci-fi part of my setup is my eye-tracker.

I use the tobii 5. It's a bar with an infrared sensor, and it tracks your eye motion. It slaps onto the front of your monitor:

Interestingly, it isn't marketed as a mouse replacement; it's designed for Windows users for some sort of competitive gaming purpose. But Talon—the same software I use for dictation—includes custom MacOS drivers that allow it to function as a mouse replacement.

Clicking is a two step process. First, you look where you want to click, and make a popping noise with your mouth. This will zoom way in, and allow you to be really precise with your click. A second pop will perform a left-click:

There are commands to double-click, to right-click, and to drag and release. It takes some getting used to, but it works surprisingly well. The accuracy is good enough to do some pretty precise things.

The tobii 5 sells for $229 USD. You can also try and find the tobii 4C, which is purported to offer a smoother experience with Talon, but they're really rare.

The bigger picture

So far, I've shared only the tip of the iceberg of what I've learned, and what I've learned is only the tip of an even-bigger iceberg—Talon is a really powerful tool, and I'm still figuring it out. It took *years* to

become proficient with a keyboard, so I'm still very early into my journey with dictation.

In fact, I'd say that this whole cottage industry is pretty new. Talon is a wonderful piece of technology, and it's already had a huge positive impact in my life, but I think there's so much potential and opportunity ahead.

Talon continues to improve every day—it uses a proprietary machine-learning algorithm to handle speech-recognition, and I've already seen a noticeable improvement with it. Other products like Serenade seem pretty compelling as well.

Meanwhile, companies like Neuralink are working on establishing a "direct link" between our brains and everyday technology. It sounds like science fiction, but I may soon be able to "think" my code into existence ✨😮✨

My results so far

I'd say I probably work at about 50% of my normal speed*. Now, this doesn't mean that I produce 50% of the results; it just means I need to prioritize a little more ruthlessly.

I've heard that learning Vim can make this much more effective. Depending on how much longer my injury lasts, I may consider switching.

The biggest issue I've found so far is voice strain; I'm not used to talking for 8+ hours a day! I imagine I need to build a tolerance, and I hope to get better at this with time.

The first few weeks were rough. In addition to it being slow and frustrating, Talon work best when you write your own commands. I'd wind up hurting myself trying to get it set up. Being able to configure Talon by voice is a real milestone, and it's gone much smoother since then.

Honestly, it's just been such a relief to discover that my hands aren't needed for me to do my work. Recently, I heard Kent C Dodds and Joel Hooks talking on the egghead podcast about how Kent's wary of injuring his hands, since as a software developer and educator, they're his money-makers*. I used to feel the same way, whereas now I see that with a bit of determination and a lot of awesome technology, nothing's gonna stand in my way 💖

Accessibility matters

There's something else I want to talk about, and it's a bit less fun.

Here's the thing: you are not likely to develop Cubital Tunnel Syndrome. Even if you do, it'll likely go away on its own after a few weeks; many cases resolve spontaneously, and most respond well to conservative treatments. I'm an edge-case.

At some point in your life, however, you will likely experience some sort of impairment, whether temporary or permanent. Almost all of us will*.

It's so so easy to fall into the trap of thinking about accessibility as something that affects ***other people***, a hypothetical abstract group. I've known that accessibility is important for years, but it felt kinda nebulous to me; I've never watched someone struggle to use a thing I

built because I neglected to test it without a mouse or keyboard. It feels more urgent to me now.

I am still incredibly privileged, and I don't mean to compare my situation to anybody else's. But this experience has given me a window into what it's like trying to operate on an internet not designed with alternative input mechanisms in mind. Before I got comfortable with the eye-tracker, things were ***tricky***. And certain things are much more difficult than they used to be.

The internet has become critical infrastructure. It's a necessary part of living in modern society, and it needs to be accessible! As front-end developers, it's our job to advocate for it, and to ensure that we build with accessibility principles in mind.

If you'd like to learn more about accessibility, I recommend checking out a11y.coffee.

No time like the present

I've learned one other thing from this experience: I should prioritize stuff which is important to me!

One of the very first web apps I built was an education platform. This was about a decade ago, and it was built with PHP, MySQL, and jQuery.

I gave up on that product when I discovered Khan Academy, which was essentially what I was doing, but way better. I would later go on to work as a software engineer at Khan Academy, and do some of the most fulfilling work of my career.

I've long since imagined that at some point, I'd start my own thing in education. Even though I've been motivated to do this for years, I kept putting it off. This experience has taught me something valuable: I don't have an infinite amount of time ahead of me. If there's something I want to do, I should do it now, since I may not be able to do it later.

A few weeks ago, I left my job as a Senior Staff Software Engineer at Gatsby Inc, to pursue this dream. My first project is an online interactive course that teaches advanced CSS skills to JS developers. I've seen *so* much frustration around CSS, and the goal is to give you rock-solid confidence, the ability to implement any layout and build all kinds of cool, next-level experiences.

You can learn more about it on the CSS for JavaScript Developers site.

Acknowledgments

I'd like to thank two friends and former coworkers who suggested the idea of dictation to me: Amberley and Madalyn. I'm not sure the idea ever would have occurred to me!

I was also inspired by two conference talks on this subject:

→ Perl Out Loud, by Emily Shea

→ Using Python to Code by Voice, by Tavis Rudd

Did you enjoy this look into an alternative workflow? Share it with your network on twitter!

Share on Twitter

A front-end web development newsletter that sparks joy

My goal with this blog is to create helpful content for front-end web devs, and my newsletter is no different! I'll let you know when I publish new content, and I'll even share ***exclusive newsletter-only content*** now and then.

No spam, unsubscribe at any time.

First Name

Email

Last Updated: ***December 9th, 2020***

Josh Comeau

Tutorials

React
CSS
Gatsby
Performance
Animation
Career
Next.js
JavaScript

Links

Twitter
Contact
Terms of Use
Privacy Policy