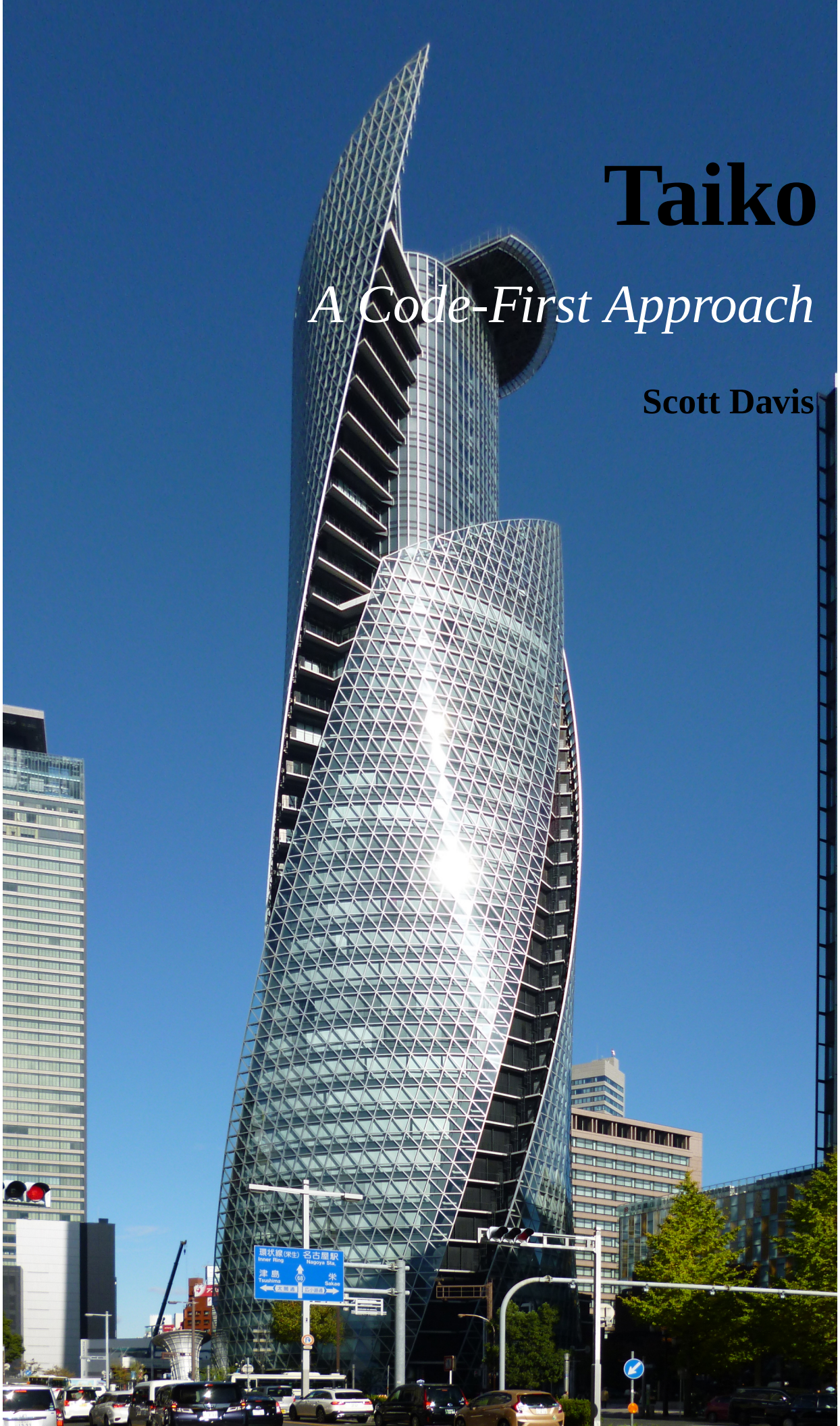

Taiko

A Code-First Approach

Scott Davis

Table of Contents

Introduction

Taiko is a JavaScript-based Domain Specific Language (DSL) for automatically driving your web browser just like a typical user does. If a user goes to your website, clicks on a link, fills in some form fields, and clicks the "submit" button, you can script up that behavior in Taiko and replay it in a reliable, automated way.

Installation and Configuration

Installing Taiko couldn’t be easier. It’s a single command: `npm install -g taiko`. But there’s plenty more that you can do to configure and customize Taiko once it’s installed.

Install Taiko

```
$ npm install -g taiko

/Users/scott/.nvm/versions/node/v12.14.1/bin/taiko ->
/Users/scott/.nvm/versions/node/v12.14.1/lib/node_modules/
  taiko/bin/taiko.js

> taiko@1.0.7 install
/Users/scott/.nvm/versions/node/v12.14.1/lib/node_modules/taiko
> node lib/install.js

Downloading Chromium r724157 - 117.6 Mb [====================] 100%
0.0s

> taiko@1.0.7 postinstall
/Users/scott/.nvm/versions/node/v12.14.1/lib/node_modules/taiko
> node lib/documentation.js

Generating documentation to lib/api.json
+ taiko@1.0.7
added 73 packages from 114 contributors in 50.835s
```

When you install Taiko, notice that you get a known-compatible version of Chromium installed as well. Chromium is an open-source, bare-bones web browser that, as you might've guessed by the name, is the core of the Google Chrome browser. Interestingly, Chromium is also the foundation of the Opera browser, the Microsoft Edge browser, and many others. Chromium-based browsers make up roughly two-thirds of the browser market, so using Chromium with Taiko covers the widest possible swath of typical web users.

Run the Taiko REPL

```
$ taiko

Version: 1.0.7 (Chromium:81.0.3994.0)
Type .api for help and .exit to quit

> openBrowser()
 ¬ Browser opened
> goto('wikipedia.org')
 ¬ Navigated to URL http://wikipedia.org
> click('Search')
 ¬ Clicked element matching text "Search" 1 times
> write('User (computing)')
 ¬ Wrote User (computing) into the focused element.
> press('Enter')
 ¬ Pressed the Enter key
> click('Terminology')
 ¬ Clicked element matching text "Terminology" 1 times
> closeBrowser()
 ¬ Browser closed
> .exit
```

The Taiko REPL (Read Evaluate Print Loop) is an interactive terminal shell that allows you to experiment with a live browser. When you type `openBrowser()`, a browser window should open on your computer. When you type `goto('wikipedia.org')`, you should end up on the Wikipedia website.

The Taiko REPL is the perfect way to experiment with Taiko whether you are brand new to the DSL or an experienced user. Once you are confident that your code works (because you've just watched it work), you can save it and run it outside of the REPL, either manually or as a part of your automated CD pipeline.

Save Code from the Taiko REPL

```
$ taiko

> openBrowser()
 ¬ Browser opened
> goto('wikipedia.org')
 ¬ Navigated to URL http://wikipedia.org
> closeBrowser()
 ¬ Browser closed
> .code

const { openBrowser, goto, closeBrowser } = require('taiko');
(async () => {
    try {
        await openBrowser();
        await goto('wikipedia.org');
    } catch (error) {
        console.error(error);
    } finally {
        await closeBrowser();
    }
})();

// If you provide a filename,
//   .code saves your code to the current directory
> .code visit-wikipedia.js
```

At any point in the Taiko REPL, you can type `.code` to see what the JavaScript will look like once you run your Taiko code outside of the REPL. Notice that this is modern asynchronous JavaScript — every command will `await` completion before moving on to the next step.

If you'd like to save this code for running outside of the REPL, simply provide a filename like `.code visit-wikipedia.js`. This will save the JavaScript code to the current directory.

Run Taiko Code Outside of the REPL

```
$ taiko visit-wikipedia.js

 ¬ Browser opened
 ¬ Navigated to URL http://wikipedia.org
 ¬ Browser closed
```

When you type `taiko` without a filename, it launches the Taiko REPL. When you type `taiko visit-wikipedia.js`, it runs the Taiko commands in the file.

You might have noticed that typing `openBrowser()` in the Taiko REPL actually opens a browser that you can see. By default, running Taiko commands outside of the REPL runs the browser in "headless mode". This means that the browser isn't actually shown on screen, but its behavior in headless mode is identical to its behavior with a visible browser. This is ideal for running Taiko commands in an automated server environment where there most likely isn't a screen to display the progress.

If you'd like to see the browser when running Taiko commands outside of the REPL, type `taiko --observe visit-wikipedia.js`. The `--observe` command-line flag, in addition to showing the browser, also inserts a 3 second (3000 millisecond) delay between steps to make them easier to observe. If you'd like to adjust this delay, use the `--wait-time` command-line flag — `taiko --observe --wait-time 1000 visit-wikipedia.js`.

Get Command-Line Help

```
$ taiko --help

Usage: taiko [options]
       taiko <file> [options]

Options:
  -v, --version                    output the version number

  -o, --observe                    enables headful mode and runs
                                   script with 3000ms delay by
                                   default. pass --wait-time
                                   option to override the default
                                   3000ms

  -l, --load                       run the given file and start the
                                   repl to record further steps.

  -w, --wait-time <time in ms>     runs script with provided delay

  --emulate-device <device>        Allows to simulate device
                                   viewport.
                                   Visit https://github.com/getgauge/
                                   taiko/blob/master/lib/devices.js
                                   for all the available devices

  --emulate-network <networkType>  Allow to simulate network.
                                   Available options are GPRS,
                                   Regular2G, Good2G, Regular3G,
                                   Good3G, Regular4G, DSL,
                                   WiFi, Offline

  --plugin <plugin1,plugin2...>    Load the taiko plugin.

  --no-log                         Disable log output of taiko

  -h, --help                       display help for command
```

There are a number of command-line flags that affect Taiko at runtime. `--observe` and `--wait-time` allow you to see the browser as the Taiko commands are performed. (Normally, Taiko runs in "headless mode" at the command-line.)

You can use `--emulate-device` and `--emulate-network` to simulate smartphone usage.

`--load` allows you to preload the Taiko REPL with commands stored in a file.

`--plugin` allows you to load Taiko plugins that extend native behavior.

Run Taiko in an Alternate Browser

```
$ TAIKO_BROWSER_PATH=/Applications/Opera.app/Contents/MacOS/Opera
  taiko visit-wikipedia.js

¬ Browser opened
 ¬ Navigated to URL http://wikipedia.org
 ¬ Browser closed
```

When you install Taiko, it ships with a known-good version of Chromium — one that won't auto-update and inadvertently break your tests. But you might want to use Taiko to drive an alternate Chromium-based browser, like Google Chrome, Opera, or Microsoft Edge. To do so, simply create a `TAIKO_BROWSER_PATH` environment variable that contains the path to the browser you'd like Taiko to use.

NOTE

Taiko uses the Chrome DevTools Protocol (CDP) to communicate with the browser. This is the same protocol that the Google Chrome DevTools use, as well as Lighthouse (for reporting) and Puppeteer (a similar tool to Taiko written by Google). As of this writing, neither Firefox nor Safari support CDP-based communications. For an alternate way to drive non-CDP browsers, look at the WebDriver[1] W3C initiative.

Emulate a Smartphone

```
$ taiko --observe
        --emulate-device 'iPhone X'
        --emulate-network 'Regular3G'
        visit-wikipedia.js

 ¬ Browser opened with viewport iPhone X
 ¬ Device emulation set to iPhone X
 ¬ Set network emulation with values "Regular3G"
 ¬ Navigated to URL http://wikipedia.org
 ¬ Device emulation set to iPhone X
 ¬ Browser closed
```

When you run Taiko on your desktop computer, it opens a desktop browser and runs at full network speed. If you'd like Taiko to emulate a different kind of device, use the `--emulate-device` and `--emulate-network` command-line flags.

To find the available values for these flags, type `taiko --help`.

For a better understanding of what these flags do, you can look at the JavaScript files that supply the values in devices.js[2] and networkConditions.js[3] on GitHub[4].

Here is the code for `iPhone X` device emulation:

```
  'iPhone X': {
    userAgent:
      'Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X)
AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372
Safari/604.1',
    viewport: {
      width: 375,
      height: 812,
      deviceScaleFactor: 3,
      isMobile: true,
      hasTouch: true,
      isLandscape: false,
    },
  },
```

The emulation code sets a device-specific User-Agent string, and adjusts the size and characteristics of the screen.

Here is the code for `Regular3G` network emulation:

```
  Regular3G: {
    offline: false,
    downloadThroughput: (750 * 1024) / 8,
    uploadThroughput: (250 * 1024) / 8,
    latency: 100,
  },
```

The emulation code throttles download and upload speeds, as well as adding some artificial latency.

[1] https://www.w3.org/TR/webdriver2/
[2] https://github.com/getgauge/taiko/blob/master/lib/data/devices.js
[3] https://github.com/getgauge/taiko/blob/master/lib/data/networkConditions.js
[4] https://github.com/getgauge/taiko

Working with the Browser

In this chapter, you'll learn how to open and close a browser, open and close tabs, and take a screenshot.

Open and Close a Browser

```
> openBrowser()
 ¬ Browser opened
> closeBrowser()
 ¬ Browser closed
> .code
const { openBrowser, closeBrowser } = require('taiko');
(async () => {
    try {
        await openBrowser();
    } catch (error) {
        console.error(error);
    } finally {
        await closeBrowser();
    }
})();
```

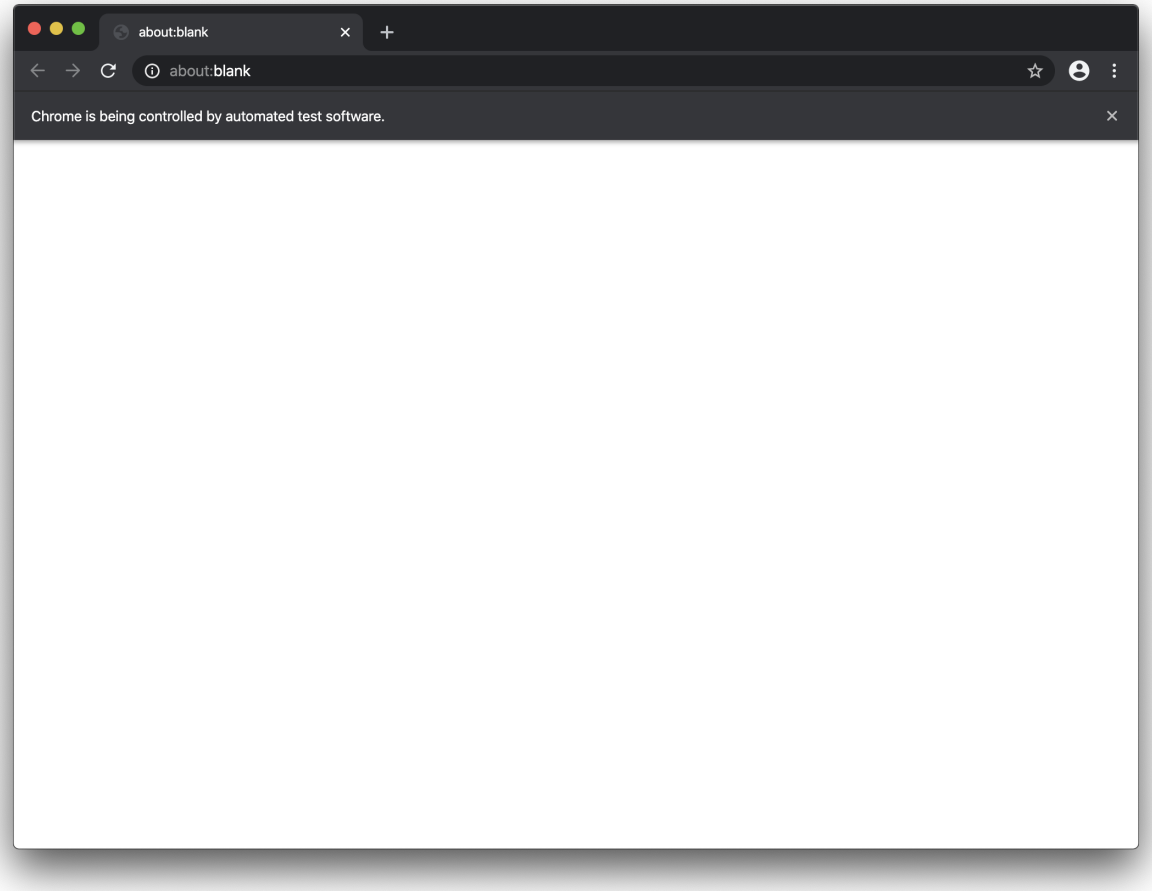

Figure 1. `openBrowser` *opens a new browser window with a single empty new tab.*

Every Taiko action assumes that you have an open, active browser window as the result of an `openBrowser` call. You'll also want to close the browser window at the end of your Taiko script by calling `closeBrowser`.

The `.code` output shows you one way to structure your code in a standard JavaScript `try/catch/finally` block. The `finally` block ensures that the browser window closes at the end of the script run, regardless of whether the run was successful (`try`) or encountered errors along the way (`catch`).

NOTE

All Taiko actions are asynchronous. When running Taiko in a script outside of the REPL, be sure to mark the function as `async` and preceed each Taiko action with `await` to ensure that it has fully completed before the next Taiko action is called.

*Open a Browser with a Specific Screen Size

```
> openBrowser({args:['--window-size=1024,768']})
 ¬ Browser opened
> .code
const { openBrowser, closeBrowser } = require('taiko');
(async () => {
    try {
        await openBrowser({args:['--window-size=1024,768']});
    } catch (error) {
        console.error(error);
    } finally {
        await closeBrowser();
    }
})();
```

*Goto a URL

```
goto
goBack
goForward
reload
currentURL
title
```

*Open and Close a Tab

```
openTab
closeTab
switchTo
```

*Open and Close an Incognito Window

```
openIncognitoWindow
closeIncognitoWindow
```

*Scroll the Web Page

```
scrollTo
scrollRight
scrollLeft
scrollUp
scrollDown
```

*Take a Screenshot

```
screenshot
```

Selecting Elements on the Page

In this chapter, you’ll learn how to select elements on the page.

*Click or Tap an Element

```
click
tap
```

*Click a Link or Button

```
link
button
```

*Select an Element Near Another

```
near
above
below
toLeftOf
toRightOf
```

*Select and Highlight Text

```
text
highlight
clearHighlight
```

*Select an Image

```
image
```

*Select a List Item

```
listItem
```

*Select a Table Cell

```
tableCell
```

*Select an Element by Class or Id

```
$
```

Working with Forms

In this chapter, you’ll learn how to work with forms.

*Write in a Text Field

```
textBox
click
focus
write
clear
to
into
press
```

*Click a Checkbox or a Radio Button

```
checkBox
radioButton
```

*Select from a Dropdown

```
dropDown
```

*Upload a File

```
fileField
attach
```

*Adjust a Time Field

```
timeField
```

*Pick from a Color Field

```
color
```

*Adjust a Range Field

```
range
```