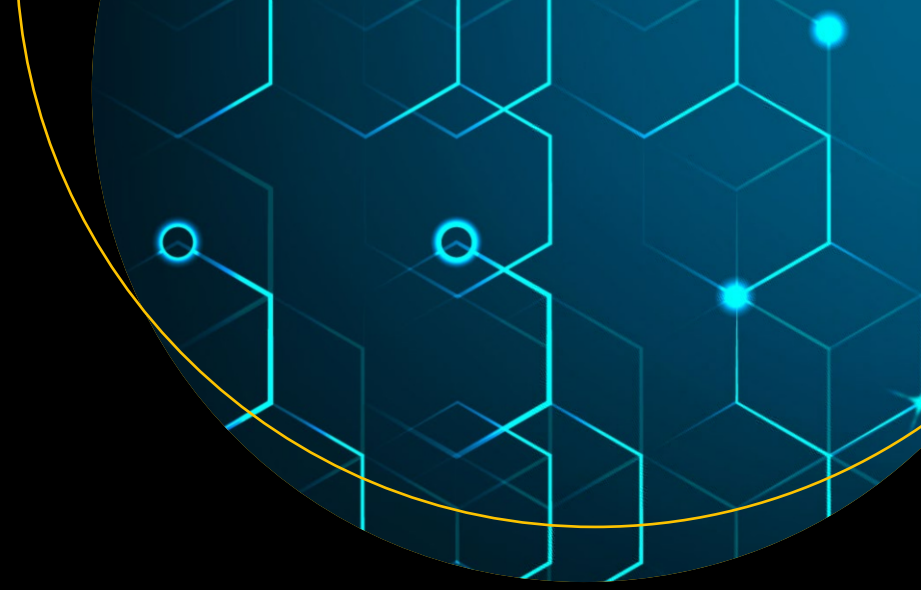

Electron: From Beginner to Pro

Learn to Build Cross Platform Desktop Applications using GitHub's Electron

Chris Griffith
Leif Wells

Apress®

Electron: From Beginner to Pro

Learn to Build Cross Platform Desktop Applications using Github's Electron

Chris Griffith
Leif Wells

Apress®

Electron: From Beginner to Pro: Learn to Build Cross Platform Desktop Applications using Github's Electron

Chris Griffith
San Diego, California, USA

Leif Wells
Atlanta, Georgia, USA

ISBN-13 (pbk): 978-1-4842-2825-8
ISBN-13 (electronic): 978-1-4842-2826-5
https://doi.org/10.1007/978-1-4842-2826-5

Library of Congress Control Number: 2017959877

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Louise Corrigan
Development Editor: James Markham
Technical Reviewer: Lily Madar
Coordinating Editor: Nancy Chen
Copy Editor: Karen Jameson
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484228258. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Contents

About the Authors

Chris Griffith is a User Experience Lead at a home automation and security company and is also an instructor at the University of California, San Diego Extension, teaching mobile application development. He is also an Adobe Community Professional specializing in PhoneGap/Cordova and Experience Design. Chris is regularly invited to speak at conferences such as Fluent, Adobe MAX, and ngConf. He has developed several mobile applications, a variety of code-hinters, and ConfiGAP for PhoneGap Build. In addition, he has served as a technical reviewer for several publications and written for uxmag.com. In his spare time, Chris spends time with his family, sea kayaking, hiking, and drinking craft beer with friends. You can follow him on Twitter @chrisgriffith or at chrisgriffith.wordpress.com.

Leif Wells is a Web and Mobile Application Developer working as a contractor from his home in Atlanta, Georgia. He currently enjoys working with Electron, the Ionic and Angular stack, and has recently become obsessed with automated testing.

His experiences working as a team member on large Enterprise-level projects as well as a single developer on small products have matured him into a seasoned professional. Leif has organized and supported technical communities both online and in Atlanta, and often speaks at conferences and user groups.

Leif enjoys good movies, great sushi, and hanging out with his canine companion, Miss Bonnie. He has been known to blog irregularly at `https://leifwells.github.io/` and can be found on Twitter as @leifwells.

About the Technical Reviewer

Lily Madar is a Creative Technologist from London, UK, who, for the last decade, has worked with various web technologies and frameworks for a range of digital creative and media agencies. Some of her work includes interactive displays powered by web technologies and can be seen in the British Museum or the Serpentine Gallery (both in London).

Outside of work, she is an active hackathon participant with recent wins at TADHack and GeoHack.
She also writes tutorials exploring the latest web and digital trends and runs hardware workshops for beginners.

When not coding, she is experimenting with Arduino circuits, crochet, and other crafts, making her a full-stack developer in hardware, software, and yarn-ware!

CHAPTER 1

Welcome to Electron

GitHub Electron (or simply Electron) allows you to build desktop applications using just HTML, CSS, and JavaScript. Sounds like a pretty ambitious statement to make. But it is indeed true, just as Apache Cordova (also known as PhoneGap) enables you to create mobile applications also with just HTML, CSS, JS, and so does Electron for the desktop.

Originally released in July 2013 by Cheng Zhao, an engineer at Github, it was part of their effort to produce a new code editor, Atom. Initially, the project was known as the Atom Shell but was soon rebranded simply as Electron. Although other solutions existed, this project quickly gained traction within the development community. In fact, Adobe AIR, released back in 2008, originally supported building desktop applications with HTML, CSS, and JavaScript, in addition to ActionScript. So the desire to leverage web technologies beyond the browser is certainly not a new one.

In this book, we will take you through the entire Electron ecosystem from its initial setup, through its key features, like creating native menus and windows and more, and how to deploy our app so it can be distributed to our users. Rather bog you down in understanding some abstract sample applications, we are going to be focusing on the core code needing to make Electron work. So, you don't need to know the latest framework to use Electron, but having some basic knowledge with Node.js is useful.

Here is a brief outline of what we are going to be covering:

- Setting up Electron
- Exploring creating the application's window
- Adding native menus
- Implementing native dialogs
- Learning how to interact with the user's system
- Creating installable and auto-updating applications

So, if you are ready to start learning about Electron, let's get started.

What Is Electron?

Electron is a blend of two incredibly popular technologies: Node.js (or simply Node) and Chromium. Thus, any web application you have written can run on Electron. Similarly, any Node application you have written can run on Electron. But the power of Electron is that you can use both solutions together.

This book is about how to use these two technologies together to create rich and engaging desktop applications. For example, we have been developing a simple desktop application that will assist developers generate their manifest.json file for their Progressive Web Apps. For those unfamiliar with Progressive Web Apps (PWAs), they are web apps that use modern web capabilities to deliver native app-like experiences

C. Griffith, L. Wells, *Electron: From Beginner to Pro*, https://doi.org/10.1007/978-1-4842-2826-5_1

within the browser. We could have simply written a Node script that developers could run from the command line. But instead we leverage Electron to create a more compelling desktop application. It is one that allows you to auto-generate the app icons simply by dragging the image on the application, and it will save out the collection for you.

Breaking Electron down into its two components (thankfully the physics naming stopped and we aren't referring to these subparts as quarks), they each have specific functions.

The Node component handles things like file system access, compiled module support, and CommonJS Module support. The Chromium component handles things like rendering HTML and CSS, its own JavaScript engine, and the full suite of Web APIs.

Electron is a straightforward runtime. It is not a massive framework/library like Angular or React, but rather a collection of APIs that we can leverage with those or other frameworks. The structure of an Electron application is also open to personal taste. Usually, the UI framework will have more to say about the directory structure than Electron's requirements. However, there are general guidelines that would be wise to follow when developing.

What Is Node?

Node.js was initially released in 2009 as an open source project, enabling developers to create server-side applications using JavaScript. What made this project interesting was that it leveraged Google's newly open sourced V8 engine to act as its JavaScript runtime. Atop of that runtime, the project added APIs for accessing the local file system, creating servers, as well as the ability to load modules.

Node has enjoyed a tremendous surge of popularity from across the development community. As such, there is a huge collection of modules that are available for use within your Electron application.

What Is Chromium?

Chromium is the open source version of Google's Chrome web browser. Although it shares much of the same code base and feature set, there are a few differences (albeit minor) and it is released under a different license. What is included with Electron is technically the Chromium Content Module (CCM). Quite the mouthful, hence why most simply refer it is as Chromium. But what is the Chromium Content Module? It is the core code that makes a web browser a web browser. It includes the Blink rendering engine and its own V8 JavaScript engine. The CCM will handle retrieving and rendering HTML, loading and parsing CSS, and executing JavaScript as well.

The CCM only focuses on the core needs to render a web page. Additional features, like supporting Chrome Extensions, or syncing your Chrome bookmarks, are not supported. Just remember that its core purpose is to render web content.

Who Is Using Electron?

So many open source projects come and go. Is Electron worth investing your time and energy into learn? Yes. Although, Electron's original purpose was to act as the shell for GitHub's Atom editor, companies large and small found it to be a good solution for their development needs. Since it was backed by a recognizable company, the risks were a bit lower than trusting your next big thing on an unproven project. If you go to atom.electron.io you can see a massive collection of applications that have been released with Electron as its core.

Obviously Github is actively supporting it, as it is the foundation of their Atom editor. But who else? The very popular team messaging application Slack is built atop Electron, enabling them to develop a common UI across the operating systems. If Atom is not your code editor of choice, then Microsoft's Visual Studio Code might be. This popular editor is also built atop Electron. This is currently our editor of choice at the moment. The team at Microsoft has leveraged common development languages of HTML, CSS,

and JavaScript to create a very compelling editor tuned for working with TypeScript and more that works across both macOS and Windows.

A variety of familiar web tools have also been able to transform themselves into the desktop-based applications. If you are familiar with Yeoman, a web project generator, there is now a version with a user interface instead of the standard command-line version you are probably familiar with. The team at Basecamp, a popular project management tool, now supports an out of browser experience. If you have worked with Zeplin.io to inspect your visual designs, then the desktop version was developed with Electron. The Postman API inspection tool is another great example of what is possible as an Electron application.

These are just some of the examples of some first-class web applications that have been able to break free from the browser and create desktop-centric versions of their applications. If you would like to explore some other applications that have been built with Electron, visit `https://electron.atom.io/apps/`.

What Do I Need to Know?

Unlike traditional desktop development, the only skills you need to have to develop with Electron are a good understanding of HTML, CSS, and JavaScript, and a touch of Node. Being comfortable with your command line wouldn't hurt either. The fact that we can leverage our existing skills and take them from the browser on to the desktop is what is exciting about Electron. We will be using Git to seed our starter Electron apps, but nothing more than that is needed. But working with a version control system is always a recommended skill.

This book is going to take a slightly different approach to covering how Electron works. Since it is simply a runtime, it is framework agnostic. So rather than working through an application built in the framework that you don't know, we are going to just stick with vanilla JavaScript. Now, you should have a modest understanding of HTML and CSS. As for your JavaScript skills, if you have a general understanding of modern JavaScript (aka ES6), you will be fine.

Another area that can be helpful to have is some experience with Node. We will be using the module system throughout this book. But we will provide some foundations on these and any advanced topics that we might need to cover in this book.

Why Should I Choose Electron?

We can assume by the fact you have bought this book, that either there is a need to build a desktop application for yourself, a client or your employer, or you are simply curious about it.

If you have done any web application developing, you no doubt understand the challenges of having to support a wide range of browsers, each with different levels of standards support. Don't get us wrong, the browser's standard support has come a long way in recent years. But, there are still workarounds and polyfills needed to properly deploy a web application to the world. For those working with enterprise clients, you may be further handicapped to legacy browsers and operating systems. When you create an Electron application, you embed a copy of the Chromium engine with the application, so you know exactly what features your application and support have and how your content will render. For example, if you want to use Flexbox as part of your layout solution, you safely can do so (Figure 1-1). If using the Service Worker or Fetch API is something needed for your application, you only need to make sure that the build Electron supports it.

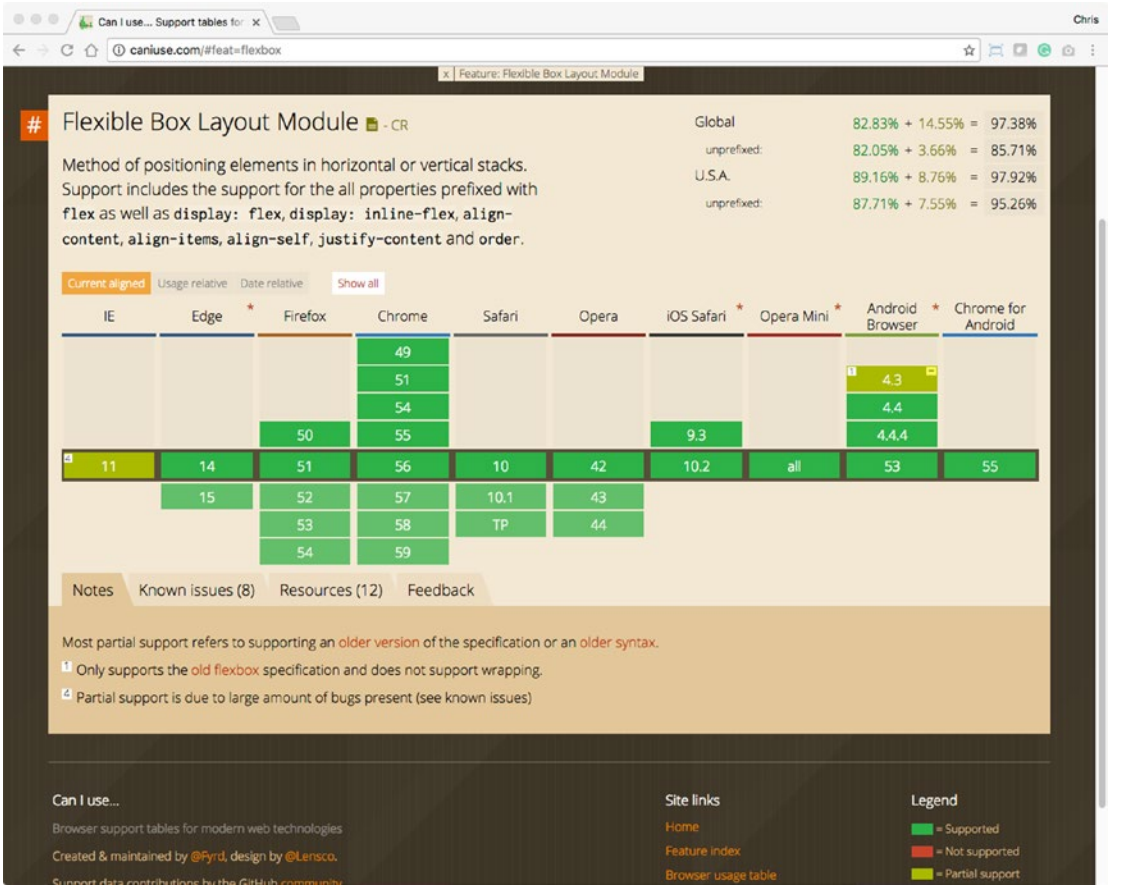

Figure 1-1. *The FlexBox support table from caniuse.com*

No longer will referencing a feature on caniuse.com be disappointing but rather one of possibilities.

As a general rule, Electron updates its Chromium module about two weeks after it is released to the public. The Node component typically takes a bit longer to update. As you begin to embark on larger Electron projects, you will want to also monitor the development process of both of these projects. There might be an issue that you need to be aware or feature added that can greatly make your life easier. But, don't worry – once you can package your application, those runtimes are baked into your application.

Electron's Advantages

Electron applications are just like any other desktop application as they are installed locally on the user's hard drive. They can be launched directly from the Windows taskbar or from the OSX Dock, and there is no need to launch a browser and navigate to some url to run your application. When you need to open or save a file, these dialogs are native in appearance and interaction. Your Electron application can support full drag-and-drop interaction with a local file system, or even associate itself with a file type, so when a user double-clicks the associated file your app will open.

We also have the ability have custom application menus that conform to each platform's user interface guidelines. Contextual menus are available that allow your user to control-click or right-click to display your custom menu. We will show you how to add this functionality in a later chapter.

If you need to trigger a system-wide notification, we can leverage Chromium's Notification API to do so. Electron will go even further that traditional window desktop applications, and create applications that only live in the menubar or system tray.

Electron provides a solid framework that will allow you to develop first-class desktop applications.

Beyond the Sandbox

If you have ever worked with an external API, then you are probably familiar with the restrictions that you have to work. We all have fought with Cross Origin Resource Sharing issues, or establishing proxies in order to allow our web application to work correctly.

Electron operates in a much looser environment with regard to security than your browser. The general assumption is that the user has actively chosen to install and run this application. As such, a degree of trust is then assumed between the user and application.

This allows our Electron application much more freedom, but at the same time we have to use this power with caution.

Offline First Design

With typical web application development, you can usually assume the user is online. Now this is changing with the increase in Progressive Web Apps, but some level of online capability is there for your web app to function. Electron applications have to take the opposite approach. You should not assume that you have an Internet connection. In fact, portions of this chapter were written at 35,000 feet on a plane without WiFi. But I was still able to write in a completely offline mode. Even if your application is dependent on communicating with a back end, you can design your application to function in an offline mode, and sync the data once a connection is reestablished. You will need to take some time to consider how this design pattern will affect the interaction and development of your Electron application.

How Does Electron Work?

Electron-based applications run in two distinct processes: the main process and the render process (Figure 1-2). Each of these two processes has a specific responsibility within your application. While Electron provides a good collection of Node modules for use within your application, many of these modules are only available within a specific process. Knowing these restrictions will help you design the code structure of your application. For example, access to the operating system APIs are restricted to just the main process, and access to the system's clipboard is available to both the main and render process. Knowing this dual-process structure is important, as it will define where some aspects of your application's code need to reside.

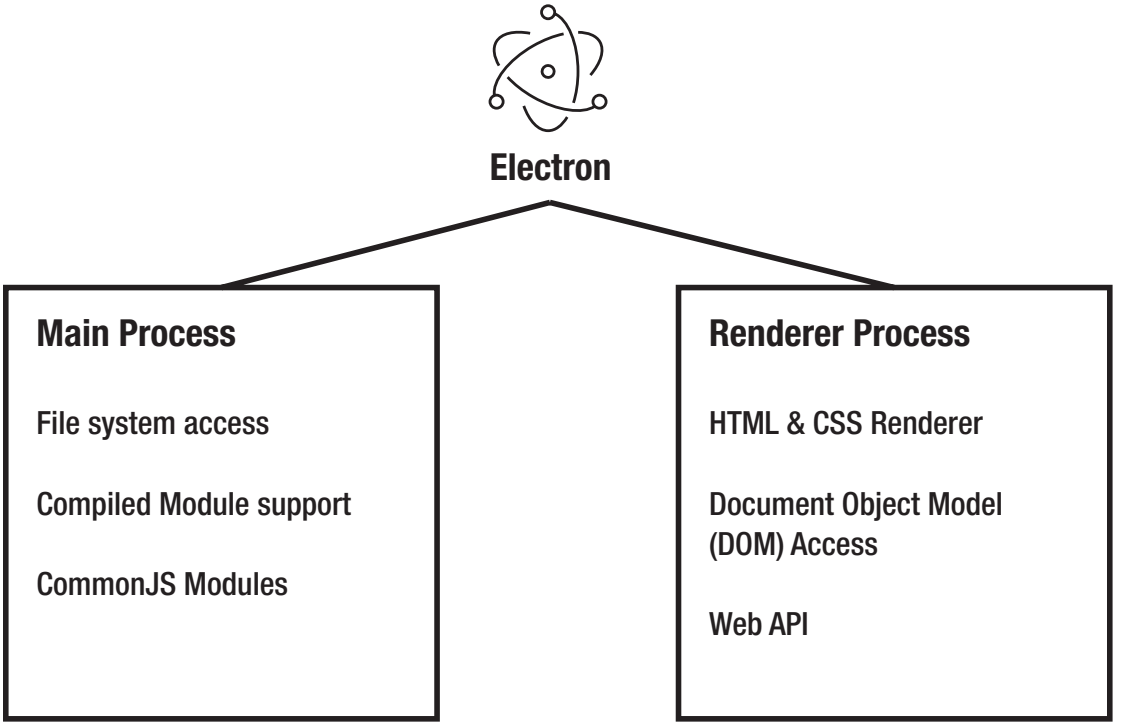

Figure 1-2. *The two processes that power an Electron application*

The Main Process

Within the main process is where your application will handle various system-level activities, like life-cycle events (starting up, preparing to quit, actually quitting, switching between the foreground and background, as just a few examples). This is also the process where application menus are created, as well as any native dialogs, like file open or file save. Our main process is what is used to bootstrap our application. This is the JavaScript file that is referenced within our package.json file, but more on that in the later chapters.

The Render Process

The main process also has another responsibility, which is to spawn any render processes. It is these processes that would display the UI of your application. Each of these render processes will load your content and execute it within its own thread. We can spawn as many windows as we need for our application. Now unlike a standard web app, each of these processes has access to all the installed Node modules, giving you a lot of capabilities.

The render process is isolated from any interaction with any system-level events. Remember, those interactions must be done within the main process. However, Electron does include an interprocess communication system to allow for events and data to be passed back and forth between the main and any renderer process.

One last thing to note, your Electron app actually does not need to have a render process, but it most likely will. This is a perfect option for taking your Node scripts and making them friendlier to use.

Other Solutions

Electron is not the only solution that will enable you to take your web content and enable it to become a desktop application. The most common alternative to using Electron is known as NW.js (originally known as node-webkit). These two projects share some common legacy, remember Cheng Zhao? Well before creating Electron, he was actively involved with the node-webkit project.

Table 1-1 lists some key differences between the projects.

Table 1-1. *Project differences*

	Electron	NW.JS
Chromium Type	Current build of Chromium	A forked version of Chromium
Node Process design	Separate Node processes	Shared Node process
Auto-Updating	Built-in API	Not included
Crash Reporting	Built-in API	Not included
Windows Support	Windows 7 or later	Windows XP or later

Some of the key takeaways from this table are the fact that NW.js uses a forked (or copy of the original code) version of Chromium. This may introduce issues such as standards support or delays in improvements or fixes within the Chromium module. Some use functions like Auto-Updating and Crash Reporting must be handled with your own custom solution, rather than leveraging the built-in APIs. The Node process design is also worth noting. Since Electron uses separate processes, it should be more performant than an NW.js application that must share the Node process. One of NW.js' advantages is the fact it supports a much older version of Windows. If your target audience might include that legacy operating system, then NW.js would be your only option between the two.

Summary

This chapter has given you a general overview of Electron. We have touched on its two core technologies: Node and Chromium, as well as introduced its dual-process design. You should have an initial sense of what an Electron-based application is capable of.

In the coming chapters, we will begin exploring these capabilities in much more detail, as well as some we did not even mention yet.

CHAPTER 2

Installing Electron

Getting your work environment configured to use Electron is fairly straightforward, but there are a couple of items required to get you started. If you are an experienced developer, you probably already have Node and Git installed. If so, feel free to skip to the Install Electron section of this chapter. If not, let's get started by installing Node.

Before Installing

These days, people install new programs on their computers and devices every day without thinking about it. While all of the programs you need to install to work with Electron are safe, any time you wish to install programs on your computer, you should always ensure that you have completed a backup of your computer. Better safe than sorry.

Installing Node

Node is the biggest thing to happen to JavaScript this century. Node is a runtime built with JavaScript that is being used by everyone from hobbyists to Enterprise developers to program anything from Internet of Things (IoT) devices to servers. JavaScript developers use Node daily to assist in the automation of their daily work. Electron uses Node to create cross platform desktop applications.

To install Node, you need to head over to `http://nodejs.org` and download Node using the easily identifiable download buttons on their site (Figure 2-1).

C. Griffith, L. Wells, *Electron: From Beginner to Pro*, https://doi.org/10.1007/978-1-4842-2826-5_2

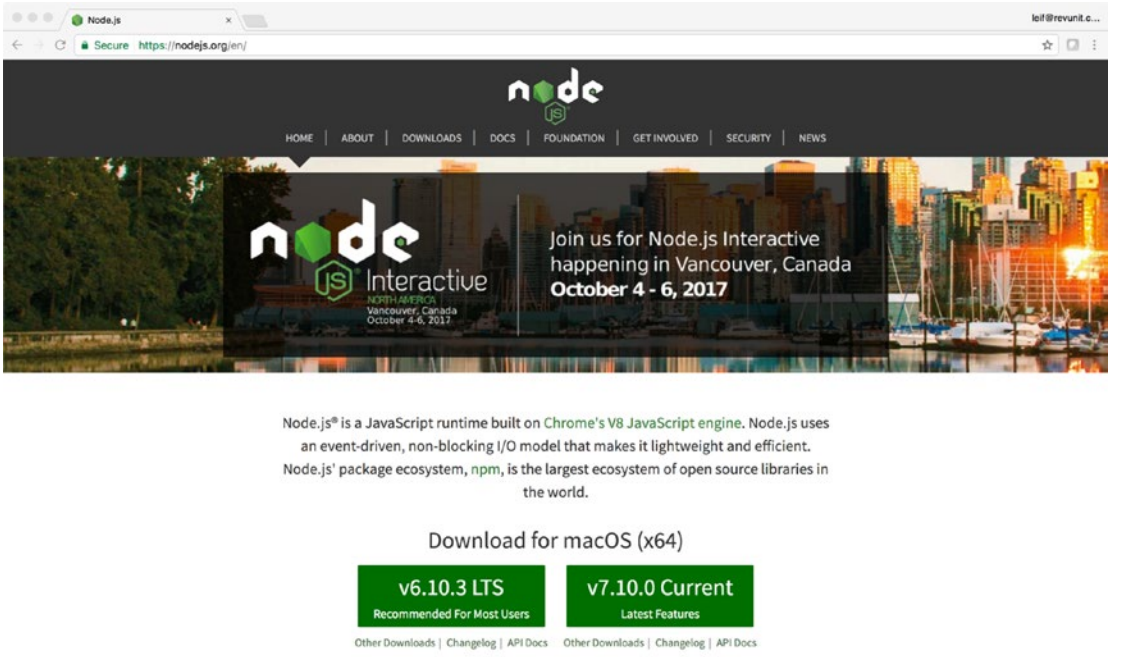

Figure 2-1. *The Node.js Website*

As you can see in this screenshot, there are two buttons available: one for the "LTS" version and another for the "Current" version. "LTS" stands for Long Term Support, meaning that the maintainers of Node decided that version 6 had reached a point of stability that everyone could rely upon; and no more development updates, beyond critical bug fixes and security updates, would be added. They did this so that development on the newer version, the one labeled "Current" could begin in earnest. While the current version can work for you, we are using the LTS version at the time of writing this book. Regardless of that, you need to be aware of your choices in this regard.

Please note: We are also citing version numbers in this chapter at the time of our writing this book. The software you need to install, specifically Node and Electron, are fast-moving projects that are updated regularly. The version numbers cited here may not match the available version numbers at the time of your reading.

Currently, Electron version 1.6.6 ships a version of Node, version 7.4.0, which is slightly behind the currently available version 7.10.0. So what does this mean to you? If there are features of Node 7.4.0 that you would like to use with your Electron app, you should download and install the current version of Node, and be aware that there may be features of 7.10.0 that will not be available in your application when you distribute it.

As mentioned before, for the purposes of this book we will be installing the LTS version of Node.

Installing Node for macOS

Download the LTS version of Node from the Node Website (`http://nodejs.org`), locate the downloaded file, and double-click it. This is a fairly simple installer. Follow the instructions provided and you will install Node (Figure 2-2).

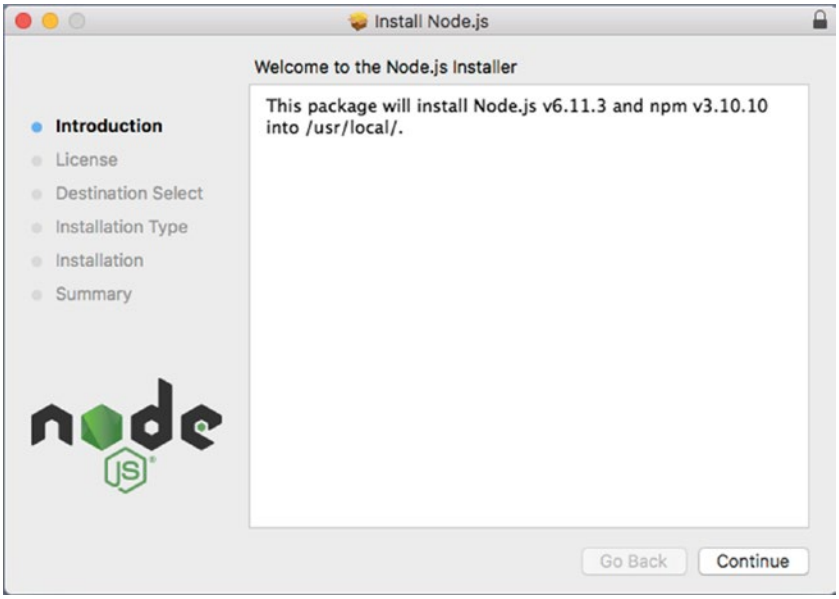

Figure 2-2. *First screen of the Node installer*

Every software that you install these days has to have a Software License Agreement (Figure 2-3). Read it (or not, we won't tell) and hit "Continue," and then click the "Agree" button of the overlaying window that appears.

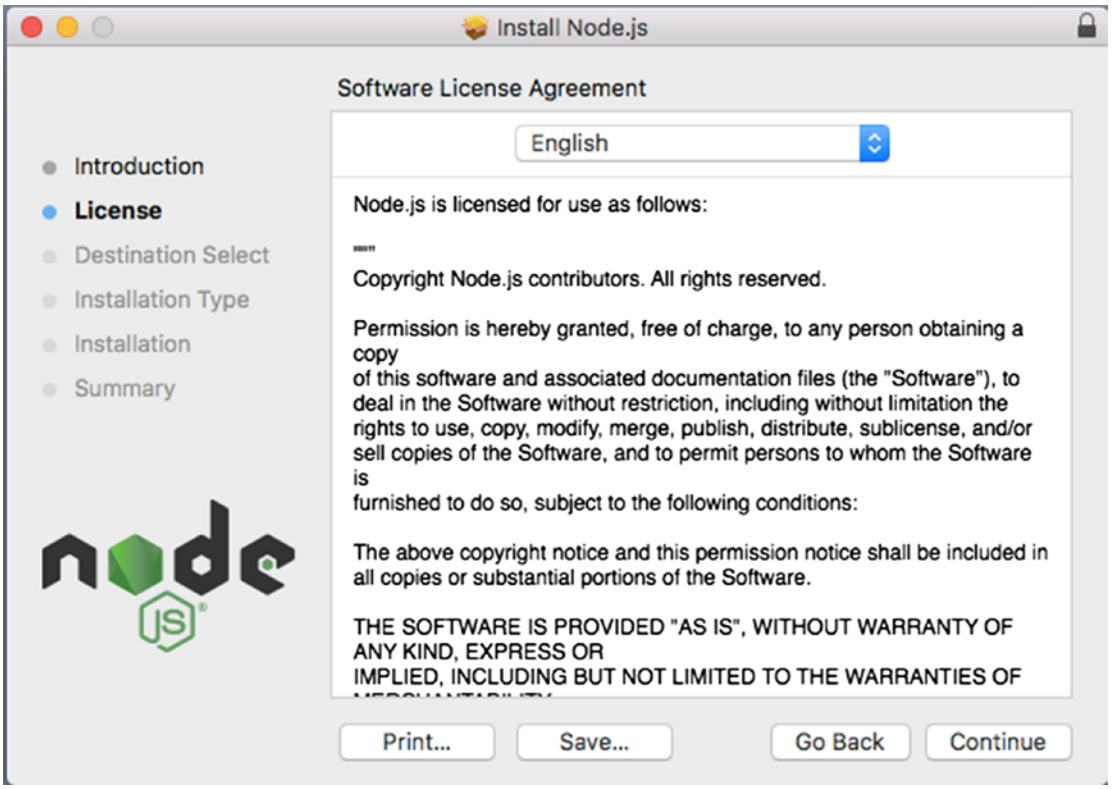

Figure 2-3. *The Software License Agreement*

Select “Install for all users of this computer,” and then click “Continue” here (Figure 2-4) as we need to install Node for all users.

Figure 2-4. *Select where to install Node*

Finally, we are ready to install (Figure 2-5). At this point, when you click “Install” you will be asked for your system’s admin password.

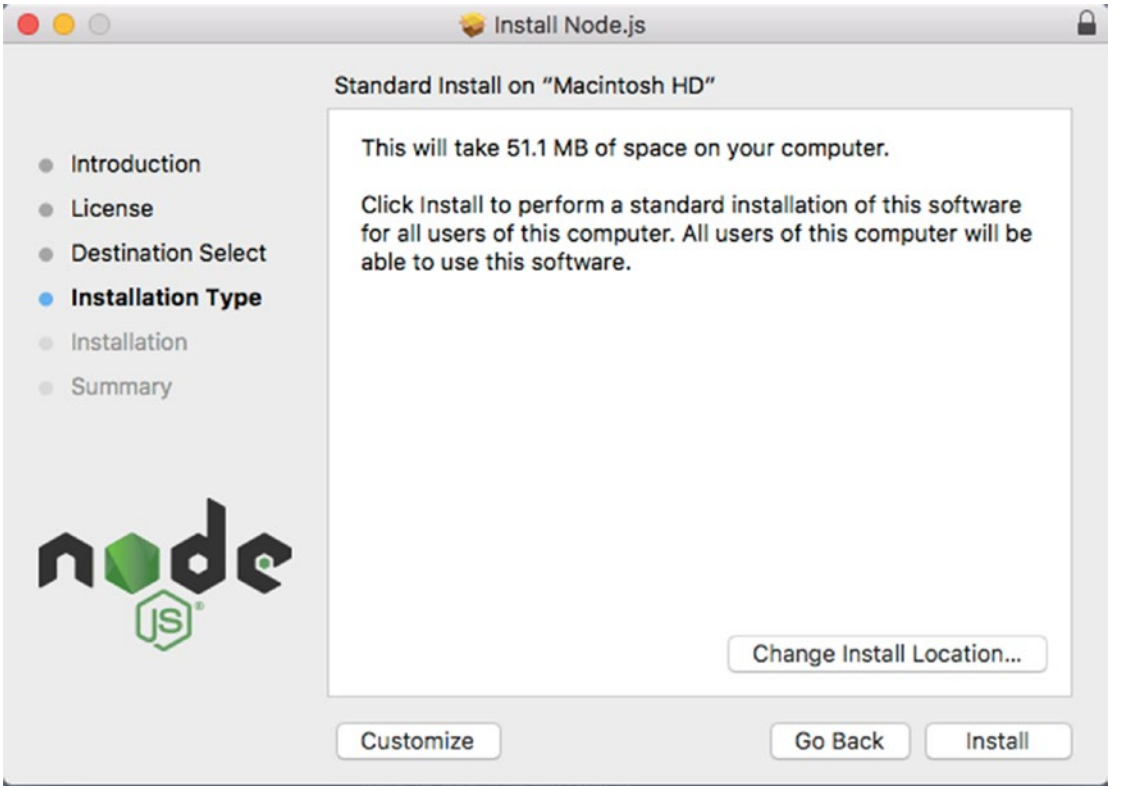

Figure 2-5. *Selecting your installation type*

The installer needs this so that it can install Node into a protected area of your operating system. Once the password is entered, you are off to the races.

Figure 2-6 shows the final screen. You've done it! To test this out, let's open up the terminal application and test the version using the node –version command. You should see the version number you installed.

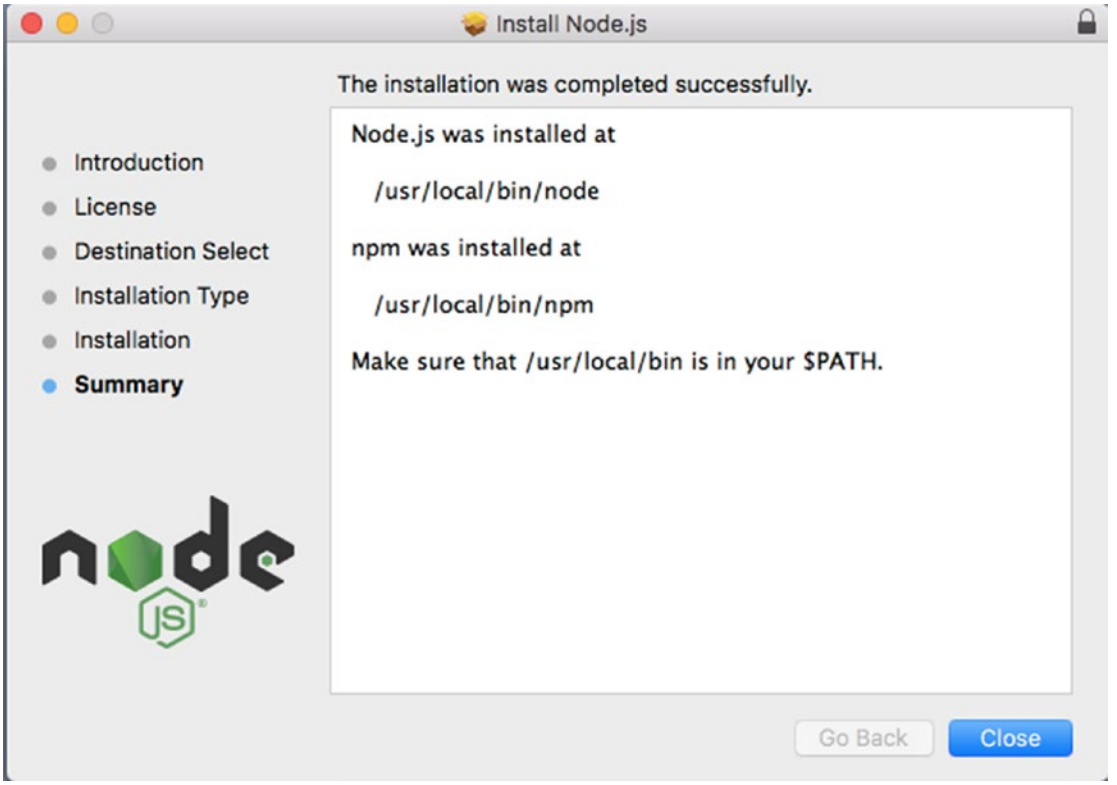

Figure 2-6. *Installation is complete*

Installing Node on Windows

The installation process for Node on Windows is very similar to the process for macOS.

From the Node Website (`http://nodejs.org`), download the LTS version of Node for Windows. Once the file is downloaded, find the downloaded file and double-click it to get the install process started (Figure 2-7).

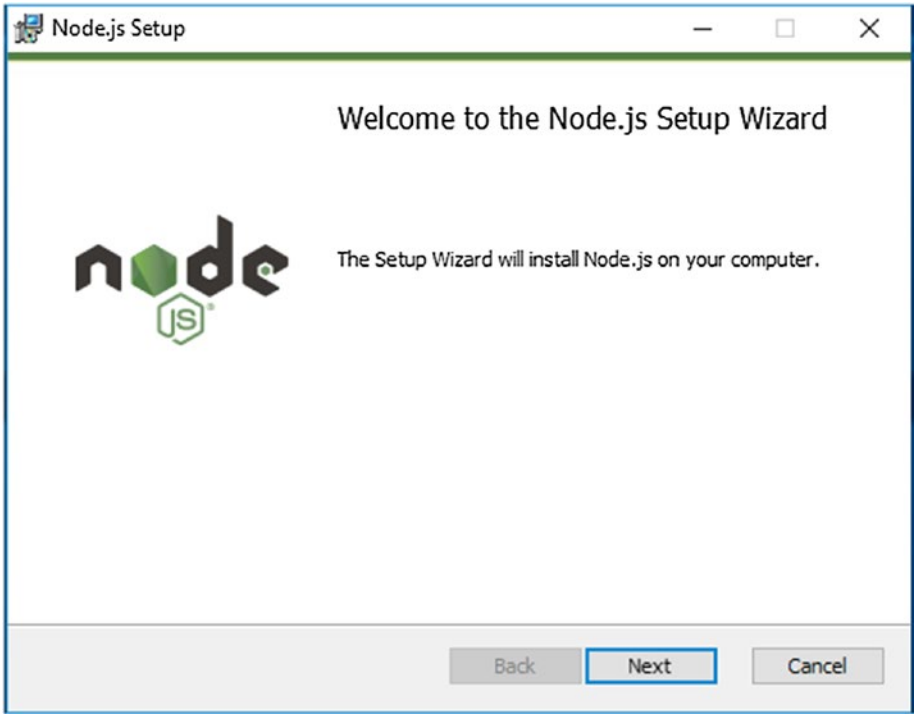

Figure 2-7. *The first screen of the Node for Windows installer*

When you click the Next button, you will see the Software License Agreement (Figure 2-8). Click the check box next to the text "I accept the terms in the License Agreement" and click the Next button to continue.

Figure 2-8. *The End-User License Agreement for Windows*