NW.js Essentials

Using Node.js, we can create web applications easily. Now, thanks to NW.js, we can also create desktop apps with it using a unique combination of HTML5 and Node. NW.js is a runtime application based on Chromium and Node.js.

In this book, you'll discover how to leverage well-known programming languages, such as JavaScript, HTML, and CSS in order to create NW.js desktop applications.

You will implement your first simple application right from the first chapter and see how easy it is to use the platform, after which you will learn about Native UI APIs and the different approaches to Node.js programming.

You’ll get a complete picture of the many possible ways to package and deploy NW.js applications on Microsoft Windows, Mac OS X, and Linux. So, get ready to explore NW.js and build a real, and complex, application.

Who this book is written for
If you are an experienced Node.js developer who wants to create amazing desktop applications using NW.js, this is the book for you. Prior knowledge of HTML5, jQuery, and CSS is assumed.

What you will learn from this book
- Develop a fully working NW.js desktop application from scratch.
- Take advantage of third-party tools to simplify the packaging process of NW.js applications.
- Create an astonishing user experience using Native UI APIs.
- Debug NW.js applications with Chromium DevTools, Crash dump log files, and third-party debugging tools.
- Package and deploy NW.js applications on Microsoft Windows, Mac OS X, and Linux.
- Take advantage of Browser Web APIs, expanding on data persistence solutions and security.
- Avoid common programming mistakes by integrating prior knowledge of Node.js with NW.js development.

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Meet NW.js'
- A synopsis of the book’s content
- More information on NW.js Essentials

About the Author

Alessandro Benoit is a 31-year-old web developer from Italy. He currently works both with backend and frontend technologies, ranging from PHP development, mostly on WordPress and Laravel, to web design and building open source jQuery plugins. He's also an early adopter of NW.js and the developer of Nuwk!, an open source application that simplifies the building process of NW.js applications on Mac OS X.
NW.js Essentials

NW.js is a web app runtime, based on Node.js and the Chromium open source browser project, which enables web developers to build native-like desktop applications for Windows, Mac OS X, or Linux, leveraging all the power of wellknown web technologies such as Node.js, HTML5, and CSS.

In *NW.js Essentials*, you'll be guided through the full development process, starting from the theoretical basis behind NW.js technology to the realization of a fully working, multiplatform desktop application.

What This Book Covers

*Chapter 1, Meet NW.js*, provides a presentation of the NW.js technology followed by a brief digression on how NW.js works under the hood, an analysis of the strengths of the library, and eventually a step-by-step tutorial on building a first, simple "Hello World" application.

*Chapter 2, NW.js Native UI APIs*, takes you through a thorough description of Native UI APIs with examples on how to interact with OS windows, work with menus, run shell commands, and much, much more.

*Chapter 3, Leveraging the Power of Node.js*, gives an introduction of how NW.js development differs from standard server/client programming and a few interesting hints on how to get the maximum out of Node.js within NW.js applications.

*Chapter 4, Data Persistence Solutions and Other Browser Web APIs*, explains more about data handling and takes advantage of Browser Web APIs to build beautiful, usable, and native-like desktop applications.

*Chapter 5, Let's Put It All Together*, builds a fully working application based on many of the concepts you've learned in the previous chapters.


*Chapter 7, Automated Packaging Tools*, takes advantage of third-party tools to simplify the packaging process of NW.js applications.

*Chapter 8, Let's Debug Your Application*, teaches you different ways to debug your application and a few common issues of NW.js.

*Chapter 9, Taking Your Application to the Next Level*, involves a collection of ideas and resources to get the best out of what you've learned in this book.
Meet NW.js

Until a while ago, developing a desktop application that was compatible with the most common operating systems required an enormous amount of expertise, different programming languages, and logics for each platform.

Yet, for a while now, the evolution of web technologies has brought to our browsers many web applications that have nothing to envy from their desktop alternative. Just think of Google apps such as Gmail and Calendar, which, for many, have definitely replaced the need for a local mail client. All of this has been made possible thanks to the amazing potential of the latest implementations of the Browser Web API combined with the incredible flexibility and speed of the latest server technologies.

Although we live in a world increasingly interconnected and dependent on the Internet, there is still the need for developing desktop applications for a number of reasons:

- To overcome the lack of vertical applications based on web technologies
- To implement software solutions where data security is essential and cannot be compromised by exposing data on the Internet
- To make up for any lack of connectivity, even temporary
- Simply because operating systems are still locally installed

Once it's established that we cannot completely get rid of desktop applications and that their implementation on different platforms requires an often prohibitive learning curve, it comes naturally to ask: why not make desktop applications out of the very same technologies used in web development?

The answer, or at least one of the answers, is NW.js!
Meet NW.js

NW.js doesn't need any introduction. With more than 20,000 stars on GitHub (in the top four hottest C++ projects of the repository-hosting service) NW.js is definitely one of the most promising projects to create desktop applications with web technologies. Paraphrasing the description on GitHub, NW.js is a web app runtime that allows the browser DOM to access Node.js modules directly.

Node.js is responsible for hardware and operating system interaction, while the browser serves the graphic interface and implements all the functionalities typical of web applications. Clearly, the use of the two technologies may overlap; for example, if we were to make an asynchronous call to the API of an online service, we could use either a Node.js HTTP client or an XMLHttpRequest Ajax call inside the browser.

Without going into technical details, in order to create desktop applications with NW.js, all you need is a decent understanding of Node.js and some expertise in developing HTML5 web apps.

In this first chapter, we are going to dissect the topic dwelling on these points:

• A brief technical digression on how NW.js works
• An analysis of the pros and cons in order to determine use scenarios
• Downloading and installing NW.js
• Development tools
• Making your first, simple "Hello World" application

Important notes about NW.js (also known as Node-Webkit) and io.js

Before January 2015, since the project was born, NW.js was known as Node-Webkit. Moreover, with Node.js getting a little sluggish, much to the concern of V8 JavaScript engine updates, from version 0.12.0, NW.js is not based on Node.js but on io.js, an npm-compatible platform originally based on Node.js. For the sake of simplicity in the book, we will keep referring to Node.js even when talking about io.js as long as this does not affect a proper comprehension of the subject.
NW.js under the hood

As we stated in the introduction, NW.js, made by Roger Wang of Intel’s Open Source Technology Center (Shanghai office) in 2011, is a web app runtime based on Node.js and the Chromium open source browser project. To understand how it works, we must first analyze its two components:

- Node.js is an efficient JavaScript runtime written in C++ and based on the V8 JavaScript engine developed by Google. Residing in the operating system's application layer, Node.js can access hardware, filesystems, and networking functionalities, enabling its use in a wide range of fields, from the implementation of web servers to the creation of control software for robots. (As we stated in the introduction, NW.js has replaced Node.js with io.js from version 0.12.0.)
- WebKit is a layout engine that allows the rendering of web pages starting from the DOM, a tree of objects representing the web page. NW.js is actually not directly based on WebKit but on Blink, a fork of WebKit developed specifically for the Chromium open source browser project and based on the V8 JavaScript engine as is the case with Node.js.

Since the browser, for security reasons, cannot access the application layer and since Node.js lacks a graphical interface, Roger Wang had the insight of combining the two technologies by creating NW.js.

The following is a simple diagram that shows how Node.js has been combined with WebKit in order to give NW.js applications access to both the GUI and the operating system:
Meet NW.js

In order to integrate the two systems, which, despite speaking the same language, are very different, a couple of tricks have been adopted. In the first place, since they are both event-driven (following a logic of action/reaction rather than a stream of operations), the event processing has been unified. Secondly, the Node context was injected into WebKit so that it can access it.

The amazing thing about it is that you'll be able to program all of your applications' logic in JavaScript with no concerns about where Node.js ends and WebKit begins.

Today, NW.js has reached version 0.12.0 and, although still young, is one of the most promising web app runtimes to develop desktop applications adopting web technologies.

Features and drawbacks of NW.js

Let's check some of the features that characterize NW.js:

- NW.js allows us to realize modern desktop applications using HTML5, CSS3, JS, WebGL, and the full potential of Node.js, including the use of third-party modules
- The Native UI API allows you to implement native lookalike applications with the support of menus, clipboards, tray icons, and file binding
- Since Node.js and WebKit run within the same thread, NW.js has excellent performance
- With NW.js, it is incredibly easy to port existing web applications to desktop applications
- Thanks to the CLI and the presence of third-party tools, it's really easy to debug, package, and deploy applications on Microsoft Windows, Mac OS, and Linux

However, all that glitters is not gold. There are some cons to consider when developing an application with NW.js:

- **Size of the application**: Since a copy of NW.js (70-90 MB) must be distributed along with each application, the size of the application makes it quite expensive compared to native applications. Anyway, if you're concerned about download times, compressing NW.js for distribution will save you about half the size.

- **Difficulties in distributing your application through Mac App Store**: In this book, it will not be discussed (just do a search on Google), but even if the procedure is rather complex, you can distribute your NW.js application through Mac App Store. At the moment, it is not possible to deploy a NW.js application on Windows Store due to the different architecture of .appx applications.
• **Missing support for iOS or Android:** Unlike other SDKs and libraries, at the moment, it is not possible to deploy an NW.js application on iOS or Android, and it does not seem to be possible to do so in the near future. However, the *portability* of the HTML, JavaScript, and CSS code that can be distributed on other platforms with tools such as PhoneGap or TideSDK should be considered. Unfortunately, this is not true for all of the features implemented using Node.js.

• **Stability:** Finally, the platform is still quite young and not bug-free.

**NW.js – usage scenarios**

The flexibility and good performance of NW.js allows its use in countless scenarios, but, for convenience, I'm going to report only a few notable ones:

- Development tools
- Implementation of the GUI around existing CLI tools
- Multimedia applications
- Web services clients
- Video games

The choice of development platform for a new project clearly depends only on the developer; for the overall aim of confronting facts, it may be useful to consider some specific scenarios where the use of NW.js might not be recommended:

- When developing for a specific platform, graphic coherence is essential, and, perhaps, it is necessary to distribute the application through a store
- If the performance factor limits the use of the preceding technologies
- If the application does a massive use of the features provided by the application layer via Node.js and it has to be distributed to mobile devices

**Popular NW.js applications**

After summarizing the pros and cons of NW.js, let's not forget the real strength of the platform—the many applications built on top of NW.js that have already been distributed. We list a few that are worth noting:

- **Wunderlist for Windows 7:** This is a to-do list / schedule management app used by millions
- **Cellist:** This is an HTTP debugging proxy available on Mac App Store
Meet NW.js

- **Game Dev Tycoon**: This is one of the first NW.js games that puts you in the shoes of a 1980s game developer
- **Intel® XDK**: This is an HTML5 cross-platform solution that enables developers to write web and hybrid apps

**Downloading and installing NW.js**

Installing NW.js is pretty simple, but there are many ways to do it. One of the easiest ways is probably to run `npm install nw` from your terminal, but for the educational purposes of the book, we're going to manually download and install it in order to properly understand how it works.

You can find all the download links on the project website at [http://nwjs.io/](http://nwjs.io/) or in the Downloads section on the GitHub project page at [https://github.com/nwjs/nw.js/](https://github.com/nwjs/nw.js/); from here, download the package that fits your operating system.

For example, as I'm writing this book, Node-Webkit is at version 0.12.0, and my operating system is Mac OS X Yosemite 10.10 running on a 64-bit MacBook Pro; so, I'm going to download the `nwjs-v0.12.0-osx-x64.zip` file.

Packages for Mac and Windows are zipped, while those for Linux are in the `tar.gz` format. Decompress the files and proceed, depending on your operating system, as follows.

**Installing NW.js on Mac OS X**

Inside the archive, we're going to find three files:

- **Credits.html**: This contains credits and licenses of all the dependencies of NW.js
- **nwjs.app**: This is the actual NW.js executable
- **nwjc**: This is a CLI tool used to compile your source code in order to protect it

Before v0.12.0, the filename of `nwjc` was `nwsnapshot`.
Currently, the only file that interests us is nwjs.app (the extension might not be displayed depending on the OS configuration). All we have to do is copy this file in the /Applications folder—your main applications folder.

If you'd rather install NW.js using Homebrew Cask, you can simply enter the following command in your terminal:

```
$ brew cask install nw
```

If you are using Homebrew Cask to install NW.js, keep in mind that the Cask repository might not be updated and that the nwjs.app file will be copied in ~/Applications, while a symlink will be created in the /Applications folder.

---

**Installing NW.js on Microsoft Windows**

Inside the Microsoft Windows NW.js package, we will find the following files:

- `credits.html`: This contains the credits and licenses of all NW.js dependencies
- `d3dcompiler_47.dll`: This is the Direct3D library
- `ffmpegsumo.dll`: This is a media library to be included in order to use the `<video>` and `<audio>` tags
- `icudtl.dat`: This is an important network library
- `libEGL.dll`: This is the WebGL and GPU acceleration
- `libGLESv2.dll`: This is the WebGL and GPU acceleration
- `locales/`: This is the languages folder
- `nw.exe`: This is the actual NW.js executable
- `nw.pak`: This is an important JS library
- `pdf.dll`: This library is used by the web engine for printing
- `nwjc.exe`: This is a CLI tool to compile your source code in order to protect it

Some of the files in the folder will be omitted during the final distribution of our application, but for development purposes, we are simply going to copy the whole content of the folder to C:/Tools/nwjs.
Meet NW.js

Installing NW.js on Linux

On Linux, the procedure can be more complex depending on the distribution you use. First, copy the downloaded archive into your home folder if you have not already done so, and then open the terminal and type the following command to unpack the archive (change the version accordingly to the one downloaded):

```
$ gzip -dc nwjs-v0.12.0-linux-x64.tar.gz | tar xf -
```

Now, rename the newly created folder in `nwjs` with the following command:

```
$ mv ~/nwjs-v0.12.0-linux-x64 ~/nwjs
```

Inside the `nwjs` folder, we will find the following files:

- **credits.html**: This contains the credits and licenses of all the dependencies of NW.js
- **icudtl.dat**: This is an important network library
- **libffmpegsumo.so**: This is a media library to be included in order to use the `<video>` and `<audio>` tags
- **locales/**: This is a languages folder
- **nw**: This is the actual NW.js executable
- **nw.pak**: This is an important JS library
- **nwjc**: This is a CLI tool to compile your source code in order to protect it

Open the folder inside the terminal and try to run NW.js by typing the following:

```
$ cd nwjs
$ ./nw
```

If you get the following error, you are probably using a version of Ubuntu later than 13.04, Fedora later than 18, or another Linux distribution that uses `libudev.so.1` instead of `libudev.so.0`: otherwise, you're good to go to the next step:

```
error while loading shared libraries: libudev.so.0: cannot open shared object file: No such file or directory
```

Until NW.js is updated to support `libudev.so.1`, there are several solutions to solve the problem. For me, the easiest solution is to type the following terminal command inside the directory containing `nw`:

```
$ sed -i 's/udev\.so\.0/udev.so.1/g' nw
```

---

[8]
This will replace the string related to libudev, within the application code, with the new version. The process may take a while, so wait for the terminal to return the cursor before attempting to enter the following:

$ ./nw

Eventually, the NW.js window should open properly.

Development tools
As you'll make use of third-party modules of Node.js, you're going to need npm in order to download and install all the dependencies; so, Node.js (http://nodejs.org/) or io.js (https://iojs.org/) must be obviously installed in your development environment.

I know you cannot wait to write your first application, but before you start, I would like to introduce you to Sublime Text 2. It is a simple but sophisticated IDE, which, thanks to the support for custom build scripts, allows you to run (and debug) NW.js applications from inside the editor itself.

If I wasn't convincing and you'd rather keep using your favorite IDE, you can skip to the next section; otherwise, follow these steps to install and configure Sublime Text 2:


2. Open it and from the top menu, navigate to Tools | Build System | New Build System.

3. A new edit screen will open; paste the following code depending on your platform:

   ° On Mac OS X:

   ```json
   {  
     "cmd": ["nwjs", "--enable-logging",  
             "$\{project_path}:$\{file_path}\"],  
     "working_dir": "$\{project_path}:$\{file_path}\"",  
     "path": "/Applications/nwjs.app/Contents/MacOS/"
   }
   
   ° On Microsoft Windows:

   ```json
   {  
     "cmd": ["nw.exe", "--enable-logging",  
             "$\{project_path}:$\{file_path}\"],  
     "working_dir": "$\{project_path}:$\{file_path}\"",  
     "path": "C:/Tools/nwjs/"
   }
   ```
"shell": true
}

   °  On Linux:
{
   "cmd": ["nw", "--enable-logging",
      "${project_path:${file_path}}"],
   "working_dir": "${project_path:${file_path}}",
   "path": "/home/username/nwjs/"
}

4. Type Ctrl + S (Cmd + S on Mac) and save the file as nw-js.sublime-build.

Perfect! Now you are ready to run your applications directly from the IDE.

There are a lot of packages, such as SublimeLinter, LiveReload, and Node.js code completion, available to Sublime Text 2. In order to install them, you have to install Package Control first. Just open https://sublime.wbond.net/installation and follow the instructions.

Writing and running your first "Hello World" app

Finally, we are ready to write our first simple application. We're going to revisit the usual "Hello World" application by making use of a Node.js module for markdown parsing.

"Markdown is a plain text formatting syntax designed so that it can be converted to HTML and many other formats using a tool by the same name."

– Wikipedia

Let's create a Hello World folder and open it in Sublime Text 2 or in your favorite IDE. Now open a new package.json file and type in the following JSON code:

```json
{
   "name": "nw-hello-world",
   "main": "index.html",
   "dependencies": {
      "markdown": "0.5.x"
   }
}
```
Chapter 1

The package.json manifest file, as you’ll see in Chapter 6, Packaging Your Application for Distribution, is essential for distribution as it determines many of the window properties and primary information about the application. Moreover, during the development process, you’ll be able to declare all of the dependencies.

In this specific case, we are going to assign the application name, the main file, and obviously our dependency, the markdown module, written by Dominic Baggott.

If you so wish, you can create the package.json manifest file using the npm init command from the terminal as you’re probably used to already when creating npm packages.

Once you’ve saved the package.json file, create an index.html file that will be used as the main application file and type in the following code:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Hello World!</title>
</head>
<body>
    <script>
        <!--Here goes your code-->
    </script>
</body>
</html>
```

As you can see, it's a very common HTML5 boilerplate. Inside the script tag, let's add the following:

```javascript
var markdown = require("markdown").markdown,
    div = document.createElement("div"),
    content = "#Hello World!\n" +
    "We are using **io.js** " +
    "version **" + process.version + "**;

div.innerHTML = markdown.toHTML(content);
document.body.appendChild(div);
```
Meet NW.js

What we do here is require the markdown module and then parse the content variable through it. To keep it as simple as possible, I’ve been using Vanilla JavaScript to output the parsed HTML to the screen. In the highlighted line of code, you may have noticed that we are using `process.version`, a property that is a part of the Node.js context.

If you try to open `index.html` in a browser, you’d get the `Reference Error: require is not defined` error as Node.js has not been injected into the WebKit process.

Once you have saved the `index.html` file, all that is left is to install the dependencies by running the following command from the terminal inside the project folder:

```
$ npm install
```

And we are ready to run our first application!

**Running NW.js applications on Sublime Text 2**

If you opted for Sublime Text 2 and followed the procedure in the development tools section, simply navigate to **Project | Save Project As** and save the `hello-world.sublime-project` file inside the project folder.

Now, in the top menu, navigate to **Tools | Build System** and select **nw-js**. Finally, press `Ctrl + B` (or `Cmd + B` on Mac) to run the program.

If you have opted for a different IDE, just follow the upcoming steps depending on your operating system.

**Running NW.js applications on Microsoft Windows**

Open the command prompt and type:

```
C:\> c:\Tools\nwjs\nw.exe c:\path\to\the\project\package.json
```

On Microsoft Windows, you can also drag the folder containing `package.json` to `nw.exe` in order to open it.
Running NW.js applications on Mac OS

Open the terminal and type:

$ /Applications/nwjs.app/Contents/MacOS/nwjs /path/to/the/project/

Or, if running NW.js applications inside the directory containing package.json, type:

$ /Applications/nwjs.app/Contents/MacOS/nwjs .

As you can see in Mac OS X, the NW.js kit's executable binary is in a hidden directory within the .app file.

Running NW.js applications on Linux

Open the terminal and type:

$ ~/nwjs/nw /path/to/the/project/

Or, if running NW.js applications inside the directory containing package.json, type:

$ ~/nwjs/nw .

Running the application, you may notice that a few errors are thrown depending on your platform. As I stated in the pros and cons section, NW.js is still young, so that's quite normal, and probably we're talking about minor issues. However, you can search in the NW.js GitHub issues page in order to check whether they've already been reported; otherwise, open a new issue—your help would be much appreciated.
Now, regardless of the operating system, a window similar to the following one should appear:

![Hello World!](image)

**Hello World!**

*We are using js version v1.2.0*

As illustrated, the `process.version` object variable has been printed properly as Node.js has correctly been injected and can be accessed from the DOM.

Perhaps, the result is a little different than what you expected since the *top navigation bar of Chromium* is visible. Do not worry! You can get rid of the navigation bar at any time simply by adding the `window.toolbar = false` parameter to the manifest file, but for now, it’s important that the bar is visible in order to debug the application.

**Summary**

In this chapter, you discovered how NW.js works under the hood, the recommended tools for development, a few usage scenarios of the library, and eventually, how to run your first, simple application using third-party modules of Node.js. I really hope I haven't bored you too much with the theoretical concepts underlying the functioning of NW.js; I really did my best to keep it short.

In the following chapter, we will be much more operational and discover how to interact with the operating system thanks to the **Native UI APIs** of NW.js. In Chapter 2, *NW.js Native UI APIs*, you will indeed discover how to handle *multiple windows*, work with *file dialogs*, create *custom menus*, and much, much more!
Where to buy this book

You can buy NW.js Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.