

Develop Desktop Application Using
HTML CSS & JavaScript

**Quick
Desktop Application
Development
Using Electron**

By Sandeep Kumar Patel



ELECTRON

Quick Desktop Application Development Using Electron

Copyright Blurb

All rights reserved. No part of this book may be reproduced in any form or by any electronic or mechanical means including information storage and retrieval systems – except in the case of brief quotations in articles or reviews – without the permission in writing from its publisher, **Sandeep Kumar Patel**.

All brand names and product names used in this book are trademarks, registered trademarks, or trade names of their respective holders. I am not associated with any product or vendor in this book.

Published By

Sandeep Kumar Patel.

Table of Content

chapter 1

Introduction to Electron

- Desktop Application
- Available Frameworks
- What is Electron
- Electron Building blocks
- Electron Processes
- Electron Installation
- Developing First Electron Application
- Summary

chapter 2

Electron with AngularJS Integration

- Introduction
- AngularJS Installation
- Working with AngularJS
- Working with Angular Router
- Summary

chapter 3

Working with Browser Window

- Introduction
- BrowserWindow Properties
- BrowserWindow Methods
- WebContent
- WebContents Methods
- Example
- Summary

chapter 4

Working with Menu

Electron Default Menu

Custom Menu

Developing a Custom Menu

Summary

chapter 5

Working with Tray Icon Menu

Tray Icon

Tray Methods

Developing a Custom Tray Icon Menu

Summary

chapter 6

Working with Remote Object

Remote Object

Remote Property and Methods

Integrating Bootstrap3

Example of Remote Object

Summary

chapter 7

IPC: Inter Process Communication

Renderer Process IPC

Main Process IPC

Example of IPC

Summary

chapter 8

Electron Dialog Window

Dialog Window

Open Dialog Window

Save Dialog Window

Message Dialog Window
Error Dialog Window
Example of Dialog Window
Summary

chapter 9

Working with Shell and Screen

Screen
Screen Example
Shell
Shell Example
Summary

chapter 10

Electron Global Shortcut Registration

Global Shortcuts
Global Shortcut Methods
Global Shortcut Example
Summary

chapter 11

Electron Node Notification

Desktop Notification
Node Notifier Module
Desktop Notification Example
Summary

chapter 12

Implementing Persistence Using Loki Database

LokiJS Database
Installing LokiJS
Implementing LokiJS with Electron
Summary

chapter 13

Working with Power Monitor

Power Monitor

Power Monitor Events

Power Monitor Example

Summary

chapter 14

Working with Clipboard

Clipboard

Clipboard Methods

Clipboard Example

Summary

About The Author

One Last Thing...

1

Introduction to Electron

In this chapter we will learn about the Electron to develop desktop application using HTML, css and javascript. We will cover following topics in this chapter.

- What is electron?
- Electron Building blocks.
- Electron Processes.
- Main Process
- Renderer Process
- Installing Electron.
- Developing First Electron Application

Desktop Application

A desktop application is a standalone piece of software which runs in desktop or laptop. These desktop applications are developed in languages like VB, C#, java etc. Desktop applications are popular among user due to the following benefits:

- Applications are available offline.
- Application has faster response.
- It provides rich user experience.

In the presents days most of the application web based and run on the cloud. These web applications are built using HTML, CSS and Javascript programming language. The shift t in in application development for web has hampered the development resources available for desktop.

What if we can use the same technologies to build a desktop application? Yes it is possible. We can develop a desktop application using HTML, CSS and javascript. We will be learning desktop application development using these web technologies throughout this book.

Available Frameworks

Desktop application can be developed using web technologies like HTML, CSS and JavaScript with a desktop application framework. The most notable frameworks that exist at presents are listed as follows:

- **NW.js:** This is an application runtime based on **Chromium** and **Node.js**. It also lets us to call Node.js modules directly from the DOM and enables a new way of writing native applications with all Web technologies.

You can find more information about NW.js in the following URL:
<https://github.com/nwjs/nw.js/>

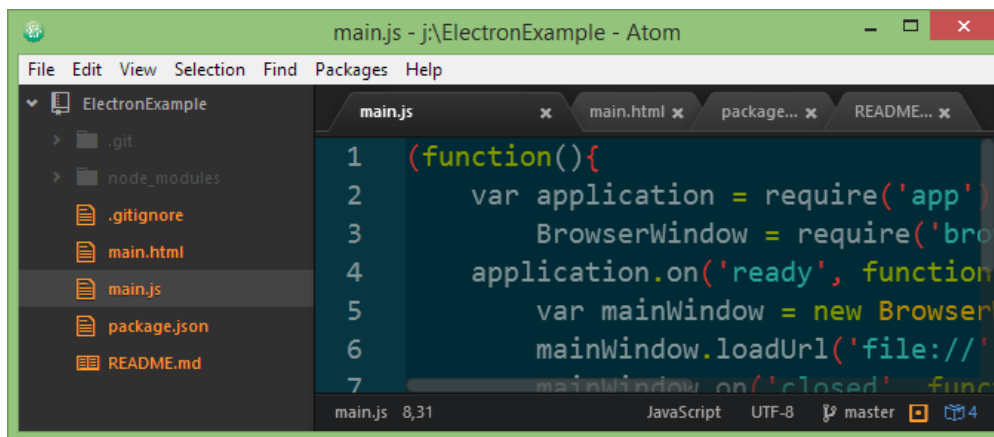
- **Bracket shell:** This project is mainly developed to create Bracket editor and maintained by Adobe. Adobe officially does not provide support to create a desktop application. However we can use it to create desktop application

You can find more information about Bracket Shell in the following URL:
<https://github.com/adobe/brackets-shell>

- **Electron:** This framework is also known as Atom shell and used to develop the Github Atom editor. This book is all about using Electron for building a desktop application. Throughout this book we will explore different features of Electron framework.

What is Electron

Electron is the underlying framework for Github Atom editor. Initially Electron was designed for Atom editor. The following screenshot shows the atom editor.



From the previous screenshot we can observe that it looks like a native desktop application while in the background it uses the Electron framework with web technologies like HTML, CSS and JavaScript. We can find the frame, menus, and status bar similar to native windows.

There are bunch of other desktop application those are built on Electron framework and web technologies. To get motivated with Electron check out few applications built on it. Some of the notable applications are as follows:

No	Application Name	Link
1	Slack client for Mac	https://slack.com/
2	Visual Studio Code	https://code.visualstudio.com/
3	Nuclide IDE	https://github.com/facebook/nuclide
4	Wagon SQL Editor	http://www.wagonhq.com/
5	Nylas Email Client	http://www.nylas.com/blog/splitting-the-atom

Electron Building blocks

The following image shows the two important building block of Electron which empower the developers to create cross platform desktop application.



Electron is made up of 2 popular projects Chromium and io.js library. The role of each blocks are listed as follows.

- **Chromium:** Chromium is the most famous browser project form Google team. It is an open source project. Chromium takes care of rendering the application.

You can find more information about chromium project in the following URL:
<http://www.chromium.org/Home>

- **IO js:** This is a JavaScript platform built on Chrome's V8 runtime. Initially IO js started with a fork of Node js library. IO js is almost compatible with all NPM module. IO js takes care of accessing native platform features as API methods.

You can find more information about IO js project in the following URL:
<https://iojs.org/en/index.html>

Electron Processes

The desktop application developed by Electron framework is mainly controlled by 2 different processes. These 2 processes are listed as follows:

- **Main Proccs:** This is the main process in an Electron application and responsible for creating web pages using **BrowserWindow** instances and a resource URL. In other word the main process is the Browser process.

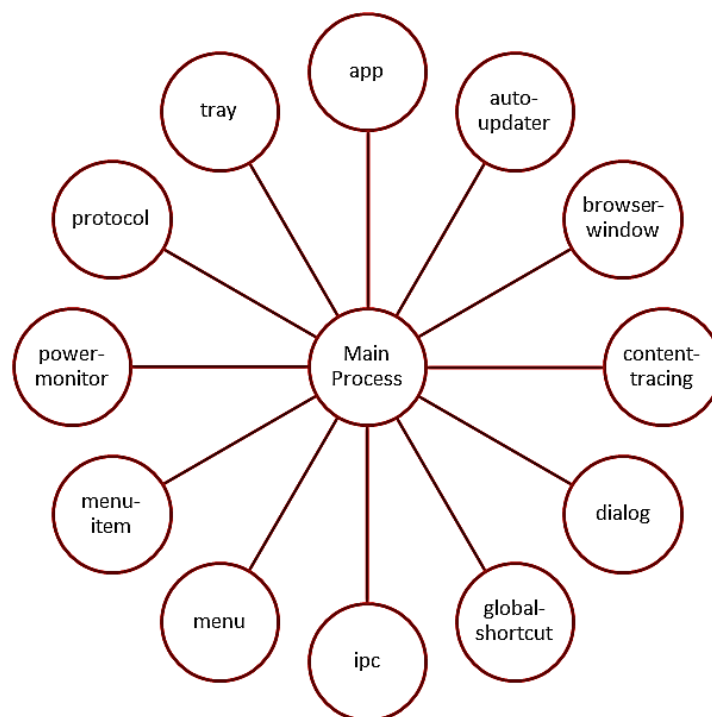
- **Renderer Process:** This process is responsible for displaying web pages in **BrowserWindow**. It uses Chromium multi process architecture. Each web page has their own renderer process.

We can find more information about Chromium multi process architecture using the following link <http://dev.chromium.org/developers/design-documents/multi-process-architecture> .

Main process is the top level process which control the web pages and the associated renderer process. These web pages can access system level resource using **IO.js** API.

Main Process Modules

Main process controls the Electron application start-up and creates the BrowserWindow that will display the content. To achieve this it has control over a list of modules. The following diagram shows the Main process specific modules.



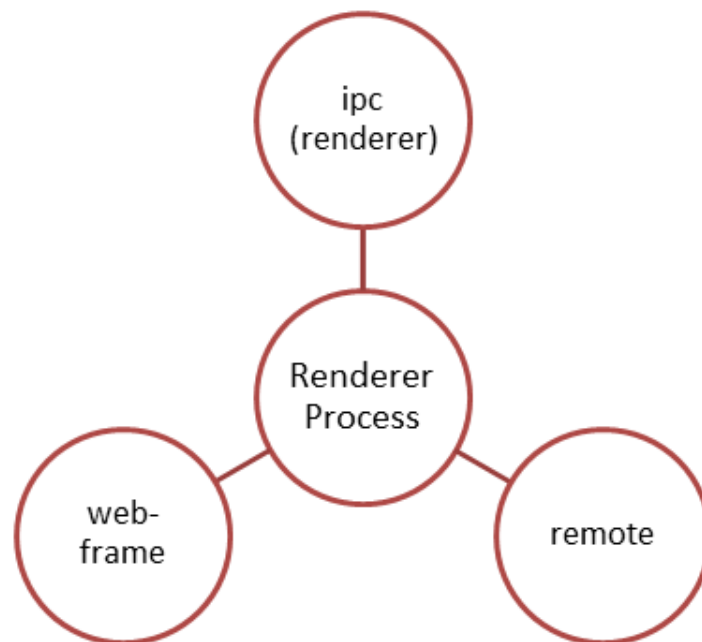
The previous diagram shows the module names specific to Main process. The details of these modules are listed as follows.

- **App:** This module is responsible for controlling the application's life time.
- **auto-updater:** This module takes care of updating the application to a new version.
- **browser-window:** This module is used for creating browser window as well as frameless window.
- **content-tracing:** This module collects tracing data generated by the underlying Chromium content module
- **dialog:** This modules provides APIs to show native system dialogs
- **global-shortcut:** This module can register/unregister a global keyboard shortcut in operating system

- **Main ipc:** This module handles asynchronous and synchronous message sent from a renderer process
- **Menu:** This module creates native menus that can be used as application menus and context menus
- **menu-item:** This module is used to add new item to menu list.
- **power-monitor:** This module monitors the power state change.
- **protocol:** This module register a new protocol or intercept an existing protocol.
- **tray:** This module represents an icon in operating system's notification area.

Renderer Process Modules

Renderer process control the Electron application for displaying content in the browser window by using Chromium multiter architecture. To achieve this it has control over a list of modules. The following diagram shows the Renderer process specific modules.

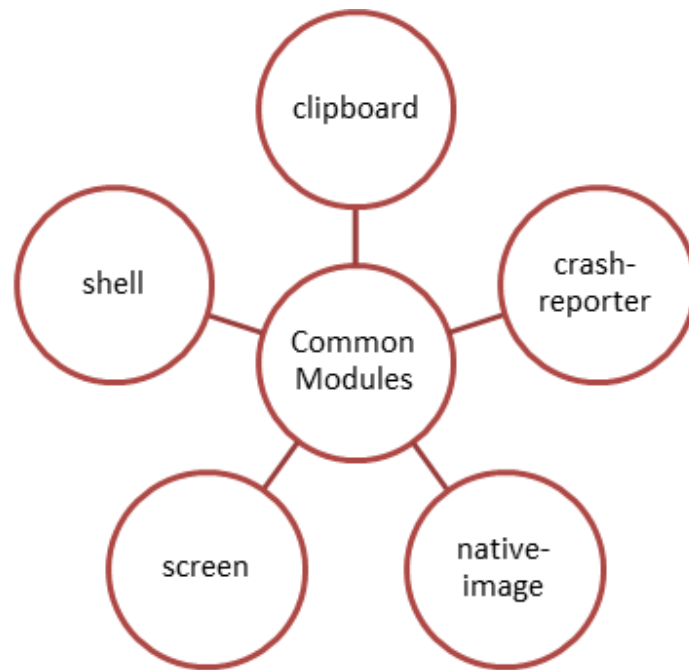


The previous diagram shows the module names specific to Renderer process. The details of these modules are listed as follows.

- **Renderer ipc:** This module can send synchronous and asynchronous messages to the main process, and also receive messages sent from main process
- **Remote:** This module provides inter-process communication between the renderer process and the main process.
- **web-frame:** This module can customize the rendering of current web page.

Common Modules

There few other modules which are common between main and renderer process. The following diagram shows the common modules.



The previous diagram shows the common module names. The details of these modules are listed as follows.

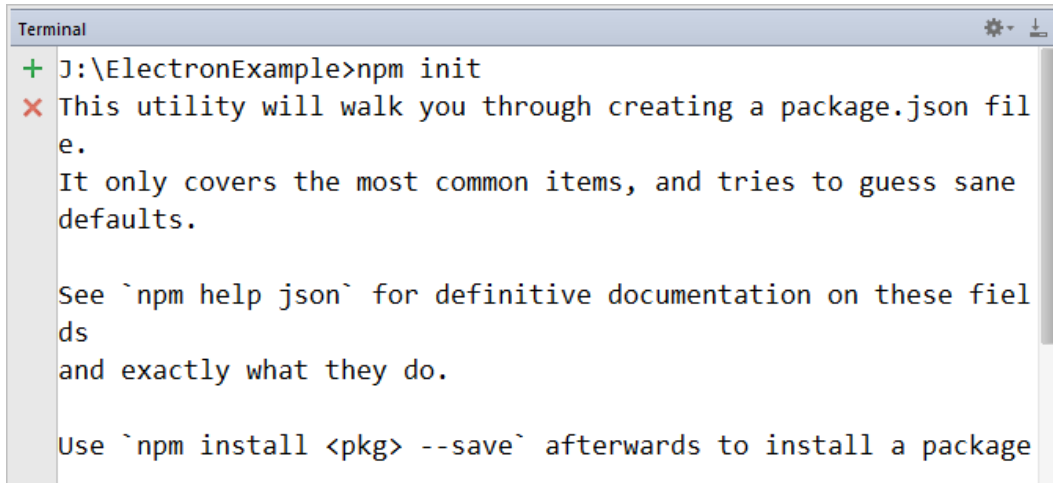
- **Clipboard:** This module provides methods to do copy/paste operations
- **crash-reporter:** This module provides crash reporting for an application to a remote server.
- **native-image:** This module provides support to load native images like PNG and JPEG.
- **screen:** This module provides various information like size, displays, cursor position about the screen.
- **shell:** This module provides functions related to desktop integration like opening a URL in a browser.

Electron Installation

To install Electron we need to have NPM (*Node Package Manager*) installed in the machine. NPM comes bundled with Node javascript library. If node is not present in your machine then download and install it in from Node web page.

The Node can be downloaded from the following link: <https://nodejs.org/download/>

Let's create **package.json** to manage all project related dependencies. To create a **package.json** file we need to issue **npm init** command. For the demonstration of this chapter we have created a directory name **ElectronExample**. The following screenshot shows the terminal with NPM initialization.



```
Terminal
+ J:\ElectronExample>npm init
X This utility will walk you through creating a package.json file.
  It only covers the most common items, and tries to guess sane defaults.

  See `npm help json` for definitive documentation on these fields
  and exactly what they do.

  Use `npm install <pkg> --save` afterwards to install a package
```

Once NPM initialization is completed a **package.json** file is created under the **ElectronExample** directory. The content of **package.json** file is as follows.

```
{
  "name": "ElectronExample",
  "version": "0.0.1",
  "description": "Electron Example",
  "repository": {
    "type": "git",
    "url": "https://github.com/saan1984/ElectronExample.git"
  },
  "author": "Sandeep Kumar Patel",
  "license": "ISC"
}
```

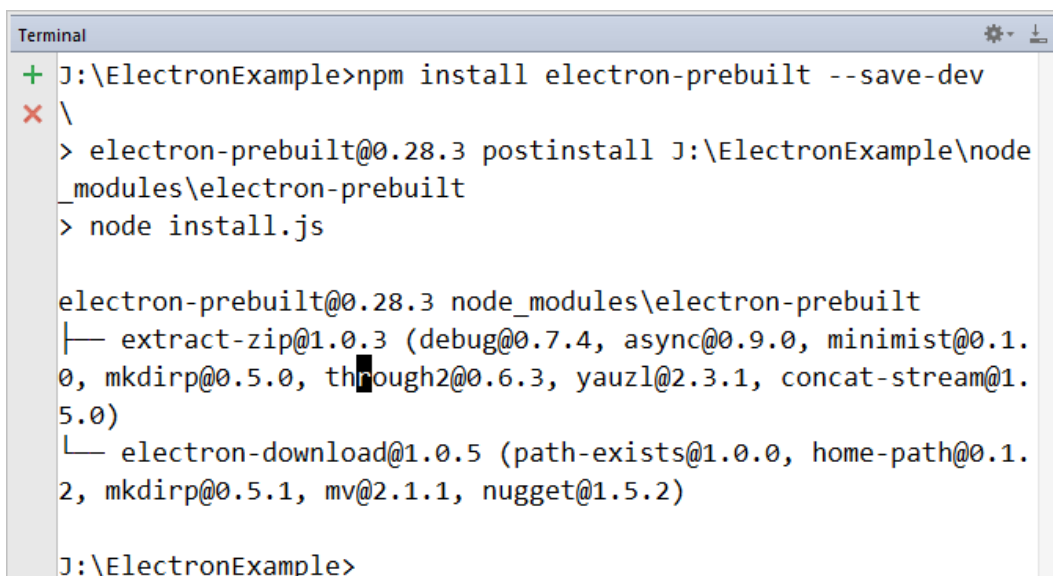
Now we can install the Electron using NPM. Electron can be installed globally by using following command.

```
npm install electron-prebuilt -g
```

Electron can be installed locally as a development dependencies using following NPM command in the terminal.

```
npm install electron-prebuilt --save-dev
```

The following screenshot shows the terminal with Electron prebuilt installed locally to the **ElectronExample** directory.

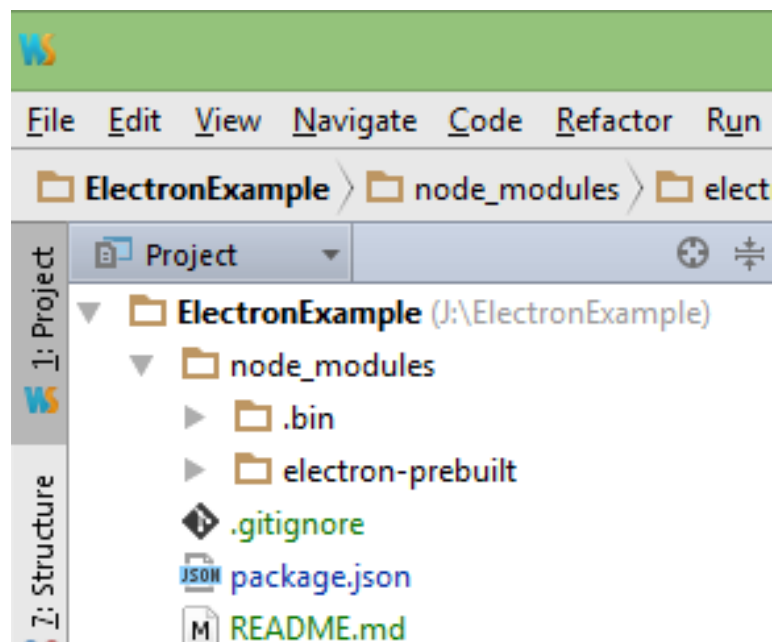


```
Terminal
+ J:\ElectronExample>npm install electron-prebuilt --save-dev
X \
> electron-prebuilt@0.28.3 postinstall J:\ElectronExample\node_modules\electron-prebuilt
> node install.js

electron-prebuilt@0.28.3 node_modules\electron-prebuilt
├─ extract-zip@1.0.3 (debug@0.7.4, async@0.9.0, minimist@0.1.0, mkdirp@0.5.0, through2@0.6.3, yauzl@2.3.1, concat-stream@1.5.0)
└─ electron-download@1.0.5 (path-exists@1.0.0, home-path@0.1.2, mkdirp@0.5.1, mv@2.1.1, nugget@1.5.2)

J:\ElectronExample>
```

On successful installation of electron prebuilt we can find a **node_modules** directory created inside the **ElectronExample** directory. The following screenshot shows the updated project structure after the installation of electron prebuilt.



After installation of **electron-prebuilt** locally the **package.json** is modified with an entry of development dependency. The updated **package.json** is as follows.

```
{
  "name": "ElectronExample",
  "version": "0.0.1",
  "description": "Electron Example",
  "repository": {
    "type": "git",
    "url": "https://github.com/saan1984/ElectronExample.git"
  },
  "author": "Sandeep Kumar Patel",
  "license": "ISC",
  "devDependencies": {
    "electron-prebuilt": "^0.28.3"
  }
}
```

Developing First Electron Application

In this section we will develop our first desktop application using Electron. This is going to be a simple hello world application displaying in a window. We will build this application step by step to understand the basic desktop application development.

Step 1 defining the entry point main.js in package.json file

An Electron based desktop application starts by reading the **package.json** file. The minimum structure of a **package.json** file is as follows:

```
{
  "name": "ElectronExample",
  "main": "main.js",
  "version": "0.0.1",
}
```

The **package.json** file has 3 different properties with values associated correspondingly. The details of these fields are as follows:

- **Name:** This field contains the name of the application.
- **Main:** This field contains the name of the main script which will be executed by Electron when application starts.
- **Version:** This field contains the version number of the application.

Step 2 developing a main.html file

The **main.html** file can contain the HTML, CSS and JavaScript code that will be displayed in the browser. The code content of **main.html** file is listed as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My 1st Electron Application</title>
</head>
<body>
  <h1>Hello world</h1>
</body>
</html>
```

The previous code contains a header message inside the **H1** element which will be displayed on the application window.

Step 3 developing main.js entry logic

The **main.js** file contains the script that is executed at first by the electron. The code content of **main.js** is as follows.

```
(function() {
  var application = require('app'),
      BrowserWindow = require('browser-window');
  application.on('ready', function() {
    var mainWindow = new BrowserWindow({width: 600, height: 300});
    mainWindow.loadUrl('file://' + __dirname + '/main.html');
    mainWindow.on('closed', function() {
      mainWindow = null;
    });
  });
})();
```

The previous code is the entrance script for our Electron based desktop application. The details of these codes are as follows.

- The application module is loaded by a **require()** function to **application** variable. The application module control the life time of the Electron application.
- A Browser window class is also referred through a **require()** function and saved in **BrowserWindow** variable.
- A callback function is attached to application ready event. This callback method is called when application is loaded.
- Inside the application ready callback method a **BrowserWindow** instance is created with **height** 300px and **width** as 600px and saved in the **mainWindow** variable.
- The browser windows loads the **main.html** file using **loadUrl()** method. Once the **main.html** is loaded the content will be displayed in the **mainWindow**.
- Finally a callback method is attached to **close** event on **mainWindow** which gets called when user closes the browser window. The callback method code makes the **mainWindow** variable to null to purge the **BrowserWindow** instance associated with it.

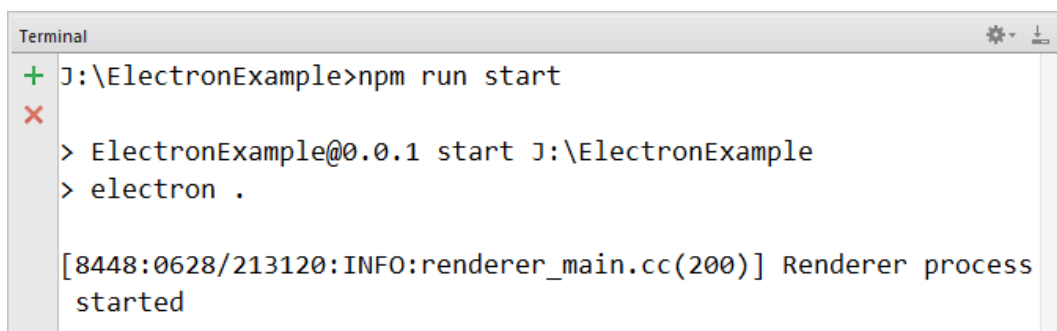
Step 4 running the desktop application

We are all set up with our first Electron desktop application. We can run this app by calling the command Electron on the project directory. For this example we have installed the Electron as local Node module we can create NPM script command to run the Electron.

The NPM script command will be **electron <dot>** where **dot (.)** represents the current directory. We will include this to our **package.json** file to reuse it to run the application. The modified **package.json** file is listed as follows.

```
{
  "name": "ElectronExample",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "version": "0.0.1"
}
```

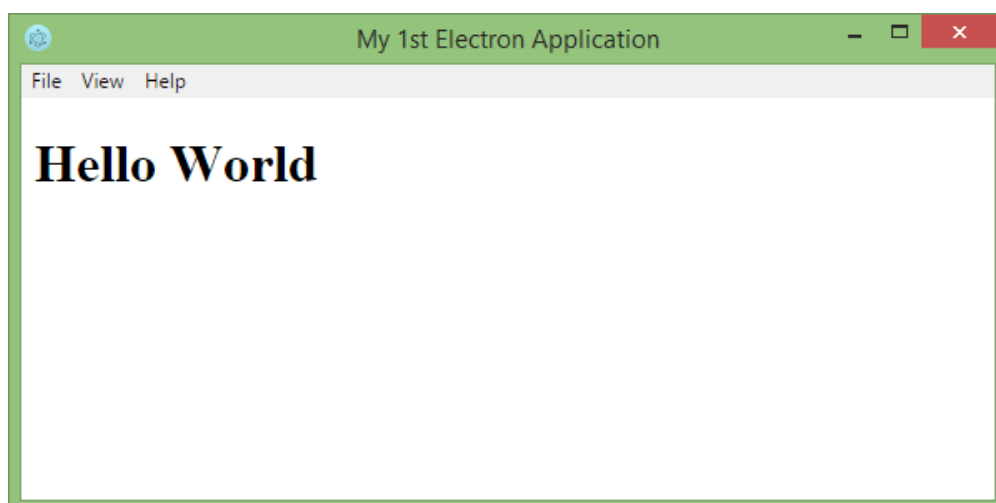
Now we can run our desktop application by calling `npm run start` command. The following screenshot shows the terminal with NPM command to run the desktop application.

A screenshot of a Windows Terminal window. The title bar says "Terminal". The command prompt shows the user running "J:\ElectronExample>npm run start". The output shows the command being executed: "> ElectronExample@0.0.1 start J:\ElectronExample" and "> electron .". Below that, a log message is displayed: "[8448:0628/213120:INFO:renderer_main.cc(200)] Renderer process started".

```
Terminal
+ J:\ElectronExample>npm run start
x
> ElectronExample@0.0.1 start J:\ElectronExample
> electron .

[8448:0628/213120:INFO:renderer_main.cc(200)] Renderer process
started
```

On successful execution of previous command we will see a window pops up in the screen with a message printed on it. The following screenshot shows the window that is generated by the Electron application.



The previous screenshot looks like a native window. The operating system of my system is windows 8 that is the reason it looks like a native window. If we run this code in Mac system the output window will look like a naïve Mac window.

Summary

From this chapter we have introduced with the ElectronJS framework for developing desktop application. We have also explored building blocks of Electron framework and 2 processes those are responsible for running an application on desktop.

About The Author



Sandeep Kumar Patel is a senior web developer and founder of www.tutorialsavvy.com, a widely-read programming blog since 2012. He has more than five years of experience in object-oriented JavaScript and JSON-based web applications development. He is GATE-2005 Information Technology (IT) qualified and has a Master's degree from VIT University, Vellore.

You can know more about him from his

-LinkedIn profile (<http://www.linkedin.com/in/techblogger>).

-He has received the Dzone Most Valuable Blogger (MVB) award for technical publications related to web technologies. His article can be viewed at <http://www.dzone.com/users/sandeepgiet>.

-He has also received the Java Code Geek (JCG) badge for a technical article published in JCG. His article can be viewed at <http://www.javacodegeeks.com/author/sandeep-kumar-patel/>.

-Author of "Instant GSON" for Packt publication, <http://www.packtpub.com/create-json-data-java-objects-implement-with-gson-library/book>

Questions or comments? E-mail me at sandeep Pateltech@gmail.com or find me on the following social networks:-

-Facebook Page: <http://www.facebook.com/SandeepTechTutorials> .

-Tutorial Blog: <http://www.tutorialsavvy.com>

One Last Thing...

When you turn the page, Kindle will give you the opportunity to rate this book and share your thoughts on Facebook and Twitter. If you believe the book is worth sharing, please would you take a few seconds to let your friends know about it? If it turns out to make a difference in their professional lives, they'll be forever grateful to you, as will I.

All the best,

Sandeep Kumar Patel.