FREE eBook

LEARNING electron

Free unaffiliated eBook created from
Stack Overflow contributors.

#electron

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: electron

It is an unofficial and free electron ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official electron.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with electron

Remarks

What is Electron?

Electron is an **open-source framework**, which is used to create desktop applications using **HTML** , **CSS** and **JavaScript**. In the inside, it works thanks to **Chromium** and **Node.js**.

Its original creator, GitHub, works with a wide community of developers to maintain the project, which can be found here.

One of the main perks of using Electron is that, since it's based in web technologies, it's **cross platform**, allowing to deploy applications for Linux, MacOS and Windows, with the same code.

It also features native elements such as menus and notifications, as well as useful developing tools for debugging and crash reporting.

Apps built on Electron

Some examples of applications that use this framework, are:

- Atom
- Slack for Desktop
- Visual Studio Code
- GitBook
- Curse
- Wordpress for Desktop

... and many others.

Versions

Version	Remarks	Release Date
1.0.0		2016-05-09
1.0.1		2016-05-11
1.0.2		2016-05-13
1.1.0		2016-05-13
1.1.1		2016-05-20

Version	Remarks	Release Date
1.1.2		2016-05-24
1.1.3		2016-05-25
1.2.0		2016-05-26
1.2.1		2016-06-01
1.2.2		2016-06-08
1.2.3	There are more between this and 1.4.7, but there were too many to list out	2016-06-16
1.4.7	Lastest version as of 19th Nov 2016	2016-11-19
1.6.11		2017-05-25
1.7.3	Lastest Version as of 19th Jun 2017	2017-06-19

Examples

Installation of Electron

Dependencies

To install electron you must first install Node.js, which comes with npm.

How to install it?

Use npm:

```
# Install the `electron` command globally in your $PATH
npm install electron -g

# OR

# Install as a development dependency
npm install electron --save-dev
```

Hello World!

Setup

An Electron project structure usually looks like this:

```
hello-world-app/
├── package.json
├── index.js
└── index.html
```

Now let's create the files and initialize our `package.json`.

```
$ mkdir hello-world-app && cd hello-world-app
$ touch index.js
$ touch index.html
$ npm init
```

Note: If the `main` parameter is not specified in `package.json`, Electron will use `index.js` as the default entry point.

The Main Process

In Electron, the process that runs `package.json`'s main script is called the **main process**. Here we can display a GUI by creating `BrowserWindow` instances.

Add the following to `index.js`:

```
const { app, BrowserWindow } = require('electron')

// Global reference to the window object
let win

// This method will be called when Electron has finished
// initialization and is ready to create browser windows
app.on('ready', function(){
    // Create the window
    win = new BrowserWindow({width: 800, height: 600})

    // Open and load index.html to the window
    win.loadURL('file://' + __dirname + '/index.html')

    // Emitted when the window is closed.
    win.on('closed', () => {
        // Dereference the window object
        win = null
    });
})

// Quit the app if all windows are closed
app.on('window-all-closed', () => {
    app.quit()
})
```

HTML Template & Renderer Process

Next we create the GUI for the app. Electron uses web pages as its GUI, each running in their own process called the **renderer process**.

Add the following code to `index.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

Running the App

There are multiple ways to run an Electron App.

With `electron-prebuilt` installed Globally

First, make sure you have `electron-prebuilt` installed.

Now we can test the app using this command:

```
$ electron .
```

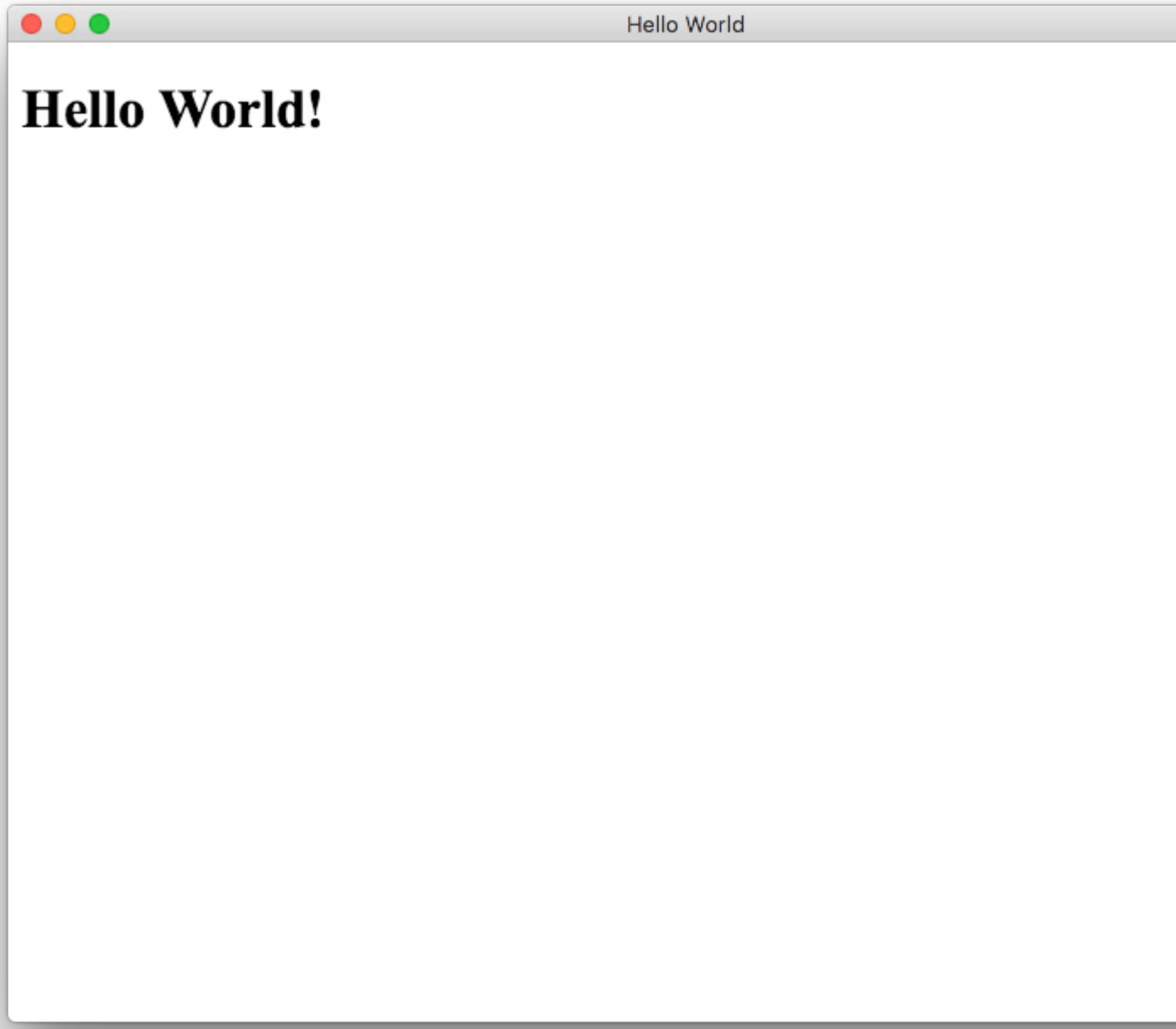
Method 2 - Without `electron-prebuilt` installed Globally

First, we'll have to enter your app's folder (the folder where package.json is).

There, open up a Terminal/Command Prompt window and type `npm install` to install the necessary into that app's folder.

Afterwards, key in `npm start` to run the app. Keep in mind that your `package.json` still has to specify a 'start' script.

If everything worked correctly, you should see something like this:



Congratulations! You've successfully created your first Electron app.

Read Getting started with electron online: https://riptutorial.com/electron/topic/4934/getting-started-with-electron

Chapter 2: Electron-tray-app

Examples

Electron Tray App

Adding a icon to your tray-bar

```
let tray = null;
let mainWindow = null;
let user = null;

app.on('ready', () => {
    /**
     * Tray related code.
     */
    const iconName = 'icon.png';
    const iconPath = path.join(__dirname, iconName);
    tray = new Tray(iconPath);
    tray.setToolTip('AMP Notifier App');
    const contextMenu = Menu.buildFromTemplate([{
        label: 'Quit',
        click: destroyApp
    }]);
    tray.setContextMenu(contextMenu);

    tray.on('click', () => {
        app.quit();
    });
});
```

Read Electron-tray-app online: https://riptutorial.com/electron/topic/8160/electron-tray-app

Chapter 3: electron-winstaller

Introduction

NPM module that builds Windows installers for Electron apps. It will help to create single EXE for Electron windows application

Syntax

- **Install Globally**
- npm install -g electron-winstaller
- **Install Locally**
- npm install --save-dev electron-winstaller

Parameters

Config Name	Description
appDirectory	The authors value for the nuget package metadata. Defaults to the author field from your app's package.json file when unspecified.
owners	The owners value for the nuget package metadata. Defaults to the authors field when unspecified.
exe	The name of your app's main .exe file. This uses the name field in your app's package.json file with an added .exe extension when unspecified.
description	The description value for the nuget package metadata. Defaults to the description field from your app's package.json file when unspecified.
version	The version value for the nuget package metadata. Defaults to the version field from your app's package.json file when unspecified.
title	The title value for the nuget package metadata. Defaults to the productName field and then the name field from your app's package.json file when unspecified.
name	Windows Application Model ID (appId). Defaults to the name field in your app's package.json file.
certificateFile	The path to an Authenticode Code Signing Certificate
certificatePassword	The password to decrypt the certificate given in certificateFile
signWithParams	Params to pass to signtool. Overrides certificateFile and certificatePassword.

Config Name	Description
iconUrl	A URL to an ICO file to use as the application icon (displayed in Control Panel > Programs and Features). Defaults to the Atom icon.
setupIcon	The ICO file to use as the icon for the generated Setup.exe
setupExe	The name to use for the generated Setup.exe file
setupMsi	The name to use for the generated Setup.msi file
noMsi	Should Squirrel.Windows create an MSI installer?
remoteReleases	A URL to your existing updates. If given, these will be downloaded to create delta updates
remoteToken	Authentication token for remote updates

Examples

Build JS

Here Is basic build file to build executable from electron windows app.

```
var electronInstaller = require('electron-winstaller');
var resultPromise = electronInstaller.createWindowsInstaller({
    appDirectory: 'Your_electron_application_path',
    authors: 'Author Name',
    description: "Description"
});

resultPromise.then(() => console.log("Build Success!"), (e) => console.log(`No dice:
${e.message}`));
```

Read electron-winstaller online: https://riptutorial.com/electron/topic/9492/electron-winstaller

Chapter 4: Main and renderer process.

Remarks

Process that runs `package.json`'s main script is called the **main process**. The main process creates web pages by creating `BrowserWindow` instances. Each web page in Electron runs in its own process, which is called the **renderer process**. The main process manages all web pages and their corresponding renderer processes. Each renderer process is isolated and only cares about the web page running in it.

Examples

Asynchronous IPC communication

Main process source code `index.js`:

```
const {app, BrowserWindow, ipcMain} = require('electron')
let win = null

app.on('ready', () => {
  win = new BrowserWindow()
  win.loadURL(`file://${__dirname}/index.html`)
  win.webContents.openDevTools()
  win.on('closed', () => {
    win = null
  })
  win.webContents.on('did-finish-load', () => {
    win.webContents.send('asyncChannelToRenderer', 'hello')
  })
})

ipcMain.on('asyncChannelToMain', (event, arg) => {
  console.log(arg + ' from renderer')
  if (arg === 'hello') {
    event.sender.send('asyncChannelToRenderer', 'world')
  }
})
```

Renderer process in `index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World IPC</title>
    <script>
      require('electron').ipcRenderer.on('asyncChannelToRenderer', (event, arg) => {
        console.log(arg + ' from main')
        if (arg === 'hello') {
          event.sender.send('asyncChannelToMain', 'world')
        }
      })
    </script>
```

```
  </head>
  <body>
    <button onclick="require('electron').ipcRenderer.send('asyncChannelToMain',
'hello')">click me</button>
  </body>
</html>
```

Remote module RMI

The `remote` module allows simple RMI (remote method invocation) of main process objects from renderer process. First create the main process in `index.js`

```
const {app, BrowserWindow} = require('electron')
let win = null

app.on('ready', () => {
  win = new BrowserWindow()
  win.loadURL(`file://${__dirname}/index.html`)
  win.on('closed', () => {
    win = null
  })
})
```

and then remote process `index.html`

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      const {BrowserWindow, app} = require('electron').remote
    </script>
  </head>
  <body>
    <button onclick= "let win = new BrowserWindow();
win.loadURL(`file://${__dirname}/index.html`)">new window</button>
    <button onclick= "app.quit()">quit</button>
  </body>
</html>
```

Synchronous IPC communication

Create `index.js` as

```
const {app, BrowserWindow, ipcMain} = require('electron')
let win = null

app.on('ready', () => {
  win = new BrowserWindow()
  win.loadURL(`file://${__dirname}/index.html`)
  win.webContents.openDevTools()
  win.on('closed', () => {
    win = null
  })
})
```

```
ipcMain.on('syncChannelToMain', (event, arg) => {
  console.log(arg + ' from renderer')
  event.returnValue = 'world'
})
```

and renderer process `index.html` as

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World IPC</title>
  </head>
  <body>
    <button onclick="console.log(require('electron').ipcRenderer.sendSync('syncChannelToMain',
'world') + ' from main')">click me</button>
  </body>
</html>
```

Read Main and renderer process. online: https://riptutorial.com/electron/topic/5432/main-and-renderer-process-

Chapter 5: Packaging an electron app

Introduction

When ready for distribution, your electron app can be packaged into an executable file.

Electron applications can be packaged to run on Windows (32/64 bit), OSX (macOS) and Linux (x86/x86_64).

To package your code, use the npm package 'electron-packager\

https://github.com/electron-userland/electron-packager

Syntax

- $ electron-packager
- sourcedir
- appname
- --platform=platform
- --arch=arch
- [optional flags...]

Parameters

Parameter	Details
sourcedir	The directory of your electron application files
appname	The name of your application
platform	The platform you want to compile your code for. Omitting this will compile for the host OS
arch	The system architecture you want to compile your code for. Omitting this will compile for the host arch

Examples

Installing electron-packager

```
# for use in npm scripts
npm install electron-packager --save-dev

# for use from cli
npm install electron-packager -g
```

Packaging from CLI

```
electron-packager C:/my-app MyApp
```

Packaging from script

```
var packager = require('electron-packager');

packager({
    dir: '/',
}, function(err, path){
    if(err) throw err;
    // Application has been packaged
});
```

Making npm scripts to automate Electron packaging

A convenient way to package your application is to write the scripts in your `packages.json` file and run them with the `npm run` command

```
{
    "name": "AppName",
    "productName": "AppName",
    "version": "0.1.1",
    "main": "main.js",
    "devDependencies": {
        "electron": "^1.6.6",
        "electron-packager": "^8.7.0"
    },
    "scripts": {
        "package-mac": "electron-packager . --overwrite --platform=darwin --arch=x64 --
icon=images/icon.png --prune=true --out=release-builds",
        "package-win": "electron-packager . --overwrite --platform=win32 --arch=ia32 --
icon=images/icon.png --prune=true --out=release-builds",
        "package-linux" : "electron-packager . --overwrite --platform=linux --arch=x64 --
icon=images/icon.png --prune=true --out=release-builds"
    }
}
```

And to run them you just write:

```
npm run package-mac
npm run package-win
npm run package-linux
```

A breakdown of the command flags is:

```
electron-packager .     // this runs the packager in the current folder
--overwrite             // overwrite any previous build
--platform=darwin       // platform for which the binaries should be created
--arch=x64              // the OS architecture
--icon=images/icon.png  // the icon for the app executable
--prune=true            // this does not copy your dev-dependencies that appear in your
```

```
packages.json
--out=release-builds    // the name of the folder were the binaries will be outputed
```

Before, running the scripts change the devDependencies to dependencies as electron-packager cannot bundle the packages in the devDependencies into the app. In packager.json, change the word (if it's there or if packages are installed using --save-dev in npm install) devDependencies to only dependencies.

Read Packaging an electron app online: https://riptutorial.com/electron/topic/8945/packaging-an-electron-app

Chapter 6: Remote function - use Electron functions in JavaScript

Introduction

If you have to change some things in renderer.js or main.js but you want to do the changes in index.html, you can use the remote function. It lets you access all the electron functions you need!

Syntax

- use remote like `require("electron")`:
    - main.js: `const electron = require("electron");`

      index.html: `const electron = require("electron").remote;`

Examples

Using remote by setting the progress bar

```
const { remote } = require("electron"); // <- The Node.js require() function is
                                        // added to JavaScript by electron

function setProgress(p) { // p = number from 0 to 1
  const currentWindow = remote.getCurrentWindow();
  currentWindow.setProgressBar(p);
}
```

Using remote by setting window to fullscreen

```
const { remote } = require("electron"); // <- The Node.js require() function is
                                        // added to JavaScript by electron

function fullscreen(f) { // p = false or true
  const currentWindow = remote.getCurrentWindow();
  currentWindow.maximize();
}
```

Read Remote function - use Electron functions in JavaScript online:
https://riptutorial.com/electron/topic/8719/remote-function---use-electron-functions-in-javascript

Chapter 7: Using bootstrap in electron

Introduction

One of the best front-end frameworks in the web world in twitter bootstrap. As electron is relies on web browser, we can easily use bootstrap with electron in order to use the power of bootstrap in our electron framework. The latest version of bootstrap as of today is 3.3.7 and bootstrap 4 is still in alpha phase.

Examples

Linking Electron with Bootstrap

In order to use bootstrap, there are 2 cases.

1. The electron app is connected to internet
2. The electron app is not connected to internet

For electron apps that are connected to internet, we can just make use of CDN links for bootstrap and include that in our html files.

The problem comes when we have to take it to offline version where the app is not connected to the net. In that case,

1. Download bootstrap from Bootstrap
2. Unzip the folder into the electron app
3. In the bootstrap directory, there are css and javascript files.
4. For better understanding, move the bootstrap css files into the CSS folder (All the styling files will be in this folder) and bootstrap js files to JS folder (All the Javascript files will be in this folder)
5. In your html files , link the html files using the following code

```
<link rel="stylesheet" href="path_to_the_offline_bootstrap_css_file">
<script scr="path_to_the_offline_bootstrap_js_file"></script>
```

In this way you can start using twitter bootstrap in electron framework.

Read Using bootstrap in electron online: https://riptutorial.com/electron/topic/10897/using-bootstrap-in-electron

Chapter 8: Using nedb in electron

Examples

Installation of nedb

It's very easy to install nedb.

```
npm install nedb --save    # Put latest version in your package.json
```

For bower loving people,

```
bower install nedb
```

Connecting electron app with Nedb

While building electron apps, usually the backend is in separate folder (js files) and front end is in a separate folder (html files). In the backend, in order to use the database, we have to include the nedb package with the require statement as follows.

```
var Datastore = require('nedb'),db = new Datastore({ filename: 'data.db', autoload: true });
```

Keep in mind that the loading of the database file is an asynchronous task.

Insert Data in nedb

Basically, in order to insert records to nedb, the data is stored in the form of json with the key being the column names and the value for those names will be the values for that record.

```
var rec = { name: 'bigbounty',age:16};

db.insert(rec, function (err, newrec) {   // Callback is optional
  // newrec is the newly inserted document, including its _id
  // newrec has no key called notToBeSaved since its value was undefined
});
```

Be careful with all the operations of database, as they are asynchronous.

Note ** : If _id is not there in the json data that you are inserting then automatically ,it will be created for you by nedb.

Search in nedb

In order to search for records in nedb, again we need to just pass the json containing the search criteria as a parameter to the find function of db object.

```
db.find({ name: 'bigbounty' }, function (err, docs) {
  // docs is an array containing documents that have name as bigbounty
  // If no document is found, docs is equal to []
});
```

In order to find only one document, as in we use limit in mysql, it's easy in nedb.

```
db.findOne({ name: 'bigbounty' }, function (err, doc) {
  // doc is only one document that has name as bigbounty
  // If no document is found, docs is equal to []
});
```

Delete in nedb

In order to remove documents in nedb, it's very easy. We just have to use remove function of db object.

db.remove({ name: 'bigbounty' }, function (err, numremoved) { // numremoved is the number of documents that are removed. });

Read Using nedb in electron online: https://riptutorial.com/electron/topic/10906/using-nedb-in-electron

Credits

S. No	Chapters	Contributors
1	Getting started with electron	Alphonsus, Community, Eslam Mahmoud, Florian Hämmerle, Hewbot, J F, Piero Divasto, SimplyCodin, Theo, vintproykt, Vishal
2	Electron-tray-app	Anavar Bharmal, nmnsud
3	electron-winstaller	Krupesh Kotecha
4	Main and renderer process.	mrkovec
5	Packaging an electron app	bigbounty, Dan Johnson, VladNeacsu
6	Remote function - use Electron functions in JavaScript	B. Colin Tim, Florian Hämmerle
7	Using bootstrap in electron	bigbounty
8	Using nedb in electron	bigbounty