To use Electron, you need to install [Node.js](https://nodejs.org/en/download/). We recommend that you use the latest LTS version available.

Electron apps follow the same general structure as other Node.js projects. Start by creating a folder and initializing an npm package.

mkdir my-electron-app && cd my-electron-app  
npm init

The interactive init command will prompt you to set some fields in your config. There are a few rules to follow for the purposes of this tutorial:

* entry point should be main.js.
* author and description can be any value, but are necessary for [app packaging](https://www.electronjs.org/docs/latest/tutorial/quick-start#package-and-distribute-your-application).

Your package.json file should look something like this:

{  
 "name": "my-electron-app",  
 "version": "1.0.0",  
 "description": "Hello World!",  
 "main": "main.js",  
 "author": "Jane Doe",  
 "license": "MIT"  
}

Then, install the electron package into your app's devDependencies.

npm install --save-dev electron

Finally, you want to be able to execute Electron. In the [scripts](https://docs.npmjs.com/cli/v7/using-npm/scripts) field of your package.json config, add a start command like so:

{  
 "scripts": {  
 "start": "electron ."  
 }  
}

This start command will let you open your app in development mode.

npm start

Note: This script tells Electron to run on your project's root folder. At this stage, your app will immediately throw an error telling you that it cannot find an app to run.

**Run the main process**[**​**](https://www.electronjs.org/docs/latest/tutorial/quick-start#run-the-main-process)

The entry point of any Electron application is its main script. This script controls the **main process**, which runs in a full Node.js environment and is responsible for controlling your app's lifecycle, displaying native interfaces, performing privileged operations, and managing renderer processes (more on that later).

During execution, Electron will look for this script in the [main](https://docs.npmjs.com/cli/v7/configuring-npm/package-json#main) field of the app's package.json config, which you should have configured during the [app scaffolding](https://www.electronjs.org/docs/latest/tutorial/quick-start#scaffold-the-project) step.

To initialize the main script, create an empty file named main.js in the root folder of your project.

Add the below code

*// main.js*  
  
*// Modules to control application life and create native browser window*  
const { app, BrowserWindow } = require('electron')  
const path = require('path')  
  
const createWindow = () => {  
 *// Create the browser window.*  
 const mainWindow = new BrowserWindow({  
 width: 800,  
 height: 600,  
 webPreferences: {  
 preload: path.join(\_\_dirname, 'preload.js')  
 }  
 })  
  
 *// and load the index.html of the app.*  
 mainWindow.loadFile('index.html')  
  
 *// Open the DevTools.*  
 *// mainWindow.webContents.openDevTools()*  
}  
  
*// This method will be called when Electron has finished*  
*// initialization and is ready to create browser windows.*  
*// Some APIs can only be used after this event occurs.*  
app.whenReady().then(() => {  
 createWindow()  
  
 app.on('activate', () => {  
 *// On macOS it's common to re-create a window in the app when the*  
 *// dock icon is clicked and there are no other windows open.*  
 if (BrowserWindow.getAllWindows().length === 0) createWindow()  
 })  
})  
  
*// Quit when all windows are closed, except on macOS. There, it's common*  
*// for applications and their menu bar to stay active until the user quits*  
*// explicitly with Cmd + Q.*  
app.on('window-all-closed', () => {  
 if (process.platform !== 'darwin') app.quit()  
})  
  
*// In this file you can include the rest of your app's specific main process*  
*// code. You can also put them in separate files and require them here.*

*// preload.js*  
  
*// All the Node.js APIs are available in the preload process.*  
*// It has the same sandbox as a Chrome extension.*  
window.addEventListener('DOMContentLoaded', () => {  
 const replaceText = (selector, text) => {  
 const element = document.getElementById(selector)  
 if (element) element.innerText = text  
 }  
  
 for (const dependency of ['chrome', 'node', 'electron']) {  
 replaceText(`${dependency}-version`, process.versions[dependency])  
 }  
})

Till here its common for all the Operating systems

"Electron Packager is a command line tool and Node.js library that bundles Electron-based application source code with a renamed Electron executable and supporting files into folders ready for distribution."

So lets go ahead and install it. Run these commands in the terminal in the app folder:

*# for use in npm scripts*

npm install electron-packager --save-dev

*# for use from cli*

npm install electron-packager -g

### MacOS

Now you can run this command from the terminal to build a package for mac:

electron-packager . --overwrite --platform**=**darwin --arch**=**x64 --icon**=**assets/icons/mac/icon.icns --prune**=**true --out**=**release-builds

Windows

And to build for Windows you can run this from the git bash:

electron-packager . electron-tutorial-app --overwrite --asar**=**true --platform**=**win32 --arch**=**ia32 --icon**=**assets/icons/win/icon.ico --prune**=**true --out**=**release-builds --version-string.CompanyName**=**CE --version-string.FileDescription**=**CE --version-string.ProductName**=**"Electron Tutorial App"

Linux

electron-packager . electron-tutorial-app --overwrite --asar**=**true --platform**=**linux --arch**=**x64 --icon**=**assets/icons/png/1024x1024.png --prune**=**true --out**=**release-builds

**overwrite**: replaces any existing output directory when packaging.

**platform**: The target platform to build for

**arch**: the architecture to build for

**icon**: sets the icon for the app

**prune**: runs npm prune –production before packaging the app. It removes unnecesary packages.

**out**: the name of the directory where the packages are created.

## **4. Shortcuts**

To make it easier to create new builds we can create scripts in package.json so that we don’t have to remember all these settings. Add the scripts below, making your package.json look like this:

{

"name": "electron-tutorial-app",

"productName": "Electron tutorial app",

"version": "0.1.0",

"main": "main.js",

"devDependencies": {

"electron": "^1.4.3",

"electron-packager": "^8.1.0"

},

"scripts": {

"package-mac": "electron-packager . --overwrite --platform=darwin --arch=x64 --icon=assets/icons/mac/icon.icns --prune=true --out=release-builds",

"package-win": "electron-packager . electron-tutorial-app --overwrite --asar=true --platform=win32 --arch=ia32 --icon=assets/icons/win/icon.ico --prune=true --out=release-builds --version-string.CompanyName=CE --version-string.FileDescription=CE --version-string.ProductName=\"Electron Tutorial App\"",

"package-linux": "electron-packager . electron-tutorial-app --overwrite --asar=true --platform=linux --arch=x64 --icon=assets/icons/png/1024x1024.png --prune=true --out=release-builds"

}

}

Now you can run:

npm run package-mac

npm run package-win

npm run package-linux