INTRODUCTION TO ELECTRON

Fábio Rodrigues

14/12/2018

WHAT IS ELECTRON?

Electron is a framework for creating native applications with web technologies.

Web Technologies

Chromium + Node.js

Open Source

Maintained by GitHub

Cross Platform

Mac, Windows and Linux

WHAT IS ELECTRON?

Electron is a framework for creating native applications with web technologies.

Automatic updates

Native menus & notifications

Crash reporting

Debugging & profiling

Windows installers

APPS BUILT USING ELECTRON

Atom

Visual Studio Code

Slack

GitHub Desktop

GitKraken

WordPress.com

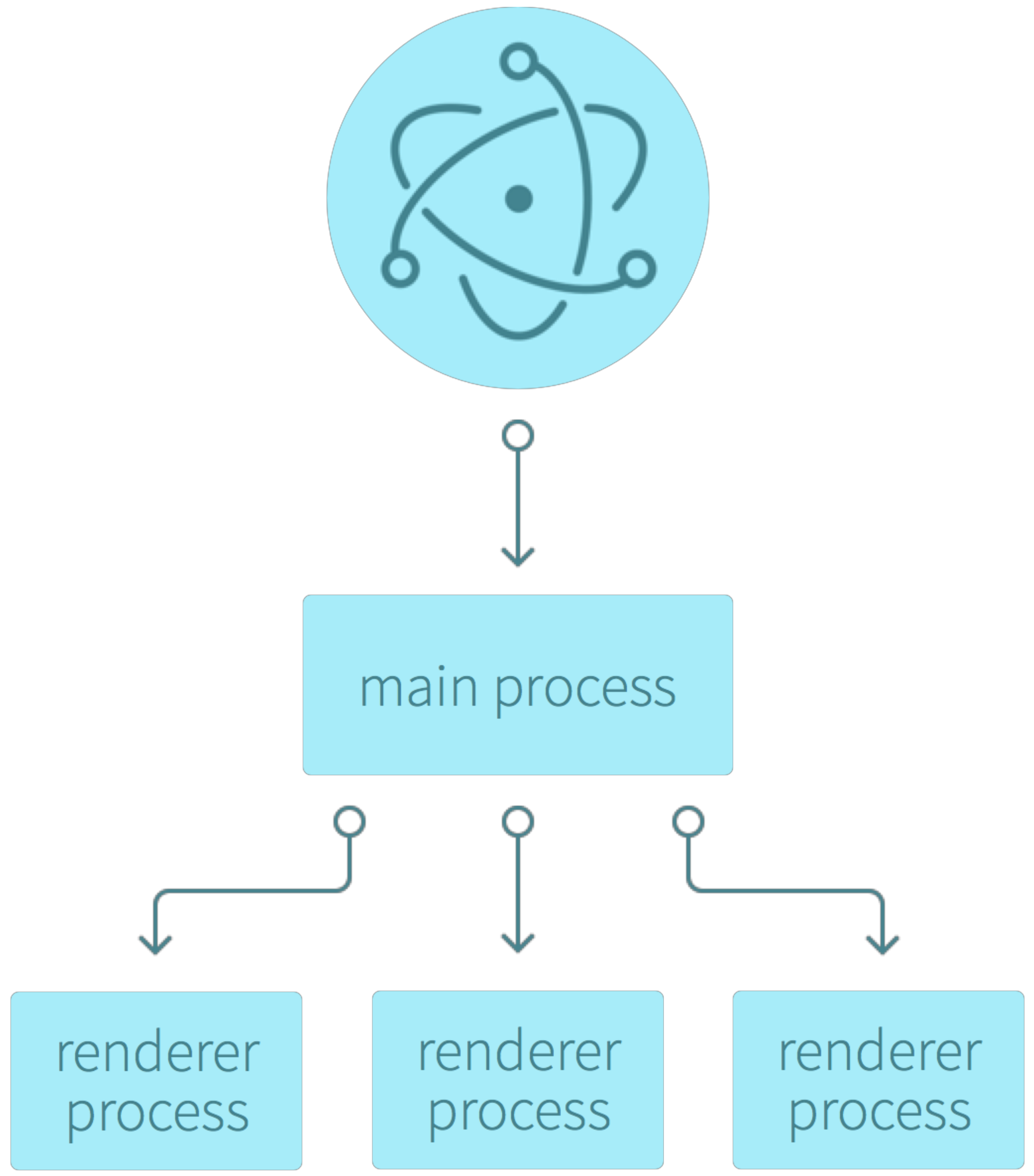
WhatsApp

Skype

ELECTRON TIMELINE

April 2013	Atom Shell is started
May 2014	Atom Shell is open sourced
April 2015	Atom Shell is re-named Electron
May 2016	Electron releases v1.0.0
May 2016	Electron apps compatible with Mac App Store
August 2016	Windows Store support for Electron apps

ELECTRON APPLICATION ARCHITECTURE



MAIN PROCESS

Controls the life of the app, from open to close.

main.js

CAN ACCESS:

- Node.js APIs
- Electron main process modules

USED TO:

- Create renderer processes
- Call native elements
- Start and quit the app

RENDERER PROCESS

A browser window in the app.

Index.html

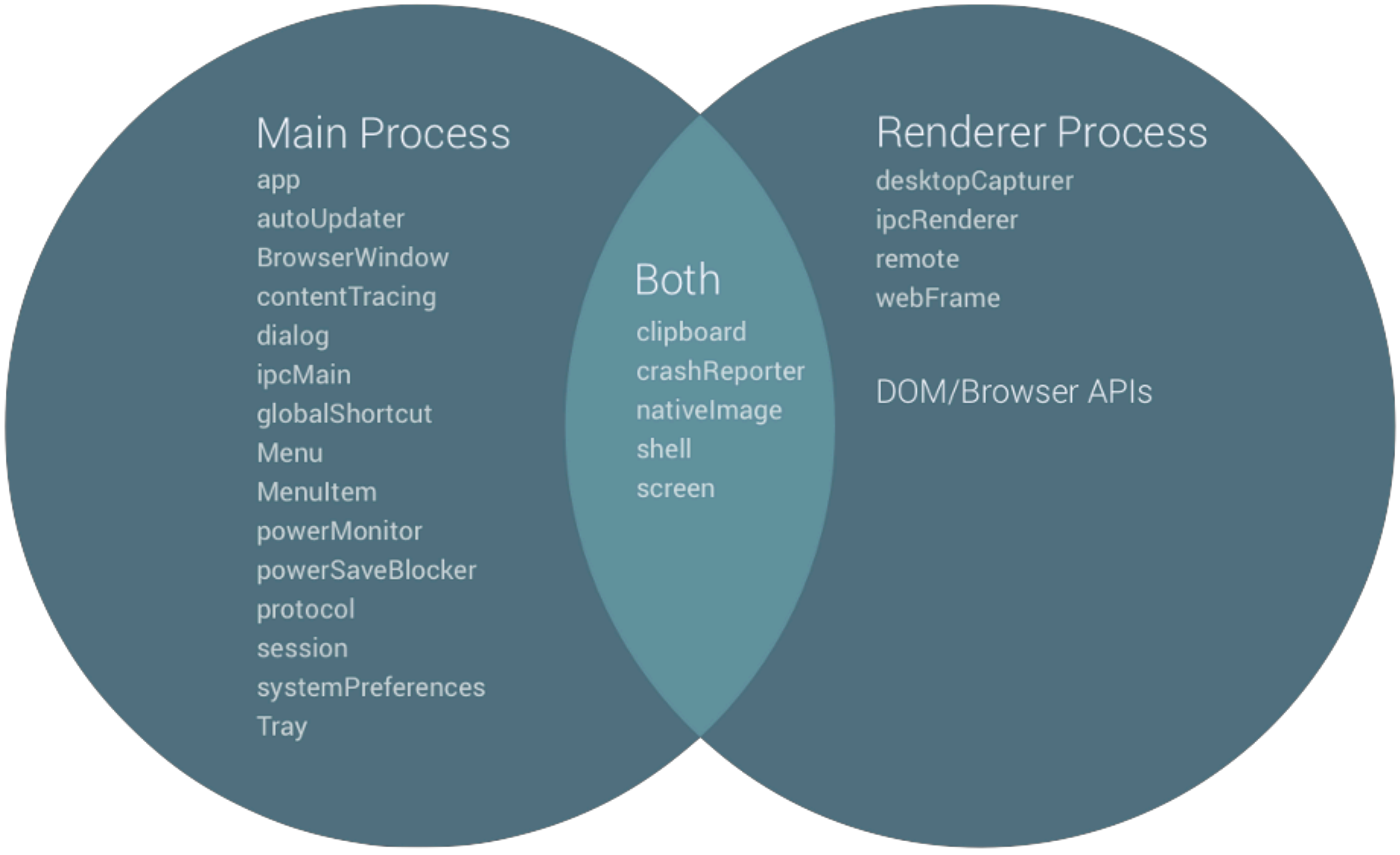
CAN ACCESS:

- Node.js APIs
- DOM APIs
- Electron renderer process modules

USED TO:

- Design the page with HTML and CSS
- Javascript page interactions

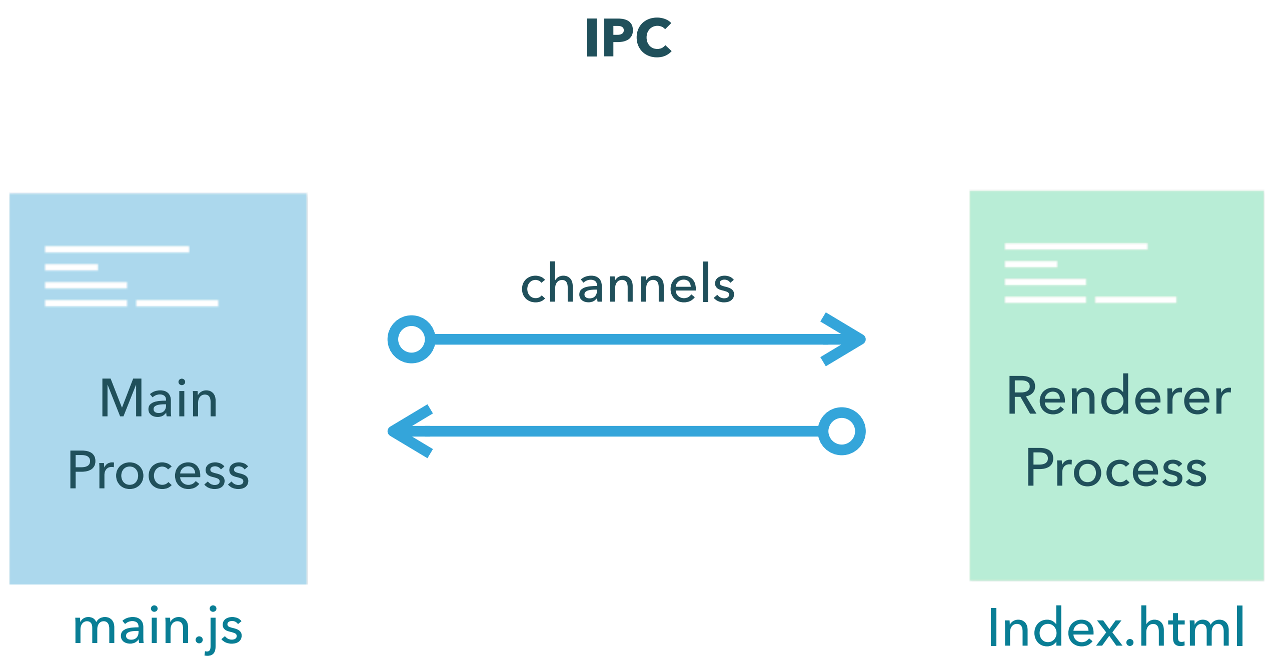
ELECTRON APIs



Node.js

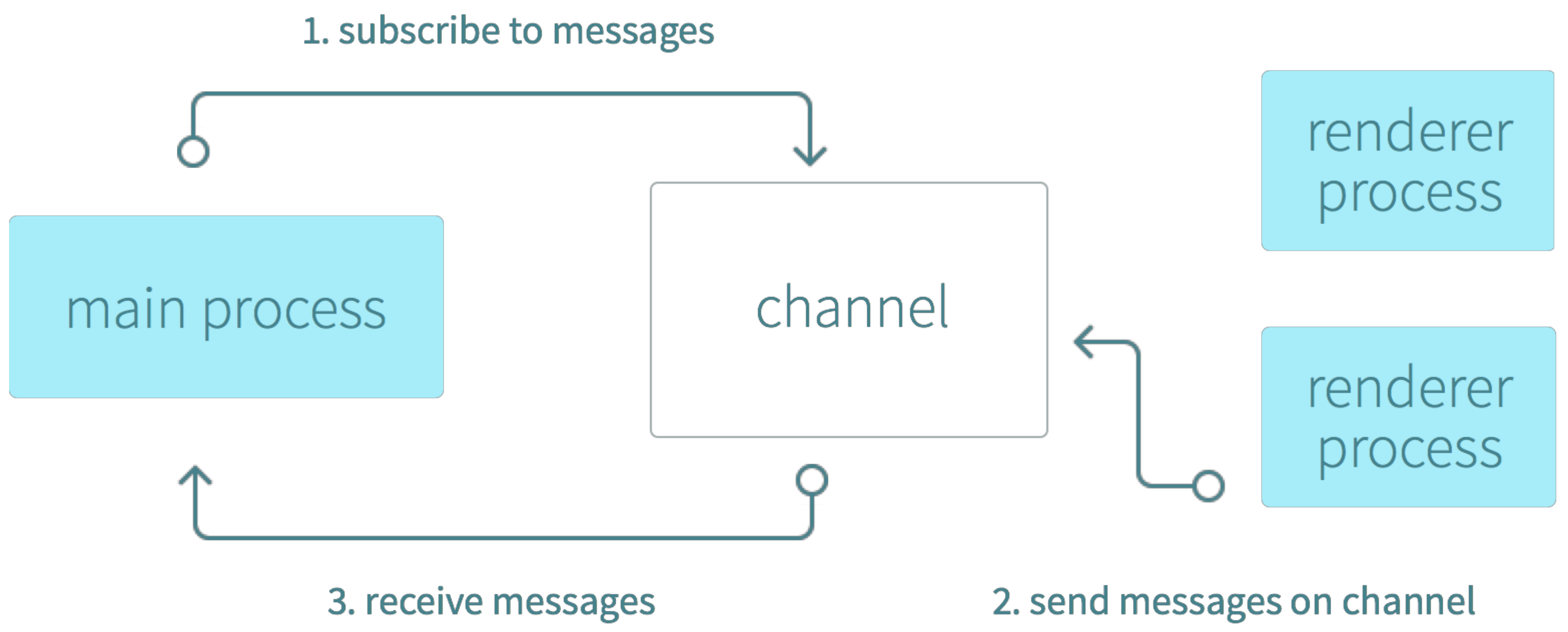
IPC - INTERPROCESS COMMUNICATION

The main process and renderer process need to communicate

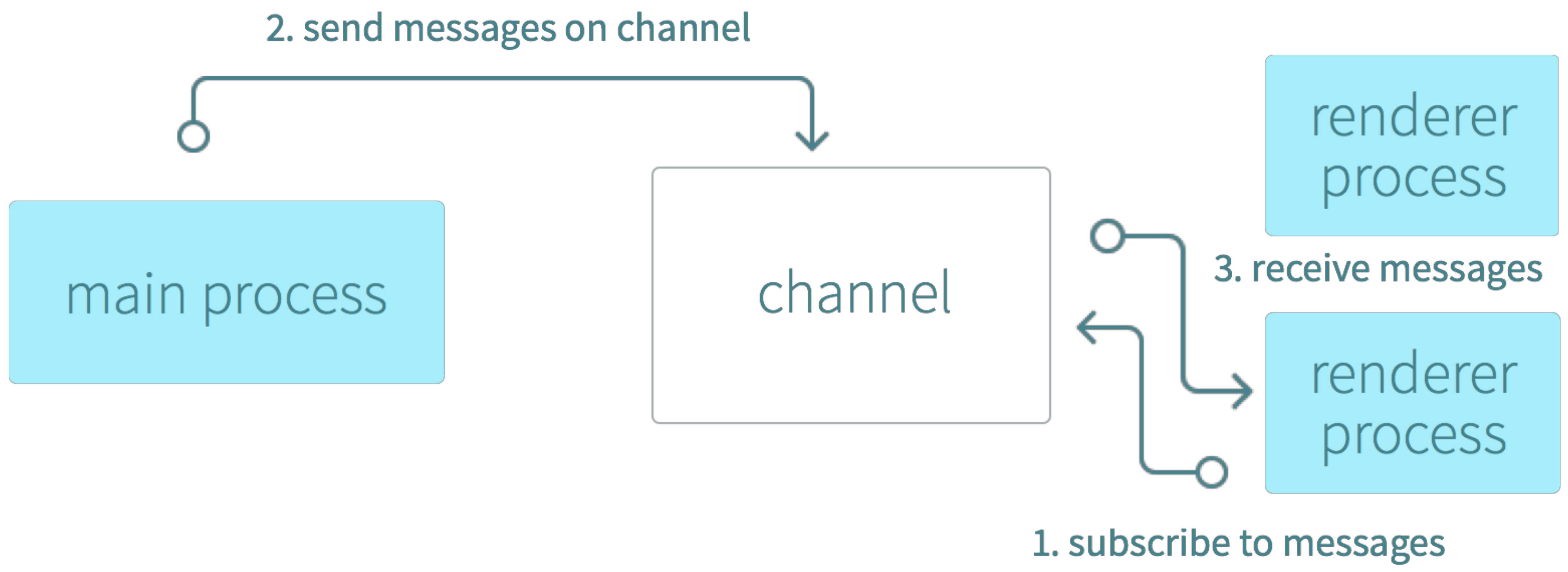


Processes can send messages and listen to messages on "named" channels.

IPC - RENDERER PROCESS TO MAIN PROCESS

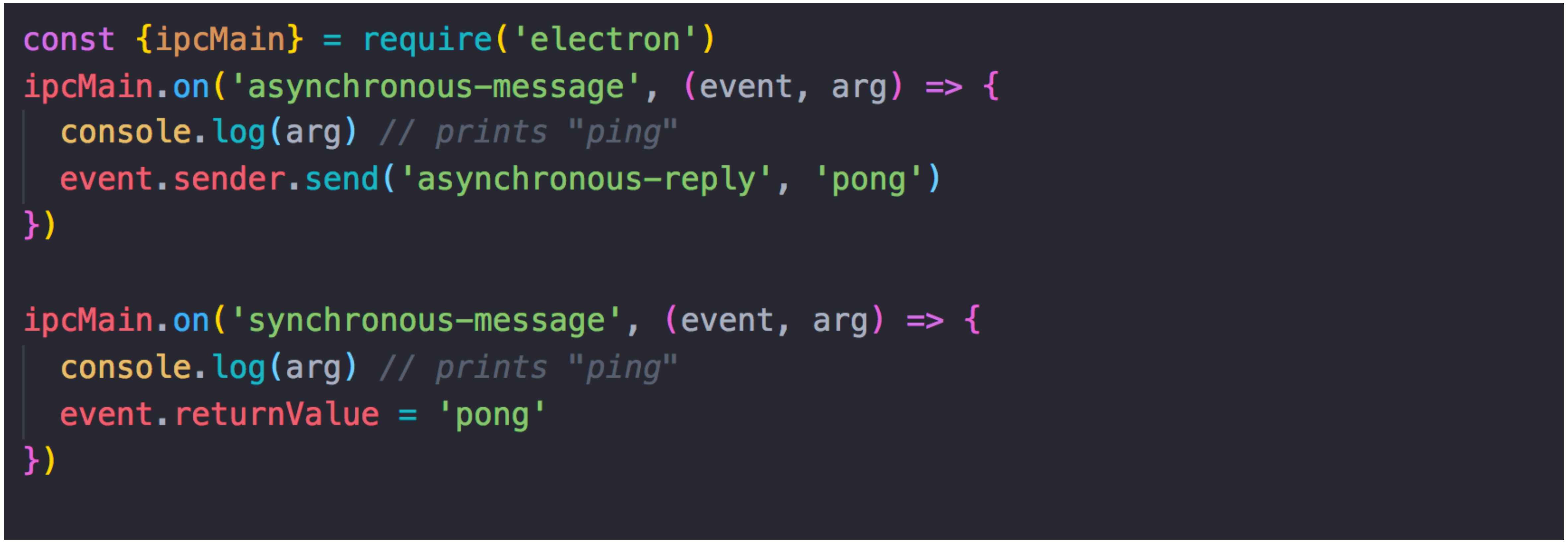


IPC - MAIN PROCESS TO RENDERER PROCESS



IPC MAIN MODULE

Communicate synchronously or asynchronously from the **main process** to the **renderer process**

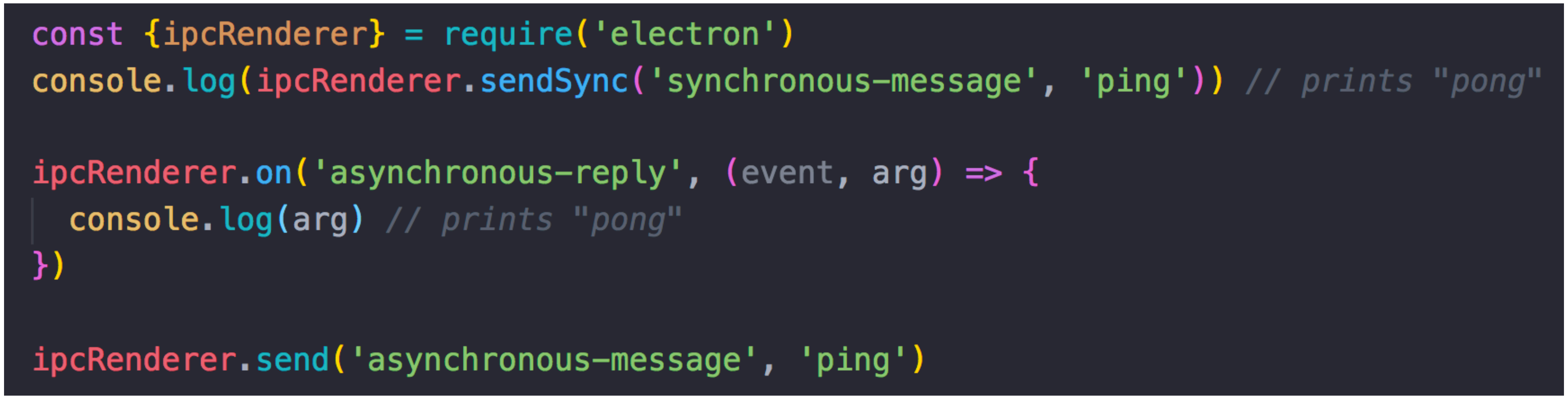


```
const {ipcMain} = require('electron')
ipcMain.on('asynchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.sender.send('asynchronous-reply', 'pong')
})

ipcMain.on('synchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.returnValue = 'pong'
})
```

*The main process is subscribed to the **asynchronous-message** and **synchronous-message** channels and sends messages using the **asynchronous-reply** channel.*

IPC RENDERER MODULE

Communicate synchronously or asynchronously from the **renderer process** to the **main process**

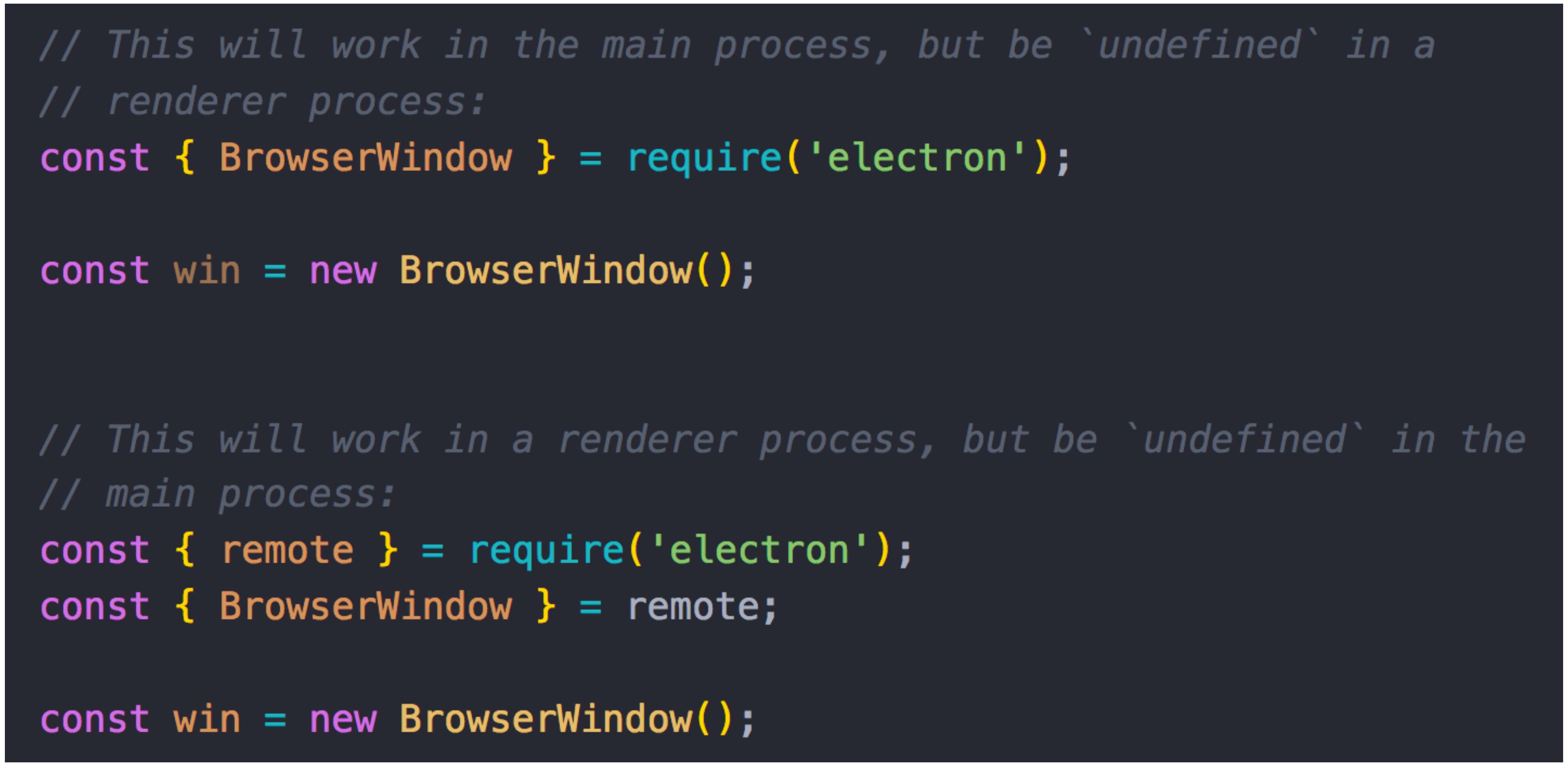


```
const {ipcRenderer} = require('electron')
console.log(ipcRenderer.sendSync('synchronous-message', 'ping')) // prints "pong"

ipcRenderer.on('asynchronous-reply', (event, arg) => {
  console.log(arg) // prints "pong"
})

ipcRenderer.send('asynchronous-message', 'ping')
```

*The renderer process is subscribed to the **asynchronous-reply** channel and sends messages using the **asynchronous-message** and **synchronous-message** channels.*

IPC - REMOTE MODULE

The remote module exposes APIs usually only available in the **main process** without the need to use IPC explicitly.

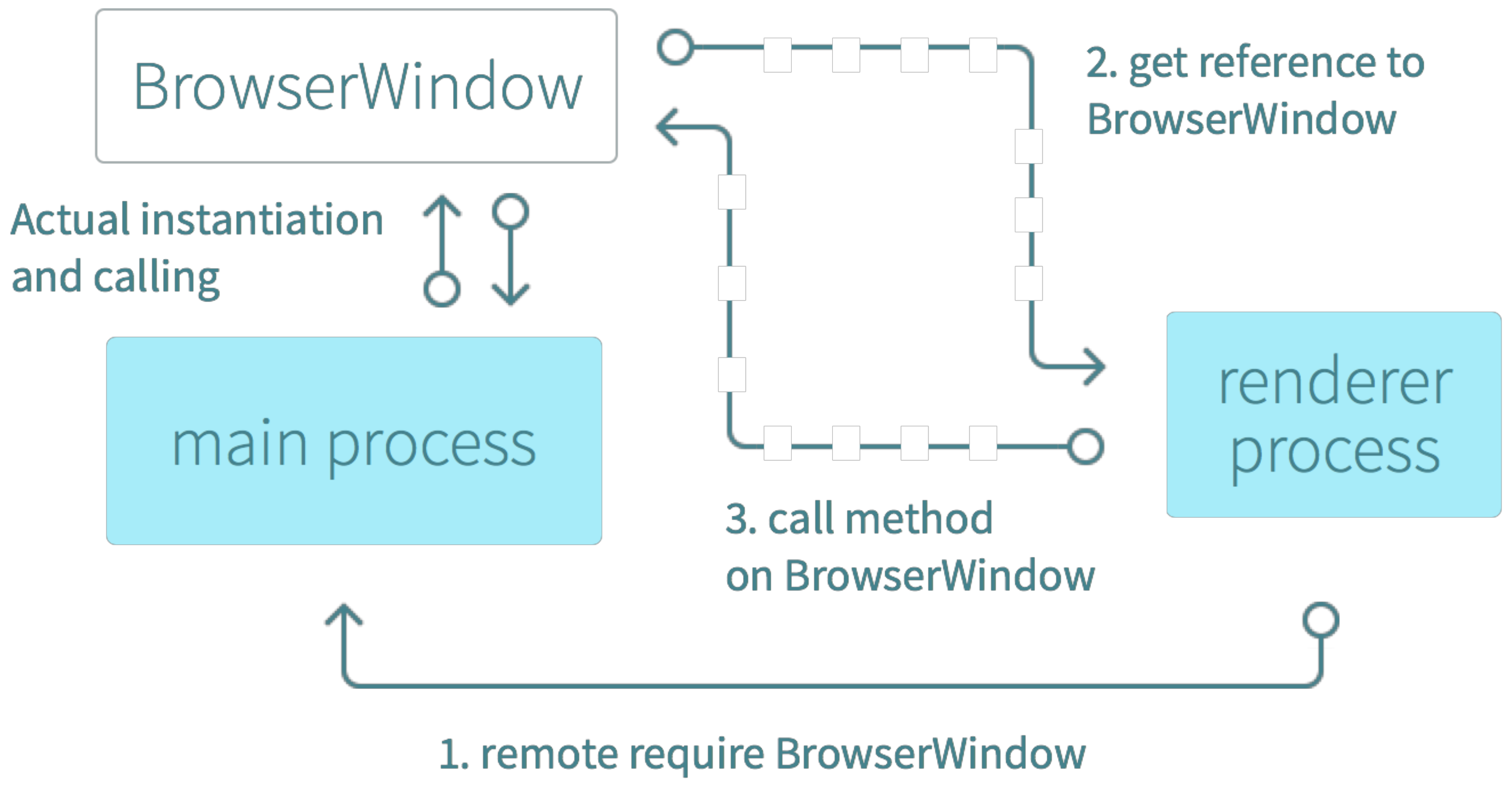


```javascript
// This will work in the main process, but be `undefined` in a
// renderer process:
const { BrowserWindow } = require('electron');

const win = new BrowserWindow();


// This will work in a renderer process, but be `undefined` in the
// main process:
const { remote } = require('electron');
const { BrowserWindow } = remote;

const win = new BrowserWindow();
```

Creating a ***BrowserWindow*** *in the main process and the renderer process*

IPC - REMOTE MODULE

The remote module exposes APIs usually only available in the main process without the need to use IPC explicitly.

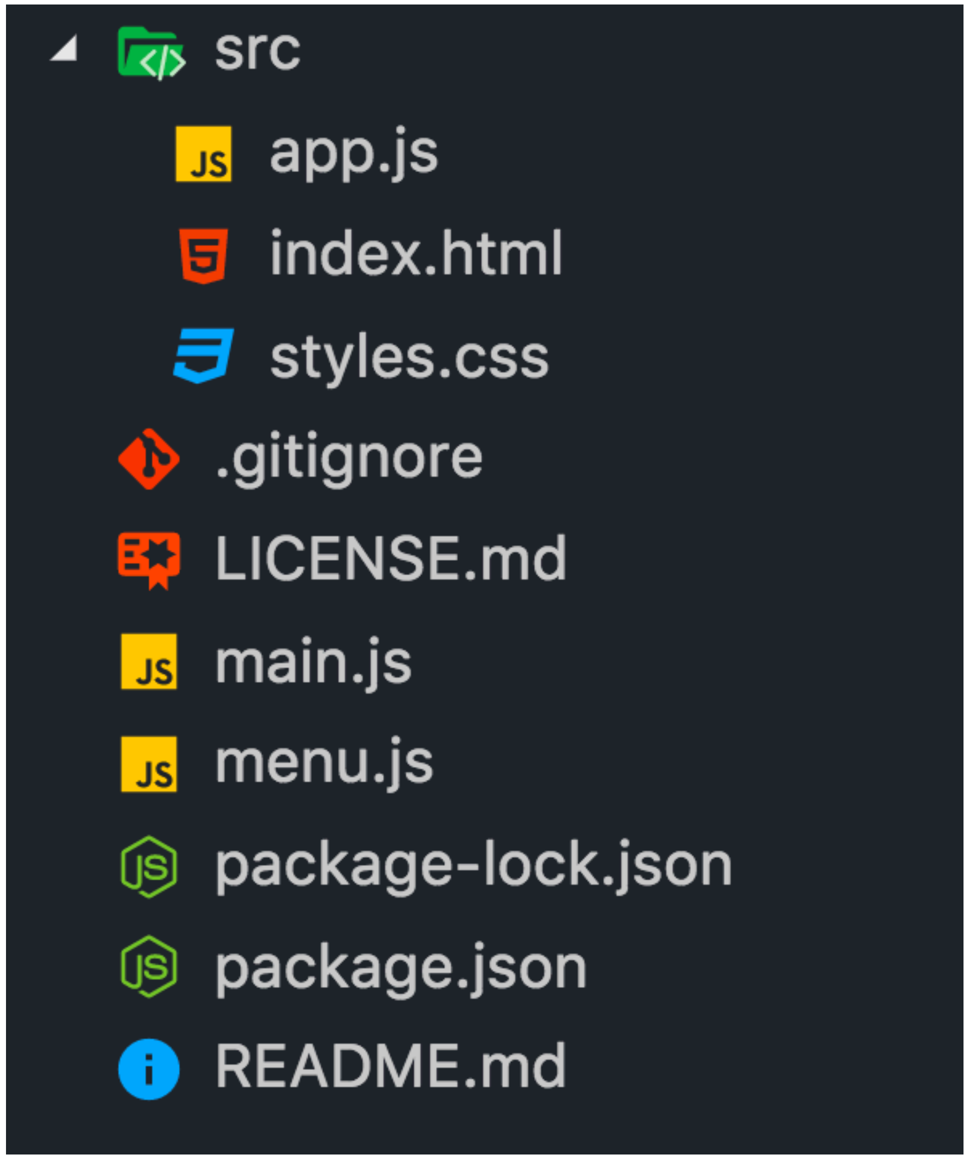

BrowserWindow
Actual instantiation and calling
main process
2. get reference to BrowserWindow
renderer process
3. call method on BrowserWindow
1. remote require BrowserWindow

*Creating a **BrowserWindow** in the renderer process*

LET’S BUILD AN APP

- App to measure CPU usage;
- Use Node os module to get logical CPU core information;
- Read and update data every ~1000ms;
- Show the usage of each cpu mode per core;
- Use Highcharts to build a stacked column chart with this information;
- Use electron-builder to package and create an installer.

BASIC FILE STRUCTURE


```
◢ src
    app.js
    index.html
    styles.css
  .gitignore
  LICENSE.md
  main.js
  menu.js
  package-lock.json
  package.json
  README.md
```

main.js - starts the app and creates a browser window to render HTML. This is the app's **main process**;

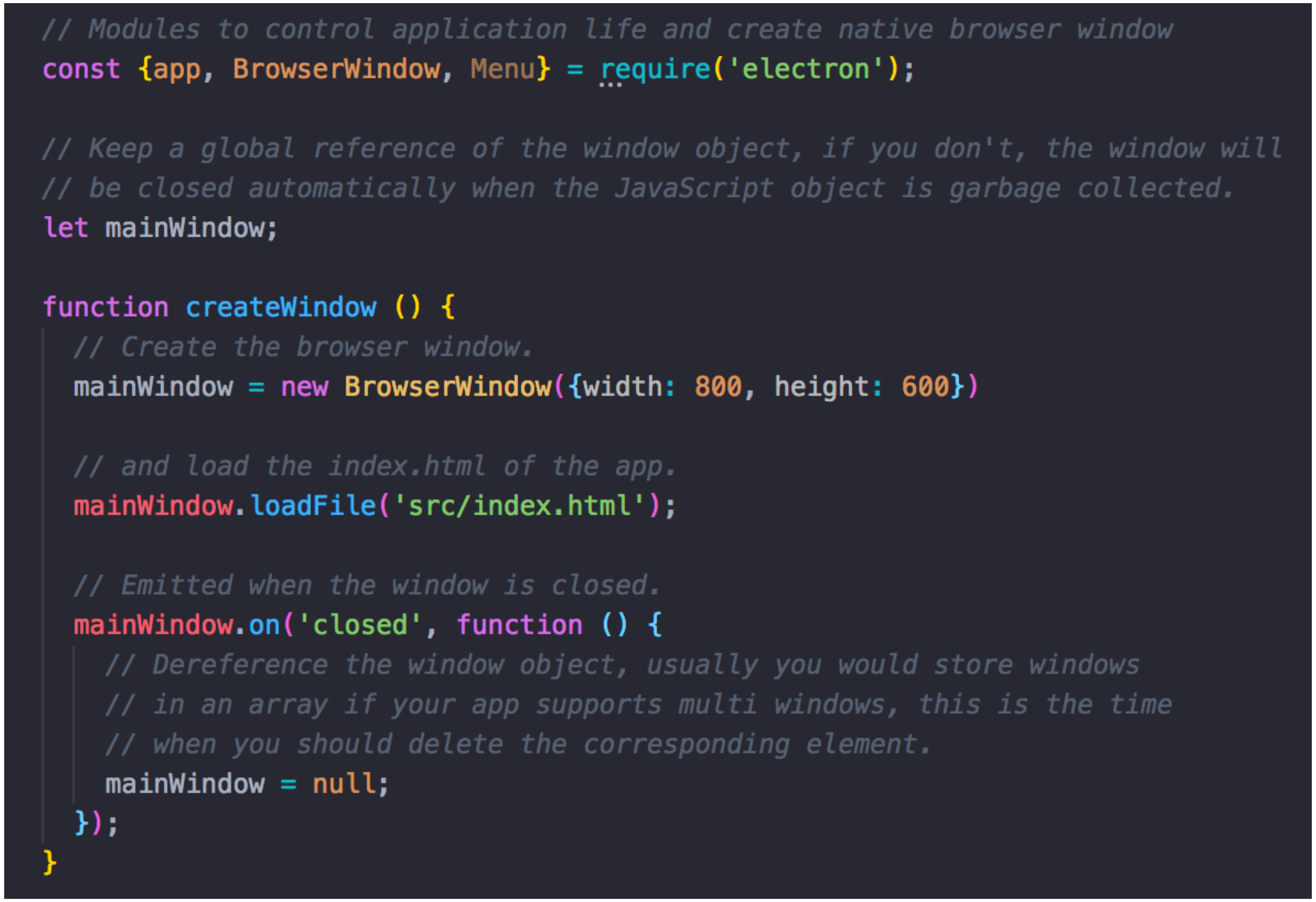
menu.js - menu template;

package.json - project info such as name, version, dependencies, etc;

src/ - web page code;

src/index.html - a web page to render. This is the app's **renderer process**;

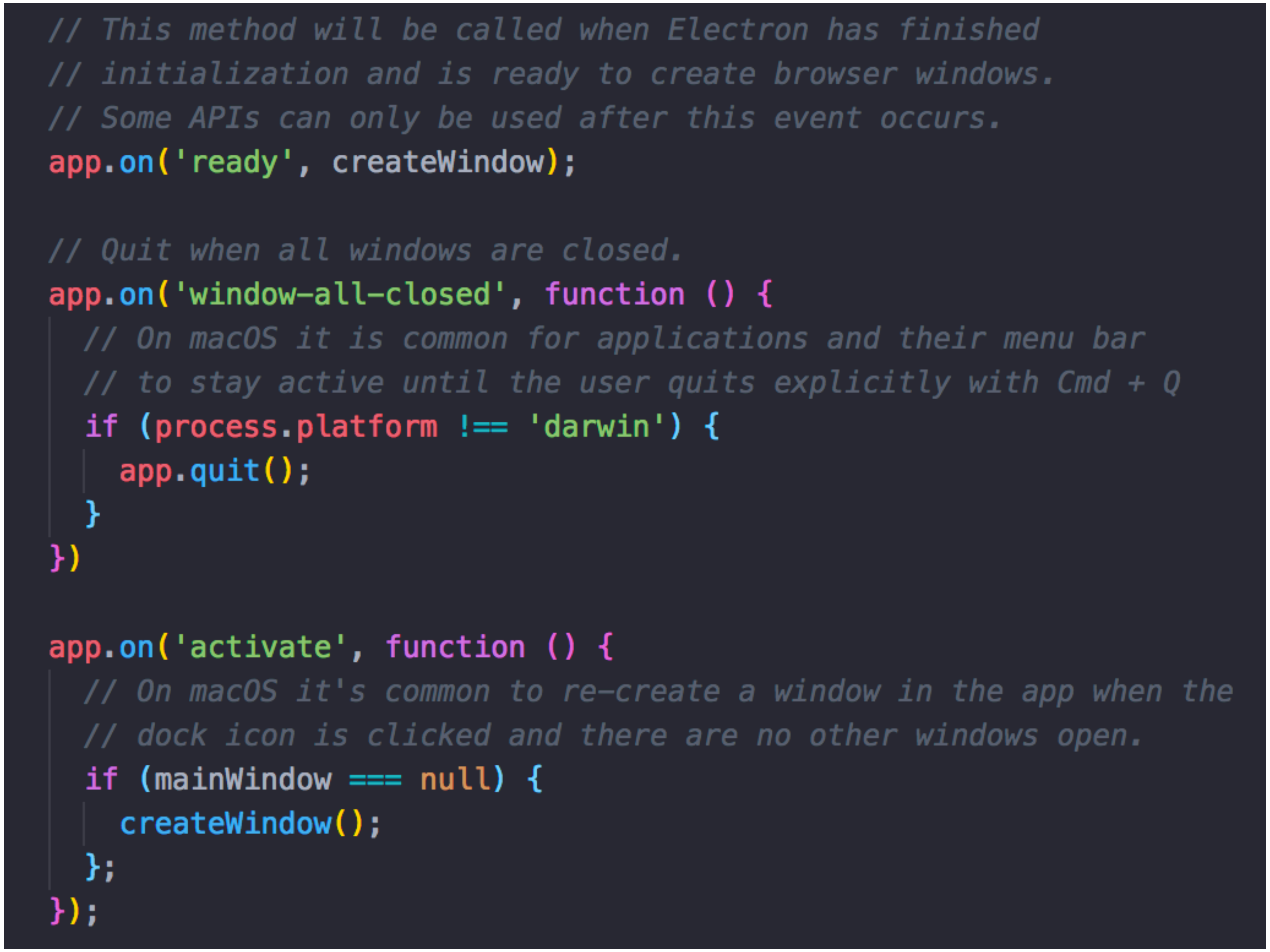
CREATING A WINDOW



```
// Modules to control application life and create native browser window
const {app, BrowserWindow, Menu} = require('electron');

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow;

function createWindow () {
  // Create the browser window.
  mainWindow = new BrowserWindow({width: 800, height: 600})

  // and load the index.html of the app.
  mainWindow.loadFile('src/index.html');

  // Emitted when the window is closed.
  mainWindow.on('closed', function () {
    // Dereference the window object, usually you would store windows
    // in an array if your app supports multi windows, this is the time
    // when you should delete the corresponding element.
    mainWindow = null;
  });
}
```

Creating a BrowserWIndow and loading a file into its webContent thus generating a renderer process

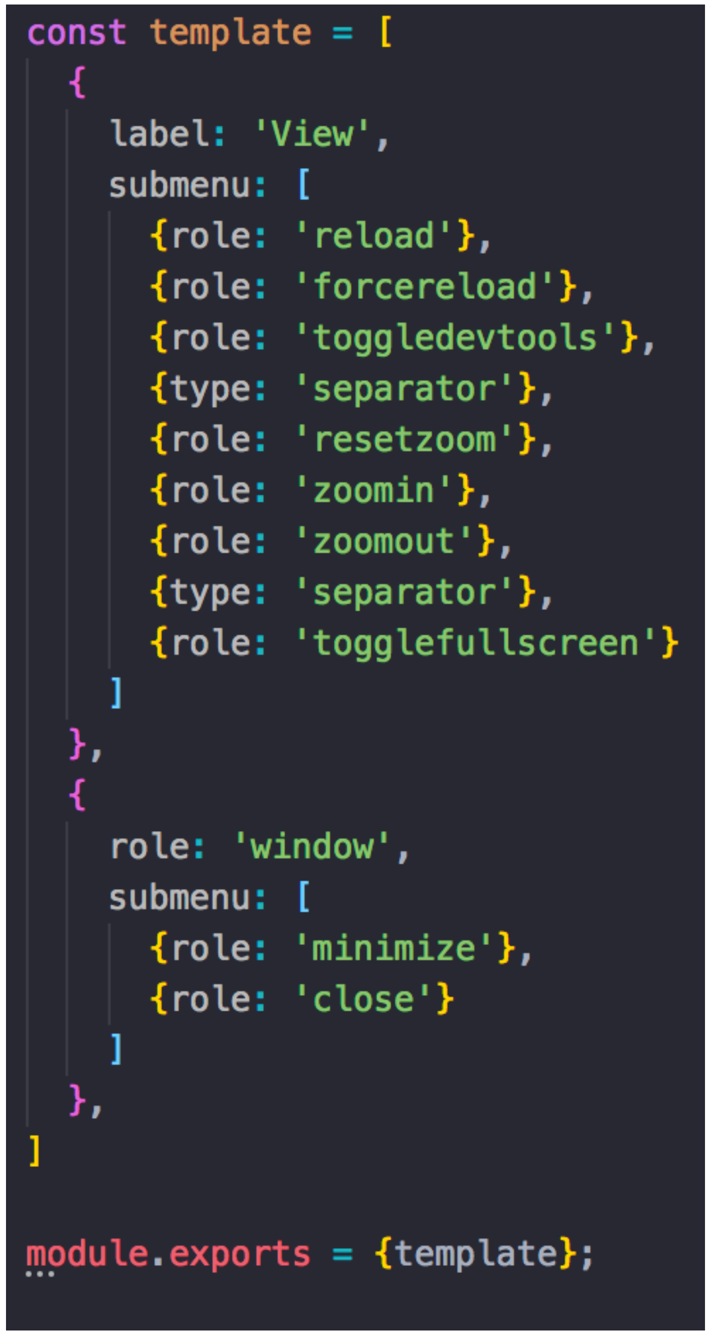
CREATING A WINDOW



```
// 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.on('ready', createWindow);

// Quit when all windows are closed.
app.on('window-all-closed', function () {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') {
    app.quit();
  }
})

app.on('activate', function () {
  // 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 (mainWindow === null) {
    createWindow();
  };
});
```

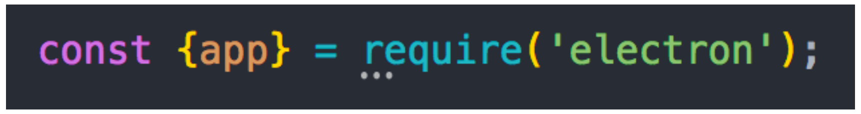
Binding the app to lifecycle events

BUILDING A MENU TEMPLATE



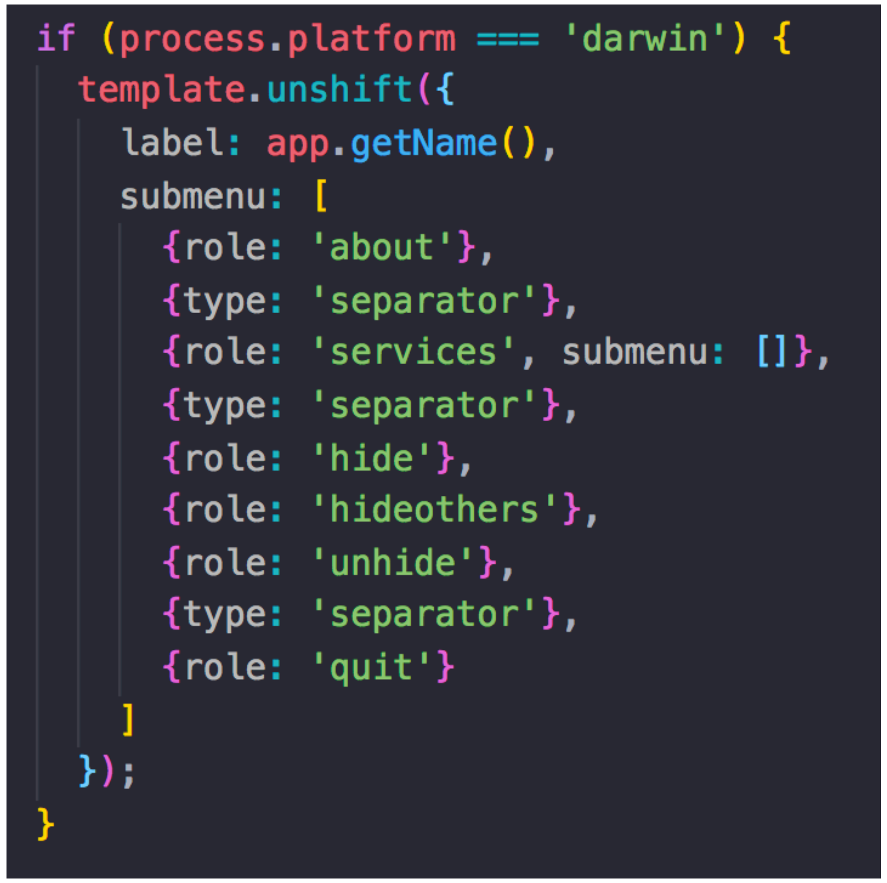
```
const template = [
  {
    label: 'View',
    submenu: [
      {role: 'reload'},
      {role: 'forcereload'},
      {role: 'toggledevtools'},
      {type: 'separator'},
      {role: 'resetzoom'},
      {role: 'zoomin'},
      {role: 'zoomout'},
      {type: 'separator'},
      {role: 'togglefullscreen'}
    ]
  },
  {
    role: 'window',
    submenu: [
      {role: 'minimize'},
      {role: 'close'}
    ]
  },
]

module.exports = {template};
```

menu.js



```
const {app} = require('electron');
```

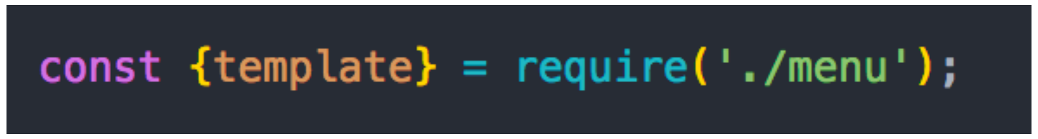
menu.js - special config for macOS



```
if (process.platform === 'darwin') {
  template.unshift({
    label: app.getName(),
    submenu: [
      {role: 'about'},
      {type: 'separator'},
      {role: 'services', submenu: []},
      {type: 'separator'},
      {role: 'hide'},
      {role: 'hideothers'},
      {role: 'unhide'},
      {type: 'separator'},
      {role: 'quit'}
    ]
  });
}
```

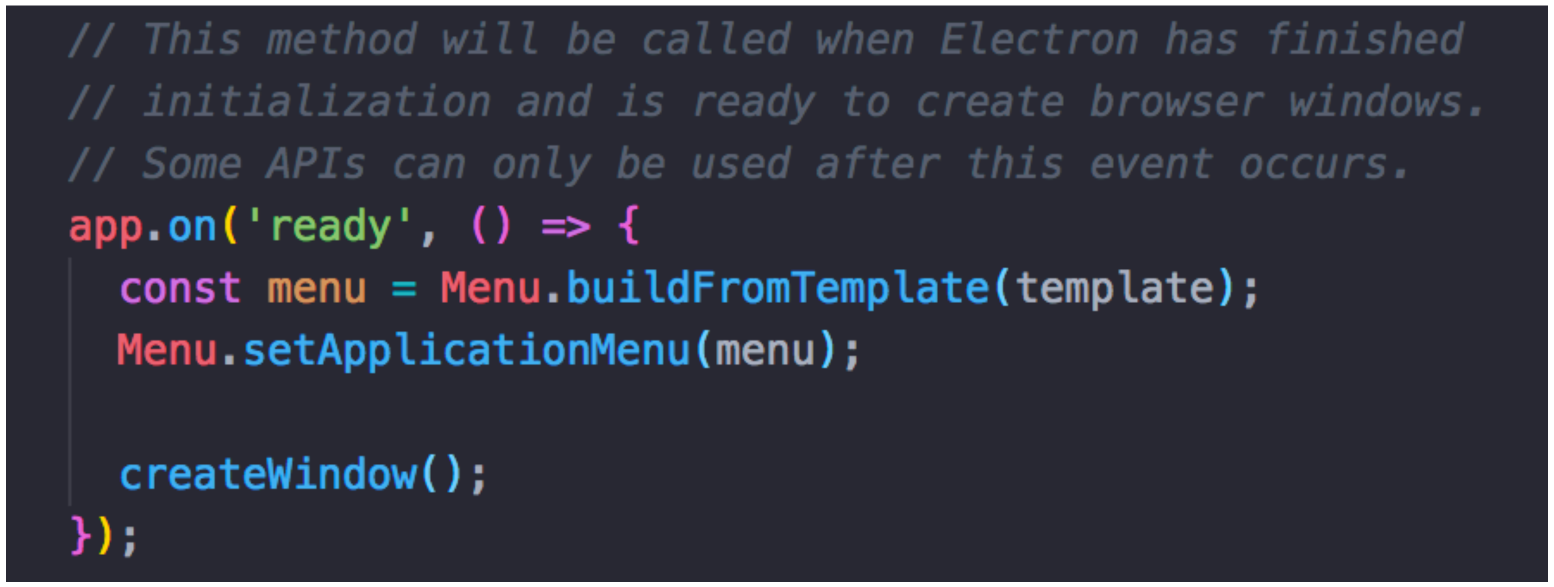
menu.js - special config for macOS

ADDING A MENU TO THE APP



```
const {template} = require('./menu');
```

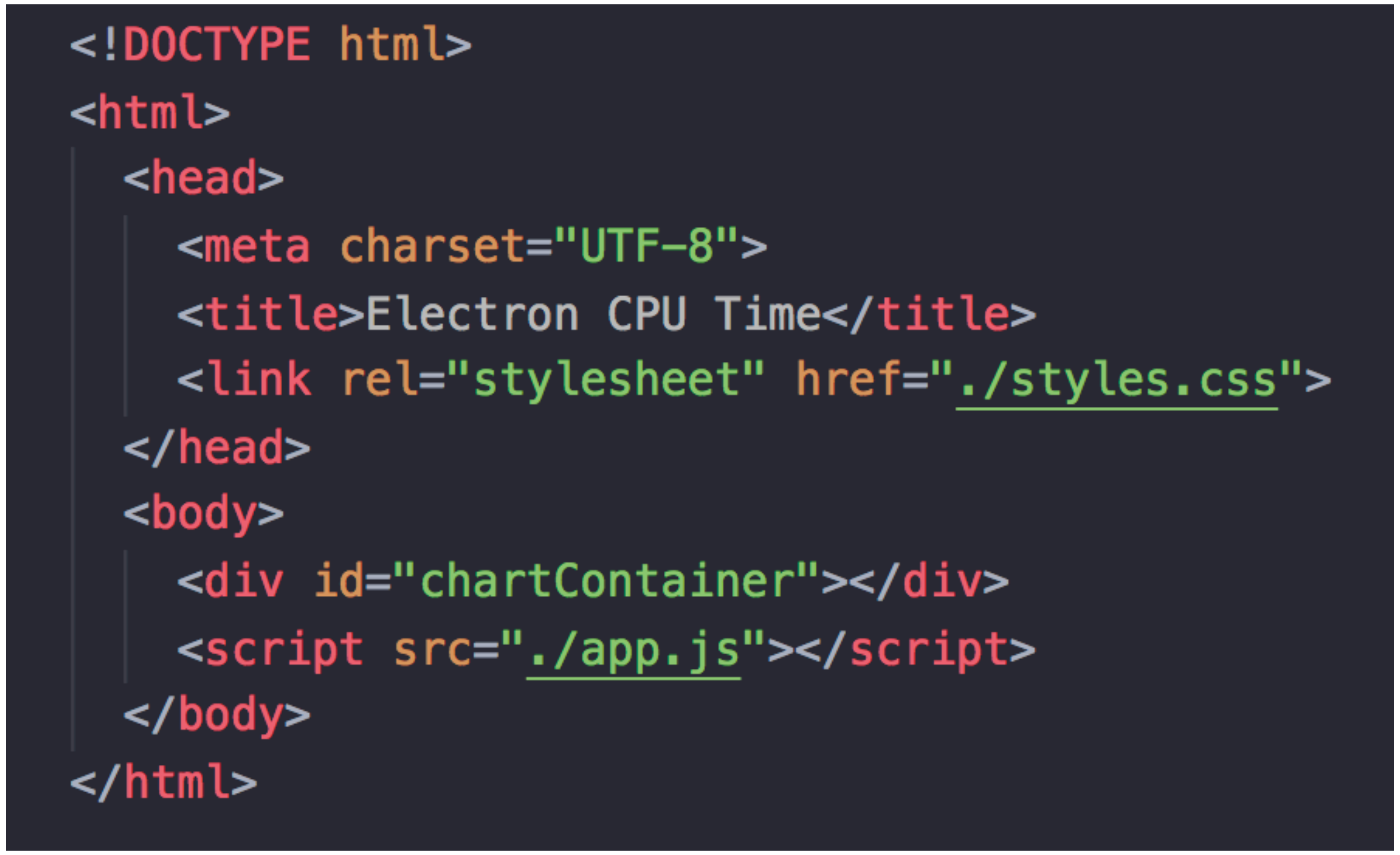
main.js



```
// 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.on('ready', () => {
  const menu = Menu.buildFromTemplate(template);
  Menu.setApplicationMenu(menu);

  createWindow();
});
```

main.js

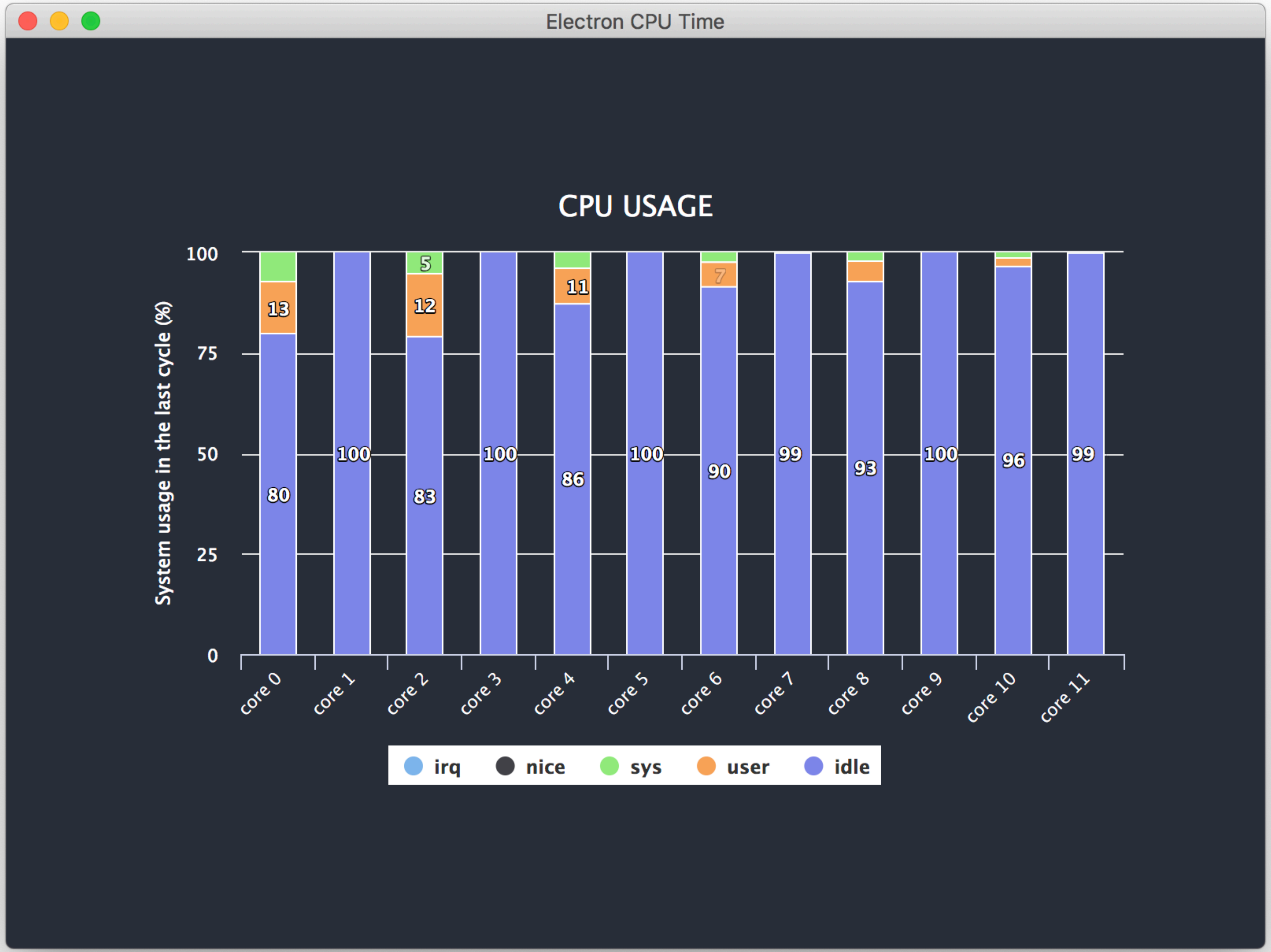
ADDING CODE TO THE WEBPAGE



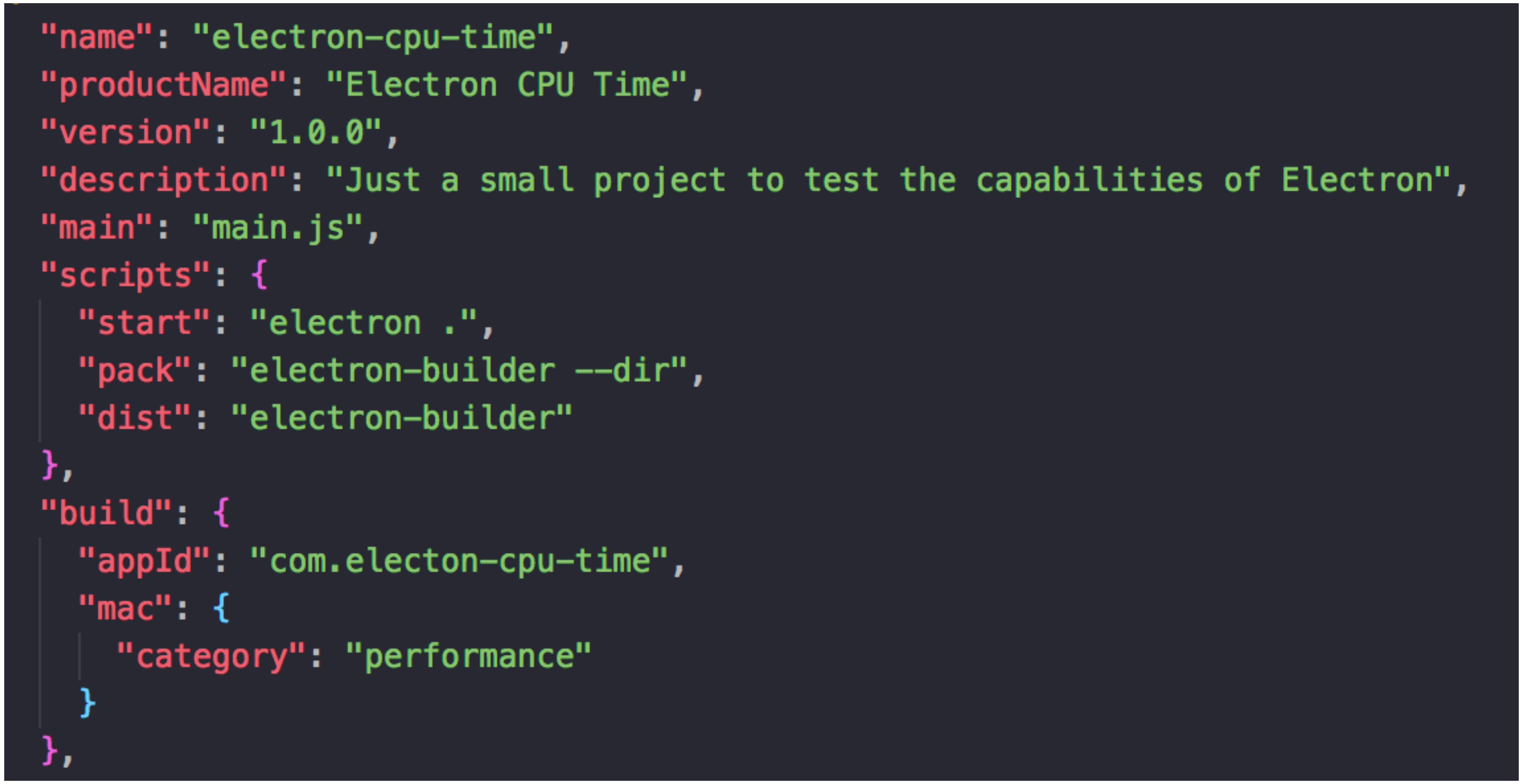
```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Electron CPU Time</title>
    <link rel="stylesheet" href="./styles.css">
  </head>
  <body>
    <div id="chartContainer"></div>
    <script src="./app.js"></script>
  </body>
</html>
```

index.html

RUNNING THE APP IN DEVELOPMENT MODE



PACKAGING THE APP - ELECTRON BUILDER

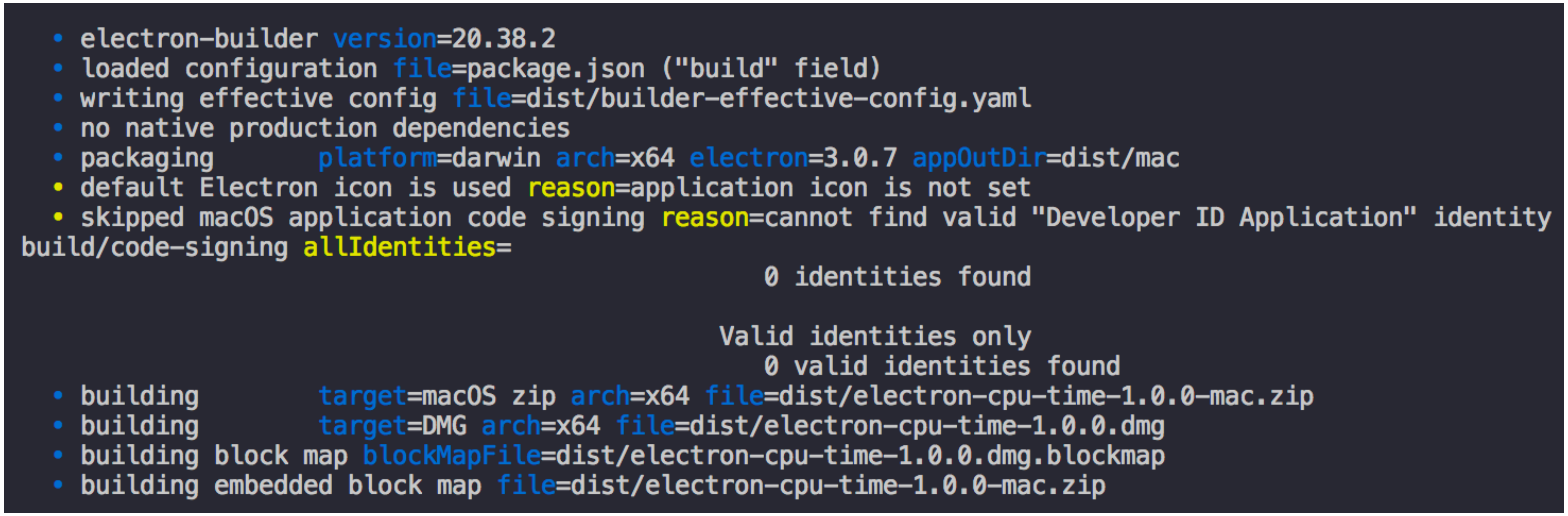


```
"name": "electron-cpu-time",
"productName": "Electron CPU Time",
"version": "1.0.0",
"description": "Just a small project to test the capabilities of Electron",
"main": "main.js",
"scripts": {
  "start": "electron .",
  "pack": "electron-builder --dir",
  "dist": "electron-builder"
},
"build": {
  "appId": "com.electon-cpu-time",
  "mac": {
    "category": "performance"
  }
},
```

package.json

CREATING AN INSTALLER

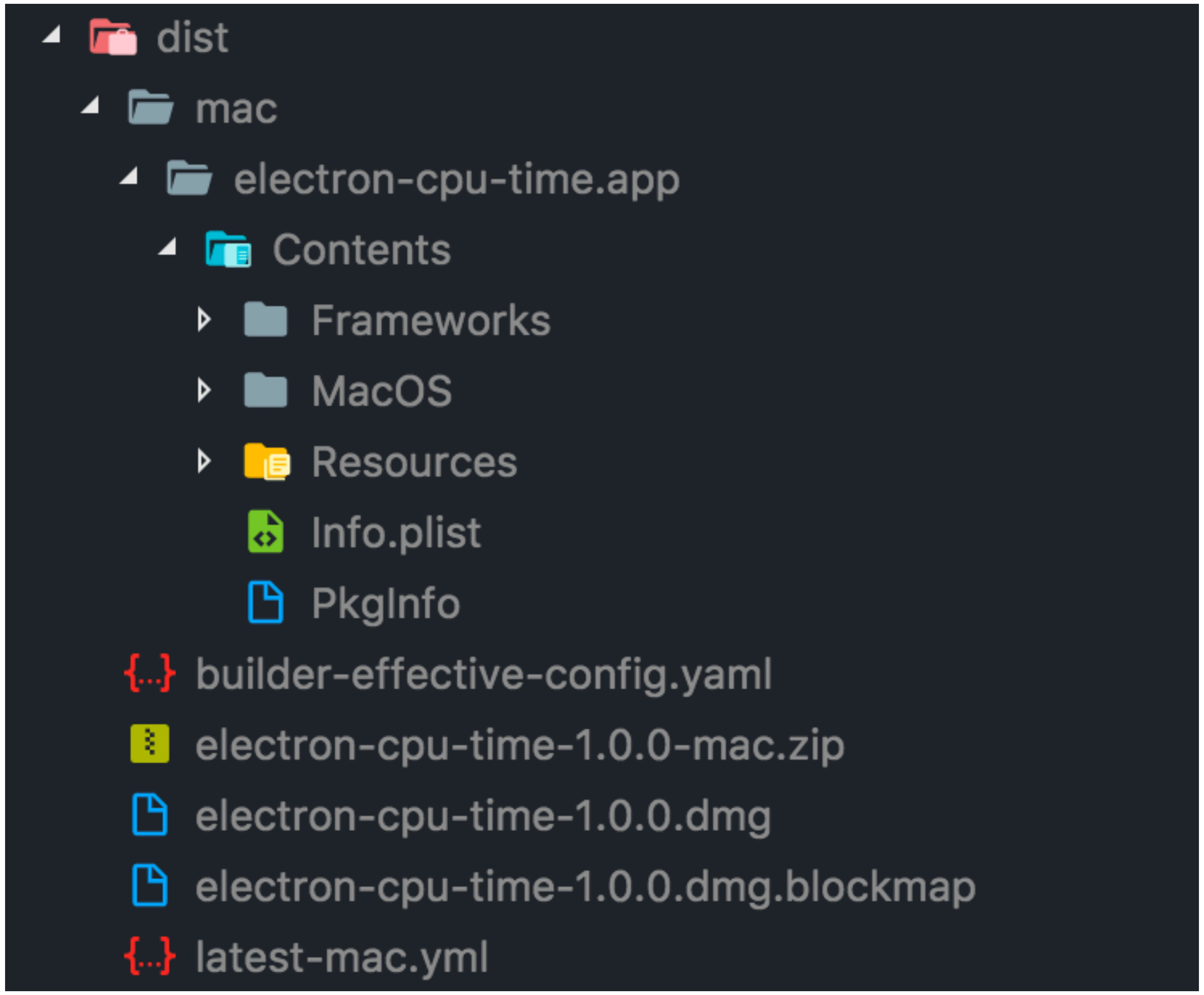
Run **npm run dist** in the terminal.



```
  • electron-builder version=20.38.2
  • loaded configuration file=package.json ("build" field)
  • writing effective config file=dist/builder-effective-config.yaml
  • no native production dependencies
  • packaging       platform=darwin arch=x64 electron=3.0.7 appOutDir=dist/mac
  • default Electron icon is used reason=application icon is not set
  • skipped macOS application code signing reason=cannot find valid "Developer ID Application" identity
build/code-signing allIdentities=
                                                  0 identities found

                                               Valid identities only
                                                  0 valid identities found
  • building        target=macOS zip arch=x64 file=dist/electron-cpu-time-1.0.0-mac.zip
  • building        target=DMG arch=x64 file=dist/electron-cpu-time-1.0.0.dmg
  • building block map blockMapFile=dist/electron-cpu-time-1.0.0.dmg.blockmap
  • building embedded block map file=dist/electron-cpu-time-1.0.0-mac.zip
```

CREATING AN INSTALLER

Run **npm run dist** in the terminal.



```
dist
  mac
    electron-cpu-time.app
      Contents
        Frameworks
        MacOS
        Resources
        Info.plist
        PkgInfo
  builder-effective-config.yaml
  electron-cpu-time-1.0.0-mac.zip
  electron-cpu-time-1.0.0.dmg
  electron-cpu-time-1.0.0.dmg.blockmap
  latest-mac.yml
```

THE FINAL RESULT

REFERENCES

- https://electronjs.org/
- https://medium.com/developers-writing/building-a-desktop-application-with-electron-204203eeb658
- https://medium.com/cameron-nokes/deep-dive-into-electrons-main-and-renderer-processes-7a9599d5c9e2
- https://www.chromium.org/developers/design-documents/inter-process-communication
- https://jlord.us/essential-electron/#what-is-electron-
- https://github.com/electron/electron-quick-start

QUESTIONS?