Electron

Romain Bohdanowicz
Twitter : @bioub - Github : https://github.com/bioub
http://formation.tech/

Introduction

Introduction - What is Electron ?

- Platform to build desktop applications that runs on Windows, Mac and Linux
- Created in 2013 by Cheng Zhao, engineer at Github to build the Atom Editor
- Allows to build GUIs with web technologies (HTML, CSS, JS) that have access to native operating system capabilities (through Node.js and Electron APIs)
- Combines Node.js, Chromium and specific APIs
- Inspired by Apache Cordova, Adobe PhoneGap or Ionic Capacitor in the mobile ecosystem
- Alternatives : NW.js, React Native for Windows + macOS, Tauri

Introduction - Made with Electron

Figma

Discord

appium

Microsoft Teams

WhatsApp

mongoDB Compass

InVision

slack

WordPress

ATOM

THEIA

1Password

Visual Studio Code

GitKraken

POSTMAN

ZEPLIN

Signal

Yammer

GitHub Desktop

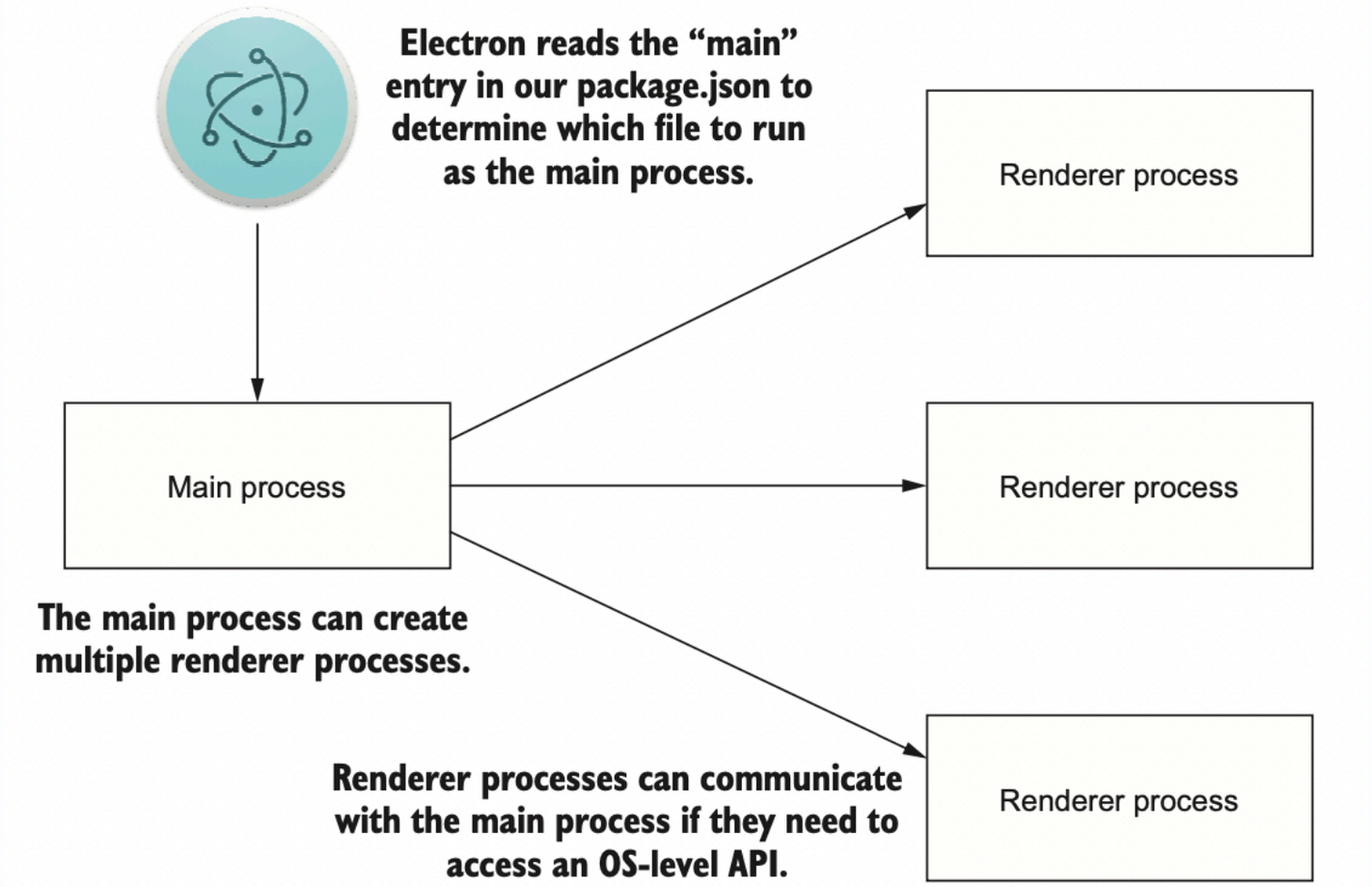
skype

Introduction - Made with Electron

```
➜  ~ sudo find /Applications -name "Electron Framework.framework"
/Applications/Arduino IDE.app/Contents/Frameworks/Electron Framework.framework
/Applications/ClickUp.app/Contents/Frameworks/Electron Framework.framework
/Applications/Dropbox.app/Contents/Frameworks/Electron Framework.framework
/Applications/Electron Fiddle.app/Contents/Frameworks/Electron Framework.framework
/Applications/Evernote.app/Contents/Frameworks/Electron Framework.framework
/Applications/Figma.app/Contents/Frameworks/Electron Framework.framework
/Applications/Microsoft Teams.app/Contents/Frameworks/Electron Framework.framework
/Applications/MongoDB Compass.app/Contents/Frameworks/Electron Framework.framework
/Applications/Postman.app/Contents/Frameworks/Electron Framework.framework
/Applications/Prisma Studio.app/Contents/Frameworks/Electron Framework.framework
/Applications/Skype.app/Contents/Frameworks/Electron Framework.framework
/Applications/Slack.app/Contents/Frameworks/Electron Framework.framework
/Applications/Todoist.app/Contents/Frameworks/Electron Framework.framework
/Applications/Trello.app/Contents/Frameworks/Electron Framework.framework
/Applications/Visual Studio Code.app/Contents/Frameworks/Electron Framework.framework
/Applications/WhatsApp.app/Contents/Frameworks/Electron Framework.framework
```

Introduction - How does Electron work?

- Two kind of processes: the main process and zero or more renderer processes
- Some abilities are available in both types of processes (JS APIs, read/write clipboard, HTTP requests...)
- Others are limited to the main process (Filesystem, low-level network access...)



Introduction - Processes responsibilities

- Main process
  - Respond to application lifecycle events : starting up, quitting, going to background/foreground...
  - Communicates with the OS APIs
  - Manage renderer processes
- Renderer process
  - Loads and display GUI
  - May have access to Node API (not recommended)

Introduction - Electron Versions

- Electron follows the SemVer spec

Major Version Increments	Minor Version Increments	Patch Version Increments
Electron breaking API changes	Electron non-breaking API changes	Electron bug fixes
Node.js major version updates	Node.js minor version updates	Node.js patch version updates
Chromium version updates		fix-related chromium patches

- 5 to 6 releases per year (8 week cadence), 4 active version (3 stable, 1 alpha/beta)

https://www.electronjs.org/docs/latest/tutorial/electron-timelines

Electron	Alpha	Beta	Stable	Chrome	Node	Supported
17.0.0	2021-Nov-18	2022-Jan-06	2022-Feb-01	M98	v16.13	🚫
18.0.0	2022-Feb-03	2022-Mar-03	2022-Mar-29	M100	v16.13	✅
19.0.0	2022-Mar-31	2022-Apr-26	2022-May-24	M102	v16.14	✅
20.0.0	2022-May-26	2022-Jun-21	2022-Aug-02	M104	v16.15	✅
21.0.0	2022-Aug-04	2022-Aug-30	2022-Sep-27	M106	TBD	✅

Our first App

Our first App - How to create an Electron App ?

- Install the npm package as a dev dependency :
  npm install electron --save-dev
- Use the official Quick Start skeleton
  https://github.com/electron/electron-quick-start
- Use the Webpack Quick Start from Electron Builder
  https://github.com/electron-userland/electron-webpack-quick-start
  https://www.electron.build/#boilerplates
- Use Electron Forge (recommended by the Electron team)
  https://www.electronforge.io/
- To try and learn, use Electron Fiddle :
  https://www.electronjs.org/fr/fiddle

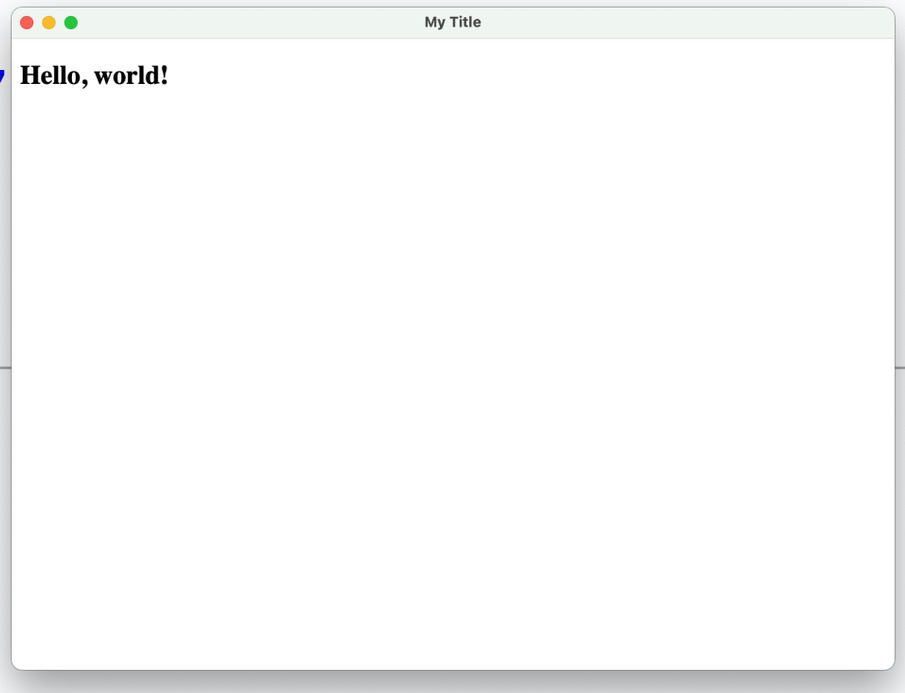
Our first App - Minimal application

- npx electron main.js

```
// main.js
const { app, BrowserWindow } = require('electron');
const path = require('path');

app.whenReady().then(() => {
  const mainWindow = new BrowserWindow();
  mainWindow.loadFile(path.resolve(__dirname, 'index.html'))
});
```

```
<!-- index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
  <title>My Title</title>
</head>
<body>
  <h2>Hello, world!</h2>
</body>
</html>
```



Our first App - macOS conventions

- Don't quit the app when all windows are closed
- Create a new window when the app is activated and all windows are closed

```
// main.js
const { app, BrowserWindow } = require('electron');
const path = require('path');

function createMainWindow() {
  const mainWindow = new BrowserWindow();
  mainWindow.loadFile(path.resolve(__dirname, 'index.html'))
}

app.whenReady().then(() => {
  createMainWindow();
});

app.on('activate', () => {
  if (BrowserWindow.getAllWindows().length === 0) {
    createMainWindow();
  }
});

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
});
```

Main process

Main process - Introduction

- Responsibilities
  - Respond to application lifecycle events : starting up, quitting, going to background/foreground...
  - Communicates with the OS APIs
  - Manage renderer processes
- Have access to Node.js APIs (require, exports...)
- Doesn't support ESM Modules : https://github.com/electron/electron/issues/21457#issuecomment-1099904505

Main process - app

- The app object control the application's event lifecycle
- Main events :
  - ready
    Emitted once, when Electron has finished initializing, can be accessed through isReady (boolean) and whenReady (Promise<boolean>), we will create the Window here
  - window-all-closed
    Quit the app, expect on macOS

```
// quitting the app when no windows are open on non-macOS
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
})
```

  - before-quit
    Before windows are closed, can prevent quitting using event.preventDefault()
  - activate (macOS only)
    launching the application for the first time, attempting to re-launch the application when it's already running, or clicking on the application's dock or taskbar icon.

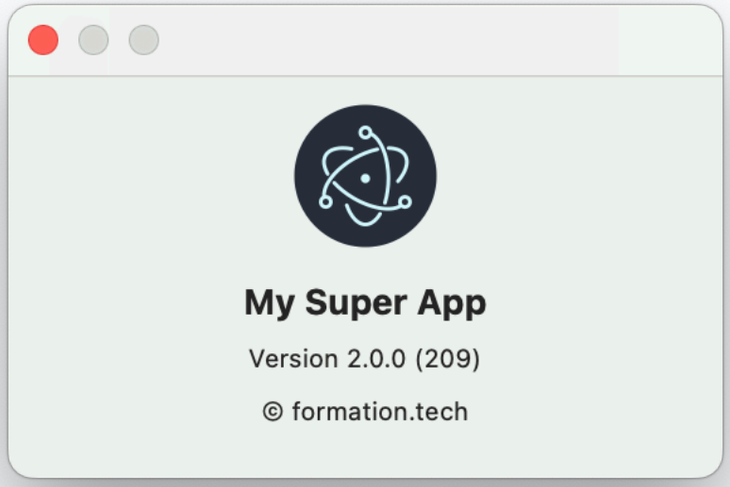
Main process - app

- app.quit()
  Try to close all windows
- app.isReady()
  Returns boolean - true if Electron has finished initializing
- app.whenReady()
  Same as isReady with a Promise
- app.getAppPath()
  The current application directory, give access to sources even if packaged
- app.getFileIcon()
  Fetches a path's associated icon.
- app.getVersion() / app.getName()
  version and name in the package.json file

Main process - app

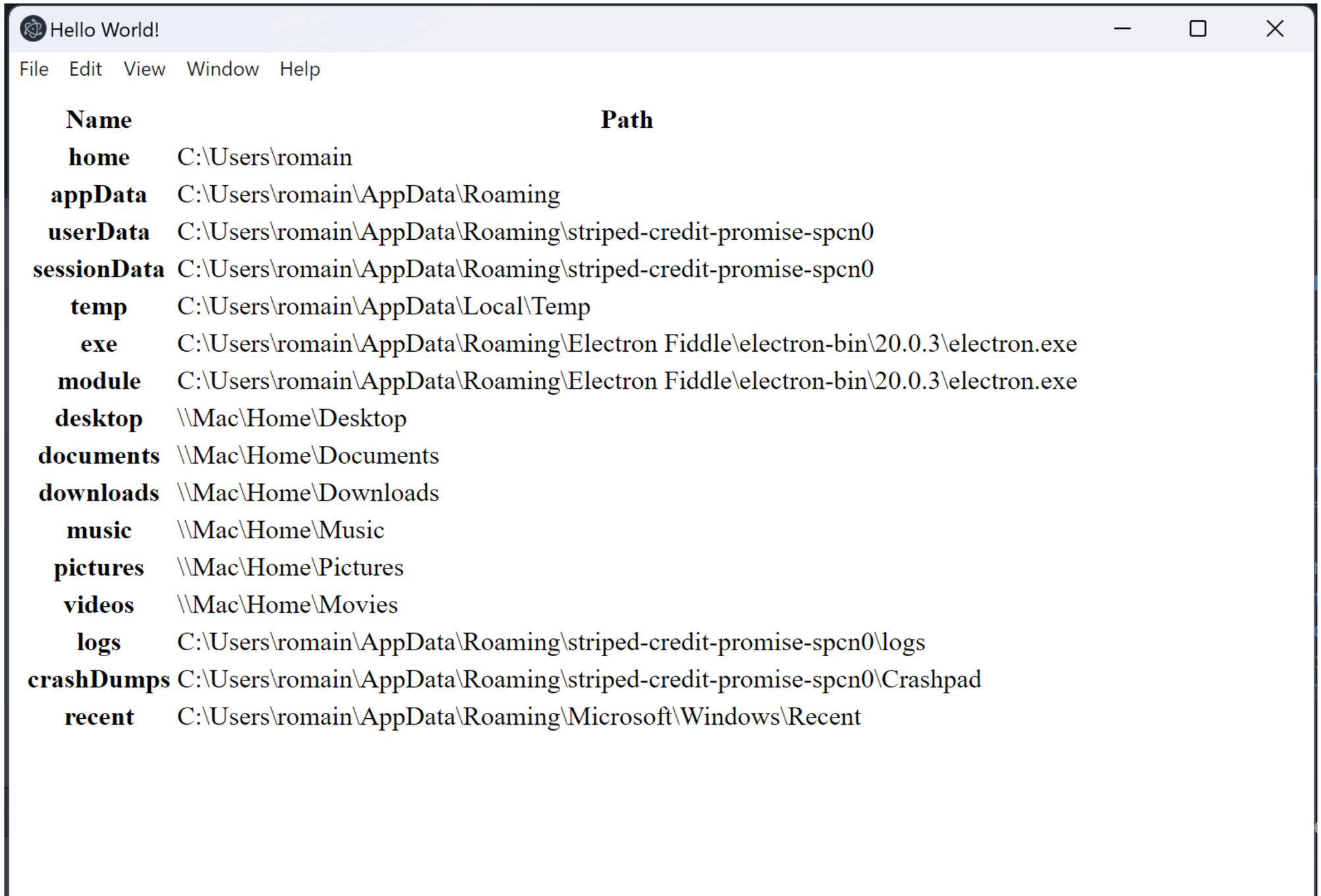
- app.setAboutPanelOptions()

```
app.setAboutPanelOptions({
  applicationName: 'My Super App',
  applicationVersion: '2.0.0',
  copyright: '© formation.tech',
  version: '209',
})
```



Main process - app

- app.getPath(name)
  Directories to store data, for example : userData, temp, logs, crashDumps...



Main process - app

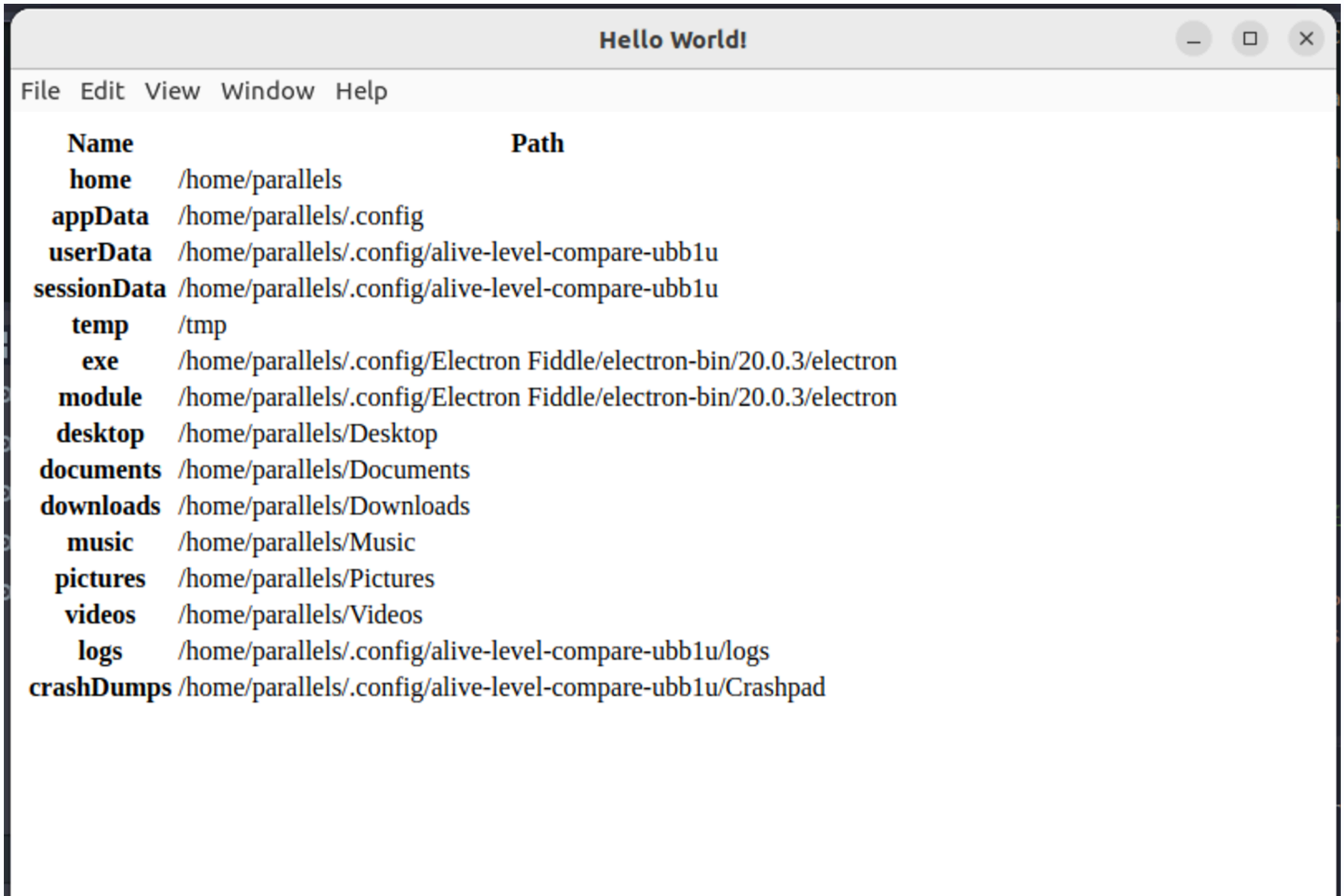
- app.getPath(name)
  Directories to store data, for example : userData, temp, logs, crashDumps...

Hello World!

Name	Path
home	/Users/romain
appData	/Users/romain/Library/Application Support
userData	/Users/romain/Library/Application Support/marked-branch-wave-u9ls3
sessionData	/Users/romain/Library/Application Support/marked-branch-wave-u9ls3
temp	/var/folders/nb/b3_dc91n22d5_lhsvyc07xkc0000gn/T/
exe	/Users/romain/Library/Application Support/Electron Fiddle/electron-bin/20.0.3/Electron.app/Contents/MacOS/Electron
module	/Users/romain/Library/Application Support/Electron Fiddle/electron-bin/20.0.3/Electron.app/Contents/Frameworks/Electron Framework.framework/Versions/A/Electron Framework
desktop	/Users/romain/Desktop
documents	/Users/romain/Documents
downloads	/Users/romain/Downloads
music	/Users/romain/Music
pictures	/Users/romain/Pictures
videos	/Users/romain/Movies
logs	/Users/romain/Library/Logs/marked-branch-wave-u9ls3
crashDumps	/Users/romain/Library/Application Support/marked-branch-wave-u9ls3/Crashpad

Main process - app

- app.getPath(name)
  Directories to store data, for example : userData, temp, logs, crashDumps...



Main process - BrowserWindow

- To create a new renderer process we can use BrowserWindow
- It can load a local HTML file

```
const mainWindow = new BrowserWindow();
mainWindow.loadFile('index.html');
```

- Or load a web page through HTTP

```
const mainWindow = new BrowserWindow();
mainWindow.loadURL('https://www.google.com/');
```

- To prevent the page from blinking (because assets are not fully loaded)

```
const mainWindow = new BrowserWindow({
  show: false
});

mainWindow.loadFile('index.html');

mainWindow.on('ready-to-show', () => {
  mainWindow.show();
});
```

Main process - BrowserWindow

- A lot of options to set the appearance :

```
const mainWindow = new BrowserWindow({
  width: 800,
  minWidth: 700,
  maxWidth: 900,
  height: 1000,
  minHeight: 900,
  maxHeight: 1100,
  maximizable: true,
  minimizable: true,
  movable: true,
  center: true,
  resizable: true,
});
```

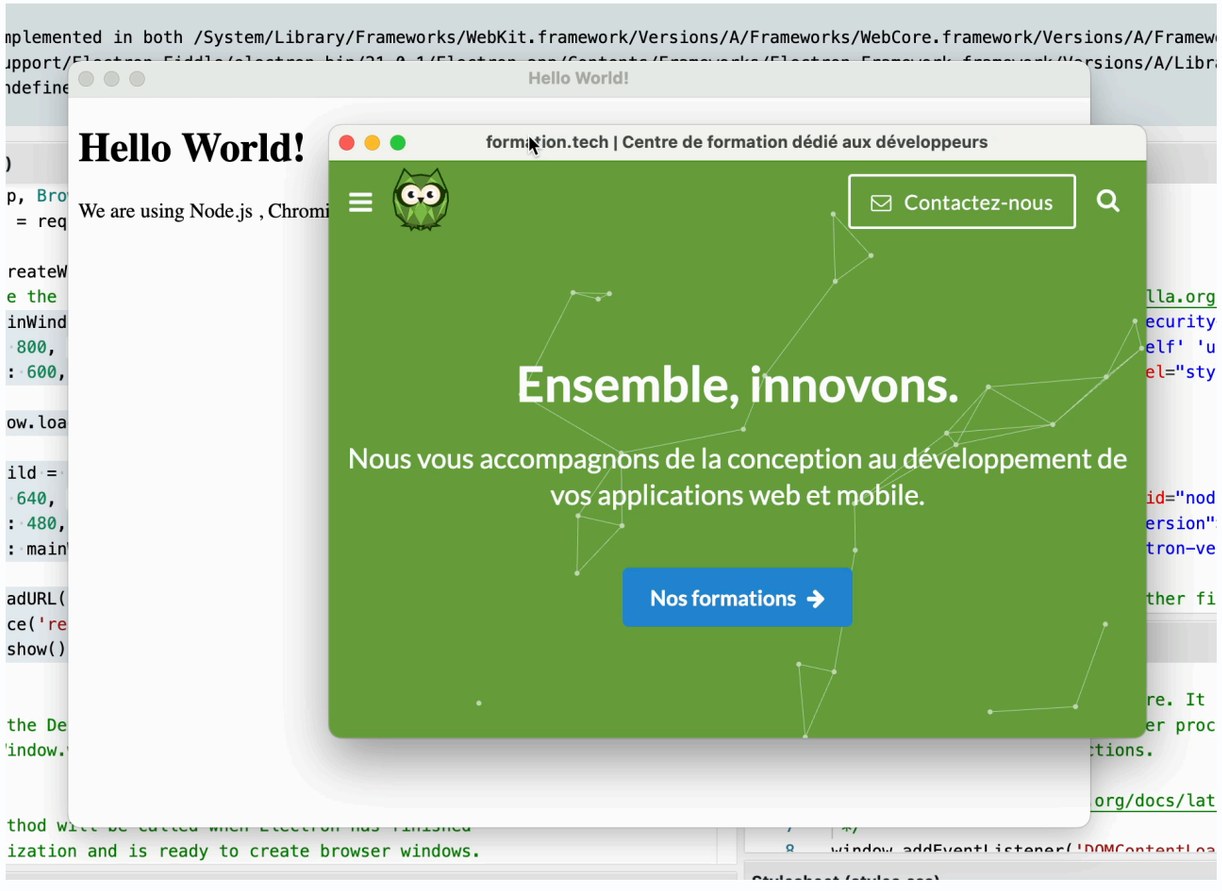
- Some are more OS related (macOS, Windows...) : https://www.electronjs.org/docs/latest/tutorial/window-customization

Main process - BrowserWindow

- Parent and child windows
  A child window will be always on top of the parent window

```
const mainWindow = new BrowserWindow({
  width: 800,
  height: 600,
});
mainWindow.loadFile('index.html');

const child = new BrowserWindow({
  width: 640,
  height: 480,
  parent: mainWindow, show: false
});
child.loadURL('https://formation.tech/')
child.once('ready-to-show', () => {
  child.show();
});
```

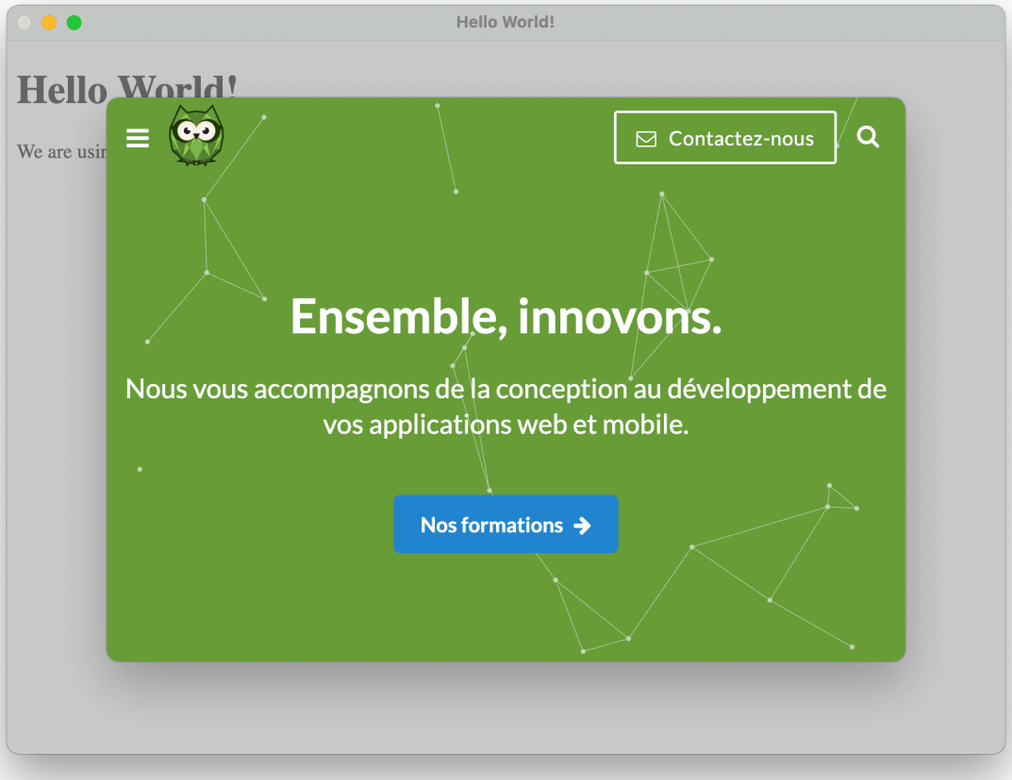


Main process - BrowserWindow

- To create modals :

```
const mainWindow = new BrowserWindow({
  width: 800,
  height: 600,
});
mainWindow.loadFile('index.html');

const child = new BrowserWindow({
  width: 640,
  height: 480,
  parent: mainWindow, modal: true, show: false
});
child.loadURL('https://formation.tech/')
child.once('ready-to-show', () => {
  child.show();
});
```



Main process - BrowserWindow

- The webPreferences options controls which features will be available in the web page :
  - devTools
  - Node.js APIs (beware of security)
  - Some uncommon Web APIs (WebSQL, WebGL...)
  - attach a preload script

```
const mainWindow = new BrowserWindow({
  webPreferences: {
    preload: path.join(__dirname, 'preload.js'),
    devTools: false,
  },
});
```

Main process - dialogs

- Electron can use system dialogs
  - showOpenDialog
  - showSaveDialog
  - showMessageBox
  - showErrorBox

Main process - Menus

- Menu Items
  https://www.electronjs.org/docs/latest/api/menu-item
  - menuItem.id
  - menuItem.label
  - menuItem.click (function)
  - menuItem.submenu
  - menuItem.type (normal, separator, submenu, checkbox or radio)
  - menuItem.role (undo, redo, cut, copy, paste, pasteAndMatchStyle, delete, selectAll, reload, forceReload, toggleDevTools, resetZoom, zoomIn, zoomOut, toggleSpellChecker, togglefullscreen, window, minimize, close, help, about, services, hide, hideOthers, unhide, quit, startSpeaking, stopSpeaking, zoom, front, appMenu, fileMenu, editMenu, viewMenu, shareMenu, recentDocuments, toggleTabBar, selectNextTab, selectPreviousTab, mergeAllWindows, clearRecentDocuments, moveTabToNewWindow or windowMenu)
  - menuItem.accelerator
    CommandOrControl+A
    CommandOrControl+Shift+Z
    https://www.electronjs.org/docs/latest/api/accelerator

Main process - Menus

- Menu.getApplicationMenu()
  MenuItems can be edited dynamically, but we have to call
  Menu.setApplicationMenu() or win.setMenu() to add or remove items
- Menu.setApplicationMenu()
  Set the menu for all windows
- win.setMenu()
  Set the menu for a specific window

```
const menu = Menu.getApplicationMenu();

const menuItem = new MenuItem({
  label: "Disable me",
  click() { menuItem.enabled = false }
});

menu.items.find((item) => item.role === 'filemenu').submenu.append(menuItem);

Menu.setApplicationMenu(menu);
```

Renderer Process

Renderer Process - Introduction

- Each window is rendered in a separate process called "Renderer Process"
- A renderer process has access to Web APIs
- By default those processes are run in context isolation which means they don't share global variables with other processes (e.g. window, document...)
- In recent versions of Electron those processes don't have access to Node APIs (ex
- contextIsolation and nodeIntegration can be controlled through the webPreference option of BrowserWindow

```
const mainWindow = new BrowserWindow({
  webPreferences: {
    contextIsolation: true,
    nodeIntegration: false,
  }
});
```

Renderer Process - Preload

- Since the windows are run in separate processes, they can't discuss with the main process
- A preload script can be used as a bridge between the main and renderer processes
- The script will run within the renderer context, but will have access to some Node.js APIs

```
const mainWindow = new BrowserWindow({
  webPreferences: {
    preload: path.resolve(__dirname, 'preload.js')
  }
});
```

- The code in the preload script will be wrapped into a function that have access to Node APIs

```
(function (require, process, Buffer, global, setImmediate, clearImmediate, exports) {

});
```

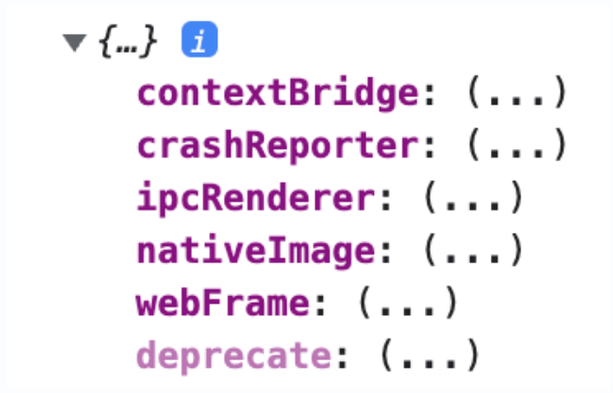
Renderer Process - Preload

- The require function is a special implementation that only have access to few APIs

```
console.log(require('electron'));
console.log(require('events'));
console.log(require('timers'));
console.log(require('url'));
```

- require('electron') will return a limited version of electron

```
console.log(require('electron'));
```



Process communication

Process communication - Introduction

- Since processes are isolated from each other, we will have to create a bridge within the main process and renderer processes
- This can be achieved using a preload script

```
const mainWindow = new BrowserWindow({
  webPreferences: {
    preload: path.resolve(__dirname, 'preload.js')
  }
});
```

- A preload script have access to the contextBridge API which allows to create a global variable within the renderer process

```
// preload.js
contextBridge.exposeInMainWorld('myObj', {
  sum: (a, b) => a + b,
});

// renderer.js
myObj.sum(1, 2); // 3
```

Process communication - Renderer to main (1 way)

- Messages can be sent from renderer to main using the ipcRenderer.send method
- In that case, the renderer doesn't expect any return

```
// preload.js
contextBridge.exposeInMainWorld('myApp', {
  quit() {
    ipcRenderer.send('quit');
  },
});
```

```
// renderer.js
myApp.quit();

// main.js
ipcMain.on('quit', () => app.quit());
```

Process communication - Renderer to main (1 way)

- The send method can take a parameter

```
// preload.js
contextBridge.exposeInMainWorld('myApp', {
  quitAsync() {
    ipcRenderer.send('quitAsync', 2000);
  },
});

// renderer.js
myApp.quitAsync();
```

```
// main.js
ipcMain.on('quitAsync', (event, delay) => {
  setTimeout(() => {
    app.quit()
  }, delay);
});
```

Process communication - Renderer to main (2 ways)

- The renderer process can get a return from the main process using the invoke method of ipcRenderer
- This method returns a promise so the operation in main can be async
- Be careful with security issues (this example give access to any file on the filesystem)

```
// preload.js
contextBridge.exposeInMainWorld('myApp', {
  sum(file) {
    return ipcRenderer.invoke('readFile', file);
  },
});

// renderer.js
await myApp.readFile('example.txt');

// main.js
ipcMain.handle('readFile', (event, file) => {
  return fs.readFile(file, { encoding: 'utf-8' });
});
```

Process communication - Main to renderer

- The main process can be the origin
- To send data to all renderer processes it can use ipcMain.send
- To send data to a specific process it can use win.webContents.send

```
// main.js
mainWindow.webContents.send('random', Math.random());

// preload.js
contextBridge.exposeInMainWorld('myApp', {
  getRandom(cb) {
    return ipcRenderer.on('random', (event, nb) => {
      cb(nb);
    });
  },
});

// renderer.js
await myApp.getRandom((randomNb) => {
  console.log(randomNb);
});
```

Packaging

Packaging - Introduction

- 3 solutions to package Electron apps :
  - electron-packager
    https://github.com/electron/electron-packager
  - electron-builder
    https://www.electron.build/
  - Electron Forge
    https://www.electronforge.io/

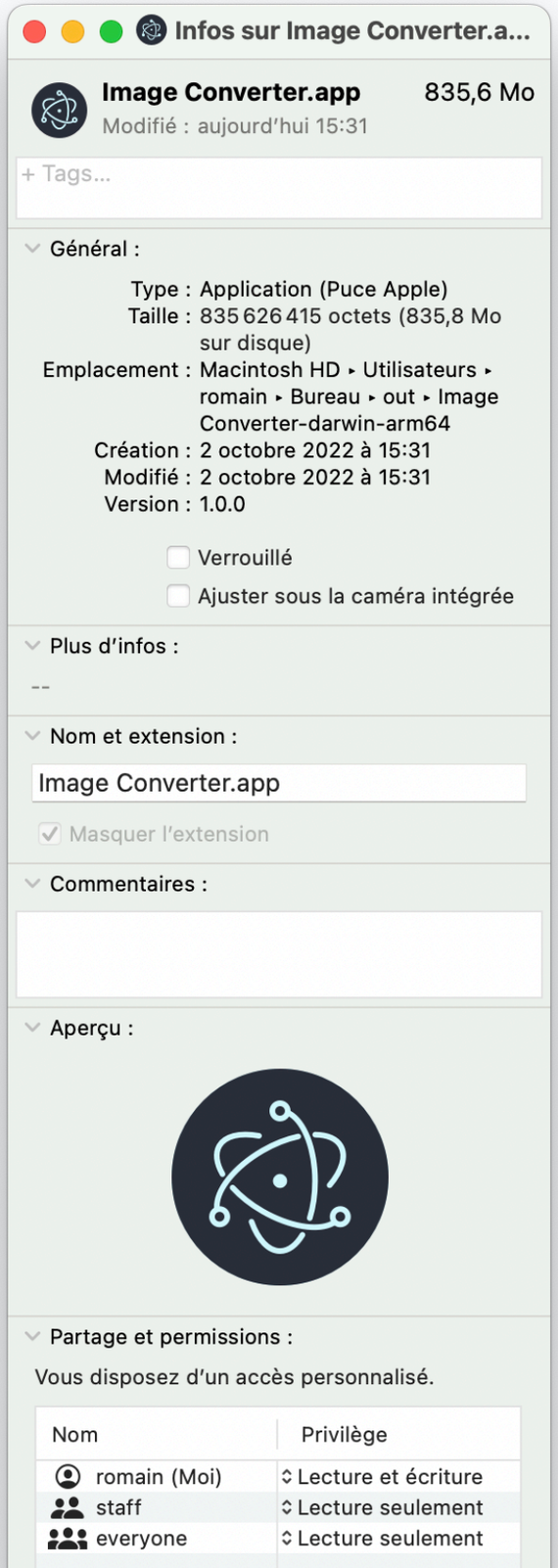
Packaging - electron-packager

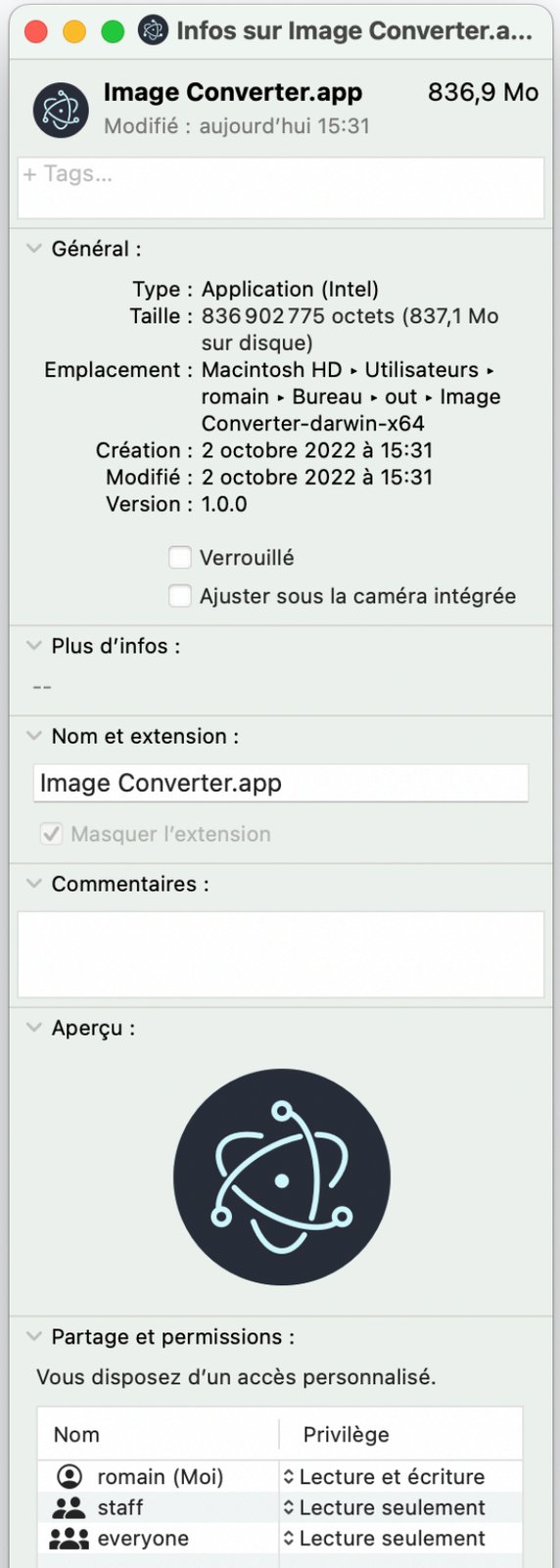
- Low level solution to package app
- Need separate libs to create a distribution of the app (.dmg, .msi...)
- Installation : npm install --save-dev electron-packager
- App Name can be set from name (lowercase) or productName key of package.json
- devDependencies will not be copied in the app bundles
- Platorm : darwin, linux, mas or win32
- Arch : ia32, x64, armv7l, arm64, mips64el or universal

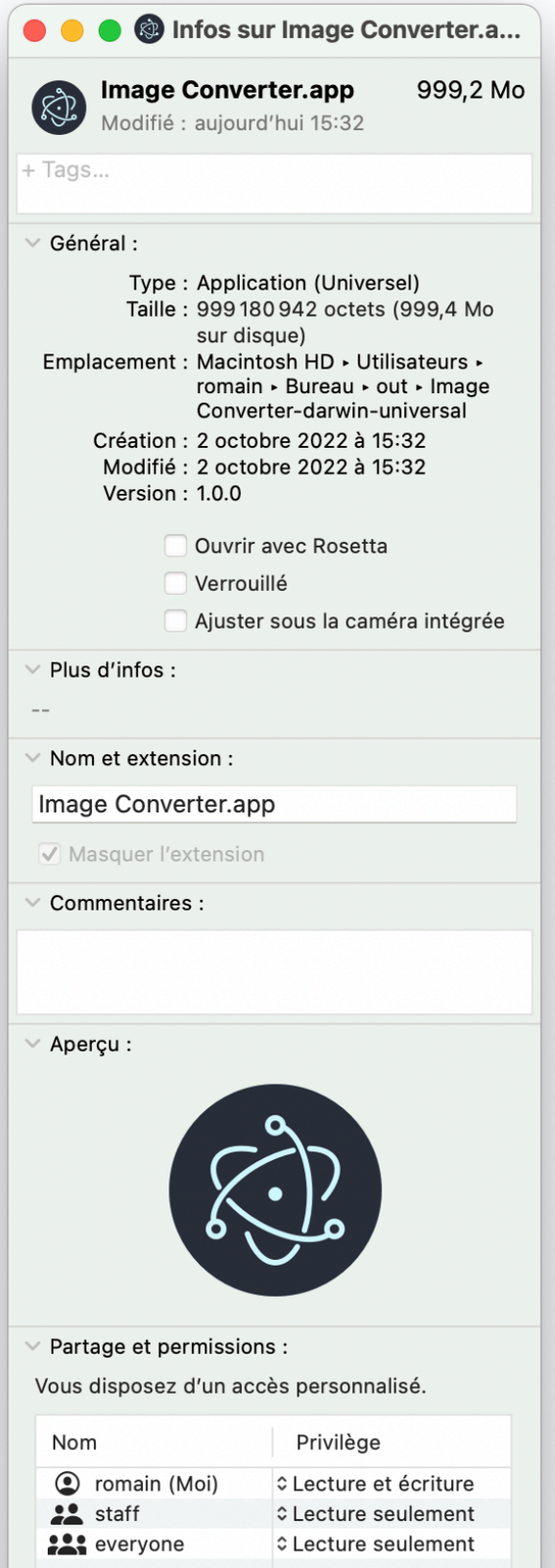
```
"scripts": {
  "start": "electron .",
  "package": "electron-packager . --asar --overwrite --out out",
  "package:mac-x64": "npm run package -- --platform darwin --arch x64",
  "package:mac-arm64": "npm run package -- --platform darwin --arch arm64",
  "package:mac-universal": "npm run package -- --platform darwin --arch universal",
  "package:windows-ia32": "npm run package -- --platform win32 --arch ia32",
  "package:windows-x64": "npm run package -- --platform win32 --arch x64",
  "package:linux-x64": "npm run package -- --platform linux --arch x64"
}
```

- Distribution : https://github.com/electron/electron-packager#distributable-creators

Packaging - electron-packager







Packaging - Native Module

- Native Node.js modules are supported by Electron but need to be recompiled specifically for Electron
- You might get this error if you don't :
  Error: The module '/path/to/native/module.node' was compiled against a different Node.js version using...
- To rebuild install and run electron-rebuild :
  https://github.com/electron/electron-rebuild

Packaging - asar

- To improve performance and solve issues with long paths on Windows we can create an asar archive
- Files can still be accessible in the asar using require or the fs module
- It is recommended that native module should be packed outside an asar file.
- To create and extract asar files : https://github.com/electron/asar

Packaging - electron-builder

- Full fledged solution for packaging, distributing, code-signing, auto-updates...
- Has a custom updater that can download update from Github Releases, Amazon S3 without an update server
- Has designed its own packager

Packaging - Electron Forge

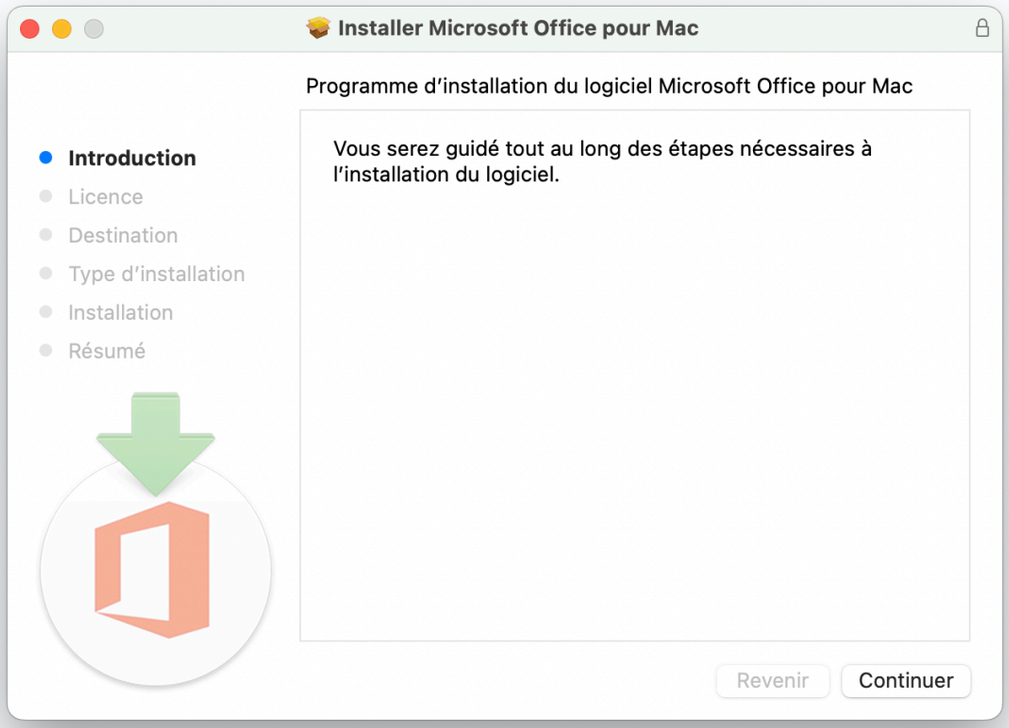
- The official documentation recommends to use Electron Forge over Electron Packager or electron-builder (since June 2022) :
  https://www.electronjs.org/docs/latest/tutorial/application-distribution
  https://github.com/electron/electron/commit/e410109a3d12e5b305f87e6e13967f2e3f88f3f5#diff-8e4106ddc0a14b2dfe4e044e758ec8d90d572481cd3c7246fafebb264671d986

- Relies on electron-packager to package the app (.app, .exe...)

- Integrates electron-rebuild to deal with native modules issues

Packaging - macOS distribution

- In electron-builder and Electron Forge a macOS app can be distributed through
  - an archive : Zip, tar.gz...
  - a DMG : a virtual image containing the app
  - a PKG : an installer that can run pre and/or post install scripts

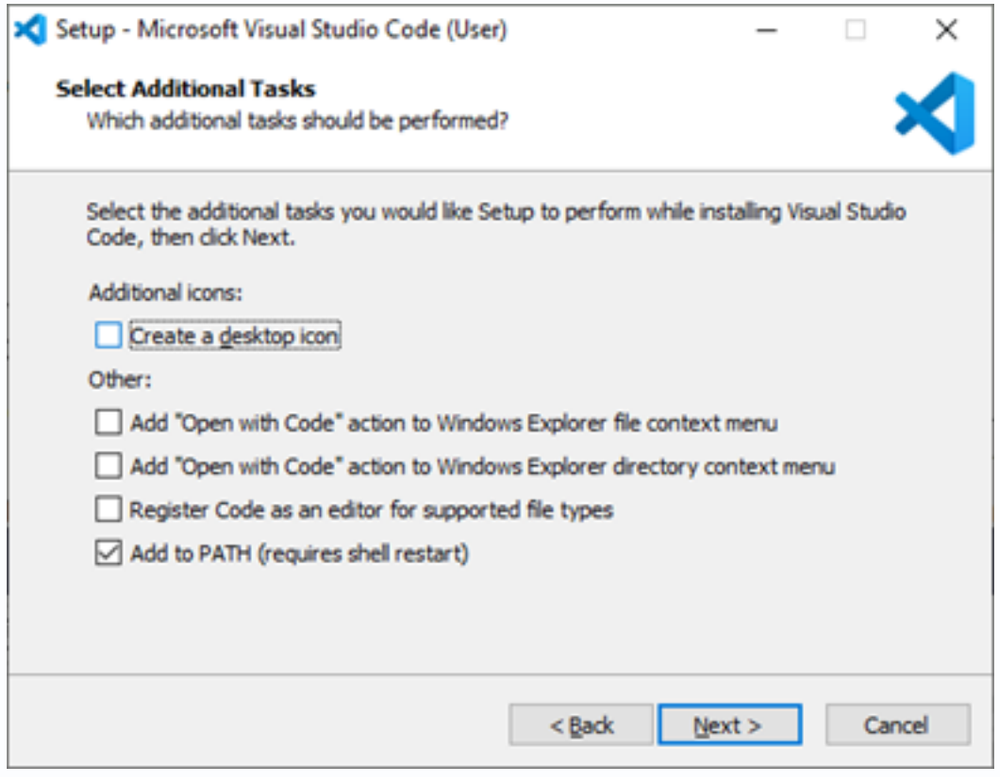


Packaging - Windows distribution

- Electron-builder supports officially 3 Windows targets :
  - NSIS (recommended)
  - AppX (Windows Store)
  - Squirrel.Windows (deprecated)
- Electron Forge supports officially 3 Windows targets :
  - WiX MSI
  - AppX (Windows Store)
  - Squirrel.Windows (recommended)
- Squirrel vs NSIS debate
  https://github.com/electron-userland/electron-forge/issues/485
  https://github.com/electron/electron/issues/17722

Packaging - Windows distribution

- VSCode uses another installer system called Inno Setup (no integration with electron-builder or Electron Forge)
  https://github.com/microsoft/vscode/blob/main/build/win32/code.iss
  https://jrsoftware.org/isinfo.php#features
  https://github.com/felicienfrancois/node-innosetup-compiler

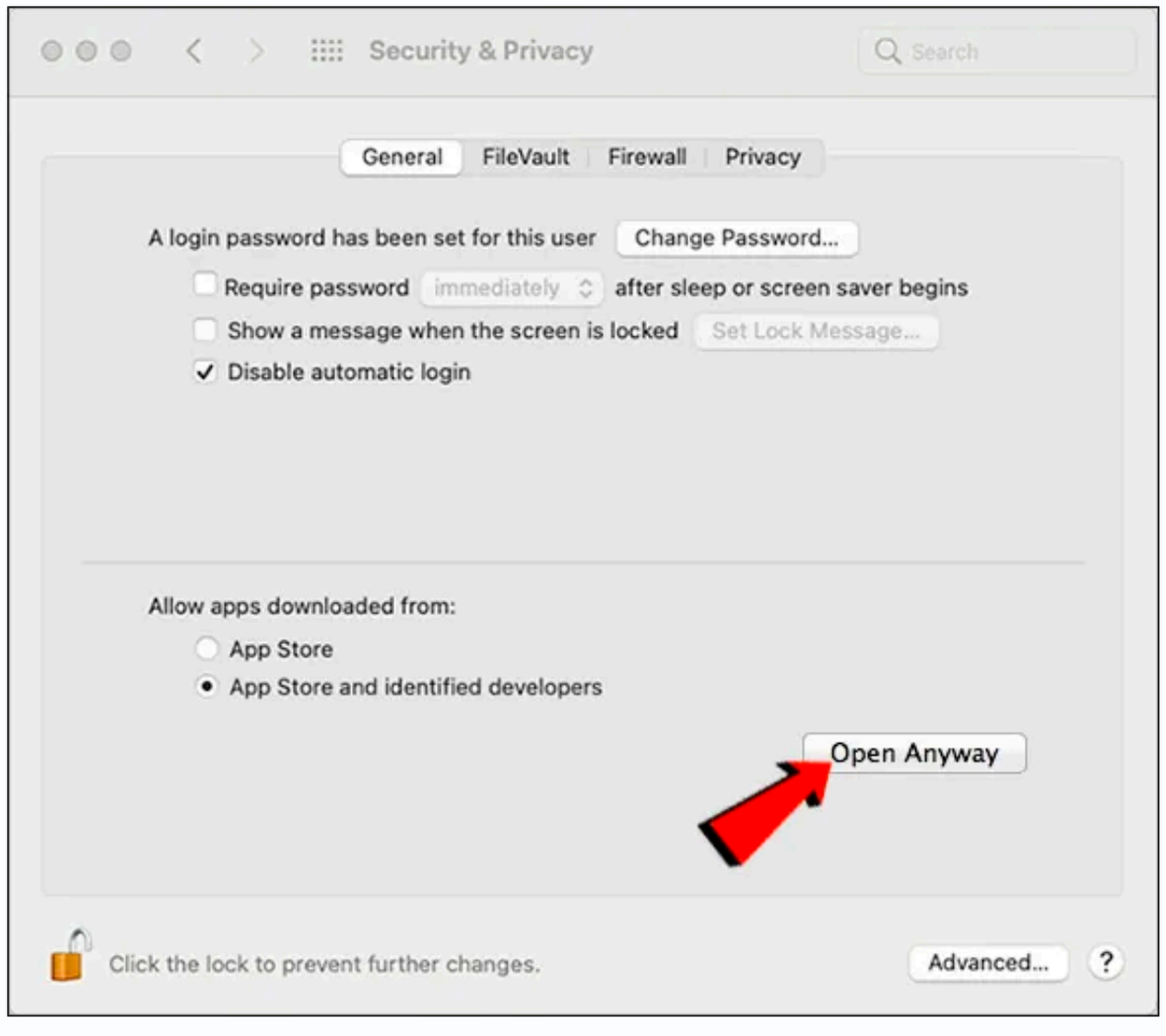


Packaging - Code signing

- Code signing is required on macOS and Windows when we distribute the app
  - because macOS will refuse to launch the app
  - because depending of the settings Windows will refuse to launch the app
  - to be able to publish on stores (Mac App Store, Windows Store)
  - to be able to auto-update the app
- On macOS code signing is only possible after registering to the Apple Developer Program (99 USD per year as individual, 299 USD as company) https://developer.apple.com/programs/
- On Windows you will have to buy a code signing certificate from a Certificate Authority :
  - https://www.digicert.com/signing/code-signing-certificates
  - https://sectigo.com/ssl-certificates-tls/code-signing

Packaging - Code signing

- On macOS a non-signed app will only launch if we do an exception in System Preferences > Security & Privacy > Open Anyway



Packaging - Code signing

- Code signing in CI
  https://felixrieseberg.com/codesigning-electron-apps-in-ci/

Debug and Testing

Debug and Testing - Debug

- Debugging the render process
  - open the Chrome DevTools using Command-Option-I on macOS or Control-Shift-I on Windows or Linux
  - calling mainWindow.webContents.openDevTools() in the main process
- Debugging the main process
  - https://www.electronjs.org/docs/latest/tutorial/debugging-vscode
  - https://www.electronforge.io/advanced/debugging

Debug and Testing - Playwright

- https://playwright.dev/docs/api/class-electron

```
// @ts-check
const { test, expect, _electron: electron } = require('@playwright/test');

test('userData path', async () => {
  // Launch Electron app.
  const electronApp = await electron.launch({ args: ['.'] });

  // Evaluation expression in the Electron context.
  const userDataPath = await electronApp.evaluate(async ({ app }) => {
    // This runs in the main Electron process, parameter here is always
    // the result of the require('electron') in the main app script.
    return app.getPath('userData');
  });
  expect(userDataPath).toBe('/Users/romain/Library/Application Support/Gallery');
});
```

Debug and Testing - Playwright

```
const { test, expect, _electron: electron } = require('@playwright/test');

test('Export button enabled on selection', async () => {
  // Launch Electron app.
  const electronApp = await electron.launch({ args: ['.'] });

  // Get the first window that the app opens, wait if necessary.
  const window = await electronApp.firstWindow();
  await window.waitForLoadState();

  expect(await window.isDisabled('text=Export Selection to WebP'));
  await window.click('img');
  expect(await window.isEnabled('text=Export Selection to WebP'));

  await electronApp.close();
});
```