Cross Platform Coding

Write once or so and run several places

Me

Kent Archie

kentarchie@gmail.com

Code and slides on GitHub

https://github.com/kentarchie/cross-platform-coding-talk.git

References at the end

Definitions

Cross platform for us means multiple OSes

Not just different versions of a given OS

Not about how applications built for Windows XP can run on Windows 10 or even using C# on Linux

Desktop here means Windows, MacOS or *nix(GNU/Linux, UNIX, Solaris etc)

Mobile means IOS or Android

History

Every machine had different instruction set, thus different assembly language

The Burroughs B5000 supported user defined instructions

https://www.smecc.org/The%20Architecture%20%20of%20the%20Burroughs%20B-5000.htm

High level languages (FORTRAN, COBOL,C etc) allowed code to be ported

Code still had to be recompiled on each new machine

UCSD Pascal compiled to a virtual instruction set, so compiled code could be moved as long as the p-code interpreter had been ported

https://en.wikipedia.org/wiki/UCSD_Pascal

History

More recently, Java was developed to be cross platform

“Write Once, Run Everywhere” was the slogan

(Also the slogan of XOJO cross platform dev environment)

Expected to run on all desktop OSes and in the web browser

Java compiled to JVM, runs on desktop, “same” code could run as applets in browser

Never quite worked as desired, applets didn’t catch on

Java still used heavily in web backends, multiple languages compile to the JVM

This Talk

In this talk we will be discussing:

Electron for desktop apps

Flutter/Dart for mobile

Both are free (mostly open source)

Development can be done on (at least) Windows and Linux (Haven’t tried MacOS)

Electron

Originally developed by Github (now owned by Microsoft)

Used by many current applications (Visual Studio Code, Slack, WhatsApp)

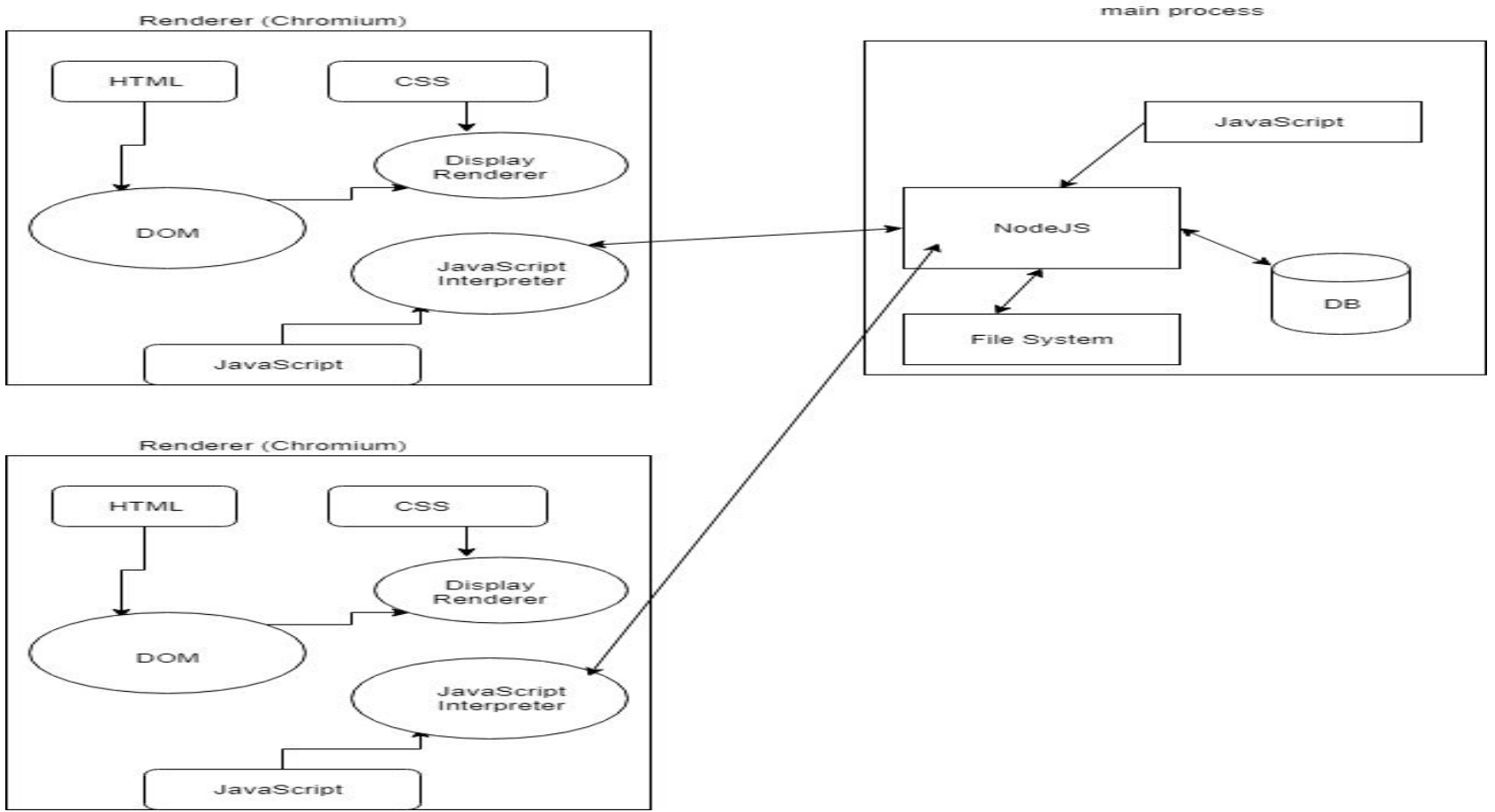
Combines Chromium (open source version of Chrome browser) and Node JS

Code written in JavaScript, but like NodeJS, you can develop C++ modules)

Docs, downloads etc here https://electronjs.org/

History https://electronjs.org/docs/tutorial/about

Electron App Structure



Electron App Notes

This is very similar to a normal NodeJS app

You create web pages using the usual techniques (CSS,HTML)

The Render process connects to the NodeJS process on the same machine

The Render process javascript is run using NodeJS so you have access to all the Node libraries

The resulting Electron app has pretty much a full Chrome browser and a NodeJS server in it

Electron Notes

When completed, you use the Electron build tools to create a separate executable for whichever of Windows, MacOS and Linux that you want.

Your web code is the same in all 3, but the Chromium and NodeJS parts are specific to the OS you are targeting

Package dependency is a concern as is the shear size of the resulting applications

Electron Application Structure

```
HowdyUniforum/
├── clientApp
│   ├── css
│   │   └── HowdyUniforum.css
│   ├── index.html
│   └── js
│       ├── HowdyUniforum.js
│       └── thisButton.js
├── mainJS
│   └── serverMain.js
├── node_modules
├── package.json
└── tabs.vim
```

package.json

Defines the project

```
{
  "name": "howdyuniforum",
  "version": "1.0.0",
  "description": "Simple Uniforum Example",
  "main": "mainJS/serverMain.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
    ,"start" : "electron mainJS/serverMain.js"
  },
  "keywords": [
    "example"
  ],
  "author": "Kent Archie",
  "license": "MIT"
}
```

Electron Application Main Process Code, part 1

```
'use strict';
global.__base = __dirname + '/';
const {app, BrowserWindow} = require('electron');

const APP_URL = 'file://' + __dirname + '/../clientApp/index.html';

let MainWindow = null;

app.on('ready', () => {
	createMainWindow();
});
```

Electron Application Main Process Code, part 2

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

app.on('activate', () => {
    if (MainWindow == null)
        createMainWindow();
});
```

Electron Application Main Process Code, part 3

```
function createMainWindow()
{
    MainWindow = new BrowserWindow({
                'width': 800
                ,'height': 600
                ,backgroundColor: "#D6D8DC"
                ,show: false
   });
       MainWindow.loadURL(APP_URL);
       MainWindow.webContents.openDevTools(); // show chrome console
       MainWindow.on('closed', () => { MainWindow = null; });
       MainWindow.once('ready-to-show', () => { MainWindow.show(); });
} // createMainWindow
```

Electron Renderer HTML

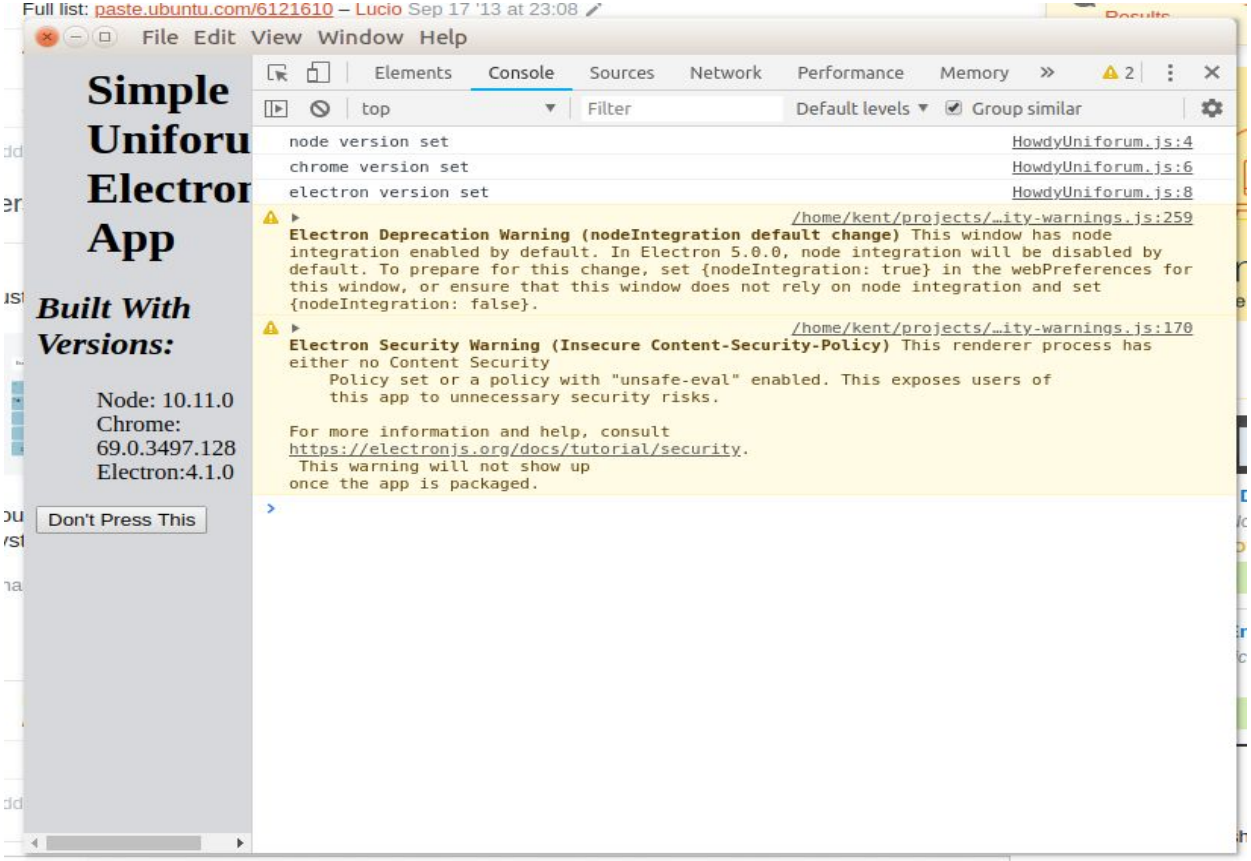
```
<html>
   <head>
      <title>Simple Uniforum Electron App</title>
     <link rel='stylesheet' type='text/css' href='css/HowdyUniforum.css' />
   </head>
<body>
  <h1>Simple Uniforum Electron App</h1>
  <h2>Built With Versions:</h2>
  <ul>
     <li>Node:    <span id='nodeVersion'></span></li>
     <li>Chrome:  <span id='chromeVersion'></span></li>
     <li>Electron:<span id='electronVersion'></span></li>
  </ul>
  <button type='button' id='thisButton'>Don't Press This</button>
</body>
<script type='text/javascript' src='js/HowdyUniforum.js'></script>
<script type='text/javascript' src='js/thisButton.js'></script>
</html>
```

Electron Renderer Code

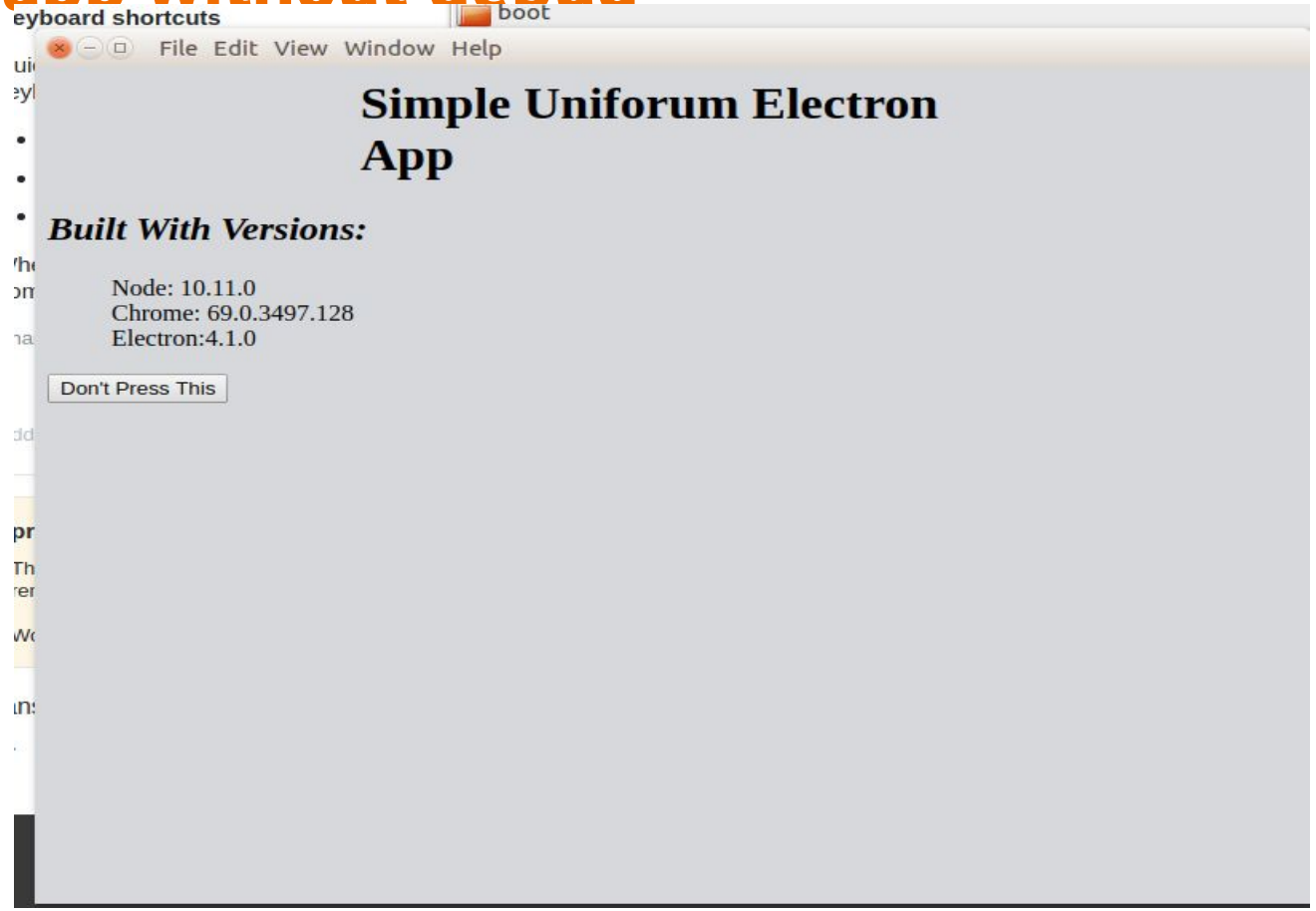
```
document.addEventListener("DOMContentLoaded", function()
{
    document.getElementById('nodeVersion').innerHTML = process.versions.node;
    document.getElementById('chromeVersion').innerHTML =
process.versions.chrome;
    document.getElementById('electronVersion').innerHTML =
process.versions.electron;

document.getElementById('thisButton').addEventListener("click",thisButton,false);
});
```

Howdy app with debug

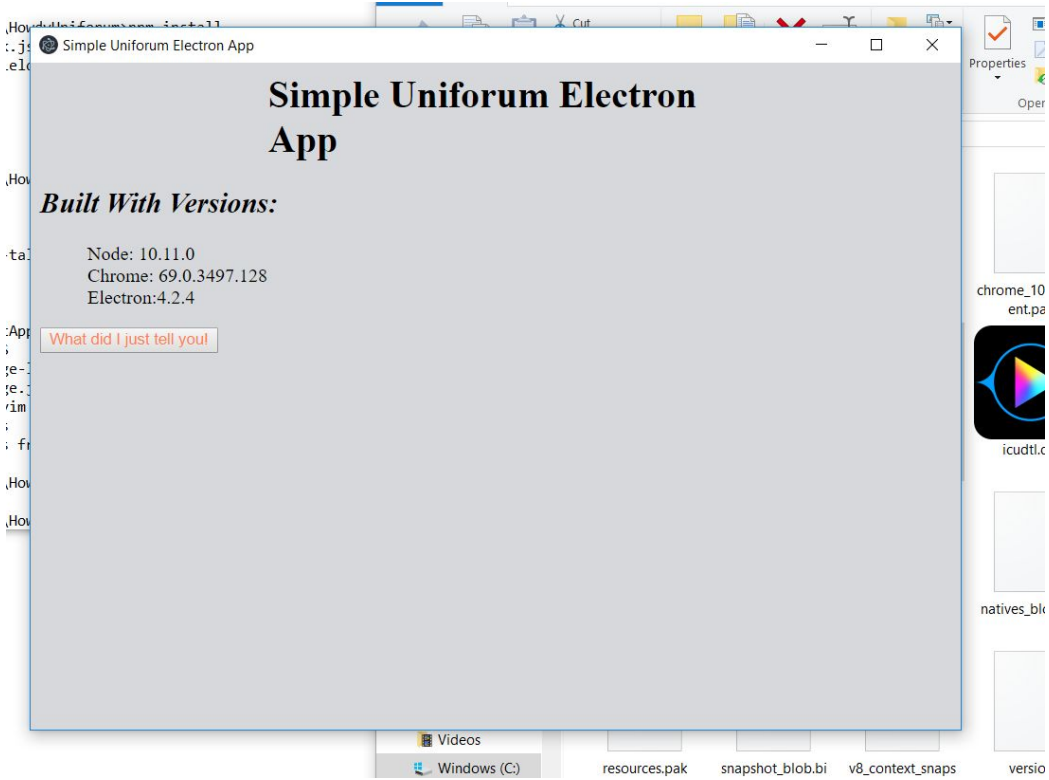


Howdy app without debug



Electron App Linux Build Test

Electron App Windows Build Test



Electron App Mac Build Test

Failed completely, I’m blaming Daryl

Many symbolic links in the build results which might not have been copied correctly from Linux to Mac

Electron Desktop App Comments

The directory for the Windows run is about 120 MB, 73 files.

The .exe file won't run outside the directory

Start seemed kind of slow as well

Convenient for the developer, maybe not so much for the customer

The default menu bar shown on the dev version is missing on the production one

But it did work on two OSes using the same source without building on two machines

Electron Desktop App Comments

Probably need to build production under the OS you are building for

If you use some libraries (like sqlite) you have to build it natively

The build tools can produce native OS installers so that might have worked better on the Mac version

Bigger Electron Application

Show some of the photo manager code

Mobile Cross Platform

There are several mobile Cross Platform system

https://www.outsystems.com/blog/free-cross-platform-mobile-app-development-tools-compared.html

https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/

Of the popular ones, Microsoft owns two, Adobe, Facebook and Google each own one

Some are encapsulated web pages like Electron, others are built using C# and Mono.

Mobile Cross Platform

I chose Flutter/Dart because it is freely available and a very recent addition to the set of systems

Google also supports another build system using a new language called Motion in addition to Java. This is not Cross platform

Flutter/Dart

A completely different approach

Flutter is the UI framework and the app is built using the Dart language

Developed by Google

Best built with Android Studio, but works well with other IDEs

No HTML, CSS or Javascript

One code base to build iOS and Android apps that look native

Architecture

Everything is a widget and a widget is

- a structural element (like a button or menu)
- a stylistic element (like a font or color scheme)
- an aspect of layout (like padding)

The UI is built of nested widgets, reminds me of Java UI

Each widget implements a build method used to specify its appearance and action

Every time a build method is called it returns a new set of widgets, the foundation system modifies the display to rreflect the changes so your code doesn’t have to

Widget State

Widgets that contain data, like user input or results of an action are children of *StateFulWidget.* If not, they are *StatelessWidget*

When a *StateFulWidget*s value changes, it calls setState() which calls the build method and a new widget is created

The widgets state is kept separate from the widget so the system can recreate the widget without losing the state

Basic Default Flutter App

When a new project is created (I am using Android Studio) it has some simple functions

The resulting project contains 78 files in 41 directories and uses 180 kB

After the first build, there are 285 files in 236 directories using 261MB

References

I found this guys videos to be handy for understanding some Flutter development techniques

https://www.youtube.com/watch?v=Un7eG5hHNPg