Contents

1 Electron

How to run and package Threepenny apps with Electron and Electron Packager.

For reference, a minimal working example is available here.

1.1 Justification

Normally when running a Threepenny app we execute our Haskell, with `stack exec` or otherwise, which starts a local server and we open our browser on a certain port to view our app. However this has a few subtle drawbacks.

One drawback is that most browsers are designed with remote servers in mind and have security features which don't make much sense for local server architectures like Threepenny's. Take file selection for example. When a user selects a file in a browser, the browser only exposes the file name and contents to the server, not allowing the server to receive the full file path. However for local server architectures there isn't much point in sending the entire file contents to the server since the server is on the same file system and could read the file directly, if only it had the full file path. Electron displays our app in a Chromium instance with many of these security features removed.

Another drawback is that the user has to run the app from the command line. Using electron-packager we can package native apps for Linux, macOS and Windows.

1.2 Running with Electron

To run a Threepenny app with Electron we need an Electron main process. We provide this one: electron.js. It runs the following on startup: - selects a free port to run on - executes our Threepenny app binary, passing the port to run on as an argument - waits for Threepenny's server to start accepting connections

- opens an Electron window which loads the URL of our Threepenny app

To get started with the linked `electron.js` first add this package.json to your project root directory. You'll need Node installed and `npm` on your PATH, confirm by running `which npm`. Now run `npm install` from the project root directory to install the necessary dependencies.

The linked `electron.js` executes the Threepenny app binary, passing the port to run on as an argument. This of course means your Threepenny app needs to take the port as an argument. We suggest also setting stdout to be line buffered, at least while developing. Altogether it should look something like this:

```
module Main where

import System.Environment (getArgs)
import System.IO
import YourApp            (start)

main :: IO ()
main = do
    hSetBuffering stdout LineBuffering
    [port, otherArg1, otherArg2] <- getArgs
    start (read port)
```

Now copy the linked `electron.js` to your project root directory. You'll have to edit the defined constants: `relBin`, which is the relative path from `electron.js` to your Threepenny application binary; and `binArgs`, which contains any additional arguments to pass to the binary. If you're not sure about the relative path to your application binary, and you're using Stack, see the next section.

Now run your app with Electron: `./node_modules/.bin/electron electron.js`

1.2.1 Explicit binary location

`relBin` is the relative path from `electron.js` to your Threepenny application binary. This might change depending on which tool or platform you are building with and thus can be a pain to set manually. If you are using Stack you can easily build your application binary to an explicit location, possibly a `build` directory:

```
stack install --local-bin-path build
```

Now you can simply set `relBin` to `./build/your-app-exe`.

1.3 Packaging with electron-packager

This section assumes the app is already setup to run with Electron based on the above instructions.

First install electron-packager: `npm install electron-packager`

Optionally edit the "name" field in `package.json` to set the name of the packaged app. Then to package the app for the current platform, simply:

```
./node_modules/.bin/electron-packager .
```

This is the most basic way to package the app, it will copy the current directory to the packaged app. However you'll likely want to avoid copying some source files, which can be achieved with the `--ignore` flag. You might end up using:

```
./node_modules/.bin/electron-packager . --ignore=app --ignore=src
```

If you are using Stack and building your application binary to an explicit location, as explained above then you might want to also ignore `.stack-work/`. An icon can be set by passing the icon path to `--icon`, note that the icon format depends on the platform. For more options use the `--help` flag.