Deep Links

Overview

This guide will take you through the process of setting your Electron app as the default handler for a specific protocol.

By the end of this tutorial, we will have set our app to intercept and handle any clicked URLs that start with a specific protocol. In this guide, the protocol we will use will be "`electron-fiddle://`".

Examples

Main Process (main.js)

First, we will import the required modules from `electron`. These modules help control our application lifecycle and create a native browser window.

```
const { app, BrowserWindow, shell } = require('electron')
const path = require('path')
```

Next, we will proceed to register our application to handle all "`electron-fiddle://`" protocols.

```
if (process.defaultApp) {
  if (process.argv.length >= 2) {
    app.setAsDefaultProtocolClient('electron-fiddle', process.execPath,
[path.resolve(process.argv[1])])
  }
} else {
  app.setAsDefaultProtocolClient('electron-fiddle')
}
```

We will now define the function in charge of creating our browser window and load our application's `index.html` file.

```
const createWindow = () => {
  // Create the browser window.
  mainWindow = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  mainWindow.loadFile('index.html')
}
```

In this next step, we will create our `BrowserWindow` and tell our application how to handle an event in which an external protocol is clicked.

This code will be different in Windows compared to MacOS and Linux. This is due to Windows requiring additional code in order to open the contents of the protocol link within the same Electron instance. Read more about this here.

Windows code:

```
const gotTheLock = app.requestSingleInstanceLock()

if (!gotTheLock) {
  app.quit()
} else {
  app.on('second-instance', (event, commandLine, workingDirectory) => {
    // Someone tried to run a second instance, we should focus our window.
    if (mainWindow) {
      if (mainWindow.isMinimized()) mainWindow.restore()
      mainWindow.focus()
    }
  })

  // Create mainWindow, load the rest of the app, etc...
  app.whenReady().then(() => {
    createWindow()
  })

  // Handle the protocol. In this case, we choose to show an Error Box.
  app.on('open-url', (event, url) => {
    dialog.showErrorBox('Welcome Back', `You arrived from: ${url}`)
  })
}
```

MacOS and Linux code:

```
// 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()
})

// Handle the protocol. In this case, we choose to show an Error Box.
app.on('open-url', (event, url) => {
  dialog.showErrorBox('Welcome Back', `You arrived from: ${url}`)
})
```

Finally, we will add some additional code to handle when someone closes our application.

```
// 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()
})
```

Important notes

Packaging

On macOS and Linux, this feature will only work when your app is packaged. It will not work when you're launching it in development from the command-line. When you package your app you'll need to make sure the macOS `Info.plist` and the Linux `.desktop` files for the app are updated to include the new protocol handler. Some of the Electron tools for bundling and distributing apps handle this for you.

Electron Forge

If you're using Electron Forge, adjust `packagerConfig` for macOS support, and the configuration for the appropriate Linux makers for Linux support, in your Forge configuration *(please note the following example only shows the bare minimum needed to add the configuration changes)*:

```
{
  "config": {
    "forge": {
      "packagerConfig": {
        "protocols": [
          {
            "name": "Electron Fiddle",
            "schemes": ["electron-fiddle"]
          }
        ]
      },
      "makers": [
        {
          "name": "@electron-forge/maker-deb",
          "config": {
            "mimeType": ["x-scheme-handler/electron-fiddle"]
          }
        }
      ]
    }
  }
}
```

Electron Packager

For macOS support:

If you're using Electron Packager's API, adding support for protocol handlers is similar to how Electron Forge is handled, except `protocols` is part of the Packager options passed to the `packager` function.

```
const packager = require('electron-packager')

packager({
  // ...other options...
  protocols: [
    {
      name: 'Electron Fiddle',
      schemes: ['electron-fiddle']
    }
  ]

}).then(paths => console.log(`SUCCESS: Created ${paths.join(', ')}`))
  .catch(err => console.error(`ERROR: ${err.message}`))
```

If you're using Electron Packager's CLI, use the `--protocol` and `--protocol-name` flags. For example:

```
npx electron-packager . --protocol=electron-fiddle --protocol-name="Electron Fiddle"
```

Conclusion

After you start your Electron app, you can enter in a URL in your browser that contains the custom protocol, for example `"electron-fiddle://open"` and observe that the application will respond and show an error dialog box.