Pier Developers guide

Many questions about Electron and React can be answered with a good online search. The Pier application however has some of its own specifics which are known only to a few. To remedy that, here (in no particular order) are some topics we hope you will find useful.

Contents

[Adding a new application 2](#_Toc26349541)

[Folder layout 2](#_Toc26349542)

[Assets 2](#_Toc26349543)

[Components 2](#_Toc26349544)

[Style 2](#_Toc26349545)

[Util 2](#_Toc26349546)

[Parent component 2](#_Toc26349547)

[Manifest 3](#_Toc26349548)

[How the apps logo is linked through css 4](#_Toc26349549)

[Utility classes 5](#_Toc26349550)

[Notify.ts 5](#_Toc26349551)

[How to send a balloon notification 5](#_Toc26349552)

[BrowserView util 6](#_Toc26349553)

[Persistent storage 6](#_Toc26349554)

[Logging 6](#_Toc26349555)

[Disposing of ipcRenderer listeners 6](#_Toc26349556)

[Npm update, don’t do it. 6](#_Toc26349557)

[Nvm, what it is and why its useful 6](#_Toc26349558)

[Debugging 6](#_Toc26349559)

[Initialising devtron 6](#_Toc26349560)

[Extensions to install in VS code 6](#_Toc26349561)

[React devtools for chrome 6](#_Toc26349562)

[NPM scripts and what they do 6](#_Toc26349563)

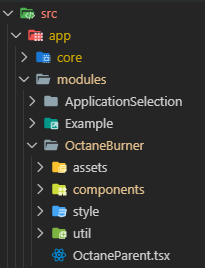
[Component Library 6](#_Toc26349564)

[Styling 6](#_Toc26349565)

# Adding a new application

When adding a new application there are a few places you need to make changes so that your application is reflected in Pier. There is a preferred folder layout for the applications and a preferred way to structure the start of the application.

## Folder layout

To keep things consistent and hopefully easy to follow, new applications should be laid out as such:

Within the src/modules folder create a folder for your application. Within that place assets, components, style and util folders. In the root of your folder should be placed the entry component for your application

### Assets

The assets folder is for holding any images you may need in your application. At minimum it should have the logo for your application.

### Components

The components folder is for the unique react components which make up your application. Your components should aim to be as simple as possible with the logic extracted where possible into util files so the react just deals with the display. There are some core components intended for reusability which can be read about in the Component Library section.

### Style

The style folder is for any scss your application needs.

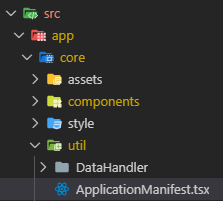
### Util

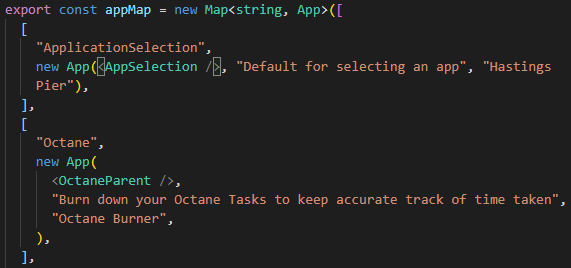
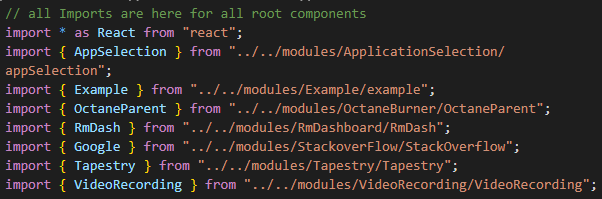
The util folder is for any files which are not react components. For example a class which holds the logic for processing a JSON response before sending it to the component for display.

### Parent component

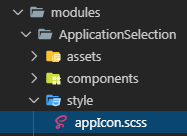
The parent or root component for your application should live in the root of your applications folder. This is so that relative paths are easy to read within the application and in the application manifest.

## Manifest

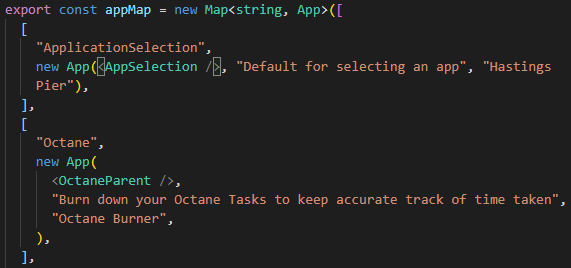
The details about available applications are kept in “appMap” in the applicationManifest.

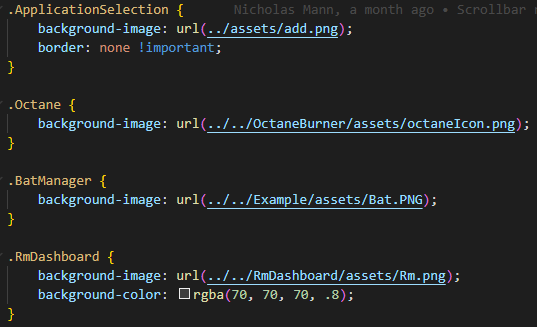
The appMap is an exported map containing the root element for the application, a description and its displayed title. The key in the map for the application is used in several places for tagging your application internally and it must not contain a space. The app map can be referenced in your application through an import if you need it. Your root component is imported at the top of the application manifest.

## How the apps logo is linked through css

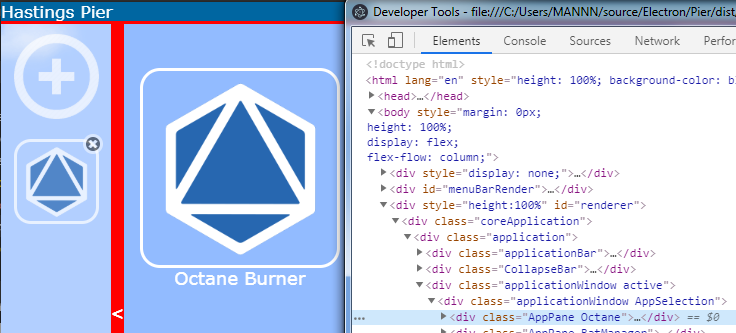
The app logos are linked to their key in the ApplicationManifest. The key is appended as part of the className of the ApplicationPane and ApplicationButton. The appIcon.scss in the ApplicationSelection module is where you put the css which describes your logo.

Example:

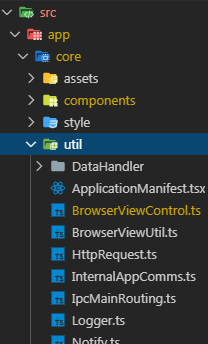
The “Octane” app is defined in the manifest as bellow

The key is “Octane”. The logo is defined in appIcon.scss as bellow. If your logo has transparencies it is worth considering setting a background colour as well.

When the ApplicationButton and ApplicationPane are created the key from the manifest is added to the className.

This has the effect that any element with the class matching the manifest key will have its background image set to the logo so bear that in mind.

# Utility classes

One of the considerations with an electron app is how to link the web based UI to the electron functionality. This is largely done with the use of the [ipcMain](https://electronjs.org/docs/api/ipc-main) and [ipcRenderer](https://electronjs.org/docs/api/ipc-renderer) functionality. To simplify this, several utility classes have be created so that you as a developer don’t have to worry too much about how to communicate with the main process. These classes can be found in the util folder of the core folder.

## Notify.ts

This class is used to let your application send some notifications which can be used to display notifications on the AppBar, balloon notifications, change the active application, change the window title and close an application.

### How to send a balloon notification

To send a balloon notification use the “Balloon” function. It requires the title of the notification and the contents. The “Source” parameter is used to tell the balloon where the notification came from so that if you were to click on the balloon that application becomes the active application. The WindowId parameter is there for future functionality. Once your application is set as the active it is up to you how you deal with the notification. In your application you can listen for notifications using ipcRenderer.on(“notify”+ *YourAppKeyFromManifest*, …...)

# BrowserView util

# Persistent storage

# Logging

# Disposing of ipcRenderer listeners

# Npm update, don’t do it.

# Nvm, what it is and why its useful

# Debugging

## Initialising devtron

## Extensions to install in VS code

## React devtools for chrome

# NPM scripts and what they do

# Component Library

# Styling