## Table of Contents

## Overview

Monocle is a proposed rewrite of Lens, the Glass implementation for systems that do not have an underlying window manager. Unlike Lens, in which most of the work is done in C, Monocle will be almost all Java code. Where required, interactions with C code will be through simple Java-C bindings where the logic is in Java and only the low-level system or library call is in C. The goals of Monocle are:

- Functional and quality parity with Lens on all platforms: Embedded Linux/ARM, Android and headless
- Simplify the process of porting to a new platform. Ideally this could be done in some cases without rebuilding JavaFX.
- Improve maintainabiity and debuggability of the embedded ports. Where possible, simplify the code.
- Pluggability of Linux input device handlers. Ideally it would be possible to add support for a new input device without rebuilding JavaFX.

Like Lens, Monocle assumes it has full control of the screen and does not have to cooperate with other graphical applications. It does not rely on an underlying window system.

## Generic Components

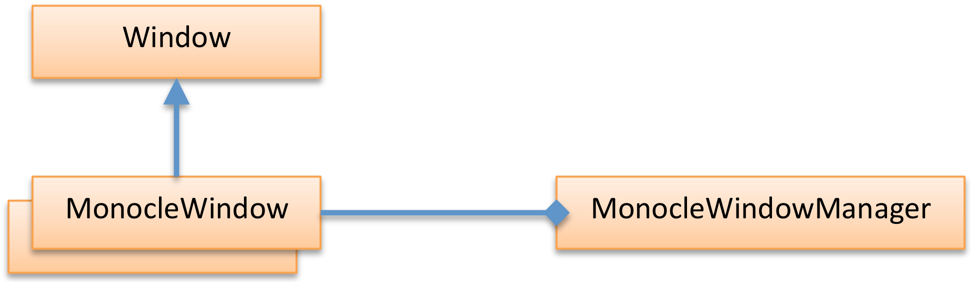
### Glass Integration

The classes named Monocle* integrate with class. So MonocleApplication, MonocleWindow and MonocleView extends the Glass classes Application, Window and View.

All code running at this level is on the application thread.

### Window Management

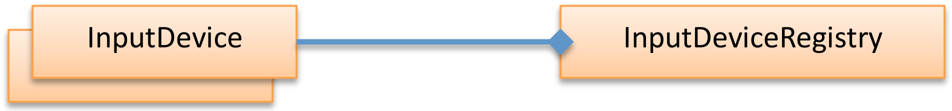
Monocle windows do not exist at a native level. Window state is held in MonocleWindow, while MonocleWindowManager maintains the Z-ordering of the window stack and assigns ID numbers to MonocleWindows.



All window management code runs on the application thread.

### Input Device Capabilities

An InputDevice represents a single device that can generate input events. An InputDevice can report on its input capabilities. For example, it can declare itself as a multitouch screen or as a 5-way keypad. InputDevices are registered with an InputDeviceRegistry; MonocleApplication listens on changes to the InputDeviceRegistry to get notification on what classes of devices are attached.



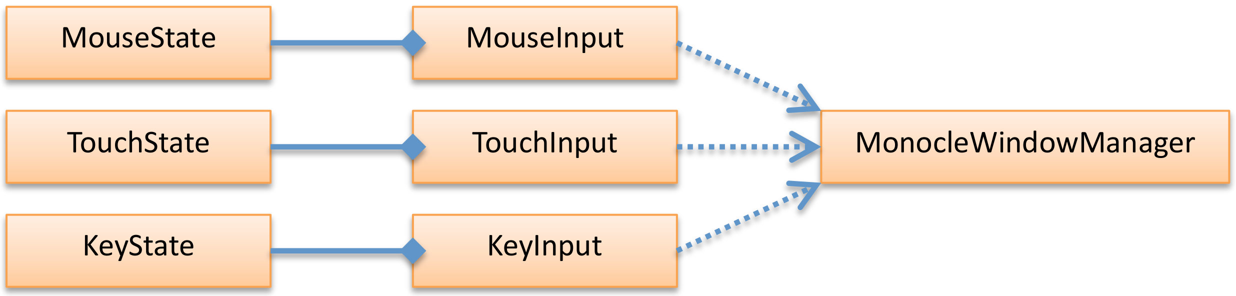
Implementations of InputDeviceRegistry are also responsible for making sure input devices are recognized and their events delivered.

All generic input device code runs on the application thread. Platform-specific InputDeviceRegistry implementations can contain code that runs on other threads.

### Input

Three input handler classes process input of different kinds: MouseInput, TouchInput and KeyInput. Each of these maintains its own state in an input state class: MouseState, TouchState or KeyState. When the one of the input handler is notified of a change to the input state it generates events accordingly based on the current state of the window stack and input focus. Low-level input classes do not communicate directly with the window stack.

Each input handler contains a single input state object. When the input handler is notified of an input state change, it receives a state object containing the new state. The input handler does not store this external state object, but copies its contents into its own records. This helps us to minimize object creation during event processing.



All generic input code runs on the application thread.

## Platform

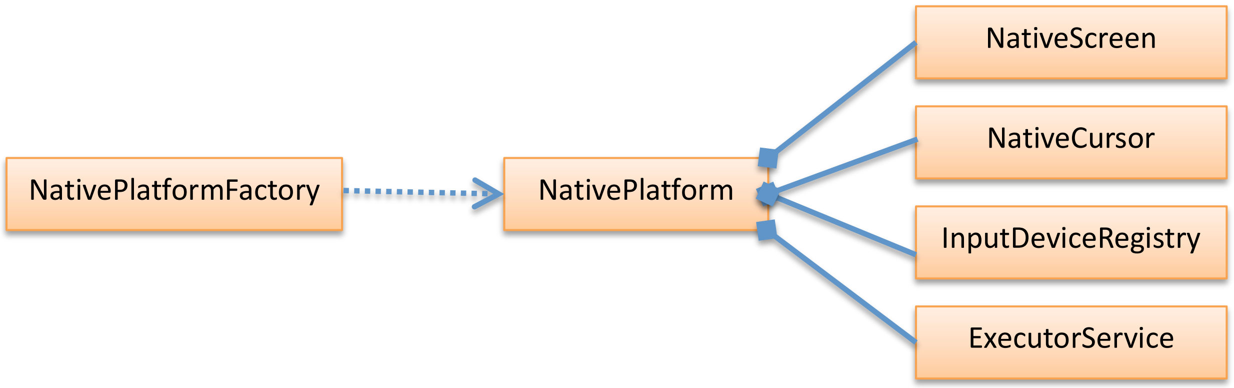
The platform-specific components are: NativePlatform, NativeScreen, NativeCursor and InputDeviceRegistry.

NativePlatform is instantiated by NativePlatformFactory. NativePlatformFactory looks at the system property `monocle.platform` to get an ordered list of factory classes to attempt to use. NativePlatformFactory then instantiates these factory classes, querying each in turn whether it can support the current platform it is running on. When a matching NativePlatformFactory is found, its corresponding NativeFactory will be created.

NativePlatform provides a single-threaded java.util.concurrent.ExecutorService. This ExecutorService is the application thread.

NativeScreen is instantiated by the NativePlatform. NativeScreen reports on the physical characteristics of the screen. It is possible that this class will be used by Prism as well; in this case the class will have to be thread-safe.

NativeCursor is instantiated by the NativePlatform. NativeCursor is responsible for updating the visible cursor state, using a platform-specific hardware cursor where possible. NullCursor is an empty implementation of this that does not display a cursor.



NativePlatform attempts to determine the best platform to use. Use of a specific platform can be forced by setting the system property `monocle.platform` to one or more of the following values in a comma-separated list:

`Linux` - the generic Linux port, using low-level input device nodes for input and no hardware cursor

`OMAP` - for the BeagleBoard xM using EGL/framebuffer with Linux device input and a hardware cursor

`OMAPX11` - for the BeagleBoard xM using EGL/X11 with Linux device input and a hardware cursor

`X11` - for generic Linux/X11 platforms, using X11 for input and cursor.

Other platforms that are not implemented yet but should fit into this system include: `Android`, `Headless`, `Dispman` and `MX6`.

## Generic Linux Port

### Native Interfaces

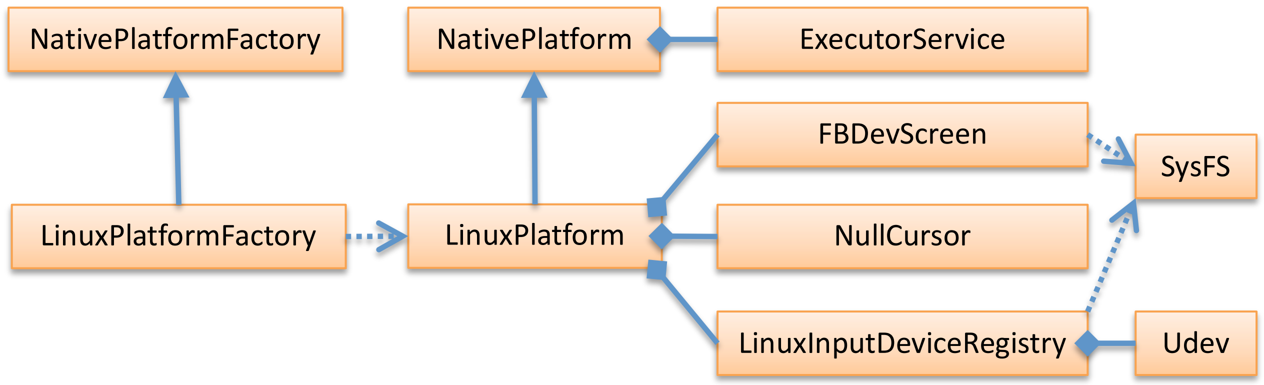
Most of the interaction between the Linux port of Monocle and the OS level is using the pseudo-filesystem under /sys. This is accessed using utility methods in the class SysFS. This is sufficient to read screen and input device characteristics and to request notification on what input devices are attached.

The Udev class is an interface to the Linux udev monitor to get notification when devices are attached and removed from the system. This requires some C code, since Java does not have an API for connecting to Unix domain sockets.

it might be necessary to add another native interface to use ioctl calls to read absolute axis range data for touch screens. This information does not seem to be available in sysfs.

### Implementations

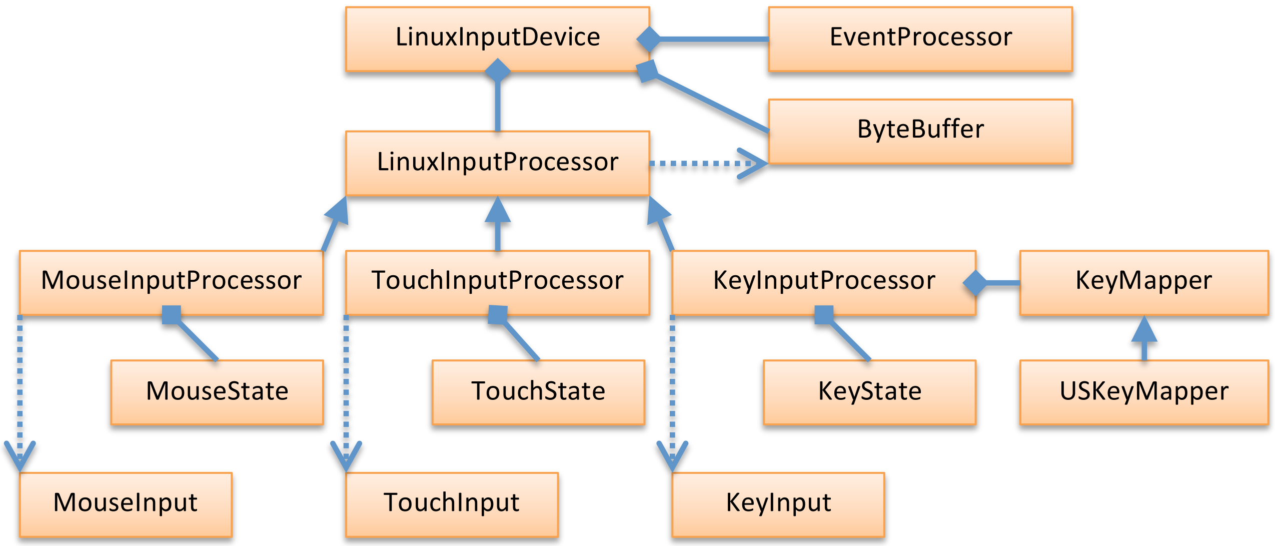
- FBDevScreen reads screen data from the framebuffer device /dev/fb0.
- No cursor implementation is provided for the generic port, since there is no usable standard for hardware cursors on Linux.
- LinuxInputDeviceRegistry receives hot plug notifications of input device addition and removal and sets up input event processors that work with the generic Monocle input handlers.



### Input

When an input device is detected by Udev, a LinuxInputDevice instance is created for it. The LinuxInputDevice reads events from the Linux input node into a ByteBuffer. As soon as any data is ready in the ByteBuffer to be processed, a Runnable (actually a singleton EventProcessor) is submitted to the application thread to process this data. "Data ready" means that a complete event has been received, including the terminating EV_SYN SYN_REPORT. If the EventProcessor is already pending execution then it is not resubmitted; the expectation is that every time the EventProcessor runs it will process all pending events.

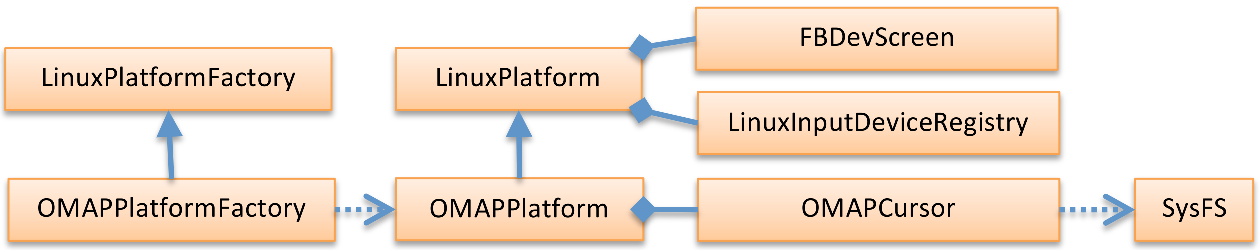
Each LinuxInputDevice has an implementation of LinuxInputProcessor attached to it. This LinuxInputProcessor is called by the EventProcessor on the application thread. The LinuxInputProcessor iterates over pending input events and notifies input handlers of input state changes.



Note that raw linux input events are not objects. Where possible we avoid object creating during input processing. This will be particularly important in touch and mouse processing where events can be received very quickly.

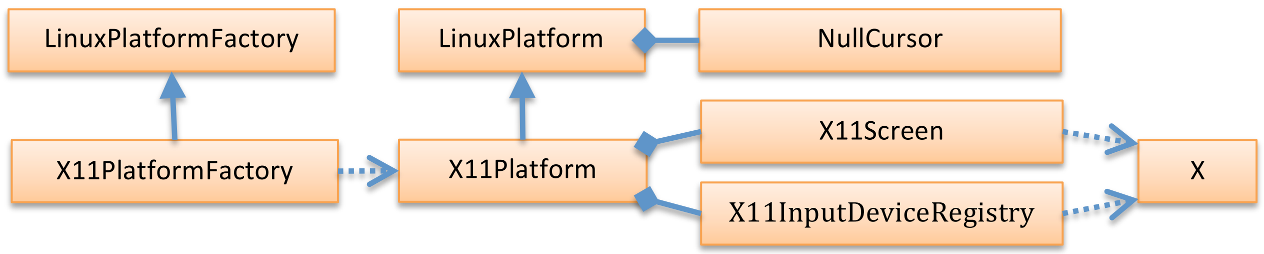
## OMAP Port

The OMAP port is a subclass of the generic Linux port that also provides a hardware cursor on OMAP platforms.



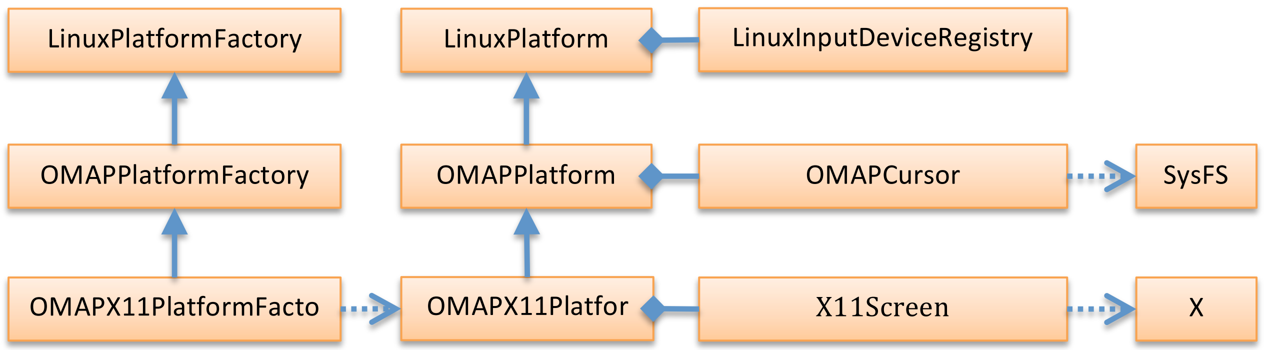
## X11 Port

The X11 port is a subclass of the generic Linux port that uses EGL/X11 for rendering instead of EGL/Framebuffer. It also takes its input from X11 events instead of directly from Linux input devices.



## OMAPX11 Port

The OMAPX11 port is a subclass of the OMAP port that uses EGL/X11 for rendering instead of EGL/Framebuffer. Like the OMAP port, it uses Linux input devices for input events and a hardware OMAP cursor.



## Status

What's working:

- Rendering and input on X11
- Rendering on Freescale i.MX6
- Mouse input
- Cursor on OMAP3 (partially)
- Most of HelloSanity on BeagleBoard xM

What's not working:

- Transparency in OMAP3 cursor
- Multiple windows

What's not done yet:

- Touch, Key input
- Freescale i.MX6 Cursor

- Dispman port
- Android port
- Nested event loops
- Window grabbing
- Integration into 8u-dev
- Robot
- Unification with Prism's platform recognition