Ozone-Wayland Architecture

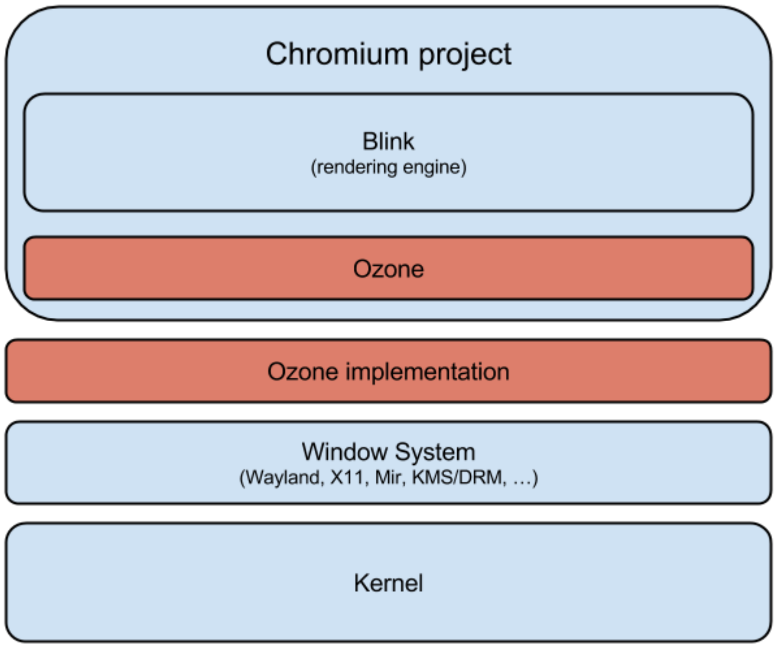
revision 1 (Jun 2014)

Overview

Ozone-Wayland is the implementation of Chromium Ozone for supporting Wayland graphics system. Different projects based on Chromium/Blink like Chrome Browser, Chrome OS, Crosswalk, among others can be enabled now on Wayland using Ozone. This document outlines its architecture behind.

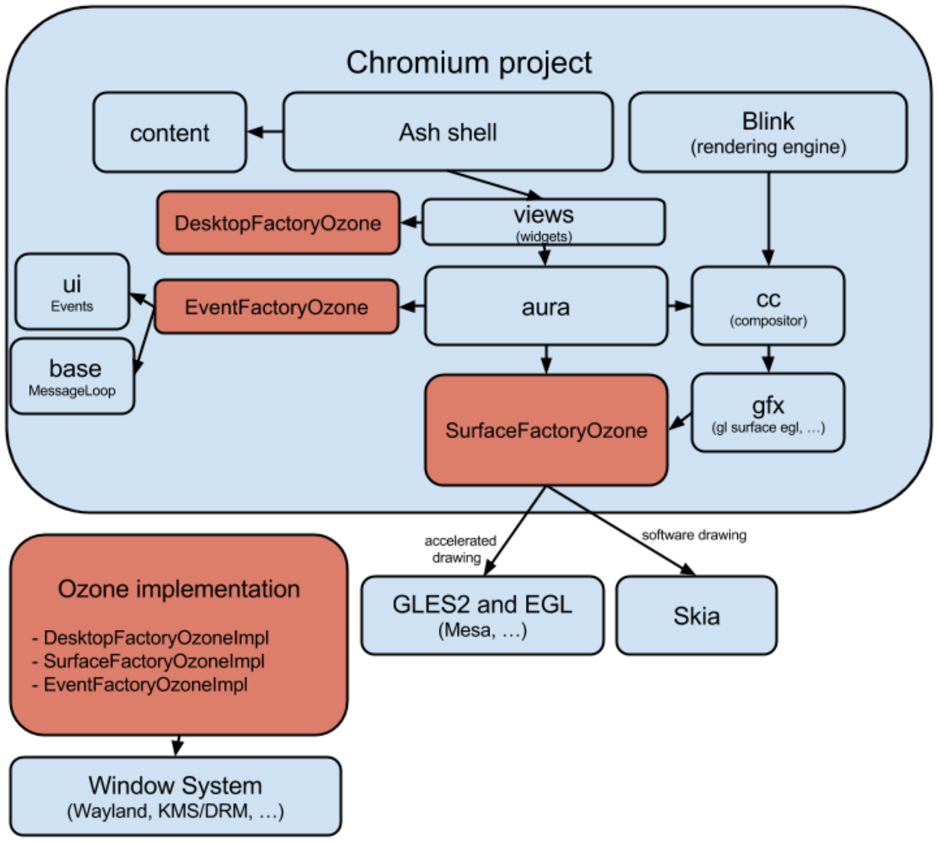
Chromium Ozone

Ozone is a set of classes in Chromium for abstracting different window systems on Linux. It provides abstraction for the construction of accelerated surfaces underlying Aura UI framework, input devices assignment and event handling.



Before, when using Aura on Linux all the native windowing system code (X11) was spread throughout Chromium tree. Now the idea is that Ozone will abstract the native code and because it's a set of class factories, it will switch for whatever is the window system.

One of the biggest advantages of Chromium Ozone is that it allows to implement the window system details externally from the Chromium tree, which is great because it is where the loaded work situates in many cases. Therefore, Ozone-Wayland can be seen as a friend project of Chromium where one is empowering the other with different characteristics. Besides Ozone-Wayland, there are different Ozone implementations like KMS/DRM, caca, etc. The following diagram shows the detailed architecture of Ozone:



Graphics, windowing and rendering

In short, graphics operations in Ozone-Wayland are not much different than Chromium on Aura/Linux but with a different windowing system, which is Wayland. As of Jun 2014, Ozone-Wayland only supports accelerated drawing through EGL (software drawing is not in our agenda, although that could be implemented). Ozone-Wayland loads the OpenGL ES 2.0 library implementation (“libGLESv2.so.2”) for Chromium internally use it for drawing. Both in “desktop aura" (Chrome Browser, content_shell, etc) and in "fullscreen aura" (ChromeOS) targets, most of the time only one single surface is used for the entire interface. The surfaces creation are handled to the Ozone backend implementation, which in case of Ozone-Wayland sends through Wayland protocol to the OpenGL library takes care of actually creating those surfaces buffers. Individual handlers for each surface are handed back then to Ozone-Wayland which can now take care of life events of the surfaces, for example reacting accordingly on resizing, positioning or layering.

Most of the complex operations using GPU like offscreen rendering, WebGL and Canvas 2D works out-of-the-box with current Ozone-Wayland implementation. Initial support for Hardware accelerated Video decoding through VAAPI ("libva") is in place and we will continue to mature it in coming days.

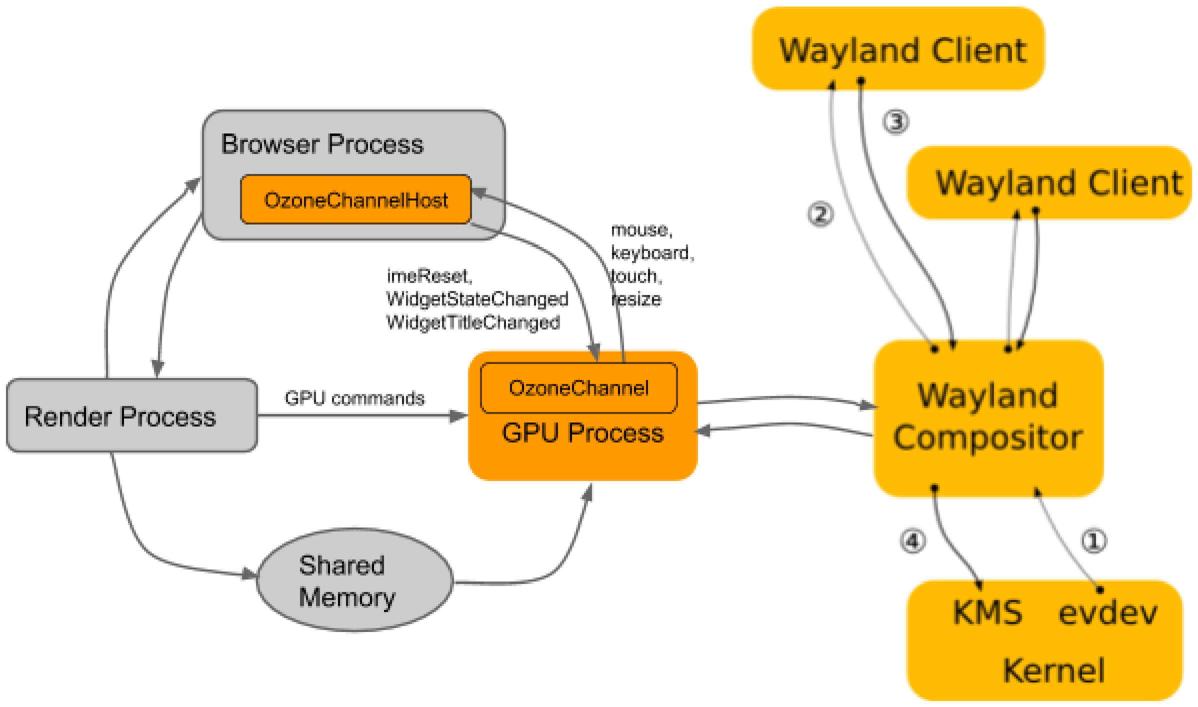
Input handling

Input events like mouse, touch, keyboard etc are generated in Wayland compositor and send through Wayland protocol to Ozone-Wayland. Ozone-Wayland in turn convert these Wayland native input events to Chromium types (`ui::Event`) and dispatches to Aura.

In short, Ozone-Wayland implements the following:

- **Cursors**: Wayland doesn't automatically change the pointer image when a pointer enters a surface, so Ozone-Wayland sets the cursor it wants in response to motion events or pointer focus.
- **Keyboard Mapping:** happening via X Keyboard extension (XKB)
- **Virtual Keyboard** (VKB) support
- **Input Methods**

Ozone-Wayland and Multi-process architecture



This diagram above shows how Chromium works under Wayland in terms of Ozone-Wayland.

Chromium has three important type of processes: the browser process, the render process and the GPU process: The browser process is the one that runs the UI, manages tabs, and deals with HTTP requests/responses. Besides, in Chromium there are multiple render processes that parse HTML documents, run JavaScript, and render web pages. More details about Chromium multi-process architecture can be found here. Chromium also has a single GPU process where GPU accelerated operations are issued to the graphics driver, which in Ozone-Wayland case, is a *single* Wayland client that communicates with Wayland compositor. Note that in Ozone-Wayland, all the window system operations are multiplexed to a single Wayland client instance in the Chromium side before wire to Wayland compositor - other approaches would prefer to use multiple clients for each browser process or say, to implement the "mini-Wayland compositor" approach. Ozone-Wayland is not doing that for convenience. However, our current approach does add unnecessary overhead in case of inputs. We receive the input callback from Wayland in GPU process and send it over IPC to Browser process, ideally we want to have the callbacks handled directly in Browser process. Not sure how to solve this with the current Wayland protocol.

Ozone Channel and Host:

Gpu process has a OzoneChannel object that manages communication with the Browser process. Browser process maintains a corresponding OzoneChannelHost. We use the existing Chromium IPC system to send and receive messages (Surface and Input related) between Browser and GPU process. The OzoneChannelHost deals with any input or surface events sent from GPU process i.e. keyboard, mouse, touch, window resize. In addition, OzoneChannel deals with messages sent from the browser process related to widget and IME state.

Contributors

The following persons contributed directly to this document:

Jonne Hur <joone.hur@intel.com>

Kalyan Kondapally <kalyan.kondapally@intel.com>

Tiago Vignatti <tiago.vignatti@intel.com>