QtWayland

Creating Window Compositors with the QtWayland module

Andy Nichols
2012

digia

A little about me...

- Andy Nichols
- Software Engineer at Digia
- 6 years of Qt experience
- Former Qt Support Engineer
- Maintainer of QtWayland

Outline

- The past (QWS)
- What is wayland?
- The QtWayland Module
- How do you use QtWayland?

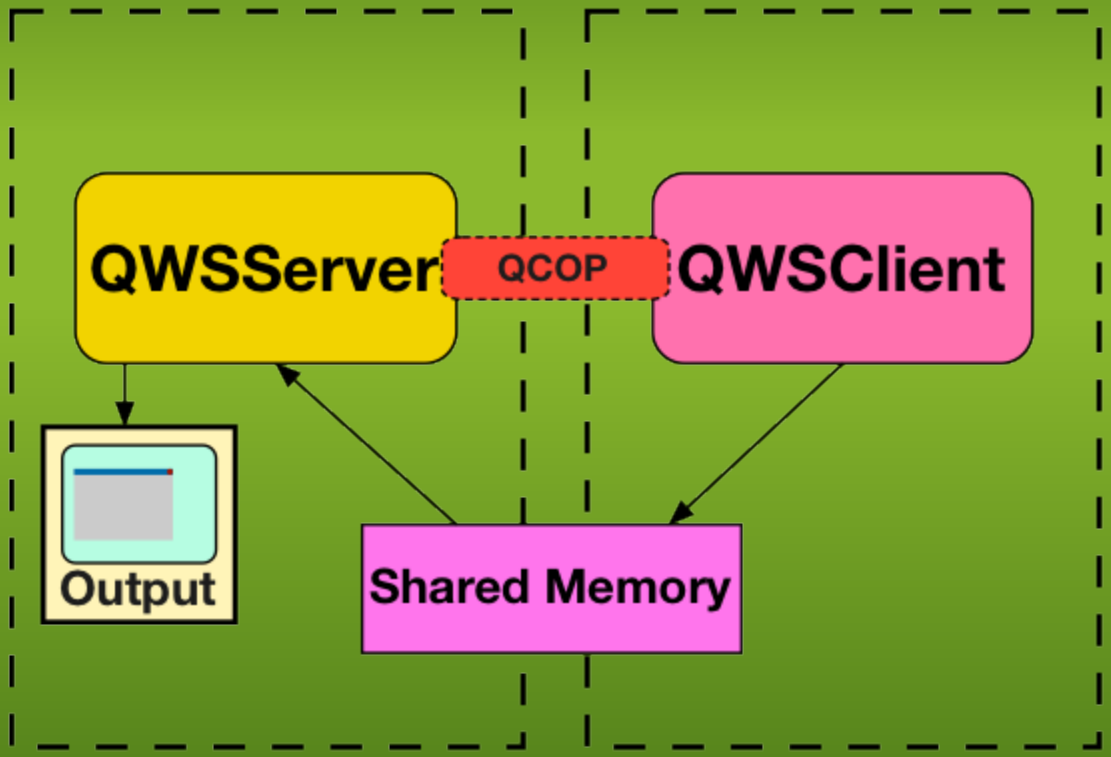
In case you have not heard...

digia

QWS has been removed in Qt 5.0

digia

QWS Graphics Stack



Direct Rendering with QWS

- Clients map regions of the framebuffer
- Regions are written to directly by client

Accelerated Graphics in QWS

- Custom QScreen plugin
- Custom paint engine
- Custom paint device

If that is the case, then why replace QWS?

digia

It works great for its original use case

digia

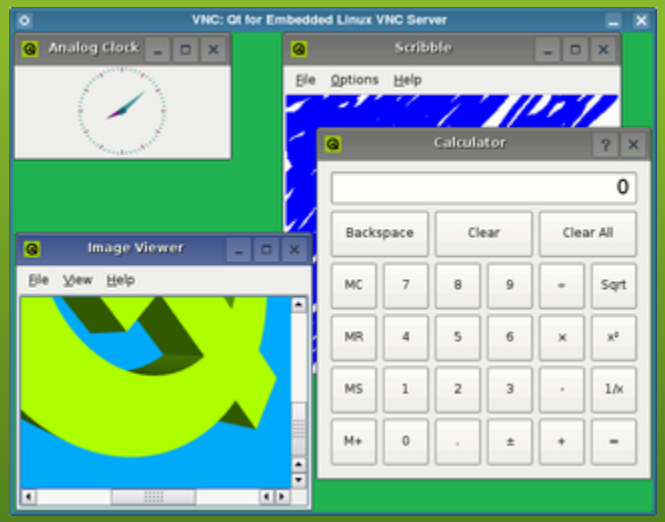
But we are not living in that world anymore

Have you ever actually had to support OpenGL in QWS?

digia

QWS is inflexible

- Supporting new hardware
- Customization of look and feel



Lack of OpenGL Support

- OpenGL QScreen Plugins do exist
- Limited to particular hardware
- Require specific API's
- Limited to a single process

QWS: Does it still make sense?

- Overlap with QPA
- Few QWS developers
- Protocol design is hard

And then came a project called Wayland

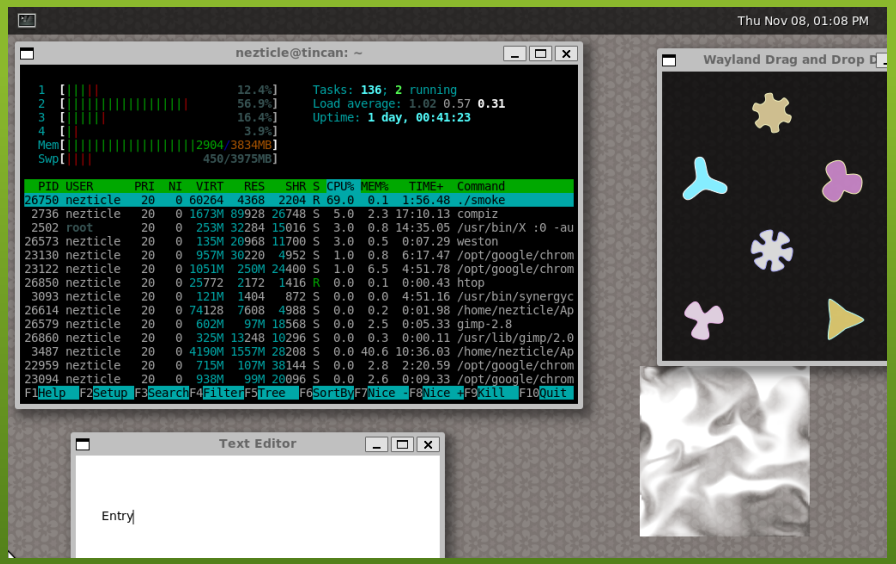
What is Wayland?

"Wayland is a protocol for a compositor to talk to its clients as well as a C library implementation of that protocol."

-wayland.freedesktop.org

Wayland Compositors can be

- a standalone display server
- an X11 application
- a wayland client

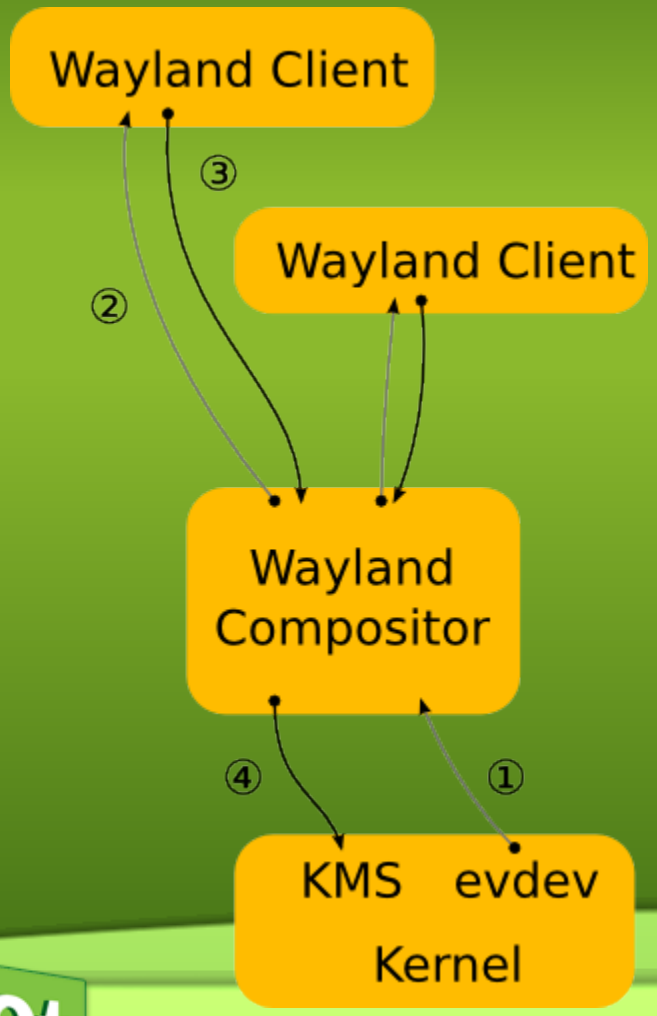


Weston Compositor

Wayland Clients can be

- Traditional applications
- X servers
  - rootless
  - fullscreen
- other display servers

How does Wayland work?



1. Input events
2. Direct events to client
   a. Location in scenegraph
   b. Inverse transform
3. Respond to UI events
   a. Render changes
   b. notify compositor of changed regions
4. Post composition to output

src: wayland.freedesktop.org

The Wayland Compositor

- Composes output
- Handles Input devices
- Forwards input events to clients
- Coordinates client buffers

The Wayland Clients

- Renders to a surface buffer
  - Shared Memory buffers
  - native buffers (GPU memory)
- Notifies compositor of changes

Wayland Buffer Sharing

- Shared Memory buffers
  - Raster based toolkits
- GPU buffers
  - shared between processes with EGL
  - mapped as a texture
  - no additional upload costs
- Wayland-EGL

Why choose Wayland?

- Lightweight
- Fast and Smooth
- External Open Source Project
- Easy to make customized Compositor
- Protocol is extendable

but the best reason is...

digia

Cross toolkit compatibility

digia

Try doing that with QWS!

digia

The QtWayland Module

- Wayland platform plugin
- QtCompositor API

Running Qt Applications in a Wayland Compositor

Just add "-platform wayland"
To run as a wayland client

```
nezticle@tincan: ~
nezticle@tincan:~$ ./qtapplication -platform wayland
```

digia

Demo Time

digia

QtCompositor API

digia

WaylandCompositor

- Subclass to create your own compositor
- reimplement
  - *surfaceCreated(WaylandSurface *surface)*
- Call *frameFinished()* after all surfaces are rendered.

WaylandSurface

- Emits signals when client's state has changed
  - mapped
  - unmapped
  - damaged
- Contains data needed to render surface

```
void MyCompositor::surfaceCreated(WaylandSurface
*surface)
{
   connect(surface, SIGNAL(destroyed(QObject *)),  this,
SLOT(surfaceDestroyed(QObject *)));
   connect(surface, SIGNAL(mapped()), this, SLOT
(surfaceMapped()));
   connect(surface, SIGNAL(unmapped()), this, SLOT
(surfaceUnmapped()));
   connect(surface, SIGNAL(damaged(const QRect &)),
this, SLOT(surfaceDamaged(const QRect &)));
   connect(surface, SIGNAL(extendedSurfaceReady()),
this, SLOT(sendExpose()));
}
```

WaylandSurface data

```
if (surface->type() == WaylandSurface::Shm) {
 image = surface->image();
} else if (surface->type() == WaylandSurface::Texture) {
 texture = surface->texture(QOpenGLContext::currentContext());
}
```

WaylandInputDevice

- Get instance from:

  WaylandCompositor::defaultInputDevice()
- Forward events to Wayland clients.
- Focus management

QWindow Compositor Demo

digia

WaylandSurfaceItem

- QtQuick 2.0 Item for WaylandSurface

QML Compositor Demo

digia

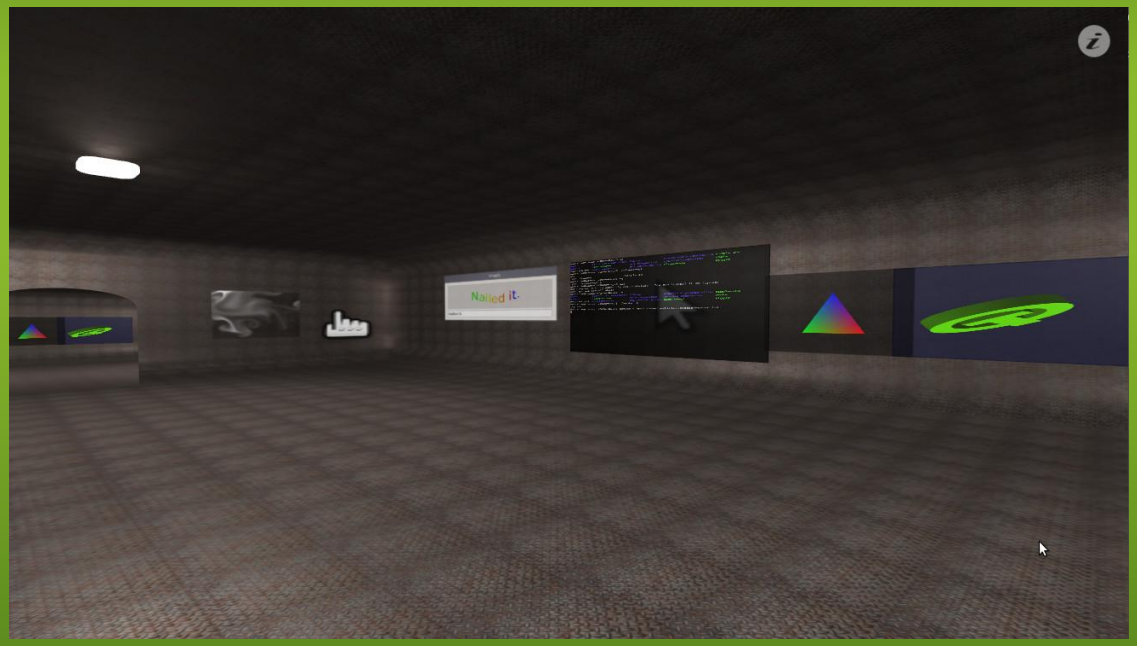
QtWayland on new hardware

- Wayland-EGL
- Hardware Integrations
  - Client
    - QWaylandGLIntegration
    - native window surface
    - native OpenGL Context
  - Server
    - GraphicsHardwareIntegration
    - native buffer
    - how to map native buffer to texture

Hardware without OpenGL

- No native GPU buffers are used
- No QtQuick 2.0
- Mesa + LLVM software rendered OpenGL possible

Compositor Creativity Demo



digia

The Future of QtWayland

- Full wayland 1.0.0 spec support
- Proper client side decorations
- qt-wayland-scanner
- Fullscreen direct rendering

Questions?

digia

Thanks for coming!

email: andy.nichols@digia.com
freenode irc: nezticle

digia