Welcome to Wayland

Martin Gräßlin
mgraesslin@kde.org
BlueSystems

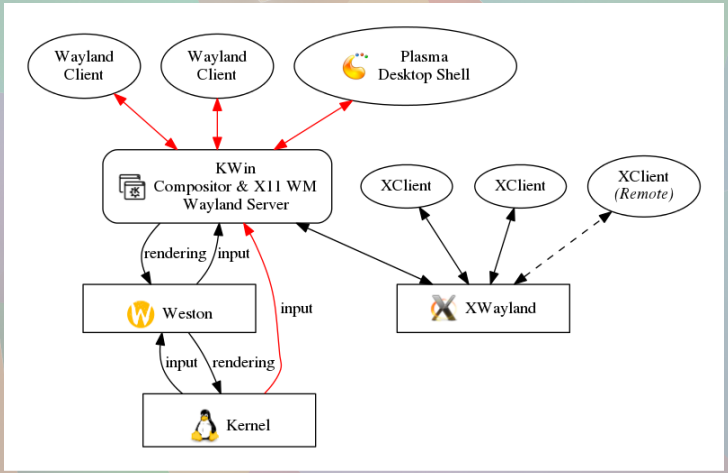
Akademy 2015

26.07.2015

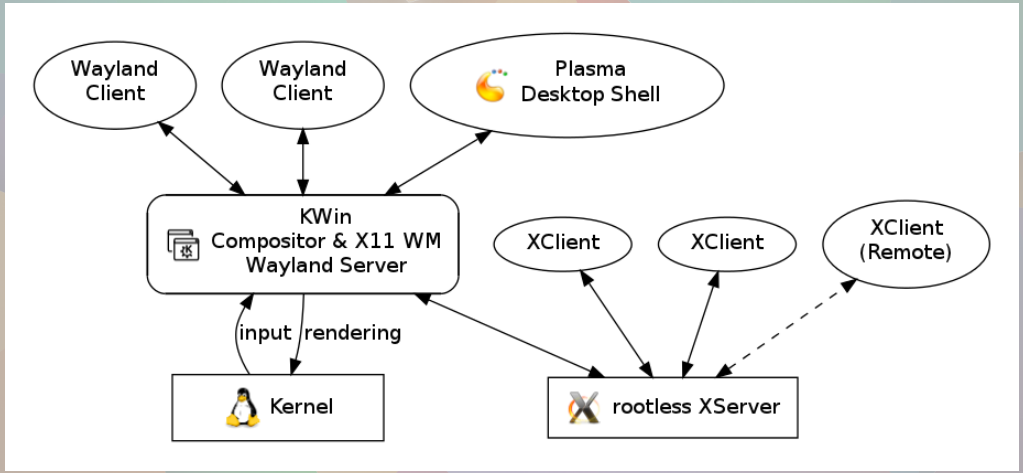
Agenda

Agenda

Architecture as presented last year



Current architecture



Backend plugins for different platforms

KWin internal Platform Abstraction

- Create OpenGL context and surface
- QPainter fallback for no OpenGL
- Output information
- Input event handling

Libinput

If backend doesn't provide input, libinput library is used.

Available backend plugins

Windowed/Nested Platforms

- X11 (supports OpenGL, QPainter)
- Wayland (supports OpenGL, QPainter)

Full Platforms

- DRM (supports OpenGL through GBM, QPainter)
- fbdev (supports QPainter)
- Android hwcomposer/libhybris (supports OpenGL, input)

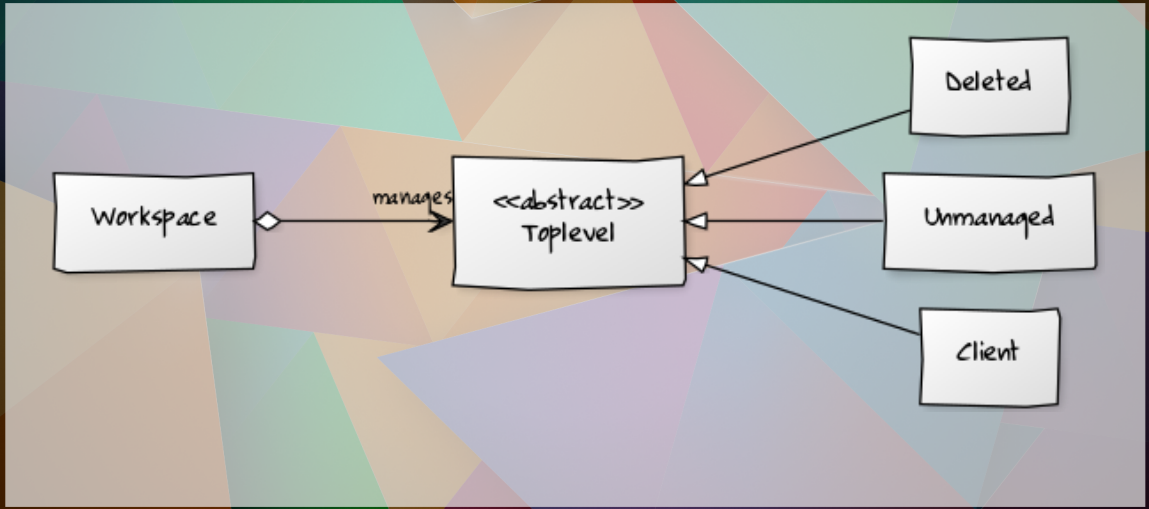
This Presentation runs on the DRM platform!

Architecture summary

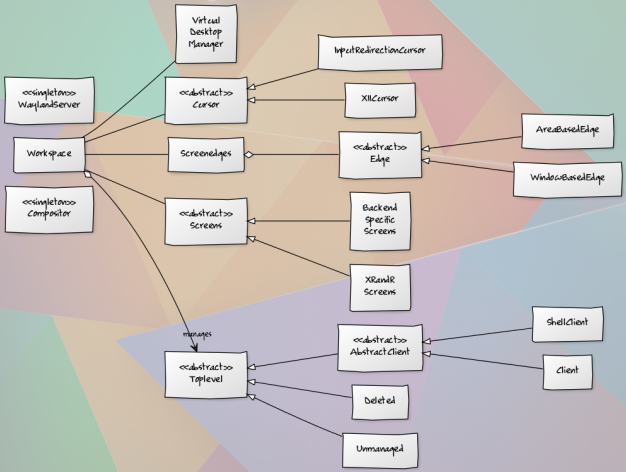
- KWin is a Wayland server
- KWin supports wl_shell clients both OpenGL and SHM
- KWin supports Xwayland based clients
- KWin can render on top of DRM/KMS
- KWin support input through libinput
- Nested KWin Wayland servers on X11/Wayland for easy testing

Agenda

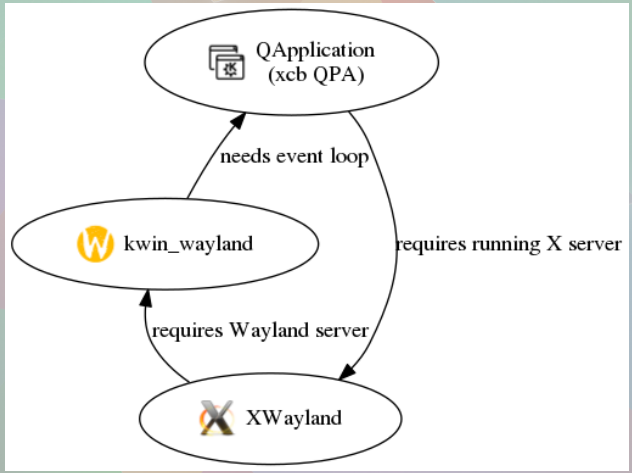
Simplified KWin (Core) architecture before Wayland



Simplified KWin (Core) architecture as of today



How to start KWin



Starting KWin as X11 application

How to start X?

- KWin's startup is highly X11 dependent
- Qt's XCB plugin requires X11 in QApplication ctor
- Starting Xwayland before KWin requires a running Wayland server
- Wayland server requires event loop
- Event loop requires QApplication
- Xwayland requires wl_drm

KWin needs to move away from xcb QPA

Is QtWayland any better?

New Startup

- First start Wayland server
- Create QApplication
- Startup Compositor/Scene
- Startup Xwayland
- Wait for Xwayland being started
- Continue with X specific startup code

Similar problems as xcb QPA

Roundtrips are evil

- Requires Wayland server at QApplication startup
- Does roundtrip to server in startup (blocks gui thread)
- Cannot create a QThread before creating QApplication
- QtWayland dispatches events while waiting for the roundtrip

Still many workarounds needed

Blocking OpenGL context creation

```
// HACK: create a QWindow in a thread to force QtWayland to create the
// client buffer integration.
// this performs an eglInitialize which would block as it does a roundtrip
// to the Wayland server in the main thread.
// By moving into a thread we get the initialize without hitting the problem
// This needs to be done before creating the Workspace as from inside
// Workspace the dangerous code gets hit in the main thread
QFutureWatcher<void> *eglInitWatcher = new QFutureWatcher<void>(this);
eglInitWatcher->setFuture(QtConcurrent::run([] {
    QWindow w;
    w.setSurfaceType(QSurface::RasterGLSurface);
    w.create();
}));
```

More workarounds

BypassWindowManagerHint needed for each Window on X11

```
bool ApplicationWayland::notify(QObject *o, QEvent *e)
{
  if (QWindow *w = qobject_cast< QWindow* >(o)) {
    if (e->type() == QEvent::Show) {
      // on QtWayland windows with X11BypassWindowManagerHint are not shown,
      // thus we need to remove it. As the flag is interpreted only before
      // the PlatformWindow is created we need to destroy the window first
      if (w->flags() & Qt::X11BypassWindowManagerHint) {
          w->setFlags(w->flags() & ~Qt::X11BypassWindowManagerHint);
          w->destroy();
          w->show();
          return false;
      }
    }
  }
  return Application::notify(o, e);
}
```

Do we need our own QPA plugin?

Further issues

- Cannot share composited OpenGL context with QtQuick
- Cannot use threaded QtQuick render loop
- QtQuick on hwcomposer aborts
- Intercept all input inside KWin anyway
- Have code to create X11 and Wayland windows
- Have code to create OpenGL context
- Have code to do low level event processing

Agenda

I propose to rename kwin to kwind because it swallows all features *(Kai-Uwe Broulik)*

Fixing the X11 security issues

Generic issues on X11

- KGlobalAccel is a global key logger
- Screen lockers are not secure (see Blog post “Why screen lockers on X11 cannot be secure”)
- All windows can edit all attributes of windows of foreign processes
- Windows can place themselves
- Windows can bypass Window Managers
- Clients can warp pointer
- Clients can grab foreign window content

KWin needs more knowledge about the windows

More control to the compositor

- KGlobalAccel moved into KWin
- Screen Locking needs to move into KWin
- Needs to know which windows belong to virtual keyboard
- Needs to know which process is desktop shell
- Needs to know which process is screen shot application
- Needs to know which process handles power management
- Needs to know the session splash screen
- Needs to authorize processes to access special interfaces

Suggestions for the problems appreciated!

It's tricky

- Don't duplicate code
- Don't hard depend on specific technology
- Everything should be flexible
- How to handle e.g. a shell process crash
- Don't harm user experience (no UAC)
- What about kded?

Agenda

XDG_Shell

So far only wl_shell support

- wl_shell is rather limited
- We make it useable with a Qt extension
- No support for Weston-demo clients
- No support for GTK+ clients

XDG_Shell under heavy development

- Unstable protocol mechanism
- GTK, Qt, Weston on real systems out of sync
- Need to get our (Qt, Plasma) needs into the protocol

Window Decorations

Issues with Qt deco

- Minimize button does nothing
- No visible distinction between active/inactive state
- Cannot configure button order
- Cannot add our own buttons
- It's not a good client-side deco solution, models server side

Possible Solution 1

Implement a better plugin based on KDecoration

Maybe better Solution?

- Disable Qt deco at runtime
- Read Qt::FramelessWindowHint in Extended surface
- Create server deco for all Qt Windows

Improvements in KWin

Lot's of features still missing

- Geometry handling missing
- Window types mostly missing
- Interaction with Plasma needs improvements
- Lots of small bugs here and there
- Window Rules missing

Please help us!

Plasma on Wayland on todo.kde.org

KWindowSystem

Modeled around X11

- Everywhere global Window Id
- Mixes API for own and foreign windows
- Platform abstraction is not a solution to support Wayland

Idea

Create a new API exposing a QAbstractItemModel which can be used by Task Managers.

Polish, polish, polish

Please Help!

- Starting KWin: kwin_wayland –xwayland
- Starting Plasma: startplasmacompositor

Tuesday is Wayland Day

Lab 0.5w

- 10:30 Wayland and Powerdevil and KScreen
- 11:30 Wayland and Plasma
- 15:00 Wayland and Applications

What is KWayland Client?

Qt style convenient library for Wayland

- Allow to use Wayland APIs in a Qt way
- Not a complete wrapper of Wayland yet
- Can integrate with QtWayland QPA
- Additional KWin/Plasma specific Wayland interfaces

Doesn't that duplicate QtWayland?

- QtWayland is a QPA plugin
- KWayland is an API which could be used to write a Wayland QPA plugin
- KWayland is to QtWayland, what KWindowSystem is to xcb QPA plugin

Additional Interfaces provided by KWayland

Already implemented

- org_kde_kwin_shadow (e.g. Plasma panel shadow)
- org_kde_kwin_idle (KF5IdleTime)
- org_kde_kwin_fake_input (kdeconnect)
- org_kde_plasma_shell
- org_kde_plasma_window_management

More to come, e.g.

- Blur and Background contrast effect
- Highlight Windows
- Present Windows
- Slide Windows

New Repository: kwayland-integration

New in Plasma 5.4

- Plugin for KWindowSystem
- Plugin for KIdleTime
- Place for any framework plugin which needs to depend on KWayland

What is KWayland Server?

The other side

- Qt-style API to implement a Wayland server
- Wrapper for the core Wayland protocols
- Wrapper for the KWin/Plasma specific interfaces
- No rendering!
- Implements lots of generic Wayland server functionality

Building a Wayland Server with KWayland

```
auto display = new KWayland::Server::Display(this);
display->start();
auto compositor = display->createCompositor(display);
compositor->create();
auto shell = display->createShell(display);
shell->create();
display->createShm();
auto seat = display->createSeat(display);
seat->create();
display->createDataDeviceManager(display)->create();
display->createIdle(display)->create();
auto plasmaShell = display->createPlasmaShell(display);
plasmaShell->create();
auto qtExtendedSurface = display->createQtSurfaceExtension(display);
qtExtendedSurface->create();
auto windowManagement = display->createPlasmaWindowManagement(display);
windowManagement->create();
auto shadowManager = display->createShadowManager(display);
shadowManager->create();
```

Interacting with the server

Example for a created object

```
connect(m_plasmaShell, &PlasmaShellInterface::surfaceCreated,
    [this] (PlasmaShellSurfaceInterface *surface) {
        if (ShellClient *client = findClient(surface->surface())) {
            client->installPlasmaShellSurface(surface);
        }
    }
);
```

Interacting with the server

Example for updating information in the server

```
void InputRedirection::processPointerMotion(const QPointF &pos, uint32_t time)
{
    // KWin internal handling for pointer motion removed for readability
#if HAVE_WAYLAND
    if (auto seat = findSeat()) {
        seat->setTimestamp(time);
        seat->setPointerPos(pos);
    }
#endif
}
```

Why not QtCompositor?

Comparable to Client vs QPA

- Our own interfaces are no fit for integration into Qt
- QtCompositor has not seen a release yet
- Focus on QtQuick useless for our needs
- Lot's of things which just doesn't fit our usecases