KDE Stack Overview and How It All Fit Together

High Level? Low Level? It'll be both my friend!

Kevin Ottens

HAUTE COUTURE

enioka

whoami

- Started to use KDE with 1.0-beta1 in 1997
- Procrastinated until 2003 to finally contribute code
- Fell in love with the community back then
- Kept doing things here and there... most notably helped with:
  - kdelibs
  - KDE Frameworks architecture
  - the KDE Manifesto
  - Community Data Analytics
- Part of the enioka Haute Couture family
- Living in Toulouse

Our Goals for Today

- Increase our general knowledge of the “KDE Stack”
  - As such it will be mostly high level views
  - Don’t worry there will be a few code snippets though
- Get a feel for how extensive it all is
  - Can’t be exhaustive though, would take days and be pointless
  - We’ll try to cover at least the most important/pervasive pieces
- Develop an idea of the integration points between all those pieces
  - Obviously has an impact on what we decided to cover or not
  - Also means we’ll have to go with lower level topics from time to time
- **Disclaimer**: Your head might spin, this is to be expected
  - It is a lot to absorb in one go
  - Ask questions along the way before you feel lost

Our Goals for Today

- Increase our general knowledge of the “KDE Stack”
  - As such it will be mostly high level views
  - Don’t worry there will be a few code snippets though
- Get a feel for how extensive it all is
  - Can’t be exhaustive though, would take days and be pointless
  - We’ll try to cover at least the most important/pervasive pieces
- Develop an idea of the integration points between all those pieces
  - Obviously has an impact on what we decided to cover or not
  - Also means we’ll have to go with lower level topics from time to time
- **Disclaimer**: Your head might spin, this is to be expected
  - It is a lot to absorb in one go
  - Ask questions along the way before you feel lost

History

Objectives

- Refresh our knowledge of signals and slots
- Have an idea of the complex history of our stack
- Identify the defining parts of our stack
- Highlight the relationship between Qt and the KDE stack
- Have a first approach to how our products fit together

Early Days

- 14 October 1996: Matthias Ettrich announces the “Kool Desktop Environment”
- Willing to use Qt which already had a lot of potential
- November 1996: kdelibs-0.0.1.tar.gz
- Just before Christmas: kwm, kpanel and kfm. . .
- February 1997: creation of the KDE-FreeQt Foundation
- May 1997: Linux-Kongress presentation
- August 1997: First paper in a german journal

KDE1

- 28 August 1997: KDE-One meeting
- 20 October 1997: Beta 1
- 23 November 1997: Beta 2
- December 1997: KDE e.V. is founded
- 1 February 1998: Beta 3
- 19 April 1998: Beta 4
- 12 July 1998: KDE 1.0

KDE1 cont'd

> *KDE is a **network transparent**, **contemporary** desktop environment for **Unix** workstations. KDE seeks to fill the need for an **easy to use** desktop for **Unix** workstations, similar to the desktop environments found under the MacOS or Window95/NT. We believe that the **Unix** operating system is the best operating system available today*

- Availability of OpenParts (CORBA based), and of KMail

KDE2

- 7 October 1999: KDE-Two meeting
  - Move away from CORBA, creation of Kanossa which will become KParts
  - Matthias Ettrich and Preston Brown get drunk and think they can write an ORB in one night. . .
  - . . . the result is DCOP!

- 9 July 2000: KDE-Three Beta meeting
- 23 October 2000: KDE 2.0
- Availability of
  - DCOP
  - KParts
  - KIO

KDE3

- 25 February 2002: KDE-Three meeting
- 3 April 2002: KDE 3.0
- 22 August 2003: Kastle (Czech Republic)
- 3 February 2004: KDE 3.2
- 21 August 2004: aKademy (Germany)
- 26 August 2005: aKademy (Spain)
- 29 November 2005: KDE 3.5

KDE4

- 23 September 2006: aKademy 2006 (Ireland)
- 14 October 2006: KDE has ten years
- 30 June 2007: aKademy 2007 (Scotland)
- 11 January 2008: KDE 4.0
- Switched from DCOP to DBus
- Availability of
  - Plasma
  - Phonon
  - Solid
  - ThreadWeaver
- 9 August 2008: aKademy 2008 (Belgium)
- 3 July 2009: Desktop Summit (Gran Canaria)
- 24 November 2009: Rebranding of KDE
- 3 July 2010: Akademy 2010 (Finland)
- December 2010: KDE Mobile meeting

KDE Frameworks 5 & 6

- June 2011: Platform 11 in Randa
- 6 August 2011: Desktop Summit (Berlin)
- 9 October 2011: Plasma Active One (KDE 4 based)
  - Calligra Active
  - Kontact Touch
- More Akademies obviously. . .

- 7 July 2014: KDE Frameworks 5.0
- 15 July 2014: Plasma 5.0
- 25 July 2015: Plasma Mobile announced

- 28 February 2024: KDE Frameworks 6.0 and Plasma 6.0

This training will focus mostly on the stack at that stage of evolution

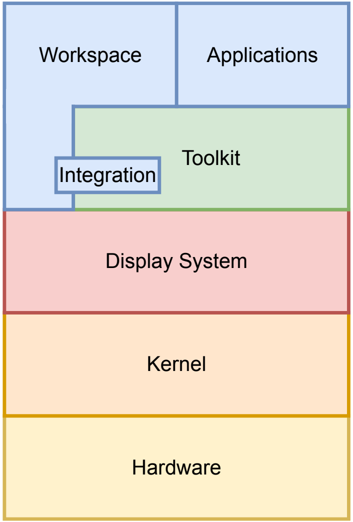
A Few Words About Qt's History

- 1991:
  - Eirik Chambe-Eng and Haavard Nord start writing Qt
  - The event loop and the signals/slots mechanisms are already there
- 1994: Trolltech incorporated in Oslo, Norway
- 1996: First Qt sale (ESA!)
- 1997: creation of the KDE-FreeQt Foundation
- 1999: Qt2
- 2000:
  - Qt/Embedded
  - Qt/X11 available under the GPL
  - uic and designer are introduced in Qt 2.2
- 2001:
  - Sharp uses Qtopia in its products
  - Qt3
- 2003: Qt/Mac available under GPL
- 2004: Qtopia Phone Edition is released

A Few Words About Qt's History cont'd

- 2005:
  - Qt4 with Interview (MVC for item views) and Arthur (2D painting engine)
  - Qt/Windows available under the GPL
- 2008: Acquired by Nokia
- 2010: QML and QtQuick are introduced in Qt 4.7
- 2011: Qt Platform Abstraction is introduced in Qt 4.8
- 2012: Acquired by Digia, Qt5 released
- 2014: The Qt Company demerger starts
- 2020: Qt 6.0
- 2024: Qt 6.7

Our Stack: 10'000 Feet View



- Base workspace: KWin, PlasmaShell, etc.
- Applications: Krita, Dolphin, Elisa, etc.
- Toolkit: Qt **and** KDE Frameworks
- Display System: X.org or Wayland
- Kernel: Linux or *BSD
- Hardware: anything that can run the kernel and the display system

- Integration: plugins for the toolkit to play nicely in the workspace
  - This is essential for what we do
  - Always keep it in mind!

Thank You For Coming!

Questions?

Just Kidding. . .

Questions and Answers

- Which of the most ancient pieces of tech in the KDE stack you spotted?
- Which are the most ancient important mechanisms in Qt?
- Which Qt Widgets defining features can you cite?
- How do the workspace and the toolkit relate to each other?

Questions and Answers

- Which of the most ancient pieces of tech in the KDE stack you spotted?
- Which are the most ancient important mechanisms in Qt?
- Which Qt Widgets defining features can you cite?
- How do the workspace and the toolkit relate to each other?

Questions and Answers

- Which of the most ancient pieces of tech in the KDE stack you spotted?
- Which are the most ancient important mechanisms in Qt?
- Which Qt Widgets defining features can you cite?
- How do the workspace and the toolkit relate to each other?

Questions and Answers

- Which of the most ancient pieces of tech in the KDE stack you spotted?
- Which are the most ancient important mechanisms in Qt?
- Which Qt Widgets defining features can you cite?
- How do the workspace and the toolkit relate to each other?

Key Takeaways

- The initial KDE vision is still resonating with our products today (e.g. network transparent is still very much a thing)
- CORBA has been defining to our stack, it's what led to KParts and ultimately DBus
  - Sidenote: You pretty much need KIO to support some of KParts features
- The form factor discussion and going mobile goes back a long way if you account for Qt history
- Plasma predates QtQuick and QML
- All our products have a well defined role in the stack
- We'll talk quite a bit about how to keep it all properly integrated when used altogether

Anatomy of a Qt Application

Objectives

- Have a better idea about how the event loop works
- Also have a slightly closer view to the platform abstraction layer in Qt
- Get a first approach at how we leverage both
- Weight how application code is structured in QtWidget applications vs QtQuick applications

Signals and Slots Refresher (1/5)

aka The Qt Dev Bread and Butter

- An object “emits” a signal when something potentially interesting to the outside happens
- One or more objects receive the signal thanks to a method having a compatible signature
- Easy to go event based programming
- Loose coupling

Signals and Slots Refresher (2/5)

aka The Qt Dev Bread and Butter

```
#include <QObject>
#include <QPoint>

class Beacon : public QObject
{
    Q_OBJECT
signals:
    void beamOfLight(const QPoint &pos, int degree);
};
```

Signals and Slots Refresher (3/5)

aka The Qt Dev Bread and Butter

```
#include <QObject>
#include <QPoint>

class Ship : public QObject
{
    Q_OBJECT
slots:
    void lightSpotted(const QPoint &pos, int degree);
};
```

Signals and Slots Refresher (4/5)

aka The Qt Dev Bread and Butter

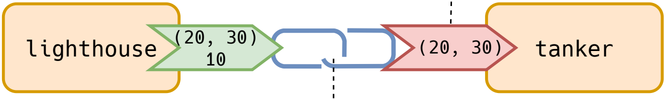
```
Beacon *lighthouse;
Ship *tanker;
connect(lighthouse, &Beacon::beamOfLight
        tanker, &Ship::lightSpotted);
```

Signals and Slots Refresher (5/5)

aka The Qt Dev Bread and Butter

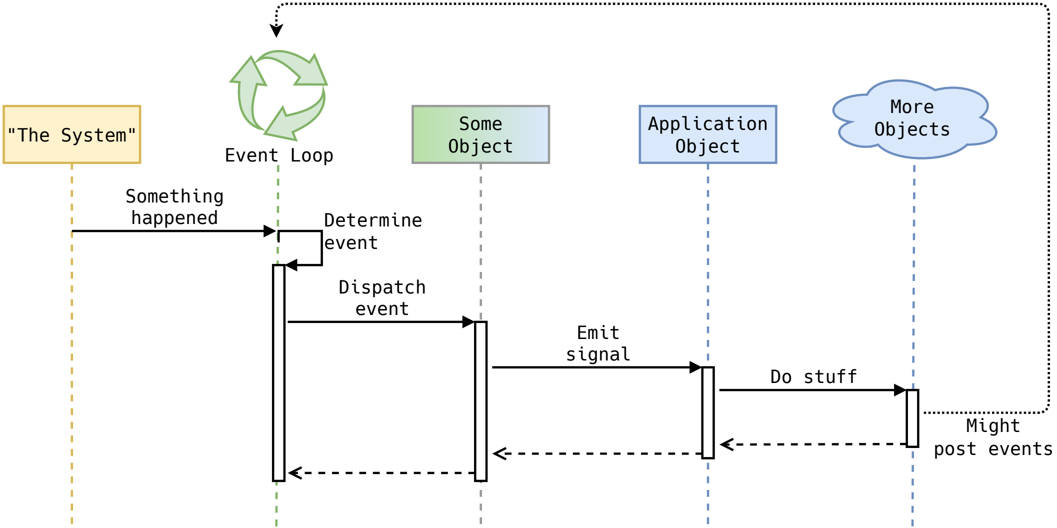
```
void Beacon::rotate(int angle) {
    // ...
    m_angle += angle;
    emit beamOfLight(m_position,
                     m_angle);
    // ...
}
```

```
void Ship::lightSpotted(QPoint pos) {
    // ...
    m_lastListPos = pos;
    // ...
}
```



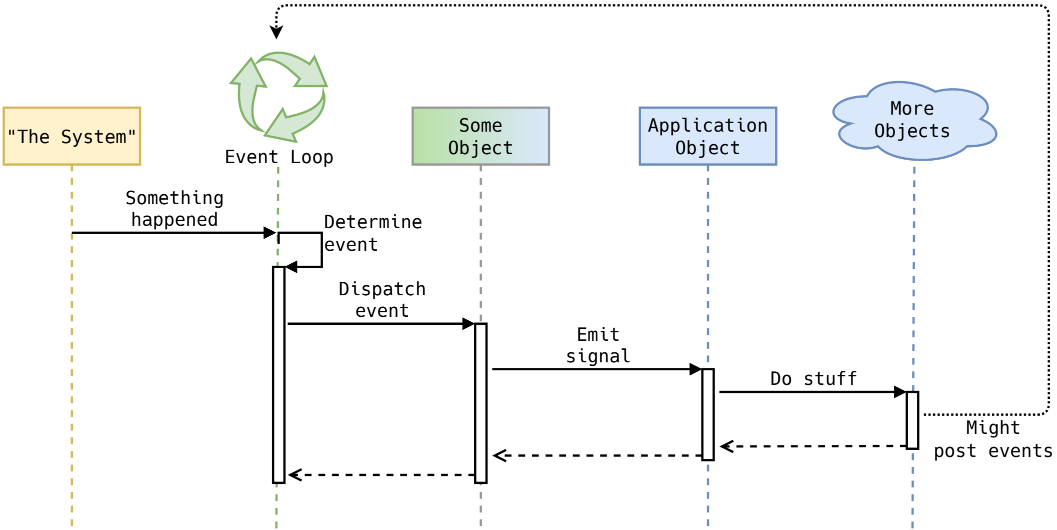
```
connect(lighthouse, &Beacon::beamOfLight,
        tanker, &Ship::lightSpotted);
```

The Event Loop



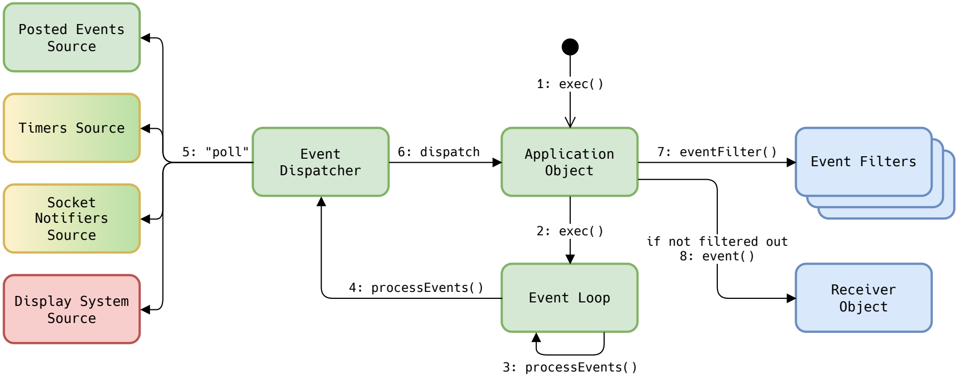
- Ultimately all application code is triggered by an event loop

The Event Loop



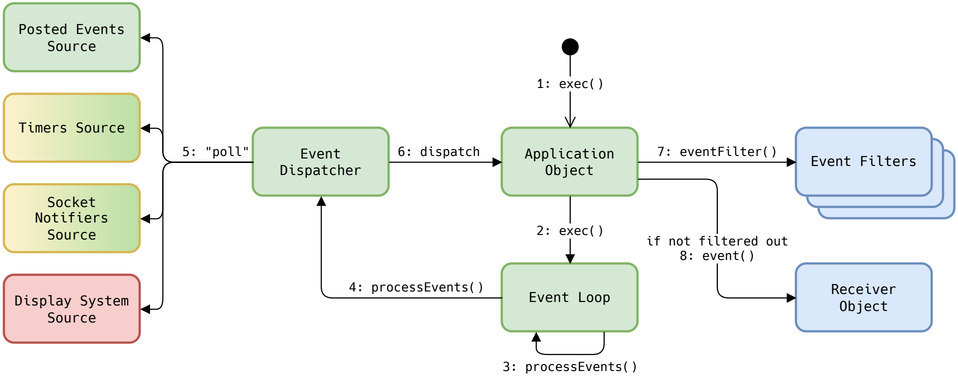
- Ultimately all application code is triggered by an event loop

The Event Loop (Extended)



- **Disclaimer:** This is still a simplification, the code has the details
- Situation may vary for user events depending on system setup, can come from:
  - display system
  - socket notifiers (typically for libinput and evdev)
- Keep in mind the application object has hooks, like startup/shutdown hooks
  - Will be important later. . .

The Event Loop (Extended)



- **Disclaimer:** This is still a simplification, the code has the details
- Situation may vary for user events depending on system setup, can come from:
  - display system
  - socket notifiers (typically for libinput and evdev)
- Keep in mind the application object has hooks, like startup/shutdown hooks
  - Will be important later...

But wait. . .

- Where do the event dispatcher come from?
- And what about the display system?

- For sure it's not all wired in at compile time. . .

But wait. . .

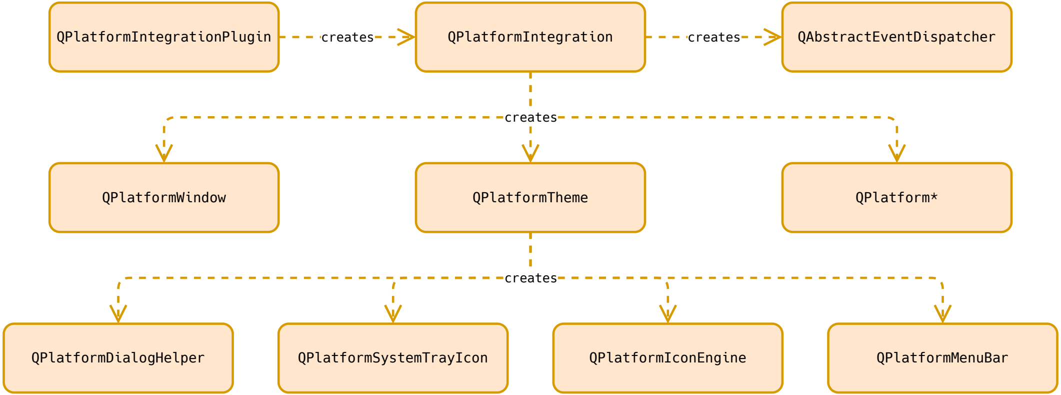
- Where do the event dispatcher come from?
- And what about the display system?

- For sure it's not all wired in at compile time. . .

Qt Platform Abstraction (QPA)

- Platform abstraction layer
- Intent is to make it invisible to application developers
- Plugin system at two level of abstractions
- Most plugins are provided with Qt:
  - `QAndroidIntegrationPlugin`
  - `QWindowsIntegrationPlugin`
  - `QXcbIntegrationPlugin`
  - `QWayland*IntegrationPlugin`
  - and more...

QPA classes


QPlatformIntegrationPlugin
creates
QPlatformIntegration
creates
QAbstractEventDispatcher
creates
QPlatformWindow
QPlatformTheme
QPlatform*
creates
QPlatformDialogHelper
QPlatformSystemTrayIcon
QPlatformIconEngine
QPlatformMenuBar

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait...

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do...
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

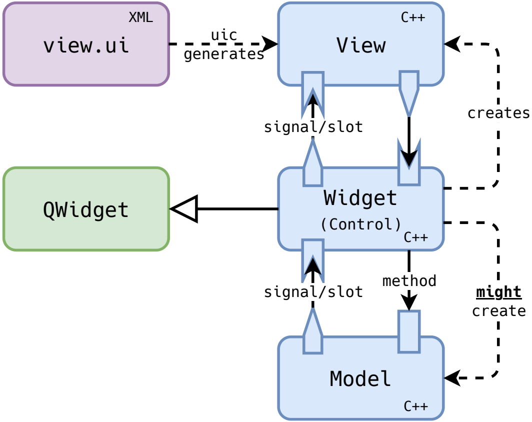
But wait. . .

Most plugins are provided with Qt

- The KDE file dialog isn't implemented **inside** Qt, is it?
- Indeed not, it's implemented **using** Qt
- We ship a plugin somewhere!?
- Yes we do. . .
  - More about this later
  - For now, make a mental note of this integration point

- Fun fact: our KDE Frameworks and Plasma code is **both** on top and below Qt
- I won't dive into it here, but take a minute to think about KWin vs QPA
  - "Interesting" problems all around

Your Typical Qt Widgets Application Architecture



- Of course, this is not about a single class of each type
- This pattern tends to be repeated over and over in applications
- Imagine
  - Numerous `Model`, `Widget` and `View` classes
  - Complex relationship between `Model` and `Widget` classes
  - Each `Widget` has only one `View` though
    - I know there are exceptions, considering the most pervasive scenario here

Your Typical Qt Widgets Application Architecture cont'd

The Control Part

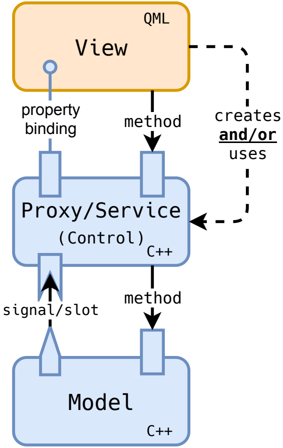
```
class Widget : public QWidget {
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = nullptr)
      : ui{std::make_unique<Ui::View>()} {
        ui->setupUi(this);
        // Probably a bunch of connects to/from widgets from ui,
        // slots likely manipulating m_model
    }
    void setModel(Model *model) { // Not always the case...
        // Maybe a bunch of connects from model, slots impacting ui
        m_model = model;
    }
private:
    std::unique_ptr<Ui::View> ui;
    Model m_model = nullptr;
};
```

Your Typical Qt Widgets Application Architecture cont'd

Putting It All Together

```
auto model = new Model;
auto widget = new Widget;
widget->setModel(model); // Not always the case...
widget->show();
```

Your Typical Qt Quick Application Architecture



- This is still not about a single class of each type
- This pattern tends to be repeated over and over in applications
- Imagine
  - Numerous `Model` and `Proxy/Service` classes, numerous `View` scripts
  - Complex relationship between `Model` and `Proxy/Service` classes
  - Also complex relationship between `Proxy/Service` classes and `View` scripts
    - Each `View` can easily use many `Proxy/Service`

Your Typical Qt Quick Application Architecture cont'd

The Control Part

```
class Proxy : public QObject {
    Q_OBJECT
    Q_PROPERTY(QString modelId READ modelId WRITE setModelId
               NOTIFY modelIdChanged)
    Q_PROPERTY(QString value READ value WRITE setValue
               NOTIFY valueChanged)
    QML_ELEMENT
public:
    using QObject::QObject

    // Getter and setters for the properties above

private:
    // Locate or create the model parts we need based on modelId
    Model *model() const;
};
```

Your Typical Qt Quick Application Architecture cont'd

The View Part

```
import QtQuick 2.0 as QQ
// Assuming Proxy has been registered in the org.kde.app namespace
import org.kde.app 1.0 as App

QQ.Item {
    App.Proxy {
        id: proxy
        modelId: "whatWeNeed"
    }
    QQ.Text {
        anchors.centerIn: parent
        text: proxy.value
    }
}
```

Your Typical Qt Quick Application Architecture cont'd

Putting It All Together

```
QQmlApplicationEngine engine;
// Assuming the View.qml has been registered in the org.kde.app module
engine.loadFromModule("org.kde.app", "View");
```

Anatomies of Qt Applications

- The pattern used for QtWidgets based applications gets in the way of reusability
  - Overfitted to the composition pattern of widgets
  - It doesn't have to be like this though
- The pattern used for QtQuick based applications is (IMHO) superior
  - Less coupling between view and control
  - Or at least the coupling goes in the "right direction"
  - It's in fact closer to the Model-View-Presenter (MVP) pattern
- It is of course not impossible to have a QtQuick like approach for widgets based applications
  - It's really not often done only for historical reasons
- Qt Bindable Properties were supposed to make that easier on the C++ side, but...
  - Nothing in QtWidgets use them
  - Limited adoption in KDE Frameworks so far

Anatomies of Qt Applications

- The pattern used for QtWidgets based applications gets in the way of reusability
  - Overfitted to the composition pattern of widgets
  - It doesn't have to be like this though
- The pattern used for QtQuick based applications is (IMHO) superior
  - Less coupling between view and control
  - Or at least the coupling goes in the "right direction"
  - It's in fact closer to the Model-View-Presenter (MVP) pattern

- It is of course not impossible to have a QtQuick like approach for widgets based applications
  - It's really not often done only for historical reasons

- Qt Bindable Properties were supposed to make that easier on the C++ side, but...
  - Nothing in QtWidgets use them
  - Limited adoption in KDE Frameworks so far

Anatomies of Qt Applications

- The pattern used for QtWidgets based applications gets in the way of reusability
  - Overfitted to the composition pattern of widgets
  - It doesn't have to be like this though
- The pattern used for QtQuick based applications is (IMHO) superior
  - Less coupling between view and control
  - Or at least the coupling goes in the “right direction”
  - It's in fact closer to the Model-View-Presenter (MVP) pattern
- It is of course not impossible to have a QtQuick like approach for widgets based applications
  - It's really not often done only for historical reasons
- Qt Bindable Properties were supposed to make that easier on the C++ side, but...
  - Nothing in QtWidgets use them
  - Limited adoption in KDE Frameworks so far

Why am I covering this?

- It's linked to the history section. . .
- We did a lot before QtQuick was around, so a lot of our stack comes from applications following the QtWidgets typical patterns
- This is sometimes a limitation to reusing "business logic" in QtQuick applications
- Pay attention to what you pick and where it's coming from
- If possible: retrofit something which exists into newer APIs instead of duplicating features

Anatomy of a KDE Application

Anatomy of a Qt Application (again)

Used to be different, now it's pretty much the same. . .

. . . maybe with more dependencies

Anatomy of a Qt Application (again)

Used to be different, now it's pretty much the same…

… maybe with more dependencies

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Questions and Answers

- Which are the sources of events for the event loop?
- Which object is responsible for dispatching events?
- What happens to events when they get dispatched?
- What is the relationship between events and application code?
- Which Qt mechanism is putting the event loop and its sources in place?
- What is the usual pattern for QtWidgets application?
- This pattern gets in the way of something, what is it?
- Which pattern the QtQuick pattern seen here is close from?

Key Takeaways

- The “event loop” is in fact quite a few objects in interaction
- We have quite a few source of events
- The application object is central to the event dispatch mechanism
- The application object provides us hooks (still mysterious for now, but important)
- QPA has a strong say in how the event loop is wired
- QPA is also an important mechanism for us to plug our own platform behaviors
- Our QPA use is fairly unique since it is “Qt based all the way”

- To maximize reusability of “business logic” aim to structure your application similarly to a QtQuick application, even if it uses QtWidgets
- There’s really not anything specific anymore to applications coming from KDE apart from their dependencies

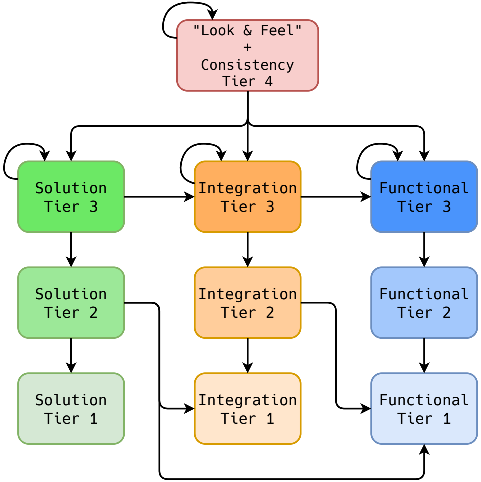
KDE Frameworks

Objectives

- Take a whirlwind tour of KDE Frameworks
- Get a feel of the staggering amount of features available
- Understand better how it is structured overall
- Highlight some of the integration points
- See how some of those integration points relate to Plasma

All About Managing Dependencies

Of Tiers and Types



- Tiers
  1. Depends only on Qt and system libraries
  2. Depends on Tier 1 and its dependencies
  3. Depends on anything in Tier 3 or below
  4. Depends on anything, has a purpose and almost no API
- Types
  - Functional: “Qt Add-ons” with no runtime dependencies
  - Integration: optional runtime dependency, aiming at integrating with a given platform
  - Solution: mandatory runtime dependency, part of the design and added value
    - e.g. scalability, resource sharing, resilience
- They are both part of the information listed in the `metainfo.yaml` files of our frameworks

Disclaimers

- This section will obviously be a bit more of a catalog
- This is not an exhaustive training covering all the APIs. . .
  - Otherwise, we'd have to cover more than 70 frameworks
  - Also expect some classes to be barely described
    - Otherwise, we'd have several days worth of content
    - We'll make sure to focus on the ones we consider very important
- Brace yourselves!

Disclaimers

- This section will obviously be a bit more of a catalog
- This is not an exhaustive training covering all the APIs...
  - Otherwise, we'd have to cover more than 70 frameworks
  - Also expect some classes to be barely described
    - Otherwise, we'd have several days worth of content
    - We'll make sure to focus on the ones we consider very important
- Brace yourselves!

Disclaimers

- This section will obviously be a bit more of a catalog
- This is not an exhaustive training covering all the APIs. . .
  - Otherwise, we'd have to cover more than 70 frameworks
  - Also expect some classes to be barely described
    - Otherwise, we'd have several days worth of content
    - We'll make sure to focus on the ones we consider very important
- Brace yourselves!

Tier 1

Remember, all of the following frameworks build straight on top of Qt

Tier 1: KCoreAddons

- KSharedDataCache allows to cache data and share it across processes
- KAutoSaveFile provides temporary files used to store unsaved data (file open for editing), also allows for the recovery of old autosaved data
- KProcess extends QProcess with extra features to ease management of the output channels and environment
- KAboutData stores information about a program (version number, authors, licence, home page. . .)
- KFuzzyMatcher provides SublimeText like fuzzy matching
- KUser represents system users (works on Windows too)
- KPluginFactory and KPluginMetaData for creating and loading plugins
- KMacroExpander provides convenient macro substitution in strings

```
QHash<QString,QString> map;
map.insert("url", "/tmp/myfile.txt");
map.insert("name", "My File");
QString s = "Title: %{url}-%name";
s = KMacroExpander::expandMacros(s, map);
```

Tier 1: KCoreAddons, Focus on Jobs

```
class CreateResourceJob : public KJob {
    Q_OBJECT
public:
    using KJob::KJob;
    void setUrl(const QUrl &url);

    void start() override {
        QMetaObject::invokeMethod(this, &CreateResourceJob::doStart,
                                  Qt::QueuedConnection);
    }

private:
    void doStart();
    void onFinished(bool success);

    QUrl m_url;
};
```

Tier 1: KCoreAddons, Focus on Jobs

```
void CreateResourceJob::doStart() {
    // Do something async, like use QNAM
    connect(..., this, &CreateResourceJob::onFinished);
}

void CreateResourceJob::onFinished(bool success) {
    if (!success) {
        setError(-1);
        setErrorText("Oops");
    }
    emitResult();
}
```

Tier 1: KCoreAddons, Focus on Jobs cont'd

```
auto job = new CreateResourceJob(parent);
job->setUrl(...);
connect(job, &CreateResourceJob::result, [](Kjob *job) { ... });

job->setUiDelegate(...);
jobTracker->registerJob(job);
job->start();
```

- All the necessary to use a job is on that slide
- We also have `KCompositeJob` allowing to build jobs managing other jobs, job queues etc.
- We'll come back to UI delegates and job trackers but they allow respectively
  - To interact with the user during the job lifetime (ask questions, display errors...)
  - To display job progress

Tier 1: KWidgetsAddons

- `KStyleExtensions` allows to declare extension points for `KStyle`
  - More about this later
- `KMessageDialog` displays messages to the user, supports notifications and “don’t show again” feature
- `KRatingWidget` displays a rating value (row of stars or other pixmap)
- `KColumnResizer` ensures columns are of the same width across layouts
- `KDualAction` provides an action with two states (texts and icons)
- `KActionMenu` is an action providing a menu of other actions
- `KBusyIndicatorWidget` is a spinning icon indicating we’re busy
- `KCapacityBar` shows the level of usage of a resource (similar to but not quite a progress bar)
- `KFontRequester` allows the user to pick a font

Tier 1: KWidgetsAddons cont'd

- `KNewPasswordWidget` and `KNewPasswordDialog` allow the user to enter a new password (needs to be entered twice and they give a hint on the password strength)
- `KPasswordLineEdit` allows the user to input a password and to get it displayed
- `KColorCombo` displays a combo box to pick colors
- `KDateComboBox` displays a combo box to pick dates
- `KDatePicker` displays a calendar to select a date
- `KUrlLabel` is a replacement for `QLabel` when you need to display URLs
- `KRecentFilesMenu` provides a menu for recently opened files
- `KPageView` and `KPageDialog` provides multiple pages support in a view, very configurable you can pick the type of rendering (list, tree, tabs) to switch between the pages

Tier 1: KConfig

```
auto config = KSharedConfig::openConfig("settingsrc");
KConfigGroup group(config, "Appearance");

const auto color = group.readEntry("AlertColor", QColor(Qt::red));

group.writeEntry("AlertFont", QFont("Hack", 12));
```

- Allows to easily dissociate actual settings from application state
  - `KSharedConfig::openConfig()` vs `KSharedConfig::openStateConfig()`
- Supports nested groups
- Supports config cascading (convenient for sysadmins)
- Allows to lock settings or provide defaults via the cascading
  - Also provides the `KAuthorized` namespace to know if sysadmins decided to lock down an action or a control module (more on this later)

Tier 1: KConfig Extended (KConfigXT)

```
<kcfg>
  <kcfgfile name="settingsrc"/>
  <group name="Appearance">
    <entry key="AlertColor" type="Color">
      <default>255, 255, 255</default>
    </entry>
    <entry key="AlertFont" type="Font">
    </entry>
  </group>
</kcfg>
```

- Most Qt data types supported
- The XML syntax allows for hints to be used in the GUI (`label`, `whatsthis`)

Tier 1: KConfig Extended (KConfigXT) cont'd

```
Settings settings;
const auto color = settings.alertColor();
settings.setAlertFont(QFont("Hack", 12));
```

- Code generated using `kconfig_compiler`
  - CMake macros provided
  - Type safe configuration with opt-in change notification
- Provides all the necessary hooks for introspecting the settings
- Config values state management
  - Is it set to the default value?
  - Reset it to the default value
- It's all KConfig under the hood so we benefit from all its features as well

Tier 1: Kirigami

```
Kirigami.ApplicationWindow {
    id: root
    globalDrawer: Kirigami.GlobalDrawer {
        title: "Global Actions"
        titleIcon: "icon-name"
        actions: [ ... ]
    }
    contextDrawer: Kirigami.ContextDrawer { }
    pageStack.initialPage: mainPageComponent

    Component {
        id: mainPageComponent
        Kirigami.ScrollablePage {
            ...
        }
    }
}
```

Tier 1: Kirigami cont'd

- UIs using Kirigami are adaptable or "convergent"
  - Work nicely both on mobile and desktop
  - Follows the KDE Human Interface Guidelines
- Provide quite a few components:
  - Windows, Actions and Drawers
  - Page system with routing
  - ScrollablePage
  - Card, CardListView, CardGridView
  - And more. . .

Tier 1: And a Few More

- `KArchive` fullfills for all your archives and compression needs
- `KCalendarCore` allows to manipulate iCal data
- `KDBusAddons` provides extra helpers for interprocess communication and IPC locks
- `KI18n` is the base of our translation system, based on gettext and very advanced
- `KItemModels` provides lots of useful proxy modesl
- `KWindowSystem` allows to interact with certain features of the window manager (exact availability of the features depend on the platform)
- `Solid` allows to query the system for available hardware, find mountpoints. . .
- `ThreadWeaver` provides a complex multithreaded job queue, it allows the creation of complex flow graphs

Tier 2

Remember, all of the following frameworks build on top of Qt + Tier 1

Tier 2: KJobWidgets

Depends on KCoreAddons and KWidgetsAddons

- `KDialogJobUiDelegate` provides dialogs for interacting with a job
- `KNotificationJobUiDelegate` provides notifications for interacting with a job
- `KWidgetJobTracker` allows to display jobs progression in a widget
- `KStatusBarJobTracker` allows to display jobs progression in a widget suitable for embedding in a status bar
- `KUiServer(V2)JobTracker` allows to display jobs progression through the DBus service exposed by Plasma

Tier 2: KNotifications

Depends on KWindowSystem, KConfig and KCoreAddons

- Cross platform library for creating popup notifications
- Requires a config file to be shipped by the application to describe its events
- `KNotification` allows to create a notification to be displayed corresponding to an event
- `KNotifyConfig` exposes the configuration for an event

```
KNotification *notification = new KNotification("contactOnline");
notification->setText(i18n("<i>%1</i> is now online", contact->name());
notification->setPixmap(contact->pixmap());
KNotificationAction *action = notification->addAction(i18n("Open chat"));
connect(action, &KNotification::activated,
        contact, &Contact::slotOpenChat);
notification->sendEvent();
```

Tier 2: KUnitConversion

Depends on KI18n

- Simple API
- Supports lots of fields:
  - Currency (yes, with daily updates of the conversion rates)
  - Acceleration
  - Angle
  - Area
  - Binary Data
  - Density
  - Length
  - Temperature
  - Voltage
  - And more. . .

Tier 2: KPackage

Depends on KArchive, KI18n and KCoreAddons

- Allows users to install and load packages of non binary content
  - Typically scripted extensions or graphic assets
- `KPackage::Package` represents a package of a given type
- `KPackage::PackageStructure` describes the allowed files and folders in a package type, shipped as plugins
- `KPackage::PackageLoader` find and loads packages of a given package type
- This is also the distribution format for Plasma extensions

Tier 3

This is where the dependencies get harder to manage!

Tier 3: KCMUtils

- KCModule provides a base class for configuration modules
- KCMultiDialog provides a settings dialog displaying a set a KCModules
- KPluginWidget allows to select which plugins to load in an application and to configure them
- KCModuleData provides an extension to KCModule to know the state of a module without loading the whole GUI

- Also contains the necessary to make QML based KCModule
- KQuickConfigModule and KQuickManagedConfigModule allow to implement the backend part of a QML based KCM

- org.kde.kcmutils on the QML side provides convenient elements to implement the frontend part of a QML based KCM
  - SimpleKCM for the root element
  - GridViewKCM for the root element in config module mostly exposing a grid of items

Tier 3: KCMUtils

- KCModule provides a base class for configuration modules
- KCMultiDialog provides a settings dialog displaying a set a KCModules
- KPluginWidget allows to select which plugins to load in an application and to configure them
- KCModuleData provides an extension to KCModule to know the state of a module without loading the whole GUI

- Also contains the necessary to make QML based KCModule
- KQuickConfigModule and KQuickManagedConfigModule allow to implement the backend part of a QML based KCM

- org.kde.kcmutils on the QML side provides convenient elements to implement the frontend part of a QML based KCM
  - SimpleKCM for the root element
  - GridViewKCM for the root element in config module mostly exposing a grid of items

Tier 3: KCMUtils

- `KCModule` provides a base class for configuration modules
- `KCMultiDialog` provides a settings dialog displaying a set a `KCModules`
- `KPluginWidget` allows to select which plugins to load in an application and to configure them
- `KCModuleData` provides an extension to `KCModule` to know the state of a module without loading the whole GUI

- Also contains the necessary to make QML based `KCModule`
- `KQuickConfigModule` and `KQuickManagedConfigModule` allow to implement the backend part of a QML based KCM

- `org.kde.kcmutils` on the QML side provides convenient elements to implement the frontend part of a QML based KCM
  - `SimpleKCM` for the root element
  - `GridViewKCM` for the root element in config module mostly exposing a grid of items

Tier 3: KConfigWidgets

- `KConfigDialog` completes `KPageDialog` with the logic necessary for making settings dialog (state management of the buttons, loading, saving...)
- `KConfigViewStateSaver` allows to save/restore item views state in KConfig
- `KCommandBar` provides a hud style menu
- `KHamburgerMenu` allows to replace the menu bar when necessary

Tier 3: KXmlGui

- Encodes lots of rules of what we consider a “KDE application on the desktop”
- Automatically respects `KAuthorized` hints
- `KAboutApplicationDialog` provides the standard about dialog
- `KActionCollection` provides a container for named actions
- `KXmlGuiWindow` provides a top level window with action management
  - It gives all the necessary to encode the menu structure and the toolbars
  - It let the user edit the toolbars
- `KXMLGUIFactory` and `KXMLGUIClient` provide the same features but without being tied to a given window
  - Each client provides actions (`KActionCollection`) and some rules on how to insert actions in the GUI (XML format)
  - The factory plugs the action into container widgets via `KXMLGUIBuilder`
  - It is possible to apply more than one client, effectively merging their actions in a single structure

Tier 3: KXmlGui cont'd

KActionConflictDetector: an example of enforcing rules

```
class KActionConflictDetector : public QObject {
    // ...

    bool eventFilter(QObject *watched, QEvent *event) override {
        if (event->type() == QEvent::Shortcut &&
            qobject_cast<QAction *>(watched)) {
            QShortcutEvent *se = static_cast<QShortcutEvent *>(event);
            if (se->isAmbiguous()) {
                KMessageBox::information(...);
                return true;
            }
        }
        return QObject::eventFilter(watched, event);
    }
};
```

Tier 3: KXmlGui cont'd

KActionConflictDetector: injection

```
void _k_installConflictDetector() {
    QCoreApplication *app = QCoreApplication::instance();
    app->installEventFilter(new KActionConflictDetector(app));
}

Q_COREAPP_STARTUP_FUNCTION(_k_installConflictDetector)
```

- This will work as soon as you link against KXmlGui
- Our `QPlatformTheme` plugin links against KXmlGui

Tier 3: KXmlGui cont'd

KActionConflictDetector: injection

```
void _k_installConflictDetector() {
    QCoreApplication *app = QCoreApplication::instance();
    app->installEventFilter(new KActionConflictDetector(app));
}

Q_COREAPP_STARTUP_FUNCTION(_k_installConflictDetector)
```

- This will work as soon as you link against KXmlGui
- Our QPlatformTheme plugin links against KXmlGui

Tier 3: KXmlGui cont'd

KActionConflictDetector: injection

```
void _k_installConflictDetector() {
    QCoreApplication *app = QCoreApplication::instance();
    app->installEventFilter(new KActionConflictDetector(app));
}

Q_COREAPP_STARTUP_FUNCTION(_k_installConflictDetector)
```

- This will work as soon as you link against KXmlGui
- Our QPlatformTheme plugin links against KXmlGui

Tier 3: KIO

- Provides network transparent access to files and data
- Also provides facilities to launch applications or open files (local or distant)
- Asynchronous API via jobs
- Plugin system to implement the protocols
  - File
  - HTTP
  - SMB
  - SFTP
  - MTP
- Allows to make virtual filesystems too
  - Trash
  - Timeline
  - Desktop
  - Applications
  - AudioCD
- Many of the workers are in `kio-extras`

Tier 3: KIO cont'd

```
auto job = KIO::listDir(url);

connect(job, &KIO::ListJob::entries, [=](
    KIO::Job *, const KIO::UDSEntryList &entries) {
    for (const auto &entry : entries) {
        KFileItem file(entry, url, false, true);
        qDebug() << "Seen entry:" << file.text()
                 << file.url().toDisplayString();
    }
});

connect(job, &KJob::result, [](KJob *job) {
    if (job->error()) {
        job->uiDelegate()->showErrorMessage();
    }
});
```

Tier 3: KIO cont'd

- `KIO::Job` subclasses are auto-start
- If linking against `KIOWidgets` they automatically
  - Get a UI delegate with extensions
    - e.g. deals with asking the user to rename a file if needed
  - Get a job tracker
    - Dynamically dispatches to `KWidgetJobTracker` or `KUiServerJobTracker`
    - Depends if the application is in a Plasma session or not

```
static void registerJobUiDelegate() {
    KIO::setDefaultJobUiDelegateFactory(globalUiDelegateFactory());
    KIO::setDefaultJobUiDelegateExtension(globalUiDelegate());
}

Q_CONSTRUCTOR_FUNCTION(registerJobUiDelegate)
```

Tier 3: KIO cont'd

- We have loads of different jobs
- KIO also comes with plenty of widgets
  - Pretty much all you need to navigate filesystems

This framework is very large, make sure to check its API documentation

Tier 3: KIO cont'd

- We have loads of different jobs
- KIO also comes with plenty of widgets
  - Pretty much all you need to navigate filesystems

This framework is very large, make sure to check its API documentation

The Mythical Tier 4

This provides mostly no API, it's mainly here to tie some pieces together

Tier 4: Framework Integration Plugin

"Don't ask again" requires KConfig

```
class KMessageBoxDontAskAgainInterface
{
public:
  // ...
  virtual bool shouldBeShownYesNo(const QString &dontShowAgainName,
                                  KMessageBox::ButtonCode &result) = 0;
  virtual void saveDontShowAgainYesNo(const QString &dontShowAgainName,
                                      KMessageBox::ButtonCode result) = 0;
  virtual void enableAllMessages() = 0;
  virtual void enableMessage(const QString &dontShowAgainName) = 0;
  virtual void setConfig(KConfig *) = 0;
  // ...
};
```

- The `dontShowAgainName` is passed to the public API of `KMessageBox` or `KMessageDialog`

Tier 4: Framework Integration Plugin

KMessageDialog interacts with the notification system

```
class KMessageBoxNotifyInterface
{
public:
    // ...
    virtual void sendNotification(QMessageBox::Icon notificationType,
                                  const QString &message,
                                  QWidget *parent) = 0;
};
```

- KMessageBox and KMessageDialog have API allowing to enable/disable notifications for specific messages

Tier 4: KStyle

- Remember in KWidgetsAddons things like `KCapacityBar`?
- They need styles to know about them for better tuning...
  - ... but `QStyle` can't know them
- That's the main reason for `KStyle` (and `KStyleExtensions`) existence

Tier 4: KStyle

In a Style Inheriting From ‘KStyle‘

- For instance Breeze contains code like this to initialize a member variable:

```
CE_CapacityBar(newControlElement(QStringLiteral("CE_CapacityBar")))
```

- And inside the rendering path:

```
if (element == CE_CapacityBar) {
    // ...
}
```

Tier 4: KStyle

In the Widget Constructor

```
KCapacityBar::KCapacityBar(KCapacityBar::DrawTextMode drawTextMode,
                           QWidget *parent)
    : QWidget(parent)
    , d(new KCapacityBarPrivate(drawTextMode))
{
    d->ce_capacityBar =
        KStyleExtensions::customControlElement("CE_CapacityBar", this);
}
```

Tier 4: KStyle

In the Widget Paint Code

```
void KCapacityBar::drawCapacityBar(QPainter *p, const QRect &rect) const
{
    if (d->ce_capacityBar) {
        QStyleOptionProgressBar opt;
        opt.initFrom(this);
        // ...
        style()->drawControl(d->ce_capacityBar, &opt, p, this);
        return;
    }

    // very long manual fallback with straight QPainter use...
}
```

Friendly Reminders

- This was really a quick and biased tour
- Go to the API documentation, you'll find many more
- I tried to focus on points which would either
  - Give a feel of how many features it all packs, or
  - Would ensure we'd bump into some lesser obvious integration points

Friendly Reminders

- This was really a quick and biased tour
- Go to the API documentation, you'll find many more
- I tried to focus on points which would either
  - Give a feel of how many features it all packs, or
  - Would ensure we'd bump into some lesser obvious integration points

Friendly Reminders

- This was really a quick and biased tour
- Go to the API documentation, you'll find many more
- I tried to focus on points which would either
  - Give a feel of how many features it all packs, or
  - Would ensure we'd bump into some lesser obvious integration points

Friendly Reminders

- This was really a quick and biased tour
- Go to the API documentation, you'll find many more
- I tried to focus on points which would either
  - Give a feel of how many features it all packs, or
  - Would ensure we'd bump into some lesser obvious integration points

Friendly Reminders

- This was really a quick and biased tour
- Go to the API documentation, you'll find many more
- I tried to focus on points which would either
  - Give a feel of how many features it all packs, or
  - Would ensure we'd bump into some lesser obvious integration points

Questions and Answers

- What is the tier and type information attached to a framework?
- Which mechanism do we use for lower tier frameworks to benefit from features of higher tier frameworks?
- Which mechanism do we use to inject behavior in a Qt app which uses a KDE Framework?
- Which type do we use for asynchronous operations?
- If I need network transparent operations which framework do I use?
- If I need to distribute scripted content for my application, which framework do I use?

Questions and Answers

- What is the tier and type information attached to a framework?
- Which mechanism do we use for lower tier frameworks to benefit from features of higher tier frameworks?
- Which mechanism do we use to inject behavior in a Qt app which uses a KDE Framework?
- Which type do we use for asynchronous operations?
- If I need network transparent operations which framework do I use?
- If I need to distribute scripted content for my application, which framework do I use?

Questions and Answers

- What is the tier and type information attached to a framework?
- Which mechanism do we use for lower tier frameworks to benefit from features of higher tier frameworks?
- Which mechanism do we use to inject behavior in a Qt app which uses a KDE Framework?
- Which type do we use for asynchronous operations?
- If I need network transparent operations which framework do I use?
- If I need to distribute scripted content for my application, which framework do I use?

Questions and Answers

- What is the tier and type information attached to a framework?
- Which mechanism do we use for lower tier frameworks to benefit from features of higher tier frameworks?
- Which mechanism do we use to inject behavior in a Qt app which uses a KDE Framework?
- Which type do we use for asynchronous operations?
- If I need network transparent operations which framework do I use?
- If I need to distribute scripted content for my application, which framework do I use?

Questions and Answers

- What is the tier and type information attached to a framework?
- Which mechanism do we use for lower tier frameworks to benefit from features of higher tier frameworks?
- Which mechanism do we use to inject behavior in a Qt app which uses a KDE Framework?
- Which type do we use for asynchronous operations?
- If I need network transparent operations which framework do I use?
- If I need to distribute scripted content for my application, which framework do I use?

Questions and Answers

- What is the tier and type information attached to a framework?
- Which mechanism do we use for lower tier frameworks to benefit from features of higher tier frameworks?
- Which mechanism do we use to inject behavior in a Qt app which uses a KDE Framework?
- Which type do we use for asynchronous operations?
- If I need network transparent operations which framework do I use?
- If I need to distribute scripted content for my application, which framework do I use?

Key Takeaways

- The “tier” of a framework is about its link time dependencies complexity
- The “type” of a framework is a mix of the amount of runtime dependencies and intended role
- There’s pretty much a KDE Framework for anything, for sure all the common tasks needed to build lots of different types of applications are covered
- We have hidden plugins to inject features in frameworks behind the scene
- We use the application object hooks to inject behavior from a framework into an application which simply links to it

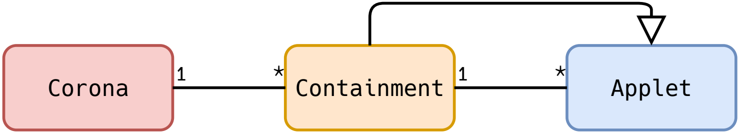
KDE Plasma

Objectives

- Introduce LibPlasma and how we go from there to actually building an environment for the user
- Have an idea of the important components distributed in our workspaces
- Understand how we reinject behavior in all the Qt application from the Plasma environment
- Have a rough idea of how KWin is structured and how it differs between X11 and Wayland
- Know how to extend System Settings

LibPlasma

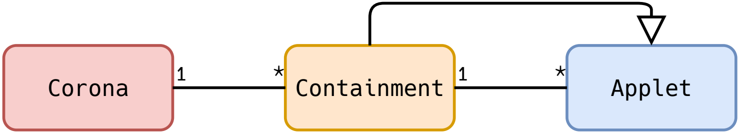
A Component Model for Workspaces



- `Plasma::Corona` represents the whole workspace or "the scene"
  - Defines the basic rules of the workspace including the overall layout
  - Manages the *Containments* in a screen and activity aware fashion
  - This is the one controlling the edit mode being enabled or not
- `Plasma::Containment` represents areas within the corona
  - Defines how its content is laid out
  - Basically can be either desktop or panel
    - There are a couple more types I'll happily ignore here
  - Form factor and activity aware

LibPlasma cont'd

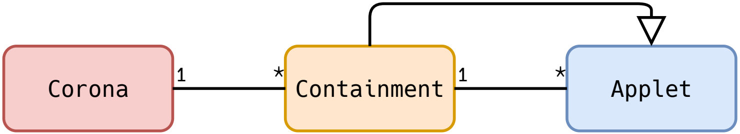
A Component Model for Workspaces



- `Plasma::Applet` represents a “widget” (also sometimes named “plasmoid”) the user can interact with
  - Applets provide the bulk of the behavior and interaction available
  - Form factor aware
- `Plasma::Theme` provides all the theming information to the other classes
  - Color scheme
  - Fonts
  - Where to load images from

LibPlasma cont'd

A Component Model for Workspaces



- They are all loaded using KPackage
  - C++ API is mainly here to develop your shell
  - The packages are all QML based
- The framework also comes with a set of items to use on the QML side
  - Most notably two important modules
    - `org.kde.plasma.core` which among other things provides convenience to load SVGs from a theme (includes caching, coloring, rendering of sub-elements)
    - `org.kde.plasma.components` provides a QtQuickControls2 implementation backed by SVG based primitives
  - Applet authors should use them to ensure a coherent look and feel

The KDE Workspace**s**

- Services necessary to build a workspace
  - Application management
  - Hardware interaction (disks, power, network, bluetooth, thunderbolt)
  - System status
- Lots of configuration modules and how to navigate them (systemsettings)
- Styles covering the whole session (Grub, Plymouth, SDDM, Plasma, QtWidgets, QtQuickControls2, GTK+)
  - They all need to be coordinated!
- Features for the users in term of applets
- A shell and a window manager to tie it all up together
- Integration plugins
  - Remember some of the extension points we've seen before in KDE Frameworks and Qt

- Did any of the above say desktop only?
- So multiply some of that for each form factor we choose to support
  - Mainly shell and containments
  - Thanks to our tech the rest mostly applies as is on all form factors

The KDE Workspaces

- Services necessary to build a workspace
  - Application management
  - Hardware interaction (disks, power, network, bluetooth, thunderbolt)
  - System status
- Lots of configuration modules and how to navigate them (systemsettings)
- Styles covering the whole session (Grub, Plymouth, SDDM, Plasma, QtWidgets, QtQuickControls2, GTK+)
  - They all need to be coordinated!
- Features for the users in term of applets
- A shell and a window manager to tie it all up together
- Integration plugins
  - Remember some of the extension points we've seen before in KDE Frameworks and Qt

- Did any of the above say desktop only?
- So multiply some of that for each form factor we choose to support
  - Mainly shell and containments
  - Thanks to our tech the rest mostly applies as is on all form factors

Qt Platform Abstraction is Back

Remember I mentioned we shipped a QPA plugin early on?

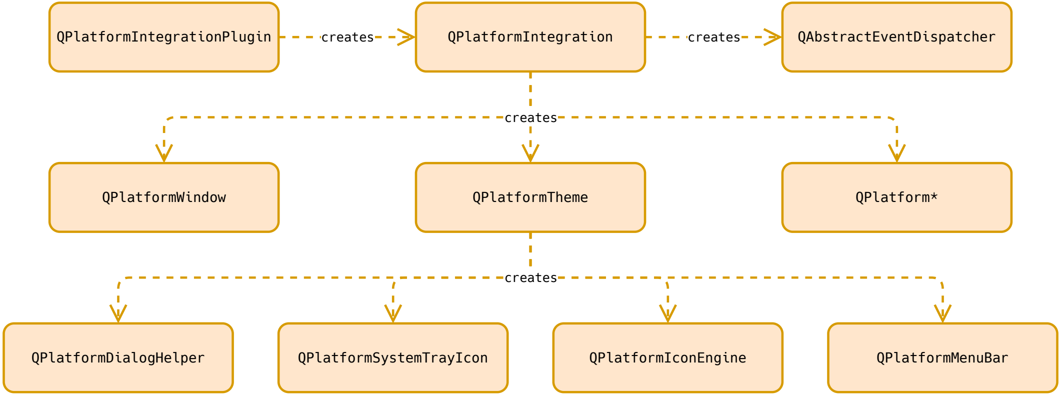
Time to look more into it. . .

Qt Platform Abstraction is Back

Remember I mentioned we shipped a QPA plugin early on?

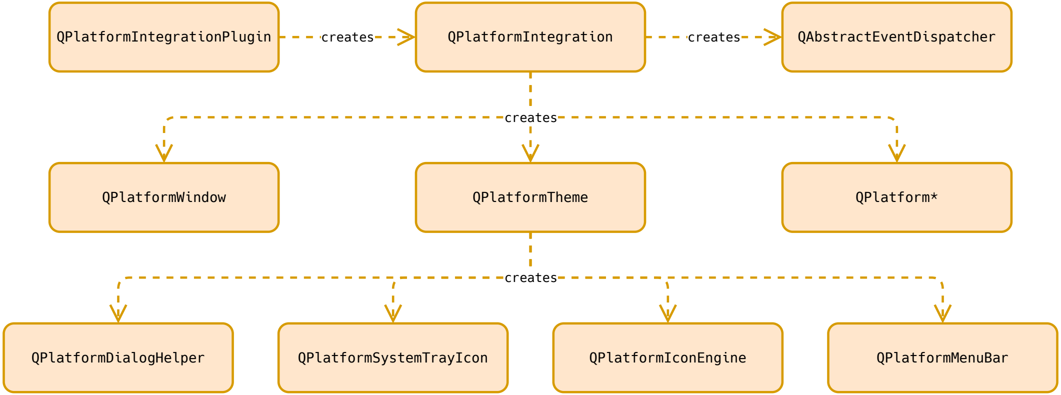
Time to look more into it. . .

QPA Classes, A Reminder



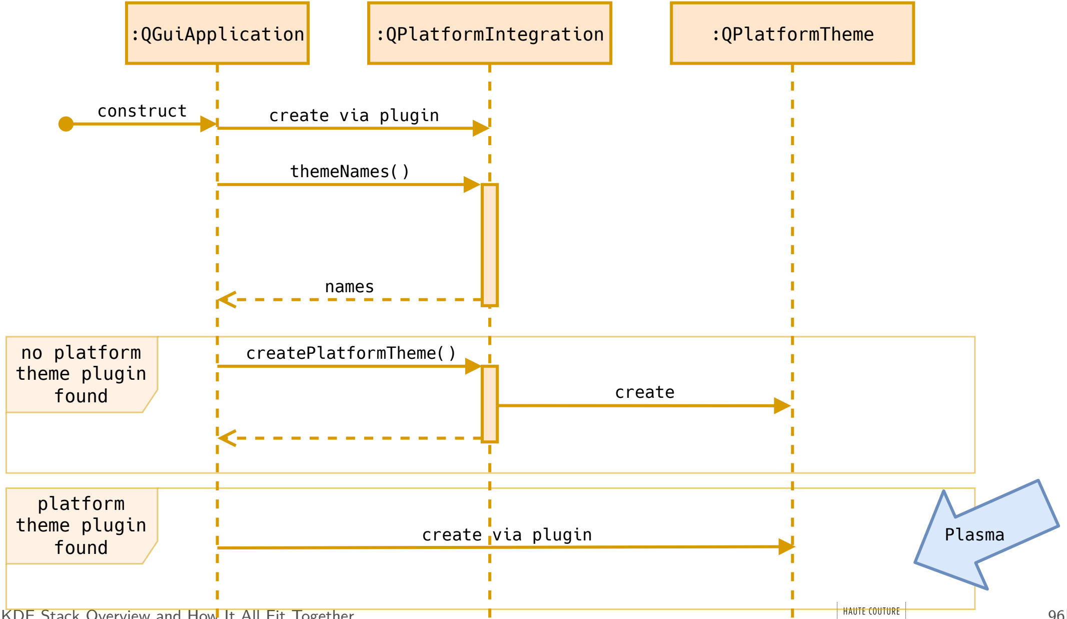
- For sure we don’t want to redo all the work of the X11 or Wayland QPA plugins!
- Indeed we don’t. . . there’s yet another trick

QPA Classes, A Reminder



- For sure we don't want to redo all the work of the X11 or Wayland QPA plugins!
- Indeed we don't... there's yet another trick

QPlatformTheme Creation Revisited



Plasma Integration

- Provides our KDE Platform Theme
- Links against a whole lot of KDE Frameworks
  - KIO
  - KConfig
  - KNotifications
  - KIconThemes
  - and more. . .
- And remember there's quite some magic we do just by linking!
- Features
  - Integrates menu bars with our global menu
  - Integrates system tray icons with `KStatusNotifierItem`
  - Overrides the file dialog with our own implementation
  - Replaces the stock `QIconEngine` with our own `KIconEngine` (respects user theme, provides caching. . .)
  - Overrides default key bindings based on `KStandardShortcut` settings
  - Injects the default palette from the settings
  - Forces our own QtQuickControls2 theme
- Everything you need to make a Qt application look native in Plasma

Plasma Integration

- Provides our KDE Platform Theme
- Links against a whole lot of KDE Frameworks
  - KIO
  - KConfig
  - KNotifications
  - KIconThemes
  - and more. . .
- And remember there's quite some magic we do just by linking!
- Features
  - Integrates menu bars with our global menu
  - Integrates system tray icons with `KStatusNotifierItem`
  - Overrides the file dialog with our own implementation
  - Replaces the stock `QIconEngine` with our own `KIconEngine` (respects user theme, provides caching. . .)
  - Overrides default key bindings based on `KStandardShortcut` settings
  - Injects the default palette from the settings
  - Forces our own QtQuickControls2 theme
- Everything you need to make a Qt application look native in Plasma

Plasma Integration

- Provides our KDE Platform Theme
- Links against a whole lot of KDE Frameworks
  - KIO
  - KConfig
  - KNotifications
  - KIconThemes
  - and more...
- And remember there's quite some magic we do just by linking!
- Features
  - Integrates menu bars with our global menu
  - Integrates system tray icons with `KStatusNotifierItem`
  - Overrides the file dialog with our own implementation
  - Replaces the stock `QIconEngine` with our own `KIconEngine` (respects user theme, provides caching...)
  - Overrides default key bindings based on `KStandardShortcut` settings
  - Injects the default palette from the settings
  - Forces our own QtQuickControls2 theme
- Everything you need to make a Qt application look native in Plasma

Plasma Integration

- Provides our KDE Platform Theme
- Links against a whole lot of KDE Frameworks
  - KIO
  - KConfig
  - KNotifications
  - KIconThemes
  - and more...
- And remember there's quite some magic we do just by linking!
- Features
  - Integrates menu bars with our global menu
  - Integrates system tray icons with `KStatusNotifierItem`
  - Overrides the file dialog with our own implementation
  - Replaces the stock `QIconEngine` with our own `KIconEngine` (respects user theme, provides caching...)
  - Overrides default key bindings based on `KStandardShortcut` settings
  - Injects the default palette from the settings
  - Forces our own QtQuickControls2 theme
- Everything you need to make a Qt application look native in Plasma

Plasma

- This is where we provide implementations for the containments and applets
  - Desktop form factor: desktop and panel containments
  - Phone form factor: homescreen, panel and taskpanel containments
- Plasma Shell ties it all together
  - Comes with its own `Corona` subclass: `ShellCorona`
  - Loads its own package formats
  - Most notably shell packages which control
    - How the user can interact with applets (the chrome to move and resize them)
    - Which GUI is used for applet or containment settings
    - How the widget explorer looks
    - Which default layout will be used on first start
  - We also have layout template packages
    - Allow to prefill containments in the shell
    - e.g. that's how we provide a default panel
  - This is all JS and QML based

Plasma

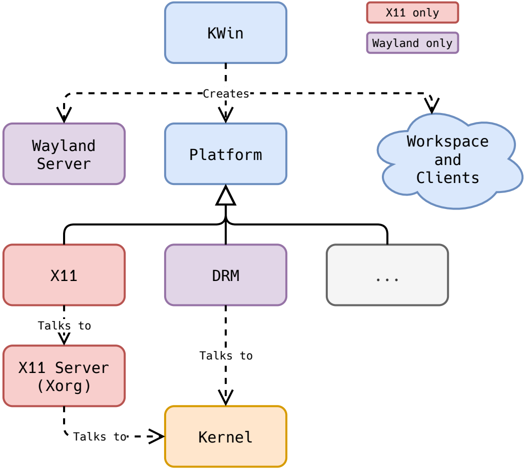
- This is where we provide implementations for the containments and applets
  - Desktop form factor: desktop and panel containments
  - Phone form factor: homescreen, panel and taskpanel containments
- Plasma Shell ties it all together
  - Comes with its own `Corona` subclass: `ShellCorona`
  - Loads its own package formats
  - Most notably shell packages which control
    - How the user can interact with applets (the chrome to move and resize them)
    - Which GUI is used for applet or containment settings
    - How the widget explorer looks
    - Which default layout will be used on first start
  - We also have layout template packages
    - Allow to prefill containments in the shell
    - e.g. that's how we provide a default panel
  - This is all JS and QML based

KWin

- Our window manager and compositor
- Works on top of X11 or Wayland
- Historically was X11 only
  - That still shows a bit in the code
  - Transition is on going

KWin

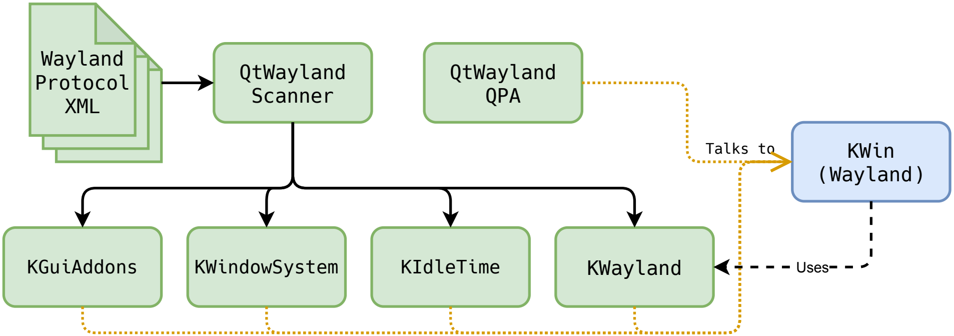
Platform Abstraction at a Glance



- This is a very high level view just to give a rough idea
- Situation is way more complicated in the workspace area
  - Code very much in transition
  - Expect different codepaths to be executed depending on the platform
- There are good talks and documentations about KWin, this is not one of them

KWin

Focus on Wayland



- Protocol files come from wayland-protocols or plasma-wayland-protocols
- Reminder: KWin is a wayland server implementation

System Settings

Where do the configuration GUIs come from?

- Simple recipe
  1. Create the KConfigXT files representing your settings
  2. Slap GUI on top of it, mostly two cases:
  - QtWidgets based: Inherit from `KCModule`
  - QtQuick based: Inherit from `KQuickAddons::ManagedConfigModule`
    - QML based GUI code is bundled as Qt resources
    - Root of the QML script will be a `KCM.SimpleKCM` or `KCM.GridViewKCM`
- Plasma workspaces have only QML based KCMs (more Plasma Mobile friendly)
- It is good form nowadays to also provide a `KCModuleData` from your plugin
  - The whole "Highlight Changed Settings" is built on top of it
  - This will open the door to other features
    - e.g. better settings search

System Settings

Where do the configuration GUIs come from?

- Simple recipe
  1. Create the KConfigXT files representing your settings
  2. Slap GUI on top of it, mostly two cases:
  - QtWidgets based: Inherit from `KCModule`
  - QtQuick based: Inherit from `KQuickAddons::ManagedConfigModule`
    - QML based GUI code is bundled as Qt resources
    - Root of the QML script will be a `KCM.SimpleKCM` or `KCM.GridViewKCM`
- Plasma workspaces have only QML based KCMs (more Plasma Mobile friendly)
- It is good form nowadays to also provide a `KCModuleData` from your plugin
  - The whole “Highlight Changed Settings” is built on top of it
  - This will open the door to other features
    - e.g. better settings search

System Settings

Where do the configuration GUIs come from?

- Simple recipe
  1. Create the KConfigXT files representing your settings
  2. Slap GUI on top of it, mostly two cases:
     - QtWidgets based: Inherit from `KCModule`
     - QtQuick based: Inherit from `KQuickAddons::ManagedConfigModule`
       - QML based GUI code is bundled as Qt resources
       - Root of the QML script will be a `KCM.SimpleKCM` or `KCM.GridViewKCM`
- Plasma workspaces have only QML based KCMs (more Plasma Mobile friendly)
- It is good form nowadays to also provide a `KCModuleData` from your plugin
  - The whole "Highlight Changed Settings" is built on top of it
  - This will open the door to other features
    - e.g. better settings search

Questions and Answers

- Which are the central concepts in LibPlasma?
- What is provided by a Plasma workspace?
- What does Plasma Integration provide?
- How is the integration between a Wayland session and our frameworks done?
- How to extend System Settings?

Questions and Answers

- Which are the central concepts in LibPlasma?
- What is provided by a Plasma workspace?
- What does Plasma Integration provide?
- How is the integration between a Wayland session and our frameworks done?
- How to extend System Settings?

Questions and Answers

- Which are the central concepts in LibPlasma?
- What is provided by a Plasma workspace?
- What does Plasma Integration provide?
- How is the integration between a Wayland session and our frameworks done?
- How to extend System Settings?

Questions and Answers

- Which are the central concepts in LibPlasma?
- What is provided by a Plasma workspace?
- What does Plasma Integration provide?
- How is the integration between a Wayland session and our frameworks done?
- How to extend System Settings?

Questions and Answers

- Which are the central concepts in LibPlasma?
- What is provided by a Plasma workspace?
- What does Plasma Integration provide?
- How is the integration between a Wayland session and our frameworks done?
- How to extend System Settings?

Key Takeaways

- Our workspaces provide essentially
  - System services
  - Configuration modules
  - Styles
  - Plasma applets
  - Integration plugins
  - A shell and a window manager
- This is largely form factor agnostic and this maximizes reusability
- To achieve this we rely quite a bit on scripting and KPackage
- Our KDE Platform Theme for QPA is key to integrate Qt applications in our workspaces
- KWin has its own platform abstraction layer
- KWin is our wayland server implementation
- System Settings API is mostly provided by KDE Frameworks

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs…
- Also we'd have needed **days** to go through the KDE Frameworks section…

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs. . .
- Also we'd have needed **days** to go through the KDE Frameworks section. . .

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs. . .
- Also we'd have needed **days** to go through the KDE Frameworks section. . .

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs...
- Also we'd have needed **days** to go through the KDE Frameworks section...

What have we done today?

- We first took a look back at the Qt/KDE history
- We identified some key technologies which go way back in the past
- We better understood the event loop and Qt Platform Abstraction in our context
- We've seen the typical patterns in QtWidgets and QtQuick applications and how they impact reusability in our stack
- We explained how KDE Frameworks is structured as a product
- We also confirmed there are an almost infinite amount of features available
- We explored some of the shady secrets we use for both have splitted frameworks and a coherent experience when it's used all together
- We listed what our workspaces contain
- We focused on how Plasma ensures Qt applications integrate with the workspace
- We also got a quick look at how KWin is structured, especially on Wayland

- Unfortunately we didn't have labs. . .
- Also we'd have needed **days** to go through the KDE Frameworks section. . .

Thank You!

Questions?

ervin@kde.org

kevin.ottens@enioka.com