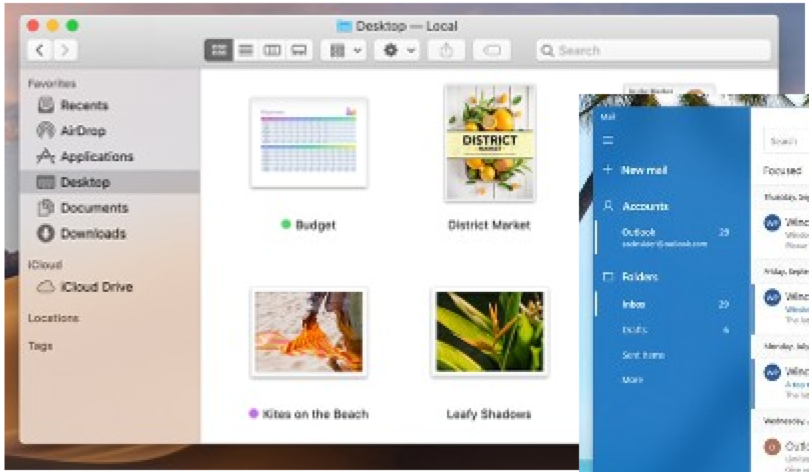
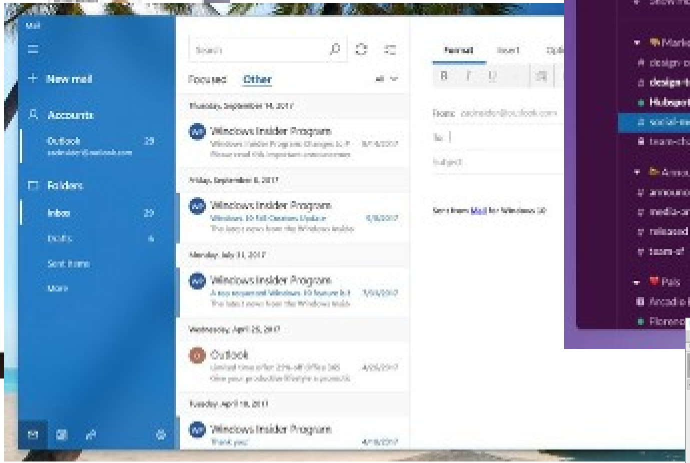
Kirigami

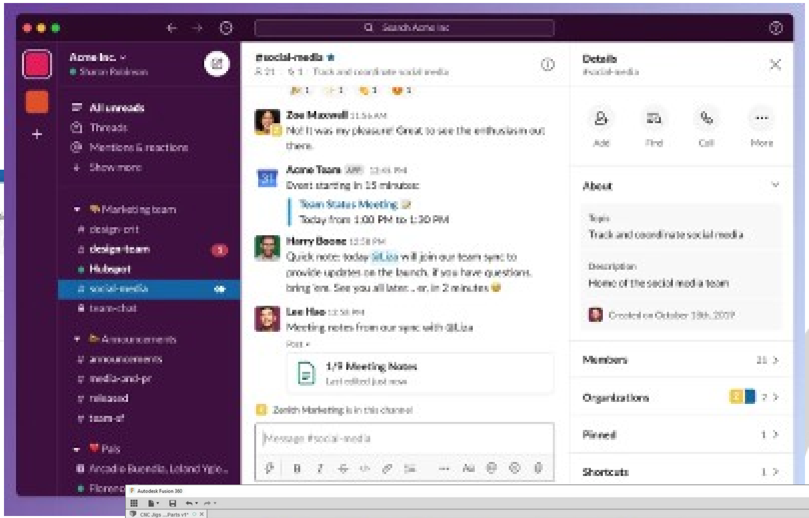
Modern Mobile and Desktop Applications

Evolution of desktop apps design

- Let’s look at some popular apps on different platforms:







Zoe Maxwell: No! It was my pleasure! Great to see the enthusiasm out there.

Acme Team: Event starting in 15 minutes: Team Status Meeting — Today from 1:00 PM to 1:30 PM

Harry Boone: Quick note: today @Liza will join our team sync to provide updates on the launch. If you have questions, bring 'em. See you all later... er, in 2 minutes 😄

Lee Hao: Meeting notes from our sync with @Liza

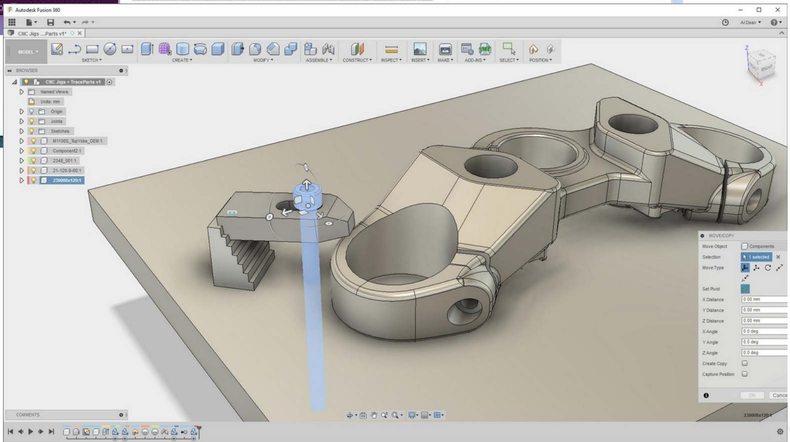
1/9 Meeting Notes — Last edited just now

Zenith Marketing is in this channel

Topic: Track and coordinate social media

Description: Home of the social media team

Created on October 18th, 2017



Evolution of desktop apps design

- Apps that are not content-creation centric (think of a CAD software, an office suite) tend to look simpler, more spaced out, design informed from mobile
- As well (and also informed from mobile) they have a lot of smooth animations, graphical effects that require a widget system that can provide efficient hardware accelerated painting
- MacOS and Windows offer some very “flashy” widget API now, as well more and more apps now are built with Electron

QWidget is awesome

- QWidget evolved through decades of polish, adressing many corner cases, makiing it possible to create an application very powerful.
- Looking through the opensource desktop apps that use Qwidgets you can find many productivity applications that are very powerful and complex (Krita, Freecad, OBS, QtCreator, Kdevelop, Kuesa, KDEnlive etc)
- As well as countless commercial applications

QWidget feels dated

- However, its evolution almost stopped in the Qt5 lifecycle, is considered “done” and works well, but it’s really showing its age.
- All the painting is still done mostly by software, so CPU intensive (same thing for animations)
- Every widget clips its contents (can’t even do a dropshadow without cheating in several ways)
- Animations are possible, but all manually with C++ api.
- It’s kinda “hostile” to designers, need better design tooling and we have to tell them too many times “we can’t really do that”.

QML is awesome

- Draws everything on an hardware accelerated scene graph
- Very easy to do user interfaces that look gorgeous and animate smoothly
- Very intuitive declarative language to create the UI

QML feels incomplete

- Unfortunately wasn’t a smooth evolution of QWidget but starting from a blank slate
- It lacks many things taken for granted especially for desktop apps
- Treeviews for complex models (Views in general are not very good)
- Drag and drop support is still sub par
- C++API is very limited (all primitive elements should be public C++ api, really)
- In QtQuickControls2 all the Popup types are not actual windows, which feels very odd on a desktop app, especially context menus and menubars
- It trades fast drawing and smooth animations for generally slower startup times and bigger memory footprint

Kirigami: on top of QML and QQC2

- Kirigami was born at first as a QML framework for mobile Applications (Plasma Mobile and Android) but since convergence was a goal from day 1 we did put a lot of attention for the desktop as well. It is now a Tier1 KDE framework
- I would recommend it for “light” applications, such as chat apps, file managers, small utilities
- As with QML in general, may be still not “quite enough” for huge content-creation oriented applications.

Kirigami is about consistence

- It gives high level components which make easier to write an application that conforms to a certain design and HIG without too much code
- All components in Kirigami are designed to work on both desktop and mobile, even changing radically their look and behavior if necessary
- It gives also some less “high level” components that are expected, but not in QtQuickControls2 (and maybe outside its scope)

Some missing controls it gives

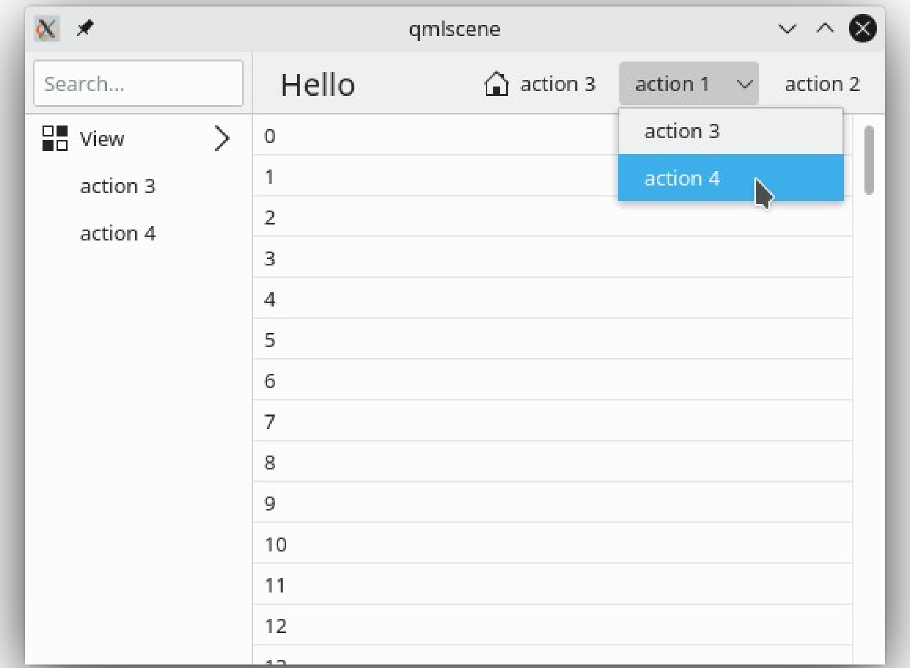
- FormLayout
- Icon
- “Cards” and shadowed/rouded rectangles or images
- Standard About Page
- “action” based toolbars

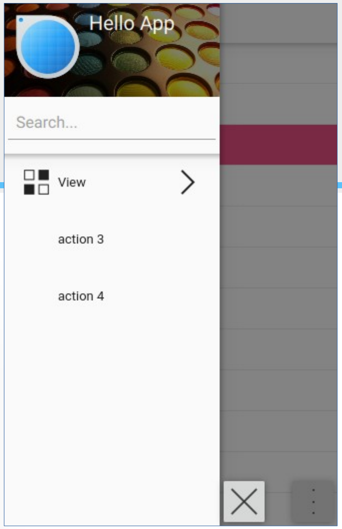
Example code

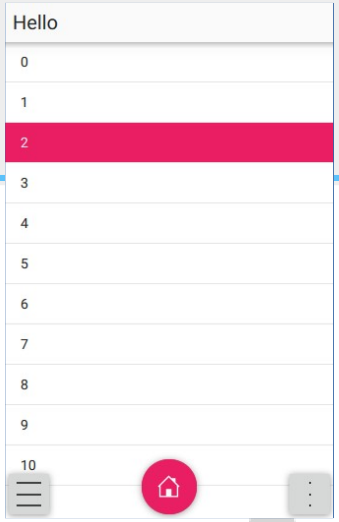
```
Kirigami.ApplicationWindow {
    id: root
    // Swipe-in drawer on mobile, normal navigation sidebar on desktop
    globalDrawer: Kirigami.GlobalDrawer {
        title: "Hello App"
        titleIcon: "applications-graphics"
        actions: [
            Kirigami.Action {
                text: "View"
                iconName: "view-list-icons"
                // Actions can be nested
                Kirigami.Action {
                    text: "action 1"
                }
                Kirigami.Action {
                    text: "action 2"
                }
                Kirigami.Action {
                    text: "action 3"
                }
            },
            Kirigami.Action {
                text: "action 3"
            },
            Kirigami.Action {
                text: "action 4"
            }
        ]
    }
    // Not visible on desktop
    contextDrawer: Kirigami.ContextDrawer {
        id: contextDrawer
    }
...
```

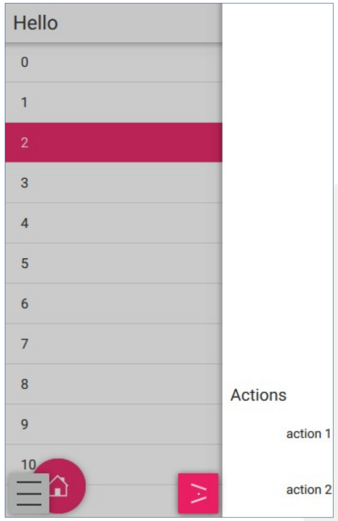
```
...
pageStack.initialPage: mainPageComponent
    Component {
        id: mainPageComponent
        Kirigami.ScrollablePage {
            title: "Hello"
            // Toolbar for those actions automatically generated
            actions {
                main: Kirigami.Action {
                    icon.name: "go-home"
                    text: "action 3"
                }
                contextualActions: [
                    Kirigami.Action {
                        text: "action 1"
                    },
                    Kirigami.Action {
                        text: "action 2"
                    }
                ]
            }
            ListView {
                ...
            }
        }
    }
}
```

Basic app


qmlscene
Search...
Hello
action 3
action 1
action 2
View
action 3
action 4
action 3
action 4
0
1
2
3
4
5
6
7
8
9
10
11
12

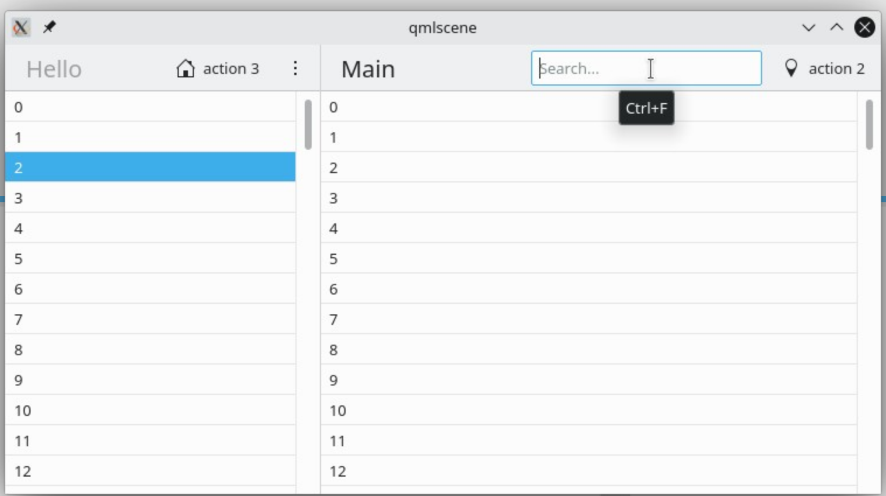

Hello App
Search...
View
action 3
action 4

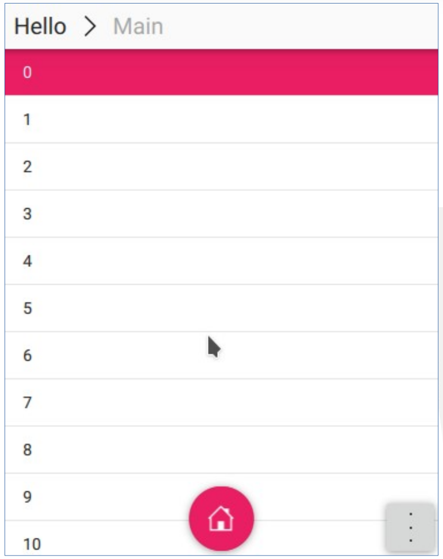

Hello
0
1
2
3
4
5
6
7
8
9
10

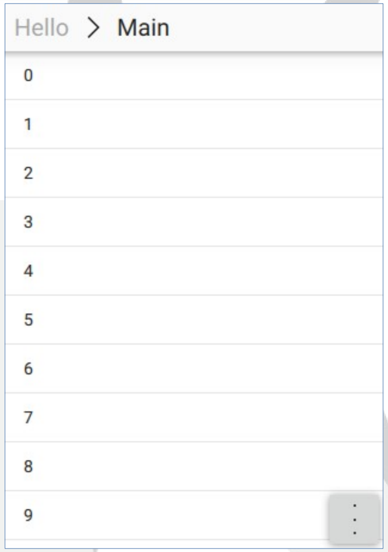

Hello
0
1
2
3
4
5
6
7
8
9
10
Actions
action 1
action 2

Multiple columns

- The base application paradigm in Kirigami.Application is based on drill down of pages
- Of course not mandaroty: Kirigami.ApplicationWindow vs Kirigami.AbstractApplicationWindow
- Property pageRow of ApplicationWindow
- `pageStack.initialPage: [firstPageComponent, secondPageComponent]`
- Can be used anywhere with the ColumnView component (or specialization PageRow oriented to push/pop stack of Pages)

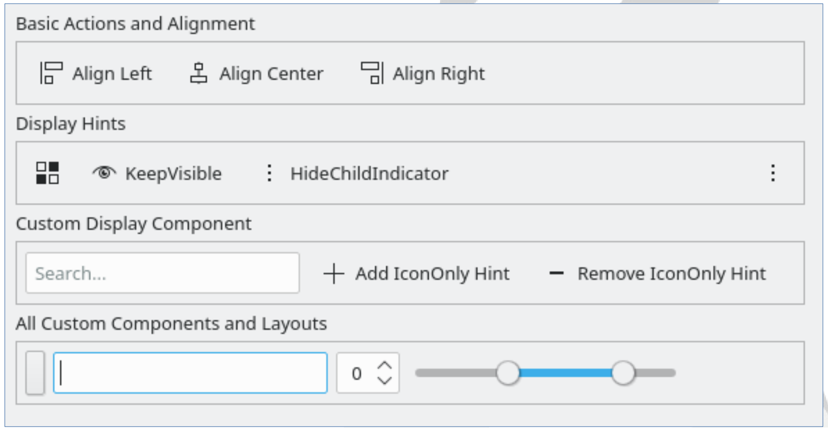






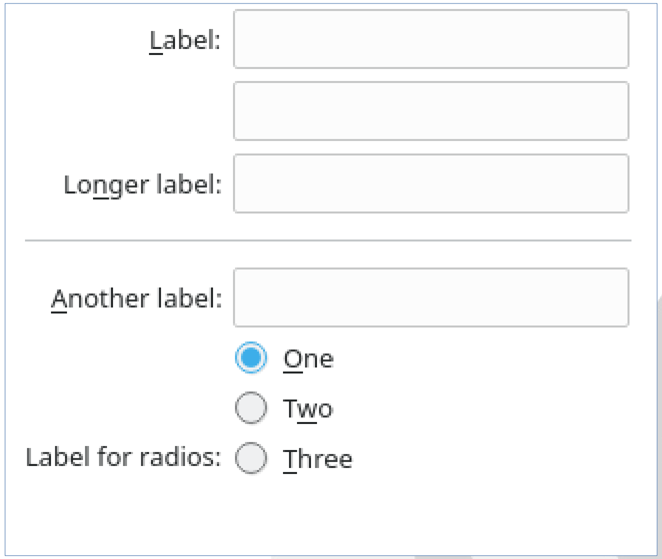
Toolbars: ActionToolbar

- Page contains an ActionToolbar disaplayed only on desktop
- Can be created also standalone
- It’s a list of Actions, the representation is decided by the platform
- Can be overridden via the displayComponent property of Kirigami.Action with an arbitrary component.



FormLayout

```
Kirigami.FormLayout {
    id: layout
    Layout.fillWidth: true
    twinFormLayouts: layout2
    TextField {
        Kirigami.FormData.label: "Label:"
    }
    TextField {
    }
    TextField {
        Kirigami.FormData.label:"Lo&nger label:"
    }
    Kirigami.Separator {
        Kirigami.FormData.isSection: true
    }
    TextField {
        Kirigami.FormData.label: "Another label:"
    }
    ColumnLayout {
        Layout.rowSpan: 3
        Kirigami.FormData.label: "Label for radios:"
        Kirigami.FormData.buddyFor: thirdRadio
        RadioButton {
            id: firstRadio
            checked: true
            text: "One"
        }
        RadioButton {
            text: "Two"
        }
        RadioButton {
            id: thirdRadio
            text: "Three"
        }
    }
}
```

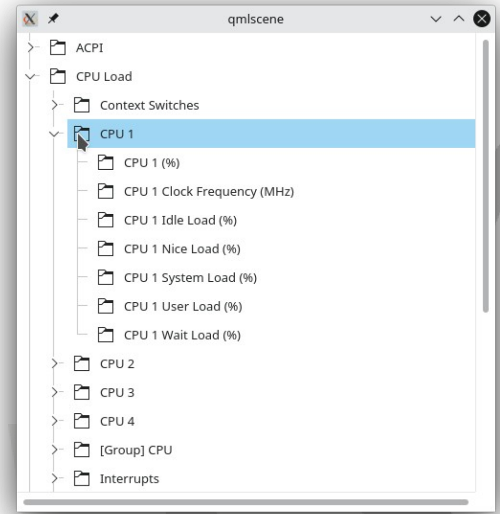


Together with other KDE frameworks

- Many KDE frameworks are starting to get QML bindings, which together can contribute to have a more featureful desktop app
- For instance KItemModels with which gives us models like a qml-binded sort and filter model, KDescendantProxyModel to flat out trees, KConcatenateRowsProxyModel to concatenate multiple models and so on

Example: Tree views

- QML doesn’t have a TreeView
- ListViews inside ListViews would be super inefficient
- “There is no problem that can’t be solved with a sufficient high number of proxy models”
- KDescendantProxyModel is a proxy present in KDE frameworks since a long time: it flattens out tree models to have only one level
- Has been added the possibility to “collapse” nodes (the proxy will emit rowsRemoved instead)
- If the model loads completely collapsed, the model can be lazy loading as intended: a KDirModel can be loaded on “/” and subfolders will be actually listed only when the corresponding node expands

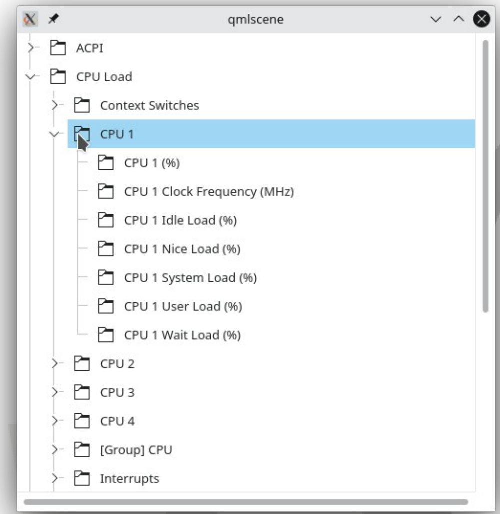


- ACPI
- CPU Load
  - Context Switches
  - CPU 1
    - CPU 1 (%)
    - CPU 1 Clock Frequency (MHz)
    - CPU 1 Idle Load (%)
    - CPU 1 Nice Load (%)
    - CPU 1 System Load (%)
    - CPU 1 User Load (%)
    - CPU 1 Wait Load (%)
  - CPU 2
  - CPU 3
  - CPU 4
  - [Group] CPU
  - Interrupts

Example: Tree views

```
import QtQuick 2.6
import QtQuick.Controls 2.2 as QQC2
import org.kde.kirigami 2.13 as Kirigami
import org.kde.kitemmodels 1.0
import org.kde.ksysguard.sensors 1.0 as Sensors
import org.kde.kirigamiaddons.treeview 1.0 as TreeView

QQC2.ScrollView {
    id: root
    width: 500
    height: 500
    TreeView.TreeListView {
        id: view
        clip: true
        model: Sensors.SensorTreeModel {
            id: allSensorsTreeModel
        }
        delegate: TreeView.BasicTreeItem {
            id: delegate
            label: model.display
            icon: "inode-directory"
        }
    }
}
```



```
qmlscene
ACPI
CPU Load
    Context Switches
    CPU 1
        CPU 1 (%)
        CPU 1 Clock Frequency (MHz)
        CPU 1 Idle Load (%)
        CPU 1 Nice Load (%)
        CPU 1 System Load (%)
        CPU 1 User Load (%)
        CPU 1 Wait Load (%)
    CPU 2
    CPU 3
    CPU 4
    [Group] CPU
    Interrupts
```

Desktop Style of QML/QtQuickControls2

- Outside of Kirigami, qqc2-desktop-style
- It’s a QtQuickControls2 style, with the Qstyle painting code forked from QtQuickcontrols1
- Unfortunately needs Qapplication (there is a widget-less qstyle in Qt6, maybe will be useful)
- Gives a good consistency with Qwidget apps
- Integrates some KcolorScheme support

Unsolved problems and proposals

- Basic components should have a public C++ API
- Popups are not windows:
  - A style could reimplement a QObject with the Popup api, which then would instantiate the contents in own QQuickWindows
  - Can cause incompatibilities on new revisions: Qt5 is frozen, but would need a proper, possibly upstream solution for Qt6
  - Or the style could use upstream Popup, but surreptously reparent the contentitem and background to a Window
  - Need to be careful on Wayland as you can’t position, it would have to be a proper XdgPopup, for relative positioning to its XdgShell parent
  - The menubar behavior is hard to replicate with qml: press on menu item, keep pressed, move mouse on a popup item, release, that item gets triggered

Questions?

https://kde.org/products/kirigami/

Kirigami on Telegram

https://webchat.kde.org/#/room/#kirigami:kde.org