Effective QML: part 2

Adenilson Cavalcanti
BSc. MSc.
KDE developer
WebKit contributor

Second part: QML basics

QML elements: Rectangle

```
import QtQuick 2.0
Rectangle {
  color:"blue"
  width: 100; height: 100
}
```

Yes, slides are QML too...

QML elements: Image

```
import QtQuick 2.0
Image {
    //source: "http://www.google.com/images/srpr/logo3w.png"
    source: "imgs/totem1.png"
}
```

QML elements: Text

```
import QtQuick 2.0
Text {
    text: "Hello world!"
    color: "#333333"
    font { pixelSize: 38 }
}
```

Hello world!

QML elements: Editable Text

```
import QtQuick 2.0
TextInput {
    text: "Editable text"
    color: "#333333"
    font { pixelSize: 38 }
}
```

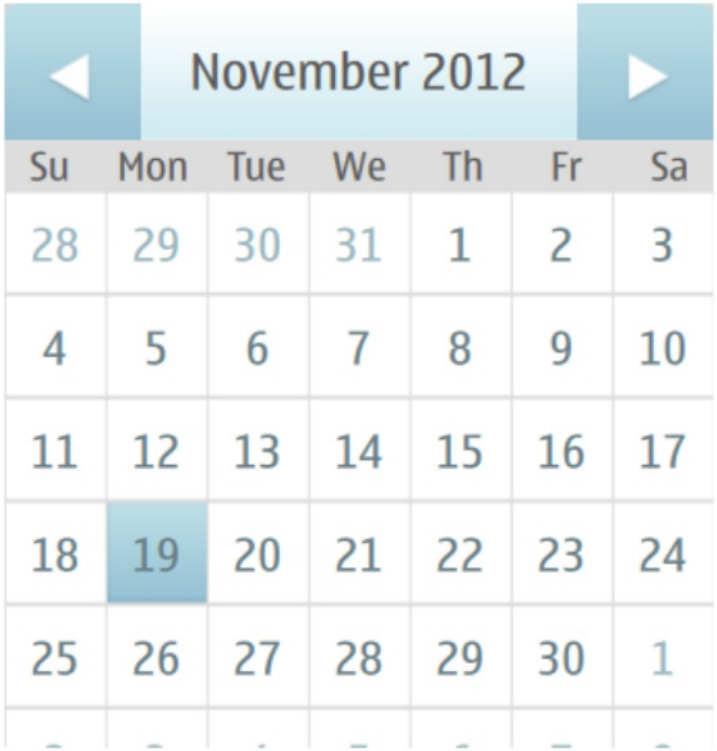
Editable text

QML elements: putting it all together


```
import QtQuick 2.0
Item {
    id: itemParent
    property string current: "imgs/home.png"
    property string background: "imgs/home.png"
    property string pressed: "imgs/home_focused.png"
    property string text: "Home"
    Image {
        id: imgBackground
        source: itemParent.current
        anchors { verticalCenter: parent.verticalCenter;
                horizontalCenter: parent.horizontalCenter }
        MouseArea { anchors.fill: parent
                            onPressed: itemParent.current = itemParent.pressed
                            onClicked: itemParent.current = itemParent.background
        }
        Text { text: itemParent.text; font { pixelSize: 28 }
              anchors { horizontalCenter: parent.horizontalCenter
                        top: parent.bottom }
        }
    }
}
```

QML elements: create your own widgets

```
import QtQuick 2.0
import "cellardoor"
Calendar {
   id: wdgCalendar
}
```



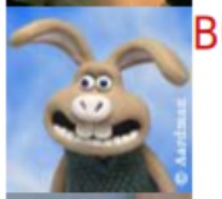
QML elements: The ListView

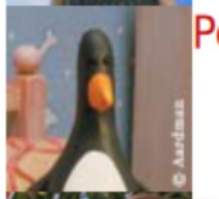
- Kinetic scrolling list
- Model: data (number of elements, properties)
- Delegate: how to paint elements

QML elements: The ListView example

```
import QtQuick 2.0
ListView {
    width: 200; height: 400
    model: Model { }
    delegate: Delegate { }
}
```


Dog 8


Bunny 5


Penguin 8

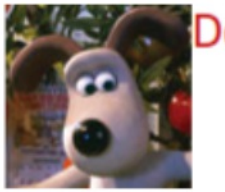

Dog 5


human 8

QML elements: ListView delegate

```
Image {
    id: image
    source: picture

    Text {
        id: text
        text: type + " " + age
        font.family: "Univers LT Std"
        color: "#c81010"
        font.pixelSize: 22
        anchors.left: image.right
    }
}
```


Dog 5

QML elements: ListView model

- Amount of items
- Properties (names, image files, etc)
- Can be exploited...

```
ListModel {
        ListElement { type: "Dog"; age: 8;
   picture: "imgs/preston.jpg" }
        ListElement { type: "Bunny"; age: 5;
   picture: "imgs/hutch.jpg" }
        ListElement { type: "Penguin"; age: 8;
   picture: "imgs/feathers.jpg" }
        ...
}
```


Third part: QML good practices

Tip 1: Coding style

☐ Show correct

```
    Item { //Forget to add new lines
    signal enabled()
    Rectangle {
      id: aStupidIdDontTellTheType
      //Great! Put everything in a single line
      width: 35; height: 35; color: "transparent";
      radius: 5
      border.color: "black"; border.width: 1 }
    }
    property string label: "myCheckbox"
// Lousy indentation
signal disabled()
     //Here is the id (obvious comment)
    id: lookMommyIdAtTheEnd
```


Tip 2: Add qsTr() in all strings

- Mark translatable strings
- Has negligible cost
- Save time later in the project

```
Text {
    id: txtTranslate
    text: qsTr("Hello")
}
```

Tip 3: Run in qmlscene and C++

Instead of qrc://image.png **use** "image.png"

Tip 4: Avoid unnecessary items

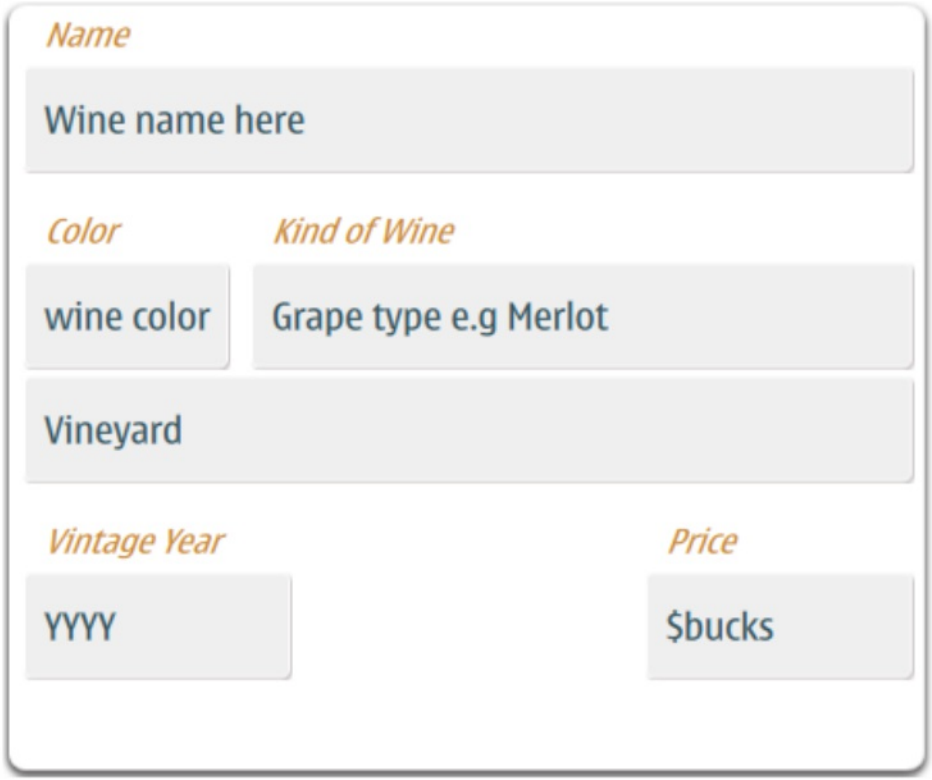
☐ Show correct

```
Component {
        Item {
            height: 100
            Image {
                id: image
                source: picture
            }

            Text {
                id: text
                text: "foo"
            }
        }
}
```


Tip 5: Self-contained Widgets

- Set a minimal size
- Have the dependencies (or stubs)
- Test properties against undefined



Tip 6: Object Oriented Programming

- Use properties to expose configuration
- Keep the software modular
- Create public interfaces

Tip 7: Object Oriented Programming

- Keep widgets encapsulated
- *Don't* access items directly through ID
- Complex apps benefit of a C++ controller

Tip 8: Beware of overuse of ListView

- Several simple items: Repeaters
- 2 or 3 items: reconsider the weight of an associated model

Tip 9: Avoid loading up-front

- longer startup time
- bigger memory usage
- may run into focus issues
- performance (WebKit?)
- Solution: use loaders

Last part: QML/C++ tricks

The generic model

- One model to rule them all...
- QObject introspection
- Templates x QObject
- No pointers!

TDLP: Time Delayed Loading/Painting

- Why? 266Mhz
- Where? Embedded

TDLP: steps

- Turn 'static' elements into Components
- Assign a loader
- Push into the queue
- Timer controls the dispatch

QMLCL: unleash GPU power

- Introduces a new Element
- Write OpenCL kernels in QML
- GPU/CPU/whatever you got

QMLCL: unleash GPU power

```
import qt.labs.opencl 1.0
OCLElement {
    source: "OCL code goes here"
    kernel: "methodName"
    active: false // true schedules execution
    data: [ 1, 2, 3, 4]
    result: // read it to access computed output
    error: // build errors, etc
    onReady: console.log("Output: " + result)
}
```


D-BUS

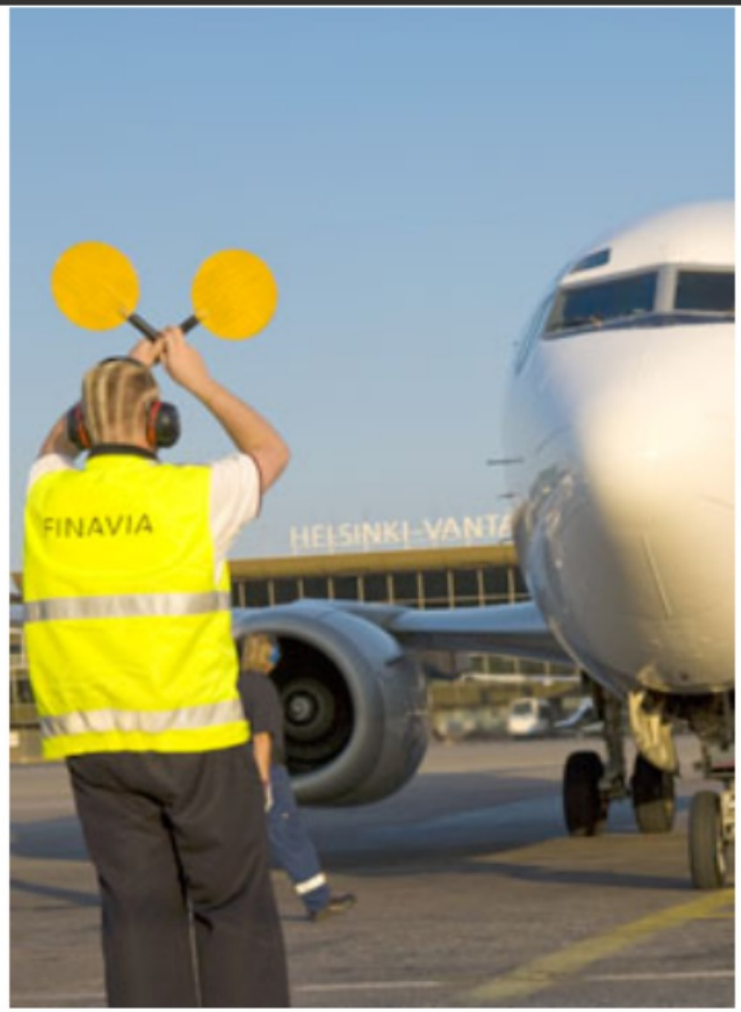
- message based
- IPC/RPC
- requires C++

D-BUS: embedded

- Latency
- CPU usage
- Solution: small batches

D-BUS: the generic marshaller

- QObjects need stream operators
- Boring...
- Solution: introspection + JSON



What is next?

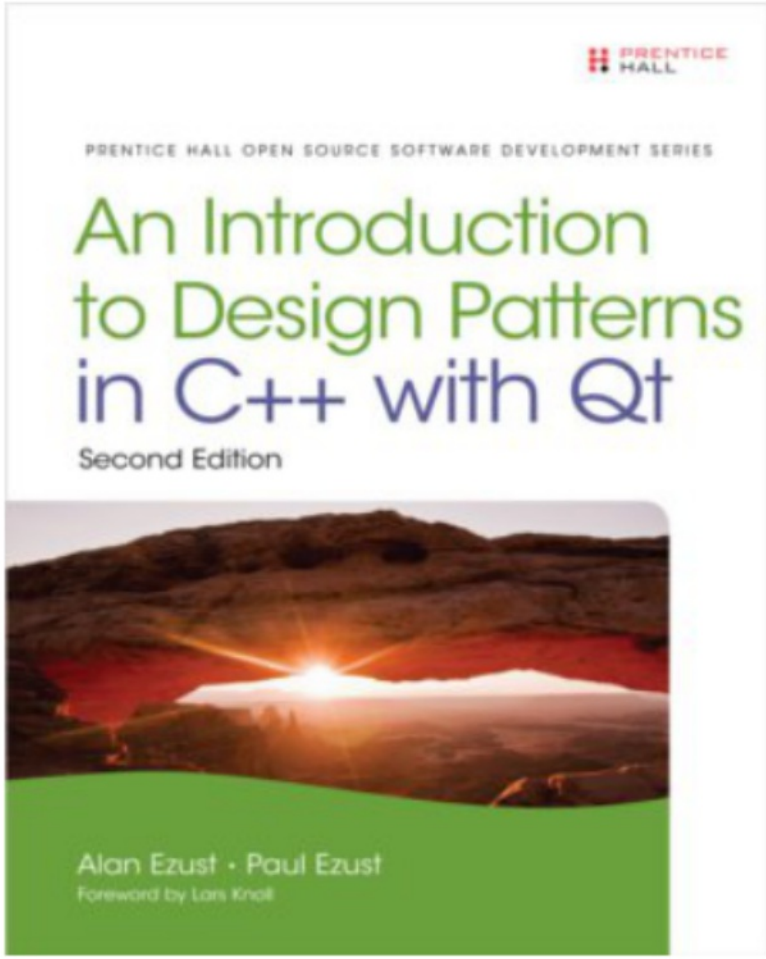
Special thanks

- ICS: sponsoring
- Alan Ezust, Gunnar Sletta, Vladimir M.
- Dev Days organization
- Qt

References

- Qt Design patterns
  Ezust A., Ezust P.,
  Prentice Hall, 2011

- Effective QML
  Cavalcanti A., Ezust A., WIP



Enjoy the ride...

... and use wicked hardware!

Last comments

- Big images x slices x scroll
- Big images: async, sourceSize, smooth
- Slowness? Animations at rescue
- Network data x caching