Two way bindings: Component Design in QtQuick

Qt World Summit, 2019

Presented by André Somers

Qt World Summit 2019 Berlin

The Qt, OpenGL and C++ Experts

Introduction

In a well-designed application:

- The UI is built using re-usable components
- The data and logic live in C++ controllers

The QML part of the application uses these components to build the UI and connects them to the controllers. The controllers provide the data and receive input from the UI.

Demo: Checkbox

We have:

- A controller written in C++
- A Checkbox component we want to hook up
- A main qml file using the Checkbox and a button to reset the controllers state.

Demo: qml-component-design/ex-basic-checkbox

Problems

If you have components that both show a state and allow the user to manipulate that state, how do you design it so that:

1. it has good API,
2. data input gets sent to the controller, and
3. bindings set on its properties don't break?

Further considderations

It gets worse... How do you deal with situations where:

- the backend may reject the change request?
- the backend may be slow to respond to the request?

Non-solutions

Non-solutions

What doesn't work...

What does **not** work:

- Explicitly re-create the binding
- Aliased-in Value
- Model

```
class BooleanValue : public QObject
{
    Q_OBJECT
    Q_PROPERTY(bool isOn READ isOn WRITE setOn NOTIFY isOnChanged)

public:
    explicit BooleanValue(QObject *parent = nullptr);
    explicit BooleanValue(bool initValue, QObject *parent = nullptr);

    bool isOn() const;
    Q_SLOT void setOn(bool isOn); // Be sure to make it a slot or Q_INVOKABLE

    //convenience API is now possible
    Q_INVOKABLE void toggle();

signals:
    void isOnChanged(bool isOn);

private:
    bool m_isOn = false;
};
```

Proposed Value approach

Proposed Value approach

Using a proposed value

Idea: control does not update the main state

- Instead of trying to update the value property, we only *propose* a new value.
- The new proposed value is only set on the control again via the binding on the value property set by the user.

```
CheckBox {
    id: colorCheckbox
    checked: SomeController.isBlue
    onProposedChecked: SomeController.isBlue = proposedChecked;
}
```

- Simple property on controller again
- The component will not change the value property by itself
- Bind as normal at the usage site
- Return connection from explicit "proposed" value
- Proposed value can either be a signal or a property.

Demo: qml-component-design/ex-proposed-value

Using a proposed value (cont'd)

+ Simple

+ Flexible, possible to extend on the side of the component with first showing the proposed state and then reverting if the backend doesn't update

+ Lightweight, no additional objects needed

- Only works on your own controls

- Easy to get wrong by accident

- Replicate handling of unresponsive backend for every control (if needed)

- Different than standard component behavior

Verdict: Quite a good solution

Unbreakable Binding approach

Unbreakable Binding approach

Unbreakable binding approach

Idea: Learn from Qt's own components and avoid breaking the binding.

What if we actually *can* change the value yet keep the binding intact? That is possible if we move the value from a simple property in the QML component to a dedicated C++ component.

```
CheckBox {
    checked: SomeController.isBlue
    onCheckedChanged: SomeController.isBlue = checked
}

Rectangle {
    id: colorIndicator
    color: SomeController.isBlue ? "blue" : "red"
}
```

- Simple property on controller again
- Bind as normal at the usage site
  - **Binding will not break**
- Return connection from value property itself

Demo: qml-component-design/ex-unbreakable-binding

Unbreakable binding approach (cont'd)

- Control internally uses C++ object to keep state
  - Avoids overwriting the property directly
  - Uses Q_INVOKABLE methods or slots on the object instead.

```
import KDAB.Components 1.0

Item {
    id: root

    property alias checked: internal.isOn
    property alias text: label.text

    //ui related code
    Rectangle { ~~~ }

    BooleanValue {
        id: internal
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            internal.toggle(); // works, using convenience function on BooleanValue
            // internal.setOn(!internal.isOn) // works too
            // internal.isOn = !internal.isOn // Wrong: breaks the binding
        }
    }
}
```

Unbreakable binding approach (cont'd)

+ Relatively robust

+ Little usage-side code needed

+ Flexible in the way you setup the return connection

+ Same behavior as most Qt elements

- Slightly confusing how and why this works

- Possible to get two ends of binding out of sync

Verdict: Good solution

Two Way Binding approach

Two Way Binding approach

Two Way Binding approach

Idea: Manage the sync between the properties ourselves

If we use a custom component to manage the sync of the properties between the controller and the component, we can circumvent the issue of the breaking binding by not using one.

```
CheckBox {
    id: colorCheckbox

    TwoWayBinding on checked {
        backendObject: SomeController
        backendProperty: "isBlue"
    }
}
```

- Simple property on controller again
- Simple property on the component again
- At usage site, use TwoWayBinding element instead of a normal QML binding

Demo: qml-component-design/ex-two-way-binding

Two Way Binding approach (cont'd)

The TwoWayBinding element:

- Separate element that keeps two objects in sync
- Written in C++ as any other custom element
- Basicly simply using two signal-slot connections
- The *on property* syntax support is a bit of syntactic sugar

No binding in the QML sense to break.

Two Way Binding approach (cont'd)

+ Explicit in expressing intent

+ No changes needed to controls, works on QML native elements

+ No adaptations to controller needed, works on normal properties

+ Extensible with policies

+ Hard to get wrong, easy to get right

- Burden of creating the connection at use site, so a bit bloaty

- 2-part and string-based API to identify a property on an object is not ideal.
It is used in QML itself too though (i.e. Binding).

- Limitations apply, like no support for binding to an expression (yet)

Verdict: Good solution

Conclusions

Conclusions

Conclusions

- Explicitly re-create the binding

- Aliased-in Value

- Model

+ Proposed Value

+ Unbreakable Binding

+ Two-way Binding

Questions?

Thank you for your time!

Contact us:

- **http://www.kdab.com**
- **KDToolbox: http://www.github.com/kdab/kdtoolbox/**
- **info@kdab.com**
- **training@kdab.com**
- **andre.somers@kdab.com**