
Community Driven Coding
Guidelines for QML

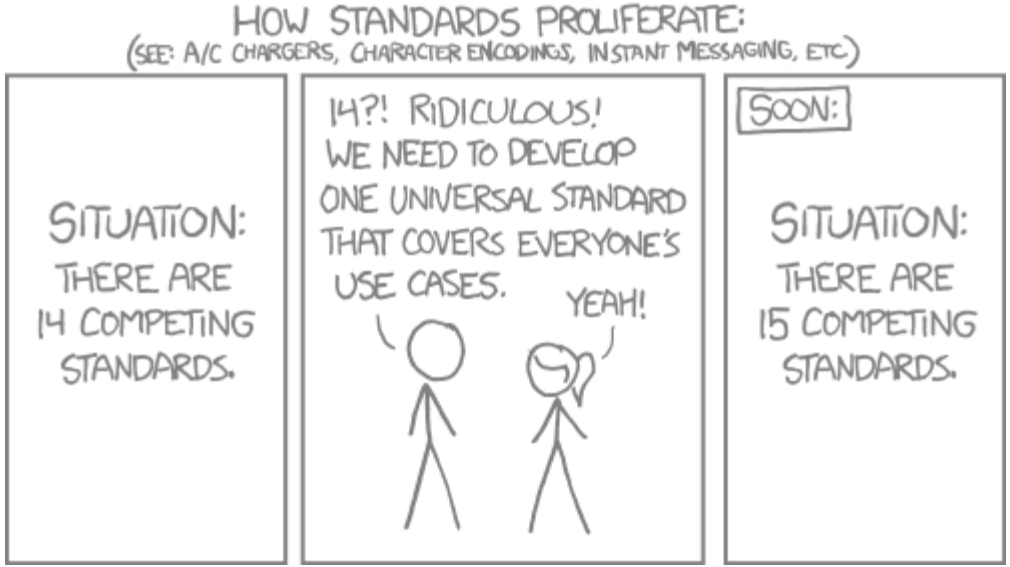
Furkan Uzumcu

- Started using QML in 2017
- ❤ Open Source
- ❤ [Split] Mechanical Keyboards
- Working at Autodesk
- Furkanzmc @ GitHub
- Furkanzmc @ Twitter
- Website: https://zmc.space

Why have guidelines?

- Pool experience from vastly different industries
- Sane defaults for new comers to the language
- Base for tooling

Yet another guideline?..



- Official Best Practices for QML and Qt Quick from Qt
- QML Best Practices Search

Guidelines are NOT…

- Unchangeable
- The absolute truth
- Be all end all

Guidelines are for…

- Making code easier to:
  - read
  - understand
  - maintain
- Making it easier to detect errors or pitfalls
- Low barrier knowledge for new comers
- Aid in tooling

How to Read QML Code

```
// CircleMouseArea.qml
Item {
    id: root

    property int radius

    signal clicked(MouseEvent mouse)

    implicitWidth: 50
    implicitHeight: 50

    Rectangle {
        width: parent.width / 2
        height: height
        radius: root.radius
    }

    MouseArea {
        // If mouse click is inside the circle, emits root.clicked(mouse)
    }
}
```

Code Style

Code Order

- id
- Property declarations
- Signal declarations
- Property initializations
- Attached properties and signal handlers
- States
- Transitions
- Signal handlers
- Child objects
  - Visual Items
  - Qt provided non-visual items
  - Custom non-visual items
- `QtObject` for encapsulating private members[1]
- JavaScript functions

Bad - ID, Properties, and Signals

```
// CustomMouseArea.qml
MouseArea {
    Component.onCompleted: { }
    onTripleClicked: { }
    onClicked: { }
    pressAndHoldInterval: 20

    signal tripleClicked()
    property point pressedPosition

    id: root
}
```

Good - ID, Properties, and Signals

```
// CustomMouseArea.qml
MouseArea {
    id: root

    property point pressedPosition

    signal tripleClicked()

    pressAndHoldInterval: 20
    onClicked: { }
    onTripleClicked: { }
    Component.onCompleted: { }
    Component.onDestruction: { }
}
```

Bad - ID, Properties, and Signals

```
Item {
    Item {
        anchors.left: parent.left
        z: 32
        x: 23
        y: 32
        implicitWidth: 300
        width: 300
        id: myItem
    }
}
```

Good - ID, Properties, and Signals

```
Item {
    Item {
        id: myItem
        x: 23
        y: 32
        z: 32
        implicitWidth: 300
        width: 300
        anchors.left: parent.left
    }
}
```

Bad - Function Ordering

```
Item {
    id: root

    function someFunction() { }

    someProperty: true
}
```

Good - Function Ordering

```
Item {
    id: root
    someProperty: true

    // Function are declared at the bottom of the document.
    function someFunction() { }
}
```

Bad - States and Transitions

```
RowLayout {

    Item {
        states: [ State { } ]
        transitions: [ Transitions { } ]
        width: 300
        Layout.fillHeight: true
        enabled: true
        layer.enabled: false
    }
}
```

Good - States and Transitions

```
RowLayout {

    Item {
        width: 300
        enabled: true
        layer.enabled: false
        Layout.fillHeight: true
        states: [ State { } ]
        transitions: [ Transitions { } ]
    }
}
```

Full Example

```
// First Qt imports
import QtQuick 2.15
import QtQuick.Controls 2.15
// Then custom imports
import my.library 1.0

Item {
    id: root

    // ----- Property Declarations

    // Required properties should be at the top.
    required property int radius: 0

    property int radius: 0
    property color borderColor: "blue"

    // ----- Signal declarations

    signal clicked()
    signal doubleClicked()

    // ----- In this section, we group the size and position information together.
```

Bindings

Declarative > Imperative

Imperative

```
ListView {
    model: ContactModel { }
    delegate: Label {
        id: dlg

        required property int index
        required property string name

        // Or onIndexChanged? onNameChanged?
        Component.onCompleted: {
            text = index + ". " + name
            rect.visible = dlg.index % 2
        }

        Rectangle {
            id: rect
        }
    }
}
```

Declarative

```
ListView {
    model: ContactModel { }
    delegate: Label {
        id: dlg

        required property int index
        required property string name

        text: index + ". " + name

        Rectangle {
            visible: dlg.index % 2
        }
    }
}
```

Declarative 2

Is this still declarative?

```
ListView {
    model: ContactModel { }
    delegate: Label {
        required property int index
        required property string name

        text: getText()

        function getText(): string {
            return index + ". " + name
        }
    }
}
```

Unnecessary Evaluations -

```
import QtQuick 2.3

Item {
    id: root

    property int accumulatedValue: 0

    Component.onCompleted: {
        const someData = [1, 2, 3, 4, 5, 20]
        for (let i = 0; i < someData.length; ++i) {
            accumulatedValue = accumulatedValue + someData[i]
        }
    }

    Text {
        text: root.accumulatedValue.toString()
        onTextChanged: console.log("text binding re-evaluated
    }
}
```

```
import QtQuick 2.3

Item {
    id: root

    property int accumulatedValue: 0

    Component.onCompleted: {
        const someData = [1, 2, 3, 4, 5, 20]
        let temp = accumulatedValue
        for (let i = 0; i < someData.length; ++i) {
            temp = temp + someData[i]
        }

        accumulatedValue = temp
    }

    Text {
        text: root.accumulatedValue.toString()
        onTextChanged: console.log("text binding re-evaluated
    }
}
```

```
void MyRectangle::setRadius(int r)
{
    m_radius = r;
    emit radiusChanged();
}
```

C++ Integration

1. Context properties [Deprecated]
2. Global object
3. Singletons
4. Instantiated object

Context Properties Are Deprecated

Don’t use them.

QTBUG-73064

```
Item {
    id: root

    property int borderWidth

    Rectangle {
        // No no...
        border.width: borderWidth
    }
}
```

Singletons for API Access

```
Window {
    onClosing: (event) => {
        event.accepted = MySingletonClass.confirmExit()
    }

    Button {
        background: Rectangle {
            color: Theme.buttonBackground
        }
    }
}
```

Singletons for API Access (Continued)

- If you use singleton for data, don’t use it inside a component.

```
// Contacts.qml
Item {
    id: root

    ListView {
        model: MySingletonClass.contacts
        delegate: Text { /* ... */ }
    }
}
```

Prefer Instantiated Types Over Singletons For Data

```
// ColorsWindow.qml
Window {
    id: root

    Column {
        Repeater {
            model: Palette.selectedColors
            delegate: ColorViewer {
                required property color color
                required property string colorName

                selectedColor: color
                selectedColorName: colorName
            }
        }
    }
}
```

Prefer Instantiated Types Over Singletons For Data

See Issue #2 for related discussions.

```
// ColorsWindow.qml
Window {
    property alias model: rp.model

    Column {
        Repeater {
            id: rp
            model: PaletteColorsModel { } // Alternatively
            delegate: ColorViewer {
                required property color color
                required property string colorName

                selectedColor: color
                selectedColorName: colorName
            }
        }
    }
}
```

Watch Out for Object Ownership Rules

Two ownership types:

- C++
- QML
- Expose data with properties only

> See example for using properties for data customization

> See this article for a real life example of a related bug in an application.

```
Q_PROPERTY( QObject* colors READ colors )

QObject* colors(); // Ownership remains in C++.
Q_INVOKABLE QObject* myData(); // Ownership is transferred to QML.
```

Memory

Profile first! Needs contributions.

Signals

Signals != Functions

Functions vs Signals

Function -> Changes Internal State

```
Imperative form -> doSomething()
```

Signal -> Announces Internal State Change

```
Passive form -> somethingChanged()
```

Avoid Using `.connect()` in QML

```
ApplicationWindow {
    id: root

    property list<QtObject> myObjects: [
        QtObject { signal somethingHappened() }, QtObject { signal somethingHappened() },
        QtObject { signal somethingHappened() }, QtObject { signal somethingHappened() },
        QtObject { signal somethingHappened() }, QtObject { signal somethingHappened() },
        QtObject { signal somethingHappened() }, QtObject { signal somethingHappened() }
    ]

    ListView {
        cacheBuffer: 1 // Low enough we can resize the window to destroy buttons.
        model: root.myObjects.length
        delegate: Button {
            text: "Button " + index
            onClicked: root.myObjects[index].somethingHappened()
            Component.onCompleted: root.myObjects[index].somethingHappened.connect(() => console.log(text))
            Component.onDestruction: console.log("Destroyed #", index)
        }
    }
}
```

Avoid Using `.connect()` in QML (Continued)

```
ApplicationWindow {
    id: root

    property list<QtObject> myObjects: [
        // ... Same model as previous code snippet
    ]

    ListView {
        cacheBuffer: 1 // Low enough we can resize the window to destroy buttons.
        model: root.myObjects.length
        delegate: Button {
            id: dlg
            text: "Button " + index
            onClicked: root.myObjects[index].somethingHappened()
            Component.onDestruction: console.log("Destroyed #", index)

            Connections {
                target: root.myObjects[index]

                function onSomethingHappened() {
                    console.log(dlg.text)
                }
            }
```

What is wrong here?

```
// ColorPicker.qml
Rectangle {
    id: root

    signal colorPicked(color pickedColor)

    ColorDialog {
        onColorChanged: {
            root.colorPicked(color)
        }
    }
}

// main.qml
Window {
    ColorPicker {
        onColorPicked: {
            color = pickedColor
            label.text = "Color Changed"
        }
    }
    Label { id: label }
}
```

```
// ColorPicker.qml
Rectangle {
    id: root

    signal colorPicked(color pickedColor)

    ColorDialog {
        onColorChanged: {
            root.color = color
            root.colorPicked(color)
        }
    }
}

// main.qml
Window {
    ColorPicker {
        onColorPicked: {
            label.text = "Color Changed"
        }
    }
    Label { id: label }
}
```

Rule of thumb:

```
When communicating up, use signals.
When communicating down, use functions.
```

States and Transitions

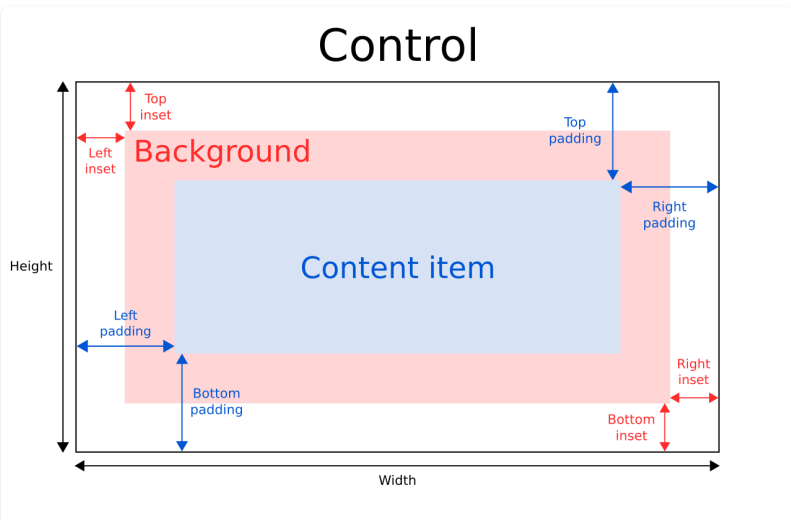
Don’t Define Top Level States

```
Item {
    component Rect: Rectangle {
        id: self

        readonly property alias pressed: ma.containsPress

        states: State {
            when: ma.containsMouse

            PropertyChanges { target: self; color: "red" }
        }

        MouseArea { id: ma; hoverEnabled: true }
    }

    Rect {
        id: rect
        states: State {
            when: rect.pressed

            PropertyChanges { target: rect; color: "yellow" }
        }
    }
```

Visual Items

One Size Does Not Fit All

- **Implicit Size**: Space occupied when no explicit size or anchors are set.
- **Explicit Size**: Space occupied when an external size is given, ie `width/height` or `anchors` are set.
- **Content Size**: Space occupied by the contents of a view.
- **Padding**: Space between the content item and the edge of a component.
- **Margin**: Space between two controls.
- **Inset**: Space between background and the edge of a component.



```qml
// CheckBox.qml
Item {
    id: root

    property bool checked
    property string text

    Rectangle {
        id: indicator
        width: 50
        height: 50
        visible: root.checked
        color: "red"
    }

    Label {
        anchors {
            left: indicator.right
            verticalCenter: indicator.verticalCenter
        }
        text: root.text
    }
}
```

```qml
// main.qml
Window {
    CheckBox {
        checked: true
        text: "CheckBox"
    }

    Column {
        width: 100

        Repeater {
            model: 5
            delegate: CheckBox {
                required property int index

                width: parent.width
                checked: index % 2 === 0
            }
        }
    }
}
```

```qml
// CheckBox.qml
Item {
    id: root

    property bool checked
    property string text

    implicitWidth: indicator.implicitWidth + label.implicitWi
    implicitHeight: Math.max(indicator.implicitHeight, label.

    Rectangle {
        id: indicator
        width: height
        height: parent.height * 0.5
        implicitWidth: 50
        implicitHeight: 50
        visible: root.checked
        color: "red"
    }

    Label {
        id: label
        anchors {
            left: indicator.right
            verticalCenter: indicator.verticalCenter
        }
        text: root.text
```

```qml
// main.qml
Window {
    CheckBox {
        checked: true
        text: "CheckBox"
    }

    Column {
        width: 100

        Repeater {
            model: 5
            delegate: CheckBox {
                required property int index

                checked: index % 2 === 0
            }
        }
    }
}
```

> … our intellectual powers are rather geared to master **static relations** and that our powers to visualize **processes evolving in time** are relatively poorly developed. - Edsger W. Dijkstra, Go To Statement Considered Harmful

JavaScript

```
// Arrow function
root.value = Qt.binding(() => root.someOtherValue)
// The old way.
root.value = Qt.binding(function() { return root.someOtherValue })

// Variables
const value = 32;
let valueTwo = 42;
{
    // Valid assignment since we are in a different scope.
    const value = 32;
    let valueTwo = 42;
}
const value = 32;
value = 42; // ERROR!
```

Use arrow function syntax for signal handlers

```
MouseArea {
    // Good!
    onClicked: (mouse) => {

    }
    // Bad...
    onClicked: {

    }
}
```

What’s next?

- OUt with the old, in with the new
- More content for Qt 6
- An architecture section
- More contributions from others

> QML Coding Guidelines on GitHub - https://github.com/Furkanzmc/QML-Coding-Guide

Thank You