QML in Qt6

Ulf Hermann

Sept 7th, 2024

Features added in Qt6

- Type Annotations
- Type Assertions
- Generalized Group Properties
- Loose Coupling
- Named Value Types
- Unified List Types
- Unified Model Data

Type Annotations

- **Hint** to tooling since 5.14
- Consistently **Enforced** since 6.7
  - Cause type coercion on mismatch
- Help **people** grasp your intention
- Help **tooling** grasp your intention
- Use **object** types or **value** types
- … but **not** JavaScript types
- **Compile to C++** via qmlcachegen
- Use **var** to escape

```qml
Item {
    function add(a: int, b: int) : int {
        return a + b
    }

    function getName(o: QtObject) : string {
        return o.objectName
    }

    function atI(l: list<font>, i: int) : font {
        return l[i]
    }

    function badAdd(a: var, b: var) : double {
        return a + b // coerces only on return
    }
}
```

QML

Type Assertions

- **Hint** to tooling since 5.15
- **Checks object** types since 6.2
- **Coerces value** types since 6.6
  - which is horribly **wrong**
  - and **doesn't work** most of the time
  - **luckily**
- pragma ValueTypeBehavior: Assertable
  - **since 6.8**
  - makes 'as' **check** value types

```qml
// Avoid surprises
pragma ValueTypeBehavior: Assertable

import QtQuick as QQ

QQ.Item {
    function getFamily(font: var) : string {
        // If not a font, 'as' produces undefined.
        // Result is a JS exception.
        return (font as QQ.font).family
    }

    function getX(o: QQ.QtObject) : double {
        // If not an Item, 'as' produces null.
        // Result is a JS exception.
        return (o as QQ.Item).x
    }
}
```

QML

Generalized Group Properties

- ID as first part of group property
- Available in **Binding**, **PropertyChanges**
- Simplifies syntax
- Avoids QQmlCustomParser
  - Makes bindings available to tooling
  - Allows compilation to C++
- TODO: Connections, AnchorChanges, ...

```qml
Item {
    id: root
    states: [
        State {
            name: "small"
            when: root.width < 100

            // Old: custom parser
            PropertyChanges {
                target: root
                opacity: 0.8
            }

            // New: generalized group property
            PropertyChanges {
                root.opacity: 0.5
            }
        }
    ]
}
```

QML

Loose Coupling

- QML member is bound by name
- Available in **Connections**
- Avoids QQmlCustomParser
  - Makes functions available to tooling
  - Allows compilation to C++
- TODO: Binding, PropertyChanges ...

```qml
Item {
    id: root

    // Old style, with custom parser
    Connections {
        target: root
        onXChanged: console.log("x changed")
    }

    // New style, with loose coupling
    Connections {
        target: root
        function onYChanged() {
            console.log("y changed")
        }
    }
}
```

QML

Named value types: Basics (since 6.4)

```cpp
class WrappedInt
{
    // Q_GADGETs are value types, Q_OBJECTs not
    Q_GADGET

    // Functionally like QML_NAMED_ELEMENT,
    // but clarifies it's not an actual element.
    QML_VALUE_TYPE(wrappedInt)

    // Any properties, signals,
    // Q_INVOKABLEs are exposed to QML
    Q_PROPERTY(int i MEMBER m_i FINAL)

private:
    int m_i = 0;
};
```

C++

```qml
Item {

    // Use the new value type like you'd use
    // color, font, rect, etc
    property wrappedInt ii

    x: ii.i / 4
}
```

QML

Named value types: Constructible (since 6.5)

```cpp
class WrappedInt
{
    Q_GADGET
    QML_VALUE_TYPE(wrappedInt)
    Q_PROPERTY(int i MEMBER m_i FINAL)

    // Any one-argument Q_INVOKABLE constructor
    // is automatically used to create instances.
    QML_CONSTRUCTIBLE_VALUE

public:
    Q_INVOKABLE WrappedInt(int i = 0) : m_i(i) {}

private:
    int m_i = 0;
};
```

C++

```qml
Item {

    // We can create wrappedInt from a number.
    property wrappedInt ii: 77 + 12

    x: ii.i / 4
}
```

QML

Named value types: Structured (since 6.5)

```cpp
class WrappedInt
{
    Q_GADGET
    QML_VALUE_TYPE(wrappedInt)
    Q_PROPERTY(int i MEMBER m_i FINAL)

    // Any one-argument Q_INVOKABLE constructor
    // is automatically used to create instances.
    // _And_ instances can be populated
    // property-by-property from any object-like.
    QML_STRUCTURED_VALUE

private:
    int m_i = 0;
};
```

C++

```qml
Item {

    // We can create wrappedInt from an object.
    // Parentheses disambiguate objects from
    // code blocks.
    property wrappedInt ii: ({i : 77 + 12})

    x: ii.i / 4
}
```

QML

Unification of Container Types

Containers used to be pretty inconsistent

- JS Array: Fully implemented
- list<ObjectType>: Some methods missing
- lists of value types:
  - Some methods missing
  - only for specific types
  - only anonymous
- Special casing for QStringList, QVariantList, QObjectList

Good news everybody

- They all support all the methods of JS Array now! (since 6.5)
  - Many methods compile to C++ (except on JS Array!)
- Most special casing is gone:
  - QQmlListProperty<Type> for all object types
    - auto-registered as list<ObjectType>, as before
  - QList<Type> for all value types
    - auto-registered as list<valueType> (since 6.4)
    - string and var are just regular value types (since 6.7)
  - You can register more (anonymous) containers (since 6.0)
    - Use QML_SEQUENTIAL_CONTAINER(Type)

Unification of Model Data

- Any list-like works as model!
  - since 6.6
  - lists of value type instances
  - lists of object type instances
  - JS arrays of JS objects
- **coerces** when instantiating delegates
- **optional** values possible

```qml
Column {
    id: root
    property list<size> model: [
        {width: 10, height: 25},
        {width: 99, height: 16}
    ]

    Repeater {
        model: root.model
        delegate: Text {
            // Require specific value:
            required property double width

            // modelData of structured value type:
            // * width and height are populated
            // * x and y remain defaulted
            required property rect modelData
        }
    }
}
```

QML

Future work

- Function types and sum types
- Fix inefficiencies around value type and list references
- Finish generalized group properties and loose coupling
- Add bidirectional bindings while at it
- Allow QML-defined value types
- Deal with context properties (How?)
- Typed writing into models from delegates (How?)
- Fix lookup hierarchy (How?)