QProperty Review

Andreas Buhr

January 12, 2021

Agenda

- Intro by Andreas
  - Some words about naming
  - How dependency tracking works
  - Problems encountered while porting to bindable properties
  - Proposals for improvement
- Comments by QtCore property porting team (Ivan, Sona, Timur, Eddy)
- Discussion how to proceed

1 Naming Things

Proposal: talking about upstream/downstream

- Situation:
  - QProperty X depends on QProperty Y
  - QProperty X is bound to QProperty Y
- Proposal:
  - Y is a upstream property of X
  - X is a downstream property of Y
- Alternatives:
  - dependency/dependent
  - leader/follower

2 How Dependency Tracking Works

How Dependency Tracking Works

- The connection between two QPropertys is stored in a QPropertyObserver in the QPropertyBindingPrivate
- Each QProperty, when evaluating, stores in a thread-global variable that it is evaluating.
- Each QProperty with a binding, when being read, reads this thread-global variable and (if it is set) establishes a connection.

3 Problems Encountered

Q_OBJECT_COMPAT_PROPERTY

- https://bugreports.qt.io/browse/QTBUG-89890

- Only works correctly if every code path in setter writes underlying object
  - Setter might remove binding
  - Cannot do it if underlying QObjectCompatProperty is not written

Buggy code:

```
#include <QObject>
#include <QDebug>

#include <private/qproperty_p.h>

class Foo : public QObject {
Q_OBJECT
    Q_PROPERTY(int a READ a WRITE setA BINDABLE bindableA)

public:
    int a() const { return Adata; }
    void setA(int value) {
        if(value == Adata)
            return;
        Adata = value;
        // do some important updates after a changed
    }
    QBindable<int> bindableA() { return &Adata; }

private:
    Q_OBJECT_COMPAT_PROPERTY(Foo, int, Adata, setter: &Foo::setA)
};
```

Q_OBJECT_COMPUTED_PROPERTY

- https://bugreports.qt.io/browse/QTBUG-89653
- Only works correctly if every code path in getter reads underlying object
  - Read triggers dependency handling
  - Cannot do it if underlying QObjectComputedProperty is not read

Buggy code:

```
#include <QObject>
#include <QDebug>

#include <private/qproperty_p.h>

class Foo : public QObject {
Q_OBJECT
    Q_PROPERTY(int a READ a WRITE setA BINDABLE bindableA)

public:
    int a() const {
        int result = 42 // some complex computation
        return result;
    }
    QBindable<int> bindableA() { return &Adata; }

private:
    Q_OBJECT_COMPUTED_PROPERTY(Foo, int, Adata, &Foo::a)
};
```

Formulating a binding

```
QProperty<int> a(1);
QProperty<int> b(2);
QProperty<int> c;
c.setBinding([&a, &b](){ return a.value() + b.value(); })
```

Some concerns:

- One can easily bind to something which is not capable of dependency handling
  https://bugreports.qt.io/browse/QTBUG-89518

- User has responsibility to make sure a and b outlive c (or am I mistaken?)
  https://bugreports.qt.io/browse/QTBUG-89848

Qt

5

Proposals for Improvement

Proposal for QObjectCompatProperty

- Remove all "magic" getters and setters. Only value() / setValue().
- In the containing class:
  - Getter must not do anything else than read from underlying object:
    "return myproperty.value();"
  - Setter must not do anything else than write to underlying object:
    "myproperty.setValue(newval);"
  - Old setter is turned into a filter function, given to Q_OBJECT_COMPAT_PROPERTY, and only called by that.

Proposal for QObjectComputedProperty

- Remove "magic" getters. Only value().
- In the containing class:
  - Getter must not do anything else than read from underlying object:
    "return myproperty.value();"
  - Old getter is turned into a private computation function, given to Q_OBJECT_COMPUTED_PROPERTY, and only called by that.

- Clearly state in the documentation:
  It is the programmer's responsibility to call markDirty() whenever it might have changed.

Proposal for new QObjectCustomProperty

- No "magic" getters and setters. Only value() / setValue().
- In the containing class:
  - Getter must not do anything else than read from underlying object:
    "return myproperty.value();"
  - Setter must not do anything else than write to underlying object:
    "myproperty.setValue(newval);"
  - A function to compute the value is given to Q_OBJECT_CUSTOM_PROPERTY.
  - A callback to set the value is given to Q_OBJECT_CUSTOM_PROPERTY

- Clearly state in the documentation:
  It is the programmer's responsibility to call markDirty() whenever it might have changed.

Proposal on how to formulate a binding

Existing:

```
QProperty<int> a(1);
QProperty<int> b(2);
QProperty<int> c;
c.setBinding([&a, &b](){ return a.value() + b.value(); })
```

Proposed:

```
QProperty<int> a(1);
QProperty<int> b(2);
QProperty<int> c;
c.setBinding(
        [](int v1, int v2){ return v1+v2; },
        a, b);
```

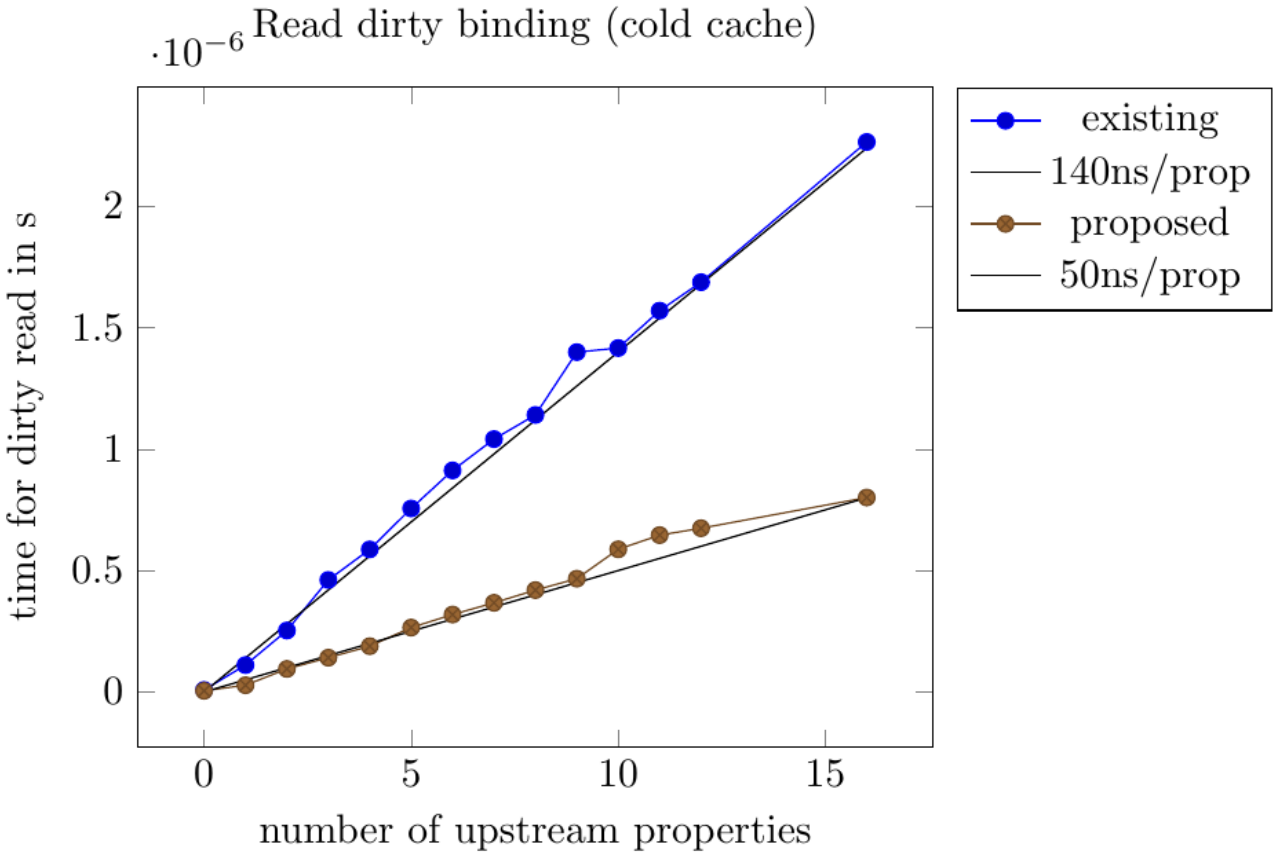
Advantages:

- Visible dependency handling
- Much faster evaluation
- Less memory requirement

Disadvantages:

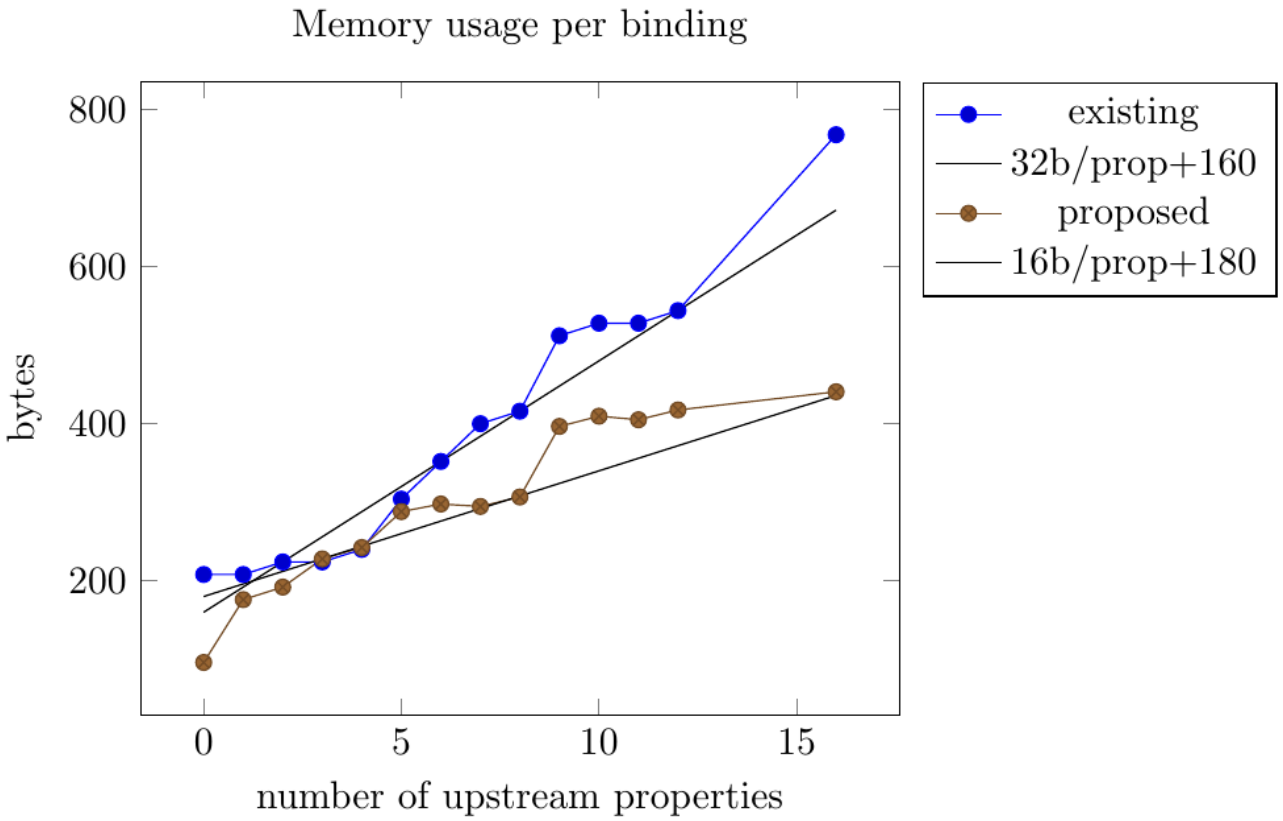
- Breaks existing code
- No dynamic change of dependencies

Measuring binding evaluation time



# upstream	Existing ns	Proposed ns	factor
1	111	28	4.0
2	253	95	2.7
3	461	141	3.3
4	587	188	3.1
8	1141	419	2.7
16	2265	801	2.8

Measuring binding memory usage



# upstream	Existing bytes	Proposed bytes	factor
1	208	176	1.18
2	224	192	1.17
3	224	228	0.98
4	240	242	0.99
8	416	306	1.36
16	768	440	1.75