Reimplementing Signals, Slots, Properties and Bindings from Qt in Pure C++17

An Introduction to KDBindings

Leon Matthes

leon.matthes@kdab.com

The Qt, OpenGL and C++ experts

KDAB

There are only two hard problems in CS:
- Cache invalidation
- naming things
- and off-by-1 errors

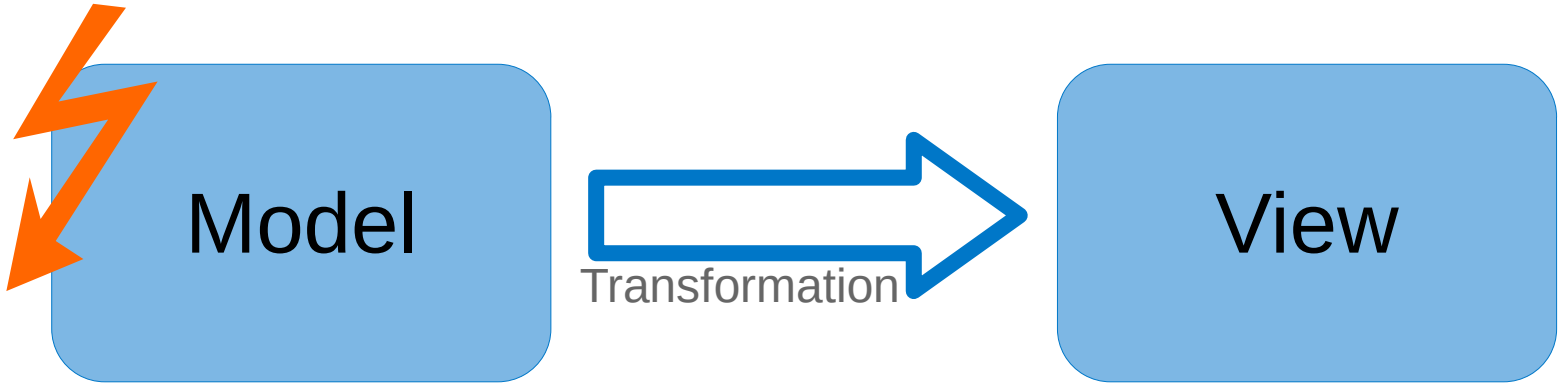
KDAB

There are only two hard problems in CS:

- **Cache invalidation**
- naming things
- and off-by-1 errors

A typical UI Problem

- Model & View store data separately
- Data must be synchronized (and transformed)
- Synchronize only when source data changes



The Qt, OpenGL and C++ experts

A naive solution


```
void setTitle(const std::string& newTitle) { /*...*/ }

void setHeight(int newHeight) {
  if(height != newHeight) {
    height = newHeight;
    setTitle(std::string("Dimensions: ") + std::to_string(width) + " x " + std::to_string(height));
  }
}
void setWidth(int newWidth) {
  if(width != newWidth) {
    width = newWidth;
    setTitle(std::string("Dimensions: ") + std::to_string(width) + " x " + std::to_string(height));
  }
}
```

Lots of boilerplate

The height knows about the title...

Code duplication

Won't scale well... Error prone...

The Qt, OpenGL and C++ experts

Qt's solution

- Wrap Data in Properties

- Notify of changes with Signals

- Data Binding updates and transforms changed data

- Declarative code – easy to write & understand

```
Window {
    visible: true
    title: "Dimensions: " + width + "x" + height
}
```

Why not just use Qt?

Depends on the Qt Meta Object System

```
class MyWindow : public QObject {
    int m_width;
public:
    Q_PROPERTY(int width READ width WRITE setWidth NOTIFY widthChanged)

    int width() const { return m_width; }

    void setWidth(int newWidth) {
        if(m_width != newWidth) {
            m_width = newWidth;
            emit widthChanged();
        }
    }

signals:
    void widthChanged();
};
```

Lots of boilerplate

Requires a special Compiler.

Data Binding not Supported in C++!

Signals in C++17

Signals

```
struct Person {
    Person(std::string const &name) : m_name(name) {}

    Signal<std::string const &> speak;
    std::string m_name;

    void listen(std::string const &message) {
        std::cout << m_name << " received: " << message << std::endl;
    }
};
```

```
Person alice("Alice");
Person bob("Bob");

alice.speak.connect(&Person::listen, &bob);
alice.speak.connect([](auto message){
    std::cout << "Alice said: " << message << std::endl;
}) ;

alice.speak.emit("Have a nice day!");
```

Creating a Signal

Signals are just normal objects

They do not return anything

```
const                                    ame) {}

    Signal<std::string const &> speak;
    std::string m_name;

    void listen(std::string
        std::cout << m_name << " received: " << message << std::endl;
    }
};
```

They can emit any kind of data

Listening to a Signal

"connect" a receiving function (slot) to the Signal

A slot can be a member function

```
alice.speak.connect(&Person::listen, &bob);
alice.speak.connect([](auto message){
    std::cout << "Alice said: " << message << std::endl;
}) ;

alice.speak.emit("Have a nice day!");
```


The Qt, OpenGL and C++ experts

Listening to a Signal

"connect" a receiving function (slot) to the Signal

A slot can be a member function

```
alice.speak.connect(&Person::listen, &bob);
alice.speak.connect([](auto message){
    std::cout << "Alice said: " << message << std::endl;
}) ;
```

Or any other callable object

```
alice.speak.emit("Have a nice day!");
```

Listening to a Signal

```
Person alice("Alice");
Person bob("Bob");

alice.speak.connect(&Person::listen, &bob);
alice.speak.connect([](auto message){
    std::cout << "Alice said: " << message << std::endl;
}) ;

alice.speak.emit("Have a nice day!");
```

The output will be:
Bob received: Have a nice day!
Alice said: Have a nice day!

KDAB

Properties in C++17

The Qt, OpenGL and C++ experts

Properties

```
Property<std::string> myProperty(“This is a property”);

myProperty.valueChanged().connect([](const std::string& value) {
  std::cout << value << std::endl;
});

myProperty = "Properties emit Signals when they change!";
```

Declaring Properties

Properties are just normal objects

Constructed like any instance of the wrapped type

```
Property<std::string> myProperty("This is a property");

myProper .connect([](const std::string& value) {
  std::cout ndl;
});

myProperty = "Properties emit Signals when they change!";
```

They wrap a value of any type

The Qt, OpenGL and C++ experts

Reacting to changes

Property<std (“This is

Signal for changes

Connected like any other Signal

```
myProperty.valueChanged().connect([](const std::string& value) {
  std::cout << value << std::endl;
});

myProperty = "Properties emit Signals when they change!";
```

Creating a change

```
Property<std::string> myProperty("This is a property");

myProperty.valueChanged().connect([](const std::string& value) {
  std::cout << value << std::endl;
});

myProperty = "Properties emit Signals when they change!";
```

Simply assign a new value

The string will be printed to std::cout

KDAB

Data Binding in C++17

The Qt, OpenGL and C++ experts

Data Binding

```
KDBINDINGS_DECLARE_STD_FUNCTION(to_string)

Property<int> width(1920);
Property<int> height(1080);
Property<std::string> title = makeBoundProperty(
    "Dimensions: " + to_string(width) + " x " + to_string(height)
);

title.valueChanged().connect([](const auto &title) { std::cout << title << std::endl; });

width = 500;
height = 300;
```


The Qt, OpenGL and C++ experts

Data Binding

KDBINDINGS_DECLARE_STD_FUNCTION(to_string)

Functions can be registered for use in data binding

```
Property<int> width(1920);
Property<int> height(1080);
Property<std::string> title = makeBoundProperty(
    "Dimensions: " + to_string(width) + " x " + to_string(height)
);

title.valueChanged().connect([](const auto &title) { std::cout << title << std::endl; });

width = 500;
height = 300;
```


The Qt, OpenGL and C++ experts

Data Binding

```
KDBINDINGS_DECLARE_STD_FUNCTION(to_string)

Property<int> width(1920);
Property<int> height(1080);
Property<std::string> title = makeBoundProperty(
    "Dimensions: " + to_string(width) + " x " + to_string(height)
);
title.valueChanged().connect([](const auto &title) { std::cout << title << std::endl; });

width = 500;
height = 300;
```

Creates a Property with a data binding

Data binding can use operators

And Registered functions

The Qt, OpenGL and C++ experts

Data Binding

```
KDBINDINGS_DECLARE_STD_FUNCTION(to_string)

Property<int> width(1920);
Property<int> height(1080);
Property<std::string> title = makeBoundProperty(
    "Dimensions: " + to_string(width) + " x " + to_string(height)
);

title.valueChanged().connect([](const auto &title) { std::cout << title << std::endl; });
```

width = 500;

title: *Dimensions: 500 x 1080*

height = 300;

title: *Dimensions: 500 x 300*

Unnecessary Calculation

KDAB

Adding Deferred Evaluation

Data Binding

```
BindingEvaluator evaluator;

Property<int> width(1920);
Property<int> height(1080);
Property<std::string> title = makeBoundProperty(
        evaluator,
        "Dimensions: " + to_string(width) + " x " + to_string(height)
);

width = 500;
height = 300;

evaluator.evaluateAll();
```


The Qt, OpenGL and C++ experts

Data Binding

```
BindingEvaluator evaluator;

Property<int> width(1920);
Property<int> height(1080);
Property<std::string> title = makeBoundProperty(
        evaluator,
        "Dimensions: " + to_string(width) + " x " + to_string(height)
);

width = 500;
height = 300;

evaluator.evaluateAll();
```

Add a BindingEvaluator to the binding

The title will not change yet

Control update frequency

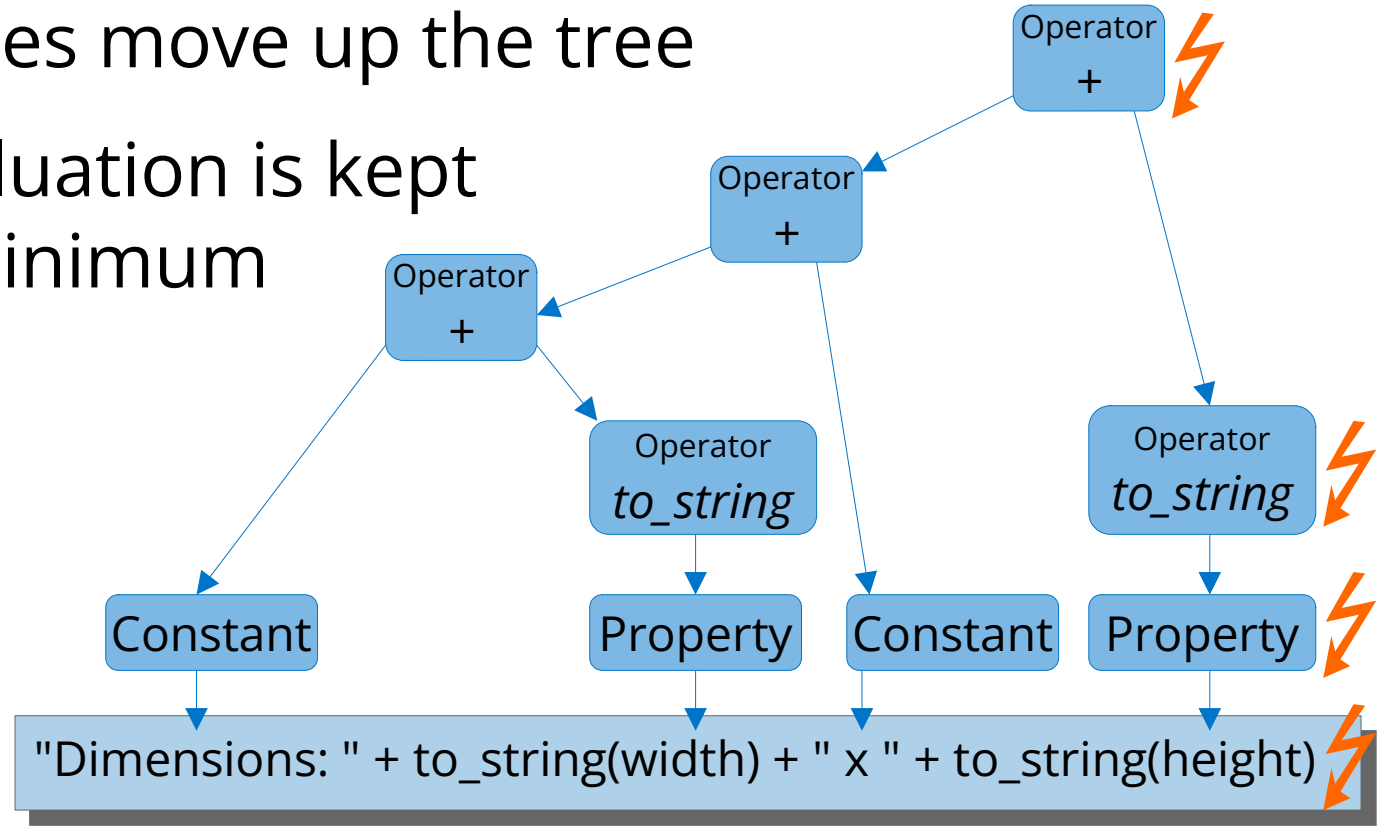
title will now be updated

KDAB

Looking under the Hood

Creating a Binding

- KDBindings generates an AST out of any Binding
- Changes move up the tree
- Reevaluation is kept to a minimum



The Qt, OpenGL and C++ experts

KDAB

Real world use

```
class Q_3DRENDERSHARED_EXPORT QCameraLens : public Qt3DCore::QComponent
{
    float farPlane() const;
    float fieldOfView() const;
    void setBottom(float bottom);
    void setTop(float top);
    void setProjectionMatrix(const QMatrix4x4 &projectionMatrix);
    void setExposure(float exposure);
Q_SIGNALS:
    void projectionTypeChanged(QCameraLens::ProjectionType projectionType);
    void nearPlaneChanged(float nearPlane);
    void farPlaneChanged(float farPlane);
    void fieldOfViewChanged(float fieldOfView);
    void aspectRatioChanged(float aspectRatio);
    void leftChanged(float left);
    void rightChanged(float right);
    void bottomChanged(float bottom);
    void topChanged(float top);
    void projectionMatrixChanged(const QMatrix4x4 &projectionMatrix);
    void exposureChanged(float exposure);
    void viewSphere(const QVector3D &center, float radius);
protected:
    explicit QCameraLens(QCameraLensPrivate &dd, QNode *parent = nullptr);
private:
    Q_DECLARE_PRIVATE(QCameraLens)
    Qt3DCore::QNodeCreatedChangeBasePtr createNodeCreationChange() const override;
    void sceneChangeEvent(const Qt3DCore::QSceneChangePtr &change) override;
};
```


The Qt, OpenGL and C++ experts

```
class CameraLens : public Component {
public:
    CameraLens();
    ~CameraLens();

    enum class ProjectionType { Perspective, Orthographic };

    Property<float> nearPlane{ 0.1f };
    Property<float> farPlane{ 1000.0f };
    Property<ProjectionType> projectionType{ ProjectionType::Perspective };

    Property<glm::mat4> projectionMatrix{};

    Property<float> verticalFieldOfView{ 25.0f };
    Property<float> aspectRatio{ 1.0f };

    Property<float> left{ -0.5f };
    Property<float> right{ 0.5f };
    Property<float> top{ -0.5f };
    Property<float> bottom{ 0.5f };

    Signal<CameraLens *> destroyed;
    Signal<CameraLens *> changed;

    void setOrthographicProjection(float left, float right, float top, float bottom, float nearPlane, float farPlane);
    void setPerspectiveProjection(float verticalFieldOfView, float aspectRatio, float nearPlane, float farPlane);
};
```


KDAB
The Qt, OpenGL and C++ experts

Reimplementing QCameraLens

```
void QCameraLens::setFarPlane(float
{
    Q_D(QCameraLens);
    if (qFuzzyCompare(d->m_farPlane, farPlane))
        return;
    d->m_farPlane = farPlane;

    const bool wasBlocked = blockNotifications(true);
    emit farPlaneChanged(farPlane);
    blockNotifications(wasBlocked);

    d->updateProjectionMatrix();
}

void QCameraLens::setFieldOfView(float fieldOfView) { ... }
void QCameraLens::setAspectRatio(float aspectRatio) { ... }
void QCameraLens::setLeft(float left) { ... }
void QCameraLens::setRight(float right) { ... }
void QCameraLens::setBottom(float bottom) { ... }
void QCameraLens::setTop(float top) { ... }
...
```

Manual check for equality

Manual Signal emit

Manual update to projectionMatrix

Code duplication for every Property

Set up the data binding In the constructor

```
CameraLens::CameraLens() {
    projectionMatrix = makeBoundProperty(
        updateProjection(verticalFieldOfView, aspectRatio,
                         left, right, top, bottom,
                         nearPlane, farPlane, projectionType));
}

CameraLens lens;
lens.farPlane  = 10.0f;
```

Signals automatically emitted & matrix updated

KDAB

KDBindings

The Qt, OpenGL and C++ experts

KDBindings – Features

- Signals & Slots
- Properties templated on the contained type
- Property bindings for reactive coding
- Lazy evaluation of property bindings
- Stand-alone “header-only” library
- Can be used with Qt if necessary

KDBindings – Limitations

- Not thread safe
- Increase in compile time
- Only available in C++17 or newer

KDBindings – Future plans

- Mutable access to Property values
- Reduce sizeof(Signal) to sizeof(void*) when empty
- Construct Properties & Bindings at compile time
- Benchmarking & Performance improvements
- Improved compiler errors

The Qt, OpenGL and C++ experts

KDAB

Installing KDBindings

KDBindings – Project integration

Clone/Download KDBindings

```
$ git clone https://github.com/kdab/kdbindings
$ cd kdbindings
$ mkdir build && cd build
$ cmake ..
$ cmake --install .
```

We recommend Out-of-source build

Configure CMake

We recommend Installing KDBindings

KDBindings – Project integration

```
cmake_minimum_required(VERSION 3.12)
project(DataBinding)

set(CMAKE_CXX_STANDARD 17)

add_executable(databinding main.cpp)

find_package(KDBindings REQUIRED)
target_link_libraries(databinding
  KDAB::KDBindings
)
```

Make sure C++17 is enabled!

```
$ git clone https://github.com/KDAB/KDBindings
$ mkdir build && cd build
$ cmake ..
$ cmake --install .
```

After installation KDBindings is available as a package

Alternatively simply **add_subdirectory(...)**

Link against the **KDAB::KDBindings** target

KDBindings – Project integration

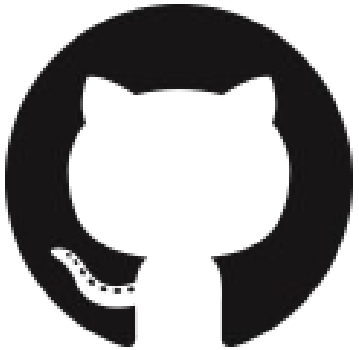
All of KDBindings
Is in the **kdbindings**
Include directory

```
#include <iostream>
#include <kdbindings/property.h>

using namespace KDBindings;

int main() {
    Property<std::string> message;

    message.valueChanged().connect([](const auto& message) {
        std::cout << message << std::endl;
        });
    message = "Hello world from KDBindings";
}
```

Don't forget the
KDBindings namespace

Enjoy
KDBindings!

KDBindings – Where to get it?



Available now on Github!

https://github.com/KDAB/kdbindings

Read the docs at:

https://docs.kdab.com/kdbindings/latest/

Published under the MIT license!

Contact info@kdab.com for different licensing options

KDAB

Questions?

Under the hood – Signals & Slots