Qt GUIs with Rust

Rust Qt Binding Generator

Jos van den Oever

- For Rust developers
- For Qt developers
- For developers of Qt
- For developers of $future_rust_ui
Goals

- Find a way to write programs that combine Qt and Rust
- Small effort to create the tool
- Small effort to use the tool
- Simple with minimal dependencies
- Good Qt
- Good Rust
What is Qt

- VLC
- a
- SAILFISH OS
- müsescore
- Krita
- LUMINA
- SLXQt
- Oracle VirtualBox
- QML
- ubports
- MAYA
- Plasma 5
Bindings for Qt5

- C#, Crystal, D, Go, Haskell, JavaScript, Java, Julia, Pascal, Python, OCaml, QML, Ring, Ruby, Rust, D

- Why have bindings?
  - Use a familiar language
  - Combine with existing code
Rust selling points

- safety
  - (de)allocation
  - bounds checking
  - concurrency
  - error handling
- algebraic data types (enum)
- macros
- cargo

*wrap Qt API in Rust: round hole / square peg*
Wrapping Qt API in Rust
Qt

Rust

JSON data model

Rust Qt Binding Generator
Turn the binding around

Your GUI (Qt / QML)

Generated binding (FFI, QObject, QAbstractItemModel)

Your model (JSON)

Qt5 (C++)

Your logic (Rust)

Rust crates
The core of Qt: QObject

- Signals and slots
  - a.ka. events and listeners
  - loosely coupled: disconnect if one party is destroyed
- Memory management
  - hierarchical ownership
- Object properties
  - runtime introspection via QMetaObject
The core of Qt: QAbstractItemModel

- Base class for providing data to lists, trees and tables
- Used *everywhere* in Qt
- Notoriously hard to implement
Bits and pieces

- Model description in JSON
- Generated
  - Binding.h
  - Binding.cpp
  - interface.rs
- Your Rust code compiled into static library, e.g. librust.a
- Imported into Qt with “#include <Binding.h>” or
  “import RustCode 1.0;”
QML

- Qt Modeling Language
- Declarative user interface markup language with JavaScript logic
- Qt Quick: 2D and 3D scene graph
- UI for Nokia Meego phones, Ubuntu phone, Sailfish, KDE Plasma Desktop, KDE Plasma Mobile
- Designed for efficiency on graphics cards (not unlike webrender)
Step 1: write bindings.json

To-do list:
- What needs to be done?
- write bindings.json
- run rust_qt_binding_generator
- check bindings.h
- check bindings.cpp
- check interface.rs
- write implementation.rs
- write main.qml

7 items left
Step 1: write bindings.json

```
"objects": {
  "Todos": {
    "type": "List",
    "properties": {
      "count": {
        "type": "quint64"
      },
      "activeCount": {
        "type": "quint64"
      }
    }
  },
  "itemProperties": {
    "completed": {
      "type": "bool",
      "write": true,
      "roles": [ [ "display" ] ]
    },
    "description": {
      "type": "QString",
      "write": true,
      "roles": [ [], [ "display" ] ]
    }
  }
}
```
Step 2: rust_qt_binding_generator

- write-bindings.json
- run rust_qt_binding_generator
- check bindings.h
- check bindings.cpp
- check interface.rs
- write implementation.rs
- write main.qml

6 items left
Step 2: rust_qt_binding_generator

- **rust_qt_binding_generator binding.json**
- Simple command-line executable to be used in any build system, e.g. CMake or build.rs
- Generates `interface.rs`, `Binding.h` and `Binding.cpp`
- Generates `initial implementation.rs`
Step 3: check bindings.h
class Todos : public QAbstractItemModel
{
    Q_OBJECT
public:
    explicit Todos(QObject *parent = nullptr);
    ~Todos();
    Q_PROPERTY(quint64 activeCount READ activeCount NOTIFY activeCountChanged FINAL)
    Q_PROPERTY(quint64 count READ count NOTIFY countChanged FINAL)
    quint64 activeCount() const;
    quint64 count() const;
    QVariant data(const QModelIndex &index, int role) const override;
    Q_INVOKABLE QVariant completed(int row) const;
    Q_INVOKABLE bool setCompleted(int row, const QVariant& value);
    Q_INVOKABLE QVariant description(int row) const;
    Q_INVOKABLE bool setDescription(int row, const QVariant& value);
Step 4: check bindings.cpp
extern "C" {
    void todos_free(Todos::Private*);
    quint64 todos_active_count_get(const Todos::Private*);
    void todos_data_description(
        const Todos::Private*, int, QString*, qstring_set);
    bool todos_set_data_description(
        Todos::Private*, int, qstring_t);
};
QVariant Todos::data(const QModelIndex &index, int role) const
{
    Q_ASSERT(rowCount(index.parent()) > index.row());
    switch (index.column()) {
    case 0:
        switch (role) {
            case Qt::DisplayRole:
            case Qt::UserRole + 0:
                return completed(index.row());
            case Qt::UserRole + 1:
                return description(index.row());
        }
    case 1:
        switch (role) {
            case Qt::DisplayRole:
            case Qt::UserRole + 1:
                return description(index.row());
        }
    }
    return QVariant();
}
Step 5: check interface.rs
Step 5: check interface.rs

```rust
pub trait TodosTrait {
    fn new(emit: TodosEmitter, model: TodosList) -> Self;
    fn emit(&self) -> &TodosEmitter;
    fn active_count(&self) -> u64;
    fn count(&self) -> u64;
    fn insert_rows(&mut self, row: usize, count: usize) -> bool { false }
    fn remove_rows(&mut self, row: usize, count: usize) -> bool { false }
    fn completed(&self, item: usize) -> bool;
    fn set_completed(&mut self, item: usize, bool) -> bool;
    fn description(&self, item: usize) -> &str;
    fn set_description(&mut self, item: usize, String) -> bool;
}
```
Step 6: write implementation.rs

- What needs to be done?
  - write-bindings.json
  - run-rust_qt_binding_generator
  - check-bindings.h
  - check-bindings.cpp
  - check-interface.rs
  - write implementation.rs
  - write main.qml

2 items left
struct TodosItem {
    completed: bool,
    description: String,
}
Step 7: write main.qml
Step 7: write main.qml

```qml
import QtQuick 2.9
import RustCode 1.0;

ApplicationWindow {
    Todos {
        id: todoModel
    }
    Component {
        id: todoDelegate
        Pane {
            CheckBox {
                checked: completed
            }
            Label {
                text: description
            }
        }
    }
    Flickable {
        ListView {
            model: todoModel
            delegate: todoDelegate
        }
    }
}
```
Step 8: done!

- write-bindings.json
- run-rust_qt_binding_generator
- check-bindings.h
- check-bindings.cpp
- check-interface.rs
- write-implementation.rs
- write-main.qml
Rust Qt Binding Generator

- Do not shoehorn Qt in a Rust API
- Wrap Rust code in QObject or QAbstractItemModel
- Rust Qt Binding Generator does this for you
- Good Qt
- Good Rust