Extend your KDE application
Using QML!

Artur Duque de Souza
Aug/2011
Agenda

• (Big) Introduction
• A problem
• KDE Solution
• Issues
• Future
Qt Script
QtScript

C++ API to make your applications scriptable
QScriptEngine

- Environment to evaluate a script
- Context
- Global Object
- Use QMetaObject system to automatically export QObjects
QObjects

Can be *exported* out of the box:

- Properties
- Signals
- Slots
- `Q_INVOKABLE`
QScriptValue

Container for QtScript data types:

• Support for ECMA-262 types
• Support for QObject, QVariant and QMetaObject
• Prototype property that is common to all instances of an object
JS Bindings
**JS Bindings for Qt**

**Bindings** are proxy objects/functions to interface with the ‘real’ libraries
JS Bindings for Qt

Steps to create your bindings:

• Create wrap code (check context arguments)
• Register your wrappers with the engine
• Be happy :)

JS Bindings for Qt

**Steps** to create your bindings:

- Create wrap code (check context arguments)
- Register your wrappers with the engine
- Be happy :)


**JS Bindings for Qt**

**Steps** to create your bindings:

- Create wrap code (check context arguments)
- Register your wrappers with the engine
- Be happy :)


Post No Bills

by Order of
King George III

IN CONGRESS, July 4, 1776.

The unanimous declaration of the States of America,

QML
Declarative language to ease the development of UIs
QDeclarativeEngine

- Handles QML code
- Does not inherit QScriptEngine
- It has a QScriptEngine inside
QDeclarativeEngine

- Handles QML code
- Does **not** inherit QScriptEngine
- It **has** a QScriptEngine inside
QDeclarativeEngine
Public API

- QML specific methods
- It has its own 'context': QDeclarativeContext
- QObject works out of the box
- It’s possible to register C++ declarative items
QDeclarativeExpression

Evaluate a JS expression in a QML context
First of all...

... why use QML?

Declarative languages are way better (and faster) to build rich UIs!

- Microblog plasmoid (C++): 1250 LOC
- Declarative Microblog: 500 LOC
First of all...

... why use QML?

Declarative languages are way better (and faster) to build rich UIs!

• Microblog plasmoid (C++): **1250 LOC**
• Declarative Microblog: **500 LOC**
Declarative languages are way better (and faster) to build rich UIs!

• Microblog plasmoid (C++): 1250 LOC
• Declarative Microblog: 500 LOC
KDE Use case

- Uses **QtScript** since a **long time ago**
- It has a lot of **JS bindings for non-QObject classes**
  - i18n
  - QGraphicsLayout
  - QFont
  - UI loader
  - ...
KDE Use case

- Uses **QtScript** since a **long time ago**
- It has a lot of **JS bindings** for **non-QObject classes**
  - i18n
  - QGraphicsLayout
  - QFont
  - UI loader
  - ...
The problem

QDeclarativeEngine does not export its QScriptEngine!
Because of this, there is no way to register our bindings.
Possible solution

Export all non-QObject classes using QObject wrappers
The rise of **libkdeclarative**

**Spoiler alert:** This is the way you’re going to use QML in your KDE app!
QScriptValue
Let’s take a look at QScriptValue API

- **QScriptEngine**\* engine() const
- All slots’ arguments are QScriptValues on the script side
Access to the internal QScriptEngine!

Example

```cpp
root->setContextProperty("__eng", engineAccess);
QDeclarativeExpression expr("__eng.setEngine(this)");
expr.evaluate();
```
Access to the internal QScriptEngine!

Example

```cpp
void EngineAccess::setEngine(QScriptValue val) {
    m_kDeclarative->d->scriptEngine = val.engine();
}
```
Still one last problem

The **Global Object** used by QML is read-only
Let’s change the global object

Example

```javascript
QScriptValue originalO = engine->globalObject();
QScriptValue newO = engine->newObject();

QScriptValueIterator iter(originalO);
while (iter.hasNext()) {
    // read props, flags
    newO.setProperty(iter.scriptName(), iter.value());
}
scriptEngine->setGlobalObject(newO);
```
Using QML right now

Use libkdeclarative in your application in order to have QML integration with KDE environment.
Integration with KDE

- QIcon
- QPixmap
- QFont
- KJob
- KConfig
- .ui loader
- Plasma’s DataEngines and Services
- ...

...
What about widgets?

KDE Components

- GSoC 2011 Project: Daker Fernandes
- Step **one**: port all Plasma Widgets to QML
- Step **two**: start porting kdelibs/ui

This GSoC project is only about step one!
What about widgets?
KDE Components

- GSoC 2011 Project: Daker Fernandes
- Step one: port all Plasma Widgets to QML
- Step two: start porting kdelibs/ui

This GSoC project is only about step one!
KDE Components

Buttons
- Button
- Checkable Button
- Different Font
- Icon Button

Check Box
- Check Box 1
- Check Box 2
- A loooooooong text

BusyIndicator

Slider
Color Selector
- Red
- Green
- Blue

Scroll Bar
0
1
2
3
4
5
6

Radio Button
- RadioButton

Button Row
- A
- B
- C

Button Column
- Alice
- Bob
- Charles

Scroll Decorator
Issues

PROBLEM SOLVED
Issues

QML internal objects may not behave as documented
Done this way because of performance issues

*Example*: QScriptString has persistent handle to the string, and that is expensive
Future
Qt 5 and QtQuick 2.0

- QML will switch from JavaScriptCore to V8
- The ‘KDE solution’ will stop working: everything needs to be QObject
QScriptValue can be used as a module API
Only in JavaScript code, as it’s imperative

Example

```javascript
import My.Qml.Module as Module

Item {
    Component.onCompleted: {
        var obj = new Module.MyType;
        Module.doSomething(obj);
    }
}
```
Thanks!

Questions?

Artur Duque de Souza

http://blog.morpheuz.cc

asouza@kde.org