Qt Quick

C++ and JavaScript

A free software application framework, which includes a declarative scripting language called QML

Logic implemented in C++ and UI implemented in QML(JavaScript). QML is to allow developers to create fancy UI without typing C++.

Some simple widgets

main.qml

```qml
import QtQuick 2.15
import QtQuick.Window 2.15
import QtQuick.Controls 2.15

// Create a Window element
Window {
    // Properties
    width: 640
    height: 480
    visible: true
    // Marked for translation
    title: qsTr("Quit button")

    Button {
        x: 20
        y: 20
        text: "Quit"
        onClicked: Qt.quit()
        background: Rectangle {
                implicitWidth: 100
                implicitHeight: 40
                color: button.down ? "#d6d6d6" : "#f6f6f6"
                border.color: "#26282a"
                border.width: 1
                radius: 4
            }
    }

    Row {
        Slider {
            id: slider
            from: 0
            to: 100
            value: 25
            x: 40
            y: 100
        }

        Label {
            text: Math.floor(slider.value)
            x: 20
            y: 100
            color: "red"
        }
    }
}
```

main.cpp

```cpp
int main(int argc, char *argv[])
{
#if QT_VERSION < QT_VERSION_CHECK(6, 0, 0)
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    // Load main.qml
    const QUrl url(QStringLiteral("qrc:/main.qml"));
    // If loading was successful, object contains a pointer to the loaded object, otherwise the pointer is NULL
    // objUrl is checked if it was created from the correct url
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                     &app, [url](QObject *obj, const QUrl &objUrl) {
        if (!obj && url == objUrl)
            QCoreApplication::exit(-1);
    }, Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

Licenses

Qt Quick is available under commercial licenses and free software license. Since Qt 5.4, these free software licenses are GNU Lesser General Public License, version 3, or the GNU General Public License, version 2.

Modules

Qt Quick control module - implements several control such as buttons, menus and views.

Several Qt modules such as QtQuick.Particles module and Qt Graphical Effects module - provide types for creating special effect in application.

The Qt Sensors module - allows applications to read information from sensors such as accelerometers and tilt sensors.

Several Qt modules provide QML API for networked and mobile device, such as Qt Bluetooth, Qt Location (viable mapping solutions), Qt Position (for example determine a position on a map).

Why people always say Qt Quick is for mobile:

Several Qt modules provide QML API for networked and mobile device, such as Qt Bluetooth, Qt Location, maybe this is why people always say Qt Quick is for mobile.

Touch based inputs. Swipe, pinch and tap gestures are handled.

Qt Quick has ready-to-use popups, animations, tabs and layouts, flickables, drawers (swipe-based side panel), and the usual controls

see wearable example

Testing

For UI unit and integration testing Qt provides the Qt Quick Test framework where you can write JavaScript functions to test your UI items. For C++, the provided Qt Test framework gives basic testing functionality and is good for testing Qt flavoured C++ code (e.g. signals and slots).

Qt Widget vs Qt Quick

Qt Widget is low level compared to Qt Quick, but Qt Widget runs faster and have better performance, more exposed to native API such as Qt Style Sheet

Qt Quick has ready-to-use popups, animations, tabs and layouts, flickables, drawers, and the usual controls, which can make fancy ui.

And it seperate UI and logic(example from Qt Documentation)

UI:

```
import QtQuick 2.15
import QtQuick.Window 2.15
import QtQuick.Controls 2.15

Window {
    width: 640
    height: 480
    visible: true
    title: qsTr("Seperate UI and Logic")

    Button {
        id: button
        text: qsTr("Call me")
        onClicked: backend.callMe()
        background: Rectangle {
                    implicitWidth: 100
                    implicitHeight: 40
                    color: button.down ? "#d6d6d6" : "#f6f6f6"
                    border.color: "#26282a"
                    border.width: 1
                    radius: 4
                }
    }
}
```

Logic:

```cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QSettings>

class Backend : public QObject
{
    Q_OBJECT
public:
    explicit Backend(QObject *parent = nullptr);

signals:

public slots:
    void callMe() {
        qDebug() << "I'm being called";
    }
};

int main(int argc, char *argv[])
{
#if QT_VERSION < QT_VERSION_CHECK(6, 0, 0)
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
#endif
    QGuiApplication app(argc, argv);

    Backend backend;

    QQmlApplicationEngine engine;
    // Set the value of the name property on this context.
    engine.rootContext()->setContextProperty("backend", &backend);
    const QUrl url(QStringLiteral("qrc:/main.qml"));
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                     &app, [url](QObject *obj, const QUrl &objUrl) {
        if (!obj && url == objUrl)
            QCoreApplication::exit(-1);
    }, Qt::QueuedConnection);
    engine.load(url);

    return app.exec();
}
```

keep C++ types unaware of QML as much as possible.

https://doc.qt.io/qt-5/qtquick-bestpractices.html

New:

QQuickWidget
will let you embed Qt Quick elements into a widget-based desktop application. The prime use case is to have the latest user interface capabilities with animation, hardware acceleration, shaders integrated into your project. In addition, it enables a smooth conversion of your user interface. (How qt quick work with qt widget)

See quickwidget example

Qt Quick 3D

allows creating modern-looking applications and devices via a unique workflow.

https://www.qt.io/blog/qt-roadmap-for-2022

Qt Quick advantage

Declarative languages are strongly suited for defining UIs.

QML code is simpler to write

JavaScript can easily be used in QML to respond to events.

Seperate UI and Logic

Things to explore

Can we have my own type element

How is the performance

How to emit a signal

Can we use css