Qt Quick

During my research at work on the feasibility of integrating Qt Quick into our codebase, I explored the capabilities of this powerful software application framework. Qt Quick is a versatile framework that features a declarative scripting language known as QML. With Qt Quick, the logic of an application can be efficiently implemented in C++, while the user interface is crafted using QML, which is based on JavaScript. QML simplifies the process of creating sophisticated and visually appealing user interfaces, reducing the reliance on extensive C++ coding. This approach enhances the development experience, enabling developers to create feature-rich UIs with ease.

As a practical demonstration of Qt Quick's capabilities, here are some examples of simple widgets created using this framework:

main.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

```
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 offers a flexible licensing model, making it accessible for both commercial and open-source projects. Starting from Qt 5.4, Qt Quick is available under two primary free software licenses:

1. **GNU Lesser General Public License, version 3 (LGPL-3.0)**: This license provides more permissive terms for open-source projects and software development.
2. **GNU General Public License, version 2 (GPL-2.0)**: This license is suitable for projects that adhere to stricter open-source guidelines.

Modules

The Qt Quick module is the cornerstone of QML application development, and it encompasses several essential submodules:

- **Qt Quick Control Module**: This submodule facilitates the creation of user interface elements, including buttons, menus, and views, streamlining the development of user-friendly interfaces.

- **QtQuick.Particles Module and Qt Graphical Effects Module**: These modules offer a rich collection of tools and types to incorporate captivating special effects into your applications, enhancing the visual appeal.
- **Qt Sensors Module**: With this module, Qt applications gain the capability to retrieve data from various sensors, such as accelerometers and tilt sensors, enabling the development of innovative sensor-driven features.

In addition to the core modules, Qt provides a range of supplementary modules, each tailored to specific application needs. These include:

- **Qt Bluetooth Module**: Offers QML APIs for seamless integration with Bluetooth devices.
- **Qt Location Module**: Equips developers with mapping solutions for location-based services.
- **Qt Position Module**: Enables the determination of precise geographical positions on maps and integration with location-based applications.

Why is Qt Quick Often Associated with Mobile Development?

Qt Quick is frequently linked to mobile application development due to the presence of several Qt modules that offer QML APIs tailored for networked and mobile devices. These modules, including Qt Bluetooth and Qt Location, contribute to the perception that Qt Quick is particularly suitable for mobile platforms.

Notable Features of Qt Quick for Mobile Development:

- **Touch-Based Inputs**: Qt Quick is equipped to handle touch-based inputs, making it proficient in recognizing and responding to gestures like swiping, pinching, and tapping.
- **Pre-Built UI Components**: Qt Quick provides an extensive collection of pre-built components, such as popups, animations, tabs, layouts, flickables, and drawers (swipe-based side panels). These components simplify the creation of stylish and user-friendly mobile interfaces.

Testing Capabilities:

For effective UI unit and integration testing, Qt offers the Qt Quick Test framework. With this framework, you can write JavaScript functions to test UI elements. Additionally, for C++ code testing, Qt provides the Qt Test framework, which is well-suited for evaluating Qt-specific C++ features like signals and slots.

Qt Widget vs. Qt Quick:

Comparing Qt Widgets and Qt Quick, Qt Widgets are considered lower-level, but they offer superior performance and faster execution. Widgets also provide greater access to native APIs, including the use of Qt Style Sheets for customization.

On the other hand, Qt Quick excels in delivering ready-to-use components like popups, animations, tabs, layouts, flickables, drawers, and common controls, empowering developers to create visually appealing

and sophisticated user interfaces. Moreover, Qt Quick efficiently separates the user interface from the logic, following a best practice as exemplified in 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:

```
#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();
}
```

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

Emerging Trends in Qt Quick:

1. QQuickWidget: Embedding Qt Quick Elements in Desktop Applications

QQuickWidget introduces a powerful capability to seamlessly integrate Qt Quick elements into traditional widget-based desktop applications. This feature is especially valuable when you want to leverage the latest user interface advancements, including animations, hardware acceleration, and shaders, within your project. Not only does it enhance your application's visual appeal, but it also facilitates the smooth transition to a more modern user interface design.

2. Qt Quick 3D: Transforming Applications and Devices with Modern Aesthetics

Qt Quick 3D is a groundbreaking trend in Qt Quick development. It empowers developers to create applications and devices with a contemporary and visually appealing look through a unique workflow. This trend opens up exciting possibilities for immersive and visually engaging user experiences.

Advantages of Qt Quick:

- **Declarative Languages for UI Design:** Qt Quick leverages declarative languages, which are exceptionally well-suited for defining user interfaces. This approach simplifies the process of specifying the structure and behavior of your UI elements.
- **Simplified QML Code:** Writing QML code is notably simpler compared to traditional approaches. The language's clear and concise syntax accelerates development and streamlines the creation of interactive interfaces.
- **Efficient Event Handling with JavaScript:** Qt Quick allows the seamless integration of JavaScript within QML. This enables you to respond to various events and user interactions with ease, enhancing the dynamic capabilities of your application.
- **Clear Separation of UI and Logic:** One of the core advantages of Qt Quick is its emphasis on separating the user interface from the application's logic. This separation promotes a clean and

maintainable codebase, making it easier to collaborate on projects and extend functionality.