

Знакомство с Qt Quick



1. Введение


Изначально, виджеты из модуля **Qt Widgets** являлись основным подходом к созданию *GUI* на **Qt**. Позднее появился второй подход, связанный с технологией **Qt Quick**.

Qt Quick vs Qt Widgets

Использование модуля **Qt Widgets**:

- Если вы стремитесь к простым desktop-приложениям. Ориентировано только на ПК.
- При создании специализированных настольных инструментов, таких как программы САПР(Система автоматизации проектирования), инструменты администрирования, приложения для графического дизайна и т. д.
- Когда вы действительно хорошо разбираетесь в C++ или Python и хотите использовать эти навыки в пользовательском интерфейсе.

Использование модуля **Qt Quick**:

- При работе с микроконтроллерами, например Qt для MCU.
 - Для проектов, ориентированных на встроенные или мобильные устройства.
 - Если вы сосредоточены на устройствах с сенсорным экраном.
 - Если вам нужен элегантный, современный и богатый пользовательский интерфейс с анимационными переходами.
 - Для настоящих кроссплатформенных целей, охватывающих как настольные, так и мобильные устройства.
- 

2. Что такое Qt Quick ?

Qt Quick – это фреймворк пользовательского интерфейса, построенный поверх **Qt** на языке **QML**.

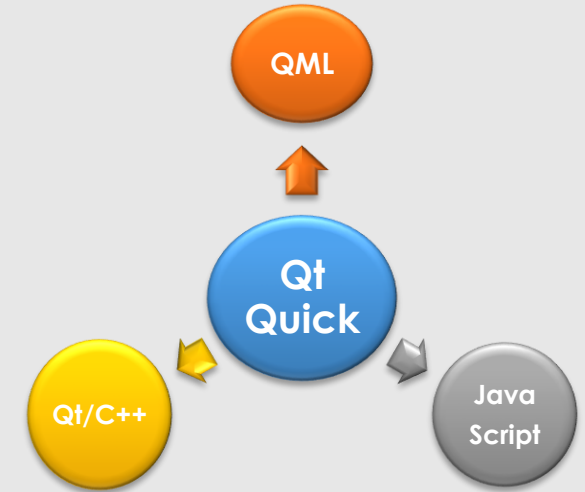
Qt Quick – набор технологий для создания анимированных, динамических, пользовательских интерфейсов нового поколения как для мобильных устройств так и для настольных ПК.

Технология **Qt Quick** является интерпретируемой, это дает преимущество - отсутствует процесс компиляции, что позволяет быстро изменять программу, сразу же запускать ее и тут же смотреть на сделанные изменения в действии.

Мобильные устройства обладают ограниченными ресурсами (ЦП, память, разрешение экрана). Одна из целей, при создании **Qt Quick**, была возможность работы именно в этих условиях.

Qt Quick использует парадигму веб-подхода:

ее пакеты содержат при себе все необходимое, поэтому инсталляция не требуется. Единственным условием является наличие на исполняющей стороне приложения, способного интерпретировать эти **Qt Quick**-программы. Такой программой может быть, например, программа **qmlscene**, которая входит в комплект поставки **Qt**.



3. Введение в QML

QML – (*Qt Modeling Language*) декларативный (описательный) язык для создания пользовательского интерфейса (синтаксис похож на *JSON*). Появился с развитием мобильных устройств с сенсорными экранами, позволяя создавать гибкие пользовательские интерфейсы с минимальным написанием кода.

QML описывает, как выглядят и как взаимодействуют друг с другом элементы пользовательского интерфейса, но не предполагает в себе использования конструкций программирования - например, циклов, использование которых возможно благодаря встроенному в **QML** языку **JavaScript**.

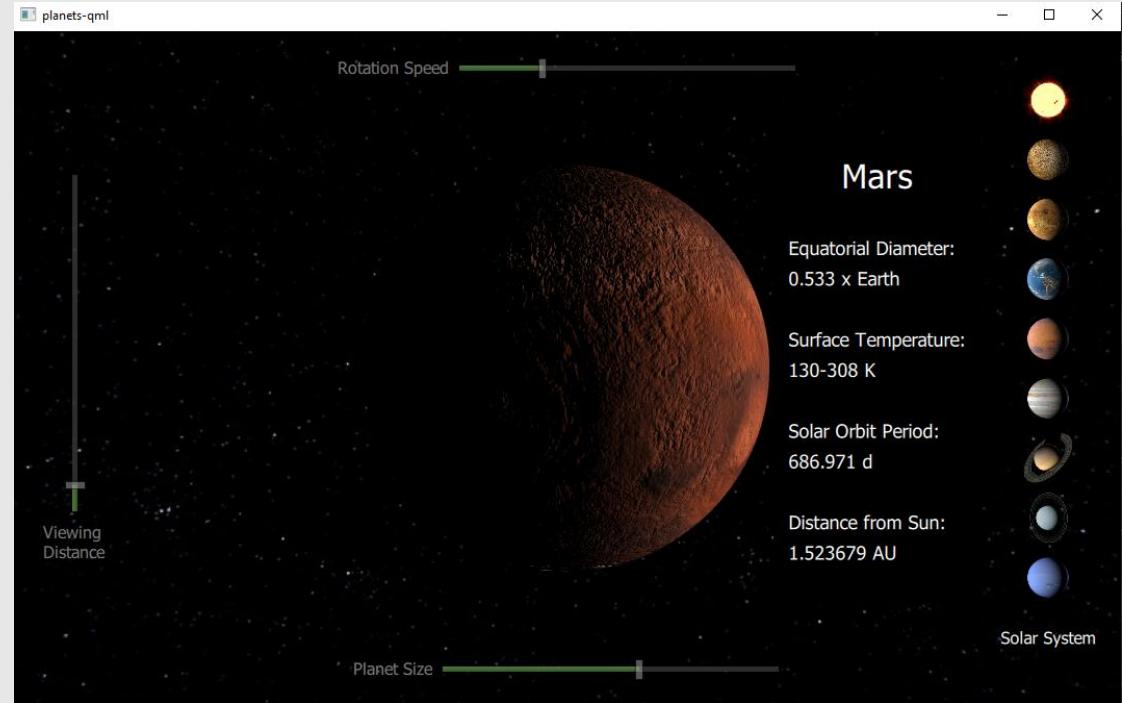
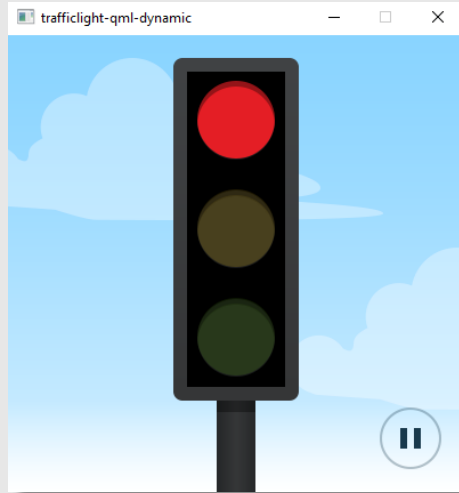
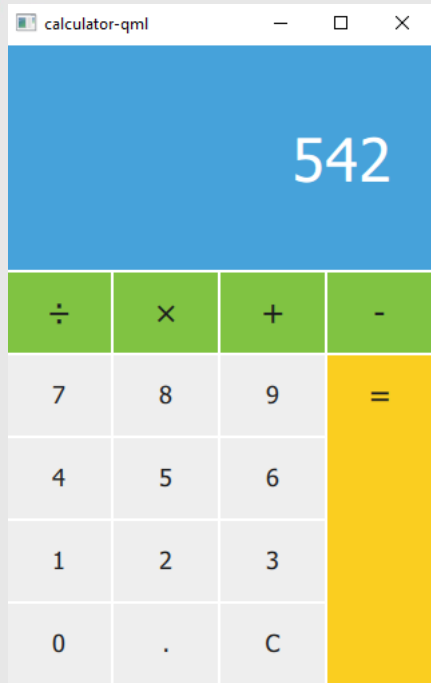
В **QML** встроен доступ ко всем имеющимся технологиям **Qt** - таким как *Qt/C++*, *QtWebEngine*, *Qt 3D*, *QtLocation*, имеется также и доступ к метаданным объектов - например, к свойствам, вызываемым методами (*invokable methods*), декларируемыми при помощи макроса `Q_INVOKABLE`, `Q_PROPERTY` и т. п.

Благодаря **QML**, дизайнер говорит с разработчиком программного приложения на одном языке, без дополнительных объяснений. Это дает возможность быстро создавать прототипы (**Rapid Prototyping**) программных продуктов.

Есть средства для конвертирования дизайна созданного в графических редакторах **Adobe Photoshop** и **GIMP**, **Figma** прямо в **QML**.

Если дизайнер предоставит прототип на языке **QML**, прототип уже является стартовой версией для готового приложения, на базе которого разработчики могут работать дальше.

4. Примеры QtQuick GUI



5. Модули QML

Модули, которые можно использовать в Qt Quick приложениях:

Основная функциональность **QML** сосредоточена в модуле **Qt QML**, который определяет и реализует язык и его инфраструктуру, а так же предоставляет интерфейсы *API* для интеграции **QML** с **JavaScript** и **C++**.

Название	Описание
Qt3D.Animation X.X	Набор элементов, предназначенных для анимации в Qt 3D
Qt3D.Core X.X	Основные возможности для поддержки 3D в режиме реального времени
Qt3D.Extras X.X	Набор дополнительных элементов для работы в Qt 3D
Qt3D.Input X.X	Элементы поддержки ввода пользователя для приложений, использующих Qt 3D
Qt3D.Logic X.X	Синхронизация кадров с движком Qt 3D
Qt3D.Renderer X.X	Поддержка 2D- и 3D-рендеринга с использованием Qt 3D
Qt3D.Scene2D X.X	Предоставляет возможность рендеринга содержимого из QML в текстуру Qt 3D
QtBluetooth X.X	Поддержка работы с беспроводными устройствами Bluetooth
QtCanvas3D X.X	Рисование в трехмерном пространстве способом, схожим с WebGL
QtCharts X.X	Набор простых в использовании элементов для отображения диаграмм и графиков
QtGraphicsEffects X.X	Набор визуальных элементов для создания и добавления графических эффектов
QtLocation X.X, QtPositioning X.X	Поддержка определения местоположения

6. Модули QML

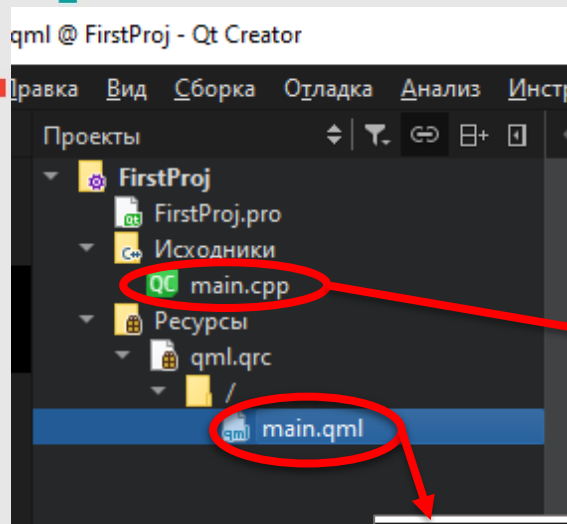
Название	Описание
QtMultimedia X.X	Набор элементов для работы с мультимедиа
QtNfc X.X	Поддержка устройств ближней бесконтактной связи (Ner field communication)
QtPurchasing X.X	Механизмы для покупки внутри приложения
QtQml X.X	Основной модуль для разработки приложений на QML. Предоставляет язык и инфраструктуру
QtQml.Models X.X	Набор элементов моделей данных
QtQuick.Controls X.X QtQuick.Controls.Styles X.X	Элементы управления и стили для создания графических пользовательских интерфейсов
QtQuick.Dialogs X.X	Набор элементов диалоговых окон
QtQuick.Extras X.X	Дополнительные элементы управления для графических пользовательских интерфейсов
QtQuick.LocalStorage X.X	Чтение и запись данных из/в базу данных SQLite
QtQuick.Particles	Набор небольших компонентов различного применения
QtQuick.Templates X.X	Набор не визуальных элементов, для создания графических пользовательских интерфейсов
QtQuick.VirtualKeyboard X.X	Модули реализации виртуальных клавиатур
QtWebSockets X.X	Поддержка протокола для двусторонней связи между клиентским приложением и удаленным компьютером (протокол WebSockets, реализованных поверх TCP-IP протокола)

7. Быстрый старт

Для начала работ с QML
понадобится
Qt Creator

Проекты ->
Создать ->
Qt Quick
Application

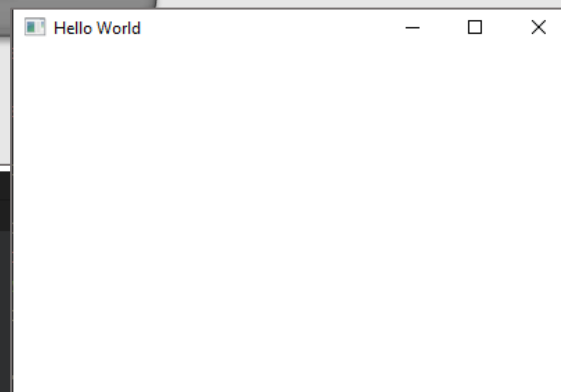
Запустить



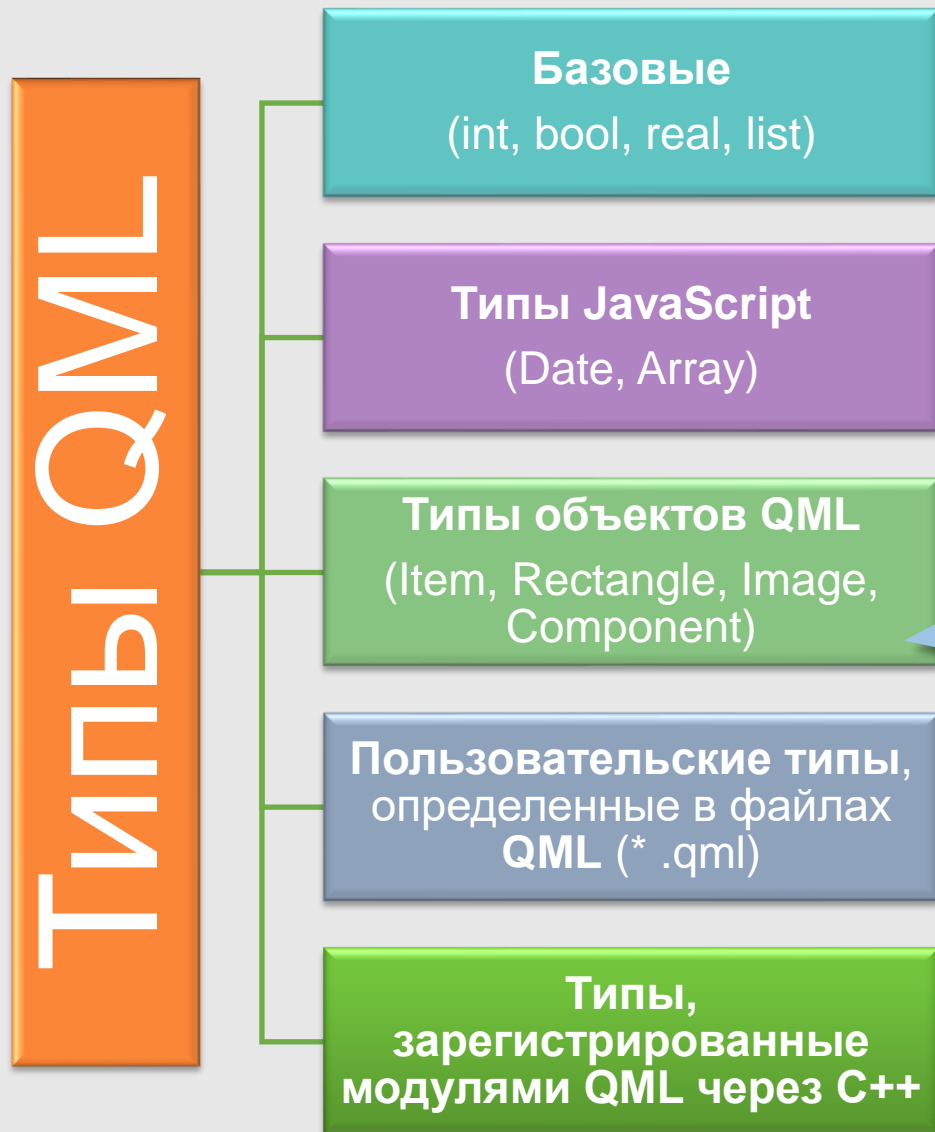
```
Инструменты Окно Справка
< > main.qml
1 import QtQuick 2.15
2 import QtQuick.Window 2.15
3
4 Window {
5     width: 640
6     height: 480
7     visible: true
8     title: qsTr("Hello World")
9 }
10
```

Директива **import** - аналог директивы **include** в C++. Здесь мы импортируем модули **QtQuick 2.15** и **QtQuick.Window 2.15**.

```
Инструменты Окно Справка
< > main.cpp
1 #include <QGuiApplication>
2 #include <QQmlApplicationEngine>
3
4 int main(int argc, char *argv[])
5 {
6     #if QT_VERSION < QT_VERSION_CHECK(6, 0, 0)
7         QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
8     #endif
9     QGuiApplication app(argc, argv);
10
11     QQmlApplicationEngine engine;
12     const QUrl url(QStringLiteral("qrc:/main.qml"));
13     QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
14         &app, [url](QObject *obj, const QUrl &objUrl) {
15         if (!obj && url == objUrl)
16             QCoreApplication::exit(-1);
17         }, Qt::QueuedConnection);
18     engine.load(url);
19
20     return app.exec();
21 }
22
23
```



8. Система типов QML



Базовым типом для всех визуальных элементов в *Qt Quick* является тип **Item**, который предоставляет общий набор свойств. Фактически он представляет собой прозрачный визуальный элемент, который можно использовать в качестве контейнера. Все остальные визуальные элементы в *Qt Quick* наследуются от **Item**.
(аналог **QWidget**)

9. Описание пользовательского интерфейса

Пользовательский интерфейс описывается как дерево элементов с их свойствами.

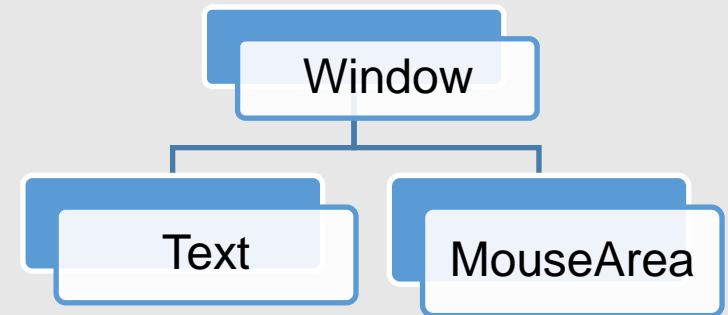
Каждый *QML*-файл должен содержать один корневой элемент. Например элемент основного окна приложения **Window**, (h = 480, w = 640 px). Все элементы *QML* выполняются по указанию их типа, содержимое элемента заключено в его тело, ограниченное фигурными скобками { }.

QML-элемент содержит свойства, которые задаются именем и значением в следующем виде:
name: value.

```
<QML-элемент> {  
    <name_1>: <value1>  
    <name_2>: <value2>  
    ...  
    // однострочный комментарий  
    /* многострочный  
       комментарий  
    */  
    <name_N>: <valueN>  
}  
//-----  
// альтернативная форма записи QML-элемента  
<QML-элемент> {<name_1>: <value1>; <name_2>: <value2> }
```

10. Редактируем “main.qml”

«Дерево QML-элементов»



```
import QtQuick 2.15
Window {
    id: wnd
    width: 640
    height: 480
    visible: true
    title: qsTr("Hello World")
    MouseArea {
        anchors.fill: parent
        onClicked: {
            console.log(qsTr("Bla bla bla. " ))
        }
    }
    Text {
        text: "Hello QML"
        font.pixelSize: 22
    }
}
```

“main.qml”

Получить доступ к *QML-элементам* можно при помощи их идентификатора **id**, который должен быть уникален. К элементу предка можно получить доступ без идентификатора - при помощи свойства-ссылки **parent** (как это делается для элемента **MouseArea**).

Сами свойства могут быть дополнены при помощи *JavaScript*(как это сделано со свойством **onClicked**).

11. Использование JS в QML

Функции JS можно интегрировать в QML-элементы

QML —это описательный язык и для реализации логики в программе он не пригоден. Для реализации логики в программе используется **JavaScript**.

Для реализации в QML сложных алгоритмов, лучше выбрать C++.

```
Rectangle {  
    width: parent.width / 5  
}
```

2. Присвоение описательной переменной

1. Вычисления с JS

Если выражение JS содержит более 2-х строк оно помещается в { }.

```
Rectangle {  
    width: {  
        var w = parent.width / 5  
        console.log("Current width:" + w)  
        return w  
    }  
}
```

```
Rectangle {  
    function maximum(a, b) {  
        return a > b ? a : b;  
    }  
    width: maximum(parent.width, 100)  
}
```

Можно поместить функции JS в отдельные файлы "*.js", а затем импортировать их в QML-файл элемента.

```
import "myfunctions.js" as MyScripts  
  
Rectangle {  
    function maximum(a, b) {  
        return a > b ? a : b;  
    }  
    width: MyScripts.maximum(parent.width, 100)  
}
```

Идентификатор доступа к файлу *.js со всем содержимым

И не забудем про директиву **.pragma myfunctions** в *.js. Это чтобы файл не включался более 1 раза.

12. Использование консоли

Отображение информации на консоли из *JS*

Метод	Описание
<code>console.assert()</code>	Проверка переданного лог. выражения на true. Если это так отображается сообщение переданное вторым аргументом.
<code>console.count()</code>	Отображение заданное количество раз ч.л.
<code>console.debug()</code> , <code>console.log()</code> , <code>console.info()</code> , <code>console.error()</code> , <code>console.warn()</code> , <code>print()</code>	Вывод информационных сообщений, которые могут быть использованы для отладки программы
<code>console.profile()</code> , <code>console.profileEnd()</code>	Используется для оптимизации кода. Включает и отключает профилировщик.
<code>console.time()</code> , <code>console.timeEnd()</code>	Замер времени от вызова <code>time()</code> до <code>timeEnd()</code>
<code>console.trace()</code>	Отображает цепочку вызовов <i>JS</i> (10 последних, по умолчанию)



13. Домашка # 1

1. Создание приложения QtQuick(см. слайд 7);
 2. Пример использования консоли в разработке приложения;
 - 3*. Создание отдельного файла “*.js” с набором функций: *max*, *min*, *sum* и использование этих функций в файле “*.qml” (см. слайд 11).
- 