QT Introduction

How to create a GUI using Qt and the Qt framework

By Matthew Hardenburgh

Agenda

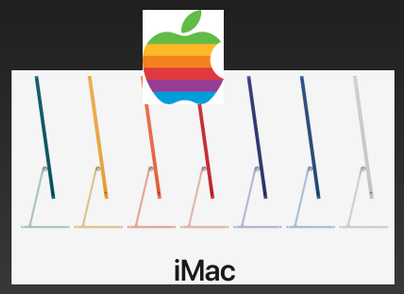
1. Motivation
2. Installing Qt
3. Creating a new project
4. The Driving Example and Guided Code Tour
    1. QML
    2. Qmake and the Qt build process
    3. Integrating C++ into QML
    4. Signals and Slots
    5. Qt's C++ API library

Motivation For ...

What is the motivation for the this presentation?

… Using embedded systems

- CSCI 430 hardware based projects. “Smart” mirrors, “Smart” alarm clocks and other “smart” devices.
- Embedded systems, such as the Raspberry Pi platform are fast and powerful.
- More companies and industries are shifting to ARM based platforms.



Popular hardware project was the “smart device” usually powered by a raspberry pi

Embedded systems have grown in capabilities in a very short time span.
For example, the Raspberry Pi 4 is a quad core, 64-bit ARM based system that can run full desktop Ubuntu and modern AI workloads (albeit at a reduced capacity).
The Nvidia Jetson Nano is another “Pi like” system with CUDA cores for increased AI and graphics capabilities.

ARM based embedded systems are a technology becoming more ubiquitous throughout the industry, therefore early exposure is good
The new Apple M1 chip in the MacBook, iMac and iPad Pro is ARM 64 based. The Microsoft Surface Pro X is also ARM based.
Both wireless VR systems, the VIVE Focus and Oculus Quest 2, use ARM based embedded processors.

... Learning a UI Framework

- We interact with GUIs everyday
- It makes your product more “attractive”
- It can make your product more intuitive

Motivation for learning a UI framework:

From apps on your smartphone to the point of sale software that takes your card at checkout line at the grocery store, most software we interact with has a GUI

Creating your algorithm or program is only half the battle
Most software isn’t ran on the command line
A well designed user interface makes your product more attractive to customers

It can make communicating information more intuitive such as voltage levels, current velocity or altitude.

... Learning QT

- It is a robust and feature rich UI framework technology
- Easy integration with C++ and Python.
- Develop once, deploy anywhere paradigm.
- It is used in the software engineering industry by many companies.
- Their documentation is AWESOME!

Motivation for learning to use QT

In the Software Engineering course, a popular hardware project was the “Smart Device” and these usually have a UI component.
My group for CSCI 430 used a python based GUI framework called Kivy.
Kivy was great, but in my opinion it lacked features and robustness.

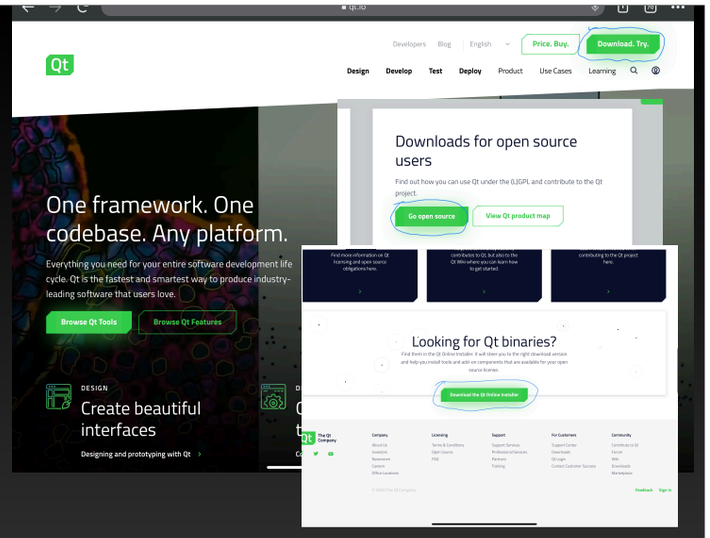
Qt is multi platform. You can prototype and develop on a desktop and then deploy production binaries to your target device
QT can also be used on Android and iOS to create apps for smartphones as well. Another popular CSCI type of project.
Everything learned in this presentation can be used to create UIs for desktop, embedded systems, microcontrollers, iOS and Android software.

Many companies in the industry use QT. Mercedes-Benz, Panasonic, AMD (for graphics UI software), LG (for their TV’s webOS), and Yuneec drones to name a few.

In assisting me in creating the example application for this presentation I used their documentation, stack overflow/stack exchange, and their example projects. Their analog clock, coffee example, Qt Quick controls examples, and the Alarms with Qt Quicl

Installing QT

- The installation process varies slightly for every platform.
- Qt offers Qt Design Studio and Qt Creator
- Qt Designer is their QML editor
- Qt Creator is their IDE.



I installed the Qt creator for Linux, but it is available for Mac and Windows.

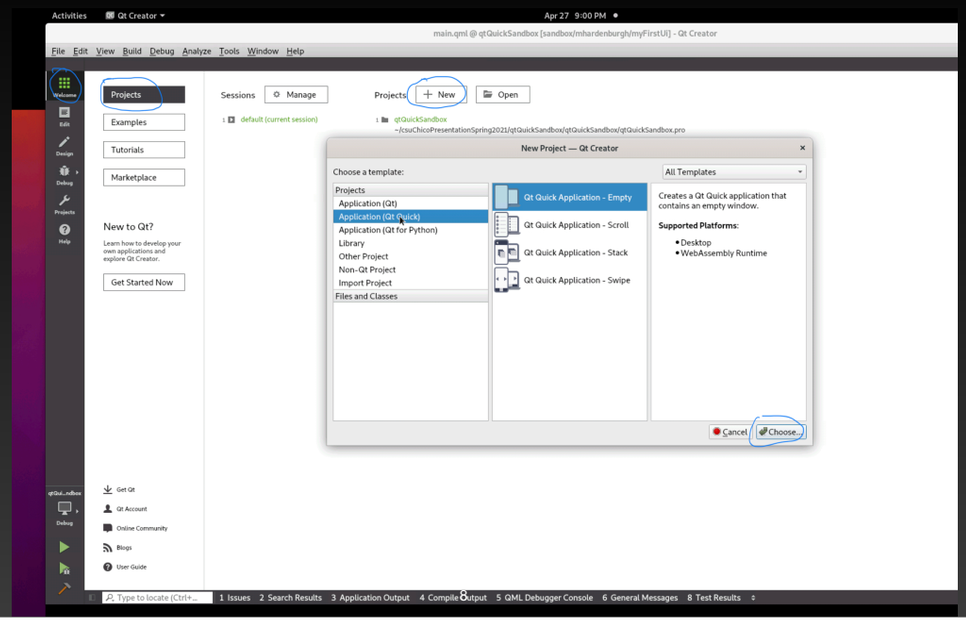
I think the command for Linux was ./runQtInstaller, double check that

When I needed help installing Qt or in general, I was able to google the answer and find it on the forums or stack exchange.

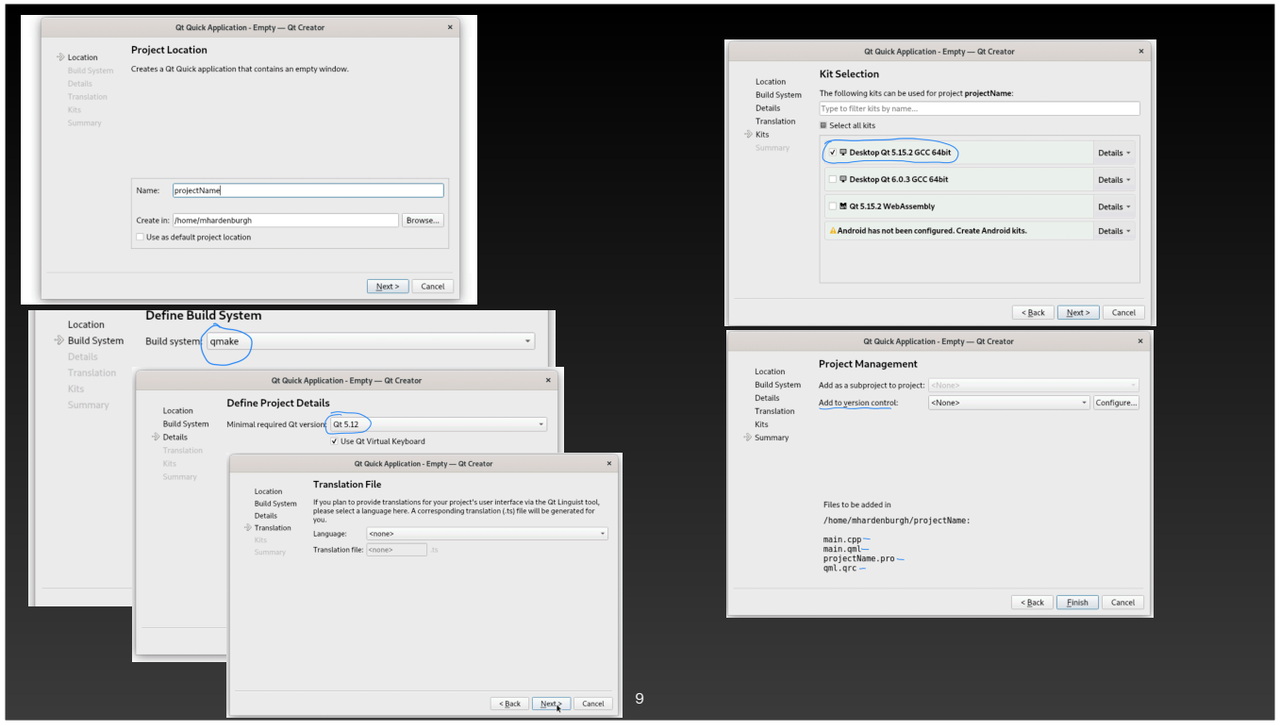
Many Udemy and Free YouTube tutorials for Qt.

Qt Design studio is a “What You See Is What You Get” advanced visual QML editor
Qt Creator is a traditional IDE that has a text editor and GUI layout and QML forms editor.
Using either is not required, but may be helpful.
I found that the Qt Creator IDE is great for designing the GUI layout because you can see in real time the changing QML code, this helped me in understanding QML more.
Qt Designer costs money while Qt Creator does not (for research and educational purposes).

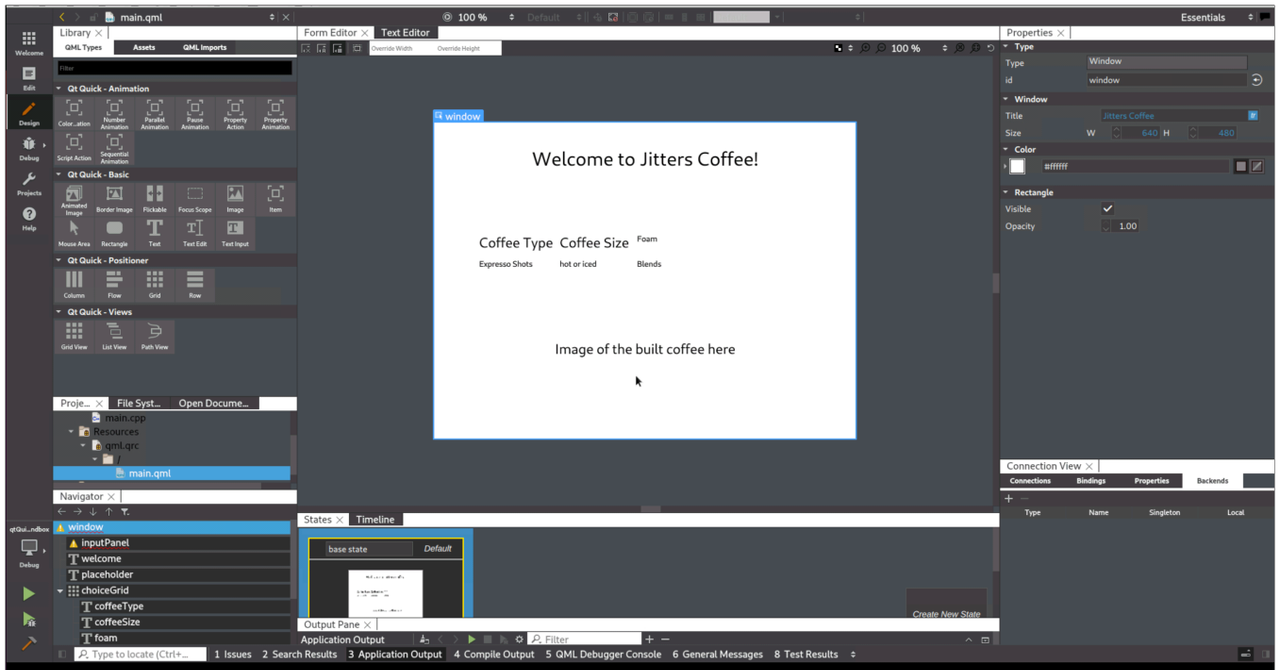
Creating a new project in Qt Creator



Qt quick is geared to mobile applications but can be used in desktop or embedded applications as well. Makes for easy and quick prototyping.
Qt widgets is geared to desktop applications.



Choose a project location and a project name
Chose a build system. I chose qmake because that that is what I am familiar with, but if you are used to another build system such as cmake you can chose another one.
The build system is used to generate make files and build rules from Qt .pro and .pri files.
Project details is where you chose the minimum required Qt version to build the project. I left this a default
I left the translation file at default
For kit selection I chose Qt 5.15.2. Currently Qt 6 is experimental.
For version control, I chose none because I had already created a git repo, but you can chose from git, svn, mercurial, etc ...
At the end the project wizard creates a main.cpp, main.qml, a top level .pro file and .qrc file. Qrc is a q resources file.



This is the Qt creator IDE with the integrated Qt designer interface.
You can you the design section to “Design” you QML files but I never used it
On the bottom left there is the build and run buttons for your program.
The build button runs qmake
I have created this UI using the visual editor and occasionally had to modify some things with the text editor.

Driving Example
“Smart” Coffee Machine

- Specifications. The customer wants to chose...
  - ... from a variety of coffee types (Americano, Cappuccino, Latte, Espresso, Macchiato, etc...)
  - ... size: small (8oz), medium (12oz), large (16oz)
  - ... how much foam: none, a little, normal, a lot
  - ... how many shots of expresso. 1 to 3
  - ... hot or iced
  - ... coffee blends: decaf, blonde, medium dark, dark

In the past, students taking the CSCI 430 course have done projects that have UIs such as “smart” devices.
To drive this introduction to qt, I will use a fictional product. A “smart” coffee machine called Jitters Coffee.
In this example product, customer research has already been completed and we have a list of specifications potential customers want to see in this product.

QML

- QML is a description style language similar in concept to CSS.
- Basic Syntax:
  - Objects,
  - Object properties and attributes
  - Object children
- Items
- States
- Animations

QML is used to describe the user interface. Objects are arranged in a hierarchical tree type manner, that is the outer parent objects can contain inner child objects, and those children can contain their children, etc. Ie rectangle objects contain buttons.
QML can also be used to describe the different states of a UI
QML can support JavaScript functions. (see line 14 in “CoffeeOrder.qml”)
C++ objects can be integrated into QML with some compiler setup (see main.cpp, coffee.cpp and coffee.h)
“Import” syntax is the same functionality as C++’s include

QML has many default primitive object types such as rectangles, buttons, grids, etc. (HotOrIcedDialog.qml)
The documentation for these primitive are listed on the Qt website and is usually the first hit on a google search.

The programmer can create custom object types and import C++ objects into QML as well (coffee.h is an example of C++ and “CoffeeOrder.qml” is a qml example)
QML Objects are defined by starting with a capital letter followed by a pair of curly brackets (Show CoffeeOrder.qml line 5)
QML objects can be instantiated multiple times by creating the same object type with different id’s (show lines 26 and 40)
An object ID is a special property used to differentiate objects of the same type between each other

Object attributes are...attributes. Some are give by the language by default and some are defined by the programmer.
EVERY QML object has the id attribute. (Show the grid and buttons near line 244 in CoffeeItem.qml)
Programmer defined attributes are called properties.
For some default primitives such as the Grid, Button or Text object, have pre-defined properties such as the x and y position that can be used straight away.
Custom properties can be defined with the “property” keyword with the unit (int, string, color, etc...)

Properties are type safe, meaning only a string can be assigned to a string
The grammar for using a property is property:value
The properties are all “public”
Properties of an object can be accessed by other objects using the dot “.” Operator (CoffeeItem.qml line 244)
On object’s property can be setup to be dependent on another object’s property. (show example)
Property aliases hold references to properties inside objects. Like how a C++ pointer holds a reference to another object (CoffeeOrder.qml and line 300 in CoffeeItem.qml). A property alias can be used to connect a newly created property as a direct reference to an existing property

Objects can have children objects inside and those children can have children. QML is setup in a hierarchical tree type fashion.
The dot operator is again used to access the children objects’s properties inside the parent objects. (Line 20 in CoffeeOrder.qml)

An Item is a base type for all Qt Quick objects, such as the rectangle.
The base object has basic attributes such as X/Y position, width/ height and anchoring
I used the Item class to build custom objects such as the “CoffeeOrder” and the “FoamAmountDialog”, “CoffeeSize” (see line 29 in CoffeeItem.qml and CoffeeOrder.qml, 303 in CoffeeItem.qml and FoamAmountDialog.qml, and 339 CoffeeItem.qml in CoffeeSizeDialog.qml)

States are...states of a particular object.
An object can change to different states when an event happens, such as a button click or mouseover.
Each state defines an action of “batched changes” that will occur when that state is active. (see the states at line 56 in CoffeeItem.qml)
States for a given object are collected under a special “states” child object which in turn has child objects called “State” to define each state of the root
Each “State” has a name and the property values that are going to change (refer to the states at line 56 in CoffeeItem.qml)

An animation is a way to transition between windows, or in my example show that the coffee order is being executed. (Refer to line 159 in CoffeeItem.qml)
Animations change the properties of objects
I used the “SequentialAnimation” object to animate changes in state with the “PropertyAction” and pause in between with “PauseAnimation”

Qmake and the Qt build Process

- What is qmake and why use it?
- Qt project file
- Qt resource files
- Qmake vs cmake

Knowing how to write makefiles and understanding them is a good skill to have
However, writing makefiles for a project of large complexity is hard.
If your project is complex with links into QML, FPGA code, etc. Writing each makefile and constructing the build system from scratch is hard
Using a build system tool such as qmake can greatly simplify creating the makefiles for large projects.
A build system tool can generate makefiles automatically.
qmake is Qt's build system tool for generating makefiles that can build the C++ code and link to the QML code.
The disadvantage of using a build system is that the makefiles kind of becomes a black box and you don't know what 100% is going on.
qmake doesn't even have to be used for qt projects. It can be used for normal software projects as well.
Qmake can be used to generate build files for Microsoft visual studio as well

For the example, I created a new project using the new project wizard and the wizard created a project file
An add file wizard can be used to add to the files to the project and the wizard updates the .pro file
When not using the IDE and wizards, the developer can create a .pro file (see the qtQuickSandbox.pro)
Some special syntax:
"SOURCES" (in line 13) are the cpp files that contain source code
"HEADERS" (line 33) are the header files
"RESOURCES" (line 17) is the resources file that references the QML files to bring them into the build system.
"TARGET" (line 29 and 30) is used to set the project name. Wizard generated files set the target to be the name of the .pro file, but with TARGET = myProjectName you can set the name how you want.

The Qt resource file ends in .qrc, so for my example its called qml.qrc
When I used the project wizard to create the project and add files to my project, it automatically updated the qrc file
You can see that all of my QML files are referenced in the qrc file.
This allows the UI to be compiled and linked into the C++ source

Cmake is a more general build system tool and not Qt specific.
Cmake can be used for building Qt projects, but some work is required for Qt, dependencies.

Integrating C++ into QML

- Add source and header files to project
- Expose C++ to the QML runtime
- Import your C++ object into QML

First step is to add the C++ source and header files to the .pro file. If you use the IDE wizard, it automatically takes care of it, otherwise you have to do it manually. (see qtQuickSandbox.pro lines 13 and 33)
In the .pro file, lines 6 and 7 registers the coffee class as a QML type that can be imported using “import coffee.order 1.0”

On the C++ side

Use #include <qqml.h> and inherit from the QObject class so that following QML macros can be used
Q_OBJECT defines the class as a Qt object (coffee.h line 16)
Q_PROPERTY is a macro that defines a QML object property for C++ object and what reading and writing to it does. (coffee.h line 17)
QML_ELEMENT makes the class available in QML(coffee.h line 17)
Q_INVOKABLE makes a class function accessible via QML (coffee.h line 29)

For my example I used “Import coffee.order 1.0” (line 7 of CoffeeItem.qml)
I instantiated on line 24 so I would be able to use it (line 23 of CoffeeItem.qml)

Signals and Slots
Qt’s Event handler system

- Introduction
- Signals
- Slots
- Connecting signals and slots

Signals and slots is Qt’s way of handling events and notify other objects that the an event has occurred

An object can emit a signal when the object’s state has changed in a way that other objects should know. Such as a button being pressed.
Signals are generally emitted from the objects that define them
The object emitting the signal care which slot or slots that signal is connected to
Lets look at some examples of signals

- C++ signals for the coffee example are defined in the “Coffee” class under “signals:” (see coffee.h:56)
- The signal “coffeeTypeBackendSignal(...)” emits a Qt string to a slot when called. (See emit on line 24 of coffee.cpp too)
- Signal “coffeeTypeSignal” defines a signal in QML (see line 10 in CoffeeType Dialog.qml) for the CoffeeTypeDialog object
- That signal is emitted with a string when a button is clicked in the dialog box (see line 30)

Slots are used for receiving signals.
They are basically functions that are called to do something when a signal is emitted

- C++ slots are defined in the Coffee class under “public slots:”
- The slot coffeeTypeSlot is an example of a slot (see line 22 of coffee.cpp).
- It accepts a QString and sets that QString to an internal variable m_coffeeType emits a signal to let others know that variable has changed.
- QML slots different and can be implicitly defined (see line 300 in CoffeeItem.qml)
- In this case “onCoffeeTypeSignal” means to assign coffeeTypeAlias to the emitted coffeeTypeString.

To connect C++ signals and slots to each other and to QML signals and slots the function QObject::connect(...) is used. (see main.cpp line 41)
This is typically done at the at the beginning of programs, shared library files, or where the object is instantiated.

To connect QML signals/slots to C++ signals/slots the QML must first be “loaded” into the C++ program so that it is aware of the QML signals/slots you are referencing (see lines 23 and 30 in main.cpp)

Qt's C++ API Library

#include <QtGlobal>

- The goal of the API is to make applications portable between all supported platforms
- Ex: QString, qint, qlong, etc are defined the same for Mac, Windows, Linux, etc
- Some basic standard Qt functions such as qMin(), qMax(), qAbs() are defined
- The API is very extensive and I only covered a small fraction of it.

https://github.com/mdhardenburgh/csuChicoPresentationSpring2021.git

Q & A

Questions about the example or presentation ?
Questions about life after college ?
Question about software engineering ?
It’s an open floor