

C++, Qt Core & QML

Inhoud

SAMENVATTING.....	3
SUMMARY (ENGLISH)	4
INLEIDING.....	5
MOTIVATIE.....	6
HET QT ECOSYSTEEM	7
AAN DE SLAG	8
HELLO QML.....	10
JAVASCRIPT IN QML	14
QT IN PYTHON	15
C++	16
HEADER FILE	17
SOURCE FILE	20
OBSERVATIES	23
SLOT	24
BRONNEN	25

Samenvatting

Dit document bevat een verslag over mijn ervaring met C++, Qt Core en QML. Lezers kunnen meer te weten komen over hoe ze aan de slag kunnen gaan met de IDE (Qt Creator), een eerste project met QML kunnen aanmaken en hoe de link kan worden gelegd met een C++ datamodel.

Dit document voorziet meer duiding en context voor het project dat lezers kunnen terug vinden op <https://github.com/dietbrand/graduaatsproef/>.

Summary (English)

This document contains a report about my experience learning and using C++, Qt Core and QML. Readers will learn more about getting started with the IDE (Qt Creator), setting up a QML project for the first time and find out how a link can be made with a C++ dataset.

This document provides more information and context regarding the project readers can find on <https://github.com/dietbrand/graduaatsproef/>.

Inleiding

De Graduaatsproef van de opleiding Graduaat Programmeren bestaat uit het herschrijven van een deel van het Projectwerk met behulp van een omgeving en/of een taal die niet in eerste orde tijdens de opleiding aan bod kwam. Het projectwerk in kwestie betrof een groepsopdracht voor AllPhi en belastte ons met het maken van een front- en backend voor het beheer van een vloot bedrijfswagens, bestuurders en tankkaarten. De opgeleverde backend bestaat uit een .Net Core API en een MySQL database. De frontend maakten we in React met React Router.

Ik koos ervoor om een alternatieve frontend te maken met behulp van C++, Qt Core en QML. Hoe die technologie precies in elkaar zit kan je in dit rapport terugvinden.

Motivatie

Vorige zomer bezocht ik de SIGGRAPH beurs in Vancouver. SIGGRAPH is een tak van het ACM en specialiseert zich in computer graphics en interactieve oplossingen. Daar werd al snel duidelijk dat een kennis van low level programmeertalen zoals C++ nog steeds een grote rol speelt. Populaire televisieprogramma's, films en animatiefilms gebruiken een combinatie van artistieke programma's en achterliggend worden die gedreven door deze programmeertalen. Enkele voorbeelden:

- In de Pixar animatiefilm Turning Red gebruikte men een op maat gemaakte generator van menigtes. Elke persoon in die menigte was een object met eigenschappen dat individueel kon gestuurd worden of in tandem met andere nabije objecten kon samenwerken.
- In de Star Wars serie The Mandalorian maakt men gebruik van Virtual Production. Scenes bestaan niet langer uit grote groene schermen waar acteurs voor staan maar LED volumes de grootte van een cinemascherm. Op die schermen wordt dan de scene zelf geprojecteerd. In real time kan men deze scene dan aanpassen waardoor een groot deel van het werk in post productie wegvalt(*). Dit alles gebeurt met behulp van Unreal Engine en gebruikt C++ om de intelligentie te drijven.

Bij de uiteenzetting van de graduaatsproef kregen we enkele mogelijkheden aangereikt, waaronder Qt met C++ voor de frontend. Dankzij ervaringen met Kubuntu (Ubuntu met de KDE Plasma desktop) hoorde ik wel al eens van Qt, maar had ik er nog geen ervaring mee. Ik wist evenmin dat Qt op basis van C++ werkte en daarom was de keuze voor mij snel gemaakt.

C++ heeft iets mysterieus en mythisch. De manier waarop ze wordt voorgesteld en vergeleken met andere programmeertalen zoals C#, Python of JavaScript maken duidelijk dat dit geen taal is die je zomaar leert. C++ duikt diep in je computer om zijn werk te doen en dat kan je voelen als je geconfronteerd wordt met de manier waarop het met bv geheugen omgaat.

Hieronder volgt een verslag over mijn ervaringen met deze technologie.

(*) Hoewel dit veel tijd kan besparen, is de technologie gloednieuw. Om correcties na de opnames aan te brengen (post productie) moet de scene vaak opnieuw worden opgenomen.

Het Qt ecosysteem

Qt begon in 1991 als een software project van de Noren Eirik Chambe-Eng en Haavard Nord. In 1994 richtten ze Quasar Technologies op wat later Troll Tech werd en uiteindelijk Trolltech. In 2006 volgde de beursgang en in 2008 werd het bedrijf overgenomen door Nokia voor 106 miljoen Euro. Het bedrijf werd omgedoopt tot Qt Software en later tot Qt Development Frameworks. Nokia verkocht in 2011 en 2012 de hele portfolio aan Digia. In 2014 richtte Digia de Qt Company op en in 2016 volgde de beursgang.

Qt maakte oorspronkelijk gebruik van de eigen Qt Free Edition License. Versie 2.0 lanceerde onder de Q Public License maar geen van beide licenties bleken echt compatibel met de GPL. Dit werd pas echt een probleem toen KDE, gebaseerd op Qt, aan populariteit begon te winnen en de open source gemeenschap vreesde dat deze toch wel zeer populaire desktop omgeving een gesloten karakter zou krijgen. In 2000 lanceerde Qt versie 2.2 onder de GPL v2 licentie. Verder werden er speciale afspraken gemaakt met de KDE gemeenschap waardoor het open karakter van de desktop in stand kon blijven. Het bekendste KDE project, de Plasma Desktop wordt tot vandaag gebouwd met Qt.

Qt is een verzamelterm voor enkele verschillende technologieën:

- Qt/Qt Core: Dit is het framework dat bovenop C++ draait. Het biedt een enorme waaier aan functionaliteiten die bij het compilatieproces worden vertaald naar C++ en zo naar een applicatie die zowat overal kan draaien.
- QML: De Qt Modeling Language is zowel een taal als een library die bovenop het Qt Core framework draait. Het voorziet een declaratieve syntax om componenten aan te maken waarbij je kan kiezen om met JavaScript of C++ de koppeling te leggen met onderliggende datastructuren. Een sterk argument om QML te gebruiken (ipv Qt Widgets) is dat het op elk systeem kan draaien dat een scherm heeft. Naast desktop en laptops is het ook perfect geschikt voor mobiele toestellen maar ook embedded devices zoals aanraakschermen voor koffiemachines of domotica apparaten.
- Qt Widgets: De voorloper van QML en voornamelijk geschikt voor het maken van desktopapplicaties.

Aan de slag

Je kan en mag Qt gratis gebruiken onder een open source licentie (LGPLv3, GPLv2 of GPLv3). Verwacht wordt dat de waarde die je voor jezelf of organisatie creëert, terugbetaalt door bij te dragen aan de open source gemeenschap of door een licentie voor een van de Tools aan te schaffen.

Qt steekt de installatietool voor hun gratis software niet ver weg maar ze is misschien niet direct te vinden. Vast staat wel dat je een account moet aanmaken. Dit kan via de website zelf (<https://login.qt.io/register>). Eens geregistreerd kan je terecht op <https://www.qt.io/download-open-source> waar je meer kan lezen over de open source filosofie. Wat je dan nodig hebt is de Qt Online Installer.

De installatietool biedt de mogelijkheid om volgende applicaties te installeren:

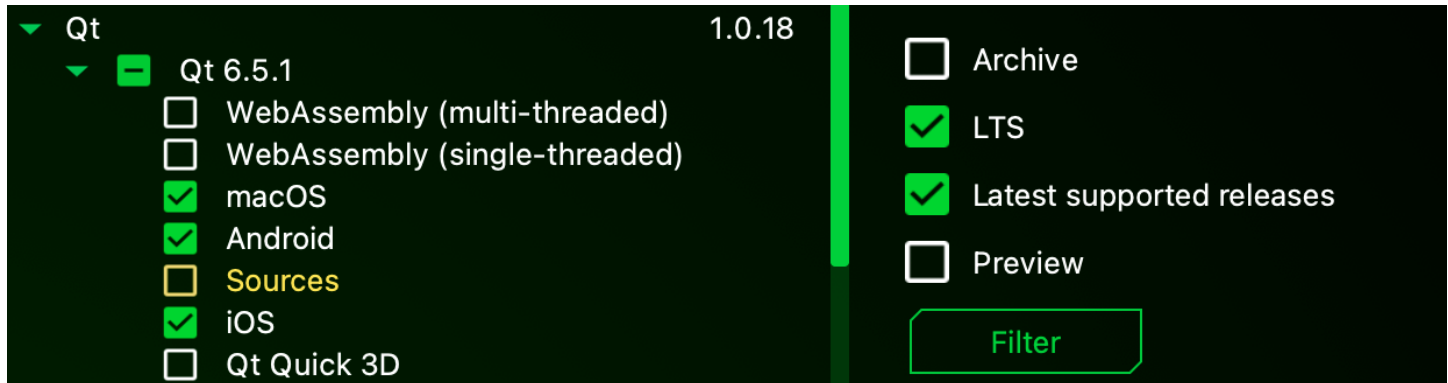
* Qt Design Studio

* Qt

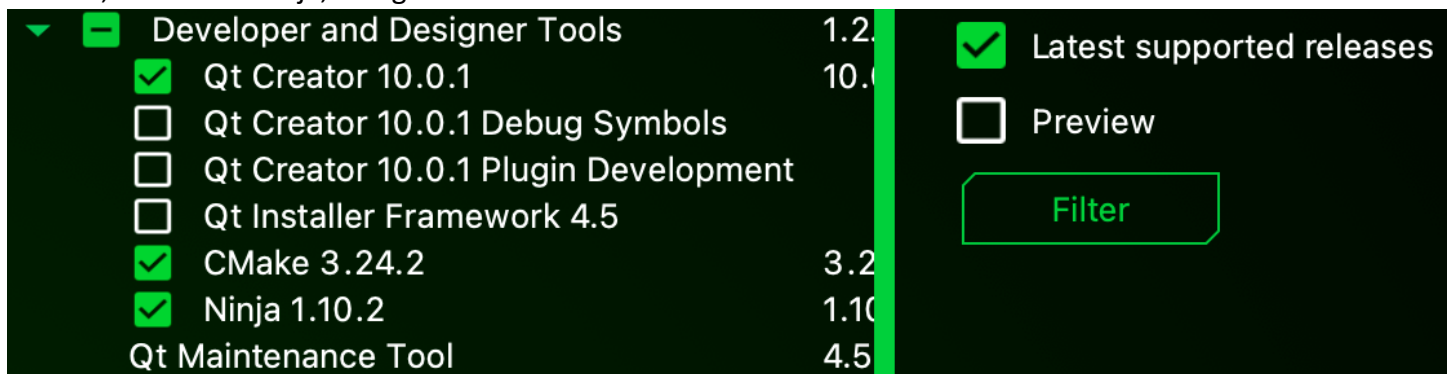
* Developer and Designer Tools

Als beginner heb je niet alles nodig. Elke ondersteunde versie van Qt is in zijn geheel tientallen gigabytes groot. De tool waarmee je Qt installeert kan je later opnieuw gebruiken om zaken toe te voegen of te updaten. Voorlopig volstaat het volgende zaken aan te duiden:

Onder Qt kies je de versie die je wenst. Een veilige keuze is de laatste versie die beschikbaar is onder de Qt noemer (niet de Preview). Maak deze open om de workloads te kiezen die je wil gebruiken.

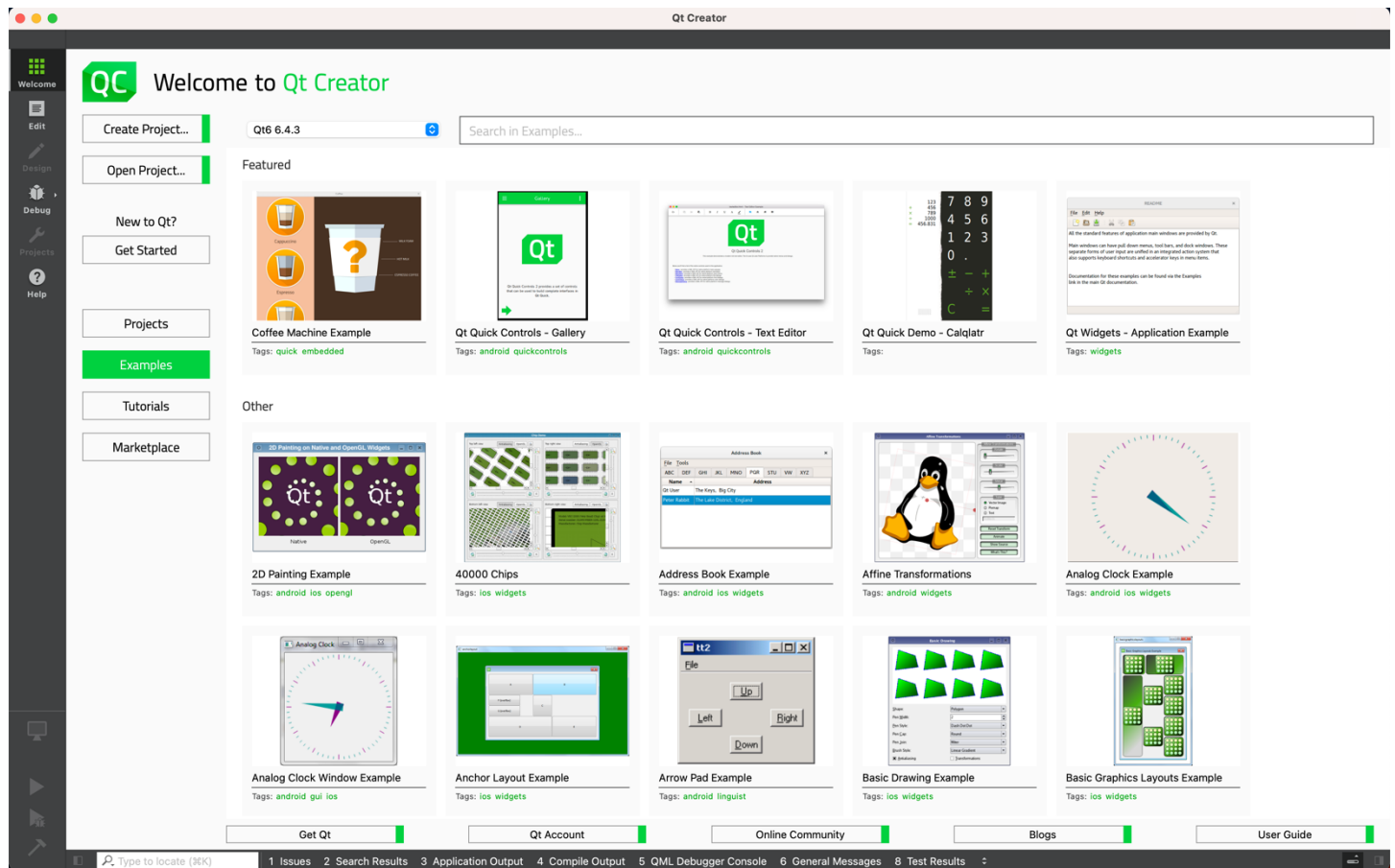


Onderaan de verschillende versies van Qt vind je "Developer and Designer Tools" terug. Kies hieruit de Qt Creator, CMake en Ninja, een generator voor CMake.



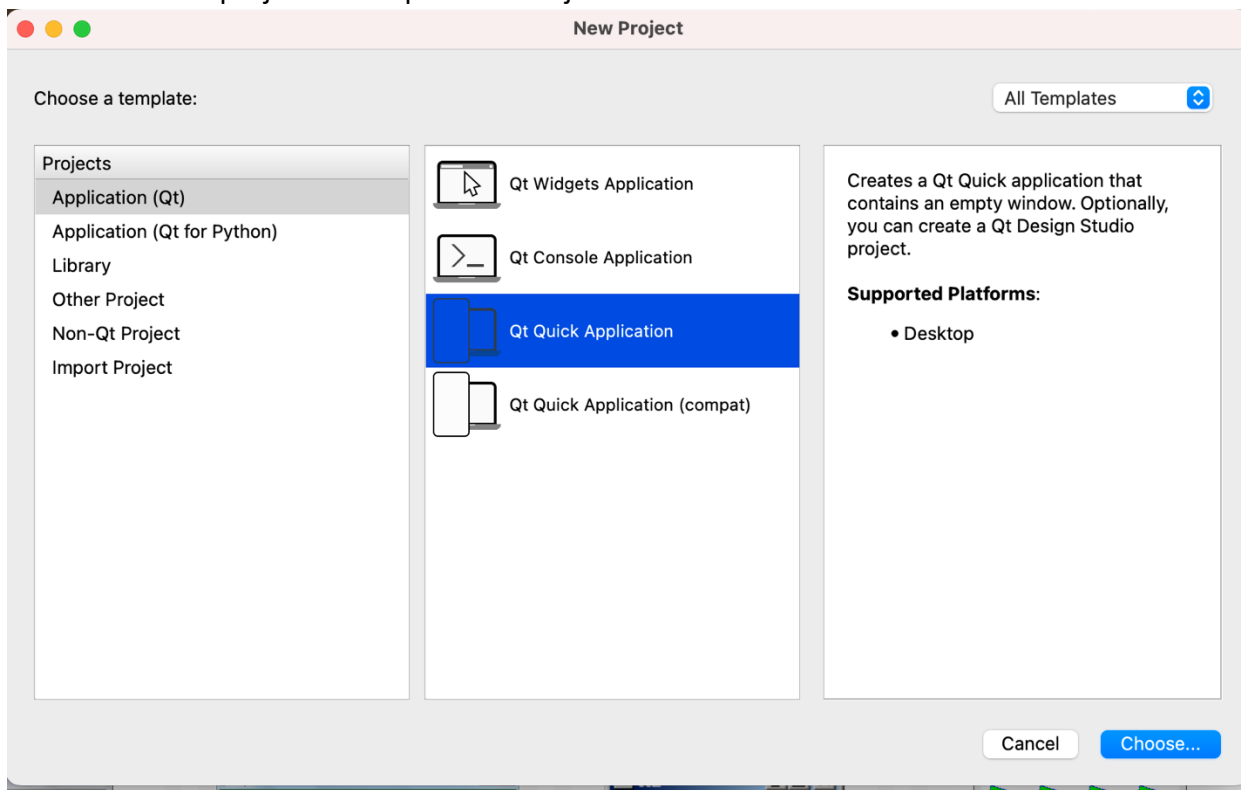
We kiezen niet voor Qt Design Studio maar voor Qt Creator. Design Studio is zeker geschikt voor het ontwikkelen van gebruikersinterfaces maar voor eenvoudige applicaties volstaat de Qt Creator.

Start Qt Creator wanneer de installatie is voltooid.



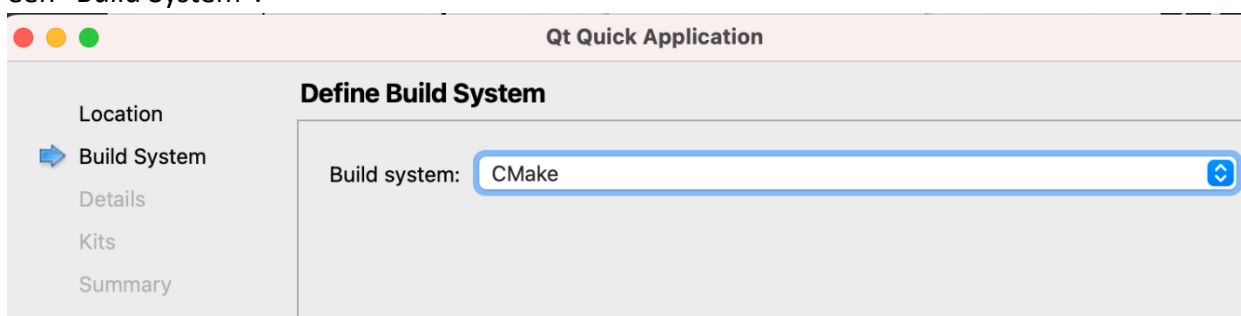
Hello QML

Start een nieuw project door op Create Project te klikken.



Kies voor een Qt Quick Application. Je kan zien dat Qt niet exclusief met C++ werkt maar ook Python ondersteunt. Zoals eerder aangehaald zouden we in dit project ook JavaScript kunnen gebruiken om logica toe te voegen aan de applicatie.

Kies een geschikte naam en pad voor het opslaan van het project en klikken op Next. Nu volgt de keuze voor een "Build System".

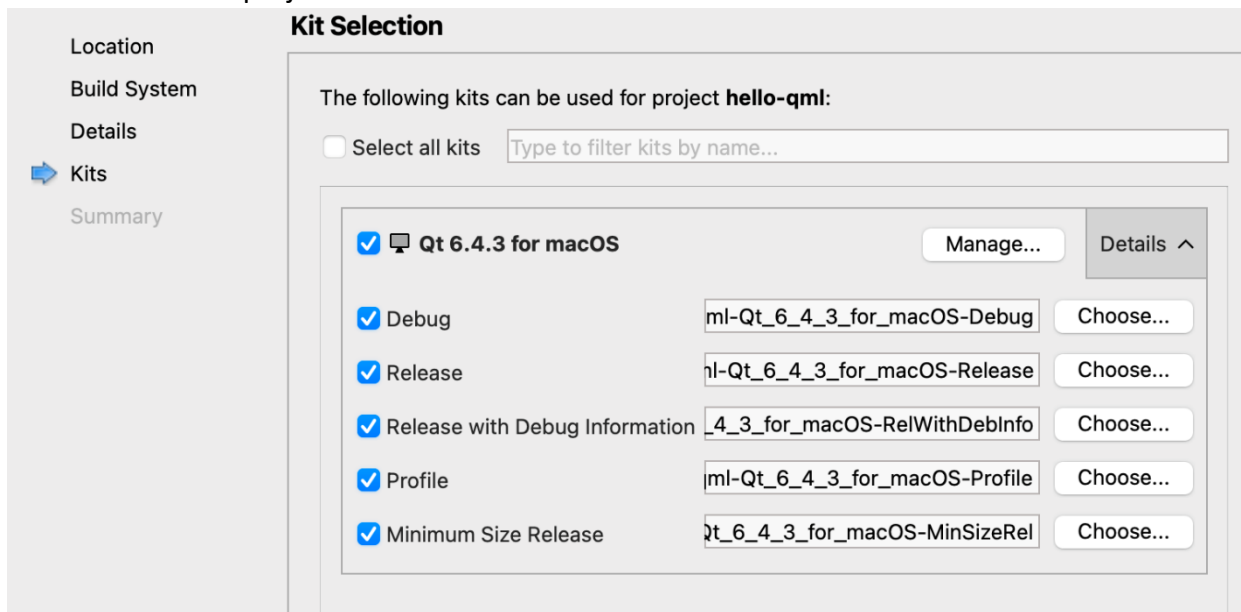


Een build system bepaalt het proces om software te compileren en klaar te stomen voor gebruik door executables of libraries te maken.

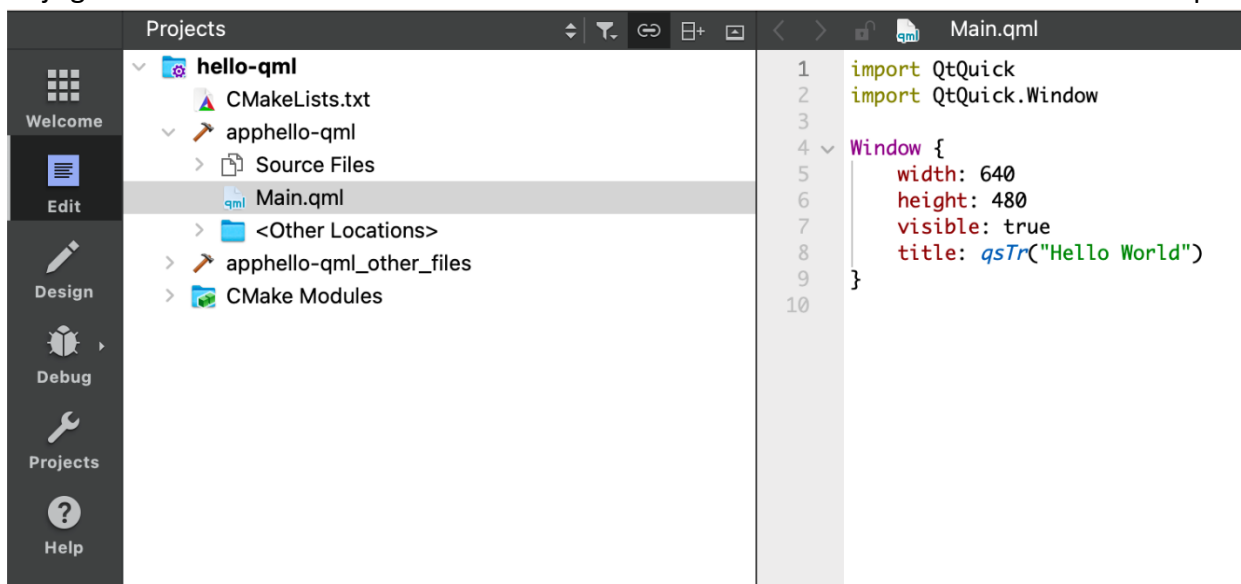
Sinds Qt 6 is CMake de standaard. Andere keuzes die je zal zien zijn Qbs of Qmake. Qt besloot in 2018 en 2019 geen nieuwe functionaliteiten toe te voegen aan de twee laatste en volledig in te zetten op ondersteuning voor CMake.

In het Details venster kan je kiezen om het project ook geschikt te maken voor gebruik in de Design Studio en ondersteuning voor een virtueel toetsenbord mogelijk te maken. Voor deze demo is geen van beide vereist.

Het volgende venster spreekt over de kit waarmee de applicatie zal worden uitgerust. Kits kan je vergelijken met ontwikkelomgevingen of environments. De standaard keuzes zullen o.a. een Build en Release environment voorzien voor het project.



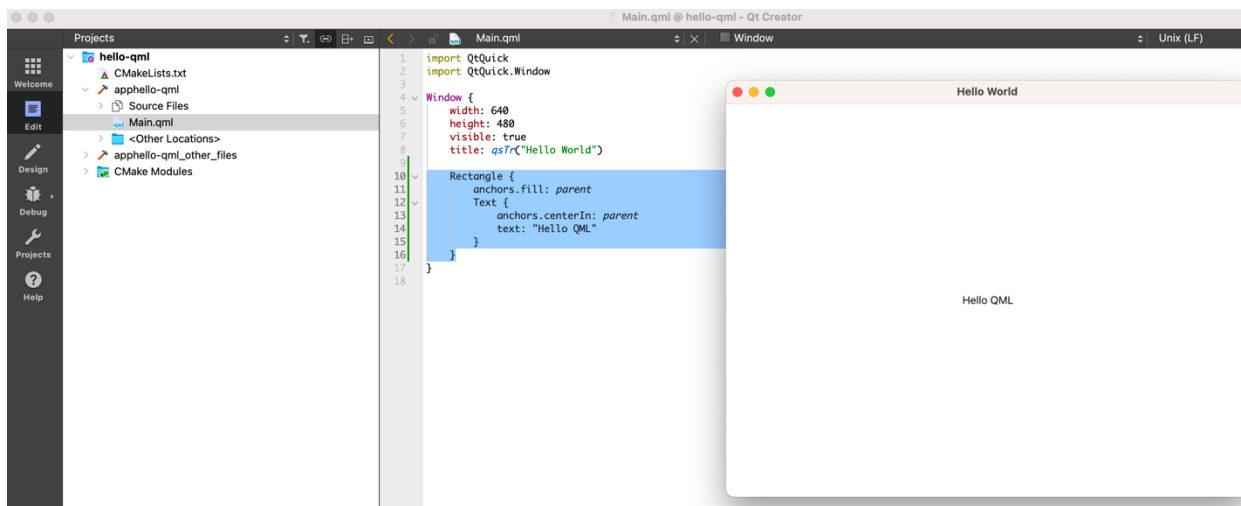
Wijzig hieraan niets en voltooi de wizard. De editor wordt nu zichtbaar met daarin Main.qml.



Voeg nu volgende code toe tussen regel 8 en 9 (voor de sluitende accolade):

```
Rectangle {
    anchors.fill: parent
    Text {
        anchors.centerIn: parent
        text: "Hello QML"
    }
}
```

Wanneer je onderaan links op de groene pijl klikt, start je applicatie en het venster (de Window).

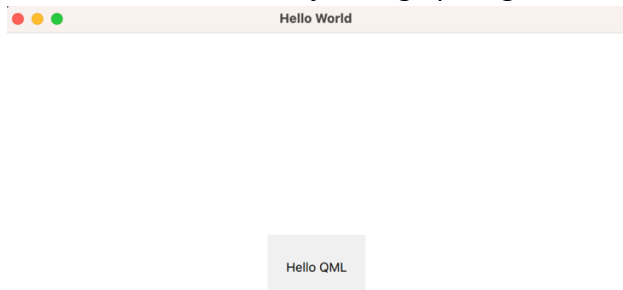


QML code schrijf je in een boomstructuur. Het Window element is de root of parent van de applicatie. Elk element heeft attributen zoals width, height, anchors en meer en zo goed als elk element kan op zijn beurt kind-elementen bevatten (zoals het element Text in het element Rectangle).

Een essentieel element voor het tonen van vormen is de Rectangle. QML kent geen andere basis vorm waarmee een vlak kan worden getekend. Geef de Rectangle wat meer stijl met behulp van volgende code:

```
Rectangle {
    width: 100
    height: 65
    color: "#f0f0f0"
    anchors.centerIn: parent
    Text {
        anchors.centerIn: parent
        text: "Hello QML"
    }
}
```

Het resultaat, wanneer je terug op de groene knop klikt, valt dadelijk op.



Het Qt framework voorziet een eigen manier om het Observer patroon toe te passen: signals en slots. Een signal is een methode van een object dat een 'signaal' kan uitsenden. Dit signaal kan je opvangen via een 'slot' in een ander object (of hetzelfde object). Dat is op zich ook een methode.

Een concreet voorbeeld hiervan vind je in de `QNetworkAccessManager` klasse. Deze heeft enkele signals waaronder 'finished'. Concreet kan je hiermee je eigen klasse bouwen. Je voorziet in jouw klasse een eigen slot die aan de slag gaat wanneer de `QNetworkAccessManager` het finished signaal uitstuurt. Dat finished signaal krijgt het op zijn beurt van een `QNetworkReply` object waarin de data van de request te vinden is.

Je verbindt een slot met een signal door middel van de static methode 'connect'.

```
QLabel *label = new QLabel;  
QScrollBar *scrollBar = new QScrollBar;  
QObject::connect(scrollBar, SIGNAL(valueChanged(int)), label, SLOT(setNum(int)));
```

Bovenstaande code zorgt er voor dat wanneer er binnen een scrollbar element gescrolld wordt, het label steeds de juiste locatie zal weergeven. De scrollbar 'emit' een signal genaamd `valueChanged` wanneer de waarde wijzigt. De slot op het label vangt het signaal op. Hierdoor beschik je steeds over de recentste data in een object. Nog een voorbeeld hiervan kan je in het hoofdstuk over C++ terug vinden.

JavaScript in QML

Je kan in QML heel eenvoudig JavaScript toevoegen:

```
Button {
    width: 200
    height: 300
    property bool checked: false
    text: "Klik op mij"

    // Dit is een JavaScript functie
    function doToggle() {
        checked = !checked
    }
    onClicked: {
        // Dit is terug JavaScript
        doToggle();
        console.log('checked: ' + checked)
    }
}
```

QML is tevens in staat om JavaScript files en modules te importeren. In theorie betekent dit dat je geen C++ hoeft te kunnen om over eenvoudige datastructuren in je applicatie te beschikken. Qt heeft hier het volgende over te zeggen:

There is an open question in the Qt community about the right mixture about QML/JS/Qt C++ in a modern Qt application. The commonly agreed recommended mixture is to limit the JS part of your application to a minimum and do your business logic inside Qt C++ and the UI logic inside QML/JS. -Qt6 QML Book (<https://www.qt.io/product/qt6/qml-book/ch16-javascript-javascript>)

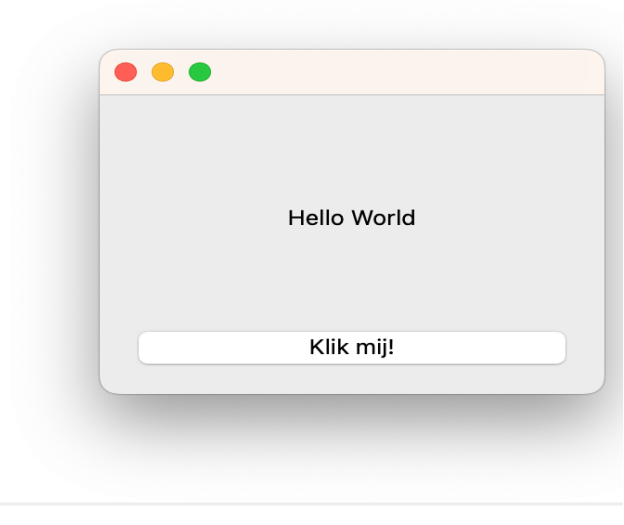
Qt in Python

Met Qt Creator maak je eenvoudig Python files aan maar hier zit de relatie iets anders in elkaar. Qt biedt tevens een package aan, waardoor je niet je favoriete Python IDE moet verlaten. Het package, PySide6, voeg je eenvoudig toe als Pythonista:

```
pip install pyside6
```

Hierna schrijf je Python code gebruik makend van de gewenste imports:

```
hello.py 1 x
1  import random
2  import sys
3
4  from PySide6.QtCore import Qt, Slot
5  from PySide6.QtWidgets import (QApplication, QLabel, QPushButton,
6  | | | | | QVBoxLayout, QWidget)
7  from __feature__ import snake_case, true_property
8
9
10 class MyWidget(QWidget):
11     def __init__(self):
12         QWidget.__init__(self)
13
14         self.hello = [
15             "Hallo wereld",
16             "Hallo Welt",
17             "你好, 世界",
18             "Hei maailma",
19             "Hola Mundo",
20             "Привет мир",
21         ]
22
23         self.button = QPushButton("Klik mij!")
24         self.message = QLabel("Hello World")
25         self.message.setAlignment = Qt.AlignCenter
26
27         self.layout = QVBoxLayout(self)
28         self.layout.addWidget(self.message)
29         self.layout.addWidget(self.button)
30
31         # Connecting the signal
32         self.button.clicked.connect(self.magic)
33
34         @Slot()
35         def magic(self):
36             self.message.text = random.choice(self.hello)
37
38
39 if __name__ == "__main__":
40     app = QApplication(sys.argv)
41
42     widget = MyWidget()
43     widget.show()
44
45     sys.exit(app.exec())
46
```



QML is een declaratieve taal. Dit maakt het binnen de Qt Creator erg eenvoudig om snel een gewenst ontwerp te visualiseren. De realiteit vereist echter veel meer van een applicatie. Als je op een knop klikt moet er iets gebeuren of veranderen. Dit lijkt nog haalbaar binnen QML. Wanneer je echter data wil ophalen of versturen en bijwerken binnen het ontwerp, dan zie je hoe QML hulp gaat nodig hebben. QML, Qt Core en C++ werken hand in hand om dit mogelijk te maken. Een GridView is een QML component dat een model en een delegate gebruikt om dynamisch over data te itereren en weer te geven. Met model bedoelen we het datamodel of de klasse waarin de data terecht komt. Heel concreet spreken we over een object met daarin een lijst van data (op zijn beurt ook objecten van een andere klasse) en de methodes om die lijst te manipuleren (vaak CRUD operaties). De delegate is dan het QML component waarin de data wordt getoond. Elk element uit de eerder genoemde lijst wordt afgebeeld met dit component.

```

5  ✓ GridView {
6      id: driversGrid
7      anchors.fill: parent
8      clip: true
9  ✓  model: DriverModel {
10         list: driverList
11     }
12     cellWidth: parent.width * 0.2
13     cellHeight: 20
14
15  ✓     delegate: Item {
16         implicitWidth: driversGrid.cellWidth
17         height: driversGrid.cellHeight
18  ✓         Rectangle {
19             anchors.fill: parent
20             color: "#fafafa"
21  ✓         Text {
22             id: driverName
23             text: model.fullname
24         }
25     }

```

Het model komt in de applicatie binnen door het als context mee te geven in main.cpp, de file waar je applicatie start.

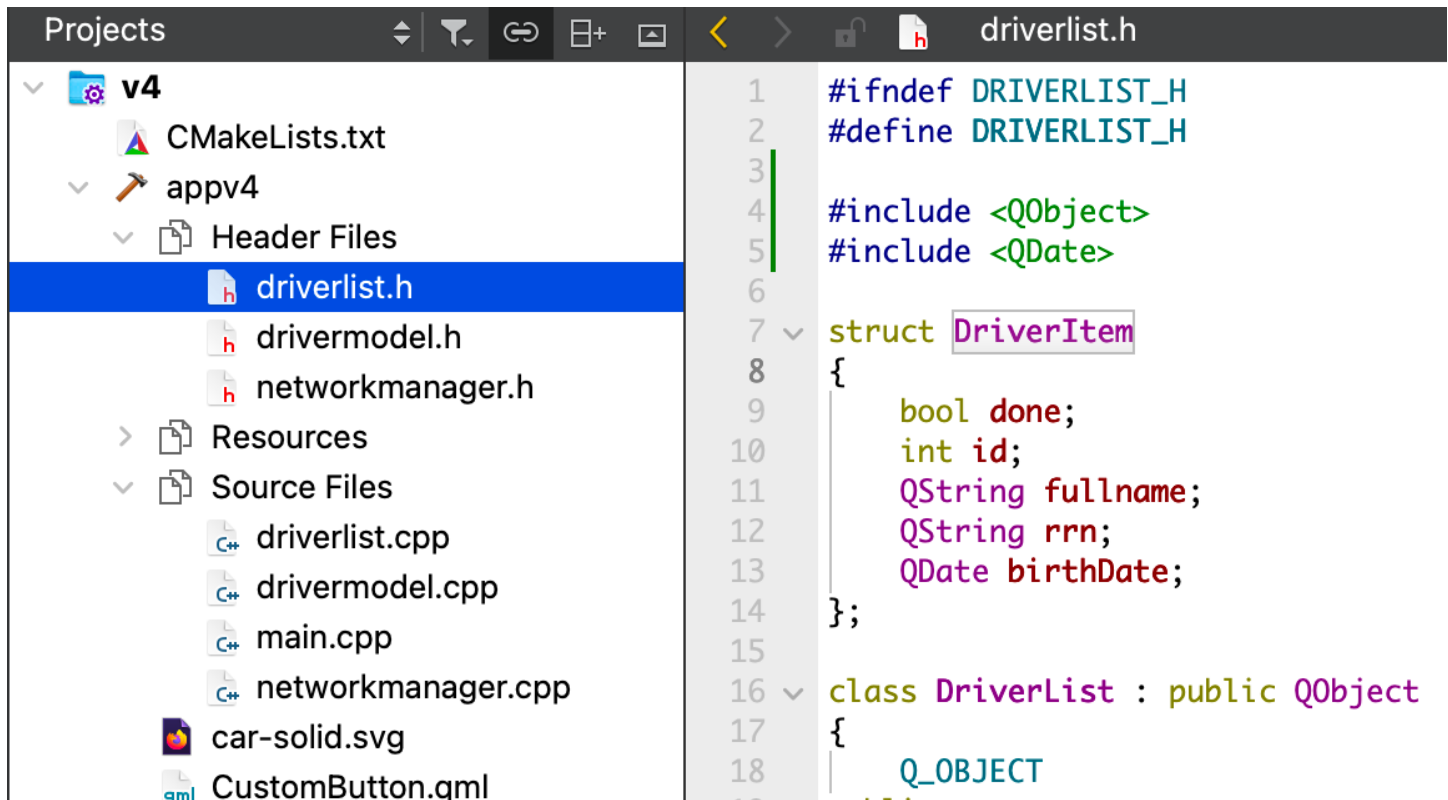
```

16  qmlRegisterType<DriverModel>("Driver", 1,0, "DriverModel");
17  qmlRegisterUncreatableType<DriverList>("Driver", 1,0, "DriverList", QStringLiteral("Driver
18
19  DriverList driverList;
20
21  QQmlApplicationEngine engine;
22  engine.rootContext()->setContextProperty(QStringLiteral("driverList"), &driverList);

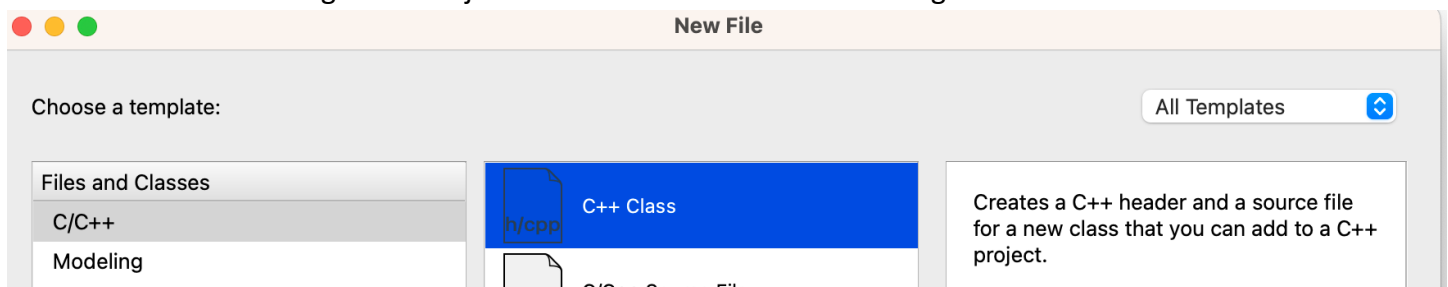
```


Wanneer je een nieuwe C++ klasse maakt, krijg je van de Qt Creator twee bestanden: een .h file of de header en een .cpp file.

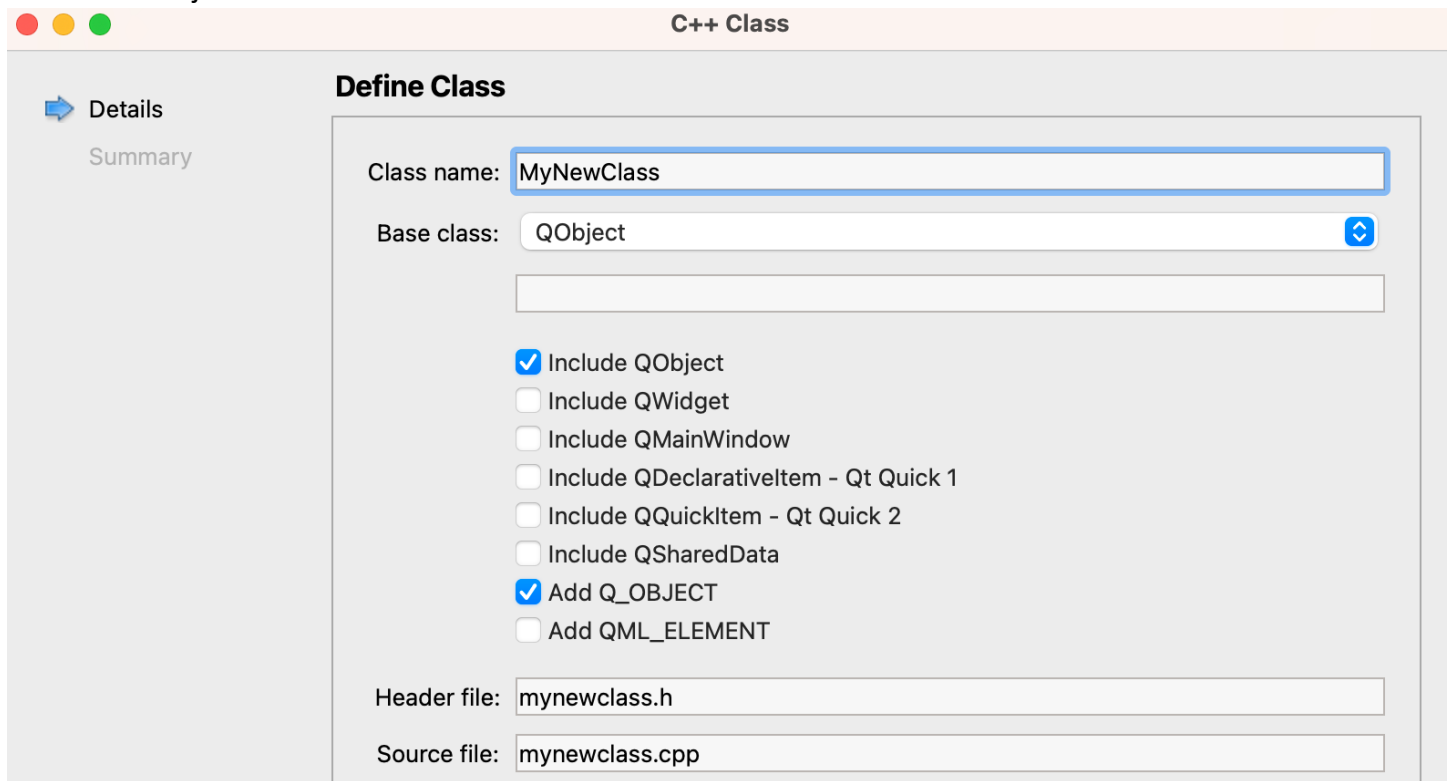
Header file



Binnen Qt maak je best steeds klassen aan die erven van een moederklasse genaamd QObject. De Qt Creator maakt dit heel eenvoudig wanneer je een nieuwe C++ klasse wil toevoegen:



Kies voor QObject in het menu Base Class:



C++ Class

Define Class

Details
Summary

Class name:

Base class:

☒ Include QObject
☐ Include QWidget
☐ Include QMainWindow
☐ Include QDeclarativeItem - Qt Quick 1
☐ Include QQuickItem - Qt Quick 2
☐ Include QSharedData
☒ Add Q_OBJECT
☐ Add QML_ELEMENT

Header file:

Source file:

Voeg beide bestanden toe aan de CMakeLists.txt file:

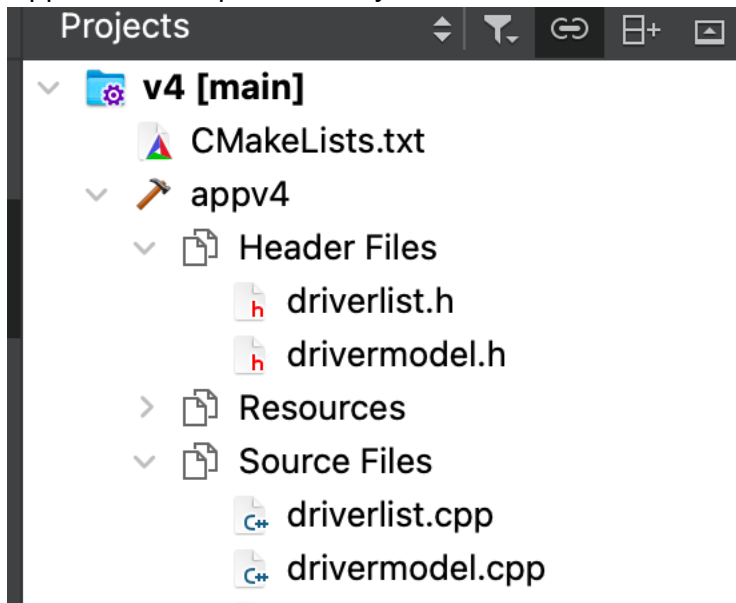
```
5 set(CMAKE_CXX_STANDARD_REQUIRED ON)
6
7 find_package(Qt6 REQUIRED COMPONENTS Quick)
8 find_package(Qt6 REQUIRED COMPONENTS Svg)
9
10 qt_standard_project_setup()
11
12 qt_add_executable(appv4
13     main.cpp
14     drivermodel.h
15     drivermodel.cpp
16     driverlist.h
17     driverlist.cpp
18     mynewclass.h
19     mynewclass.cpp
20 )
```

Wanneer je dit bestand opslaat zal je de header en source file terug vinden in de Qt Creator interface. De header file creëert een geraamte voor de source file (.cpp) die de header implementeert. Strikt genomen ben je niet verplicht deze te gebruiken maar het geeft je een heel concreet zicht op wat de code precies kan en doet zonder de details te moeten bekijken. Elke header file heeft ruimte voor public en private methodes en variabelen en zoals eerder gezien signals en slots dankzij de overerving van de QObject klasse:

```
19 public:
20     explicit DriverList(QObject *parent = nullptr);
21
22     QVector<DriverItem> items() const;
23
24     bool setItemAt(int index, const DriverItem &item);
25
26     Q_INVOKABLE void fetchDriverData();
27
28 signals:
29     void preItemAppended();
30     void postItemAppended();
31
32     void preItemRemoved(int index);
33     void postItemRemoved();
34
35 public slots:
36     void appendItem(const DriverItem *item);
37     void removeItems();
38     void convertData(QString reply);
39     void setResponse(const QString &newResponse);
40
41 private:
42     QVector<DriverItem> mItems;
43     QString _response;
44 };
```

Source file

Met voorgaande instructies beschik je over twee bestanden. In de source file of het bestand eindigend op de .cpp extensie implementeer je de definities uit het header bestand.



Eerst en vooral vinden we de includes terug die nodig zijn voor de werking van de code. Bovenaan prijkt alvast de header file. Erna volgen andere includes, vaak van standaard libraries zoals bv stdio.h. In een Qt/QML project ga je echter vaak klassen uit de Qt library terug vinden, deze beginnen steeds met een Q:

```
1  #include "driverlist.h"
2
3  #include <QVector>
4  #include <QDebug>
5  #include <QEventLoop>
6  #include <QNetworkAccessManager>
7  #include <QNetworkRequest>
8  #include <QNetworkReply>
```

```
#include <QJsonArray>

DriverList::DriverList(QObject *parent)
    : QObject{parent}
{
}

 QVector<DriverItem> DriverList::items() const
{
    return mItems;
}

bool DriverList::setItemAt(int index, const DriverItem &item)
{
    if (index < 0 || index >= mItems.size())
        return false;

    const DriverItem &oldItem = mItems.at(index);
    if (item.id == oldItem.id)
        return false;

    mItems[index] = item;
    return true;
}

void DriverList::fetchDriverData()
{
    QEventLoop eventLoop;
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)), &eventLoop, SLOT(quit()));
```

Slots zijn methodes die acties uitvoeren en tijdens de uitvoering kunnen ze een signaal uitsturen of "emitter".

Dit zijn de signals die je terug vindt in de header:

```
60 void DriverList::appendItem(const DriverItem *item)
61 {
62     emit preItemAppended();
63     mItems.append(*item);
64     emit postItemAppended();
65 }
66
67 void DriverList::removeItems()
68 {
69     for (int i=0; i < mItems.size(); ++i) {
70         emit preItemRemoved(i);
71         mItems.removeAt(i);
72         emit postItemRemoved();
73     }
74 }
```

Om deze signalen op te vangen maak je gebruik van connect of QObject::connect:

```
connect(mList, &DriverList::preItemAppended, this, [=]() {
    const int index = mList->items().size();
    beginInsertRows(QModelIndex(), index, index);
});
```

Eerst vullen we de dataset in die we in de gaten willen houden. Het tweede argument is het signaal waarop we willen reageren. Daarna volgt het object (vaak "this" dus het eigen object) waarin we acties willen uitvoeren wanneer dat signaal binnenkomt en als laatste geven we een functie die uitgevoerd wordt zodra het signaal is ontvangen. De "[=]()" staat voor een anonieme functie die inline wordt uitgevoerd maar dit kan ook gerust een gedefinieerde functie zijn of een slot dat op zijn beurt een signaal zou kunnen uitsturen.

Observaties

We zagen eerder dat Qt beschikt over de mogelijkheid om binnen Python te werken en JavaScript kan interpreteren. Een voordeel want hierdoor trek je veel meer developers aan om grafische applicaties te bouwen met Qt. Toch blijft Qt sterk gegrond in de C++ wereld. Qt is immers een framework dat de C++ taal uitbreidt. Deze relatie is naar mijn gevoel symbiotisch. Door Qt te leren, leer je C++ en omgekeerd, als je C++ wil gebruiken of leren werkt er persoonlijk niets beter dan visualisatie van je werk met behulp van Qt en QML.

Voor eenvoudige zaken kan ik even snel C++ schrijven als C# of JavaScript. Qt Core biedt een intuïtieve ervaring om een console applicatie te maken zodat je met in- en uitvoer kan omgaan, functies kan declareren en variabelen kan opslaan. QML is tevens een heel eenvoudige syntax om te begrijpen. Tot ik nood had aan het GridView component en data wou ophalen uit een API. Dan begon het te dagen hoe krachtig maar ook hoe complex deze oplossingen zijn t.o.v. JavaScript of C# met WPF. Het begrip voor deze materie begint langzaam te komen en dat maakt me enthousiast om er meer mee te gaan doen.

Wat me vaak opvalt bij het lezen van alle naslagwerken, bronnen op het internet, video's en documentatie is dat er weinig aandacht wordt besteed aan het "waarom". Nu we aan het einde van de opleiding zijn begin ik me meer vragen te stellen over het waarom dan over het hoe of wat. De "waarom"-vragen krijg je voor C en C++ wel beantwoord, maar misschien zijn we gewoon wat te verwend in C#, Java en zeker JavaScript. Een concreet voorbeeld hiervan zijn pointers. Het idee erachter kan je mits een beetje oefening onder de knie krijgen. Wat heel wat minder voor de hand ligt is het wanneer. Wanneer moeten we het precies gebruiken in plaats van een reguliere variabele? Mijn gevoel zegt dat het om discipline gaat: discipline om de materie echt te begrijpen waardoor je die beter kan toepassen en zo tot een beter design en duidelijkere code komt. Het blijkt vooral belangrijk om deze concepten onder de knie te hebben wanneer je gebruik gaat maken van libraries en frameworks van derde partijen.

Verder inzicht in de taal leert me dat het een krachtige manier is om gebruik te maken van het beschikbare geheugen. Daarbij komt wel kijken dat je terug die discipline moet hebben om het geheugen goed te beheren. De boekhouding van een bedrijf is hierbij een interessante analogie. Complex, maar doe het juist en goed en de zaak blijft draaien. doe het fout en slordig en je zal gauw zien dat er klachten komen en van waar.

Vast staat dat ik veel heb geleerd. Door dit verslag te schrijven ben ik concepten en begrippen beter gaan begrijpen, hoewel ik ze reeds had toegepast in de code van de graduaatsproef. Ik heb mezelf ook echt uitgedaagd om de complexere vereisten te implementeren. Wat ik heb gemaakt is verre van volledig en de kennis die ik heb vergaard is zeker niet compleet.

Dit project heeft ook mijn noties over C++ volledig omgegooid. Ik achtte C++ een mythische taal, gesproken en gebruikt door de oude rotten van het vak, die lijdzaam moesten toekijken hoe moderne talen zoals C# en Java met alle eer zijn gaan lopen. Ik ben er nu van overtuigd dat C++ nog steeds een toekomst heeft, maar het zal niet zo gauw de voorgrond betreden. Het werkt in alle rust en stilte aan de vooruitgang van technologieën zoals Unreal Engine, applicaties zoals Google Chrome en zelfs Firefox maar het stuwt ook andere programmeertalen voort zoals Qt en zelfs JavaScript. Enkele van de genoemde voorbeelden kan je onmogelijk nog wegdenken uit de IT wereld of zelfs de maatschappij. C++ heeft alle karakteristieken van een goede leider. Het heeft geen ego en het biedt je alle kansen om te groeien, om zelf een leider te worden. Dat is een bemoedigende gedachte om met deze taal verder aan de slag te gaan.

Bronnen

<https://www.qt.io/>

<https://www.qt.io/qt-for-python>

<https://www.qt.io/product/qt6/qml-book/ch16-javascript-javascript>

<https://www.qt.io/blog/2019/08/07/technical-vision-qt-6>

<https://www.qt.io/blog/2018/10/29/deprecation-of-qbs>

https://en.wikipedia.org/wiki/Qt_Project

https://raymii.org/s/tutorials/HTTP_GET_requests_in_Qt_and_Qml_async.html