Merhaba ben **Batuhan Çulhacıoğlu,**

Düzce Üniversitesi Elektrik-Elektronik bölümünden 2023 yılında mezun oldum. 2018'de DÜ Havacılık ve Uzay Topluluğunda başlayan gömülü yazılım çalışmalarımı hala devam ettirmekteyim ve bu alanda kendimi geliştirmekteyim. Havacılık ve uzaya olan ilgimden dolayı gömülü yazılım projelerimi genel olarak bu alanlara yönelik gerçekleştirmekteyim. Bu süreçte bir çok IMU(atalet, jiroskop, manyetometre), basınç (ölçülen basınç verisi ile irtifa hesaplaması) sensörleri, GPS, LoRa ve Xbee Rf modülleri ile bir çok çalışma, test, kütüphanelerin ve algoritmaların oluşturulması gibi çalışmalar gerçekleştirdim. Ayrıca FOTA (Firmware Over The Air), SPI Bootloader gibi projeler de gerçekleştirdim.

Bu döküman genel hatlarıyla Qt Designer uygulamasında C++ tabanlı Windows arayüz uygulamaları projelerinin nasıl oluşturulacağına ve standart araçların nasıl kullanılacağına dair temel bir açıklama dosyasıdır.

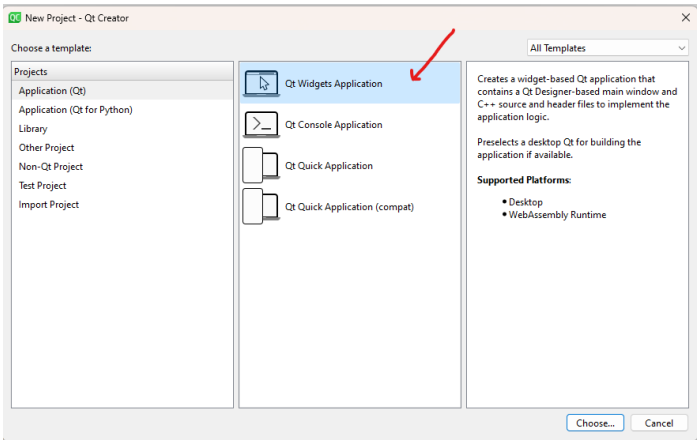
İçindekiler

1. Qt Designer

Qt Designer, Qt Framework tarafından sağlanan bir grafik kullanıcı arayüzü (GUI) tasarım aracıdır. Qt Designer, geliştiricilere kullanıcı arayüzlerini kolayca oluşturmak ve düzenlemek için bir dizi araç ve özellik sunar. Kullanıcılar, sürükle ve bırak yöntemiyle arayüz öğelerini ekleyebilir, düzenleyebilir ve özelleştirebilirler. Qt Designer'ın en büyük avantajlarından biri, oluşturulan arayüzlerin C++ veya Python gibi programlama dilleriyle kolayca entegre edilebilmesidir. Bu, geliştiricilerin GUI uygulamalarını hızlı ve verimli bir şekilde geliştirmelerine olanak tanır. Qt Designer ayrıca, arayüzlerin görsel olarak nasıl görüneceğini önizleme imkanı sunar, böylece tasarımın gerçek zamanlı olarak izlenmesine olanak tanır. Bu özellikleriyle Qt Designer, Qt Framework kullanarak GUI uygulamaları geliştiren geliştiriciler için güçlü bir araçtır.

2. Yeni Proje Oluşturma

Oluşturulacak bir proje eğer C++ ile bir GUI ise Application(Qt) -> Qt Widgets Application seçeneği seçilmelidir.



3. Define Build System

Qt Designer'da "Define Build System" bölümünde "qmake" ve "cmake" seçenekleri, proje derleme sistemi olarak kullanılacak olan iki farklı araçtır. İşte bu iki araç arasındaki temel farklar:

qmake:

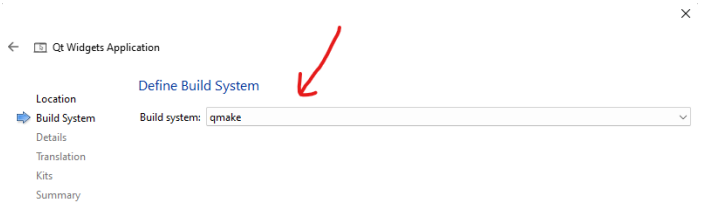
Qt Framework tarafından sağlanan bir proje derleme aracıdır. Qt proje dosyalarını (.pro uzantılı) kullanarak projenin derleme işlemlerini yönetir. Qt Creator gibi Qt geliştirme ortamlarında sıkça kullanılır. Qt proje dosyaları, proje yapılandırması, kaynak dosyalarının listesi ve proje bağımlılıklarının belirlendiği bir yapıya sahiptir.

CMake:

CMake, genel amaçlı bir proje derleme aracıdır ve Qt ile sınırlı değildir. Platformlar arası proje yapılandırması sağlar ve farklı derleme sistemleriyle uyumlu olacak şekilde tasarlanmıştır. Proje yapılandırması için CMakeLists.txt adlı metin dosyalarını kullanır. Qt tabanlı projelerde de sıklıkla kullanılabilir, özellikle birden fazla platformda çalışacak projeler için tercih edilir.

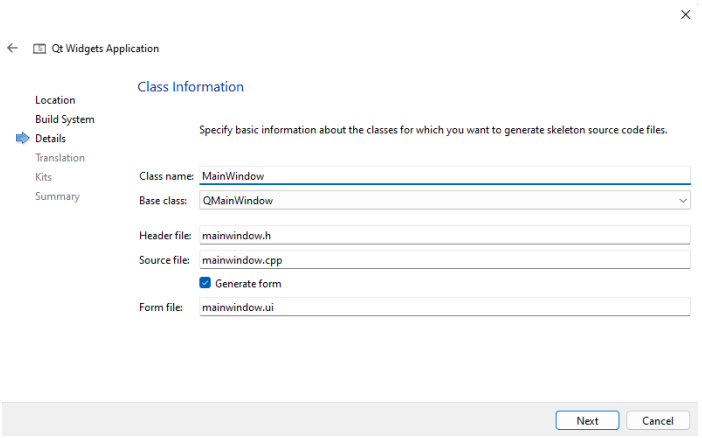
Bu seçenekler arasında tercih yaparken, projenizin gereksinimlerini ve hangi derleme aracının sizin için daha uygun olduğunu değerlendirmeniz önemlidir. Örneğin, Qt Creator gibi Qt tabanlı bir IDE kullanıyorsanız, genellikle qmake tercih edilirken, CMake daha genel amaçlı bir çözüm sunar ve projenizin Qt dışındaki kısımlarıyla daha uyumlu olabilir.

Qt Creator IDE ‘sini kullanacağımızdan dolayı qmake seçerek .pro dosyasının oluşturulmasını sağlıyoruz.



4. Class Information

Oluşturacağımız mainclass’ın isimlendirmesini gerçekleştirdiğimiz bölümdür. Burada oluşturulan class ana pencerenin özelliklerinin oluşturulduğu ve düzenlendiği class’tır.



5. Kit Selection

"Kit Selection" bölümü, Qt Designer'da yeni bir proje oluştururken hedef platform ve derleme aracını belirlemek için kullanılan bir bölümdür. Bu bölüm, projenizin hangi kit veya kiti kullanarak derleneceğini ve hedefleneceğini belirlemenizi sağlar. Burada genellikle şu seçenekler bulunur:

Mingw (Minimalist GNU for Windows):

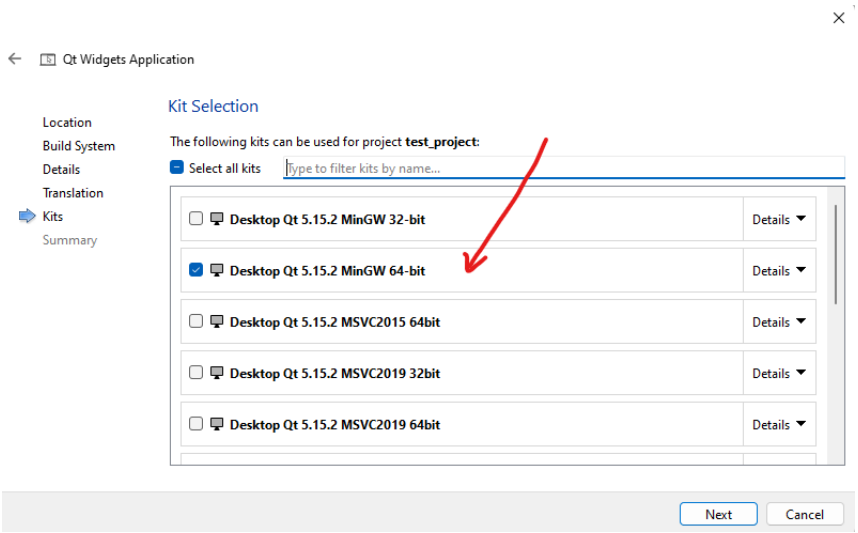
Mingw, Windows platformu için GCC (GNU Compiler Collection) derleyicisine dayalı bir derleme aracıdır. Qt Designer ile Windows'ta Qt tabanlı uygulamalar geliştirirken tercih edilebilir.

MSVC (Microsoft Visual C++):

MSVC, Microsoft Visual Studio tarafından sağlanan C++ derleyicisidir. Qt Designer ile Windows üzerinde Visual Studio kullanarak Qt tabanlı uygulamalar geliştirmek istiyorsanız bu seçeneği kullanabilirsiniz.

Bu seçenekler, projenizin hedef platformunu ve kullanılacak derleme aracını belirlemek için kullanılır. Örneğin, Mingw'i seçerseniz, projeniz Windows platformunda GCC derleyicisiyle derlenecek ve çalıştırılacak. MSVC seçeneğini tercih ederseniz, projeniz Visual Studio'nun C++ derleyicisiyle derlenecek ve çalıştırılacaktır.

Hangi kiti seçeceğinizi belirlerken, hedef platformunuz, mevcut geliştirme ortamınız ve projenizin gereksinimleri gibi faktörleri göz önünde bulundurmanız önemlidir. Burada Windows tabanlı çalıştığım için MinGW derleyicisini seçiyorum.

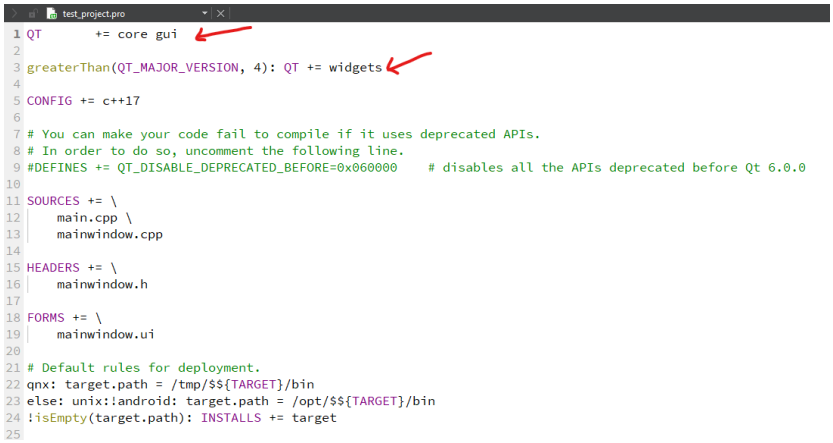


6. “.pro” Dosyası

Qt proje dosyasının uzantısıdır ve Qt projesinin yapılandırma ve derleme işlemlerini yönetmek için kullanılır. Qt Creator gibi Qt geliştirme ortamlarında sıkça kullanılır.

Qt proje dosyası, projenin kaynak dosyalarının listesi, projenin derleme seçenekleri, projeye dahil edilecek kütüphaneler ve diğer proje bağımlılıklarının belirlendiği bir yapıya sahiptir. Bu dosya, Qt'nin qmake adlı aracı tarafından işlenir ve derleme işlemleri için gerekli komutlar oluşturulur.

Tipik bir .pro dosyası, aşağıdaki gibi görünebilir:



QT += core gui:

Bu satır, Qt projesinin hangi Qt modüllerini kullanacağını belirtir. Core modülü, temel Qt sınıf ve işlevlerini içerir ve her Qt uygulamasında genellikle kullanılır. Gui modülü, grafik kullanıcı arayüzü (GUI) oluşturmak için gerekli olan Qt sınıflarını içerir. Bu satır, projenin temel Qt bileşenlerini içe aktararak Qt'nin temel fonksiyonelliğini sağlar.

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets:

Bu satır, Qt sürümünün büyük versiyon numarasının 4'ten büyük olup olmadığını kontrol eder. Eğer Qt sürümü 4'ten büyükse, widgets modülünü ekler. widgets modülü, Qt'nin grafik kullanıcı arayüzü (GUI) bileşenlerini içerir. Bu, QPushButton, QLabel gibi widget'ları kullanarak arayüzler oluşturmayı sağlar. Bu satır, Qt sürümüne bağlı olarak projeye ek işlevselliği sağlar. Eğer Qt sürümü 4'ten büyükse, widgets modülü projeye dahil edilir.

CONFIG += c++17:

Bu satır, C++ derleyiciye ek derleme ayarlarını belirtir. C++17 standardına uygun derleme işlemi için yapılandırmayı sağlar. Bu satır, projenin C++17 dil standardına göre derlenmesini sağlar. Bu, projenin C++17 dil özelliklerinden faydalanmasına ve uyumlu bir şekilde derlenmesine olanak tanır.

Bu satırlar, .pro dosyasında projenin Qt bağımlılıklarını, Qt modüllerini ve derleme ayarlarını belirtir. Bu ayarlar, projenin Qt framework'ünü nasıl kullanacağını ve derlenmesini nasıl yapılandıracağını belirler.

*** Proje içerisine bir class dahil edildiğinde bunun .pro dosyası içerisinde belirtilmesi gerekmektedir.

7. Main.cpp

Main.cpp dosyası genellikle bir C++ Qt uygulamasının giriş noktasıdır. Bu dosya, uygulamanın ana işlevini ve başlangıç ayarlarını içerir. İşlevi aşağıdaki gibidir:

Giriş Noktası: main.cpp, uygulamanın başlangıç noktasıdır. Program çalıştırıldığında, işletim sistemi tarafından ilk olarak main işlevi çağrılır.

Uygulama Nesnesi Oluşturma: main işlevi genellikle bir QApplication nesnesi oluşturur. QApplication, Qt uygulamasının temelini oluşturur ve uygulama döngüsünü başlatır.

Ana Pencere Oluşturma: QMainWindow veya başka bir widget sınıfından bir ana pencere oluşturulabilir. Bu pencere genellikle uygulamanın ana arayüzünü içerir.

Arayüz Nesnesi İşlemleri: Ana pencere ve diğer widget nesneleri, gerekirse ui dosyasından yüklenir ve ayarlanır.

Uygulama Döngüsü Başlatma: QApplication::exec() çağrısıyla uygulama döngüsü başlatılır. Bu, kullanıcı etkileşimi ve olaylarla ilgili işlemleri dinler ve yanıtlar.

Özetle, main.cpp dosyası, Qt uygulamasının başlangıç noktasını tanımlar ve uygulamanın ana penceresini ve diğer bileşenlerini oluşturarak uygulamanın çalışmasını başlatır. Bu dosya, genellikle Qt projesinin merkezi bir bileşenidir ve uygulamanın temel yapılandırmasını ve işleyişini sağlar.

main.cpp | main

```cpp
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

8. Mainwindow.h ve Mainwindow.cpp Dosyaları

Qt tabanlı bir uygulamanın ana penceresini (MainWindow) tanımlar ve uygulamanın kullanıcı arayüzüyle ilgili işlevselliği sağlar. İşlevleri şu şekildedir:

mainwindow.h:

Ana Pencere Sınıfı: mainwindow.h, genellikle bir sınıf tanımlar, bu sınıf MainWindow veya benzeri bir isim alabilir. Bu sınıf, genellikle QMainWindow sınıfından türetilir.

Başlık Dosyası: Bu dosya, MainWindow sınıfının başlık dosyasıdır. Bu dosya, mainwindow.cpp dosyasında mainwindow.h başlığı dahil edilerek kullanılır.

Uygulama Arayüzü Tanımları: mainwindow.h, genellikle kullanıcı arayüzü bileşenlerinin (pencere, düğmeler, menüler vb.) tanımlarını içerir. Bu bileşenler, Qt Designer veya elle oluşturulmuş olabilir.

Yardımcı Fonksiyon ve Değişkenler: Gerekli ise, başlık dosyası, ana pencere sınıfında kullanılacak yardımcı fonksiyonları ve değişkenleri de içerebilir.

mainwindow.cpp:

Ana Pencere Fonksiyonları: mainwindow.cpp, MainWindow sınıfının üye işlevlerinin (member functions) uygulamalarını içerir. Bu fonksiyonlar, ana pencerenin davranışını tanımlar. Örneğin, pencere başlatma işlevi (MainWindow sınıfının kurucu işlevi) burada tanımlanır.

Olay İşleyicileri: Kullanıcı etkileşimlerine yanıt olarak çalışacak olay işleyicileri (slots) burada tanımlanabilir. Örneğin, bir düğmeye tıklandığında yapılacak işlemler bu dosyada tanımlanır.

Başlık Dosyası Dahil Edilir: mainwindow.cpp dosyası, mainwindow.h başlık dosyasını dahil eder. Böylece MainWindow sınıfının tanımlarına ve başlık dosyasındaki diğer tanımlara erişim sağlar.

Qt Kütüphaneleri Dahil Edilir: Qt kütüphaneleri (QMainWindow, QWidget gibi) burada genellikle #include yönermesiyle dahil edilir.

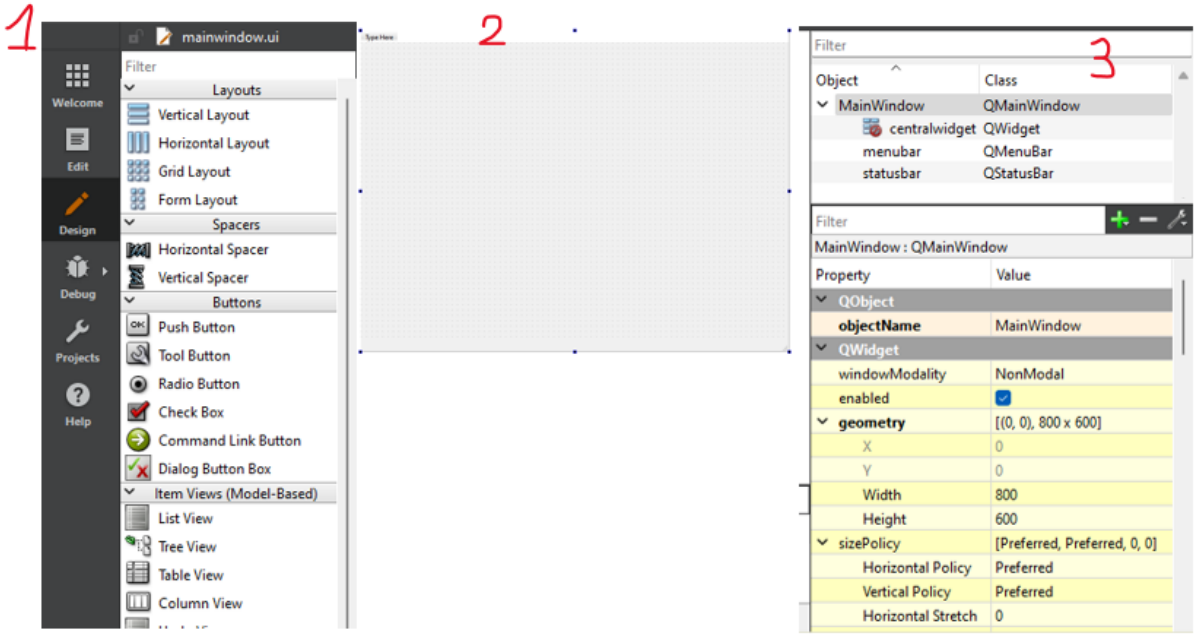
Bu iki dosya, genellikle bir Qt uygulamasının ana penceresinin (MainWindow) tanımlandığı ve bu pencerenin davranışını belirlediği yerlerdir. Başlık dosyası, sınıfın arabiriminin tanımlandığı yerken, kaynak dosyası ise bu arayüzün nasıl gerçekleştirileceğini ve işleneceğini belirler.

mainwindow.cpp | MainWindow::MainWindow(QWidget *)

```cpp
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

9. Design

Bu kısım Qt Designer'da kullanıcı arayüzü bileşenlerinin tasarlandığı ve düzenlendiği bölümdür.



1.Panel Araçlar

Design sekmesinde sol tarafta kalan bu araç panelinde kullanılabilecek temel araçlar yer almaktadır. Bu araçlar sürekle bırakla yöntemiyle editör kısmına yerleştirilmelidir.

2.Panel Editör

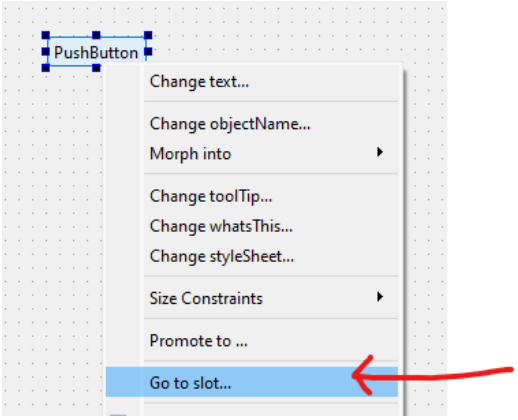
Bu kısım editör kısmıdır ve Design sekmesinde ekranın ortasında yer alır. Burada GUI'nin tasarım düzenlemeleri yapılmaktadır. Yapılan tasarımın önizlemesi burada görüntülenir.

3.Panel Araç Özellikleri

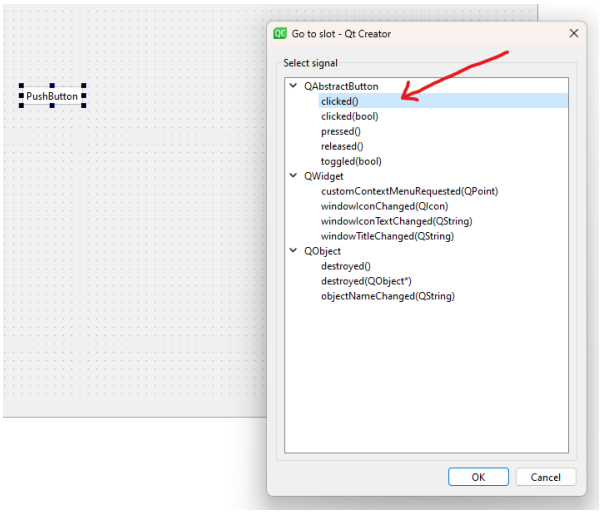
Araç panelinden editöre eklenen her bir araç burada nesne olarak görüntülenir. Bu araç özellikleri paneli Design sekmesinde sağ tarafta yer alır. Burada nesnelerin isimlendirmeleri ve düzenlemeleri yapılabilmektedir. Alt kısımda yer alan panelde nesnenin özel ayarlamaları ve özellik değişiklikleri gerçekleştirilebilmektedir.

10. Go to slot (Nesnelere Görev Ataması)

Eklenen her bir nesnenin gerçekleştireceği görev tanımı nesnenin üzerine sağ tıklayıp go to slot seçeneği ile kolayca seçilebilmektedir. Aynı şekilde bu nesnenin tanımlaması mainwindow class yapısının içerisinde yapılabileceği gibi buradan yapılan nesne görev oluşturma işlemleri kolaylık sağlamaktadır. Ve seçilen her bir özellik method olarak mainwindow class'ı içerisinde otomatik olarak oluşturulur. Bu methodlar içerisinde nesnenin gerçekleştireceği görev tanımı belirlenir.



Burada clicked seçeneği seçilerek nesneye tıklandığı an bu method çağrılarak içerisinde ki görevin gerçekleştirilmesi sağlanır.

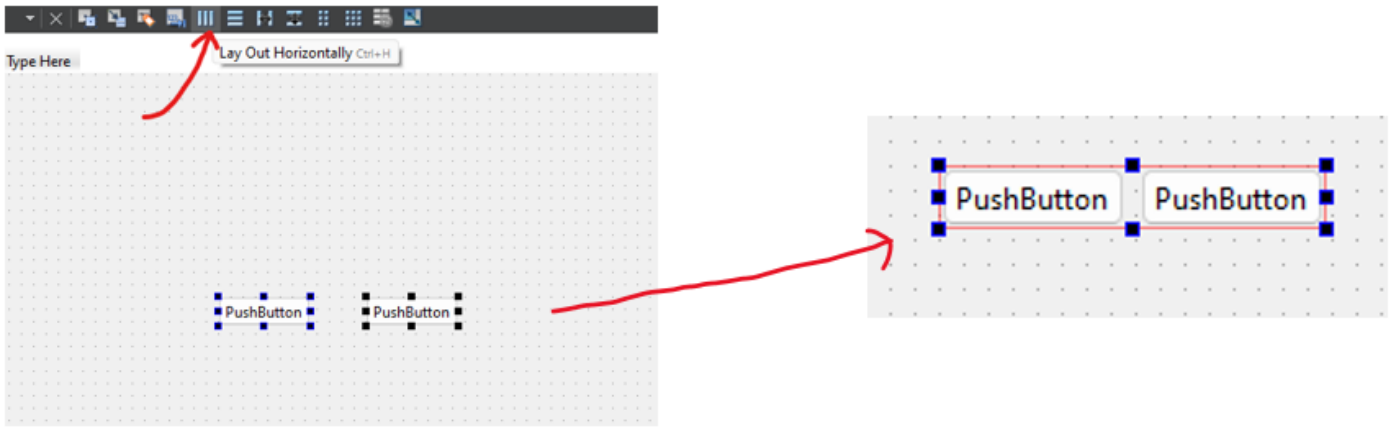


Mainwindow class'ı içerisinde oluşturulan push button -> click methodu örneği ;

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{

}
```

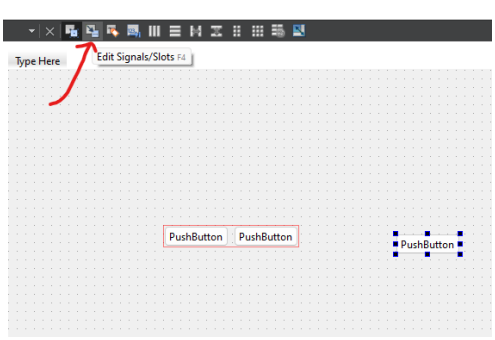
11. Lay Out (Hizalama)

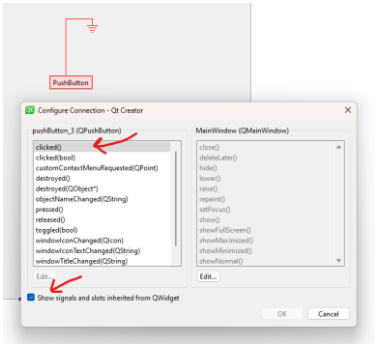
Tanımlanan birden fazla nesnenin ctrl tuşu ile birlikte seçilmesi ile yukarıda yer alan lay out seçenekleri ile hizalanması veya gruplandırılması gerçekleştirilebilir. Bu seçenek oluşturulan arayüz üzerinde düzenin sağlanması ve nesnelerin yerlerinin kolayca değiştirilebilmesi amacıyla kullanılmaktadır.

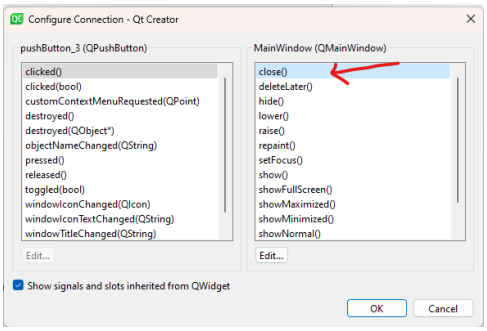


12. Signal (Standart Görev Ekleme)

Oluşturulan bir nesnenin Qt Designer Signal kısmında yer alan standart bir görevi var ise nesne seçildikten sonra Edit Signal seçilerek nesnenin görev methodunu mainwindow dosyası içerisinde tanımlanması önlenir ve burada seçilen görev otomatik olarak gerçekleştirilir.



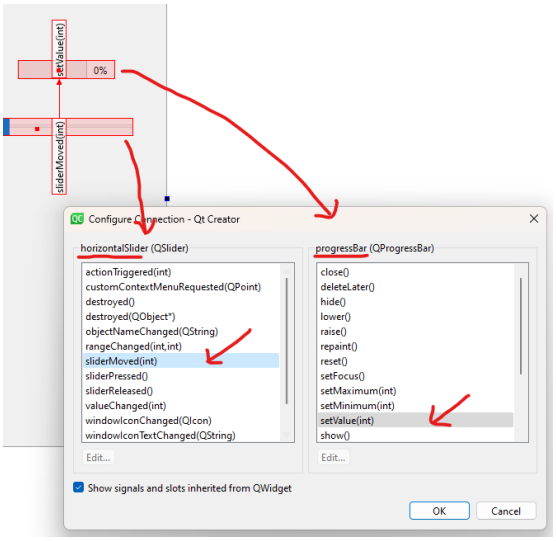




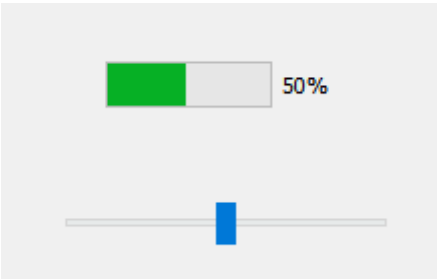
Örnek olarak clicked (butona tıklandığında) GUI ekranını kapatılmasını istiyor isek closed seçeneği seçilerek GUI'nin kapatılması sağlanabilir. Özet olarak go to slot kısmı ile aynı görevi görür fakat go to slot kısmı kullanıcıya daha spesifik özellikler eklemesine olanak sağlar. Signal ise standart özellikler ve kolaylık sağlar. Signal için nesne seçildikten sonra Signal seçeneği ile nesne üzerinden boş bir noktaya çekilen ok ile oluşturulur.

13. Signal/Slot (Standart Nesnelerin Bağlanması)

İki nesnenin Signal ile birbirine bağlanması sağlanır. Bunun için öncelikle olayı başlatacak olan nesne seçilir ve onun üzerinden gerçekleşecek veya bağlanacak nesne ok yardımı ile seçilir.



Burada ki örnekte Slider nesnesinin değeri kaydırılarak değiştirildiğinde yukarıda yer alan progressBar'ın değeri de değişmektedir.



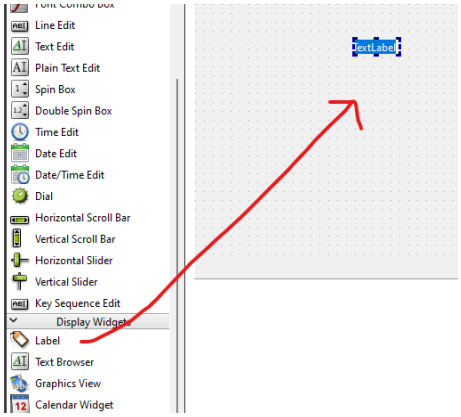
14. Label

Label, kullanıcı arayüzünde metin veya görsellerin görüntülendiği bir bileşendir. Genellikle statik metinleri veya bilgilendirici mesajları göstermek için kullanılır. Etiketler, kullanıcıya belirli bir bilgiyi iletmek veya kullanıcı arayüzünü açıklamak için kullanışlıdır.

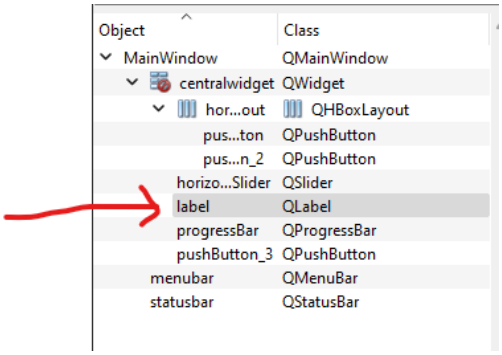
Qt framework'ünde QLabel sınıfı, etiketlerin oluşturulması ve yönetilmesi için kullanılır. Bir QLabel nesnesi oluşturulduğunda, içeriği (metin veya görsel) ve görünümü (renk, boyut, yazı tipi vb.) belirlenebilir. QLabel, kullanıcı etkileşimi beklemeyen ve genellikle değişmeyen içeriği göstermek için ideal bir bileşendir.

Örnek olarak, bir kullanıcı giriş formunda, "Kullanıcı Adı" veya "Şifre" gibi etiketler, kullanıcının hangi alanlara ne tür veri girmesi gerektiği konusunda bilgilendirici metinler olarak kullanılabilir. Ayrıca, bir resim veya logonun görüntülenmesi için de QLabel kullanılabilir.

Kısacası, bir label, kullanıcı arayüzünde metin veya görsellerin görüntülendiği bir bileşendir ve kullanıcıya belirli bir bilgiyi iletmek veya arayüzü açıklamak için kullanılır.



Label nesnesi eklendikten sonra eğer sağ tarafta ki menüden isim değeri değiştirilmez ise standart olarak label ismini alır.



Bu nesnenin içerisinde yer alan isim mainwindow içerisinde ui->label->setText methodu ile değiştirilebilir.

15. Ui_MainWindow (Oluşturulan Nesnelerin Tanımlamaları)

Editör aracılığı ile oluşturulan her bir nesne ui_mainwindow class'ı içerisinde tanımlanır. Burada değişiklik yapıldıktan sonra eğer Editör kısmında yeni bir düzenleme yapılır ise tekrar mainwindow içeriği baştan oluşturulur. Bu nedenle manuel yapılan mainwindow değişiklikleri kaybolabilir ve her seferinde tekrar yapılması gerekir.

mainwindow.h — MainWindow(QWidget *)

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

ui_mainwindow.h — MainWindow

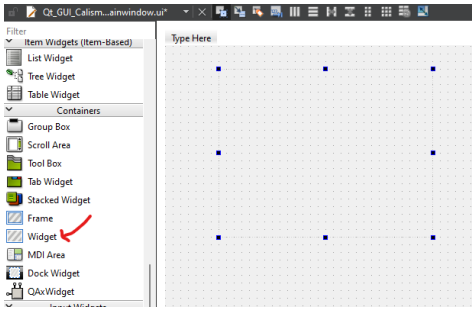
```
QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QWidget *centralwidget;
    QPushButton *pushButton_3;
    QProgressBar *progressBar;
    QSlider *horizontalSlider;
    QLabel *label;
    QWidget *widget;
    QHBoxLayout *horizontalLayout;
    QPushButton *pushButton;
    QPushButton *pushButton_2;
    QMenuBar *menubar;
    QStatusBar *statusbar;
```

```
void setupUi(QMainWindow *MainWindow)
{
    if (MainWindow->objectName().isEmpty())
        MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
    MainWindow->resize(800, 600);
    centralwidget = new QWidget(MainWindow);
    centralwidget->setObjectName(QString::fromUtf8("centralwidget"));
    pushButton_3 = new QPushButton(centralwidget);
    pushButton_3->setObjectName(QString::fromUtf8("pushButton_3"));
    pushButton_3->setGeometry(QRect(450, 190, 75, 24));
    progressBar = new QProgressBar(centralwidget);
    progressBar->setObjectName(QString::fromUtf8("progressBar"));
    progressBar->setGeometry(QRect(620, 110, 118, 23));
    progressBar->setMaximum(99);
    progressBar->setValue(0);
    horizontalSlider = new QSlider(centralwidget);
    horizontalSlider->setObjectName(QString::fromUtf8("horizontalSlide
    horizontalSlider->setGeometry(QRect(600, 180, 160, 22));
    horizontalSlider->setOrientation(Qt::Horizontal);
    label = new QLabel(centralwidget);
    label->setObjectName(QString::fromUtf8("label"));
    label->setGeometry(QRect(180, 310, 49, 16));
    widget = new QWidget(centralwidget);
```

16. Widget

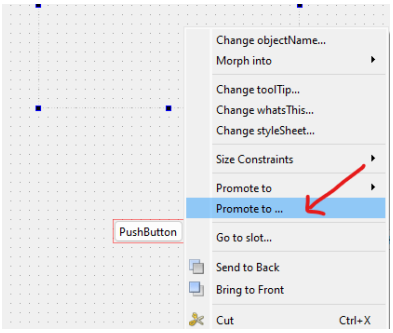
Widget, bir kullanıcı arayüzünde görüntülenen herhangi bir öğeyi (nesneyi) temsil eder. Widget'lar, kullanıcı arayüzünde farklı işlevleri gerçekleştiren ve kullanıcı etkileşimini sağlayan grafiksel öğelerdir. Örneğin, düğmeler, metin kutuları, etiketler, pencere çerçeveleri ve menüler gibi birçok farklı widget türü bulunmaktadır. Qt framework'ünde, QWidget sınıfı, widget'ların temel sınıfını oluşturur. Bu sınıf, bir widget'in genel özelliklerini ve davranışlarını tanımlar ve Qt tabanlı uygulamalarda kullanıcı arayüzünü oluşturmak için yaygın olarak kullanılır. Widget'lar, kullanıcı arayüzünün farklı bölümlerini oluşturmak ve düzenlemek için kullanılabilir. Örneğin, bir ana pencere, birçok farklı widget içerebilir ve bu widget'lar, kullanıcı arayüzündeki farklı işlevleri gerçekleştirir. Widget'lar, genellikle Qt Designer gibi araçlar kullanılarak görsel olarak tasarlanır ve düzenlenir. Her widget, belirli bir işlevi yerine getirmek için özel olarak yapılandırılabilir ve ayarlanabilir.

Özetle, widget'lar, kullanıcı arayüzünde görüntülenen ve kullanıcı etkileşimini sağlayan grafiksel öğelerdir. Qt framework'ünde QWidget sınıfı, widget'ların temelini oluşturur ve kullanıcı arayüzü geliştirme sürecinde yaygın olarak kullanılır.



Burada eklenen widget üzerine sağ tıklanarak promote to ile hangi class ile bağlancağı seçilir. Bu sınıf kullanıcı tarafından oluşturulmuş bir class olabilir ve böylece kullanıcı tarafıdan içeriği düzenlenebilecek bir görsel öğe oluşturulmuş olur.

Bu yöntem ile resim, metin veya 3 boyutlu görsel eklemeleri gerçekleştirilebilir.



Opengl kullanılarak 3 boyutlu nesne çizimi gerçekleştirilebilmesi için bu widget üzerinde qcustomplot seçeneği eklenerek seçilmiştir. Tabi daha öncesinde bu sınıfın projeye dahil edilmesi gerekmektedir.