Qt'ye Giriş

Can Kavaklıoğlu

19 Temmuz 2007

1 Önsöz

Bu yazı Can Kavaklıoğlu[1] tarafından, Cerberus[2] takımının ihtiyaçları doğrultusunda yazılmıştır. Doğu Kaya'ya elzem yardımlarından için teşekkürler.

Yazıdaki hataları, önerileri, soruları ve/veya yazının oluşturduğu tepkileri eposta@cankavaklioglu.name.tr adresine yollarsanız pek de güzel olur.

Herzamanki gibi bu yazıda yazanlar etkisinde kalarak yaptığınız hareketlerden yazar sorumlu değildir.

sürüm 1.2 - Temmuz 2006

2 Gereksinimler

Trolltech'in Qt 4.0 alt yapısını kullanarak programlama yapmak için gerekli kütüphaneleri yüklemek için:

- Debian sisteminde qt4-dev-tools paketini yüklemeniz yeterli olacaktır.
- Diğer birkaç sistemin paketi de mevcuttur[13].
- Herhangi bir Linux sisteminde indireceğiniz .tar.gz dosyası ile kurulumu gerçekleştirebilirsiniz.
- Diğer sistemler için kurlum ve diğer konularda detaylı anlatım için [10].

3 Hello World

Programla ilgili her konuda olduğu gibi ilk önce ekrana "Hello World" yazdıran program ile işe başlayalım:

Bu kodu

```
gcc helloworld.cpp
```

komutu ile çalıştırdığınızda şöyle bir hata kümesi ile karşılaşıyorsanız:

```
helloworld.cpp:1:28: QApplication: No such file or directory
helloworld.cpp:2:27: QPushButton: No such file or directory
```

gcc programı kütüphanelerinizi bulamıyor demektir.

Daha doğrusu qmake kullanarak, düzgün bir Makefile oluştumanız gerekiyor demektir. qmake in de doğru sürümünü kullandığımıza dikkat etmemiz gereklidir. Bunu anlamanın en iyi yolu aşağıdaki komutların çıktısındaki ilk satırlara bakarak, hangi Qt sürümü kullanıldığını yazan satıra göz atmaktır.

Benim sistemimde bu komut şöyle çalışıyor:

```
/****************************************************************
**
** Qt tutorial 1
**
****************************************************************/

#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QPushButton hello("Hello world!");
    hello.resize(100, 30);

    hello.show();
    return app.exec();
}
```

Şekil 1: Örnek: Hello World[11]

```
qmake -project -nopwd  helloworld.cpp
```

Komutu bir adet ¡klasöradı¿.pro dosyası oluşturuyor. Bu dosyanın içerisinde qmake in Makefile ını oluşturacağı projeyi temsil ediyor.

Eğer bulunduğumuz klasördeki bütün .cpp ler projeye katılsın istiyorsak, o zaman son iki parametreyi vermeden de komutumuzu çağırabiliriz:

```
qmake -project
```

.pro dosyamızı oluşturduktan sonra gerekli Makefile ı oluşturması için

```
qmake
```

komutunu çağırıyoruz ve Makefile ımız oluştuyor. Burada dikkat edilecek nokta (en azından benim sistemimde öyle idi) qmake komutu Qt 3.0 sürümünün qmake ini kullanıyordu. Bu da yanlış Makefile oluşmasına sebep oluyordu. O yüzden oluşan Makefile ın ilk satırlarını kontrol ederek orada Qt 4... ifadesinin geçtiğine emin olmak gereklidir. Benim sistemimde bunu yapabilmek için qmake yerin şu komutu kullanmam gerekti:

```
qmake-qt4
```

Düzgün bir Makefile oluştu ise, Makefile ın bulunduğu klasörde,

```
make
```

komutu ile ilk Qt programımızı oluşturduk demektir. Çalıştırmak için komut satırından

```
./<klasöradı>
```

komutu ile programı çalıştırabiliriz.

4 Sinyaller ve Yuvalar (Signals and Slots)

Qt'nin objeler arası iletişim için kullandığı yapılar Sinyaller ve Yuvalar[8] olarak adlandırılırlar.

Throlltech'in dediğine göre eski sistemlerde kullanılan birbirine pointer gönderip, "callback" yaptırmak, hem "type-safe" olmadığı, hem de "strongly coupled" olduğu için sakıncalı oluyormuş. Bu yüzden bu yeni sistemlerini oluşturmuşlar.

Binevi Sony'nin OPEN-R sistemindeki gibi Subject(Signal) ve Observer(Slot) lar mevcut. Bir sinyal iki yuvaya bağlanabiliyor. Yukariki iki probleme verilen cevap ise şöyle:

- Bağlantılar tek elden bir fonksiyon imzası (function signature) ile belirtildiği için tip çakışma problemleri yaşanmıyor. Hatta bu açık ayırım sayesinde derleyici bize olası hataları gösterebiliyor.
- "Loosely coupled" yapı sayesinde, sinyali yayan objeler, yuva sahibi objelerle kurdukları bağlantılarda direk ilişki kurmadıkları için, "strongly coupled" sistemlerdeki sinyal yayan objelerin, bütün yuva sahibi objelerden haber olma zorunluluğu yoktur.

Sinyal ve yuvaların birkaç özellikleri:

- Sinyaller ve yuvalar, bağlantılarından haberdar değillerdir.
- Bir yuvaya gerektiği kadar sayıda sinyal bağlanabilir.
- Bir sinyale gerektiği sayıda yuvaya bağlanabilir.
- Sinyaller sinyallere de bağlanabilir, bu durum, birinci sinyal oluştuğunda, ikinci sinyali tetikleyecektir.
- Bir sinyal oluştuğunda, kodun çalışması sinyali oluşturan satırın altına, ancak sinyalin yuvasından geri dönüş olduğunda devam eder. Eğer bu durum yerine, yuvadaki işlemin ayrı bir süreç olarak devam etmesi isteniyorsa, "sıralı bağlantı" (queued connections) yöntemi kullanılır.
- Yuvaların erişim hakları, hangi sinyallerin bu yuvaya bağlanabileceğini belirtir.

4.1 Sinyal ve Yuvalar Örneği

2. şekildeki örnekte Counter (sayaç) sınıfı, bir yuva ile tuttuğu sayıyı değiştirir. Oluşturduğu bir sinyal ile de, kendisini izleyenlere, sakladığı değerin değiştiğini haber verir.

```
#include <QObject>

class Counter : public QObject
{
    Q_OBJECT

public:
    Counter() { m_value = 0; }

    int value() const { return m_value; }

public slots:
    void setValue(int value);

signals:
    void valueChanged(int newValue);

private:
    int m_value;
};
```

Şekil 2: Örnek: Sinyaller ve yuvalar[8]

Saklanan değişmesi sinyalinin üretilmesinin en uygun noktası ise setValue fonksiyonu olacaktır:

```
void Counter::setValue(int value)
{
        if (value != m_value) {
                m_value = value;
                emit valueChanged(value);
        }
}
```

Şekil 3: Örnek: Sinyaller ve yuvalar: setValue fonksiyonu [8]

Dikkat edilecek satır *emit valueChanged(value);* satırıdır. Bu komut ile, *signals:* ifadesi ile belirttiğimiz bölümdeki *void valueChanged(int newValue);* fonksiyon çağırılarak oluşturulan sinyal yayılır.

Daha açık bir örnek için 4. şekile bakınız:

```
Counter a, b;
QObject::connect(&a, SIGNAL(valueChanged(int)),
                 &b, SLOT(setValue(int)));

a.setValue(12);     // a.value() == 12, b.value() == 12
b.setValue(48);     // a.value() == 12, b.value() == 48
```

Şekil 4: Örnek: Sinyaller ve yuvalar: bir sinyali yuvaya bağlamak [8]

Bu örnekte gösterilen işlemler şunlardır:

1. İki tane Counter objesi, *a* ve *b* oluşturulur.
2. *a* ve *b* objeleri birbirlerine bağlanırlar. Bağlama komutundaki değiştirge sıralaması şöyledir: *sinyali oluşturan obje, sinyal, sinyali alan obje, yuva*. Bu şekilde sinyal ve yuva tanımlarken, SIGNAL ve SLOT makroları kullanılmak zorundadır.
3. *a.setValue(12);* komutu *a* objesinin sakladığı değeri 12 ye eşitler, ve hemen ardından, *valueChanged(int)* sinyali oluşturmasını sağlar. Bu sinyal *b* objesinin *setValue(int)* yuvasına ulaşır ve *b* objesinin sakladığı değer de 12 ye eşitlenir.
4. *b.setValue(48);* ise sadece *b* objesinin değerini değiştirebilir, çünkü *a* objesinin yuvası hiçbir sinyale bağlanmadığı için, *b* objesinin *setValue(int)* fonksiyonu tarafından oluşturulan sinyal *a* objesine ulaşmaz.

5 Meta-Obje Sistemi

Qt'nin Meta-Obje Sistemi[15], başta sinyal ve yuvalar sistemi olmak üzere birkaç işlev sunar:

- QObject::metaObject() ilgili sınıfın meta objesini verir.
- QMetaObject::className() RTTI desteğine gerek duymadan çalışan sınıfın adını obje çalışırken verir.
- QObject::inherits() istenen sınıfın ilgili sınıf tarafından kullanılıp (inherit) kullanılmadığını belirtir.
- QObject::tr() ve QObject::trUtf8() karakter dizilerini istenen dile çevirir.
- QObject::setProperty() ve QObject::property() dinamik olarak obje özelliklerinin değiştirir ve verir.

Meta-Obje Sistemi aynı zamanda "dynamic_cast()" işlemi için de kullanılabilir.

6 Özellikler Sistemi

Qt'nin Özellikler Sistemi (Property System [14]) bir sınıfın içerisindeki standart veri yapılarına denk gelir. Ancak aşağıda belirtilen birkaç önemli işlev sağlar:

- Okuma fonksiyonu her zaman mevcuttur.
- İstenirse yazma fonksiyonu her zaman mevcut olabilir.
- Kayıtlı ("stored") özniteliği, özelliğin kaydedilip edilmediğini belirtir.

- Ender olsa da yeniden başlatma ("reset") fonkiyonu.
- Tasarlanabilir ("designable") özniteliği, ilgili özelliğin, Qt Designer gibi bir tasarlama programında gösterilip, gösterilmeyeceğini belirtir.

Q_PROPERTY() makrosu kullanılarak tanımlanır. Bu makro ancak QObject'i kullanan (inherit) sınıflarda kullanılabilir.

6.1 Özellik Okumak ve Yazmak

```
QPushButton *button = new QPushButton;
QObject *object = button;
// button and object point to the same object

button->setDown(true);
object->setProperty("down", true);
```

Şekil 5: Örnek: Özellik okuma yazma örneği[14]

`button->setDown(true)`

yöntemi derleyicinin, derleme sırasında hataları raporlayabilmesini sağlar.

`button->setPropety("down",true);`

yöntemi ise derleme sırasında kullanılabilir olmayan özellikleri programın çalışması sırasında erişebilmemizi sağlar.

Özellikler Sisteminin diğer belirtilen diğer iki fonksiyonu ise şunlar:

- QMetaObject::propertyCount() bir objenin kullanılabilecek bütün özelliklerini verir.
- QMetaObject::property() istenen objenin istenen özelliğini bir QMetaProperty objesi olarak verir.

6.2 Özellikler Sistemi Örneği

```
class MyClass : public QObject
{
    Q_OBJECT
    Q_PROPERTY(Priority priority READ priority WRITE setPriority)
    Q_ENUMS(Priority)

public:
    MyClass(QObject *parent = 0);
    ~MyClass();

    enum Priority { High, Low, VeryHigh, VeryLow };
    void setPriority(Priority priority);
    Priority priority() const;
};
```

Şekil 6: Örnek: Özellik okuma yazma örneği[14]

Bu tanımlamayı kullanmak için şöyle bir ifade deneyebiliriz:

```
obj->setProperty("priority", "VeryHigh");
```

Q_PROPERTY makrosunun yapısı şu şekildedir:

```
Q_PROPERTY(type name
           READ getFunction
           [WRITE setFunction]
           [RESET resetFunction]
           [DESIGNABLE bool]
           [SCRIPTABLE bool]
           [STORED bool])
```

- *name*, QVariant[16] tarafından desteklenen bir tip olabilir yada yukarıdaki örnekteki gibi sınıf içerisinde tanımlanan bir tip olabilir.
- *getFunction*, const olmalıdır. Döndüğü cevabın tipi da, ya ilgili tipin kendisi, ya o tipte bir pointer yada yine o tipte bir referans olmalıdır.
- seçmeli *setFunction*, void dönmelidir ve sadece tek bir değiştirge ("parameter") almalıdır. Alınan değiştirgenin tipi, ilgili tipin kendisi, ya ilgili tipte bir pointer, yada ilgili tipte bir const reference olmalıdır.
- seçmeli *resetFunction*, özelliği ilk haline geri çevirecek fonksiyon. Hiçbir değiştirge almamalı ve void dönmelidir.
- seçmeli *DESIGNABLE*, GUI tasarım arıcında bu özelliğin kullanımı açıp kapatmak için kullanılır. Öntanımlı değer yazılabilen özellikler için evettir.
- seçmeli *SCRIPTABLE*, Betik motoru tarafından kullanıma uygunluğu belirtir. Öntanımlı değer evettir.
- seçmeli *STORED*, Objenin durumu kaydedilirken, bu özelliğin değerinin hatırlanıp hatırlanmayacağını belirtir. Öntanımlı değer evettir. Yazılamayan değerler için kullanılması mantıklı değildir.

7 Qt - Eclipse bağlantısı

10 Temmuz'da Eclipse ile Qt yi birbirine bağlamak için bir eclipse eklentisi test sürümü yayınlandı[17].

Qt eklentisi yüklemesini yapmadan önce Eclipse'te C/C++ desteği için kullanılan CDT eklentisini kurmanız gereklidir. Bu işlemi Eclise'in Help > Software Updates > Find and install menüsünden yapabilirsiniz. Eliniz değmişken bir de Eclipse sisteminiz için güncelleme yaparsanız iyi olacaktır. Qt - Eclipse bağlantısı eklentisi test sürümü olduğu için, Eclipse'i güncel tutmakta fayda olacaktır.

Haberde verilen bağlantıdan[18] indirebileceğiniz paketi, eclipse klasörünüzün üzerine açtığınız zaman Help > Cheat Sheets > Qt Development yoluyla açabileceğiniz bir anlatım mevcut.

Bu anlatımda anlatıldığı üzere, Window > Preferences > Qt panelinden, sistemdeki Qt yüklemenizin yerini belirtmeniz gerekiyor.

Kullandığım Debian sistemde libqt4* paketleriyle kurduğum Qt kurulumunda, istenen değerleri 4.0 ve /usr olarak girdiğimde Qt yi Eclipse'e göstermeyi başardım.

Bir not, eğer qt3 sürümü önceden yüklü ise, qmake komutu eski qmake-qt3 komutuna işaret ediyor olabilir. Bu durumda komut satırından çağırdığınız qmake komutunun doğru qmake (qmake-qt4) komutuna işaret ettiğinden emin olun.

Eğer Qt yüklemenizi başarıyla belirtebilirseniz, New > Project > Other > Qt ve Qt Designer menüsünden yeni bir Qt projesi başlatabilirsiniz.

İşaret edilecek özellikler:

- Window > Show View menüsünden seçilebilecek bir çok Qt aracı.
- ui dosyalarını Eclipse'in içinden Qt Designer kullanarak düzenleyebilme.

Kaynaklar

[1] http://cankavaklioglu.name.tr/

[2] Cerberus http://robot.cmpe.boun.edu.tr/aibo/home.php3

[3] The Independant Qt Tutorial - http://www.digitalfanatics.org/projects/qt_tutorial/

[4] Qt 4.0: Qt Tutorial - http://doc.trolltech.com/4.0/tutorial.html

[5] Qt 4.0: How to Learn Qt - http://doc.trolltech.com/4.0/how-to-learn-qt.html

[6] Qt 4.2 Whitepaper - http://dist.trolltech.com/pdf/qt42-whitepaper-a4.pdf

[7] Qt 4.0: Qt Object Model - http://doc.trolltech.com/4.0/object.html

[8] Qt 4.0: Qt Signals and Slots - http://doc.trolltech.com/4.0/signalsandslots.html

[9] Qt 4.0: Examples - http://doc.trolltech.com/4.0/examples.html

[10] Qt 4.0: Installation - http://doc.trolltech.com/4.0/installation.html

[11] Qt 4.0: Qt Tutorial 1 - http://doc.trolltech.com/4.0/tutorial-t1.html

[12] Qt 3.1: qmake User Guide - http://doc.trolltech.com/3.1/qmake-manual.html

[13] Downloads – Trolltech - http://trolltech.com/developer/downloads

[14] Qt 4.0: Qt's Property System - http://doc.trolltech.com/4.0/properties.html

[15] Qt 4.0: Meta-Object System - http://doc.trolltech.com/4.0/metaobjects.html

[16] Qt 4.0: QVariant Class Reference – http://doc.trolltech.com/4.0/qvariant.html

[17] Trolltech Provides Qt Developers Access to Eclipse with Integration Release Canadidate – http://trolltech.com/company/newsroom/announcements/press.2007-07-10.3777186202

[18] http://trolltech.com/download?target=ftp://ftp.trolltech.com/qteclipse/qt4eclipse-0.0.7-linux.x86.tar.gz