Qt ile GUI Programlama'ya Giriş

Kaya Oğuz

<kaya@kuzeykutbu.org>

Sürüm 1.0.1

Bu belge "yardımcı" olması amacıyla hazırlandı. Dökülen saçlarınız, kaybettiğiniz bilgileriniz, zamanınız ve aklıma gelmeyen diğer zararlar için mesuliyet kabul etmiyorum :)

Table of Contents

Önsöz
1. Hazırlıklar
    1. Gerekli Ön Bilgiler
    2. GNU/Linux ve GUI Araçları
    3. Qt Hakkında
    4. Neden Qt?
    5. İndirmeniz Gerekenler
    6. Arayüz Tasarımı Hakkında
2. Qt ile "Merhaba Dünya"
    1. O biiir... klasik!
    2. GUI Programlama Terimleri
3. Qt Temelleri
    1. QObject, QMake, Makrolar, MOC; Özel Durumlar
    2. Assistant Hakkında
    3. Pencereler
        3.1. İlk Penceremiz
        3.2. Daha çok parçacık ve Yerleşim
    4. SIGNAL ve SLOT
4. Designer
    1. Designer
    2. Menü ve Araç Çubuğu Düzenleme
    3. Simgeler
    4. Tam bir proje?
5. QMetin
    1. Arayüz Tasarımı
    2. Programın Tasarımı
    3. Kodlamaya Girişiyoruz
    4. QMake
    5. Kodların İncelenmesi
6. Son Söz

Önsöz

Qt ile ilgili böyle bir belge hazırlamak uzun zamandır aklımda. Sürekli ertelenmesinin nedenlerinin başında Qt ile daha yeni olmam geliyordu. Bir diğeri ise Qt'nin 4. sürümünün çıkmasına az kaldığı zamanlardı. Qt 3'ü temel alan bir belgenin 4'ün çıkmasına az kala bir anlamı olmayacaktı elbet. 4. sürümün çıkması ve benim olan biteni kavramamla tekrar niyetlendim. Okulda projelerde kullandıktan sonra bir iki küçük belge hazırlığına giriştim, ama hep engeller çıktı, biraz da vaktim olmadığı için motivasyonum eksildi.

Son olarak Qt Türkiye'nin ortaya çıkması ile belge daha da bir önem kazandı! Bu motivasyonumu arttırdı ve sonunda bitirebildim :)

Qt, diğer arayüz araçlarıyla karşılaştırıldığında gerek belge, gerekse de büyüklük ve kalite açısından en iyilerden biri. Öğrenmesi kolay, uygulama yazması zevkli. Bu yüzden kısa zamanda yeni uygulamaların ortaya çıkacağını düşünüyorum.

Baştan söyleyeyim: Qt ile ilgili her şeyi bilmiyorum :) Sadece uğraştığım alanlarda bilgi sahibiyim ve bildiklerimi bu belgeye döküp paylaşmak istedim. Hatalarım varsa mutlaka e-posta gönderin. Düzeltilmesi gereken yerler ve cümle düşüklükleri gibi hatalar olabilir, bunları da bildirirseniz belgenin daha iyi olmasına katkıda bulunmuş olursunuz.

Bu belgenin başlığında "Giriş" yazmaktadır :) O yüzden size temel bilgileri verip ivme kazanmanızı sağlamak istedim. Yazacak o kadar çok şey var ki, belki onları daha sonraki belgelere katmakta fayda olabilir...

Bu belge XXE kullanılarak DocBook olarak yazılmıştır. LaTeX ve LyX'ten sonra kullanmanın en zevkli olduğu belge yazma ortamı olduğunu söylememe gerek yok...

Chapter 1. Hazırlıklar

1. Gerekli Ön Bilgiler

Bu belgeyi yazarken, C++ bildiğinizi düşündüm. Eğer eksikleriniz olduğunu ya da hatırlamanız gerektiğini düşünüyorsanız bir tekrar etmenizde fayda var. Tabii bunun dışında temel bir bilgisayar kullanımı, az biraz Linux (Qt'yi Windows üzerinde kullanacaksanız, az biraz Windows™) bilmeniz gerekiyor. Sabırlı olmak, ve en önemlisi sadece bu belge ile yetinmemek gerekiyor.

2. GNU/Linux ve GUI Araçları

Linux işletim sistemini günlük olarak kullanmaya başladığınızda bazı yazılımların farklı arayüz araçlarıyla hazırlandığını fark etmeniz uzun zamanınızı almaz. İlk olarak duyacağınız kelimeler KDE ve GNOME'sa, bunların da temel yapıtaşları olan Qt ve GTK'yı mutlaka bir şekilde duymuşsunuzdur. Bu ikisinin dışında wxWidgets, FLTK, FOX, XForms, Tcl/Tk ve benim şu an aklıma gelmeyen bir yığın daha aracın olduğunu söylemek istiyorum. Fakat GTK ve Qt, uygulama sayısında diğerlerine nazaran daha yüksekler.

Neden bu kadar çok araç var? Bu konu biraz uzun da olsa kısaca değinmekte fayda

var. Linux üzerinde grafiksel işlemler için, bildiğiniz gibi, X sunucusu kullanılmaktadır. X sunucusu beraberinde XLib denen bir kitaplık ile gelir. Bu kitaplık kullanılarak görsel programlama yapabilirsiniz, ama bunu istemeyeceğinizden emin olun. XLib kitaplığı düşük seviyede yazmanızı zorunlu tutar. Düşük seviyeden kasıt temele daha yakın olmasıdır, bu yüzden bir arayüz hazırlamak uzun vaktinizi alabilir.

Bu sebepten dolayı, XLib kitaplıklarını sarmalayan bu arayüz araçları çıkmıştır. Bu sayede belli sınıflarla hızlı uygulamalar geliştirilebilir. Bu kitaplıkların ilk örnekleri XForms, Motif gibi şimdi görseniz hemen eski olduğunu anlayacağınız görsel olarak beğenilmeyecek fakat hızlı olan arayüzlerdir. Bu sıralar daha "modern" kabul edilen Qt, wxWidgets ve GTK gibi kitaplıklar kullanılır.

GTK, C tabanlıdır. GLib denen bir yardımcı kütüphane kullanır. Bu kütüphane ile daha esnek uygulamalar yazılır. Değişkenler de glib'in gint, gchar gibi başlarına g alan değişkenleridir. GTK için C++, Java, Python vs. dillerde bağlar vardır. Buna rağmen, belki de bana öyle geldi, öğrenmesi ve uygulama geliştirmesi, en azından Qt'ye göre biraz daha uğraştırıcıdır. Uzun süre GTK uygulamaları kullanmış biri olarak, GTK öğrenmeyi denediğim hemen her girişimden saçlarımı yolarak ve sinirli bir şekilde vazgeçtim.

Ta ki yüzüne bakmadığım Qt'ye bir göz atana kadar.

> **Note**
>
> Burada Linux üzerinde temel alınarak anlatılacak olsa da, bunların çoğunu Qt'nin çalıştığı her yerde (Mac™ ve Windows™) uygulayabilirsiniz.

3. Qt Hakkında

Qt, Trolltech denen, Kuzey Avrupa'nın refah dolu ülkelerinden Norveç'te kurulmuş bir şirket tarafından geliştiriliyor. Norveç, balıkçıları dışında, Qt ve yine yazılım dünyasından hızlı web tarayıcı Opera ile de ünlü bir ülkedir. Nitekim bildiğim kadarıyla Opera'nın Linux sürümü Qt kullanmaktadır.

Ben "küu-ti" diye okumaya alışmış olsam da, doğru okunuşu İngilizce "Cute" kelimesi gibidir. Yani "I'm a Qt programmer" dediğinizde "Ben güzel bir programcıyım" demiş gibi olursunuz... Norveç'in havası gibi soğuk bir espri :)

KDE masaüstü için Qt'nin kullanılması ilk zamanlarda gayet tepkiyle karşılandı. Çünkü Qt o zamanlar "GPL" lisanslı değildi. Tamamen özgür bir işletim sisteminin masaüstünün kapalı bir arayüz aracına bağlı olması elbette biraz tepki çekti, çünkü KDE giderek popüler oluyordu. Bu iki sonuç doğurdu:

- GTK kullanılarak GNOME masaüstünün doğması
- Qt'nin önce QPL sonra da tamamen GPL olarak dağıtılması

Qt, GPL olarak yayınlandıktan sonra, Qt benzeri "açık" geliştirilen bir proje de durduruldu.

Burada GPL lisansı hakkında biraz konuşmak lazım. GPL lisansı ile lisanslanan yazılımlar hep GPL kalmalıdırlar! Bu yüzden Qt ile yazdığınız yazılımlar GPL olmalıdır.

Bunun anlamı uygulamanız ile birlikte kaynak kodları da **sunmalısınız.**

Eğer niyetiniz ticari bir yazılım yapmak ve kodları açmamaksa, Qt'nin ticari ekler içeren ticari sürümünü satın alabilir, GPL lisansı yerine istediğiniz lisansı kullanabilirsiniz.

Qt hakkındaki bu kısa tarih ve bilgilendirmeden sonra bölümün asıl konusuna geçelim...

4. Neden Qt?

Aslında zaten Qt öğrenmek istiyorsunuz, bunun Neden'i sorulmaz, yani zaten kandırmışsınız kendinizi, burada Qt'ye yağ çekmeme gerek yok. Ama gerçekten de Qt çok eğlenceli! Eğlenceli olmasının nedeni bir şeyleri başarabilmeniz ve de yazabilmeniz :) Öğrenmesi kolay, yapı olarak kararlı, düzenli ve derli toplu. Dahası KDE programlamaya ilk adım. KDE üzerinde geliştirilmiş birçok proje var ve bu masaüstüne katkıda bulunabilmek için güzel bir şans...

Qt'nin diğer bir olayı da yazdığınız programın diğer platformlarda da çalışması. Yani Windows'ta Qt programlama öğrendikten sonra aynı şekilde Linux'ta da geliştirebilirsiniz. Arkadaşınız Mac kullanıyorsa (benim kullanıyor, Melih) onun başı kel mi? Onda da çalışıyor :)

5. İndirmeniz Gerekenler

Eğer sisteminizde Qt 4 yüklü değilse, Trolltech'in web sayfasından, ya da dağıtımınıza özel paketleri bularak Qt'nin 4. sürümünden en yenisini (bu yazı sırasında 4.1.3) indirip derleyin / kurun. Sisteminizde Qt 3 yüklüyse sorun yapmayın, çünkü Qt 4'ün çalıştırılabilir dosyaları farklı (örneğin qmake-qt4). İndireceğiniz elbette "Open Source Edition"...

Bunun dışında, elbette, sisteminizde make ve gcc paketleri kurulu olmak zorunda. Derleme için gerekli bütün paketleri indirin, kurun.

> **Note**
>
> Windows™ üzerinde kurulumu herhangi bir programın kurulumu gibi Next Next şeklinde... Kurulum sırasında size MinGW var mı diye soracak. Şimdiki gibi "O ne?" diyorsanız, yok deyin, kendi indirip kurar. MinGW(=Minimal GNU for Windows) Windows™ sisteminize GCC ve araçlarını kurmanızı sağlar

> **Note**
>
> Bir not daha... Derlemek için düz bir komut penceresi (terminal penceresi, adı her neyse) açmayın. Başlat menünüzde Qt altında bir Qt Terminal Window çıkması lazım, oradan derleme işlemlerini vs. yapın...

KDE masaüstündeyseniz düzenleyici olarak Kate / KWrite ikilisinden (tercihen Kate) kullanabilirsiniz. Bu ikisi hafif olmakla beraber Qt'ye özel renklendirmeler de içermektedirler.

IDE olarak kullanmayı seven arkadaşlar KDevelop, Eclipse gibi devlere bir iki ince ayar çekerek Qt'yi buradan kullanabilirler.

6. Arayüz Tasarımı Hakkında

Buna da özel olarak değinmek istiyorum. Masaüstünde bütün zamanımızı geçirdiğimiz için hepimiz kendimizi GUI arayüz tasarlamada *uzman* gibi hissediyor olabiliriz. İnanın bu yanılgıya düşmek çok kolay. Burada Qt hakkında yazacağımız için konumuz dışında kalıyor ama GUI tasarlama konusunda internetteki çeşitli siteleri ziyaret etmenizde fayda var. Google'da bir arama güzel belgelere yönlendiriyor bizi. Bunlara göz atmak, okumak ve iyi bir arayüz tasarlamak elimizde.

Özellikle GNOME, kendi oluşturduğu "User Interface Guidelines" belgesine uymayan uygulamaları GNOME'a dahil etmiyor. Sırf bu yüzden Galeon denen tarayıcı yerine Epiphany'yi öntanımlı tarayıcıları yaptılar. Netten bu belgeleri de bulup okumanız tavsiye edilir. Ben tembellik edip buraya bağlantılarını koymadım diye, lütfen siz de etmeyin :)

Chapter 2. Qt ile "Merhaba Dünya"

1. O biiir... klasik!

Bütün programlama belgelerinde ilk yazılan uygulama "Merhaba Dünya"dır. Herhalde en çok port edilen uygulama olduğunu söylesek yanlış olmaz. Sırf bu "Hello World"leri toplayan bir site var internette...

Burada bir sürü tantana yapmak yerine, ilk Merhaba Dünya kodunuzu sunmakta fayda görüyorum. İlk bölümü okuduysanız "Hadi başlayalım" dediğinizi biliyorum, insanoğlu sabırsızdır... İşte bütün o Qt ile ilgili detaylardan önce "Merhaba Dünya"...

Example 2.1. Merhaba Dünya

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
        QApplication app(argc, argv);
        QLabel merhaba("Merhaba Dunya!");
        merhaba.show();
        return app.exec();
}
```

Bu yukarıdakileri yeni bir dizin altında main.cpp olarak kaydedin. Umarım buradan kopyala yapıştır yapmadınız :) Yaptıysanız da olsun. Bunu yapıştırmış olsanız bile, aşağıdaki komutları sırayla vermeniz gerekmekte :)

1. qmake-qt4 -project
2. qmake-qt4
3. make

Burada qmake ile qmake-qt4 arasındaki farka biraz değinelim. Bu komutun asıl adı qmake, PiSi paketine özel mi bir -qt4 var açıkçası araştırmadım, ama Qt 3 ve 4 şu an bu -qt4'ler sayesinde sistemimde rahat rahat çalışıyor. Windows üzerinde ve Qt 3 ile qmake komutu kullanıyorsunuz. Metnin devamında qmake gördüyseniz bu Qt 4'ün qmake'idir :)

Peki biz bu 3 adımda ne yaptık? Verdiğiniz ilk komut sonrasında dizinde "pro" uzantılı bir dosya oluşacak. Bu dosya Qt uygulamanız hakkında çeşitli bilgiler içeriyor. Bundan sonra gelen qmake komutu bu dosyayı okuyarak bir Makefile oluşturuyor. Bir linux kullanıcısı olarak bu Makefile denen olayı ilk kere duyuyor olamazsınız. Ama ben yine de üzerinden geçeyim.

Makefile, bir programın derlenmesi için gereken işlemleri içeren bir tarif dosyasıdır diyebiliriz kabaca. Örneğin onlarca c kodu dosyası olan bir projede iki de bir gcc komutu ile derleme yapmak, sonra hepsini bir çalıştırılabilir dosyaya çevirmek kabus olabilir. Makefile derlenecek dosyaları, gcc veya belirtilen derleyiciye verilecek parametreleri vs. tutar. Eğer bu komut sonrasında oluşan Makefile dosyanıza bakarsanız biraz uzun tuttuğunu (bende 170 satır kadar) göreceksiniz. Her zaman bu kadar uzun olmasa da Qt için bu basit uygulamada bile uzun olmasının nedenleri var elbette. Bu nedenler de hemen koda atlamamızın ve başınızı ağrıtmak istememe nedenlerime dayanıyor. Ama elbette kaçış yok, anlatacağım :)

O konulara girmeden önce, son komut olan make ile programı derliyorsunuz. Derleme işlemi bittiğinde dizin adıyla aynı adı taşıyan bir çalıştırılabilir dosyanız olacak. Onu çalıştırdığınızda karşınıza işte bu gelecek:

Figure 2.1. Merhaba Dünya Ekran Görüntüsü

Bu noktadan sonra bana soracağınız bir sürü soru olmalı. İlki neden "Merhaba Dunya" ve "Merhaba Dünya" değil? Bu yukarıdaki kod nasıl oldu da çalıştı? Bir derleme için niye 3 tane komut verdim, bu komutlar neler yaptı?

Hemen anlatmaya girişeyim. "Ü" harfini kullanmama nedenim sistemimin UTF-8 olması ve UTF-8 dizgeleri kullanmak için QString::fromUtf8()'i kullanıp başınızı ağrıtmak istememem. Bu üç komutu vermiş olsanız da yeni dosya eklemediğiniz sürece, yani kod yazarken değişiklikleri görmek için sadece make komutunu vermeniz yeterli.

Gelelim asıl konuya; nasıl çalıştı? Qt'de temel olarak bir QApplication yarattıktan sonra bir parçacık (widget) yaratmalısınız. Bu parçacık bir pencere olabilir. Pencere değilse kendini bir pencerede gösterir. Burada QLabel denen -- bütün Qt parçacıkları Q ile başlıyor -- bir Label, yani etiket kullandık. Kullanırken içine ne yazacağımızı da belirttik. Sonra uygulamayı çalıştırdık. Bu en temel olduğu için Qt'nin bir çok özelliğinden mahrum bırakılmış durumda, ama sonuçta ilk Qt uygulamanızı yazdınız.

2. GUI Programlama Terimleri

İlk uygulamanızı yazdığınıza ve biraz rahatladığınıza göre, bir süre o metin düzenleyicileri kapatın bakalım, azcık başınızı ağrıtayım. Şu an bunu okuduğunuz yerin bir internet tarayıcısı olduğunu düşünürsek onun arayüzünde neler

gördüğünüzü konuşalım.

Eğer sadece bu sayfa açıksa, program öylece duruyor olmalı. Yani hiçbir şey yapmıyor. Öyle size bakıyor, siz de ona bakıyorsunuz. Ne güzel. İşte bu durum programın "Ana Döngüsü" ya da yabancı deyimiyle "Main Loop". Bu sırada program aslında bakmanızı değil, bir şeyler yapmanızı bekliyor. Mesela bir yerlerine tıklamanızı, sayfa çok uzunsa aşağı doğru kaydırmanızı ya da bir klavye kısayolu ile bir işlem yapmanızı.

Bu yaptığınız, ya da yapmayı düşündüklerinize de "olay", "event", deniyor. Bir olay olduğunda, genelde program buna bir tepki verir. Bunun nedeni bir olay olduğunda bir SİNYAL gönderilir. Bu sinyal bir yere bağlıysa, o metod çalışır, gerekli işlemleri yapar. Bu olaya da "olay güdümlü programlama", "event-driven programming", deniyor.

Şimdi biraz daha bakınalım. Bu tıklayabileceğiniz yerler arasında düğmeler, menüler falan var. Hatta bu yazıyı okuduğunuz ve tarayıcının neredeyse yüzde doksanını kaplayan bir alan var. İşte bunların hepsi, yukarıda dediğim "parçacıklar". İngilizcesi Widget olan bu kelime, Window Gadget'tan oluşmuş, yani Pencere Şeysi gibi bir şey. Qt'nin kocaman bir parçacık desteği var. Gördüğünüz ve alışık olduğunuz parçacıklar dışında belki daha az karşınıza çıkmış olanları da görebilirsiniz.

Düğmelere yakında bakarsak sol başta, yüksek ihtimalle "Geri" düğmesi vardır. Siz bu düğmeye bastığınızda bir önceki sayfaya gidersiniz. Aynı şekilde menüdeki "Git" menüsünde de bir "Geri" vardır, yanında da "Alt + Sol Ok" yazmaktadır. Vay, yani o "Geri" düğmesine basmak yerine menüden "Git"'in altındaki "Geri"ye de basabilirim, o da zor gelirse klavye kısayolunu kullanabilirim. Aynı işlem için üç ayrı yol... Hepsi de aynı işlemi yapıyor. İşte birden fazla yerde kullanacağımız bu tip işlemler için Qt'nin QAction sınıfını kullanıyoruz.

Şimdi, pencerenin ekranı kapladığını olduğunu düşünürsek, bütün parçacıklar mutlu mesut duruyorlar. Peki pencereyi tekrar boyutlandırsak, mesela küçültsek, o zaman ne olur? Bütün parçacıklar daha siz küçültürken tekrar yerleşirler. Düğmelerin boyu aynı kalırken bu yazıyı okuduğunuz alan küçülür ve yazılar biraz daha aşağıya uzar vs. İşte bu işlem Qt'de "Yerleşim", "Layout" ile yapılıyor. Birkaç yerleşim şekli var, mesela parçacıkları yatay ya da dikey yerleşmesini isteyebilirsiniz. Ya da bir ızgaraya (grid) koyabilirsiniz. Bu yerleşimler sayesinde parçacıklarınız yeni boyutlara kendilerini adapte edebilirler. Boyutlarının sabit kalmasını istediğiniz parçacıkları sabit tanımlayabilirsiniz.

Arayüzle ilgili ne kadar çok konu varmış değil mi? Daha bitmedi. Menüye bakarsanız menü başlıklarının bazı harflerinin altlarının çizili olduğunu görebilirsiniz. Diyelim ki "Dosya" menüsünde D harfinin altı çizili. O zaman Alt+D yaptığınızda o menü açılacaktır. O menü açılınca klavye kısayolları gözükse de, menü içinden seçmek için yine bazı menü girdilerinin bir harflerinin altı çizilidir. Oklarla aşağı inip onu seçmeniz gerekmez. Bunu da Qt ile yapacağız.

Son olarak, arayüzün içindeki parçacıklar TAB tuşu ile gezebilirsiniz. Büyük ihtimalle "Geri" düğmesinden başlayıp sağa doğru teker teker her parçacığı seçebilir, hatta Shift+TAB ile bir öncekine geçebilirsiniz. İşte bu "sıralama"yı da yapacağız. Dikkat etmemiz ve kodlamamız gereken ne kadar çok şey varmış değil mi? Bir de "Merhaba Dünya"ya bakın, bunların neredeyse hiç biri yok.

Chapter 3. Qt Temelleri

1. QObject, QMake, Makrolar, MOC; Özel Durumlar

Kod yazarken karşınıza çıkacaklara şaşırmamanız için size biraz Qt'nin özel durumlarından bahsedeceğim.

Qt'nin sınıfları QObject denen bir sınıftan türemişlerdir. Bütün Qt nesne modelinin temelinde bu sınıf bulunur. Bütün bu olay güdümlü programlamanın (signal / slot) bu sınıfın başının altından çıktığını söyleyebiliriz.

QObject ve ondan türeyen nesneler, *kodlarken* bildiğiniz C++ sınıflarından *biraz farklı* formatlar barındırırlar. Bir sınıf türeteceğiniz zaman, sınıf tanımından sonra gelen süslü parantezin hemen başına `Q_OBJECT` denen makroyu eklemeniz gerekmektedir. Bundan başka sinyal ve slotları tanımlamak için public / private / protected gibi erişimi belirleyen anahtar kelimelerin sonuna slots ve signals eklemeniz gerekmektedir. Örneğin:

Example 3.1. Basit bir türetme

```
class ornekSinif:public QObject
{
        Q_OBJECT
        public:
        ornekSinif():QObject()
        {
                // constructor ile ilgili kod

                // burada aynı zamanda sinyal
                // slot bağlantılarını da yapabilirsiniz
        }
        public slots:
        void olay();
};
```

Buradaki "`public slots`" ifadesinin normal bir C++ derleyici ile sorun çıkaracağını düşünmemek saçmalık olur. İşte bütün hepsi bir araya QMake ve MOC denen Meta Object Compiler ile bir araya geliyor.

QMake, Qt ile bağlantılı sınıfları bu MOC'tan geçirecek bir Makefile üretiyor. QMake aslında daha birçok şey yapıyor, mesela Designer ile üreteceğimiz UI dosyalarını, simgelerin bulundukları QRC dosyalarını hep uygun komutlardan geçiriyor. Bu sınıflar MOC ile işlendikten sonra, mesela yukarıdaki ornekSinif.cpp, moc_ornekSinif.cpp olarak dizinde yer aliyorlar. Eğer bu dosyaların içine bakarsanız bu "public slots" gibi olayların uygun şekilde C++ kodlarına dönüştürüldüğünü göreceksiniz. Arkaplanda bu yaptıklarından sonra GCC ile güzel güzel derliyor... Bunun bize kazancı daha temiz ve rahat kod yazmamız. Şimdi en azından Qt sınıflarını üretirken, türetirken arkada olan biteni az-çok biliyorsunuz.

Burada slot olarak tanıttığımız `void olay()` fonksiyonunu connect() denen bir fonksiyon ile kullanabilirsiniz. Bunun detaylarını ilerleyen başlıklarda göreceğiz.

2. Assistant Hakkında

Bunu daha sonra yazacaktım, ama daha yukarılara almak daha iyi olacaktır, hazır

daha tam detaylara girmemişken. Qt ile birlikte gelen Assistant, Qt ile ilgili bütün her şeyi bulabileceğiniz tam bir yardım dosyaları topluluğudur. Kendi içinde arama ve sınıfların detaylı anlatımları ile ilk başvuru kaynağınızdır.

Ben, assistant'ı daha çok sınıfların fonksiyonlarına, hazır olan sinyal ve slotlarına bakmak, nasıl kullanıldıklarını öğrenmek için kullandım. Bunun dışında değişik Qt modülleri hakkında bilgi, örnek kodlar ve "neden böyle?" sorularının cevapları bu 20-30 MB'lık belgeler bütününde gizli. Örneğin, çok sık kullanılan QPushButton hakkında öğrenmek istedikleriniz varsa, soldaki yan çubuğun üzerindeki sekmelerden Index'e gelin ve QPushButton yazın. Listede sadece QPushButton kalacak, tıkladığınızda QPushButton ile ilgili sayfa karşınıza gelecek. Hemen başında sınıfın tek satırlık bir özeti yer alıp, More... diye bir bağlantı ile sizi detaylara davet ediyor olacak. Ondan önce de bütün fonksiyonlar, sinyaller ve slotlar detaylı hallerine bağlantı şeklinde listelenir. More... ifadesine tıkladığınızda sınıf hakkında detaylı bilgiler, varsa uyarılar ve nasıl kullanmanız gerektiği konusunda basit, açıklayıcı kodlar içermektedir.

Sadece sınıfların tanımları yok elbette. Linguist, Designer ve Assistant'ın kendisi hakkında bölümler, qmake hakkında bir bölüm, bu belge benzeri kısa ve öz dersler, temel özelliklerin detaylı anlatımları, SQL, Network gibi modüller vs. hepsi burada. Buradan alacağınızı umduğum başlangıçtan sonra en çok didikleyeceğiniz belgeler buradadır :) Bu yüzden, bazı sorunları internette vs. aramadan önce burada ilgili sınıfa ya da özelliğe bakarak çözüm getirme olasılığınızın yüksek olduğunu unutmayın.

3. Pencereler

O kadar konuştuk, artık ilk ana parçacık olan pencerelere değinebiliriz. Pencereler çeşit çeşittir. Gülmeyin, öyle. Mesela, yine tarayıcı örneğine devam edersek, burayı okuduğunuz yer yüksek ihtimalle bir üst seviye, TOPLEVEL penceredir. Qt'de buna Ana Pencere, MainWindow diyoruz. Bu ana pencere dışında kullanıcıdan bilgi alıp kapanan, hayatları kısa ama kısa hayatlarında çok iş yapan pencereler de vardır; Diyalog pencereleri. Mesela internet tarayıcınızın dosya menüsünden "Aç" derseniz karşınıza bir diyalog penceresi çıkar. O diyalog penceresi ile işiniz bitene kadar alttaki bölümü elleyemezsiniz. İşte bu diyalog pencerelerine Modal Window, daha doğrusu Modal Dialog denir. O penceredeki "Tamam" ya da "İptal" gibi bir yeri tıklamadan ana pencereye dönemezsiniz.

Diğer bir pencere türü, yine Dialog diyebileceğimiz ama ana pencereyi bloklamayan Modeless ya da Non-Modal denen diyaloglardır.

Qt'de bu iki pencere türü QMainWindow ve QDialog olarak durmaktadır. QDialog'un Modal ya da Modeless olmasını biz kendimiz ayarlayacağız (çok basit, merak edecek bir durum yok).

Genelde bu iki sınıfı doğrudan kullanmak yerine kendimiz bu sınıflardan yeni bir sınıf türeterek işimizi görürüz. Bunun nedeni QMainWindow'un aslında içinde hiçbir şey olmamasıdır. İçine bir şeyler eklemek, kendimize göre özelleştirmek için onu temel alan bir sınıf üretip, adına ben genelde AnaPencere diyorum, kafamıza göre düzenleme yaparız. Diyalog pencerelerini de aynı şekilde farklı isimlerle türetip programda kullanırız. Bir programda birden çok Diyalog penceresi olacağını düşünürseniz hepsinin de QDialog olmasını bekleyemezsiniz, hepsi QDialogdan türemiş, farklı isimleri olan yeni sınıflardır.

Eğer Java'nın Swing'ini kullanarak kod yazdıysanız, orada da JFrame'den benzer şekilde (artık ilk kim kime baktı orasını bilemeyeceğim) türetildiğini görürsünüz.

İyi haber: bu sınıfların görsel tasarımı için tek satır kod yazmayacağız. Aslında yazsak iyi olur, ama yazmayacağız. En azından siz yazmama lüksüne sahipsiniz. Onun yerine ben size bu pencerelerin nasıl olduğunu anlatacağım, örnek kodlarla ben yazacağım, eğer çok öğrenmek isterseniz bir sınıfı kendiniz türetip içini elle kendiniz doldurabilir, layout ekleyerek tasarımını "elle" kodlayabilirsiniz. Qt 3 ile ben öyle yapıyordum, çok zor değil, ama aceleniz varsa, Designer denen araçla bir iki dakika içinde bu işi halledebilirsiniz.

3.1. İlk Penceremiz

Penceremizi Designer kullanmadan QMainWindow denen sınıftan türetelim. Bundan bir sonraki bölümde içine parçacıklar eklerken yerleşimlerden (layout) bahsedeceğiz.

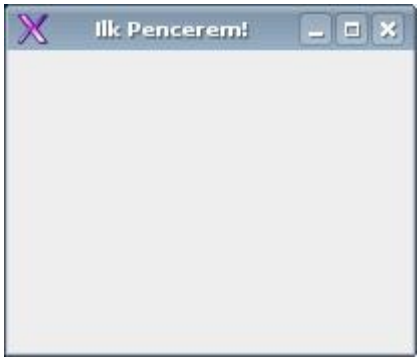
Yeni bir dizin içine yeni sınıfımız olan ilkPencere'yi ilkPencere.h ya da istediğiniz isimle kaydedin... Kodlar hemen aşağıda...

Example 3.2. İlk Penceremiz

```
// bu main.cpp
#include <QApplication>
#include "ilkPencere.h"

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

        p.show();
        return app.exec();
}

// bu da yukarıda include ettiğimiz ilkPencere.h
#include <QMainWindow>

class ilkPencere:public QMainWindow
{
        Q_OBJECT
        public:
        ilkPencere():QMainWindow()
        {
                setWindowTitle("Ilk Pencerem!");
        }
};
```

Bu kodları yazdıktan sonra, qmake -project, qmake ve make komutlarını verdiğimizde, dizin adıyla aynı isimde bir çalıştırılabilir dosyamız oluşacak. Çalıştırırsanız:

Figure 3.1. İlk Pencerenin görüntüsü



Vay! Çok kolay oldu değil mi? Aslında bu kadar okudunuz, elinizdeki tek şey başlığı "Ilk Pencerem!" olan bu boş gri dikdörtgen :)

Ama burada dikkat etmeniz gereken birçok nokta var. İlki; ilk örnekteki `QLabel` yerine bu sefer bir ana pencere kullandık! Bu pencerenin içini artık istediğimiz gibi doldurabiliriz! Bir diğeri, belki uzun süre pek değişmeyecek bir `main.cpp` elde ettiniz, ki genelde ana pencere ile başlayacağınız için onun pek değişmesine gerek yok. Diğeri ilk defa bir Qt sınıfından özel bir sınıf türettiniz ve kurucusunu değiştirdiniz! Gayet iyi gidiyoruz. Tek ihtiyacımız olan bu doyumsuz insanlara biraz daha parçacık!

3.2. Daha çok parçacık ve Yerleşim

Qt'nin zengin parçacık listesinden bir iki tane parçacık daha seçip bu pencerenin içine atalım! İlk olarak, daha önce tanıştığımız `QLabel`, altına metin girilebilecek bir `QLineEdit` ve onun yanına her arayüzün vazgeçilmez öğesi düğmeyi, `QPushButton` koyalım. Az önceki ilkPencere.h dosyamız üzerinden gidebiliriz:

Example 3.3. Parçacıklar geldi

```
#include <QMainWindow>
#include <QLabel>
#include <QPushButton>
#include <QLineEdit>
#include <QLayout>

class ilkPencere:public QMainWindow
{
        Q_OBJECT
        public:
        QLabel *etiket;
        QPushButton *dugme;
        QLineEdit *satir;

        // kurucu
        ilkPencere():QMainWindow()
        {
                setWindowTitle("Ilk Pencerem!");

                QWidget *merkez = new QWidget(this);
                setCentralWidget(merkez);

                etiket = new QLabel("Tekrar merhaba", merkez);
                dugme = new QPushButton("Click click", merkez);
                satir = new QLineEdit(merkez);


                QVBoxLayout *ana = new QVBoxLayout(merkez);
                ana->addWidget(etiket);
                QHBoxLayout *yanYana = new QHBoxLayout();
```

```
                yanYana->addWidget(satir);
                yanYana->addWidget(dugme);
                ana->addLayout(yanYana);

                resize(200,50);
                show();


        }
};
```

Bir iki parça eklemek için baya bir kod yazdık değil mi? İşte burada anlatmamız gerekenler var. Ama önce siz bir make komutunu verip bu yeni sınıfı bir derleyin, arkasından ben de size ekran görüntüsünü göstereyim:

Figure 3.2. İlk pencerenin bol parçacıklı hali

Az önceki kocaman gri boşluktan daha iyi değil mi? Burada amacım, Designer'a gelmeden elle yazılmış bir arayüz kodunu görmenizi istemem. Yani az sonra Designer ile iki dakikada tık tık yerleştirmeden önce ana pencerenin yerleşimi, parçacıkların yerleşimlerini şöyle ham halleri ile bir görelim.

Sınıf içinde yeni bir parçacık ya da Qt'nin diğer sınıflarından birini kullandıysanız `#include` ile onu eklemeniz gerekmektedir. Henüz iki önemli araç olan Assistant ve Designer'dan bahsetmedim, ama Assistant altında da bütün bu sınıfların kurucuları hakkında bilgi alabilir, doğru parametreleri verebilirsiniz. Şimdi yerleşimlere bir bakalım...

Qt belgeleri der ki, QMainWindow denen ana pencerenizin kendine özel bir yerleşimi vardır. O yüzden bir parçacık (Widget) yaratıp, onunla yapın yerleşimlerinizi. Bunun için de bize bir setCentralWidget gibi bir fonksiyon verir. Şimdi, başta yarattığımız merkez denen parçacık, daha sonra bu fonksiyon yardımı ile merkez parçacık oluyor ve bütün parçacıklarımızı ve yerleşimlerimizi onun üzerinden yapıyoruz. Hiçbir parametresi olmayan bu QWidget'ı yaratmamız sorun değil, çünkü parçacığın illa temel parçacıklar olmasına gerek yok. Yine tarayıcınıza bakarsanız, adres çubuğu aslında satır girilen bir alan, yanında bir düğme ve aşağı doğru büyüyen bir listeden oluşmaktadır. Yani temel parçacıkları kullanıp karmaşık bir parçacık elde edebilirsiniz :) Yukarıda da bunun basit bir örneğini gördünüz :)

Merkez parçacığımızı setCentralWidget ile yerine yerleştirdikten sonra kafamızdaki tasarımın en dış haline bakıyoruz. Yukarıdaki ekran görüntüsüne de bakarsanız aslında en dıştaki yerleşim alt altadır. "Tekrar Merhaba" yazan etiket bir üst satırda, alttaki satırdaki iki parçacığın uzunluğunda (her ne kadar renk farkı olmadığı için fark edilmese de), alt satırdakiler de yan yana dizilmiş durumdalar. Elinizde tek parça varsa bir yerleşim belirlemenize gerek yok, ama iki ya da daha fazla varsa onları bir arada tutmak için bir yerleşim kullanmanız iyi olur. Designer ile tasarım yaparken en içteki yerleşimlerden başlayacakken, elle kodlarken, yukarıdan aşağıya yazdığımız için en dıştan başlamak zorunda kalıyoruz.

Şimdi en dışta alt alta olacağını biliyoruz, o yüzden merkez parçacığını QVBoxLayout ile yerleştiriyoruz. Bu sınıf Vertical yani dikine parçacıkları yerleştiriyor. Dikine

dediğinizde tavandaki bir vidaya arka arkaya somunları sıkıştırdığınızı düşünebilirsiniz. İlk somun, tavana en yakın, yani en yukarıda olacaktır. Bu yüzden ilk addWidget ile etiketi en üste koyduk. Alttakine geçmeden önce o ikisini QHBoxLayout, yani yatay yerleşim ile bir arada tutmamız gerekecek. Bu yüzden bir QHBoxLayout yapıp önce satırı, sonra da düğmeyi oraya geçirdik. Burada da ilk yerleştirdiğiniz solda kalıyor.

Daha sonra bu yerleşimi addWidget ile değil, addLayout ile merkeze ekliyoruz. İşlem tamam... En sondaki `resize()`, tahmin ettiğiniz gibi pencereyi yeniden boyutlandırıyor, `show()` ise bütün parçacıkları gösteriyor.

Bu noktadan sonra SIGNAL ve SLOT'lara değineceğim. Qt ile iş yapmanın en önemli parçaları olan bu ikisi üzerinde basit örnekleri verdikten sonra daha karmaşık uygulamalara geçeceğiz.

4. SIGNAL ve SLOT

Değinmemiz gereken diğer önemli konu, Qt'nin "olay güdümlü" programlamasının merkezini oluşturuyor: **SIGNAL** ve **SLOT**. Türkçe'ye sinyal / yuva şeklinde çevirebileceğimiz bu mekanizma sayesinde olayları kontrol edebiliyoruz. Kendi sinyal ve yuvalarımızı da tanımlayabiliyoruz.

Bir önceki pencereye yeni parçacıklar koymuştuk. Bunlar bir girdi satırı ve düğme. Bu iki parçacığın hazır birçok sinyalleri vardır. Düğme için bunlardan en çok kullanılanı `clicked`, yani `tıklandı` olayıdır. Bu tıklanma olayını istediğimiz bir yuvaya bağlayabiliriz (`connect`). Bu yuva tanımlı olabilir, ya da biz kendimiz tanımlayabiliriz. Ana pencere parçacığının bir yuvası olan `kapa`, yani `close` başlangıç için iyi olacaktır. Bağlamak için tek yapmamız gereken connect fonksiyonunu kullanarak bu iki sinyali bağlamak. Eğer yukarıdaki kodun içine resize satırından önce

```
connect(dugme, SIGNAL(clicked()), this, SLOT(close()));
```

ekleyip derlerseniz, düğmeye tıkladığınızda pencere kapanacaktır. `connect` fonksiyonu kendini gayet açıklar niteliktedir. İlk parametresi `dugme`, olayın oluştuğu nesneyi söyler. İkincisi mutlaka `SIGNAL()` içine yazılması gereken ilk nesnenin bir sinyalidir. Sonra olayın etkileyeceği nesne yazılır. Burada `this` bizim penceremizi gösteriyor. Son olarakta hedef nesnenin yuvası `SLOT()` içinde çağırılır. Ama biz bu hazır SLOT ile yetinmeyip kendi SLOT'umuzu yazalım.

Bunun için kurucu fonksiyondan sonra public slots ile slotları normal bir fonksiyon gibi tanımlayabiliriz. İsterseniz metin satırında ne yazıyorsa onu ekrana bastıran bir SLOT yapalım bunu. Sanırsam kod her zaman olduğu gibi gene işe yarayacaktır...

Example 3.4. Örnek bir SLOT

```
#include <QMainWindow>
#include <QMessageBox>
#include <QLabel>
#include <QPushButton>
#include <QLineEdit>
#include <QLayout>

class ilkPencere:public QMainWindow
{
        Q_OBJECT
```

```
        public:
        QLabel *etiket;
        QPushButton *dugme;
        QLineEdit *satir;

        // kurucu
        ilkPencere():QMainWindow()
        {
                setWindowTitle("Ilk Pencerem!");

                QWidget *merkez = new QWidget(this);
                setCentralWidget(merkez);

                etiket = new QLabel("Tekrar merhaba", merkez);
                dugme = new QPushButton("Click click", merkez);
                satir = new QLineEdit(merkez);


                QVBoxLayout *ana = new QVBoxLayout(merkez);
                ana->addWidget(etiket);
                QHBoxLayout *yanYana = new QHBoxLayout();
                yanYana->addWidget(satir);
                yanYana->addWidget(dugme);
                ana->addLayout(yanYana);

                // connect satırları
                connect(dugme, SIGNAL(clicked()), this, SLOT(slotDugme()));

                resize(200,50);
                show();
        }
        public slots:
        void slotDugme()
        {
                // bu slot dugmeye basinca calisacak
                QMessageBox::information(this, "Tebrikler!", satir->text());
        }
};
```

Buradaki kod, yukarıdaki örneğin biraz güncellenmiş hali. QMessageBox pop-up pencere dediğimiz küçük mesaj pencereleri göstermemizi sağlıyor, onu kullanabilmek için en başa include ile eklemeyi unutmadık. Sonra public slots altında normal bir fonksiyon tanımlar gibi slotDugme diye bir fonksiyon tanımladık. connect bu ikisini bağladı ve artık güzel güzel çalışıyorlar.

Programı çalıştırdığınızda içgüdüsel olarak satıra yazdıktan sonra düğmeye tıklamak yerine ENTER tuşuna basmış olabilirsiniz. Fark edeceğiniz gibi o zaman istediğimiz "olay" olmamış olacağından, slotDugme çalışmaz. Ama isterseniz, satir nesnesinin bir SIGNAL'ini kullanarak enter tuşuna basıldığında da çalışmasını sağlayabilirsiniz. Bunun için tek yapmanız gereken QLineEdit sınıfının uygun bir sinyalini aramak, ki o da returnPressed, ve bu sinyali de aynı slot'a bağlamak:

```
connect(satir, SIGNAL(returnPressed()), this, SLOT(slotDugme()));
```

Bu satırı diğer connect satırının oraya yazdıktan sonra, tekrar derlediğinizde hem düğmeye tıkladığınızda, hem de enter tuşuna bastığınızda aynı fonksiyonun satir nesnesinin içindeki metni ekrana bir MessageBox ile yazdırdığını göreceksiniz...

Chapter 4. Designer

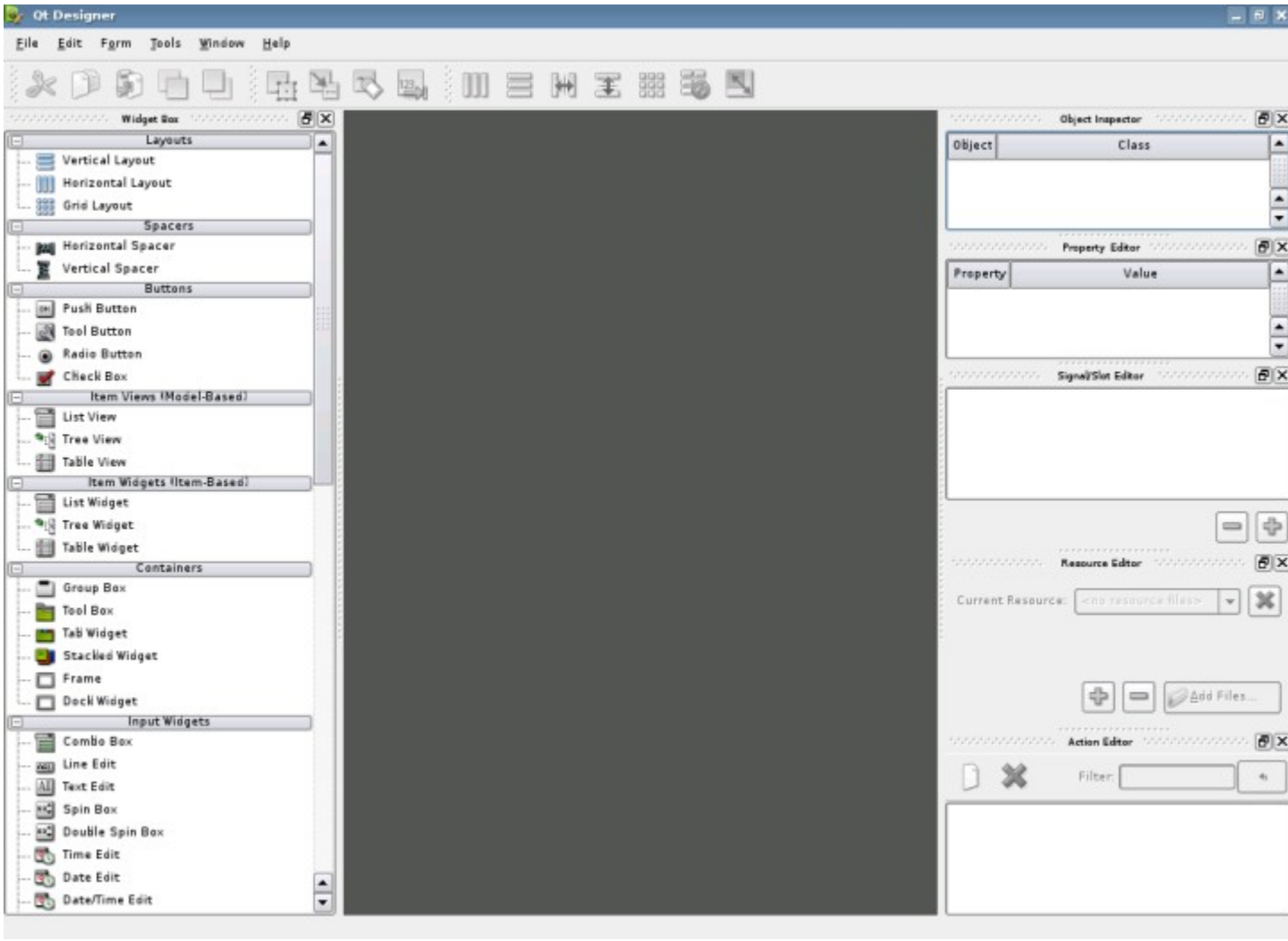
Bu bölümde bizi arayüz kodlarını elle yazmaktan kurtaracak Designer uygulamasına detaylı bir şekilde bakarken, Qt programlama ile ilgili daha çok şey öğreneceğiz.

Önceki bölümde anlattığım bütün o yerleşimleri falan burada Designer'ı kullanırken mantığı daha rahat görebilin diye anlattım :)

1. Designer

Designer ilk açıldığında birden çok pencere ile gelir, alttaki ekran görüntüsü sizinkinden biraz farklı, yerleri değişik olabilir. Designer'ı her açışınızda size yaratmak istediğiniz pencereyi sorar. Şimdilik o diyalog penceresini kapatın ve arayüze bir bakalım:

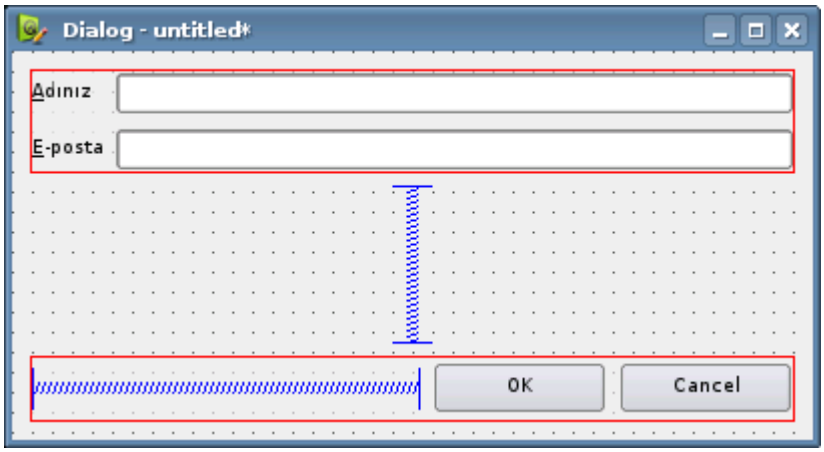
Figure 4.1. Designer Ekran Görüntüsü



Designer ile tek yaptığınız "arayüz" tasarlamak gibi gözükse de aslında bütün bir projenin ana hatlarını burada belirlemeniz mümkün. Genel olarak baktığınızda sağda ve solda gözüken araç kutularının ne işe yaradıklarını, buraya kadar okuduysanız az çok çıkarabilmeniz lazım. Solda parçacık listemiz var. Burada kullanacağınız hazır parçacıkları sürükle bırak ile bir pencere üzerine bırakabilirsiniz.

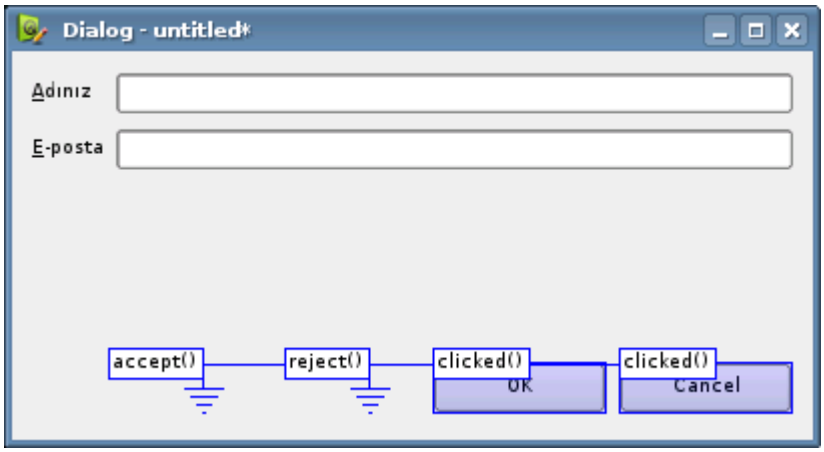
Üstteki araç çubuğunda "kes", "kopyala" gibi simgelerden sonra dört tane farklı kipi simgeleyen düğmeler var. Bunların en baştaki üzerine geldiğinizde "Edit Widgets" yazan parçacıkları düzenleme kipi. Bu kip en çok kullanacağımız kip ve burada pencere üzerindeki parçacıkların yerleşimini hazırlayacağız. Az önce elle yazdığımız yerleşim olaylarını da bu dört düğmeden sonra gelen dik ve yatay simgeleri olan yerleşim düğmeleri ile yapacağız. Onlara gelmeden üç simgemiz daha var.

Figure 4.2. Parçacık Düzenleme Kipi



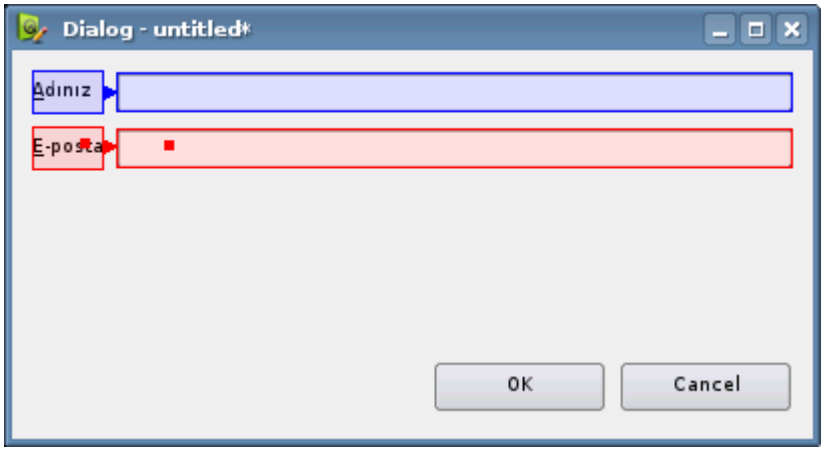
"Edit Widgets"tan sonra "Edit Signal / Slots" denen düğme yer alıyor. Bu yazılar simgelerin üzerinde biraz durunca çıkıyor (tooltip). Ekranda göremiyorsanız sakin olun :) Bu kipteyken pencere üzerindeki parçacıklar sinyal ve slotlarını sürükle bırak şeklinde düzenleyebiliriz. Önce sinyali üretecek parçacığı seçin, sonra slotun olacağı parçacığı... Sonra ikisi arasında hangi sinyal ve slotları bağlamak istediğinizi söyleyen bir pencere çıkacak. Aşağıdaki diyalog penceresinde OK düğmesi clicked() sinyalinde diyalog penceresinin accept() slotuna bağlı, diğeri ise reject()'e... Bu olayı "Signal Slot Editor" denen kutu ile de yapabilirsiniz.

Figure 4.3. Sinyal / Slot Düzenleme



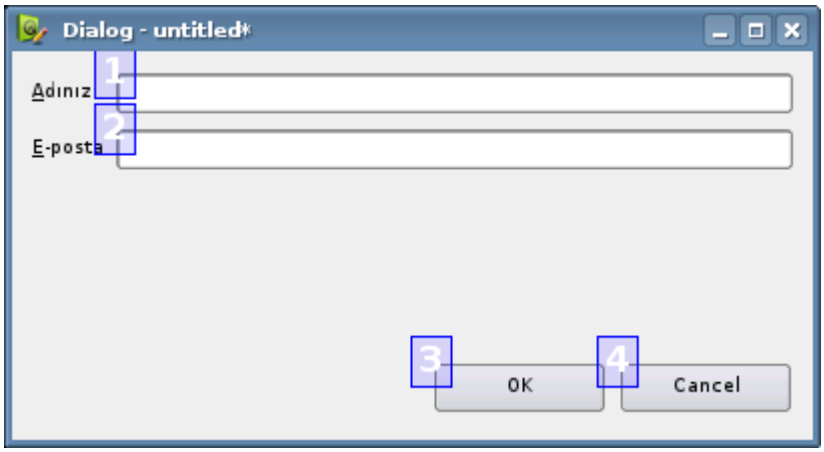
Bu düğmeden sonra gelen "Edit Buddies" ile, parçacıkları ilişkilendiriyorsunuz. Bu ilişkilendirme şu işe yarıyor: Diyelim ki bir QLineEdit kullandınız, üstteki ekran görüntülerindeki gibi. Bu QLineEdit'e bir kısayol olduğunu ekran görüntüsündeki gibi "Adınız" yazarken A harfinin altının çizili olduğundan anlarsınız. Ama "Adınız" yazan parçacık bir QLabel, diğeri ise bir QLineEdit'tir. İkisi arasında bir ilişki kurmak için bunları "Buddy" yani arkadaş yapmalısınız. Alttaki ekran görüntüsünde görebilirsiniz. Bunun için yine önce QLabel'ı sonra da QLineEdit'i seçmeniz bu ikisi arasında bir ok ile gösterilecek...

Figure 4.4. Buddy Düzenleme



Son olarak ise "Sekme Sırası"nı belirleyen "Tab Order" kipi var. Bu kip ile sekme tuşu ile parçacıkları gezerkenki sırayı belirliyorsunuz. Burada tek yapmanız gereken çıkan 1, 2, 3 ve 4 gibi numaraları sekme sırasını oluşturmak istediğiniz sırada tıklamak... Bu pencere ilk açıldığında OK ve Cancel düğmeleri 1 ve 2, diğerleri 3 ve 4 idi. Ben 3'e tıklayınca orası 1 oldu, sonra 4'e tıkladım orası 2 oldu vs. Bu şekilde sekme sırasını basitçe düzenleyebilirsiniz...

Figure 4.5. Sekme Sırası Düzenleme Kipi



İlk parçacık düzenleme kipindeyken parçacıkların etrafındaki kırmızı çizgiler yerleşim (layout) çizgileridir. Gördüğüz dikey yay ise bu diyalog penceresi büyüdükçe satırları yukarıda, düğmeleri aşağıda tutarak yayı koyduğumuz yerin genişlemesini sağlar. Bu yayın bir de yatay olanı vardır, onu da Ok ve Cancel düğmelerinin orada görüyorsunuz...

Bu kiplerle arayüzünüzü tasarlarken sağ tarafta yer alan kutulardan "Property Editor" yani Özellik Düzenleyici ile nesnelerin her türlü özelliğini değiştirebilirsiniz. Bütün bunlar sayesinde arayüz ile ilgili kod yazmanız gerekmez.

Daha önce de dediğim gibi bir projenin genel hatlarını burada belirleyebilirsiniz. Projede asıl iş yapan arayüz değil fonksiyonlardır ve bu fonksiyonları çalıştıracak olan olayları buradan belirlemek mümkün. Hatırlarsanız bir eylemi hem araç çubuğu üzerinden, hem menüden, hem de kısayollar ile çağırabildiğimizi söylemiştim. İşte bunu yapmak için Designer'da Action Editor denen kutuyu (ekran görüntüsünde sağ en altta) kullanacağız. Burada tanımladığımız QAction'ları menü ve araç çubuklarına sürükle bırak ile yerleştirebilir, Özellik Düzenleyici ile kısayollarını, ipuçlarını vs. belirleyebiliriz. Bunları bir sonraki bölümdeki tam bir Qt uygulaması yazarken daha detaylı konuşacağız.

2. Menü ve Araç Çubuğu Düzenleme

Menü ve Araç çubuğu için bir ana pencereye ihtiyacınız var. Yeni bir ana pencere yaptığınızda boş bir menü çubuğu gelecektir. Tek yapmanız gereken "Type Here" yazan yere çift tıklayıp menü adınız yazmak. Daha sonra Action Editor bölümündeki eylemleri buraya sürükleyip bırakabilirsiniz ya da açılan menüye yazdıkça eylemlerin Action Editor bölümünde gözükmesini izleyebilirsiniz.

Menü ile işleriniz tamamlandıktan sonra ana pencerenin boş bir yerine tıklatıp "Add Toolbar" diyerek bir araç çubuğu ekleyebilirsiniz. Action Editor'deki eylemleri buraya sürükle bırak ile taşıyabilirsiniz.

Ama o da ne? Araç çubuğunda simgeler yok diye sadece yazı mı çıkıyor? Bir diğer kutu olan "Resource Editor" bölümüne göz atmanın vakti gelmiştir.

3. Simgeler

Simgeleri eklemenin iki yolu var. Ya bir yol belirterek, dizin yapısı içinde, ya da Resource yani kaynak kullanarak. Resource Editor bölümü bu yüzden çok önemli. Yeni bir ana pencere yarattığınızda bu Kaynak Düzenleyicide "<no resource files>" yazmaktadır. Oraya tıklayıp "New" ile yeni bir kaynak yaratın. Adına simgeler.qrc diyebilirsiniz. Ondan sonra altta duran +'ya basıp yeni bir yol yaratıp, "Add Files" ile simgelerinizi ekleyin! Artık uygulama içinde simgeleri kullanabilirsiniz! Tek yapmanız gereken Property Editor'de simge seçerken "Specify image file" yerine "Specify resource" seçip açık olan kaynaklardan kullanacağınızı söyleyin. Bu şekilde simgeler çalıştırılabilir dosya içine gömülür, kaybetme derdiniz de olmaz :)

4. Tam bir proje?

Qt'nin ve Designer'ın bütün olayı bu değil tabii ki. Ama giriş için bu yeterli. Şimdi tam bir uygulama yazıp bütün bunları ve fazlasını kullanalım!

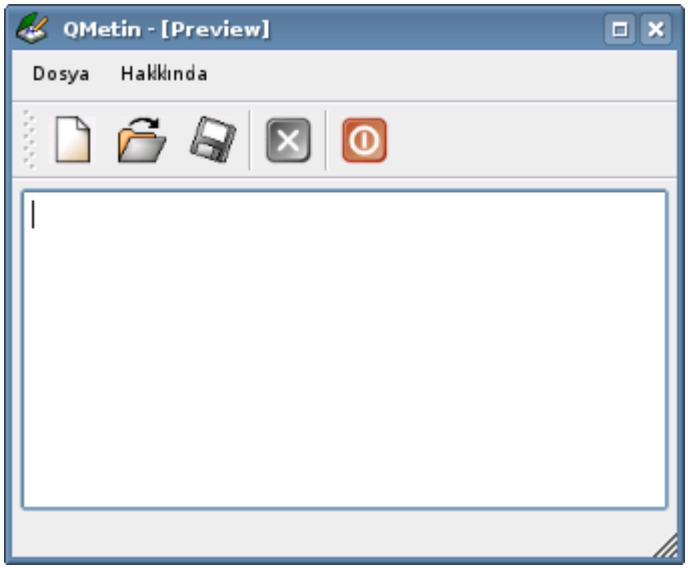
Chapter 5. QMetin

Ne yazsam ne yazsam diye düşündüm, ve en güzelinin bir metin düzenleyici olduğuna karar verdim. Adım adım bir metin düzenleyici karşınızda! Programın kaynak kodlarını buradan indirebilirsiniz.

1. Arayüz Tasarımı

Arayüz tasarımı, basit bir metin düzenleyiciden beklediğiniz gibi, bir metin alanından oluşuyor. Elbette üstte menü ve araç çubuklarını yarattım. Onlardan önce Action Editor ile eylemleri belirledim: Yeni, Aç, Kapat, Kaydet, Çıkış ve Hakkında. Bütün bunlar için KDE'nin simgelerinden bir iki tane kopyalayıp Resource Editor ile onları ekledim. Sonuçta karşıma işte bu çıktı:

Figure 5.1. QMetin Önizleme



Bütün bunları yapmak üç dört dakikamı aldı. İnce detaylar için biraz uğraşmanız gerekebilir. Mesela bütün eylemler için bir kısayol belirleyin (Ctrl+N yeni için mesela) arkasından "Tooltip" denen ipuçlarını ve "Status Bar" mesajlarını girebilirsiniz... Ben hepsine bir iki dakika daha ayırdım...

Ana pencere için seçtiğim simgenin ne olduğunu hatırlayan var mı? (Bu simge KDE 1 masaüstünün Kedit simgesiydi...)

Genelde bu adımdan başlamamak gerekir, ama basit bir program olduğunu düşünürsek arayüz ile birlikte tasarlamak bize bir iki düşünce sunmadı değil. Eylemleri baştan belirleyebilmek önemli...

2. Programın Tasarımı

Bu ilk bilmeniz gereken şeydir. Hemen öyle kodlamaya girişirseniz, karşınıza bir sürü sorun çıkar, planlı programlı gitmek lazım. O yüzden programın nasıl işleyeceğini bir düşünelim...

Bu program ilk başladığında boş bir belge olacak. Henüz kaydedilmemiş, bir dosya adı olmayan bir belge. Bu iki kelime bana hemen bir "kaydedilme" ve "dosya adı" değişkenlerine ihtiyacımız olduğunu söylüyor. Bunları "Çıkış" sırasında henüz kaydedilmemiş bir belge için de kullanabiliriz...

Dosyanın kaydedilip kaydedilmeyeceğini birden çok kontrol edeceğimizi de biliyoruz... O yüzden bu tip olayları fonksiyonlara bağlamakta fayda var.

Bu çok basit bir düzenleyici olduğu için şimdilik bu kadar tasarımın yeteceğini düşünerek, girişelim:

3. Kodlamaya Girişiyoruz

Elbette ilk olarak bir main.cpp'ye ihtiyacımız var. Sonra ana penceremizin header dosyasına. Bunlar basitçe yukarıdaki örnekler gibi hazırlanabilir:

```
// main.cpp
#include <QApplication>
#include "anapencere.h"
```

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

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

Ana penceremizin header dosyası (daha ilk hali elbette)

```
#include <QMainWindow>
#include "ui_anapencere.h"

class anapencere:public QMainWindow, Ui::MainWindow
{
        Q_OBJECT
        public:
        anapencere():QMainWindow()
        {
                setupUi(this);
        }
};
```

Buradan sonra artık yavaş yavaş eylemleri kodlamaya başlayabiliriz. Bunun için önce hepsini connect ile bağlamak, arkasından da anapencere.cpp dosyasında da kodlarını yazmamız gerekecek. QAction sınıfının en önemli sinyali activated() sinyalidir, bunu kullanacağız daha çok. Güncel bir anapencere.h işte şu şekilde:

```
#include <QMainWindow>
#include "ui_anapencere.h"

class anapencere:public QMainWindow, Ui::MainWindow
{
        Q_OBJECT
        public:
        anapencere():QMainWindow()
        {
                setupUi(this);

                connect(actionYeni, SIGNAL(activated()), this, SLOT(slotYeni()));
                connect(actionAc, SIGNAL(activated()), this, SLOT(slotAc()));
                connect(actionKaydet, SIGNAL(activated()), this,
SLOT(slotKaydet()));
                connect(actionKapat, SIGNAL(activated()), this, SLOT(slotKapat()));
                connect(actionCikis, SIGNAL(activated()), this, SLOT(slotCikis()));
                connect(actionAbout, SIGNAL(activated()), this, SLOT(slotAbout()));
        }
        public slots:
        void slotYeni();
        void slotAc();
        void slotKaydet();
        void slotKapat();
        void slotCikis();
        void slotAbout();
};
```

Buradan sonra, bu slotların yazılmalarını ayrı bir dosya olan anapencere.cpp'de devam edeceğiz...

Buraya aynı zamanda tekrar edecek fonksiyonları da yazmamız gerekecek.

Programı yazarken bol bol yapacağımız işlemler arasında Makefile olayını QMake'e bırakmak var. Yeni pencere vs. eklemediğiniz ve aynı dosyalar üzerinde çalıştığınız

sürece son adımı uygulamanız yeterli olacaktır.

4. QMake

QMake'e biraz daha detaylı bakmamızda fayda var. İlk dosyalarımız oluştuğunda dizinde main.cpp, anapencere.h, anapencere.cpp, anapencere.ui ve simgeler.qrc ile bir simge dizini var... Bu anda vereceğiniz

```
qmake-qt4 -project
qmake-qt4
make
```

komutları ile ilk derleme işlemini yaparsınız. Burada make dediğinizde anapencere.ui dosyanızdan bir ui_anapencere.h oluşur. Sonra derlenme satırlarını görürsünüz. Sonlara doğru simgeler.qrc'den qrc_simgeler.cpp oluşur, o da derlenir... Tabii bu arada bir QObject olan anapencere sınıfının da MOC, yani Meta Object Compiler'dan geçmiş halini moc_anapencere.cpp dosyasında görebilirsiniz...

Olur da yeni bir diyalog penceresi ya da başka bir dosyada bir sınıf eklerseniz,

```
make clean
```

komutu ile öncelikle bu moc_*, ui_* ve *.o gibi dosyaların kaldırılmasını sağlarsınız. Arkasından *.pro dosyasını ve Makefile'ı silip üstteki üç adımı tekrar etmeniz gerekir.

Kodları yazarken, bu üç adımı devamlı tekrarlamanıza gerek yok. Büyük ihtimalle bol bol make komutunu verip adım adım ilerleyeceksiniz...

5. Kodların İncelenmesi

Kodları burada anlatmaktansa yorum satırları olarak incelemekte fayda var:

```
// anapencere.h
#include <QMainWindow>
#include "ui_anapencere.h"

class anapencere:public QMainWindow, Ui::MainWindow
{
        Q_OBJECT
        public:
        bool kaydedildi;
        QString dosyaAdi;
        anapencere():QMainWindow()
        {
                setupUi(this);

                kaydedildi = true;
                dosyaAdi = QString::null;

                setWindowTitle("Yeni Belge - QMetin");

                connect(actionYeni, SIGNAL(activated()), this, SLOT(slotYeni()));
                connect(actionAc, SIGNAL(activated()), this, SLOT(slotAc()));
                connect(actionKaydet, SIGNAL(activated()), this,
SLOT(slotKaydet()));
                connect(actionKapat, SIGNAL(activated()), this, SLOT(slotKapat()));
                connect(actionCikis, SIGNAL(activated()), this, SLOT(slotCikis()));
                connect(actionAbout, SIGNAL(activated()), this, SLOT(slotAbout()));
                connect(textEdit, SIGNAL(textChanged()), this,
```

```
SLOT(slotTextChanged()));
        }

        /* Kaydedilip edilmediğini birden fazla yerde kontrol
         * etmemiz gerekiyor. O yüzden bunu ayrı bir fonksiyonda
         * toplamak iyi olur... Bu fonksiyon kaydedilip edilmediğini
         * kaydedildi denilen bool değişkeni ile bakıyor. Kaydedilmemişse
         * slotKaydet ile kaydetmesini sağlıyor
         */
        void kayitKontrol();

        protected:
        /* Dosya menüsünden "Kapat"ı seçmek yerine Alt+F4
         * ya da sağ üstteki X ile kapatmayı seçerse
         * buradaki closeEvent ile onu yakalıyoruz.
         * Normal koşullarda bu bir QWidget virtual fonksiyonu
         * onu tekrar yazarak kapanmadan önce slotCikis
         * slotumuza gönderiyoruz. Böylece kapanmadan bir
         * kaydetme şansı da vermiş oluyoruz.
         * slotCikis()'in işi bittikten sonra QMainWindow'un
         * kendi closeEvent'i çalışabilir :)
         */
        void closeEvent(QCloseEvent *event)
        {
                slotCikis();
                QMainWindow::closeEvent(event);
        }

        public slots:
        void slotYeni();
        void slotAc();
        void slotKaydet();
        void slotKapat();
        void slotCikis();
        void slotAbout();
        void slotTextChanged();
};


// anapencere.cpp
#include <QMessageBox>
#include <QFileDialog>
#include <QDir>
#include <QFile>
#include <QTextStream>
#include "anapencere.h"

void anapencere::slotTextChanged()
{
        /* Bu sayede bir harf bile değiştirilse metin kutusunun
         * içindeki değişimlerden haberimiz oluyor!
         * Metin düzenleyicilerinde değiştirildikten sonra
         * başlıklarında bir yıldız (*) ifadesi ile
         * değişikliği gösterme gibi bir olay var. Onu da burada
         * sağlıyoruz
         */
        if (kaydedildi)
        {
                kaydedildi = false;
                if (dosyaAdi != QString::null)
                        setWindowTitle(dosyaAdi + "* - QMetin");
        }
}

void anapencere::kayitKontrol()
{
        /* anapencere.h dosyasında bu fonksiyonun yorumu var :) */
        if (kaydedildi == false)
        {
                if (QMessageBox::question (
                                this,
```

```
                                QString::fromUtf8("Dosya kaydedilmedi!"),
                                QString::fromUtf8("Dosya kaydedilmedi!\nKaydetmek
ister misiniz?"),
                                QMessageBox::Yes | QMessageBox::Default,
                                QMessageBox::No,
                                QMessageBox::NoButton
                                ) == QMessageBox::Yes
                        )
                {
                        slotKaydet();
                        kaydedildi = true;
                        /* Buradaki QMessageBox::question sayesinde kullanıcıya
                         * basit bir soru soruyoruz. Hangi düğmeleri kullanacağımızı
                         * belirtebiliyoruz. Son düğmeyi kullanmak istemezsek
NoButton
                         * kullanıyoruz. Öntanımlı olmasını istediğimizi OR'luyoruz,
                         * şu bildiğiniz | işareti ile :) Bu kadar basit.
                         * Buradan dönen değer QMessageBox::Yes ise diyerekten de
seçilen
                         * düğmeyi kolayca görebiliyoruz...
                         */
                }
        }
}


void anapencere::slotYeni()
{
        /* Adından da anlaşılacağı gibi yeni bir dosya açıyor.
         * Tabii bizim programımız için bu değişkenlerin
         * sıfırlanması anlamına geliyor. Dosyayı devamlı
         * açık tutmadığımız için metin alanını temizleyip,
         * dosyaAdi değişkenini tekrar null yapıyoruz.
         * textEdit->clear() çağırıldığında metin değiştiği
         * için kaydedildi değişkeni değişiyor, onu en son true
         * yapıyoruz
         */
        kayitKontrol();
        setWindowTitle("Yeni Belge - QMetin");
        textEdit->clear();
        dosyaAdi = QString::null;
        kaydedildi = true;
}


void anapencere::slotAc()
{
        /* Dosya açmak için QFileDialog sınıfını kullanıyoruz. Bu sınıf
         * bize bir dosya açma penceresi sunuyor. Filtre belirleyip istediğimiz
         * türlerin listelenmesini sağlayabiliyoruz. Ben burada düz metin
         * dosyaları olan txt'leri koydum, yanına da Tüm Dosyalar diyerek
         * diğer seçenekleri de belirttim.
         * Buranın bir diğer özelliği ise QDir ve QFile gibi sınıfları kullanması.
         * Bu sınıflar dosya ve dizinler hakkında bilgi verirler. QDir::homePath()
         * bize kullanıcının ev dizininin tam yolunu gönderir.
         * Burada kullanılan son sınıf QTextStream... Bir IO sınıfı olan bu
         * sınıf sayesinde kolayca dosyadan okuyabiliyoruz. slotKaydet()te de
         * bu sınıfı kullanacağız...
         */
        kayitKontrol();
        QFileDialog *fd = new QFileDialog(this, QString::fromUtf8("Dosya Aç"),
QDir::homePath());
        fd->setFileMode(QFileDialog::ExistingFile);

        QStringList filters;
        filters << QString::fromUtf8("Metin Belgeleri") + "(*.txt)"
                << QString::fromUtf8("Tüm Dosyalar") + "(*.*)";
        fd->setFilters(filters);

        if (fd->exec())
        {
```

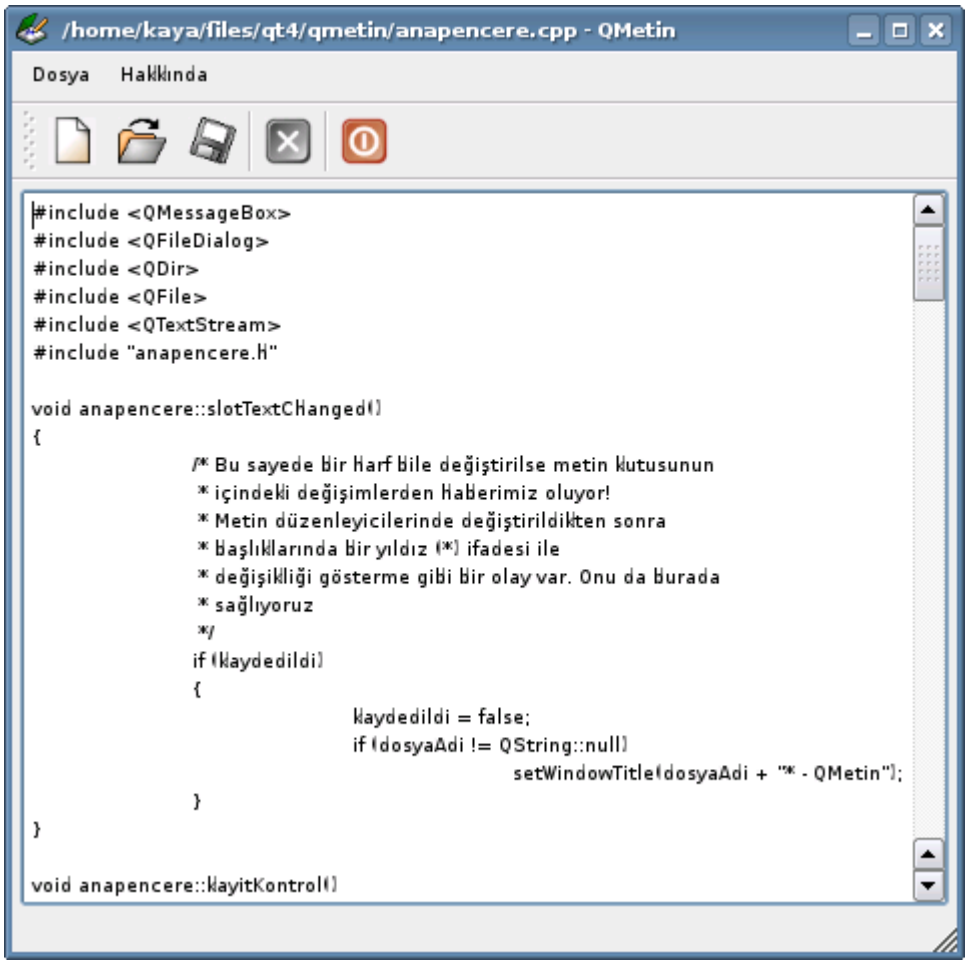
```
                dosyaAdi = fd->selectedFiles().at(0);
                // dosyayı yükle
                QFile ac(dosyaAdi);
                if (!ac.open(QIODevice::ReadOnly | QIODevice::Text))
                {
                        QMessageBox::critical(this, QString::fromUtf8("Dosya
Açılamadı!"),
                        QString::fromUtf8("Dosya okumak açılamadı, belki izniniz
yok?"));
                        dosyaAdi = QString::null;
                        return;
                }
                QTextStream in(&ac);
                textEdit->setPlainText(in.readAll());
                setWindowTitle(dosyaAdi + " - QMetin");
                kaydedildi = true;
                ac.close();
        }
}


void anapencere::slotKaydet()
{
        /* Kaydetme anı! Burada dikkat etmemiz gereken daha önce kaydedilmişse
         * yani bir dosya adı varsa onu kullanmak. Yoksa yine QFileDialog ile
         * bir dosya adı istemek. Burada QFileDialog'a AcceptMode olarak AcceptSave
         * dedik. Bu sayede diyalog penceresinde Aç yerine Kaydet çıkacak...
         * QFile ve QDir burada da yukarıdaki işlevlerini yerlerine getirirken
         * QTextStream'e sanki cin << ile yazar gibi rahatça yazdığımızı görmenizi
         * isterim...
         * textEdit denen metin düzenleme kutucuğumuz aynı zamanda zengin metin
         * ve/veya HTML desteklediği için burada ve yukarıda setPlainText ve
         * toPlainText gibi fonksiyonlarını kullandık..
         */
        if (dosyaAdi == QString::null)
        {
                QFileDialog *fd = new QFileDialog(this, QString::fromUtf8("Dosya
Kaydet"), QDir::homePath());
                fd->setFileMode(QFileDialog::AnyFile);

                QStringList filters;
                filters << QString::fromUtf8("Metin Belgeleri") + "(*.txt)"
                        << QString::fromUtf8("Tüm Dosyalar") + "(*.*)";
                fd->setFilters(filters);
                fd->setAcceptMode(QFileDialog::AcceptSave);

                if (fd->exec())
                        dosyaAdi = fd->selectedFiles().at(0);
        }
        QFile yaz(dosyaAdi);
        if (!yaz.open(QIODevice::WriteOnly | QIODevice::Text))
        {
                QMessageBox::critical(this, QString::fromUtf8("Dosya Açılamadı!"),
                QString::fromUtf8("Dosya yazmak için açılamadı, belki izniniz
yok?"));
                return;
        }
        QTextStream out(&yaz);
        out << textEdit->toPlainText();
        setWindowTitle(dosyaAdi + " - QMetin");
        kaydedildi = true;
        yaz.close();
}
void anapencere::slotKapat()
{
        /* Dosyanın kapanıp yenisinin açılması. Aslında açık dosya olmadığı
         * için bu yine bir bakıma sıfırlama işlemi sayılır
         */
        kayitKontrol();
        dosyaAdi = QString::null;
        slotYeni();
```

```
}

void anapencere::slotCikis()
{
        /* çıkmadan önce de kontrol ediyoruz */
        kayitKontrol();
        close();
}

void anapencere::slotAbout()
{
        /* QMessageBox ile basit bir Hakkında penceresi... */
        QMessageBox::about(this, QString::fromUtf8("QMetin Hakkında"),
        QString::fromUtf8("Bu düzenleyici tamamen eğitim amaçlı
hazırlanmıştır!\nKaya Oğuz, 2006\nQt Türkiye, http://qt.comu.edu.tr"));
}
```

Burada yeri gelmişken şu QString::fromUtf8() fonksiyonundan da bahsedeyim. Yazdığım dosyaların karakter kodlaması Utf8 ve bu yüzden yarattığım QString'lerin bu kodlamada olması için bunu kullanıyorum.

Bu kısa program ile süper basit bir düzenleyicimiz oldu :) Ve işte sonuç, kendi kaynak dosyalarını gösterirken:

Figure 5.2. QMetin Son



Chapter 6. Son Söz

Buraya kadar geldiğinize göre artık ilk adımlarınızı atmış bulunmaktasınız. Bundan sonra yapmak istediğiniz bir program için elinizin altında bir Assistant yeterli olacak.

Genel olarak işlemleri anladığınızı düşünüyorum.

Qt çok fazla parçacık destekliyor. Bunun yanında SQL ve Network gibi modülleri var. Bunların hepsini ihtiyaç duydukça kullanabilirsiniz. Qt kendi içinde kararlı bir isimlendirme ve düzen politikası izler. Çoğu zaman fonksiyon isimlerini doğru tahmin ettiğinizi göreceksiniz.

Buradan sonra ilk adımınız kendinize bir proje belirleyip üzerinde çalışmak olacak. Biraz kendinizi iyi hissettiniz mi ilk ciddi projeniz için Qt Türkiye ya da Pardus gibi gruplara katılıp "Var mı yapabileceğim bir şey?" diyebilirsiniz :) Özellikle Pardus için Qt bilen taze kana ihtiyaç var :)

Geçmiş olsun, yeni bir belgede görüşmek üzere! (-Daha olacak yani? -Niye olmasın?)