# QT 'YE GIRIŞ

Linux dünyasında geliştirilen uygulamaların büyük bir çoğunluğu iki farklı grafik kütüphanesi kullanılarak yazılmaktadır. Bunlardan biri GTK diğeri ise Qt kitaplıklarıdır. Qt kütüphanesi X altında uygulama geliştirmek isteyen geliştiriciler için *Trolltech* firması tarafından geliştirilen bir görsel kütüphanedir. Çoğunlukla görsel uygulama yazacak C++ geliştiricileri tarafından rağbet görmektedir. Qt kütüphanesi ilk olarak tamamen paralı bir ürün olarak piyasa sürülmüştür. Daha sonra GPL sürümü de yerini almıştır. KDE masa üstü ortamı tamamen qt ile yazılmış olması qt kütüphanesinin gücünü anlamamızı sağlamaktadır. Qt kütüphanesiyle yazılmış bir çok uygulama piyasada karşımıza çıkabilmektedir. Qt sadece bir görsel uygulama kütüphanesinden başka birçok eklentileri de bulunmaktadır. Bunları örnek vermek gerekirse Qt kitaplığı içinde Sql ve network uygulamaları geliştirmek için gerekli bir çok fonksiyon bulunmaktadır. Özellikle veritabanı uygulamaları geliştirenler için büyük kolaylık sağlamaktadır. Qt ile ilgili örnekleri derlemek ve nasıl işlediğini anlamak için C++ bilginizin olması gerekmektedir.

Qt için uygulama geliştirme aşamasında öncelikle *qtdesigner* ve *kdevelop* isimli uygulamaların kullanımı öğrenilmelidir. Bu uygulamalar tıpkı *Glade* ve *Anjuta* da olduğu gibi uygulama geliştiricisini büyük dertlerden kurtarmaktadır. *Kdevelop* ile baştan bir proje oluşturma yapılarak istenen projeye başlanabilir. Bu proje *kdevelop* içerisinde yönetilmesi sonderce kolaydır. *Kdevelop* bünyesinde *debugger (hata ayıklayıcı),* proje yönetimi, fonksiyonları otomatik tamamlama gibi özellikleri bünyesinde taşımaktadır. *Qtdesigner* ise tamamen arayüz tasarlama işlemlerini gerçekleştirmektedir. Bu kısımda öğrenilecek bilgiler ile bu iki geliştirme aracı kullanılarak çok başarılı uygulamalar yazılabilir.

## Örnek qt Uygulaması Nasıldır? Nasıl Derlenir?

Basit olarak bir qt uygulaması yazılacaktır. Uygulama adım adım yapılanlar anlatılacaktır. Daha sonra da uygulamanın nasıl derleneceği ve neler yazılması gerektiği anlatılacaktır. Basit bir qt uygulaması uygulamanın ana formunun oluşturulması ile başlar ve diğer çalışan olaylara devredilmesi ile sona erdirilir.

Örnek basit uygulama aşağıdaki şekildedir.

```
#include <Qapplication>
#include <QPushButton>

int main( int argc, char **argv )
{
    QApplication a( argc, argv );
    QPushButton hello( "Merhaba Dünya", 0 );
    hello.resize( 100, 30 );
```

```
    a.setMainWidget( &hello );
    hello.show();
    return a.exec();
}
```

Yukarıda yazılı olan satırları `merhaba.cpp` olarak diske kaydediniz. Uygulamamız şu an için sadece ekrana yatayda 100 nokta düşeyde 30 nokta olan bir form göstermektedir.

```
#include <Qapplication>
#include <QPushButton>
```

Satırları uygulamamız içinde kullanılacak bileşenlerin bulunduğu kütüphane dosyalarını bulundurmaktadır. Uygulama içinde kullanılacak olan her bileşen ismine uygun olan dosya eklenmelidir.

```
QApplication a( argc, argv );
```

Bu satırla birlikte bir qt uygulaması oluşturulmuştur. Bu uygulamamız doğal olarak bir formdan oluşmaktadır. `argc` ve `argv` ise uygulamanın argümanında yazılan değerleri içermektedir.

```
QPushButton hello( "Merhaba Dünya", 0 );
```

Uygulama üzerine eklenecek olan *pushbutton* bileşeni oluşturuldu ve üzerinde gözükecek olan ilk yazısı belirlendi. Buton bileşeninin ilk tanımlaması yapılırken `QpushButton` tipinde tanımlanması gerekir. Bilindiği gibi verilen isimler ve kullanılan tip isimleri büyük küçük harfe karşı duyarlıdır.

```
hello.resize( 100, 30 );
```

Uygulamanın şu anına kadar iki adet bileşen oluşturmuş olduk. Biri formu işaret eden "a" diğeri ise "hello" isimli buton işaretçisidir. Bu satırla birlikte buton büyüklüğü (100,30) büyüklüğüne ayarlanmaktadır.

Kullanımı kısaca `bileşen.resize(x.y)` biçiminde kullanılır. Bu sadece bu butona ait bir alt fonksiyon değildir örnek olarak `a.resize(500,50);` yazarak formun büyüklüğünde ayarlanabilir.

```
hello.show();
```

Oluşturulan buton bileşeni ekranda görünür hale getirilmektedir. Aynı şekilde `hide()` fonksiyonu bulunmaktadır ve tabi ki diğer bileşenlerde de kullanımı aynı şekildedir.

```
return a.exec();
```

Uygulama bu satıra gelince çalışmaya başlar ve bu satırın altına geçtikten sonra uygulama sonlanmış olacaktır. Bu satırın çalışması aşamasında tanımlanmış diğer olayların çalışması beklenmektedir.

Buraya kadar basit bir uygulama yazıldı ve anlatıldı bu uygulamanın derlemesi aşamasında gerekli bir takım parametreler yazılması gereklidir. Derleme için gerekli olan örnek satırlar aşağıdaki şekildedir.

```
qmake-qt4 -project

qmake-qt4

make
```

Bu komutlar sizlere makefile dosyası oluşturacaktır. Makefile içinmde olması gerekenlerden bahsedelim. *-I* den sonra gelen bilgiler uygulama içine eklenmiş olan *.h* dosyalarının nerede bulunduklarını bildirmektedir. *-L* den sonra bilgiler ise uygulamaya ait olan kitaplıkların hangi dizin içinde bulunduğu belirtilmektedir. Arkasından ise *-o* ile modül dosyasına verilecek olan isim belirtilir. *-lqt* ise mutlaka eklenmelidir.

## Bileşenler Nasıl Formun Üzerine Yerleştirilir ve Konumu Belirlenir

Bileşenler *setGeometry* fonksiyonu ile formun üzerinde istenilen yere ve büyüklüğe konumlanması sağlanabilmektedir. Örnek *button* isimli bileşen sırasıyla yatay ve düşey olmak üzere aşağıdaki verilen örnek satırdaki noktalar üzerine konumlandırılmıştır.

```
button->setGeometry(100, 110, 120, 90);
```

## Label Bileşeni Kullanımı

Form tasarımında belki de en fazla kullanılan bileşen *label* bileşenidir. Bu bileşene dışarıdan veri girişi olmamaktadır. `qlabel.h` dosyası uygulama içine eklenmelidir. `setText()` alt fonksiyonu ile değeri değiştirilebilir.

```
#include <QLabel>
```

Yeni bir *label* bileşeni aşağıdaki satırda olduğu gibi oluşturulur.

```
QLabel *label = new QLabel;
```

Oluşturulan bileşene aşağıdaki satırda olduğu biçimde değer ataması yapılır.

```
label->setText( "ilk satır\nikinci satır" );
```

Yazı hizalaması amacıyla aşağıdaki satır kullanılabilir. `AlignRight, AlignVCenter, AlignHCenter` kullanılabilir.

```
label->setAlignment( AlignBottom | AlignRight );
```

Önemli bir hatırlatma `label->setText(tr("yazı"));` biçiminde yazılan bir metin o an kullanılan dil ayarlarına göre uygun olan ülkenin lisanı ile çıkar tabi ki çevirim işlemlerinin yapılması gerekmektedir. Bu sadece *label* bileşeni için geçerli bir durum değildir. Tırnak içinde kullanılan ve ekrana bir *text* çıkartan bütün bileşenlerde örnekte olduğu gibi kullanılır.

Örnek uygulama: *Label* bileşeni üzerinde dışarıdan dosya halinde bir *pixmap* gösterimi

```
#include <QApplication>
#include <QLabel>
#include <qpixmap.h>

int main (int argc, char* argv[])
{
```

```
QApplication myapp(argc,argv);
```

//Uygulama argümanı içerisinde bir dosya adı alınmaktadır

```
if (argc!=2)
{
debug("uygulama pixmapdosya biçiminde kullanınız");
}
```

//Bu dosya daha sonra okunarak bir *QPixmap* tipinde *bileşen* içine yerleştirilir

```
QPixmap mypixmap;
mypixmap.load(argv[1]);
```

//Okunan *pixmap* dosyası *Label* bileşeni içine yerleştirilir

```
QLabel *mylabel = new QLabel();
mylabel->resize(mypixmap.size());
mylabel->setPixmap(mypixmap);

myapp.setMainWidget(mylabel);
mylabel->show();
return myapp.exec;
}
```

*Pixmap* konusunda aşağıdaki bilgiler işinize yaracaktır. Bu amaçla uygulama içerisine *qpixmap.h* dosyası eklenmelidir.

```
QPixmap pixmap;
if (pixmap.load("ornek.bmp"))
form1->drawPixmap(100,90,pixmap);
```

## Button Bileşeni Kullanımı

*Button* bileşeni kullanımı için qt kütüphanesine ait olan `QPushButton` sınıfı kullanılmaktadır. Butonun üzerindeki yazıyı `setText();` ile üzerine resim yerleştirme işini ise `setPixmap()` alt komutları ile yapabilmekteyiz.

Aşağıdaki dosya uygulama içerisine eklenmiş olmalıdır.

```
#include <QPushButton>
```

Örnek bir *button* bileşeni oluşturma şekli aşağıdaki biçimdedir. Bu oluşturma esnasında bileşene bir takım değerler de verilebilmektedir.

```
QPushButton *p = new QPushButton( "M.Ali VARDAR", this );
```

Burada ise bileşen oluşturulup değerleri sonradan değiştirilmektedir.

```
QPushButton *p;
p->setPixmap( QPixmap("print.png") );
p->setAccel( ALT+Key_F7 );
p->setText("Filiz VARDAR");
```

## Lineedit Bileşeni Kullanımı

Ekrandan veri girişleri için kullanılan bileşendir. *QLineEdit* sınıfı bileşeni temsil etmektedir. Diğer pek çok bileşende olduğu gibi `setText("yazı")` biçiminde içinde bulunan değer değiştirilebilir. İçinde bulunan ve kullanıcı tarafından yazılmış olan değeri almak için `bileşen->text()` biçiminde kullanılır.

Örnek olarak;

```
#include <qlineedit.h>

text1->text();
```

## MultiLineedit Bileşeni Kullanımı

Bu bileşenin kullanımı sayesinde basit bir *notepad* yapılması son derece kolaydır. Birden fazla *Lineedit* bileşeninden (çok satırlı) olduğu düşünülebilir. Bileşen içerisine satır ekleme ve bu satırı okumak amacıyla aşağıdaki satırlar incelenebilir.

```
mlined1->clear; //Bileşen içerisinde bulunan bütün satırları silmektedir.
mlined1->append("Filiz VARDAR"); //İstenen yazıyı eklemektedir.
mlined1->textline(5);//İstenen bir satır içerisinde bulunan değeri okumaktadır.
```

Dosyadan okuma ve bileşen içerisine bu okunan satırları yazmak amacıyla aşağıdaki satırlar kullanılabilir.

```
Qstring file=QfileDialog::getOpenFileName();
if(file.isEmpty()) return;
mlined1->clear();
if stream inc(file.data());
char *buff=new char [300];
while (inc.getline(buff,200,'\n'))
mlined1->append(buff);
delete[]buff;
inc.close();
```

Üstteki satırların aksine bileşen içerisinde bulunan yazıları dosyaya yazma amacıyla aşağıda bulunan satırlar kullanılabilir.

```
QString file=QfileDialog::Getsavefilename();

if(file.isEmpty()) return;
ofstream inc(file.data());
for (int x=0; x<mlined1->numLines(); x++)
inc<<mlined1->textline(x)<<end1;
inc.close;
```

## Signal - Slot Kullanımı Nasıldır.?

Uygulamaların çalışma zamanları içinde çeşitli olayların oluşması ve bu olayların oluşma zamanları içinde bir takım istenen fonksiyonların çalışması halihazırda X altında uygulama yazmanın temelini oluşturmaktadır. Bu amaçla kullanılan fonksiyon *QObject* sınıfına bağlı olan *connect* fonksiyonudur.

```
#include <qapplication.h>
#include <QPushButton>
#include <qfont.h>

int main( int argc, char **argv )
{
    QApplication a( argc, argv );
    QPushButton quit( "Quit", 0 );
    quit.resize( 75, 30 );

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

```
    a.setMainWidget( &quit );
    quit.show();
    return a.exec();
}
```

Uygulama içinde `clicked()` olayı içinde `close()` fonksiyonu çağrılmaktadır. `close` yerine `quit()` yazılması da uygulamadan çıkılmasını sağlayacaktır. `close()` ise sadece o formu kapatır burada ki örnek içinde tek form olduğu için uygulama kapatılmaktadır.

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

Diğer yandan *close* yerine çalışması istenen başka bir fonksiyon yazılabilir. İstenen fonksiyonun çalıştırılmasına örnek olarak aşağıdaki satır incelenebilir.

```
connect(quit, SIGNAL(clicked()), this, SLOT(butona_basilma()));
```

*connect* fonksiyonu içerisinde bulunan parametrelerin incelenmesine sıra geldi. Birinci parametre olarak verilen "quit" olayına bir fonksiyon atanacak olan bileşenin adıdır. Uygulama içinde buna dikkat ediniz.

İkinci parametre olarak ise bileşenin hangi olayına fonksiyon bağlanacağı seçilmektedir. Bu olaylar listesi bileşenlere göre değişiklik gösterebilmektedir.

SLOT() parametresine ise olayın çalışma zamanında çalışacak olan fonksiyonun adı yazılmalıdır.

```
disconnect(quit, SIGNAL(clicked()) );
```

Satırı kullanılarak oluşturulan bağlantı koparılabilir.

## Dialog Box Kullanımı Nasıldır.?

Mesaj kutuları kullanımı son derece kolay bir biçimde bulunmaktadır. Ayrıca göze son derece hoş gelecek bir şekilde düzenlenmiştir. Uygulama içerisinde mesaj kutularının kullanılabilmesi amacıyla `qmessagebox.h` dosyasının eklenmesi gerekmektedir.

```
#include <qmessagebox.h>
```

Biçiminde yapılan bir ekleme sonrası aşağıda kullanım şekli görülmektedir.

```
QMessageBox::information(this,"üst yazı","iç yazı");
```

Fonksiyonun parametreleri; *information* kısmına çıkacak olan mesaj kutusundaki mesajın ne türde olduğuna dair verilen ufak resimciğin ne olduğuna karar verilebilmektedir. Aşağıda ne türde olabilecekleri yazmaktadır.

- `information`
- `critical`
- `about`
- `warning`

biçimlerini alabilmektedir. Bu tip bir kullanım ekrana bir "Ok" butonu çıkaracaktır. Aşağıdaki örnekle birlikte kullanımda istenen buton çıkarılması gösterilmektedir.

```
QMessageBox::information(this, "üst yazı", "iç yazı", QMessageBox::Ok);
```

Bu satırla birlikte ekranda tamam butonu ile birlikte yazılan mesaj çıkacaktır. Örnek içinde "" içinde gösterilen "iç yazı" mesaj kutusunun içinde bulunacak olan yazıdır. "üst yazı" ise mesaj kutusunun çerçevesinde karşımıza çıkacak olan yazıdır.

Mesaj kutuları ile basit bir *about* (hakkında) kutusu gösterilebilmektedir. Bu amaçla aşağıdaki biçimde kullanılmaktadır.

```
QMessageBox::about(this, "hakkında", "Bu program ...."VERSION" \n (c)....");
```

Mesaj kutuları aynı zamanda birden fazla buton ile birlikte bir sorgulama amacıylada kullanılabilir. Bu bir çok açıdan uygulama yazanları büyük bir dertten kurtarmaktadır.

```
int exit=QmessageBox::information(this,"çıkılsın
    mı?","çıkalımmı", QMessageBox::Ok, QMessageBox::Cancel);

if (exit==0)
{
```

/*uygulamanın burasında ok butonuna basıldığı anlaşılmaktadır*/

```
}
```

*exit* değişkeni "0" değerini alması durumunda ok *butonuna* basıldığı anlaşılmaktadır. Dikkat edilecek olan buton sırasına göre değerin arttığıdır. Aşağıdaki listede kullanılabilecek olan buton tipleri listelenmiştir.

- `QMessageBox::NoButton`
- `QMessageBox::Ok`
- `QMessageBox::Cancel`
- `QMessageBox::Yes`
- `QMessageBox::No`
- `QMessageBox::Abort`
- `QMessageBox::Retry`
- `QMessageBox::Ignore`

## File Dialog Bileşeni Kullanımı

Kullanıcılardan bir dosya adı veya dizin adı girilmesi istenmesi durumunda *qt* içerisinde hazır olarak kullanılmakta olan bir diyalog penceresi mevcuttur. Bu diyalog kullanıcının seçtiği dosyayı dosya veya dizini geri döndürmektedir.

Örnek olarak oluşturma işlemi aşağıdaki şekilde olmaktadır.

```
QFileDialog* fd = new QFileDialog( this, "file dialog", TRUE );
```

*setMode* değeri ile gösterme modu belirlenmektedir.

```
fd->setMode( QFileDialog::AnyFile );
```

- `QFileDialog::AnyFile` : Herhangi bir dosya adı çıkmasıdır.
- `QFileDialog::ExistingFile` : Var olan tek dosya adıdır.
- `QFileDialog::Directory` : Dizin adıdır ancak dosyalar gözükür.
- `QFileDialog::DirectoryOnly` : Sadece directory dosyalar gözükmez.

- `QFileDialog::ExistingFiles` : Birden fazla var olan dosyadır.

Kullanıcı seçim işlemlerini yaparken seçilecek olan dosyalar arasında sorgulama işlemini yapmaktadır.

```
fd->setFilter( "Images (*.png *.xpm *.jpg)" );
fd->setFilter( "All C++ files (*.cpp *.cc *.C *.cxx *.c++)" );
fd->setFilter( "*.cpp *.cc *.C *.cxx *.c++" );
```

Dosya adı aşağıdaki şekilde alınabilmektedir.

```
QString fileName;
if ( fd->exec() == QDialog::Accepted )
    fileName = fd->selectedFile();
```

## Form Kullanımına İlişkin Bilgiler

Formların tasarlanması sırasında kullanılan birtakım fonksiyonlar bulunmaktadır. Formun başlığının değiştirilmesi amacıyla aşağıdaki satır kullanılabilir.

```
setCaption("M.Ali VARDAR 1999 Personqt");
```

Büyüklüğü istenen bir büyüklüğe ayarlanabilir.

```
resize(400,400);
```

İstenen bir büyüklükten daha büyük olamaması sağlanabilir.

```
setMaximumSize(QSize(100,200));
```

Geri plan rengi değiştirilebilir.

```
setBackgroundColor(QColor(0,0,0));
```

Yalnız bu özellikler aşağıda verilen örneklerde olduğu gibi her bileşen için geçerlidir.

```
MyWidget* widget = new MyWidget
...
QPoint p = widget->pos();
QSize s = widget->size();
...
widget = new MyWidget;
widget->resize( s );
widget->move( p );
widget->show();
```

### Anında Yardım Fonksiyonları

## *Whatsthis* ve Tip kullanımı nasıldır.?

Formlar üzerinde bulunan bileşenlerin hangi amaçla düzenlendiğini sadece yardım butonuna ve daha sonra bilgi almak istenen bileşene tıklanması durumunda çıkan bilgilerdir. Fare ile tıklanabilen her bileşen aşağıdaki şekilde yardım mesajı yerleştirilebilir.

```
editCopy->setWhatsThis(tr("Bileşen\n\nYardım için
                           menulere bakınız"));
```

*Toolbar* üzerinde bir yardım başlatma amaçlı butonun nasıl konulduğu aşağıda bulunan örnekte gösterilmiştir.

```
void AsdApp::initToolBar()
{
  fileNew = new QAction(tr("New File"), newIcon, tr("&New"),
                   QAccel::stringToKey(tr("Ctrl+N")), this);
  fileNew->setStatusTip(tr("Creates a new document"));
  fileNew->setWhatsThis(tr("New File\n\nCreates a new document"));
  connect(fileNew, SIGNAL(activated()), this, SLOT(slotFileNew()));

  fileOpen = new QAction(tr("Open File"), openIcon, tr("&Open..."), 0, this);
  fileOpen->setStatusTip(tr("Opens an existing document"));
  fileOpen->setWhatsThis(tr("Open File\n\nOpens an existing document"));
  connect(fileOpen, SIGNAL(activated()), this, SLOT(slotFileOpen()));

  fileToolbar = new QToolBar(this, "file operations");
  fileNew->addTo(fileToolbar);
  fileOpen->addTo(fileToolbar);

  fileToolbar->addSeparator();
  QWhatsThis::whatsThisButton(fileToolbar);
}
```

## Tip Kullanımı

Bileşenler üzerine fare getirilerek üzerine çıkan yazı kullanımı tüm bileşenler için aynı şekilde yapılmaktadır. Bu canlı olarak yapılan yardım aşağıdaki şekilde olmaktadır.

```
bileşen->setStatusTip(tr("Açıklama yazısı"));
```

## Status Bar Kullanımı

Status bar bileşeni anlık mesajlar için oldukça kullanışlıdır. Form üzerine yerleştirilen bir *status* bar ile birlikte mesaj düzenleme işlemi sadece aşağıdaki satır kullanılarak kolaylıkla yapılabilir.

```
statusbar->message("örnek mesaj");
```

## Clipboard Kullanımı Nasıldır.?

Linux altında bir çok farklı kütüphaneler kullanılarak yazılan uygulamalar olması sebebiyle yazılan yazıları uygulamalar arasında kopyalama işlemlerinde bir takım sorunlar ortaya çıkmaktadır. Aşağıdaki anlatımda qt kütüphanesine ait olan kopyalama ve yapıştırma işleminden bahsedilmektedir.

Burada daha önceden `clipboard` içine alınmış olan bir metne nasıl ulaşılacağı görülmektedir. `QClipboard` tipinde oluşturulan bir değişkene uygulama içinden değer alınmaktadır.

```
QClipboard *cb = QApplication::clipboard();
```

Alınan metnin yerleştirileceği değişken tanımlanmaktadır.

```
QString text;
```

Tanımlanan değişken içerisine `clipboard` içinden alınan metin bu aşamada yerleştirilmektedir.

```
text = cb->text();
```

Bu aşama yazı yazmak için kullanılan uygulamalarda yapıştırma *(paste)* işlemi gerçekleştirilmiştir. Aşağıdaki satırlar ile birlikte bu metne ulaşılmış olur.

```
if ( text )
   qDebug( "Clipboard içeriği: %s", text );
```

Tuş takımı kontrol edilerek "Ctrl-v" tuslarına basılması durumunda yukarıdaki satırların çalıştırılması durumunda uygulama içine eklenebilir.

Bu aşamaya kadar hazır olan bir yazıyı aldık peki biz nasıl `clipboard` içerisine bu atamayı gerçekleştirebiliriz. Metin yazma uygulamalarında çoğunlukla kullanılan "Ctrl-c" tuş takımına atanacak olan değer aşağıdaki şekilde olmalıdır.

```
cb->setText("Bu yazı clipboard içine yerleştirilmektedir.");
```

Anlatılanları kısaca özetlemek gerekirse oluşturulan `cb` isimli `clipboard` bileşeni içindeki değere `cb->text()` biçiminde ulaşıyor, eğer değeri değiştirmek istiyorsak `cb->setText("yazı")` biçiminde değiştiriyoruz.

### Font Kullanımı Nasıl Olmaktadır.?

Uygulamalarda kullanılan yazılara çeşitli amaçlarla kullanılmak üzere çeşitli fontlar ve büyüklükler kullanılabilir. Bu amaçla yazım amaçlı kullanılan bütün bileşenler içinde `setFont` kısmına çeşitli değerler yazan yazılar çeşitli şekillerde ayarlanabilir. Örnek olarak daha önceden tanımlanan bir bileşenin font bilgilerini değiştirmek amacıyla aşağıdaki şekilde yazılabilir. `button10` isimli bileşenin font bilgisi *Times new roman,* kalın ve 18 punto olarak ayarlama şekli aşağıda gözükmektedir

```
button10->setFont( QFont( "Times", 18, QFont::Bold ) );
```

Aynı şekilde sadece butonun (veya başka yazı içeren bileşenin) font bilgisi de değiştirilebilir.

```
defaultButton->setFont( QFont( "times" ) );
```

Aynı şekilde bütün bahsedilen işlemler butonun oluşturulma zamanında da belirlenmesi mümkündür.

```
sansSerifButton = new QPushButton("Sans Serif",
                                      this,"pushbutton2" );
```

### Cursor Kullanımı Nasıl Olmaktadır.?

Farenin belli bir bileşen üzerine geldiği zaman değişikliğe uğraması istenebilir ve yahut ta uygulama çalışırken istenen bir zamanda fare tipi değiştirilmek istenebilir. Bu durumda kullanılacak olan komut `setCursor()` fonksiyonudur. Bir bileşen için kullanımı aşağıdaki şekilde olmaktadır.

```
label->setCursor( deger );
```

Genel olarak değiştirilmek isteniyorsa aşağıdaki şekilde yazılabilir.

```
setCursor(deger);
```

`qcursor.h` dosyası aşağıdaki şekilde uygulama içerisine eklenmelidir.

```
#include <qcursor.h>
```

Aşağıda verilen fare imleçleri hazır olarak kullanılabilmektedir.

- `ArrowCursor`
- `UpArrowCursor`
- `CrossCursor`
- `WaitCursor`
- `IbeamCursor`
- `SizeVerCursor`
- `SizeHorCursor`
- `SizeBDiagCursor`
- `SizeFDiagCursor`
- `SizeAllCursor`
- `BlankCursor`
- `SplitVCursor`
- `SplitHCursor`
- `PointingHandCursor`
- `ForbiddenCursor`
- `WhatsThisCursor`

## Ses Kullanımı Nasıldır.?

Bir çok uygulama içinde sesli mesajlar uygulama kalitesini artırıcı ve daha hoş olmasını sağlamaktadır. `Qt` kütüphanesi içinde kullanımı son derece kolaydır. Tüm işi `QSound::play(dosyaadı)` kullanılması yeterlidir. Uygulama içerisine aynı zamanda "`qsound.h`" kitaplığının eklenmesi gerekmektedir.

Uygulama içerisine eklenecek olan dosya başlığı;

```
#include "qsound.h"
```

Kullanım şekli;

```
QSound::play(dosyaadı)
```

Ses çıkarma işlemini yerine getiren ses server varlığı kontrol işlemi aşağıdaki şekilde yapılabilir. Koşulun oluşması durumunda ses çalacak bir ortam bulunamamaktadır.

```
if (!QSound::isAvailable())
{
 QMessageBox::warning(this,"Ses mevcut değil","Ses aygıtı
                                                bulunamadı");
}
```

Ses kullanımına dair örnek satır aşağıda bulunmaktadır.

```
void sescal()
{
    QSound::play("ses.wav");
}
```

## Qt ile Çok Görevlilik

### Timer Kullanımı Nasıldır?

Uygulamalar içerisinde belli zaman aralıklarıyla birtakım fonksiyonların çalışması sırasında başka bir takım fonksiyonlarının çalışmasının istenmesi durumunda *timer* bileşenini kullanma zorunluluğu karşımıza çıkacaktır. Buradan anlaşılacak olan tam olan tam çoklu görevlilik değildir. Tam bir çok görevlilik söz konusu olunca `thread` kullanımı incelenmelidir. `Timer` kullanımı için öncelikli olarak aşağıdaki satırda belirtildiği üzere bir `QTimer` nesnesi tanımlanmalıdır.

```
QTimer * counter = new QTimer( this );
```

`counter` ismiyle tanımlanan `QTimer` nesnesinin belirlenen zaman aralıkları içerisinde hangi fonksiyonun çalışacağı belirlenmelidir. Bu amaçla `connect` fonksiyonu ile `counter` bileşenin `timeout` olayına `updateCaption` zamanını bağlantı kurulmaktadır.

```
connect( counter, SIGNAL(timeout()), this, SLOT(updateCaption()));
```

Uygulamanın ne kadar süre aralığı sonunda belirlenen fonksiyonu çalıştıracağı belirlenmektedir. Bu zaman 1000 birime karşılık olarak 1 saniyedir. `counter->start` komutu ile birlikte artık sayaç zamanı başlayacaktır.

```
counter->start( 1000 );
```

Kullanılan zamanlayıcının çalışıp çalışmadığını anlamak amacıyla aşağıdaki örnek satır anlaşılmalıdır. `if (!counter->isActive())` kullanılarak zamanlayıcının çalışıp çalışmadığı kontrol edilebilir. Zamanlayıcı durdurmak amacıyla aşağıdaki komut kullanılabilir.

```
counter->stop();
```

### Timer, Canvas  Kullanımına Örnek Uygulama

*Timer* ve *canvas* kullanımına örnek olması açısından *personqt* isimli uygulamanın ana kodunun olduğu alanı detaylı olarak anlatılmasını uygun buldum. Uygulama içinde form üzerinde çeşitli noktalar yerleştirilmektedir. Bu noktalar ise belli zaman aralıklarıyla sola doğru hareket ettirilmek suretiyle bir animasyon etkisi yaratılmaktadır.

```
#include "anaform.h"

anaform::anaform(QWidget *parent, char *name)
:QDialog(parent,name,TRUE,WStyle_Customize | WStyle_NormalBorder)
{
```

//Uygulamanın ana formunun başlığı değiştirilmektedir.

```
setCaption("M.Ali VARDAR 1999 Personqt");
```

//Ana formun büyüklüğü yatayda ve düşeyde 400 noktaya ayarlanmaktadır.

```
resize(400,400);
```

//Formun geri plan rengi r,g,b olmak üzere siyah yapılmaktadır.

```
setBackgroundColor(QColor(0,0,0));
```

//Uygulama içinde kullanılacak `olan` zamanlayıcı oluşrulmaktadır.

```
timer1 = new QTimer(this, "timer1");
```

//Zamanlayıcı saniyenin onda biri bir zamanda çalışmak üzere başlatılmaktadır.

```
timer1->start(10,FALSE);
```

/*Zamanlayıcının çalışması amacıyla `timeout` olayına `zamanlayici()` isimli fonksiyon bağlanmaktadır*/

```
QObject::connect(timer1,SIGNAL(timeout()), this,SLOT(zamanlayici()));
```

//Yıldız hissi oluşturacak olan koordinatlar gelişigüzel olarak belirlenmektedir

```
int sayac;
for (sayac=1;sayac<101;sayac++)
{
x1[sayac]=rand()%400;
y1[sayac]=rand()%400;
x2[sayac]=rand()%400;
y2[sayac]=rand()%400;
x3[sayac]=rand()%400;
y3[sayac]=rand()%400;
}
}
```

/* Zamanlayıcının çalışacağı fonksiyon burada başlamaktadır. Yıldızların kayma işlemi yapılmaktadır.*/

```
void
anaform::zamanlayici()
{
int sayac;
```

//Noktaların basılacağı formun üstünde olan alan *`QPainter`* olarak oluşturulmaktadır

```
QPainter p(this);

for (sayac=1;sayac<101;sayac++)
{
```

//*`setPen`* komutu ile renk belirlenmektedir

```
p.setPen(15);/*siyah renk 15*/
```

//Aşağıdaki komut ile birlikte yatay ve düşey pozisyonu belirtilen yerlere üst satırda tanımlanan renkte bir nokta basılmaktadır*/

```
p.drawPoint(x1[sayac],y1[sayac]);
p.drawPoint(x2[sayac],y2[sayac]);
p.drawPoint(x3[sayac],y3[sayac]);

x1[sayac]=x1[sayac]-1; if (x1[sayac]<1) x1[sayac]=401;
x2[sayac]=x2[sayac]-2; if (x2[sayac]<1) x2[sayac]=401;
x3[sayac]=x3[sayac]-3; if (x3[sayac]<1) x3[sayac]=401;
p.setPen(255);

p.drawPoint(x1[sayac],y1[sayac]);
p.drawPoint(x2[sayac],y2[sayac]);
p.drawPoint(x3[sayac],y3[sayac]);
```

```
}
p.end();  }
```

## Qthread Kullanımı

*Thread* kullanımı için *qt* kütüphanesi içerisinde bu işlemleri son derece kolaylaştıracak şekilde tasarlanmıştır. İstenirse *pthread.h* kullanılabilir ancak qt içerisinde bulunan kullanımı son derece kolaydır. Sınıf yapısı kullanımı ile son derece basite indirilmiştir. Uygulama içerisinde kullanılmak istenmesi durumunda *qthread.h* eklenmelidir. Eğer uygulama *kdevelop* ile geliştirilmiyorsa derleme parametresine veya *makefile* dosyasına *–lqt –mt* parametreleri eklenmelidir. *Kdevelop* ile geliştirme söz konusu ise kendisi proje üzerinde değişiklik yapmış olacaktır.

`Qthread` tipinde tanımlanan bir sınıf üzerinde gerekli olan çalışması istenen kodlar bu sınıfa ait olan `void MyThread::run()` içerisine yazılacaktır. Oluşturulan `thread_adi.start()` komutu ile birlikte oluşturulan süreç başlatılacaktır. Sınıf yapısından dolayı istenildiği kadar bu süreçten oluşturulabilecektir. Buradaki basit uygulama *qthread* yapısı anlaşılması açısından oldukça basittir. Sadece basit bir sınıf tanımlanacaktır. Bu sınıf daha sonra *main()* içerisinde başlatılmaktadır.

/**qthread* işlemlerinin kullanılabilmesi amacıyla bu başlık dosyası eklenmelidir:*/

```
#include "qthread.h"
```

/*Uygulama içerisinde kullanılacak olan *thread* sınıfımız tanımlanmaktadır. Bu sınıf içerisinde çalışma zamanında çalışacak olan fonksiyonumuz tanımlanmaktadır. */

```
class MyThread : public Qthread
{
public:
          virtual void run();
};
```

/*Çalışacak olan kodlar bu fonksiyon içerisinde yazılmaktadır. */

```
void MyThread::run()
{
```

/*Çalışan süreçler arasında ekrana yazılan yazılarda tutarsızlık olacaktır.*/

```
    for(int count=0;count<20;count++)
    { sleep(1);
    printf("Thread işlemi:%i\n",count);
    }
}

int main()
{
  MyThread a;
  MyThread b;

  a.start();
  b.start();
  a.wait();
}
```

Kullanılan *qthread* sınıflara ilişkin bir takım bilgiler edinilebilir bu bilgiler aşağıda anlatılmaktadır.

```
bool QThread::finished () const
```

Çalıştırılan *thread* sonlandırılacaktır. Dönen değer neticesinde işlemin başarılı olup olmadığı anlaşılacaktır.

```
void QThread::usleep ( unsigned long usecs ) [static protected]
```

Çalıştırılan *qthread* belirtilen süre kadar (*mikrosec*) bekletilecektir.

```
void QThread::start ()
```
Süreç başlatılacaktır.

M.Ali VARDAR – 2006 Haziran

ali@linuxprogramlama.com
www.linuxprogramlama.com