Mohamad Iqbal

# Membuat Aplikasi Mobile dengan Qt V2


Code less.
Create more.
Deploy everywhere.

Nokia.Ice
Nokia Indonesia Community Enthusiast

# Kata Pengantar

Alhamdulillah, puji syukur kepada Allah SWT. sehingga saya dapat menyelesaikan ebook yang berjudul **Membuat Aplikasi Mobile dengan Qt** dalam waktu yang relatif sangat singkat. Ebook ini merupakan revisi dari ebook sebelumnya dengan judul yang sama namun ada penambahan materi baru terkait teknologi mobile Qt.

Ebook ini ditulis dengan tujuan untuk memberikan pengetahuan tentang teknologi mobile Qt yang memungkinkan Anda membuat aplikasi mobile dengan mudah khususnya pada device Nokia.

Penulis adalah mahasiswa asal Serang-Banten yang saat ini sedang menempuh studi S1 di UNIKOM Bandung jurusan Manajemen Informatika. Silakan kirimkan kritik, saran ataupun koreksi tentang konten hingga tata bahasa dalam ebook ini ke email ciebal745@gmail.com. Semuanya jelas akan saya terima dengan senang hati karena saya sendiri masih dalam tahap pembelajaran.

Tidak lupa saya sampaikan terimakasih kepada **Nokia Indonesia Community Enthusiast (NICE)**, **Bapak Firstman Marpaung** (MVP Windows Phone) yang memperkenalkan teknologi Qt kepada saya, **Bapak Narenda Wicaksono** (Developer Marketing Manager Nokia Indonesia) yang mendorong saya untuk terus belajar sejak beliau masih menjabat sebagai Technical Advisor Microsoft Indonesia, dan pembaca yang sudah merelakan bandwidth-nya untuk mendownload ebook ini serta meluangkan waktu untuk membacanya.

Bandung, April 2011

Mohamad Iqbal

# Pendahuluan

Indonesia merupakan sebuah pasar yang cukup unik. Telepon genggam bukan merupakan sekedar perangkat untuk bertelekomunikasi, namun sudah menjadi sesuatu yang menjadi bagian dari gaya hidup. Tak heran di Indonesia kita bisa melihat mayoritas pengguna telepon genggam tidak memaksimalkan semua fitur yang ditawarkan oleh sebuah telepon genggam. Tak jarang beberapa orang memilih telepon genggam tertentu sebagai parameter kelas sosial.

Dinamika jaringan sosial seperti facebook dan twitter yang berkembang pesat di Indonesia juga memberikan kontribusi yang signifikan pada adopsi piranti telepon genggam, mengingat telepon genggam adalah piranti utama bagi pengguna jaringan sosial untuk berinteraksi.

Kehadiran Ovi Store memberikan kesempatan kepada developer untuk menjual langsung aplikasinya kepada pengguna Nokia yang tidak hanya ada di Indonesia, namun juga diseluruh dunia. Ovi Store saat ini telah mencapai 4 juta unduh per hari diseluruh dunia dan telah memiliki kerjasama dengan lebih dari 100 operator diseluruh dunia menghadirkan operator billing.

Nokia juga telah bekerja sama dengan beberapa operator seperti Telkomsel, Indosat, XL dan 3 memberikan untuk data plan yang kompetitif bagi penggunanya. Nokia Unlimited Data Plan memungkinkan pengguna memperoleh layanan internet tak terbatas langsung dari ponsel Nokia untuk memudahkan akses pengguna terhadap aplikasi yang ditawarkan oleh developer melalui Ovi Store.

## Potensi Bagi Developer

Bila integrated operator billing hadir di Indonesia, maka pengguna Nokia dimungkinkan untuk membeli aplikasi melalui Ovi Store dengan mekanisme potong pulsa, tentu ini sangat relevan dengan market Indonesia dimana penetrasi kartu kredit masih belum signifikan.

Terima kasih kepada siapapun yang telah membawa teknologi Ring Back Tone (RBT) pasar ke Indonesia yang telah membuat pengguna layanan seluler di Indonesia terbiasa dengan skenario potong pulsa. RBT 3 tahun lalu merupakan sesuatu yang tidak populer, namun saat ini RBT adalah andalan para musisi lokal yang sudah tidak peduli dengan penjualan CD / kaset bajakan, karena yang jelas RBT tidak bisa dibajak

dan pengguna diharuskan mendaftar dengan kemudian dipotong pulsanya. Integrated operator billing di Ovi Store adalah komitmen jangka panjang Nokia dengan developer di Indonesia.

## Relevansi QT

Perkembangan penggunaan Qt di Indonesia sangat pesat. Qt tidak hanya digunakan untuk pengembangan aplikasi mobile untuk platform Symbian dan Meego, namun Qt juga digunakan untuk pengembangan aplikasi desktop di platform Linux. Penggunaan Linux di Indonesia yang semakin tinggi memicu pesatnya perkembangan penggunaan Qt sebagai kakas pemrograman utama diplatform tersebut.

Qt akan tetap menjadi kakas pengembangan aplikasi yang penting bagi Nokia. Saat ini telah beredar lebih dari 200 juta telepon genggam dengan sistem operasi Symbian dan Nokia memiliki target penjualan 150 juta Symbian kedepannya. Mengingat semua platform Symbian memiliki akses ke Ovi Store, tentu kesempatan bagi developer di area ini masih terbuka lebar.

Selain itu Qt adalah sebuah platform yang dibelahan dunia lain digunakan sebagai kakas untuk berinovasi, tidak hanya untuk membuat aplikasi mobile, sebagai contoh Dreamworks merubah semua kakas animasi internalnya ke Qt dan menggunakannya untuk membuat film seperti “MegaMind” dan “How to Train Your Dragon”. Qt kedepannya akan tetap menjadi kakas untuk berinovasi baik di desktop, web, maupun mobile.

## Kesimpulan

Nokia masih akan terus membuat piranti berbasis sistem operasi Symbian dan melihat angka yang cukup signifikan dari piranti Symbian yang hadir di pasar global, hal ini merupakan kesempatan yang menjanjikan bagi developer. Melihat market Indonesia, Symbian masih sangat relevan dan investasi belajar di Qt merupakan sesuatu yang penting.

Akan adanya integrated operator billing merupakan kesempatan emas yang tidak boleh disia-siakan mengingat para developer dari luar saat ini sedang berlomba-lomba untuk menggarap pasar aplikasi mobile di Indonesia. Saat aplikasi telah menjadi komoditas di ranah mobile, sudah selayaknya developer lokal yang menjadi tuan rumah dan ini adalah saat yang terbaik untuk memulai.

*Dikutip dari tulisan Narenda Wicaksono (http://blog.narenda.com/index.php/archives/838)*

# Daftar Isi

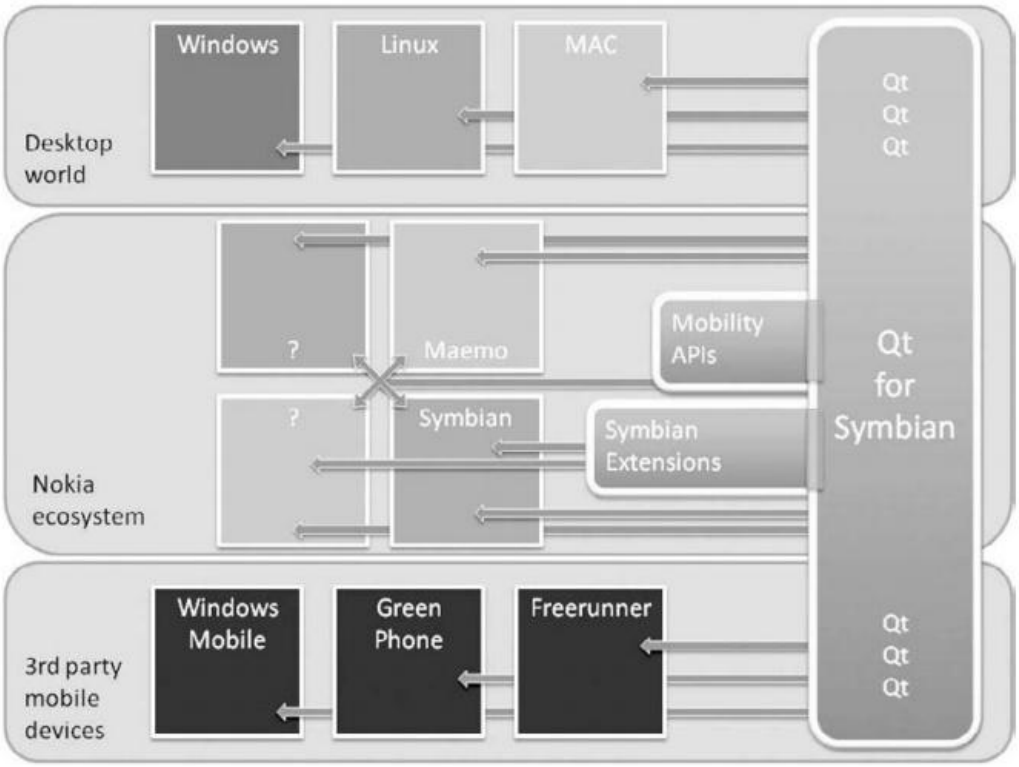
# Apa itu Qt?

Qt (dibaca : kiut) dibuat pada tahun 1996 oleh perusahaan dari swedia yang bernama Trolltech. Qt memiliki sifat lintas platform maka developer dapat membuat aplikasi yang berjalan pada platform Windows, Linux, dan Mac. Dengan Qt kode yang sama dapat dijalankan pada target platform yang berbeda.

Qt dirancang untuk pengembangan aplikasi dengan C++. Oleh karenanya, Qt berisi sekumpulan kelas-kelas yang tinggal dimanfaatkan saja, mulai dari urusan antarmuka (user interface), operasi input ouput, networking, timer, template library, dan lain-lain. Qt mendukung penuh Unicode (mulai versi 2.0) sehingga urusan internationalization (I18N) dan encoding teks bukan menjadi masalah. Walaupun merupakan free software, Qt terbukti stabil dan lengkap. Dibandingkan toolkit lain, Qt juga mudah untuk dipelajari dan dipersenjatai dengan dokumentasi dan tutorial yang ekstensif dan rinci.

Pada tahun 2008, Nokia mengakuisisi Trolltech untuk memperlancar strategi pengembangan aplikasi lintas platform. Saat ini strategi Nokia adalah memfokuskan teknologi pengembangan aplikasi mobile pada Qt sebagai single app development framework.

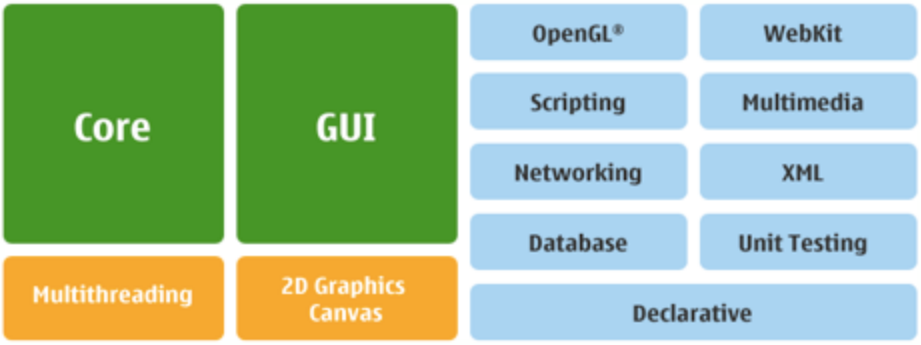
Untuk platform mobile, Qt mendukung beberapa sistem operasi diantaranya Symbian S60, Maemo, Symbian^3, dan MeeGo. Sedangkan untuk platform desktop, Qt mendukung sistem operasi Windows, Linux, dan Mac.




Para developer dapat membuat aplikasi yang ditujukan untuk 80 juta pengguna symbian devices karena Qt mendukung untuk pengembangan berbasis Symbian S60 dan Symbian^3.

Beberapa aplikasi ternama juga telah menggunakan Qt Framework diantaranya Google Earth map application, Skype telephony application, VLC media player, KDE desktop environment, dan masih banyak lagi. Qt juga digunakan pada berbagai perangkat elektronik dan aplikasi industri, sebagai contoh adalah mobile transportation system yang dibuat oleh Volvo, MeVisLab digital imaging platform, dan RealFlow visual effect application pada industri dunia hiburan.

Tersedia beberapa class library yang dapat digunakan oleh developer untuk mempercepat pembuatan program, misalnya library untuk membuat GUI (Graphical User Interface), network programming, dan library untuk bekerja dengan XML.




Nokia mempermudah pengembangan aplikasi mobile dengan menyediakan Nokia Qt SDK yang berisi class library, IDE (Qt Creator), dan Qt Simulator.

## Bahasa Pemrograman untuk Qt

Qt mendukung pengembangan dengan dua bahasa utama yaitu Object Oriented C++ dan Java. Namun untuk membuat aplikasi mobile diatas Nokia Qt SDK, bahasa pemrograman yang harus Anda kuasai adalah C++. Penggunaan bahasa java digunakan untuk membuat aplikasi berbasis desktop menggunakan Qt SDK.


## Lisensi

Isu yang paling penting untuk developer adalah lisensi. Qt menawarkan 3 jenis lisensi, yaitu:

- Qt Commercial
- Qt GNU LGPL v2.1.
- Qt GNU GPL v3.0.

Perbedaan dari ketiga jenis lisensi tersebut bisa dilihat dalam table berikut:

| | Commercial | LGPL | GPL |
|---|---|---|---|
| Biaya Lisensi | Berbayar | Tidak Berbayar | Tidak Berbayar |
| Harus menyediakan setiap perubahan kode program pada Qt | Setiap perubahan kode program tidak wajib diserahkan ke Qt. | Kode program harus disediakan | Kode program harus disediakan |
| Dapatkah membuat aplikasi komersial? | Ya, kode program tidak wajib dibuka. | Ya, disesuaikan dengan aturan LGPL versi 2.1 | Tidak boleh, untuk setiap aplikasi GPL, kode program harus disediakan. |
| Ketersediaan Pemutakhiran | Ya, langsung diinformasikan jika ada pemutakhiran framework Qt | Ya, Tersedia | Ya, Tersedia |
| Support | Ya, tersedia sesuai dengan perjanjian dan lisensi yang dimiliki | Tidak termasuk, tetapi dapat dibeli sesuai dengan kebutuhan | Tidak termasuk, tetapi dapat dibeli sesuai dengan kebutuhan |
| Biaya untuk runtime | Ya, untuk beberapa pemakaian pada platform embedded | Tidak ada | Tidak ada |

Untuk lebih jelas mengenai lisensi Qt, dapat dilihat di : http://qt.nokia.com/products/licensing/

## Kebutuhan Sistem

Untuk penggunaan Qt ini ada beberapa kebutuhan software dan hardware yang harus terpenuhi, diantaranya :

- Windows 7
- Windows XP Service Pack 2
- Windows Vista
- (K)Ubuntu Linux 8.04 (32-bit and 64-bit) or later, with the following:
    - g++
    - make
    - libglib2.0-dev
    - libSM-dev
    - libxrender-dev
    - libfontconfig1-dev
    - libxext-dev
    - libfreetype6-dev
    - libx11-dev
    - libxcursor-dev
    - libxfixes-dev
    - libxft-dev
    - libxi-dev
    - libxrandr-dev
    - If you are using QtOpenGL, libgl-dev and libglu-dev
- Mac OS 10.5 or later with the following:
    - Xcode tools for your Mac OS X version available from your Mac OS X installation DVDs or at http://developer.apple.com.

Tabel berikut merangkum dukungan sistem operasi untuk mengembangkan aplikasi untuk platform perangkat mobile.

| Operating system | Platform | | | |
|---|---|---|---|---|
| | Desktop | Qt Simulator | Maemo | Symbian |
| Windows | Yes | Yes | Yes | Yes |
| Linux | Yes | Yes | Yes | Yes (by using Remote Compiler for building) |
| Mac OS X | Yes | Yes | Yes | Yes (by using Remote Compiler for building) |

## Persiapan Pengembangan

Untuk memulai pengembangan dan belajar membuat aplikasi menggunakan Qt maka kita harus mempersiapkan:

1. Platform yang akan dipakai. Dalam hal ini apakah system operasi yang Anda gunakan Windows, Linux, atau Mac.
2. Download Qt SDK sesuai dengan platform. Anda dapat mengunduh Qt SDK pada alamat http://qt.nokia.com/downloads.
3. Instal Qt SDK. Untuk proses instalasi akan dibahas pada bab selanjutnya.
4. Mulai membuat program dari yang sederhana sampai yang sulit. Atau Anda bisa jalankan contoh program yang telah disediakan.

Pada saat akan mengunduh Qt SDK pada website ada dua pilihan yang tersedia untuk membuat aplikasi mobile dengan Qt yaitu:

- Qt SDK
- Nokia Qt SQK

Adapun perbedaan dari masing-masing SDK dapat dilihat pada gambar berikut:

| Nokia Qt SDK 1.0 | Qt SDK (Qt 4.6.3) | Feature/Component | Comments |
|---|---|---|---|
| ✔ | ✔ | Qt source | |
| ✔ | ✔ | Qt Creator | |
| ✘ | ✔* | Qt binary build - Windows | * Qt SDK for Windows |
| ✘ | ✔* | Qt binary build - Linux | * Qt SDK for Linux |
| ✘ | ✔* | Qt binary build - Mac | * Qt SDK for Mac |
| ✔ | ✘ | Qt binary build - Symbian | |
| ✔ | ✘ | Qt binary build - Maemo | |
| ✘* | ✘* | Qt binary build - MeeGo | * under roadmap review |
| ✘ | ✘ | Qt binary build - Windows Mobile | |
| ✘ | ✘ | Qt binary build - Windows CE | |
| ✔* | ✔ | Developer environment - Windows | * Symbian and Maemo support |
| ✔* | ✔ | Developer environment - Linux | * Maemo support |
| ✘* | ✔ | Developer environment - Mac | * Maemo support |
| ✔ | ✔ | Debugger support | |
| ✔ | ✘* | Qt APIs for mobile development | * available as add-on |
| ✔ | ✘ | Build tool chain - Symbian/S60 | |
| ✔ | ✘ | Build tool chain - Maemo | |
| ✔ | ✘ | Qt Simulator | |
| 13 MB | ✘ | Package size - online installer | |
| 840 MB | <500 MB | package size - offline installer | |

Untuk keperluan development, saya sarankan untuk mengunduh Nokia Qt SDK karena didalamnya sudah terdapat Qt Simulator sehingga walaupun Anda tidak memiliki device nokia kita tetap dapat menjalankan aplikasi yang telah kita buat melalui simulator.

Anda dapat mengunduh Nokia Qt SDK pada alamat http://www.forum.nokia.com/Develop/Qt/. Unduh sesuai dengan platform yang digunakan dan saya sarankan mendownload offline installer bukan online installer.

## Qt Development Application

Ada 2 tool utama yang terdapat pada Nokia SDK, yaitu:

### *Qt Creator*

Qt Development Framework menawarkan Qt Creator sebagai lingkungan pengembangan terintegrasi untuk pengembangan Qt. Beberapa fitur dari Qt Creator adalah:

- Muncul dengan alat yang berguna yang memungkinkan pengembangan aplikasi efisien dan terlihat professional.
- Ringan, open source dan mudah digunakan. Juga berjalan pada beberapa platform.
- Memiliki fitur seperti debugger visual, wizard proyek pembuatan dan integrasi dengan version control systems.

### *Qt Linguist*

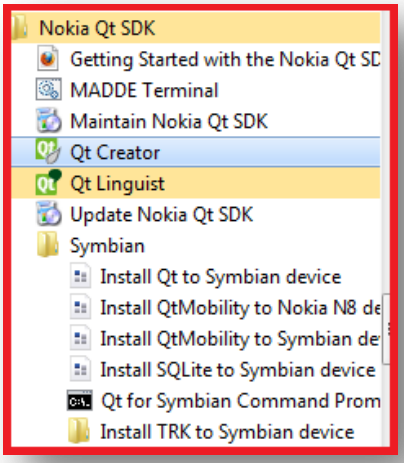
Qt Linguist adalah aplikasi terpisah untuk membuat terjemahan dan lokalisasi ke aplikasi Qt. Dalam Aplikasi Qt memungkinkan untuk beralih bahasa bahkan pada saat aplikasi berjalan.

Qt Linguist digunakan untuk membuat program support Multilanguage. Hampir semua bahasa dapat didukung, tergantung kita, bahasa apa yang akan kita gunakan untuk program kita, Bahasa Indonesia, Nahasa Inggris, Latin, Jepang ataukah Bahasa daerah Anda sendiri. Penterjemahan program masih dilakukan secara manual, yakni memasukkan satu per-satu terjemahan dari property Text dari komponen-komponen visual, misal Menu, Label dan lain sebagainya. Harapannya sih Qt dapat mengintegrasikan dengan Server penterjemah (mis Google Translate) untuk menterjemahkan Program yang kita buat secara otomatis.

Qt Linguist dilengkapi dengan 3 binary utama yakni:

- linguist yang digunakan sebagai editor penterjemahan.
- lupdate untuk membuat File *.ts yakni untuk mengupdate pilihan bahasa yang ditambahkan ke program.
- lrelease untuk membuat file *.qm yakni untuk merelease terjemahan dari program tersebut.

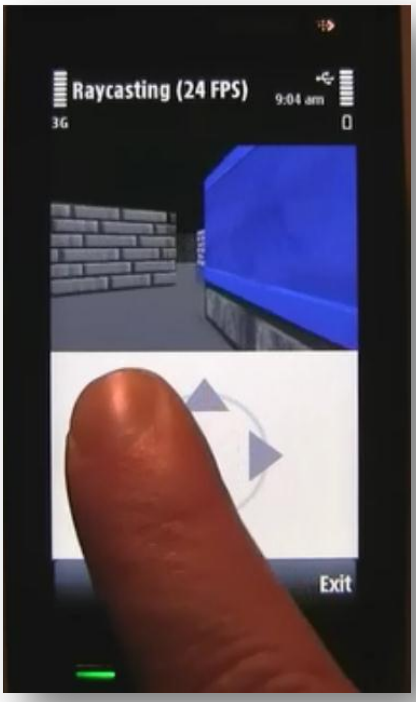
File-file dengan extensi *.ts adalah file XML yang berisi terjemahan bahasa dari text-text yang berada dalam fungsi tr(“”) dalam program kita. sedangkan *.qm adalah terjemahan yang telah siap realese/siap pakai oleh program.


Nokia Qt SDK
Getting Started with the Nokia Qt SD
MADDE Terminal
Maintain Nokia Qt SDK
Qt Creator
Qt Linguist
Update Nokia Qt SDK
Symbian
Install Qt to Symbian device
Install QtMobility to Nokia N8 de
Install QtMobility to Symbian de
Install SQLite to Symbian device
Qt for Symbian Command Prom
Install TRK to Symbian device

## Qt Smart Installer

Smart installer merupakan komponen kecil yang tertanam pada paket aplikasi. Smart installer memudahkan untuk mengembangkan aplikasi Qt untuk ponsel symbian.

Smart installer akan mendownload dan menginstal komponen yang diperlukan selama instalasi, jika diperlukan. File yang dibutuhkan untuk paket aplikasi secara otomatis dibuat oleh qmake.



# Pemrograman Qt Berorientasi Objek

Qt dibuat dengan menggunakan C++ yang pada hakikatnya sudah men-support dengan orientasi objek. Semua library Qt dibangun dengan konsep objek. Sebagai ilustrasi, perhatikan kode program di bawah ini.

```
#include <QApplication>
#include <QLabel>
int maun(int argz, char *argv[])
{
    QApplication app(argc, argv);
    QLabel *myLabel = new QLabel1("Hello Qt!");
    myLabel->show();

    return app.exec()
}
```

Kode program di atas menunjukan akses API Qt berbentuk kelas. Inilah salah satu ciri bahasa pemrograman yang berorientasi objek.

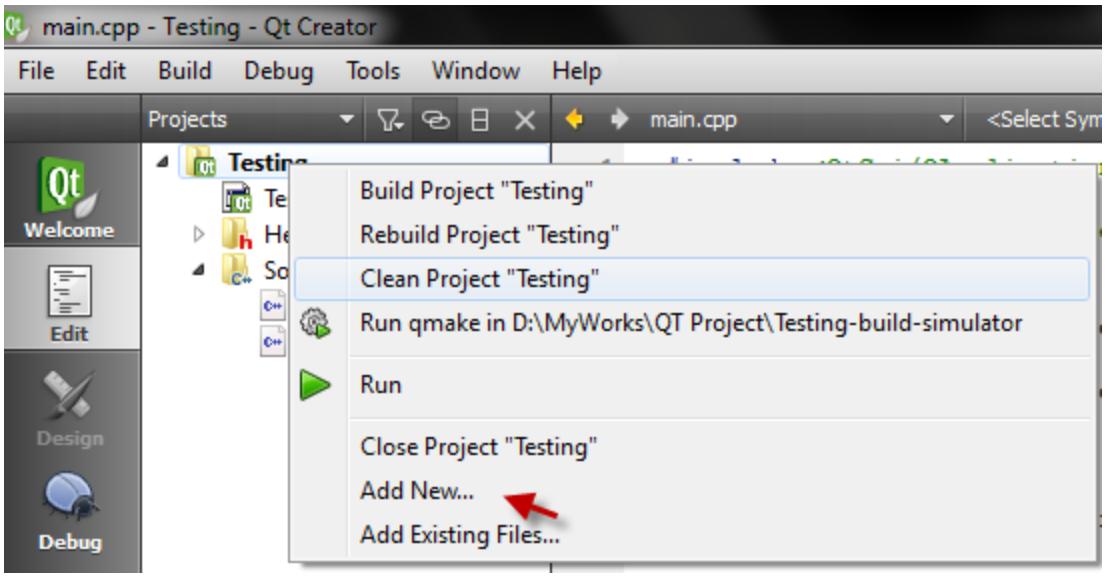
Jika kita merasa kelas tersebut tidak sesuai dengan kebutuhan, kita dapat melakukan perubahan kelakuan dengan melakukan inheritance dari kelas tersebut.

## Deklarasi Kelas

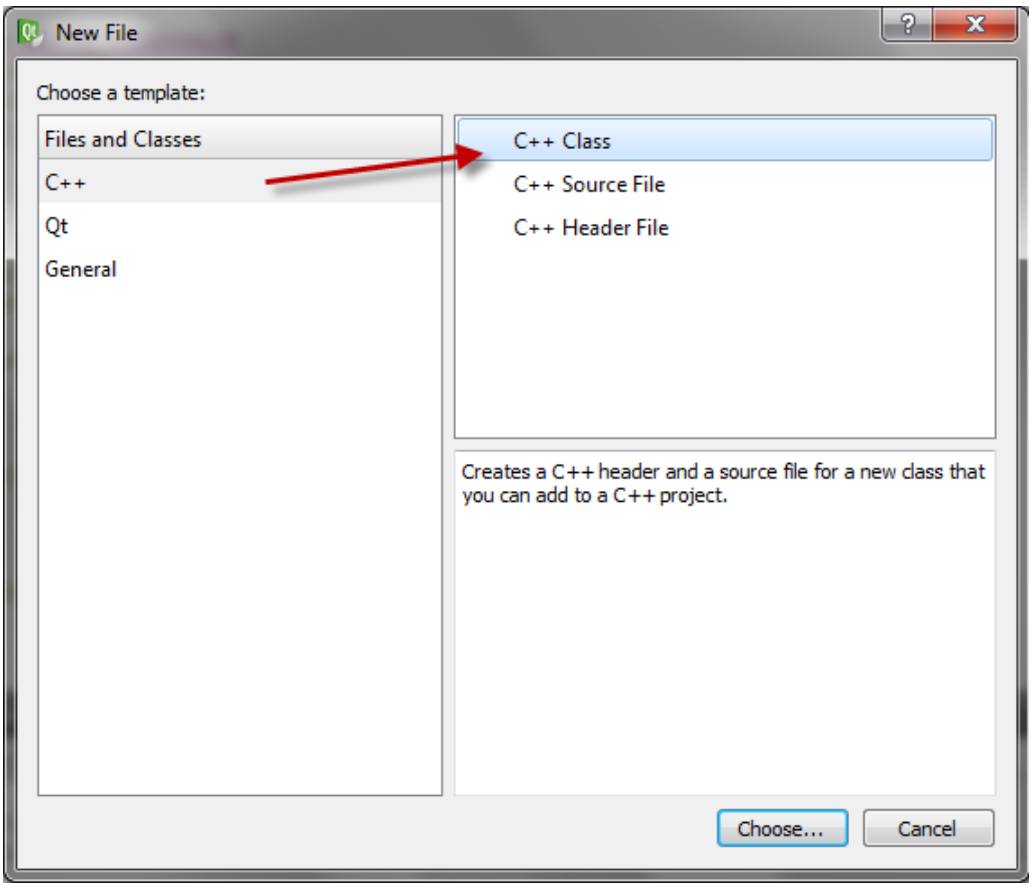
Seperti yang kita ketahui, Qt menggunakan pemrograman C++ sehingga cara mendeklarasikan suatu kelas juga mengikuti kaidah bahasa pemrograman C++. Secara umum, deklarasi suatu kelas seperti kode peogram di bwah ini.

```
class namaKelas {
    members
};
```

Pada Qt Creator kita dapat menambahkan suatu kelas baru baik manual ataupun wizard. Jika ingin menggunakan wizard, kita cukup klik kanan pada project Qt sehingga muncul menu seperti yang terlihat pada gambar di bawah ini. Kemudian pilih menu Add new sehingga memperoleh kotak dialog.




Pilih template C++ dan C++ Class untuk jenisnya. Jika selesai, klik tombol Choose. Selanjutnya, kita mengisi nama kelas, misalkan “MyClass” dan klik tombol Next jika selesai. Kemudian, kotak dialog wizard akan memberikan konfirmasi apa yang telah kita pilih dan isi. Klik tombol finish untuk mengakhiri proses pembuatan class.



Pada proses ini, kita akan memperoleh dua file header dan implementasi. Misalkan myclass.h dan myclass.cpp. sebagai ilustrasinya kita akan membuat suatu fungsi yaitu Foo(). Berikut, realisasi di bagian file header.

```
#ifndef MYCLASS_H
#define MYCLASS_H

class MyClass
{
public:
    MyClass();
public:
    void Foo();
};

#endif // MYCLASS_H
```

Sedangkan realisasi untuk file myclass.cpp sebagai berikut.

```
#include "myclass.h"
#include <iostream>
using namespace std;

MyClass::MyClass()
{
}

void MyClass::Foo()
{
    cout<<"Perform Foo()"<< end1;
}
```

Pada Kode program di atas, kita membuat fungsi Foo() yang isinya hanya menampilkan suatu tulisan pada console.

Setelah kita membuat sebuah kelas, kita akan menggunakan kelas tersebut ke program utama pada file main.cpp. Berikut realisasinya.

```
#include <QtCore/QCoreApplication>
#include <myclass.h>

int main(int argc, char *argv);
{
    QCoreApplication a(argc, argv);

    MyClass obj;
    obj.Foo();

    return a.exec()
}
```

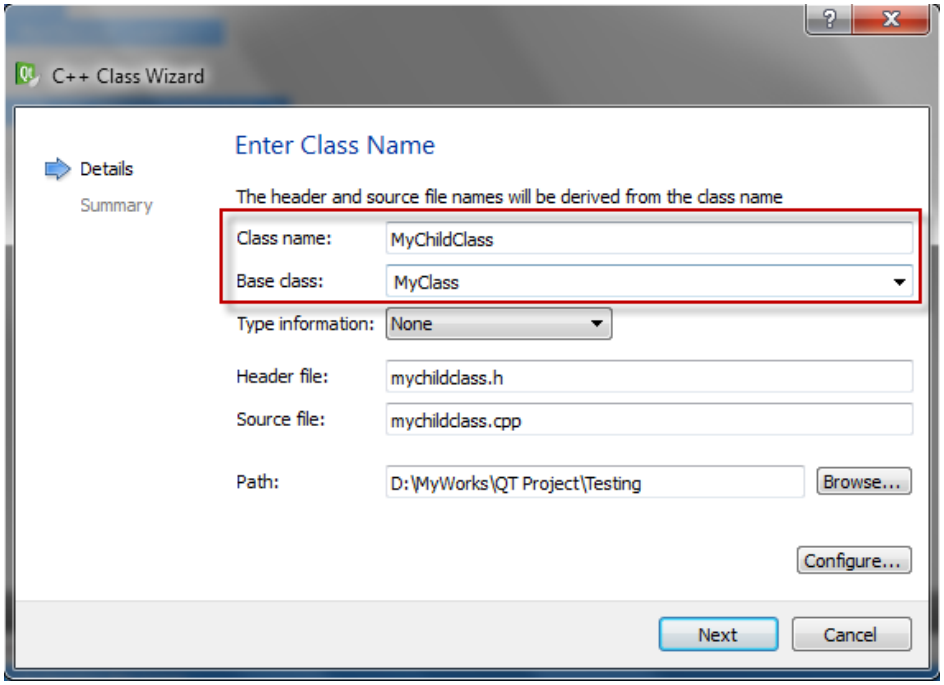
## Inheritance

Ketika suatu kelas mempunyai fitur yang tidak cocok menurut kebutuhan, fitur disini dapat berarti sebagai propertu dan fungsi yang tidak sesuai, maka pada kondisi ini kita dapat menambah atau menggantikan fitur yang sesuai. Hal ini dapat dilakukan dengan memanfaatkan apa yang dikenal dengan inheritance.

Sebelumnya, kita sudah membuat suatu kelas dengan nama MyClass. Apabila kita ingin melakukan inheritance dengan nama kelas inheritance "MyChildClass", deklarasi kelas "MyChildClass" ini sebagai berikut.

```
#ifndef MYCHILDCLASS_H
#define MYCHILDCLASS_H
class MyChildClass : public MyClass
{
public:
    MyChildClass();
};
#endif // MYCHILDCLASS_H
```

Jika Anda menggunakan wizard untuk membuat kelas, kita dapat langsung memberikan nama Base Class nya.




Cara penggunaannya sama seperti cara penggunaan suatu kelas.

## Interface

Salah satu fitur pemrograman berorientasi objek adalah interface. Ketika kita menginginkan suatu objek wajib mempunyai suatu tingkah laku baik variable yang tersedia (property) ataupun fungsinya. Kita dapat memanfaatkan interface.

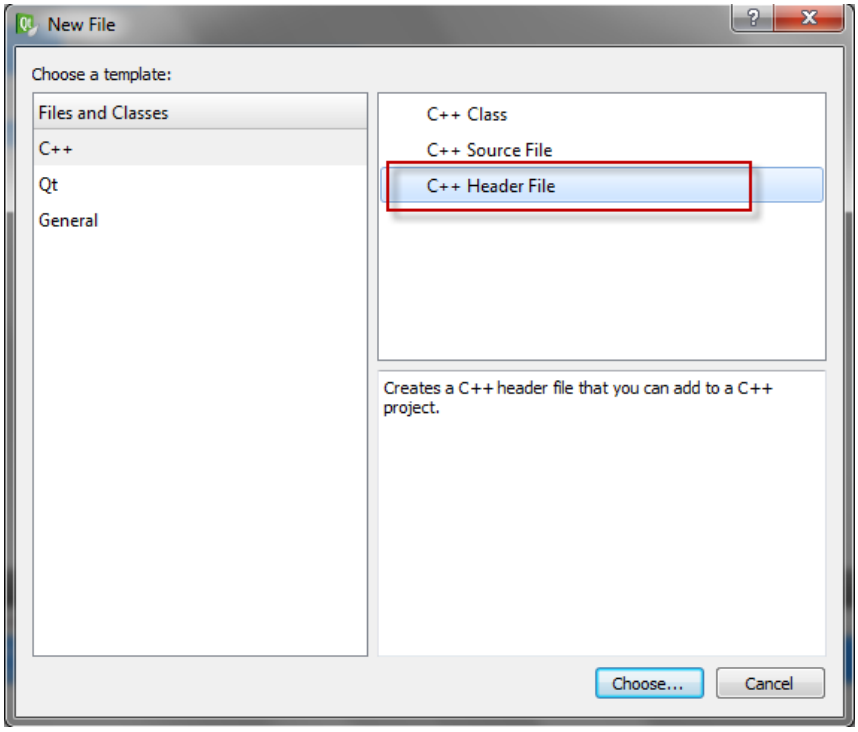
Pada bahasa C++, khususnya Qt, kita dapat mendeklarasikan suatu interface dengan memanfaatkan fitur virtual.

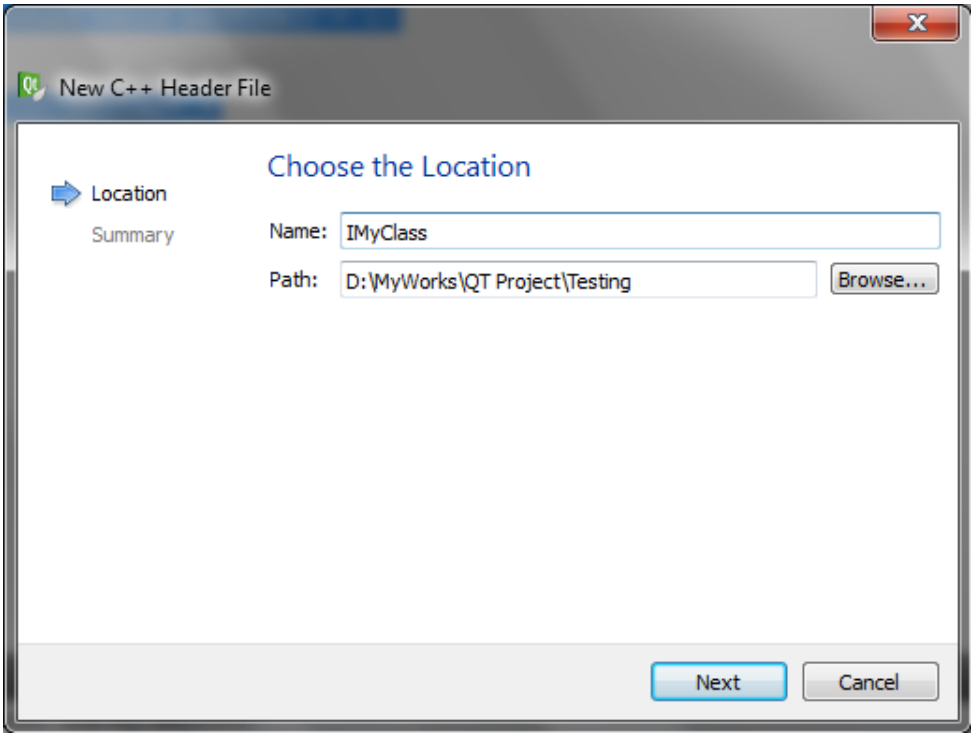
Berikut ini contoh deklarasi sebuah interface.

```
#ifndef IMYCLASS_H
#define IMYCLASS_H
class IMyClass
{
public:
    virtual vaoid perform() = 0;
};
#endif // IMYCLASS_H
```

Pada kode program di atas, kita membuat sebuah interface yang direalisasikan pada sebuah kelas yaitu IMyClass dan terdapat satu fungsi virtual yaitu Perform().

Pada umumnya, deklarasi sebuah kelas interface disimpan pada file header. Misal, untuk kasus ini adalah file imyclass.h. pada Qt Creator kita cukup menambahkan item baru yaitu “C++ Header” sehingga kita akan memperoleh kotak dialog seperti berikut ini.




Untuk implementasi interface, kita dapat memanfaatkan konsep inheritance di mana kelas tersebut langsung di inheritance ke kelas interface kita. Contoh realisasinya, kita membuat kelas MyClass dan contoh implementasi pada file header, misal nama filenya myclass.h, sebagai berikut.

```
#ifndef MYCLASS_H
#define MYCLASS_H
#include <IMyClass.h>

class MyClass : public IMyClass
{
public:
    MyClass();
    void Perform();
};
#endif // MYCLASS_H
```

Pada kode program di atas, kelas MyClass di inheritance kelas interface IMyClass. Kemudian, kita membuat public fungsi Perform(). Untuk realisasinya, kelas MyClass pada file myclass.cpp sebagai berikut.

```
#include "myclass.h"
#include <iostream>
using namespace std;
MyClass::MyClass()
{
}
void MyClass::Perform()
{
    cout <<"Do perform"<< end1;
}
```

Pada kode program di atas, saya memberikan contoh sederhana untuk menampilkan tulisan pada console.

Bagaimana menggunakan interface? Caranya cukup mudah. Perhatikan kode program untuk penggunaan interface di bawah ini.

```
#include <QtCore/QCoreApplication>
#include <myclass.h>
#include <IMyClass.h>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    IMyClass* obj = new MyClass();
    obj->perform();
    return a.exec();
}
```

Pada kode program diatas terdapat baris kode:

```
IMyClass* obj = new MyClass();
```

Ini menunjukan obj akan mempunyai instansi objek MyClass. Oleh karena itu, ketika kita menjalankan fungsi Perform(), fungsi ini akan mengeksekusi fungsi Perform() pada kelas MyClass.

## Copy Construction

Kadang kala kita ingin menduplikasi nilai suatu objek ke objek yang baru, misal objek Employee yang mempunyai id dan name pada objeknya. Perhatikan kode program di bawah ini.

```
#include <QString>
class Employee
{
public:
    Employee();
    Employee(int id,QString nm);
    Employee(const Employee& newEmp);
private:
    int id;
    QString name;
};
```

Selanjutnya, jika obj1 bertipe Employee dan sudah diisi nilai id dan name, kita dapat meng-copy ke obj2 baru melalui copy construction. Implementasinya sebagai berikut.

```
#include <employee.h>
Employee::Employee(){

}
Employee::Employee(int id,QString nm)
    :id(id),name(nm){

}
Employee::Emoloyee(const Employee& newEmp)
    :id(newEmp.id), name(newEmp.name){

}
```

Untuk realisasinya pada kode program sebagai berikut.

```
Employee obj1(10,"Iqbal");
Employee obj2(obj1);
```

## Manipulasi String dengan QString

Dalam bahasa C dan C++, sebuah string direpresentasikan sebagai array dari char dengan karakter 0 atau null sebagai pertanda akhir string. Yang semacam ini sering kemudian disebut sebagai null-terminated

string. Pustaka standar C juga menyediakan sejumlah fungsi untuk melakukan manipulasi string, bisa dilihat di file header *string.h*.

Problem utama penggunaan string sebagai array char adalah keterbatasan dalam fleksibilitas penanganan memori. Hal ini karena string sering dialokasikan secara dinamik, sebagaimana array, sehingga membutuhkan pula proses dealokasi memori. Bila tidak dilakukan dengan tepat, alokasi/dealokasi ini bisa menyebabkan si programer sakit kepala mengingat kesalahan sedikit saja bisa berakibat fatal: segmentation fault karena salah dealokasi atau boros penggunaan memori (memory leak) karena lupa melakukan dealokasi. Belum lagi masalah dereferensi pointer yang harus dilakukan dengan hati-hati.

Masalah lain untuk string muncul ketika program yang dikembangkan harus berurusan dengan karakter yang bukan latin, misalnya huruf-huruf Kanji dari Jepang. Mengingat string adalah array dari char dan char hanya 8 bit, maksimal kombinasi karakter yang ditampung adalah 256 karakter. Hampir pada semua kasus, ke-256 karakter ini akan diisi dengan karakter ASCII. Lantas, bagaimana dengan orang-orang Rusia atau Urdu yang juga tidak mau ketinggalan bermain program komputer ? Bagaimana mereka harus merepresentasikan semua huruf-huruf yang non-latin ?

Solusi dari masalah ini adalah penggunaan Unicode, yaitu sebuah standar untuk menyimpang karakter atau string secara lebih baik sehingga sanggup menampung semua aneka karakter dari semua bahasa-bahasa di dunia. Triknya sederhana: memperluas kemungkinan kombinasi karakter dari 256 menjadi 65536 dengan mengubah representasi satu karakter menjadi 16-bit (tidak lagi 8-bit). Lebih jauh tentang Unicode, berikut pemetaan karakter-karakternya, bisa dilihat di www.unicode.org.

Sayang sekali Unicode belum diadopsi sebagai standar untuk digunakan di bahasa C ataupun C++. Untuk mengatasi hal ini, Qt menyediakan kelas QString, sebuah kelas yang dapat dipergunakan untuk menyimpan dan mengolah string dan sudah mendukung Unicode. Kalau string adalah array dari char, maka QString adalah array dari QChar, sebuah kelas yang mewakili satu karakter Unicode. Kelas QString ini juga dilengkapi sekian puluh fungsi-fungsi yang sangat berfaedah untuk memanipulasi string dan pastinya akan menjadi 'teman baik' seorang programmer (Tidak percaya ? Sembari maupun setelah membaca tulisan ini silakan merujuk ke referensi Qt tentang QString, mudah dipahami bahwa fungsionalitas yang disediakan QString memang lengkap dan yang betul-betul dibutuhkan).

Bagaiamana membuat string dengan QString ? Mudah sekali. Bisa dilakukan misalnya sebagai berikut:

```
QString s = "Hello"
```

Atau:

```
QString s("Hi there")
```

Karena QString bisa melakukan konversi secara otomatis, maka beralih dari string (atau `char*`) bisa dilakukan semudah:

```
char msg[] = "Hello Qt!";
QString s = msg;
```

Sebaliknya, casting dari QString ke char* juga dimungkinkan. Lihat yang berikut ini.

```
QString s = "hi";
printf( "%s\n", (const char*) s );
```

Ataupun:

```
cout << s;
```

Pun mengalihkan objek QString ke stream tidak akan menimbulkan masalah:

```
QString ask = "Who are you?";
cout << ask << endl;
```

Fungsi `uint QString::length` akan mengembalikan panjang string (mirip dengan `strlen`). Sebagai contoh, bila str adalah "Linux", maka `str.length()` menghasilkan 5.

Fungsi `bool QString::isNull()` dan `bool QString::isEmpty()` digunakan untuk memeriksa apakah string NULL dan apakah string tersebut kosong. Perhatikan di sini bahwa empty != null. Yang dimaksud dengan empty string adalah string yang tidak berisi apa-apa (`length()` akan menghasilkan 0). Sementara itu null string bisa dianalogikan dengan NULL pada pointer. Terdapat juga fungsi statik `QString::null` yang akan mengembalikan null string. Lazimnya, dua fungsi ini digunakan pada pemeriksaan kondisi tertentu. Potongan kode berikut mengilustrasikannya:

```
// do nothing if null
if( s.isNull() ) return;

// empty or not ?
if( s.isEmpty() ) cout << "string is empty" << endl;

cout <<Your string is << s << endl;
```

Catatan: pada kode di atas, alih-alih `s.isNull()`, bisa juga digunakan `s == QString::null`. Bagaimana mengambil sebagian string ? Ada fungsi `QString QString::left( uint len )`, `QString QString::right( uint len )`, dan `QString QString::mid( uint index, uint len )` yang masing-masing akan menghasilkan sebuah string baru yang

```
QString s = "Linus Torvalds" ;
QString p = s.left( 4 ) ;        // Linu
QString q = s.right( 3 );        // lds
QString r = s.mid( 6, 3 );       // Tor
```

merupakan bagian tertentu dari string yang dipanggil. Contoh berikut menunjukkan bagaimana hasil pemanggilan ketiga fungsi ini:

Kalau begitu, sekarang bagaimana menggabungkan string ? Dibandingkan pustaka standar C yang hanya menyediakan `strcat`, Qt memberikan fungsi `QString& QString::append( const QString& s )` dan `QString& QString::prepend( const QString& s )` yang akan membubuhkan string `s` berturut-turut ke akhir dan awal string.

```
QString name = "Maddog" ;
name.append( "Hall" );      // Maddog Hall
name.prepend( "John" );     // John Maddog Hall
```

Ada pula `QString& QString::insert( uint index, const QString& s )` yang akan menyisipkan string s pada posisi index. Lihat contoh ini:

```
QString jfk = "John Kennedy" ;
cout << jfk.insert( 5,"F. " ) << endl; // John F. Kennedy
```

Dokumentasi Qt yang rinci dan lengkap menerangkan lebih banyak lagi mengenai kelas QString dan QChar ini.

## Tipe Data

Pada dasarnya Qt memiliki tipe data sama seperti C++ yaitu:

- int
- Bool
- Double
- Char

Semua jenis tipe data didefinisikan dalam header `<QtGlobal>`, berikut adalah table tipe data lintas platform.

| Type | Size | Minimum value | Maximum value |
|---|---|---|---|
| uint8 | 1 byte | 0 | 255 |
| uint16 | 2 bytes | 0 | 65 535 |
| uint32 | 4 bytes | 0 | 4 294 967 295 |
| uint64 | 8 bytes | 0 | 18 446 744 073 709 551 615 |
| int8 | 1 byte | -128 | 127 |
| int16 | 2 bytes | -32 768 | 32 767 |
| int32 | 4 bytes | -2 147 483 648 | 2 147 483 647 |
| int64 | 8 bytes | -9 223 372 036 854 775 808 | 9 223 372 036 854 775 807 |
| quintptr | “pointer sized” | n/a | n/a |
| qptrdiff | “pointer sized” | n/a | n/a |
| qreal | fast real values | n/a | n/a |

Qt dilengkapi dengan berbagai kelas yang kompleks dan beragam untuk menyederhanakan suatu tipe data yang mendasar. Sebagai contoh :

`QFont`, `QColor`, `QString`, `QList`, `QPoint`, `QImage`, dan lain sebagainya.

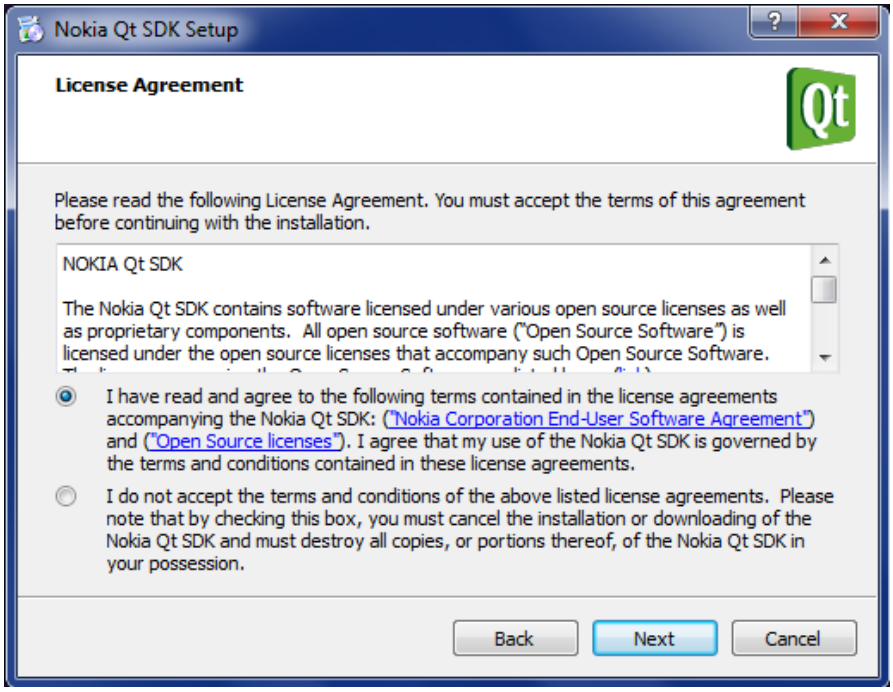
Dan kita juga bisa membuat tipe data sendiri sesuai dengan kebutuhan.

# Instalasi Nokia Qt SDK

Pada proses instalasi ini, saya menggunakan Nokia Qt SDK untuk OS Windows. Berikut ini langkah-langkahnya:

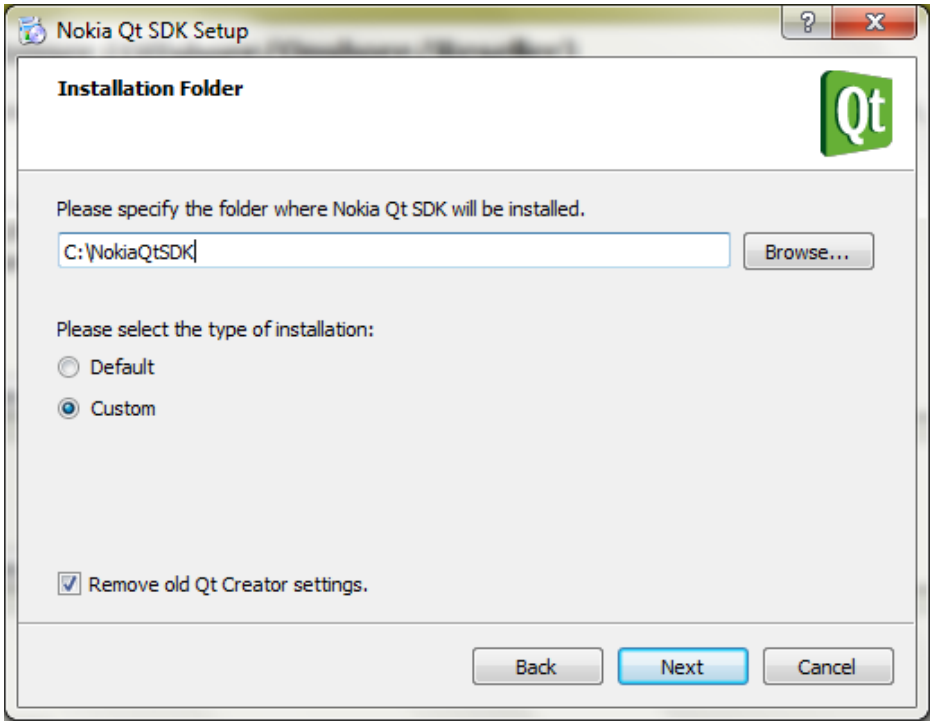
1. Klik dua kali file installer Nokia Qt SDK yang telah Anda unduh. Kemudian akan memperoleh kotak dialog seperti gambar di bawah. Klik tombol *Next* untuk memulai Instalasi.


2. Kemudian Akan memperoleh kotak dialog seperti gambar berikut:

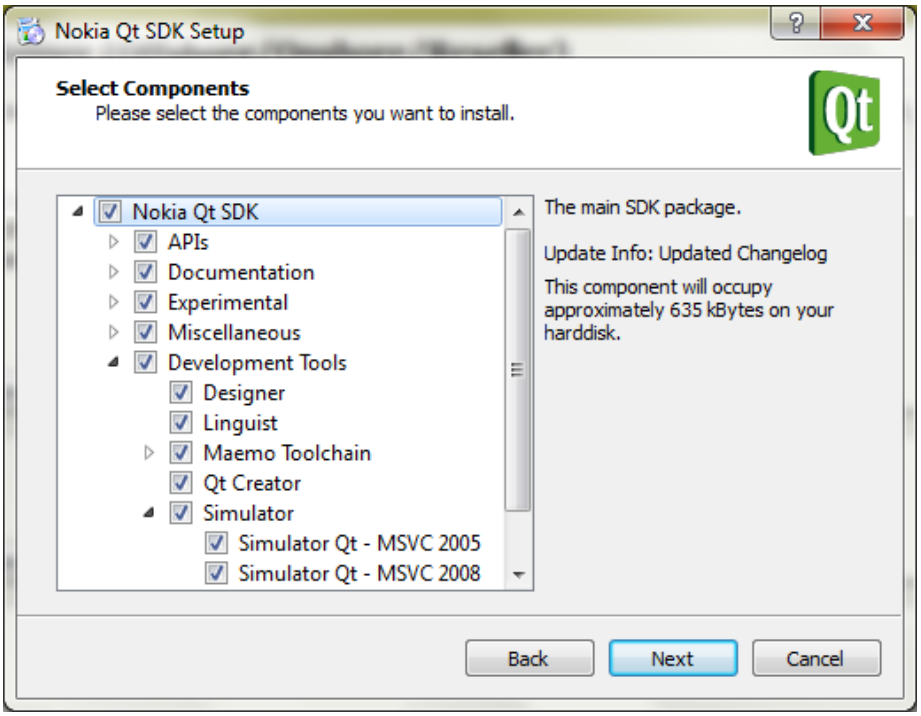



3. Pilih opsi "*I have read and agree to the following terms…*" Klik tombol *Next*.

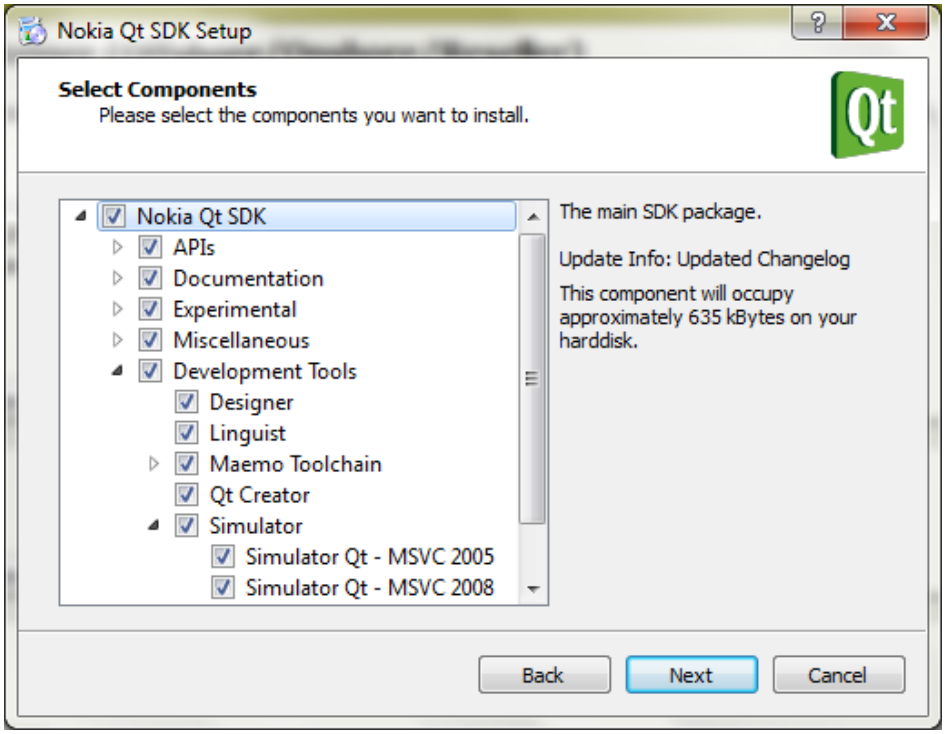
4. Selanjutnya Anda akan memperoleh kotak dialog seperti gambar dibawah ini. Pilih folder tempat Nokia Qt SDK akan di-instal.



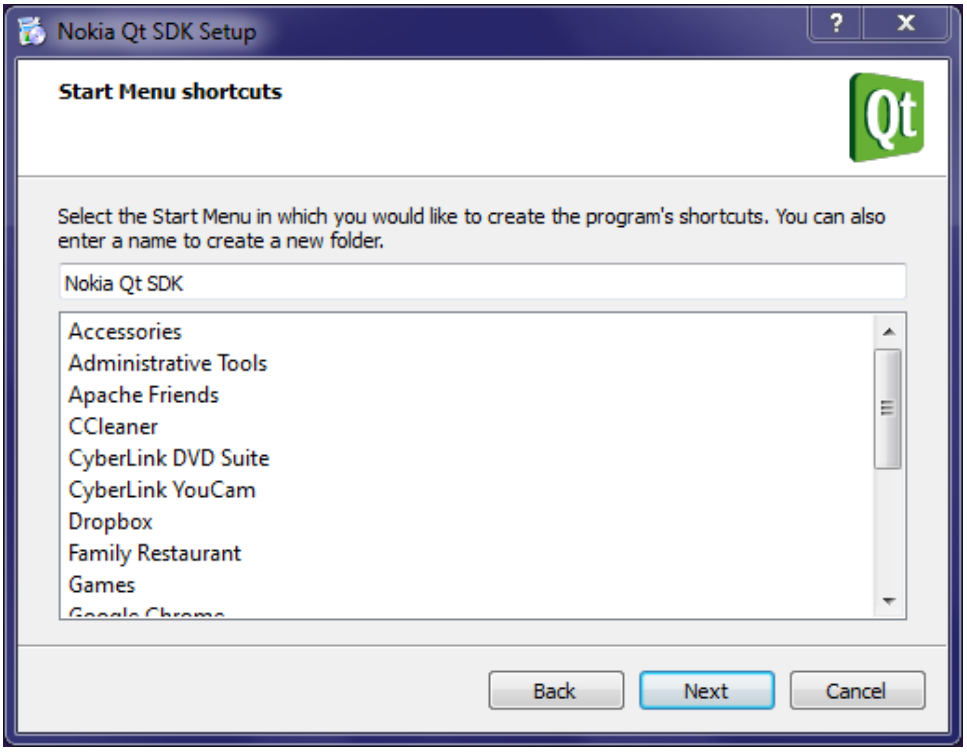
5. Pilih juga *Custom* untuk tipe instalasi aga Anda dapat memilih module apa saja yang akan di-instal. Klik *Next*.



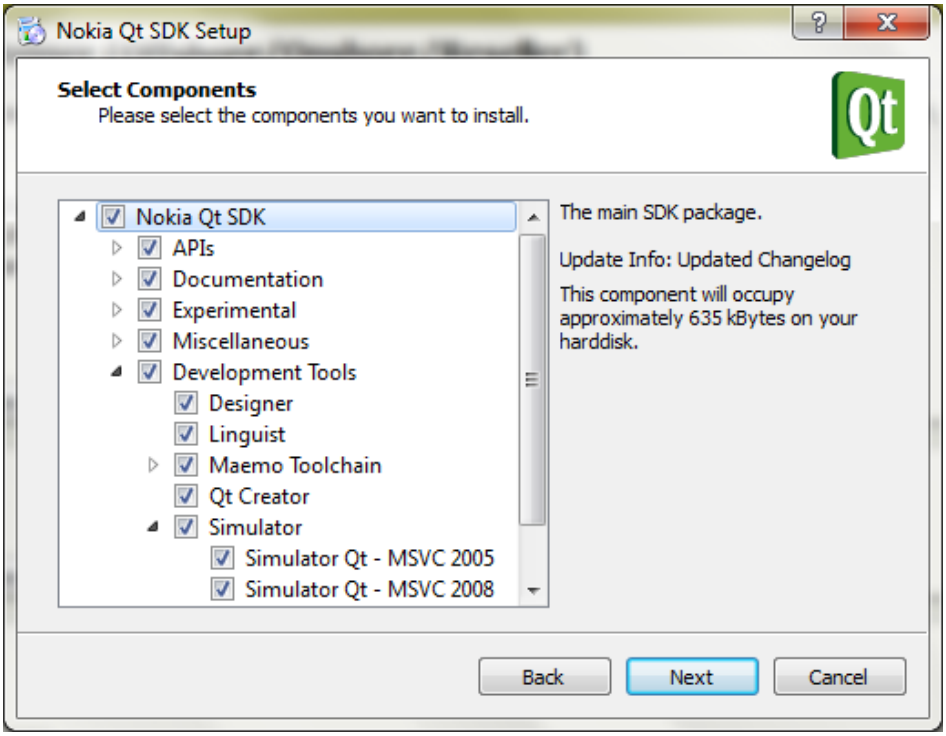

6. Pilih module yang akan di-instal. Direkomendasikan utuk install semuanya. Klik *Next*.



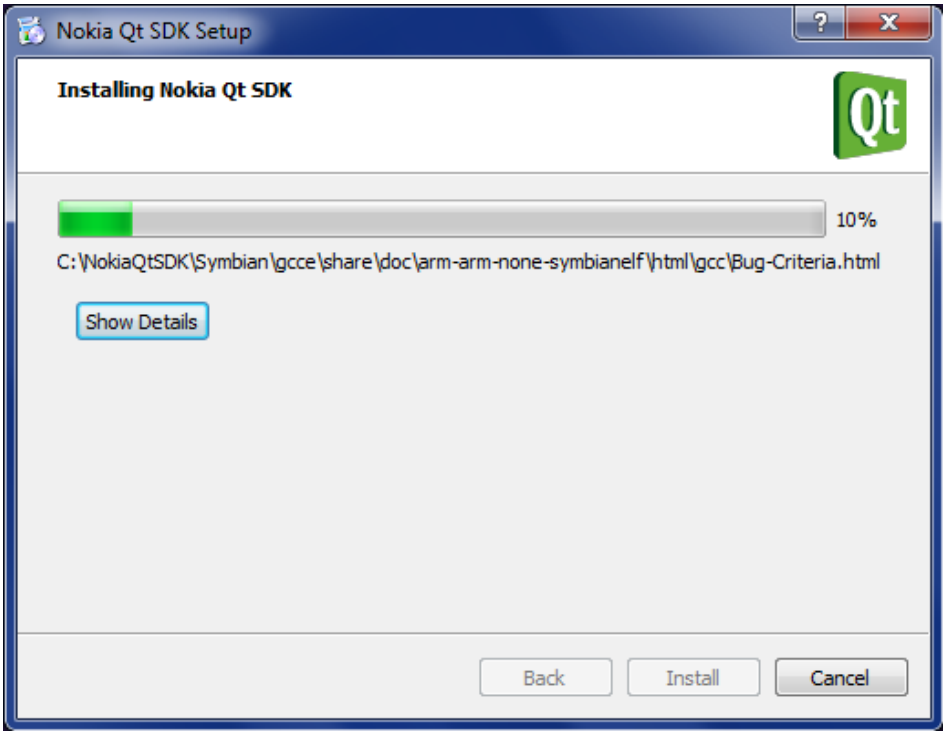
7. Isi nama menu shortcut sesuai keinginan. Klik tombol *Next* untuk melanjutkan.



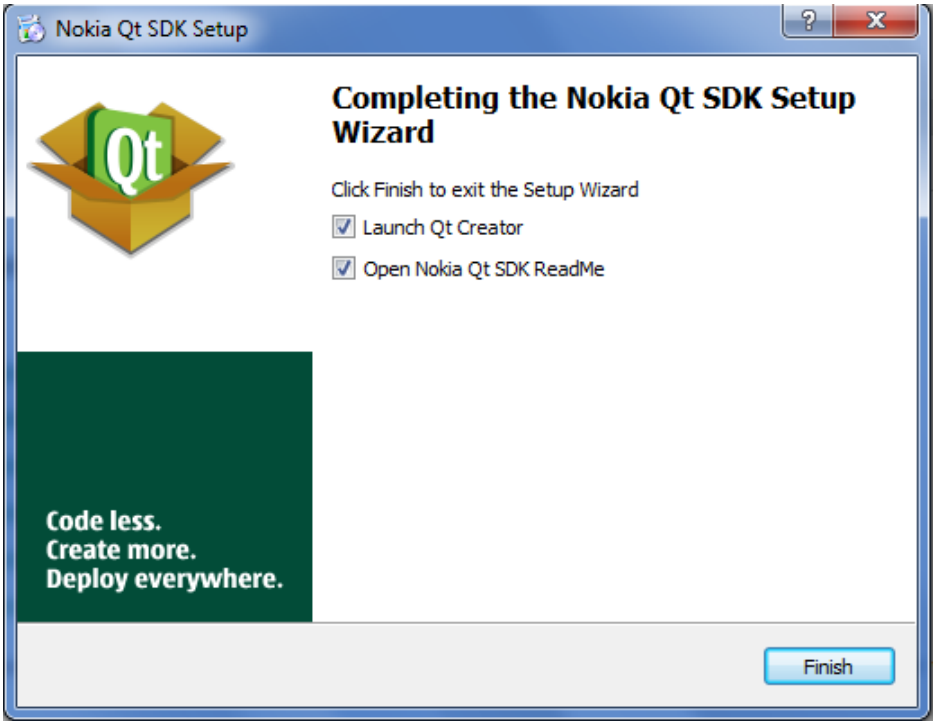

8. Kemudian klik tombol *Instal* untuk memulai proses instalasi.



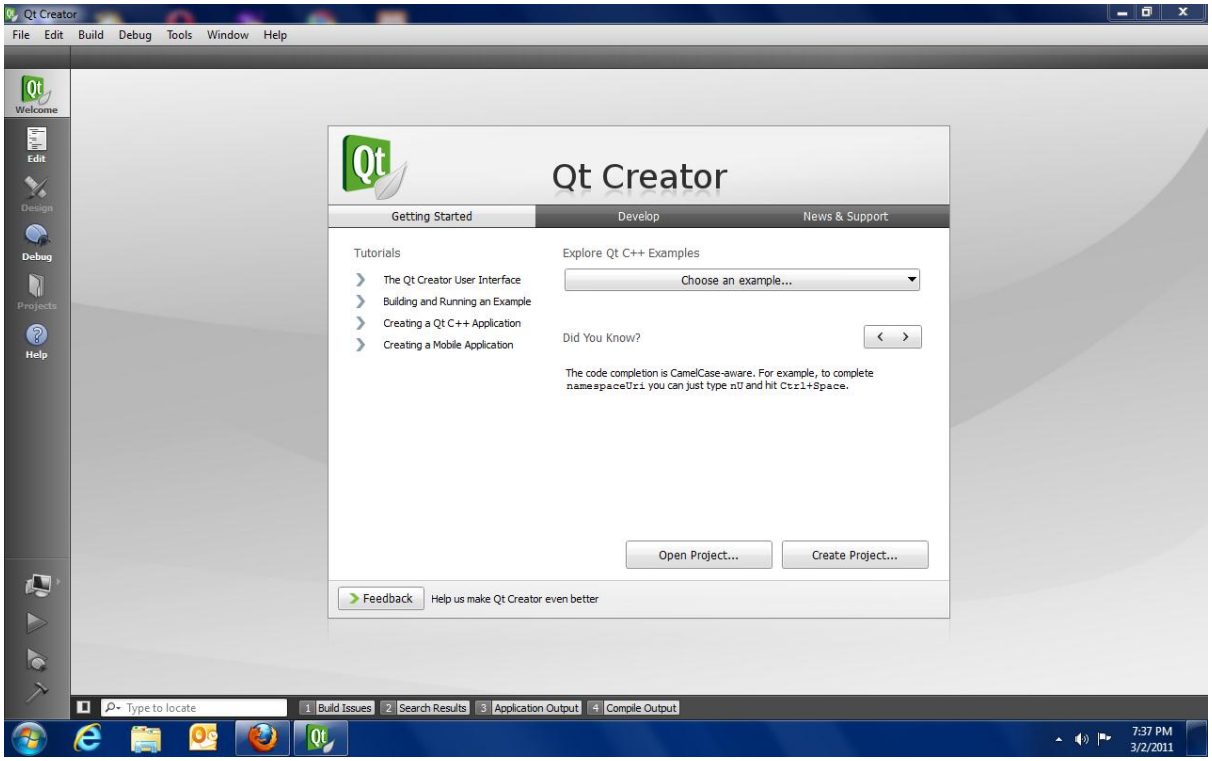
9. Sistem akan melakukan proses instalasi sesuai dengan module yang dipilih.




10. Jika proses instalasi selesai. Klik tombol *Finish*.



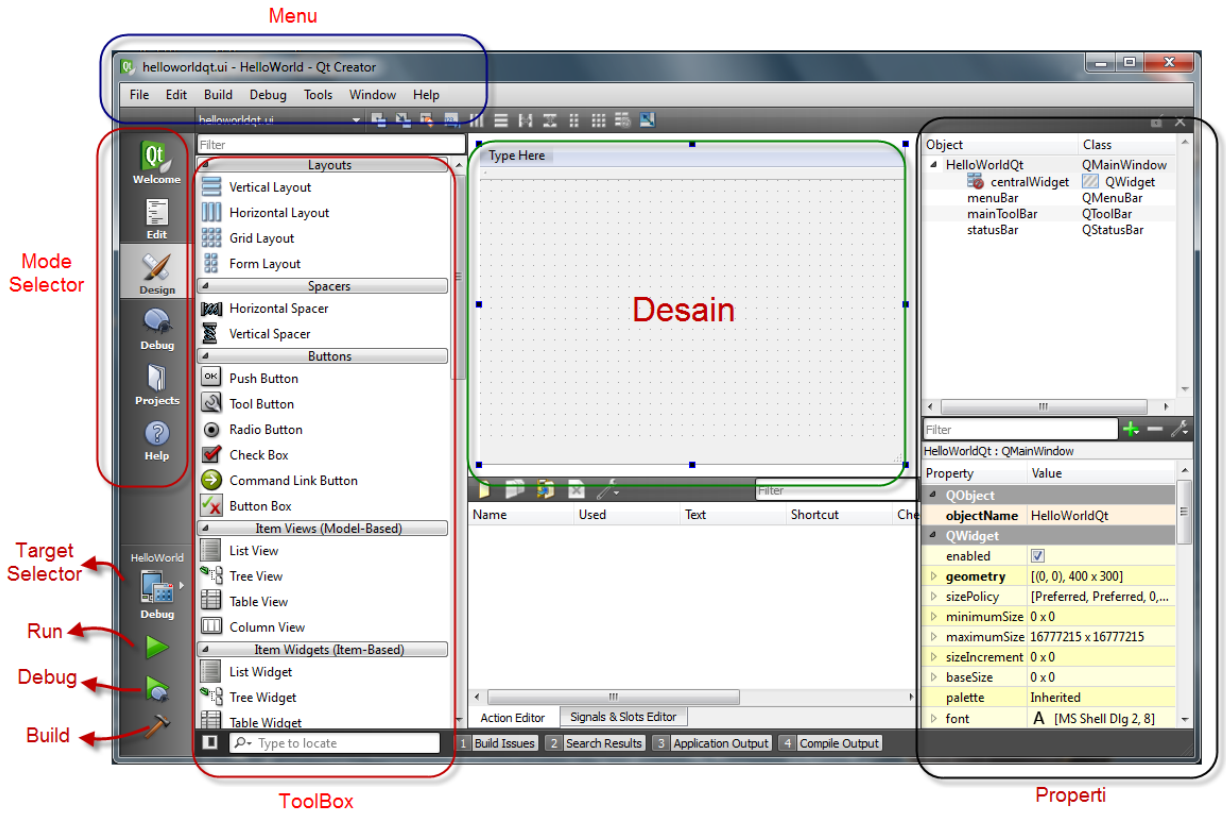
11. Nokia Qt SDK Anda siap digunakan. ☺




Gambar di atas adalah antar muka dari Qt Creator. Dengan Qt Creator kita dapat membuat aplikasi Nokia Mobile atau Symbian OS. Qt Creator juga menawarkan kemudahan bagi programmer dalam melakukan debugging.

# Antar Muka Qt Creator

Bagi yang sudah cukup mengenal Visual Studio atau Netbeans, maka sudah tidak asing dengan antar muka disertai panel Desain, ToolBox, dan Properties yang berada disekitar layar. Bagi Anda yang masih asing dengan antar muka Qt Creator, mari kita mengenal fungsi dari masing-masing tombol dan navigasi pada Qt Creator.




Pemilih mode memungkinkan Anda untuk beralih cepat antara tugas-tugas seperti mengedit proyek dan source file, merancang aplikasi UIs, mengkonfigurasi bagaimana proyek dibangun dan dijalankan, dan debugging aplikasi Anda. Untuk mengubah mode, klik ikon, atau menggunakan cara pintas keyboard yang sesuai.

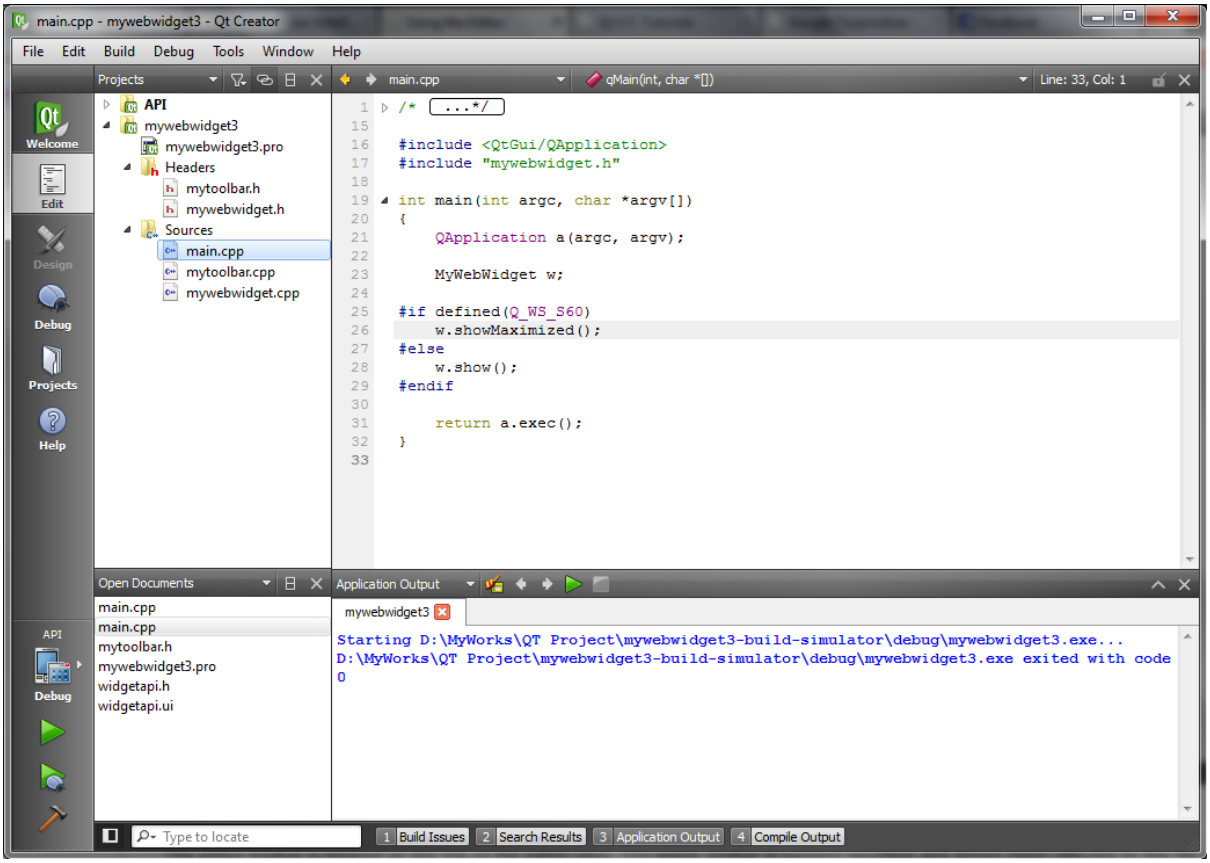
Anda dapat menggunakan Qt Creator dalam mode berikut:

- **Mode edit** untuk mengedit proyek dan file source.

- **Mode Desain** untuk merancang dan mengembangkan aplikasi user interface. Mode ini tersedia untuk file UI (.ui atau qml.).
- **Mode Debug** untuk memeriksa keadaan program Anda saat debugging.
- **Mode Proyek** untuk mengkonfigurasi dan membangun proyek dan eksekusi. Mode ini tersedia ketika sebuah proyek terbuka.
- **Bantuan** modus untuk melihat dokumentasi Qt.

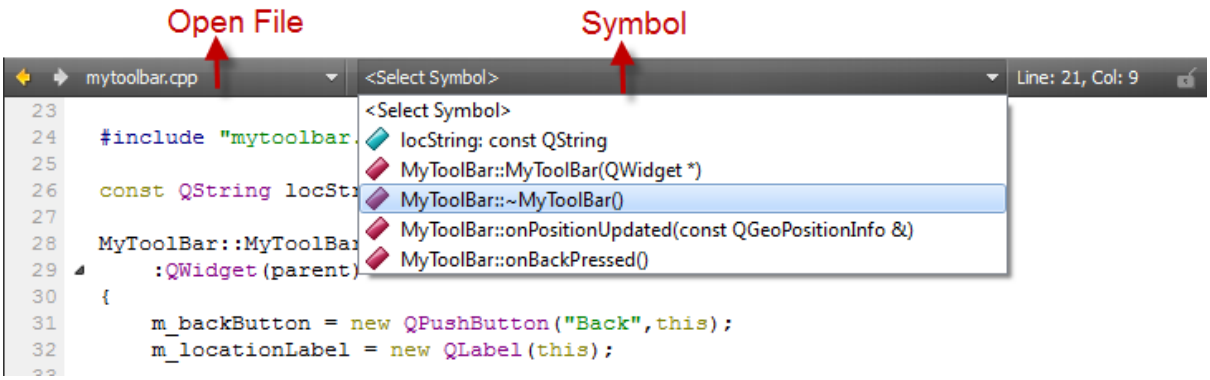
## Modus Edit

Code editor Qt Creator dirancang untuk membantu Anda dalam membuat, mengedit dan navigasi kode. Code editor Qt Creator dilengkapi dengan syntax checking, code completion, dan indikator kesalahan dalam-baris kode saat Anda mengetik.




## Toolbar Editor

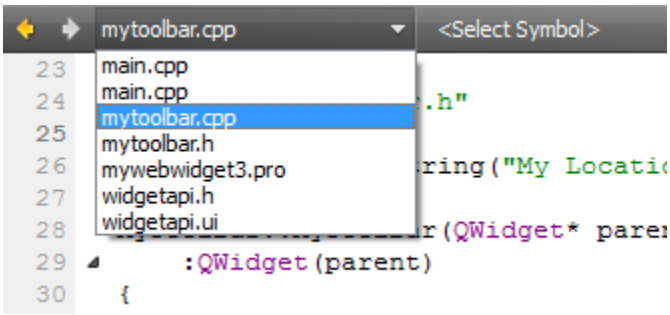
Toolbar editor terletak di bagian atas tampilan editor. Toolbar editor konteks sensitif yang menunjukkan item yang relevan dengan file yang sedang terbuka di editor.




Untuk melompat ke simbol yang digunakan dalam file saat ini, pilih dari menu drop-down Simbol. Secara default, simbol akan ditampilkan dalam urutan di mana mereka muncul dalam file. Klik kanan judul menu dan pilih *Sort Alphabetically* untuk mengatur simbol dalam urutan abjad.

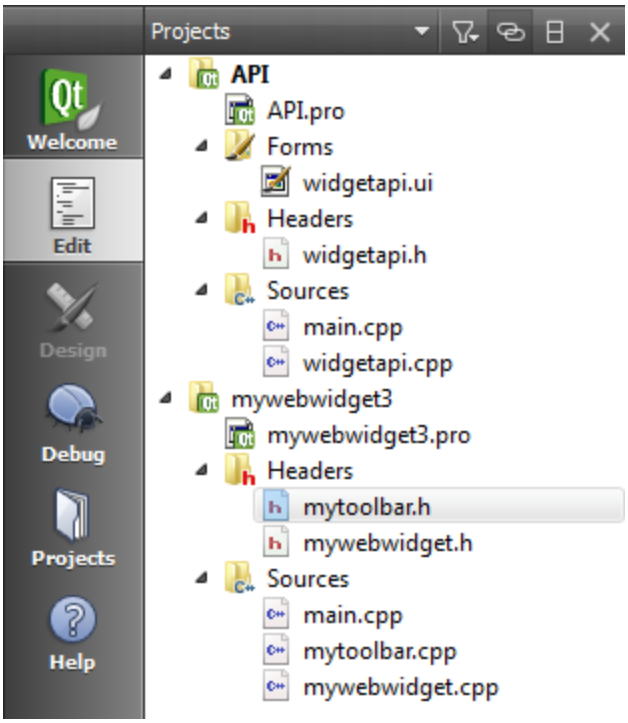
Gunakan toolbar untuk menavigasi antara file untuk membuka file proyek lainnya dengan mengklik tombol navigasi  dan  .

Untuk menuju ke file terbuka, pilih Open File dari menu drop-down. Anda juga bisa meng-Klik kanan judul menu dan pilih *Copy Full Path to Clipboard* untuk menyalin path dan nama file saat ini ke clipboard.




## Browsing Project Contents

Pada mode Edit dan mode Debug, gunakan sidebar untuk menelusuri proyek-proyek, file, dan bookmark, dan untuk melihat hirarki kelas.




Anda dapat memilih isi sidebar di menu sidebar:

- **Projects** menunjukkan daftar proyek terbuka di sesi saat ini.
- **Open Documents** menunjukkan file yang sedang terbuka.
- **Bookmarks** menunjukkan semua bookmark untuk sesi saat ini.
- **File System** menampilkan semua file dalam direktori yang sedang dipilih.
- **Class View** menunjukkan hirarki kelas dari proyek yang saat ini terbuka.
- **Outline** menunjukkan hirarki elemen dari sebuah file QML.
- **Type Hierarchy** menunjukkan kelas dasar dari sebuah kelas.

Anda dapat mengubah tampilan sidebar dengan cara berikut:

- Untuk mengaktifkan sidebar, klik atau tekan Alt +0 (Cmd +0 di Mac OS X).
- Untuk split sidebar, klik . Memilih konten baru untuk melihat pada tampilan split.
- Untuk menutup tampilan sidebar, klik .

## Build Issues / Debugger

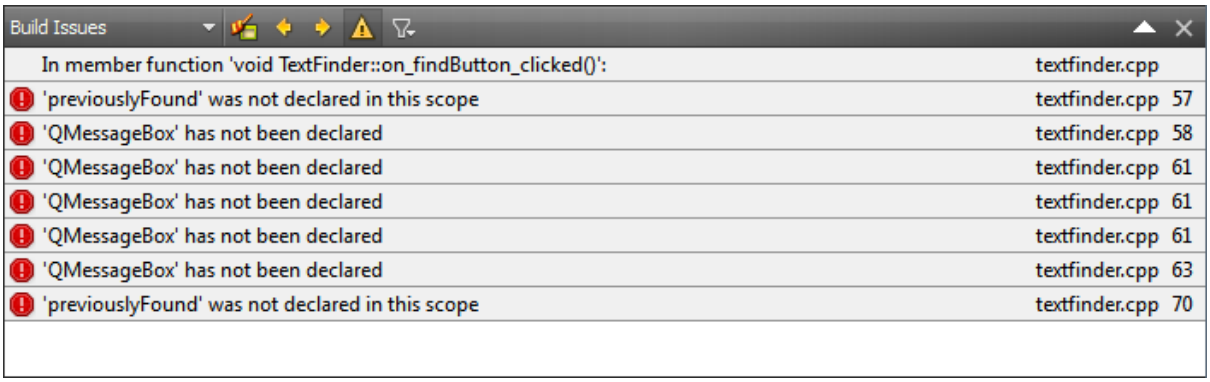
Qt Creator menyediakan debugger plugin yang bertindak sebagai antarmuka antara inti Qt Creator dan external native debugger seperti GNU Symbolic Debugger (gdb), Microsoft Console Debugger (CDB), dan QML / JavaScript debugger.

Bagian berikut menjelaskan debugging dengan Qt Creator:

- **Debugging Contoh Aplikasi,** menggunakan aplikasi contoh untuk menggambarkan bagaimana debug Qt C + + aplikasi dalam modus debug.
- **Berinteraksi dengan Debugger,** menggambarkan pandangan dan fungsi yang tersedia dalam modus debug.
- **Menyiapkan Debugger,** merangkum dukungan untuk debugging C + + kode dan persyaratan untuk instalasi. Biasanya, interaksi antara Qt Creator dan native debugger sudah diatur secara otomatis dan Anda tidak perlu melakukan apapun.
- **Peluncuran Debugger dalam Mode yang berbeda-beda,** menjelaskan modus operasi di mana plugin debugger berjalan, tergantung di mana dan bagaimana proses ini dimulai dan dijalankan.
- **Menggunakan Debugging Helpers,** menjelaskan cara untuk mendapatkan lebih banyak data rinci pada data yang kompleks.
- **Debugging Proyek Qt Quick** menjelaskan cara debug proyek Qt Quick.
- **Mendeteksi Kebocoran Memori**, menjelaskan cara menggunakan alat Valgrind Memcheck untuk mendeteksi masalah dalam manajemen memori.
- **Mengatasi Masalah Debugger**, daftar beberapa masalah yang khas yang mungkin Anda hadapi saat debugging dan memberikan solusi.

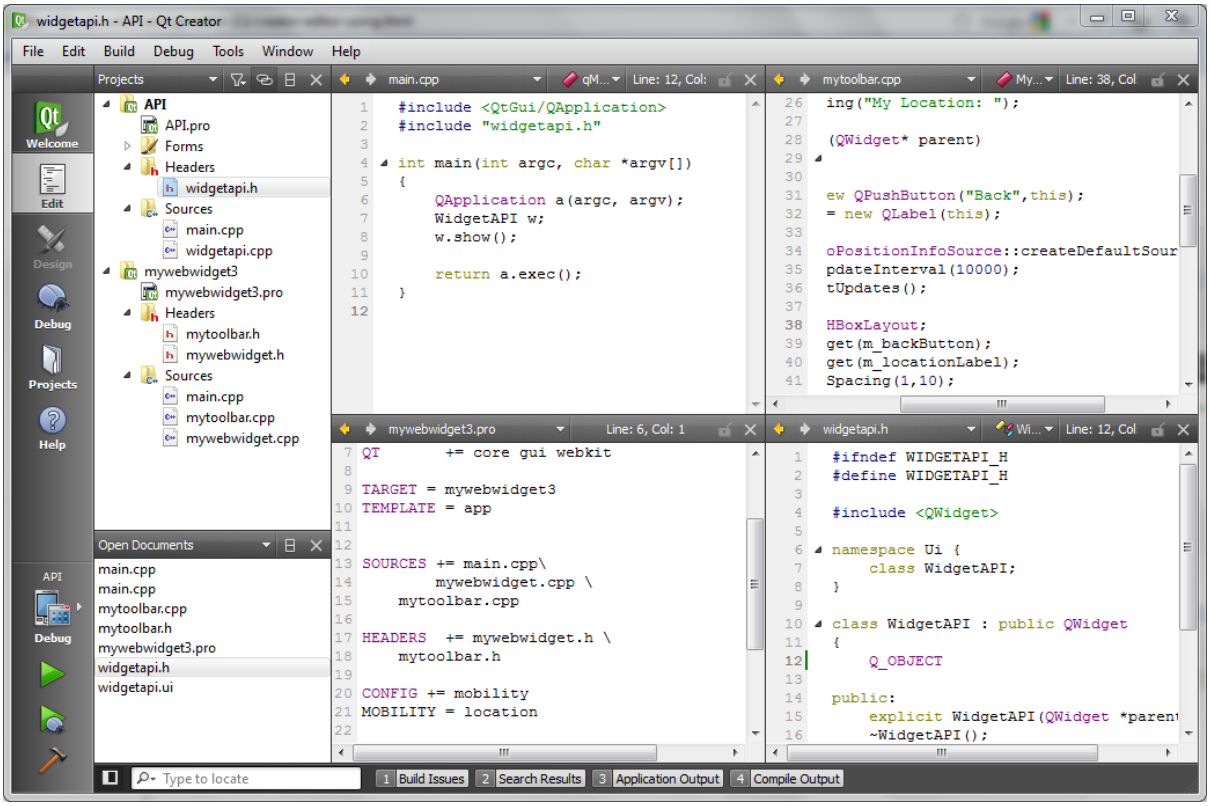
Pada Qt Creator terdapat Build Isu pane yang menyediakan daftar kesalahan dan peringatan yang ditemukan selama membangun suatu aplikasi. Panel itu menyaring output tidak relevan dari perangkat dan menyajikan isu-isu secara terorganisasi.

Untuk mengetahui dimana letak kesalahannya Anda bisa klik dua kali pada baris yang menujukan kesalahan.




## Split Tampilan Editor

Tampilan split editor digunakan bila Anda ingin bekerja dan melihat beberapa file di layar yang sama.




Anda dapat membagi tampilan editor dengan cara berikut:

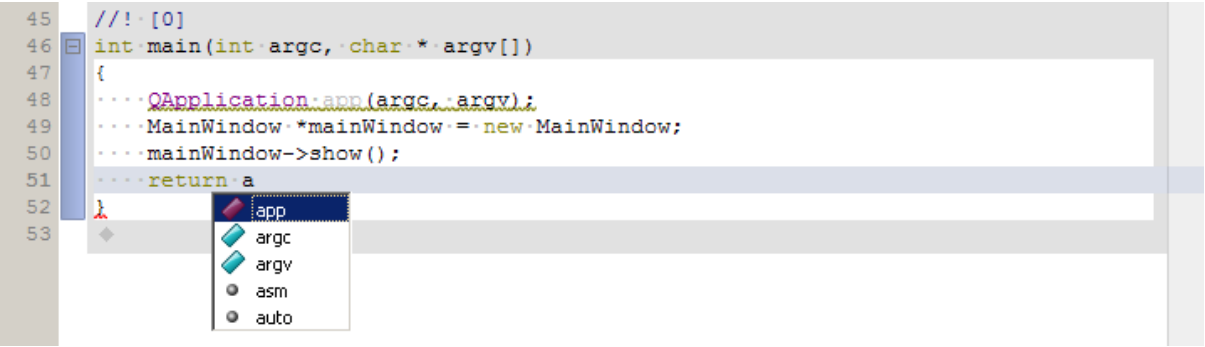
- Untuk membagi tampilan editor ke tampilan atas dan bawah, pilih **Window** > **Split** atau tekan **Ctrl + E, 2**.
- Untuk membagi tampilan editor menjadi pemandangan yang berdekatan, pilih **Window** > **Split Side by Side** atau tekan **Ctrl + E, 3**. Side dengan perintah split menciptakan pandangan sisi sebelah kanan tampilan editor yang sedang aktif.

Untuk bergerak di antara tampilan split, pilih **Window** > **Go to Next Split** atau tekan **Ctrl + E, O**.

Untuk menghapus tampilan split, tempatkan kursor dalam tampilan yang Anda inginkan untuk dihapus, lalu pilih **Window** > **Remove Current Split** atau tekan **Ctrl + E, 0**. Untuk menghapus semua tapi tampilan split yang dipilih, pilih **Window**> **Remove All Splits** atau tekan **Ctrl + E, 1**.

## Completing Code

Ketika Anda menulis kode, Qt Creator menunjukkan sifat, ID, dan potongan kode untuk menyelesaikan kode. Ini memberikan daftar saran konteks-sensitif terhadap statement yang sedang dalam kursor Anda.




Untuk membuka daftar saran setiap saat, tekan **Ctrl + Space**. Jika hanya salah satu pilihan yang tersedia, Qt Creator menyisipkan secara otomatis.

Ketika selesai dipanggil secara manual, Qt Creator melengkapi awalan umum dari daftar saran. Hal ini sangat berguna untuk kelas dengan anggota bernama sama. Untuk menonaktifkan fungsi ini, hapus centang **Autocomplete common prefix** dalam preferensi code completion. Pilih **Tools** > **Options ...** > **Text Editor** > **Completion**.

Secara default, penyelesaian kode mempertimbangkan hanya huruf pertama **case-sensitive**.

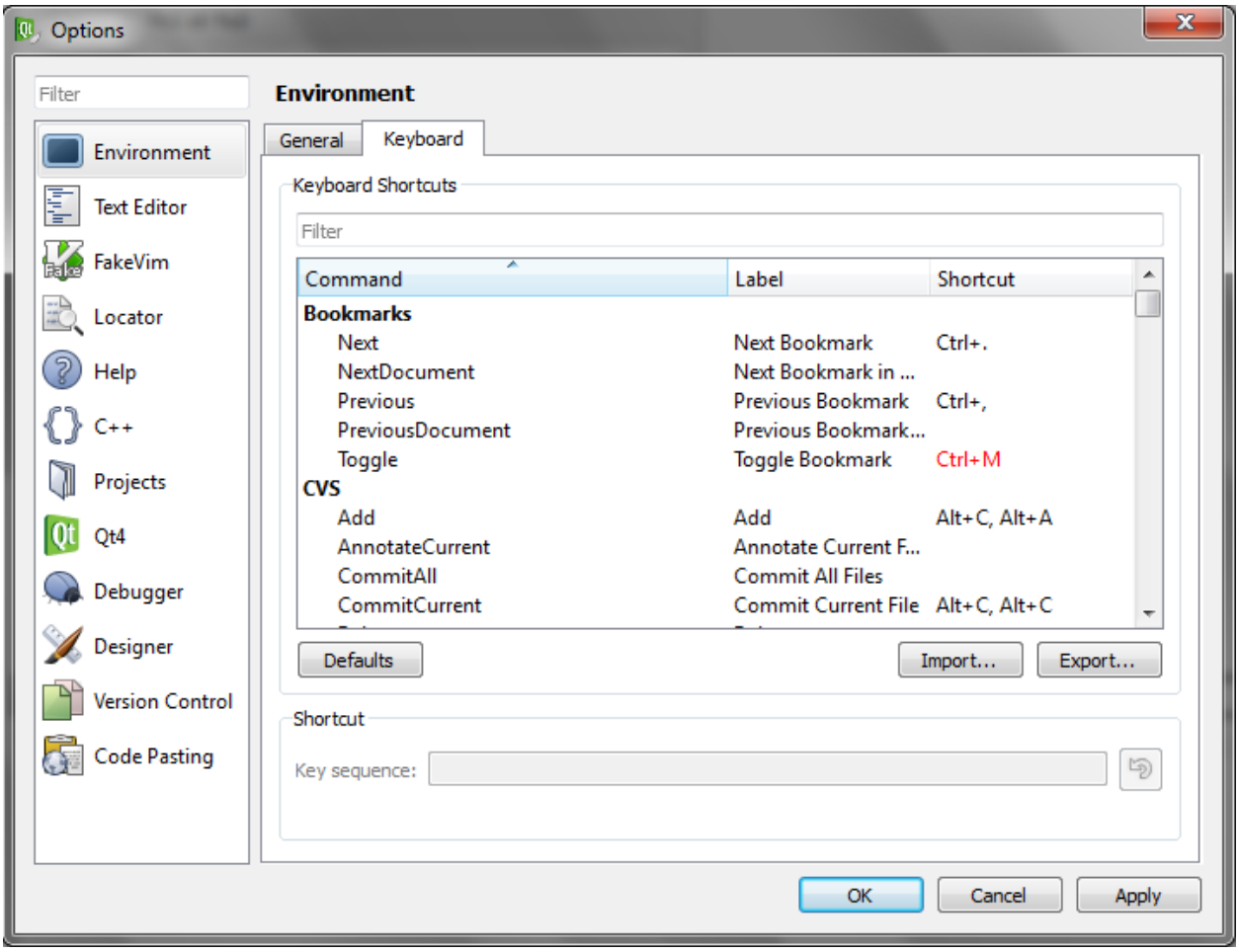
Tabel berikut berisi jenis yang tersedia untuk penyelesaian masing-masing kode dan ikon yang digunakan :

| Icon | Description |
| --- | --- |
|  | A class |
|  | An enum |
|  | An enumerator (value of an enum) |
|  | A function |
|  | A private function |
|  | A protected function |
|  | A variable |
|  | A private variable |
|  | A protected variable |
|  | A signal |
|  | A slot |
|  | A private slot |
|  | A protected slot |
|  | A C++ keyword |
|  | A C++ code snippet |
|  | A QML element |
|  | A QML code snippet |
|  | A macro |
|  | A namespace |

## Shortcut Keyboard

Qt Creator menyediakan berbagai shortcut keyboard untuk mempercepat proses pengembangan Anda.

Untuk menyesuaikan shortcut keyboard, Pilih **Tools > Options ... > Environment > Keyboard**.




Anda dapat merubahnya dengan mengklik salah satu perintah kemudian isi pada key sequence.

Qt Creator memungkinkan Anda untuk menggunakan skema pemetaan cara pintas keyboard yang berbeda:

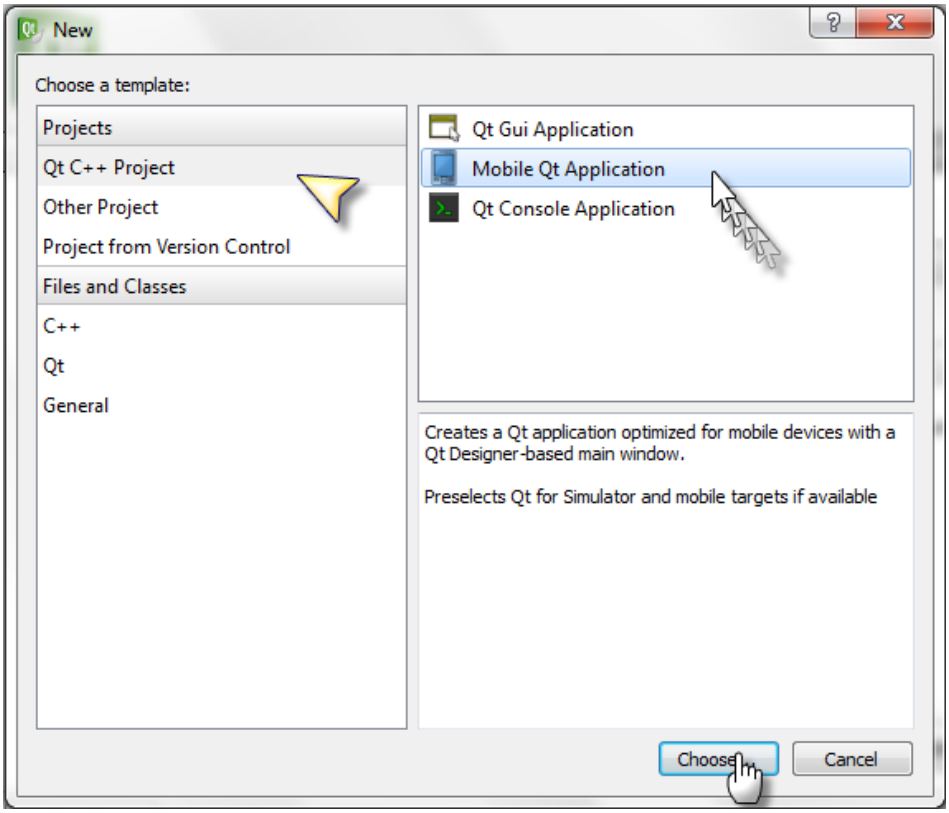
- Untuk mengimpor skema pemetaan keyboard shortcut, klik Impor dan pilih .kms file yang berisi skema pemetaan keyboard shortcut yang ingin Anda impor.
- Untuk mengekspor cara pintas keyboard skema pemetaan saat ini, klik Ekspor dan pilih lokasi di mana Anda ingin menyimpan file km diekspor.

# “Hello World” pada Qt

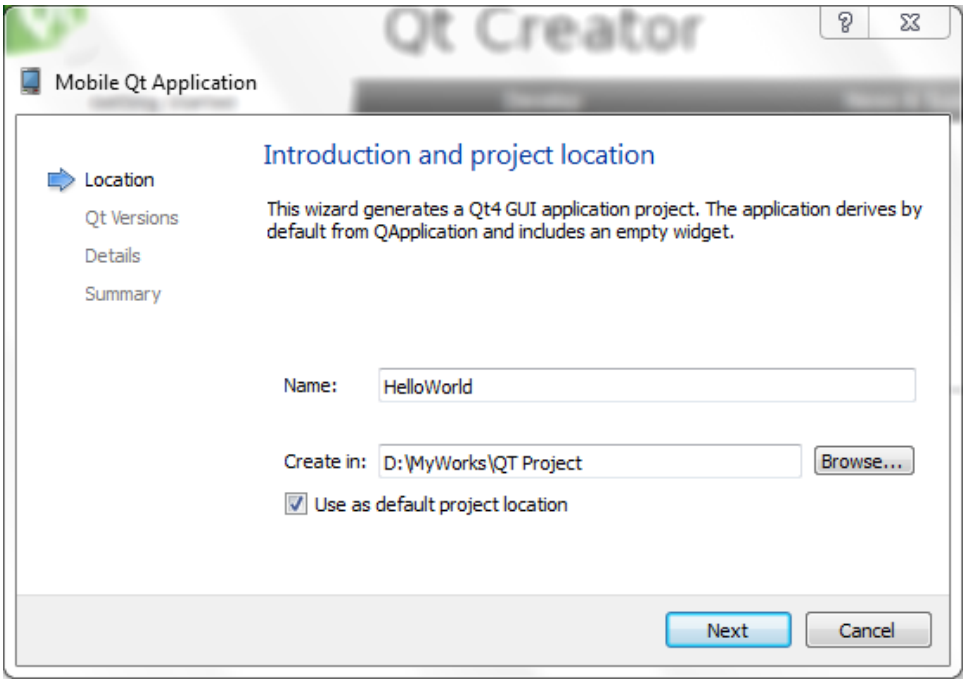
Sudah tradisi turun-temurun seorang programmer untuk memulai membuat sebuah aplikasi adalah dengan Membuat Hello. Sangat sederhana, tapi bagaimana mungkin kita dapat membuat aplikasi super canggih tapi hal kecil seperti ini tidak dapat kita lakukan. ☺

Tanpa panjang lebar, berikut langkah-langkahnya:

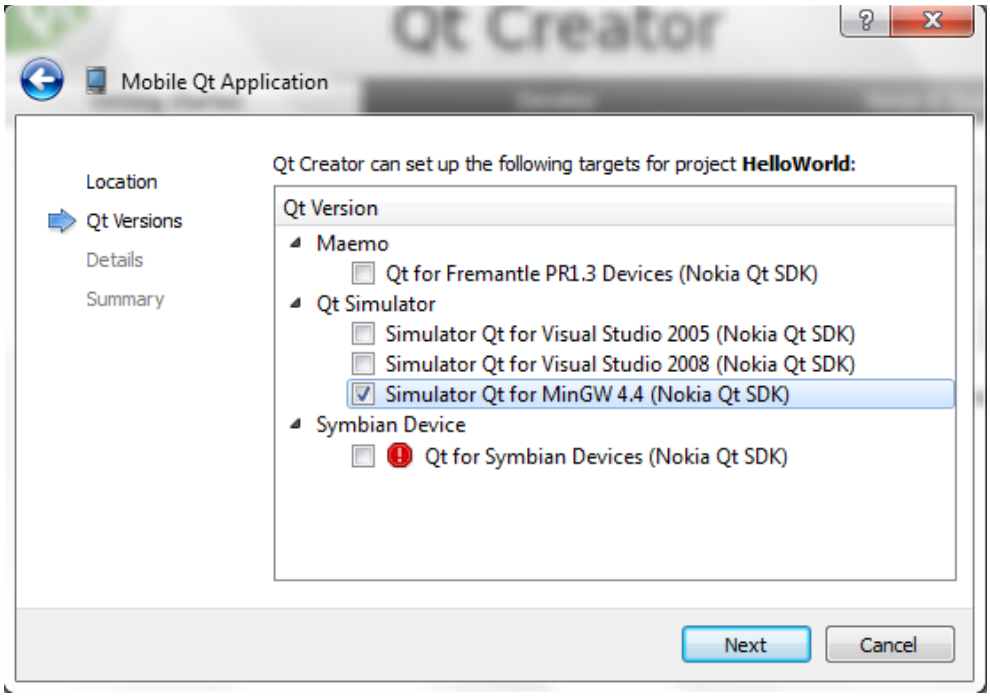
1. Buka Qt Creator, kemudian pilih **File -> New File or Project** . Pada template kita pilih **Qt C++ Project** kemudian pilih **Mobile Qt Application**. Klik tombol *Choose..*




2. Selanjutnya isi nama project dan tempat project akan disimpan. Centang “*use as default project location”* agar folder yang Anda pilih menjadi lokasi standar Qt Project. Klik tombol *Next*.

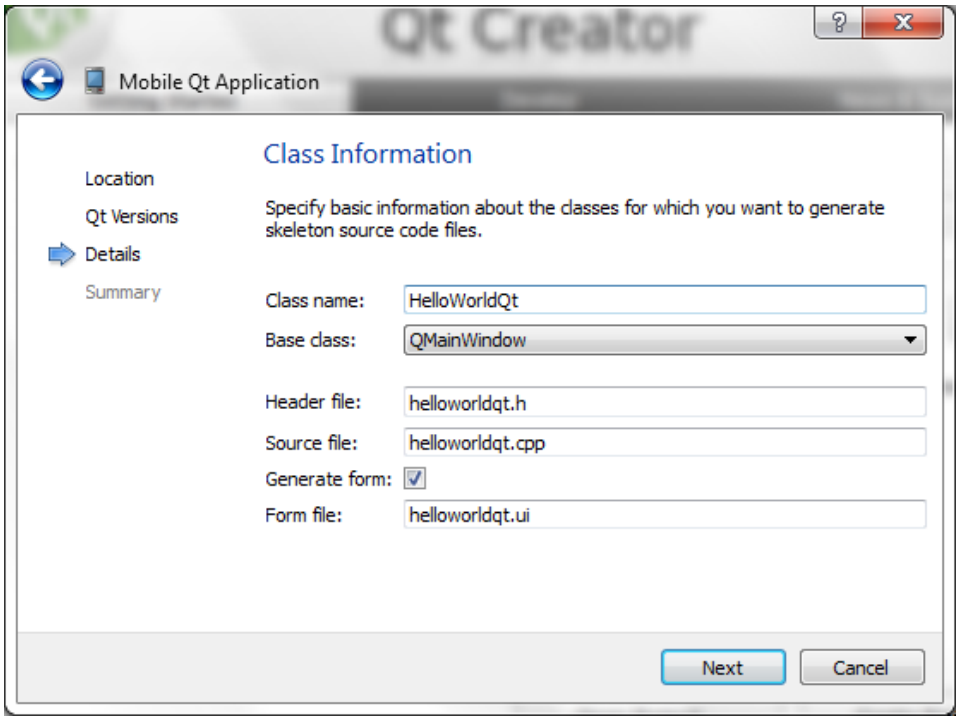



3. Anda akan diminta untuk memilih simulator yang akan digunakan.

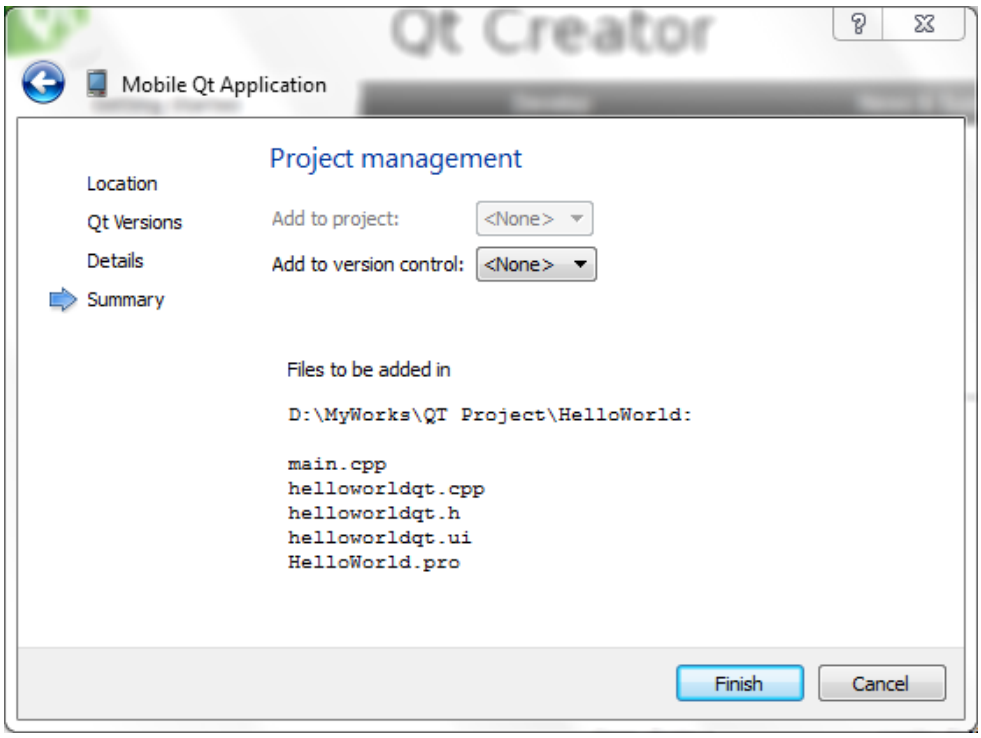



4. Karena kita akan membuat aplikasi di atas Nokia Mobile atau Symbian OS, maka cukup *Simulator Qt for MinGQ 4.4 (Nokia Qt SDK)* yang dicentang. Kita juga bisa memilih Meamo atau langsung ke Device Nokia. Klik tombol *Next*.

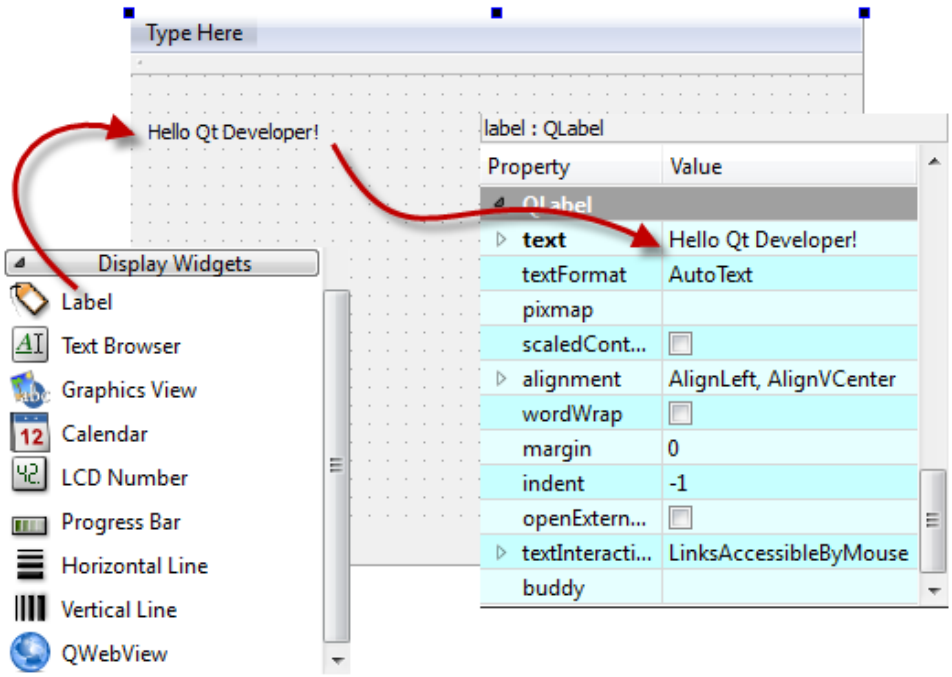
5. Selanjutnya masukkan beberapa detail program berupa Class, Base Class, Header File, Source File, dan From File.




6. Nama Header file, Source file dan Form file akan menyesuaikan dengan nama Class. Untuk Base class pilih *QMainWindows* karena kita akan menggunakan tata letak standar Windows. Klik tombol Next.
7. Terakhir akan tampil summari dari project yang akan kita buat.




8. Klik tombol Finish.
9. Setelah pembuatan project maka akan terlihat tampilan untuk desain aplikasi.
10. Drag dan Drop ToolBox *Label* pada Form Desain seperti gambar berikut:




11. Selanjutnya klik tombol *Run*. Maka akan tampil Simulator Qt.


12. Anda juga bisa rotate device dan memilih device pada simulator, baik berupa device touch screen, atau tidak.

Sekarang Anda sudah bisa membuat aplikasi yang sangat sederhana. ☺ Tapi tenang saja, secara perlahan pada ebook ini kita akan terus mencoba membuat aplikasi yang lebih menarik lagi. So, tetap semangat!

# Signals and Slots

Salah satu karakteristik dari Qt adalah penggunaan signal dan slot. Karena Qt dibangun dengan C++, maka baik signal maupun slot terintegrasi sebagai bagian dari kelas-kelas yang terdapat pada Qt. Lebih jauh, signal/slot menjadi bagian dari object model yang digunakan Qt. Object model ini didesain khusus untuk memudahkan pemrograman aplikasi yang berbasis GUI.

Sebuah signal menandakan bahwa sesuatu telah terjadi. Signal dibangkitkan manakala terjadi interaksi antara user dan program, seperti misalnya ketika user mengklik mouse atau mengetikkan sesuatu di keyboard. Signal juga bisa dipicu oleh hal-hal yang merupakan internal program, misalnya karena sebuah timer yang diaktifkan sebagai alarm.

Slot adalah fungsi yang berespon terhadap signal tertentu. Untuk dapat melakukan hal ini, sebelumnya slot harus dikoneksikan dengan signal yang dimaksud.

Singkatnya signal dan slot digunakan untuk berkomunikasi antar objek dalam program. Mekanisme signal dan slot ini merupakan pusat dari interaksi yang terjadi dalam program.

Di dalam pemrograman berbasis GUI, ketika kita menggunakan sebuah komponen atau widget, (misalkan saja sebuah tombol Close) kita perlu membuat sebuah interaksi dengan widget lainnya (misalkan dialog atau window tempat button tersebut berada) agar jika kita meng-klik tombol close tersebut, jendela atau dialog tersebut akan di tutup. Di dalam programnya, kita membutuhkan sebuah singnal yang dikirim oleh tombol Close (dalam hal ini clicked() ) dan sebuah slot methode penerima dari widget jendelanya (slot close()).

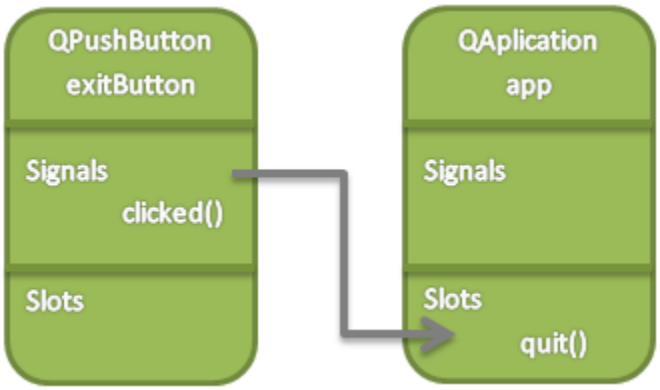
## Basic Signals

Untuk mendemonstrasikan penggunaan signals dan slots, kita menambah interaktivitas. Langkah pertama melengkapi tombol ‘Quit’ dengan fungsionalitas untuk menutup aplikasi. Berikut contoh implementasinya.

```
QObject::connect(exitButton, SIGNAL(clicked()),
                 &app, SLOT(quit()));
```

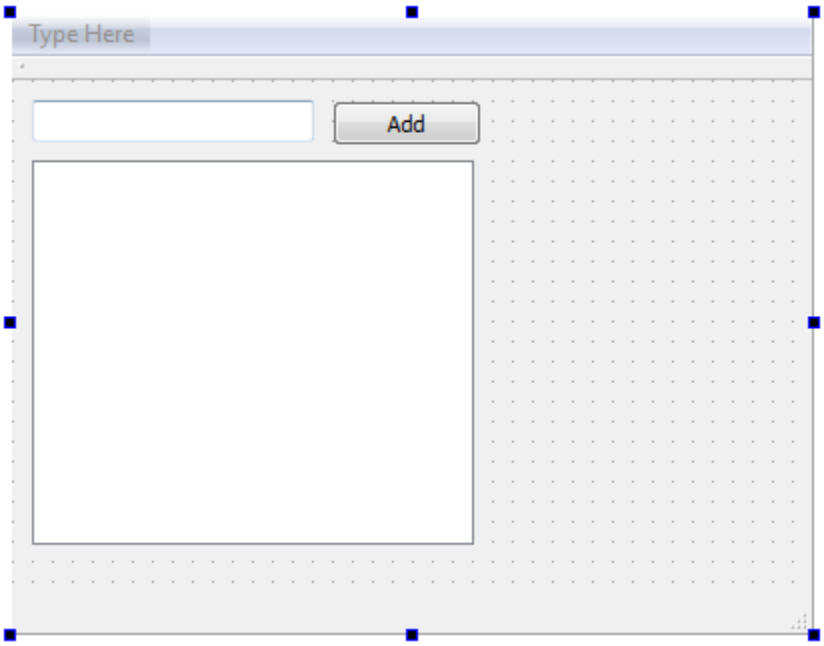
Widget Qt memiliki banyak standar signals, namun Anda juga dapat menambahkan sinyal Anda sendiri. Panggilan untuk QObject::connect() membuat koneksi antara signal dan slot.

Berikut gambaran sederhana Signal and Slots.

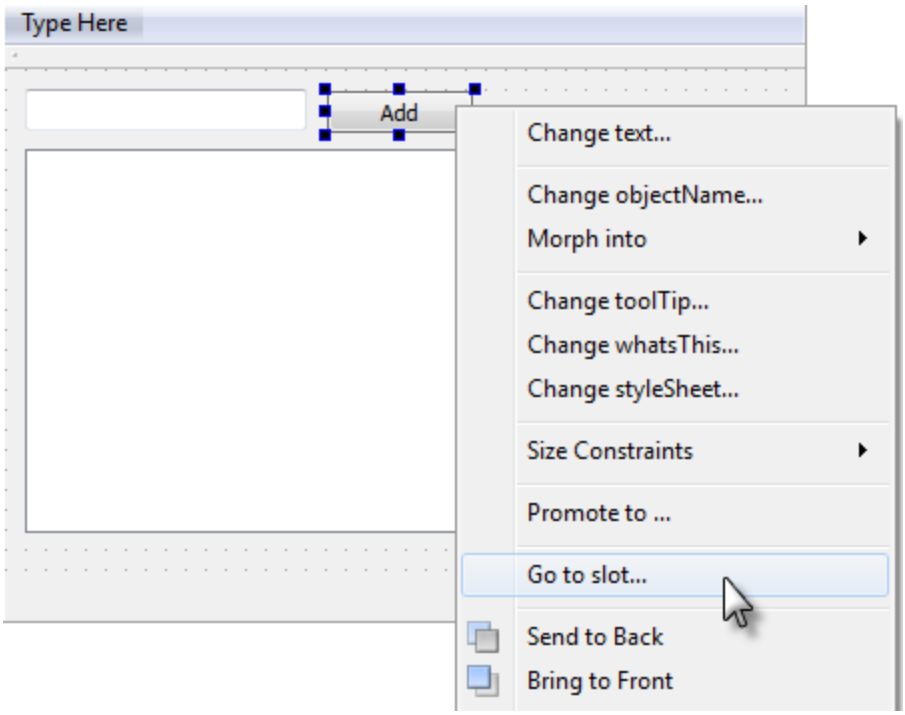



Bingung dengan penjelasan di atas? Mari kita membuat aplikasi menggunakan Signals and Slots.

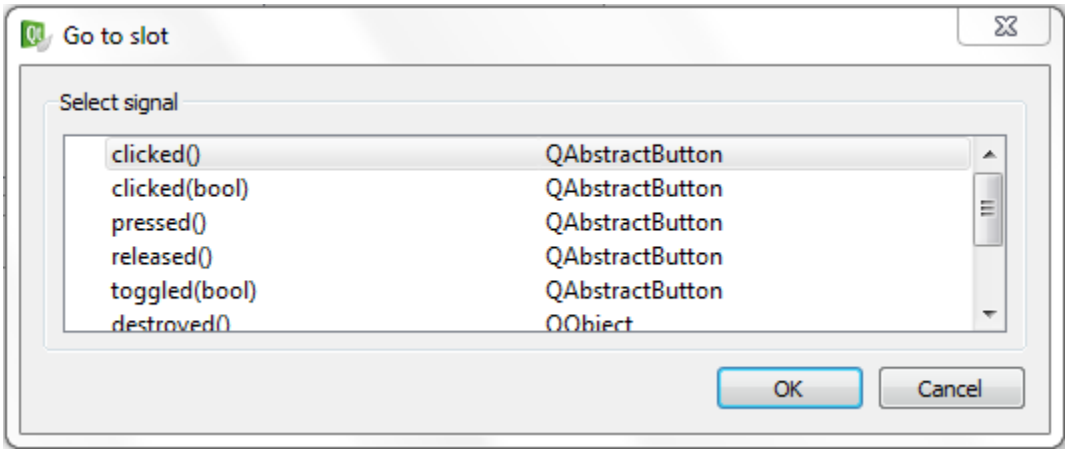
1. Buat sebuah project baru dengan nama **SignalsSlots** dengan nama Class SignalsSlots juga. Caranya sama seperti membuat project Hello World.
2. Tambahkan ToolBox *Push Button, Line Edit,* dan *List Wiget* dan bentuk desain seperti berikut:




3. Kemudian Klik kanan pada Push Button, pilih *Go to slot*.



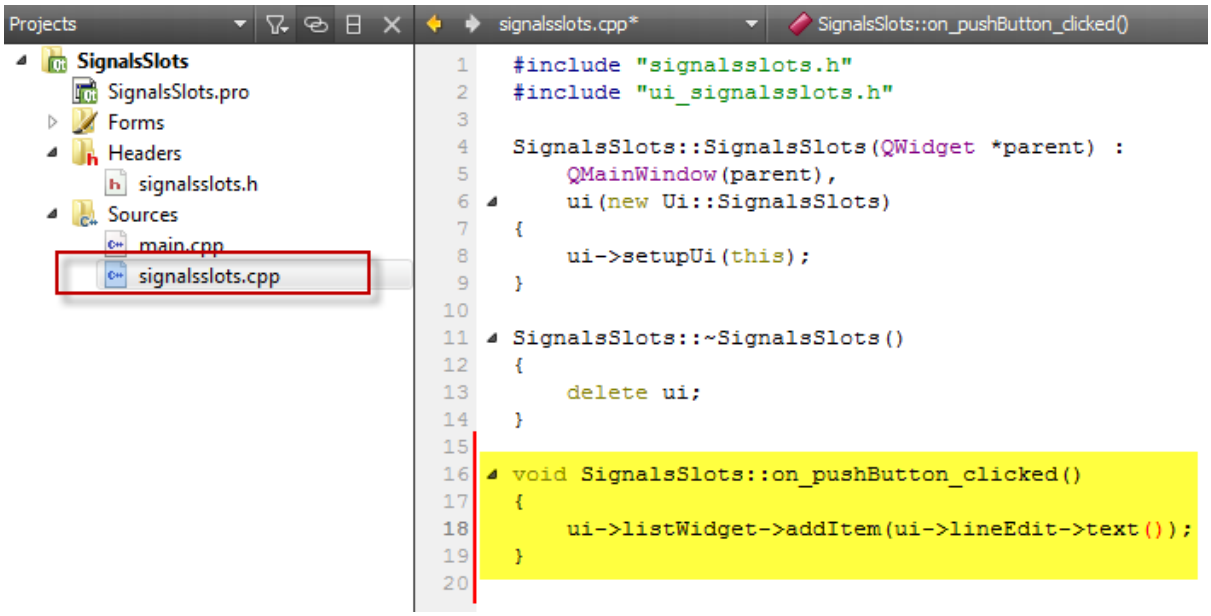

4. Pilih *clicked().* Klik tombol *OK*.




5. Akan masuk ke kode editor, tambahkan baris kode berikut.

```
void SignalsSlots::on_pushButton_clicked()
{
    ui->listWidget->addItem(ui->lineEdit->text());
}
```

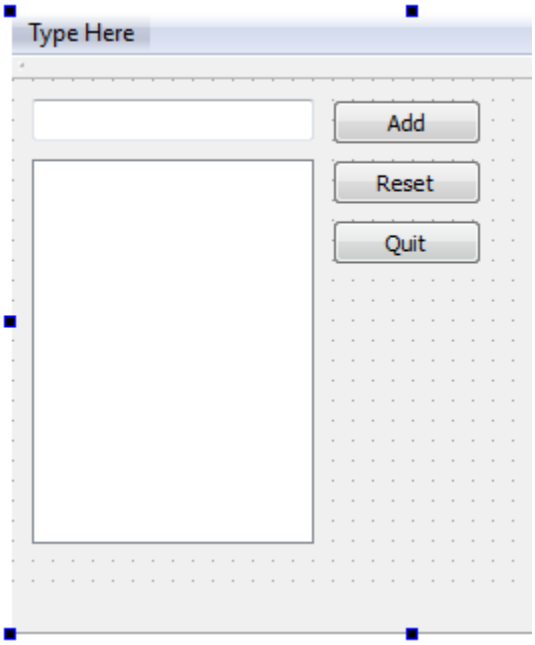
Untuk lebih jelas, lihat gambar di bawah ini:



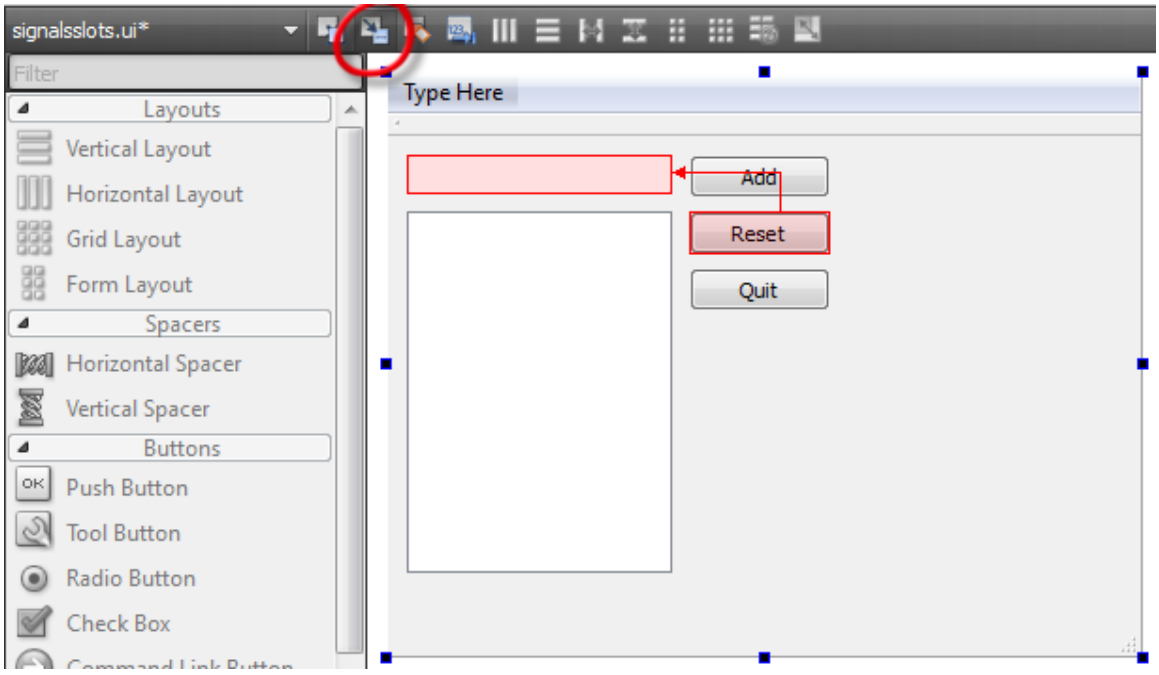

6. Selanjutnya klik tombol *Run*. Maka akan tampil Simulator Qt.


Anda juga bisa menambahkan berbagai signals/slots pada ToolBox, contohnya membuat tombol Quit, Reset, Save, dsb. Jika Anda masih kurang puas, mari kita pelajari lebih dalam signals and slot. ☺

1. Masih pada project SignalsSlots, sekarang Anda tambahkan 2 buah Push Button dan desain menjadi seperti berikut :

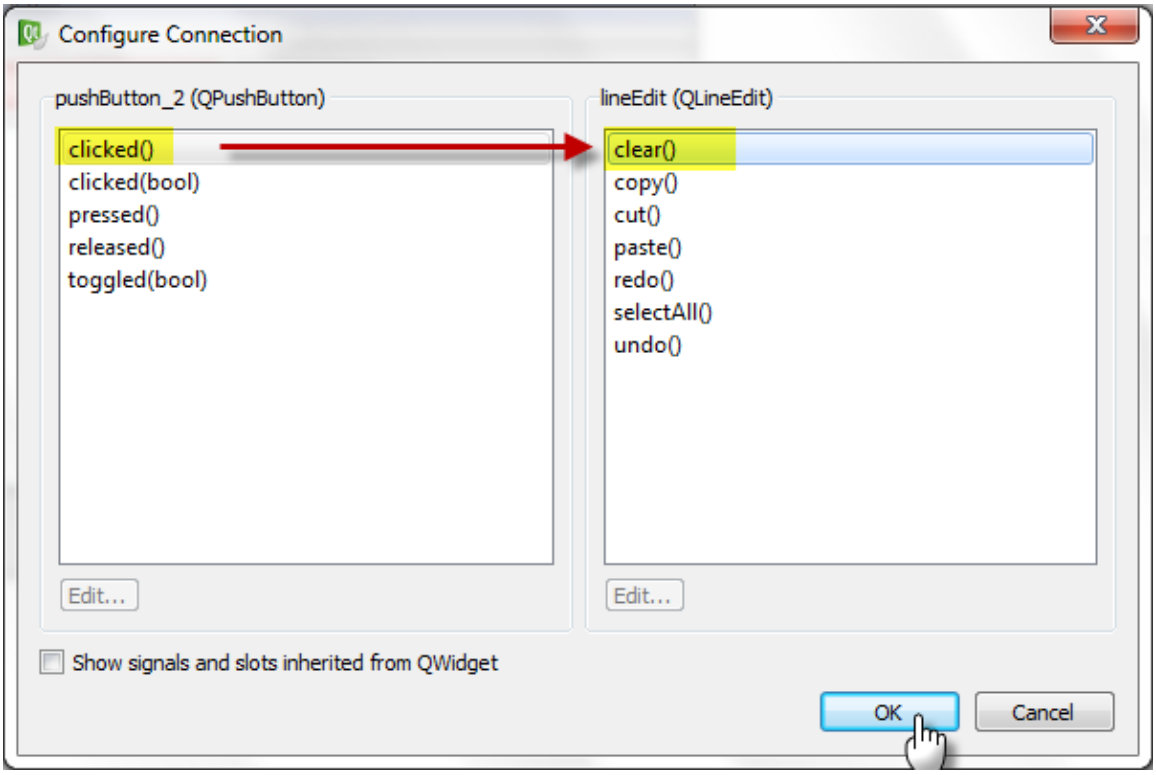



2. Setelah itu masuk pada mode Edit Signals/Slots. Lihat gambar di bawah ini:

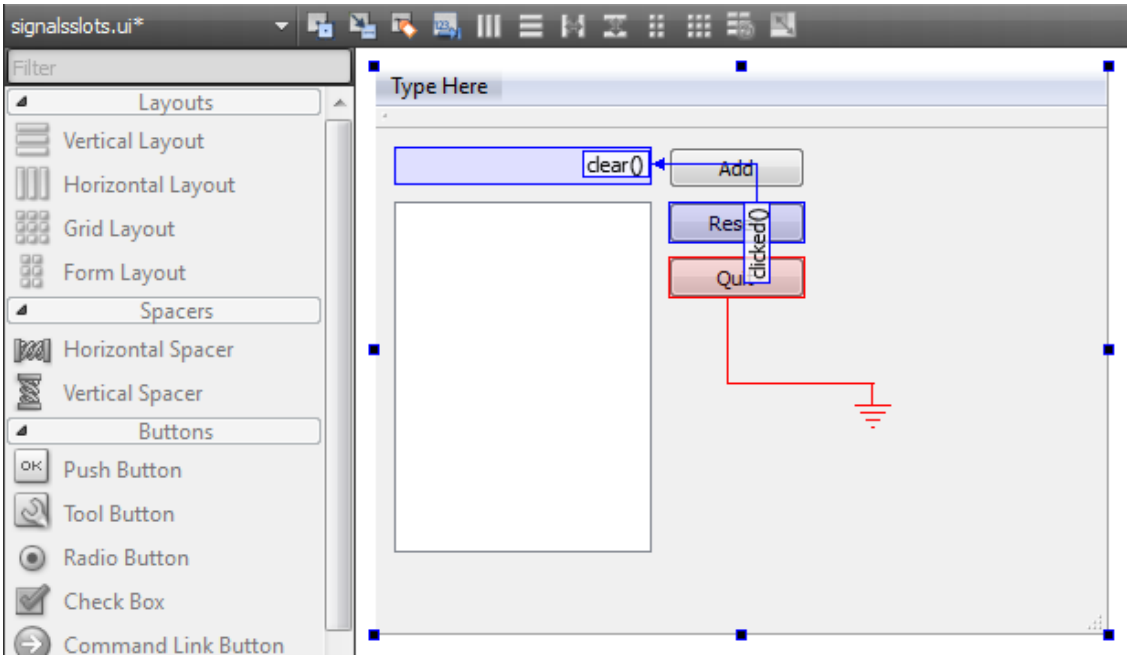



3. Kemudian klik pada tombol reset dan geser ke arah Line Edit kemudian lepaskan. Nantinya tombol reset ini berfungsi untuk menghapus data yang ada pada Line Edit.

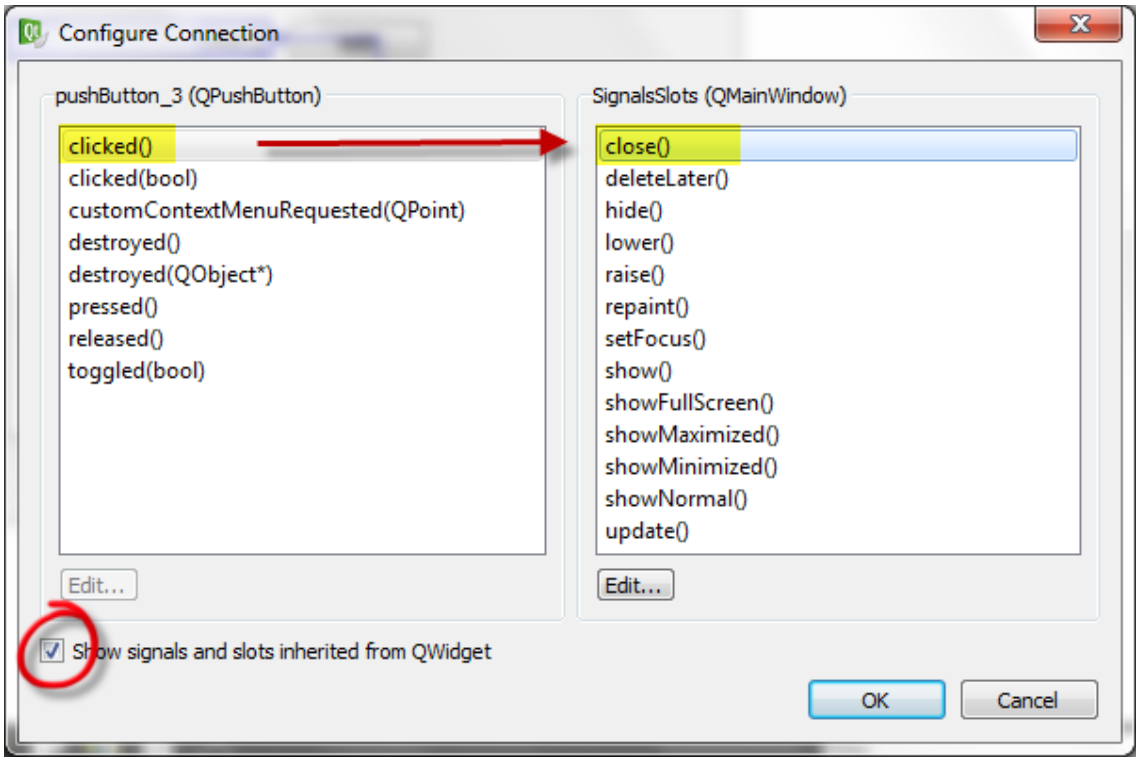
4. Selanjutnya pada **QPushButton** pilih *clicked()* dan pada **QLineEdit** pilih *clear().*




5. Setelah menambahkan signals pada tombol reset, sekarang lakukan hal yang sama kepada tombol Quit. Namun tidak perlu digeser kea rah Line Edit, cekup geser ke sembarang saja.



6. Selanjutnya centang “Show signals and slots…” kemudian pada **QPushButton** pilih *clicked()* dan pada **QMainWindows** *pilih close().*




7. Run kembali project SignalsSlots. Pada Qt Simulator akan tampil seperti gambar berikut:

Ketika Anda menuliskan sesuatu pada Line Edit dan meng-klik tombol reset maka pada Line Edit akan kosong kembali. Anda juga bisa melakukannya ke List Widget. Caranya, sama seperti membuat signal reset pada Line Edit. Sedangkan tombol Quit berfungsi untuk menutup aplikasi yang sedang berjalan.

# Layouts

Pada kebanyakan antar muka suatu aplikasi yang dibuat dari beberapa widget diatur sesuai dengan ukuran layar berdasarkan device. Dalam Qt, hal ini dilakukan melalui berbagai jenis pengaturan tata letak yang secara otomatis menyelaraskan dan mengubah ukuran widget berdasarkan besarnya layar suatu device.

Pengelolaan layout sangat penting pada perangkat mobile. Sebuah aplikasi sederhana pada PC desktop mungkin sangat baik memiliki ukuran yang tetap yang tidak dapat diubah oleh pengguna. Di sisi lain, kebanyakan telepon selular mendukung rotasi layar (misalnya dengan menggunakan sensor percepatan). Untuk menjaga kegunaan, aplikasi biasanya harus beradaptasi dengan orientasi layar baru.

## Kelas Qt Layouts

Selain menggunakan Qt Designer untuk mendesain form, juga terdapat kelas Layout yang dapat Anda gunakan untuk kebutuhan layout.

| Kelas | Keterangan |
|---|---|
| **QBoxLayout** | Membentuk baris vertikal atau horisontal |
| **QButtonGroup** | Container untuk mengorganisir kelompok tombol widget |
| **QFormLayout** | Mengatur bentuk widget inputan dan label |
| **QGraphicsAnchor** | Anchor antara dua item dalam suatu QGraphicsAnchorLayout |
| **QGraphicsAnchorLayout** | Layout dimana seseorang dapat widget anchor bersama dalam Graphic view |
| **QGridLayout** | Layout widget dalam kotak |
| **QGroupBox** | Kelompok kotak bingkai dengan sebuah judul |
| **QHBoxLayout** | Membentuk baris horisontal |
| **QLayout** | Kelas dasar geometri manajer |
| **QLayoutItem** | Abstrak item yang memanipulasi QLayout |
| **QSizePolicy** | Atribut layout yang menggambarkan ukuran horisontal dan vertikal |
| **QSpacerItem** | Ruang kosong pada Layout |
| **QStackedLayout** | Stack widget dimana hanya satu widget yang terlihat pada suatu waktu |
| **QStackedWidget** | Stack widget dimana hanya satu widget yang terlihat pada suatu waktu |
| **QVBoxLayout** | Membentuk baris vertikal |

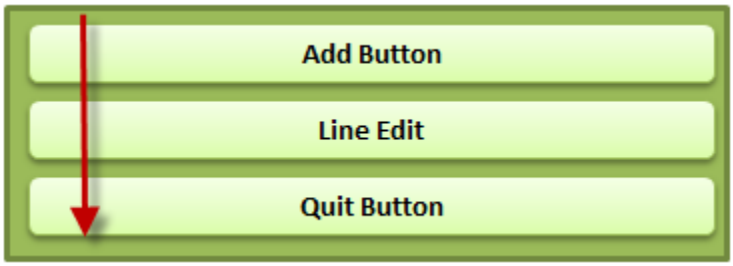
| | |
|---|---|
| **QWidgetItem** | Item layout yang mewakili sebuah widget |

## Komponen Layout

Ada 4 jenis komponen layout yang disediakan Qt, yaitu:

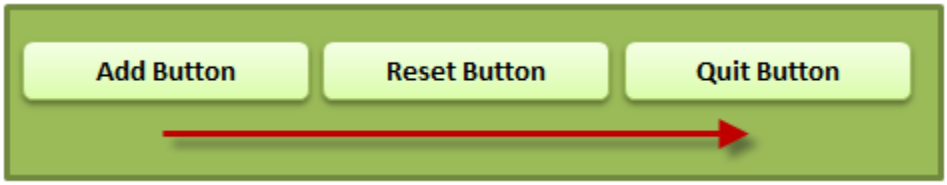
1. **Vertical Layout (QVBoxLayout)**

Layout ini membuat tampilan aplikasi full vertical berbentuk baris ke bawah. Contoh skema layout:



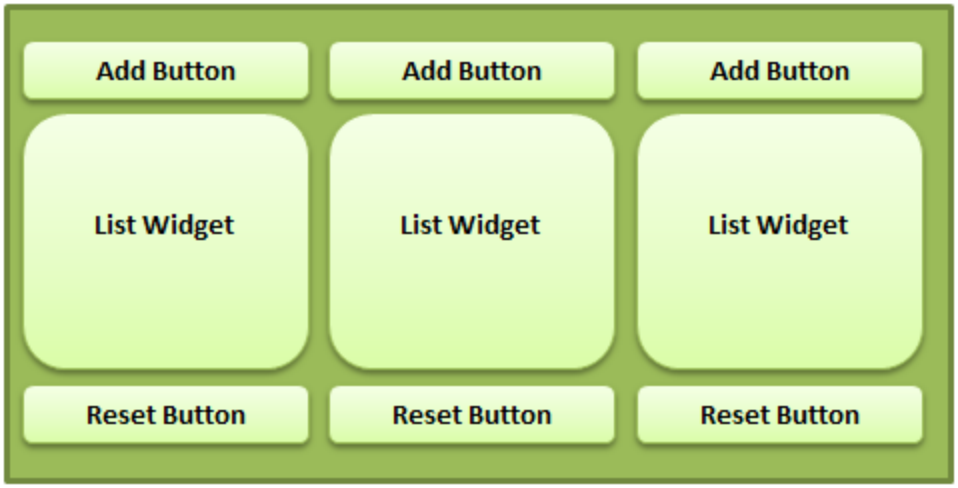

2. **Horizontal Layout (QHBoxLayout)**

Layout ini membuat tampilan aplikasi secara horizontal. Contoh skema layout:




3. **Grid Layout (QGridLayout)**

Layout ini membuat tampilan aplikasi membentuk kotak-kotak. Contoh skema layout:




4. **Form Layout (QFormLayout)**

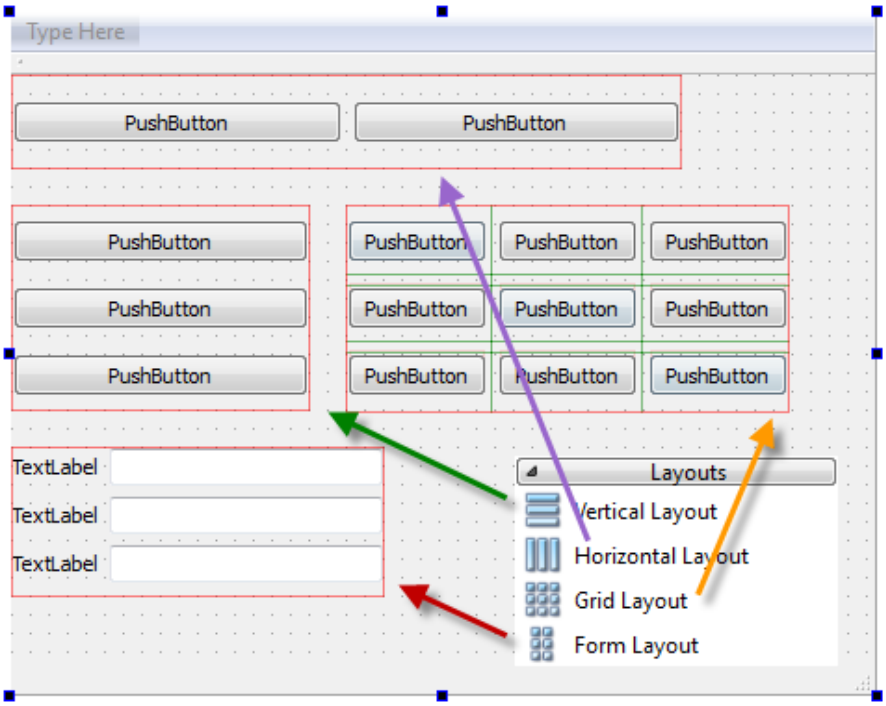
Layout ini membuat tampilan aplikasi seperti formulir. Contoh skema layout:


Ke-empat komponen layoutmerupakan turunan kelas QLayout. Cara menggunakan komponen layout pada Qt Creator cukup mudah. Kita cukup meletakan jenis layout yang diinginkan, pilih satu jenis dari empat layout yang disediakan oleh Qt Creator.

## Group Layout

Anda juga bisa mengkombinasikan semua layout dalam satu project. Cukup drag and drop toolbox layout pada lembar kerja Anda.

Beirkut contoh group layout :

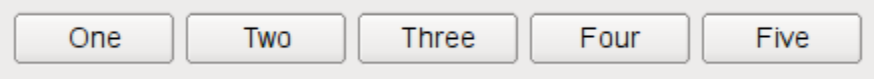



## Pemasangan Kode Layout

Kode berikut membuat sebuah QHBoxLayout yang mengelola geometri lima QPushButtons.

```
QWidget *window = new QWidget;
   QPushButton *button1 = new QPushButton("One");
   QPushButton *button2 = new QPushButton("Two");
   QPushButton *button3 = new QPushButton("Three");
   QPushButton *button4 = new QPushButton("Four");
   QPushButton *button5 = new QPushButton("Five");

   QHBoxLayout *layout = new QHBoxLayout;
   layout->addWidget(button1);
   layout->addWidget(button2);
   layout->addWidget(button3);
   layout->addWidget(button4);
   layout->addWidget(button5);
   window->setLayout(layout);
 window->show();
```

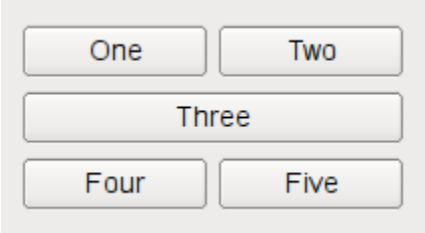
Maka akan membuat geometri layout seperti berikut :




Untuk kode QVBoxLayout sama seperti QHBoxLayout , kecuali pada garis mana layout dibuat. Kode untuk QGridLayout agak sedikit berbeda, karena kita harus menentukan posisi baris dan kolom child widget :

```
QWidget *window = new QWidget;
     QPushButton *button1 = new QPushButton("One");
     QPushButton *button2 = new QPushButton("Two");
     QPushButton *button3 = new QPushButton("Three");
     QPushButton *button4 = new QPushButton("Four");
     QPushButton *button5 = new QPushButton("Five");

     QGridLayout *layout = new QGridLayout;
     layout->addWidget(button1, 0, 0);
     layout->addWidget(button2, 0, 1);
     layout->addWidget(button3, 1, 0, 1, 2);
     layout->addWidget(button4, 2, 0);
     layout->addWidget(button5, 2, 1);

     window->setLayout(layout);
window->show();
```

Maka akan membuat geometri layout seperti berikut :




QFormLayout akan menambah dua widget pada setiap baris, biasanya sebuah QLabel dan QLineEdit. Menambahkan QLabel dan QLineEdit pada baris yang sama akan mengatur QLineEdit sebagai buddy QLabel itu. Kode berikut akan menggunakan QFormLayout untuk menempatkan tiga QPushButtons dan QLineEdit sesuai pada baris.

```
QWidget *window = new QWidget;
    QPushButton *button1 = new QPushButton("One");
    QLineEdit *lineEdit1 = new QLineEdit();
    QPushButton *button2 = new QPushButton("Two");
    QLineEdit *lineEdit2 = new QLineEdit();
    QPushButton *button3 = new QPushButton("Three");
    QLineEdit *lineEdit3 = new QLineEdit();

    QFormLayout *layout = new QFormLayout;
    layout->addRow(button1, lineEdit1);
    layout->addRow(button2, lineEdit2);
    layout->addRow(button3, lineEdit3);

    window->setLayout(layout);
window->show();
```

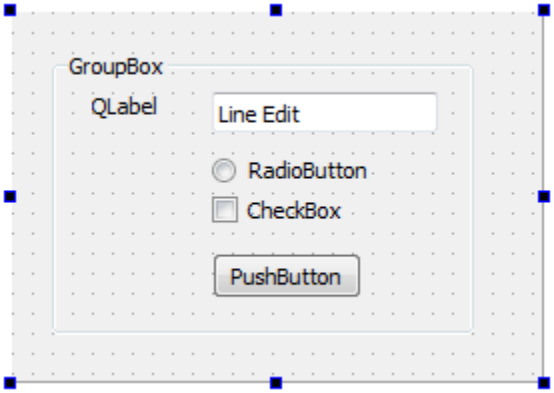
Maka akan membuat geometri layout seperti berikut :


## Komponen Umum

Komponen umum GUI yang sering digunakan dalam pembuatan aplikasi menggunakan Qt meliputi :

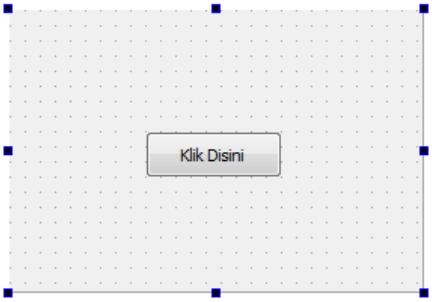
- QPushButton
- QLabel
- QLineEdit
- QCheckBox
- QRadioButton
- QGroupBox

Contoh tampilan komponen umum ini dapat dilihat pada gambar di bawah ini :



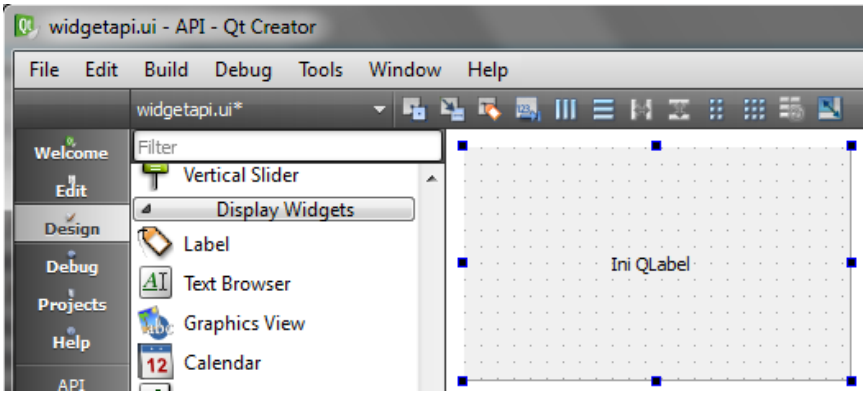

### QPushButton

Komponen ini adalah sebuah tombol. Komponen QPushButton sudah menyediakan signal yaitu *clicked().* Untuk mengganti nama tombol yang terlihat di sisi user, kita dapat melakukannya melalui *property* di sebelah kanan bawah dan mengganti nilai text-nya, misalkan “Klik Disini”, Atau mudahnya, Anda cukup klik dua kali pada QPushButton Anda kemudian ubah Text pada tombol, contoh:




## QLabel

Komponen QLabel digunakan untuk menampilkan sebuah tulisan. Komponen ini dapat dilihat dibagian Display Widget dengan nama Label. Kita dapat langsung menampilkan tulisan secara runtime dengan mengisi nilai string pada text di bagian property QLabel tersebut atau mudahnya Anda cukup klik dua kali pada QLabel Anda kemudian ubah textnya.

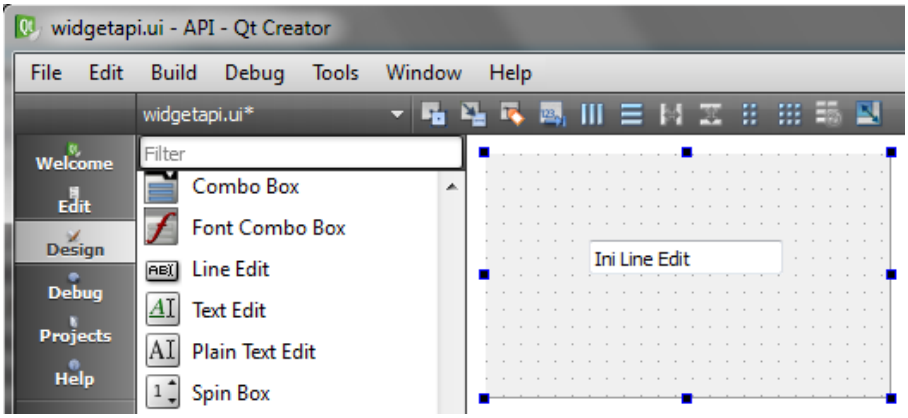



## QLineEdit

Komponen ini adalah Line Edit yang merupakan bagian dari input Widget. Line Edit kata lainnya adalah TextBox, fungsinya untuk menerima input user.

Untuk mengisi nilai pada QLineEdit, kita dapat memanfaatkan fungsi setText(). Atau mudahnya Anda cukup klik dua kali pada QLineEdit. Contohnya, missal nama variabelnya lineEdit untuk QLineEdit :

```
ui->lineEdit->setText("Isi  nama Anda");
```




## QCheckBox

Komponen QCheckBox digunakan untuk input user berupa aksi mencentang. Kita dapat mengetahui status apakah komponen ini dicentang atau tidak dengan menggunakan *isChecked().*

Ketika ada perubahan kondisi dari centang ke tidak centang, komponen QCheckBox akan menghasilkan signal stateChanged(int). untuk model checkbox non-tristate QCheckBox akan menghasilkan signal toggled(bool).

Sebagai ilustrasi, kita akan menambahkan sebuah label, jika ini dicentang maka label ini akan ditulis "Saya Setuju" dan sebaliknya ditulis "Saya tidak Setuju". Kita dapat menguji checkbox itu tercentang atai tidak dengan memanfaatkan fungsi isChecked(). Berikut contoh kode programnya :

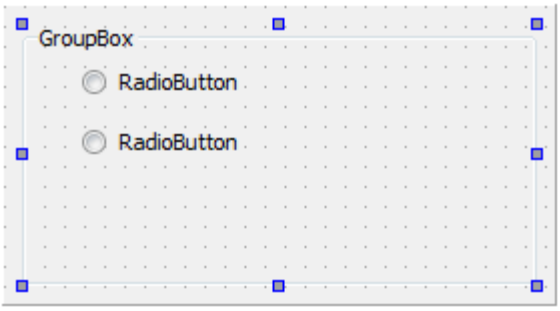
```
void MainWindows::on_checkBox_stateChanged(int)
{
    if(ui->checkBox->isChecked())
        ui->label->setText("Saya Setuju");
    else
        ui->label->setText("Saya tidak Setuju");
}
```

## QRadioButton dan QGroupBox

Komponen QRadioButton umumnya digunakan untuk menyediakan beberapa opsi kepada user untuk memilih salah satu opsi tersebut. Di sisi lain, QGroupBox digunakan untuk mengelompokan beberapa komponen sehingga secara GUI lebih menarik.

Untuk me-monitoring kondisi QRadioButton apakah dipilih atau tidak, kita dapat memanfaatkan signal toggled(bool) dan fungsi isChecked().

Contoh penggunaan RadioButton di dalam group box dapat dilihat pada gambar dibawah ini:

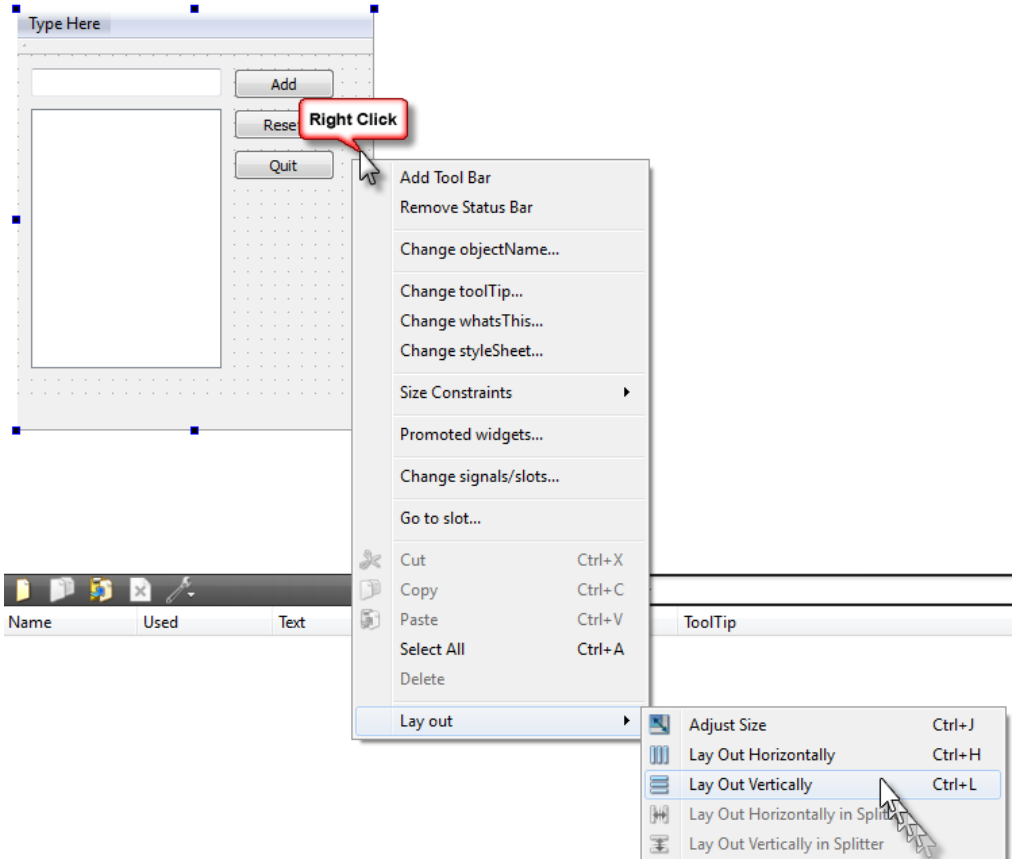



## Membuat Aplikasi Full Screen

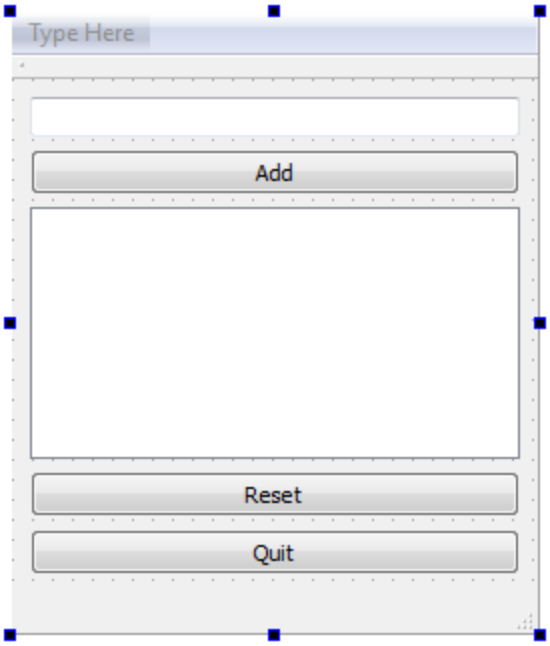
Dengan Qt, membuat aplikasi tampil full screen sangat mudah. Anda cukup merubah satu baris kode pada file main.cpp. Kita akan coba membuat project SignalsSlots yang telah kita buat menjadi vertical layout dan tampil full screen.

Berikut langkah-langkahnya:

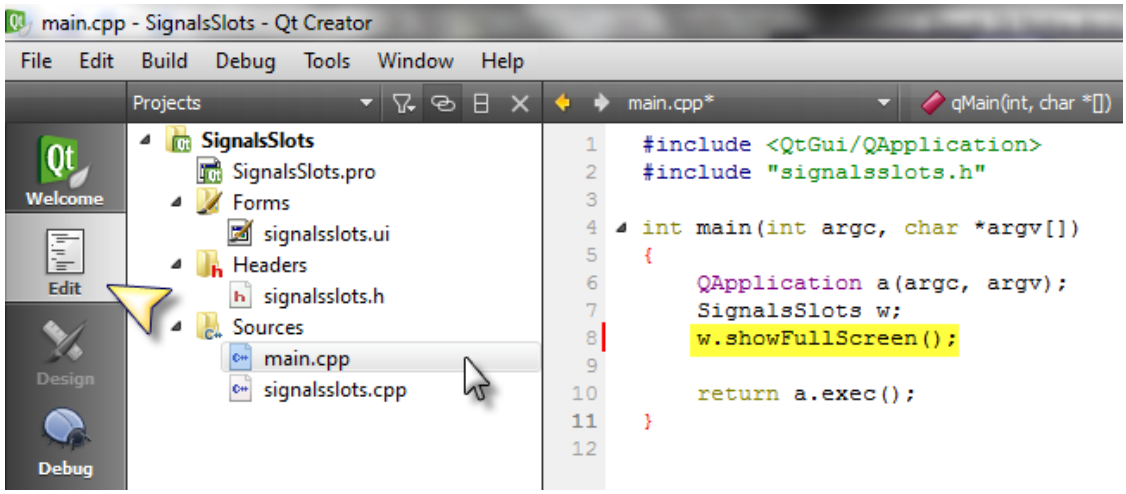
1. Klik kanan lembar kerja, pilih **Lay Out -> Lay Out Verticaly**




2. Layout aplikasi kita akan berubah seperti berikut:




3. Selanjutnya masuk pada mode Edit, kemudian buka file *main.cpp*




4. Pada baris `w.show();` ubah menjadi `w.showFullScreen();`
5. Kemudian *Run* Project. Pada Qt simulator akan tampil seperti berikut:


Full Screen


Bukan Full Screen

# Message Dialog

Kelas QMessageBox menyediakan dialog modal untuk menginformasikan pengguna atau untuk menanyakan pertanyaan pengguna dan menerima jawaban.

Sebuah kotak pesan menampilkan teks utama untuk memperingatkan pengguna. Sebuah kotak pesan juga dapat menampilkan tombol ikon standar untuk menerima respon pengguna.

Telah disediakan dua API untuk menggunakan QMessageBox, API berbasis properti, dan fungsi-fungsi statis. Memanggil salah satu fungsi statis adalah pendekatan yang lebih sederhana, tetapi kurang fleksibel dibandingkan menggunakan API berbasis properti, dan hasilnya kurang informatif. Dianjurkan Menggunakan API berbasis properti.

## Properti berbasis API

Untuk menggunakan API berbasis properti, membangun sebuah instance dari QMessageBox, mengatur properti yang diinginkan, dan panggilan `exec()` untuk menampilkan pesan. Konfigurasi yang paling sederhana adalah hanya untuk mengatur properti pesan teks.

```
QMessageBox msgBox;
msgBox.setText("Hello Qt Developer!");
msgBox.exec();
```

Pengguna harus mengklik tombol OK untuk mengabaikan kotak pesan. Sisanya GUI diblokir sampai kotak pesan diberhentikan. Mungkin ada sedikit perbebaan tampilan antara Desktop dan Mobile.

Jika Anda ingin menggunakan Message Dialog, pastikan Anda menambahkan library MessageBox. Contoh full kode program MessageBox dapat dilihat di bawah ini.

```
#include <QtGui/QApplication>
#include <QMessageBox>
#include "mainwindow.h"

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

    QMessageBox msgBox;
    msgBox.setText("Hello Qt Developer!");
    msgBox.exec();

    MainWindow w;
#if defined(Q_WS_S60)
    w.showMaximized();
#else
    w.show();
#endif
    return a.exec();
}
```

Message dialog seperti ini tepatnya hanya berguna untuk sebuah peringatan saja, Anda juga bisa membuatnya lebih tepat guna. Seperti menggunakan tombol ok, cancel atau no.

```
    QMessageBox msgBox;
    msgBox.setText("The document has been modified.");
    msgBox.setInformativeText("Do you want to save your
changes?");
    msgBox.setStandardButtons(QMessageBox::Save |
QMessageBox::Discard | QMessageBox::Cancel);
    msgBox.setDefaultButton(QMessageBox::Save);
    int ret = msgBox.exec();
```


## Security Level Icon

QMessageBox mendukung empat tingkat keparahan pesan yang telah ditetapkan, atau jenis pesan yang benar-benar berbeda pada ikon standar. Tentukan salah satu dari empat jenis pesan yang telah ditetapkan dengan menyeting properti ikon ke salah satu ikon standar. Aturan Berikut ini adalah panduan:

| Gambar Icon | Nama Icon | Keterangan |
|---|---|---|
|  | Question<br>`QMessageBox::Question` | Untuk mengajukan pertanyaan selama operasi normal. |
|  | Information<br>`QMessageBox::Information` | Untuk pelaporan informasi tentang operasi normal |
|  | Warning<br>`QMessageBox::Warning` | Untuk pelaporan kesalahan tidak penting |
|  | Critical<br>`QMessageBox::Critical` | Untuk pelaporan kesalahan kritis. |

## Fungsi API Statis

Membuat kotak pesan dengan fungsi API statis, meskipun nyaman, kurang fleksibel dibandingkan menggunakan API berbasis properti, karena kurangnya tanda tangan fungsi statis parameter untuk pengaturan teks informatif dan properti teks secara rinci. Karena ini memiliki kelemahan yang jelas untuk membuat kotak pesan kurang dibaca, pedoman platform tidak menganjurkan hal tersebut. Microsoft Windows User Interface merekomendasikan menggunakan nama aplikasi sebagai judul jendela, yang berarti bahwa jika Anda memiliki teks yang informatif di samping teks utama Anda, Anda harus menggabungkan ke parameter teks.

Fungsi statis yang tersedia untuk membuat kotak pesan adalah information(), question(), warning(), dan critical().

```
    int ret = QMessageBox::warning(this, tr("My Application"),
                                   tr("The document has been
modified. \n""Do you want to save your changes?"),
QMessageBox::Save | QMessageBox::Discard | QMessageBox::Cancel,
QMessageBox::Save);
```

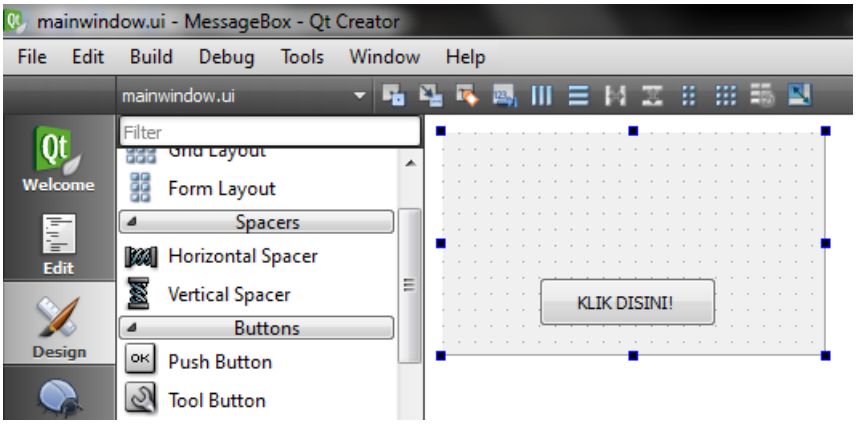
Jika tombol-tombol standar tidak cukup fleksibel untuk kotak pesan Anda, Anda dapat menggunakan addButton() yang mengambil teks dan ButtonRoleto untuk menambahkan tombol kustom. ButtonRole digunakan oleh QMessageBox untuk menentukan urutan tombol-tombol pada layar (yang bervariasi sesuai dengan platform). Anda dapat menguji nilai clickedButton() setelah memanggil exec(), Misalnya.

```
    QMessageBox msgBox;
    QPushButton *connectButton = msgBox.addButton(tr("Connect"),
QMessageBox::ActionRole);
    QPushButton *abortButton = msgBox.addButton(QMessageBox::Abort);

    msgBox.exec();

    if (msgBox.clickedButton() == connectButton) {
        // connect
    } else if (msgBox.clickedButton() == abortButton) {
        // abort
}
```

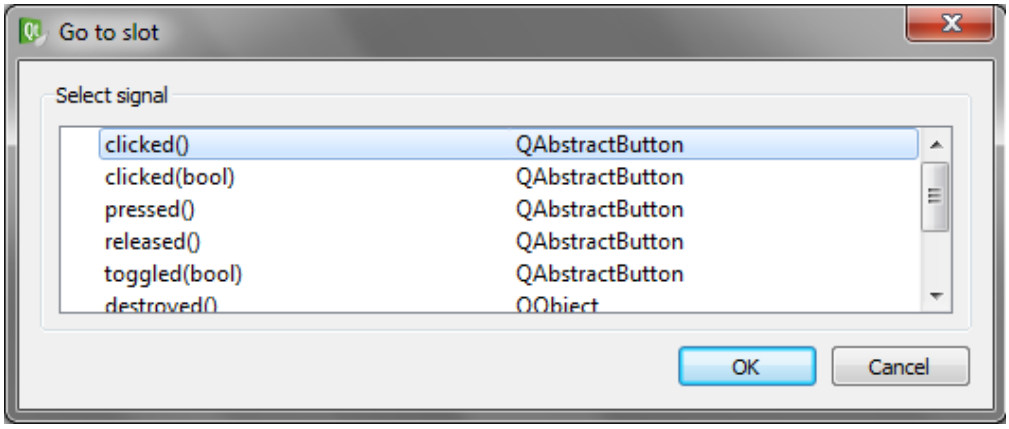
## Button Message Dialog

Kali ini kita akan mencoba membuat sebuah tombol yang jika di klik/tekan akan menampilkan kotak pesan tertentu.

Yang Anda butuhkan adalah membuat sebuah tombol pada GUI dengan menambahkan slot clicked(). Tampilan gui seperti berikut.




Klik kanan pada tombol, kemudian pilih Go to slot… kemudian pilih slot clicked()



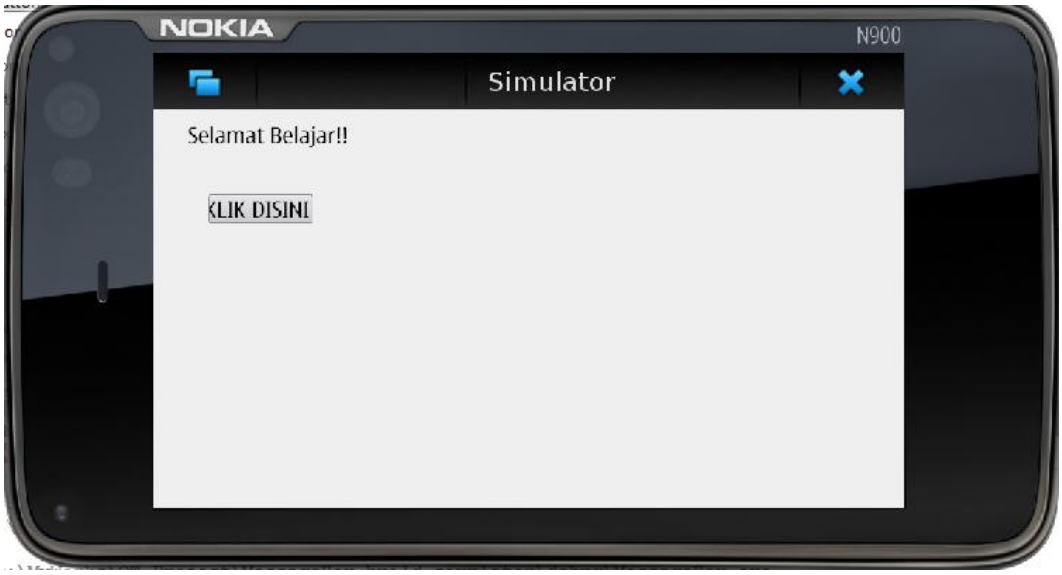

Selanjutnya akan masuk ke mode edit, tepatnya pada file mainwindow.cpp. Anda cukup menambahkan beberapa baris kode untuk menampilkan kotak pesan.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>

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

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

void MainWindow::on_pushButton_clicked()
{
    QMessageBox msg;
    msg.setText("Selamat Belajar!!");
    msg.exec();
}
```

Pada kode program di atas terdapat fungsi Message box yang berfungsi untuk menampilkan kotak pesan dan menambahkan slot clicked() pada tombol pushButton.

```
void MainWindow::on_pushButton_clicked()
{
    QMessageBox msg;
    msg.setText("Selamat Belajar!!");
    msg.exec();
}
```

Jangan lupa untuk menambahkan library QMessageBox pada file mainwindow.ccp. Jalankan aplikasi.



# Qt Style Sheets

Qt Style Sheets adalah mekanisme yang memungkinkan Anda untuk menyesuaikan tampilan widget, selain apa yang sudah dimungkinkan oleh subclassing QStyle. Konsep, terminologi, dan sintaks Qt Style Sheets yang sangat terinspirasi oleh HTML Cascading Style Sheets (CSS) tetapi disesuaikan dengan widget.

Style Sheets adalah spesifikasi tekstual yang dapat diatur pada seluruh aplikasi menggunakan QApplication:: setStyleSheet () atau pada widget tertentu (dan anak-anak nya) dengan menggunakan QWidget:: setStyleSheet (). Jika beberapa style sheet ditetapkan pada tingkat yang berbeda, Qt berasal style sheet efektif dari semua yang ditetapkan. Ini disebut cascading.

Sebagai contoh, style sheet berikut ini menetapkan bahwa semua QLineEdits harus menggunakan kuning sebagai warna latar belakang mereka, dan semua QCheckBoxes harus menggunakan warna merah sebagai warna teks:

```
QLineEdit { background: yellow }
QCheckBox { color: red }
```

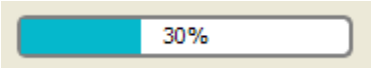
Untuk jenis kustomisasi, style sheet jauh lebih kuat daripada QPalette. Sebagai contoh, mungkin akan tergoda untuk mengatur QPalette::Button berwarna merah untuk QPushButton untuk mendapatkan sebuah tombol push merah. Namun, ini tidak dijamin untuk bekerja untuk semua gaya, karena gaya akan dibatasi oleh platform yang berbeda.

Style sheet membiarkan Anda melakukan segala macam kustomisasi yang sulit atau tidak mungkin untuk melakukan menggunakan QPalette saja. Style sheet adalah jawabannya.

Selain itu, style sheet dapat digunakan untuk memberikan tampilan dan nuansa berbeda untuk aplikasi Anda, tanpa harus subclass QStyle. Sebagai contoh, Anda dapat menentukan gambar sewenang-wenang untuk tombol radio dan kotak centang untuk membuat mereka menonjol. Dengan menggunakan teknik ini, Anda juga bisa mencapai kustomisasi kecil yang biasanya membutuhkan beberapa gaya subclassing, seperti menentukan sedikit gaya.

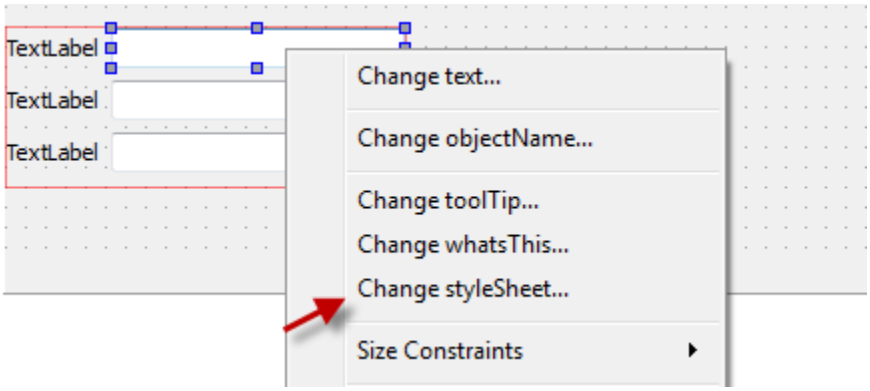
Contoh penggunaan Style Sheet pada Progress bar. Misalnya, kita mengubah perbatasan menjadi abu-abu dan isi dari progress bar berwarna biru langit.

```
QProgressBar {
    border: 2px solid grey;
    border-radius: 5px;
}
QProgressBar::chunk {
    background-color: #05B8CC;
    width: 20px;
}
```

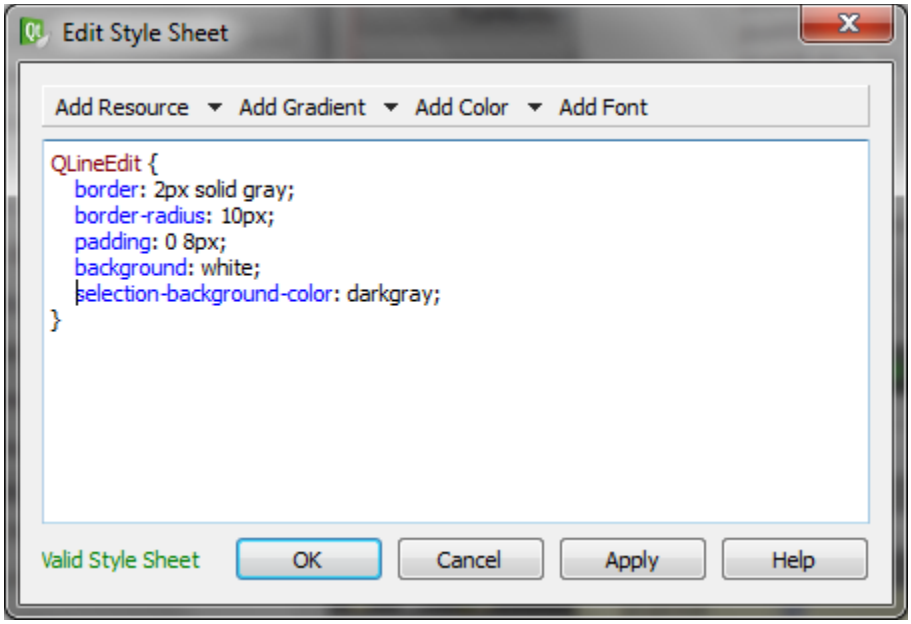



## Qt Designer Integration

Qt Designer adalah alat yang sangat baik untuk pratinjau style sheet. Anda dapat klik kanan pada setiap widget di Designer dan **Change styleSheet...** untuk mengatur style sheet.




Anda dapat merubah style seperti halnya membuah sebuah CSS.



# Input/Output

Kebutuhan untuk membaca dari atau menulis ke file atau perangkat lain yang umum bagi hampir setiap aplikasi. Qt menyediakan dukungan yang bagus untuk I/O melalui `QIODevice`, suatu abstraksi yang kuat yang mengenkapsulasi "perangkat" mampu membaca dan menulis blok byte. Qt termasuk subclass `QIODevice` berikut:

| Kelas | Keterangan |
|---|---|
| **QFile** | Mengakses file dalam sistem file lokal dan embedded resource |
| **QTemporaryFile** | Menciptakan dan mengakses file-file sementara dalam sistem file lokal |
| **QBuffer** | Dibaca data dari atau menulis data ke `QByteArray` |
| **QProcess** | Menjalankan program eksternal dan menangani komunikasi antar proses |
| **QTcpSocket** | Transfer aliran data melalui jaringan menggunakan TCP |
| **QUdpSocket** | Mengirim atau menerima datagram UDP melalui jaringan |
| **QSslSocket** | Transfer aliran data yang dienkripsi melalui jaringan menggunakan SSL / TLS |

`QProcess`, `QTcpSocket`, `QUdpSocket`, dan `QSslSocket` adalah perangkat berurutan, yang berarti bahwa data dapat diakses hanya sekali, mulai dari byte pertama dan maju seri ke byte terakhir. `QFile`, `QTemporaryFile`, dan `QBuffer` adalah acak-mengakses perangkat, sehingga byte dapat dibaca beberapa kali dari posisi apapun, mereka menyediakan fungsi `QIODevice::seek()` untuk memposisikan pointer file.

Selain kelas-kelas device, Qt juga menyediakan dua kelas stream tingkat yang lebih tinggi yang dapat kita gunakan untuk membaca, dan menulis untuk, setiap I/O device: `QDataStream` untuk data biner dan `QTextStream` untuk teks. Kelas-kelas ini mengurus masalah seperti byte pemesanan dan pengkodean teks, memastikan bahwa Qt aplikasi yang berjalan pada platform yang berbeda atau di tempat yang berbeda dapat membaca dan menulis file masing-masing. Hal ini membuat saya Qt I/O jauh lebih nyaman dari kelas yang sesuai standar kelas C + +, yang meninggalkan isu-isu ini dengan pemrogram aplikasi.

QFile memudahkan untuk mengakses file individu, apakah mereka berada dalam sistem file atau tertanam dalam aplikasi dieksekusi sebagai sumber daya. Untuk aplikasi yang perlu mengidentifikasi set seluruh file untuk bekerja, Qt menyediakan kelas `QDir` dan `QFileInfo`, yang menangani direktori dan memberikan informasi tentang file-file di dalamnya.

Kelas QProcess memungkinkan kita untuk meluncurkan program eksternal dan berkomunikasi dengan mereka melalui masukan standar, standard output, dan saluran standar error (`cin`, `cout`, and `cerr`). Kita dapat mengatur variabel lingkungan dan direktori kerja yang eksternal akan digunakan

aplikasi. Secara default, komunikasi dengan proses asinkron (non-blocking), tetapi juga memungkinkan untuk memblokir pada operasi tertentu.

Network dan Membaca dan menulis XML topik besar seperti yang kita bahas secara terpisah dalam bab-bab selanjutnya. ☺

## Membaca dan Menulis Data Biner

Cara paling mudah untuk memuat dan menyimpan data biner dengan Qt adalah untuk instantiate `QFile`, untuk membuka file tersebut, dan untuk mengaksesnya melalui objek `QDataStream`. `QDataStream` menyediakan format penyimpanan platform-independen yang mendukung dasar C++ seperti `int` dan `double`, dan jenis berbagai data Qt, termasuk `QByteArray`, `QFont`, `QImage`, `QPixmap`, `QString`, dan `QVariant`, serta kelas wadah Qt seperti `QList<T>` dan `QMap<K, T>`.

Berikut ini bagaimana kita akan menyimpan integer, sebuah `QImage`, dan sebuah `QMap<QString, QColor>` dalam sebuah file bernama facts.dat:

```
QImage image("philip.png");
QMap<QString, QColor> map;
map.insert("red", Qt::red);
map.insert("green", Qt::green);
map.insert("blue", Qt::blue);
QFile file("facts.dat");
if (!file.open(QIODevice::WriteOnly)) {
    std::cerr << "Cannot open file for writing: "
              << qPrintable(file.errorString()) << std::endl;
    return;
}
QDataStream out(&file);
out.setVersion(QDataStream::Qt_4_3);
out << quint32(0x12345678) << image << map;
```

Jika kita tidak bisa membuka file, kami menginformasikan kepada user dan kembali. `qPrintable()` macro kembali dan `const char *` untuk QString. (Pendekatan lain akan menggunakan `QString::toStdString()`, yang mengembalikan `std::string`, yang `<iostream.h>` memiliki `<<` kelebihan).

Jika file berhasil dibuka, kita membuat QDataStream dan menetapkan nomor versinya. Nomor versi adalah sebuah integer yang mempengaruhi cara tipe data Qt direpresentasikan (dasar C++ Tipe data selalu direpresentasikan dengan cara yang sama).

Untuk memastikan bahwa jumlah `0x12345678` ditulis sebagai sebuah integer unsigned 32-bit pada semua platform, kami dilemparkan ke `quint32`, jenis data yang dijamin akan tepat 32 bit. Untuk memastikan interoperabilitas `QDataStream` secara default standarisasi pada big-endian, hal ini dapat diubah dengan memanggil `setByteOrder()`.

Kita tidak perlu secara eksplisit menutup file tersebut, karena ini dilakukan secara otomatis jika variabel QFile keluar dari ruang lingkup. Jika kita ingin memastikan bahwa data tersebut telah benar-benar telah ditulis, kita bisa memanggil `flush()` dan memeriksa nilai pengembaliannya (`true` pada sukses).

Kode untuk membaca kembali cermin data kode yang kita digunakan untuk menulis itu:

```
quint32 n;
QImage image;
QMap<QString, QColor> map;
QFile file("facts.dat");
if (!file.open(QIODevice::ReadOnly)) {
    std::cerr << "Cannot open file for reading: "
              << qPrintable(file.errorString()) << std::endl;
    return;
}
QDataStream in(&file);
in.setVersion(QDataStream::Qt_4_3);
in >> n >> image >> map;
```

Versi `QDataStream` kita gunakan untuk membaca adalah sama dengan yang kita digunakan untuk menulis. Ini harus selalu terjadi. Dengan hard-coding nomor versi, kita menjamin bahwa aplikasi selalu dapat membaca dan menulis data (dengan asumsi dikompilasi dengan Qt versi yang lebih baru).

QDataStream menyimpan data sedemikian rupa sehingga kita dapat membaca kembali. Sebagai contoh, sebuah `QByteArray` direpresentasikan sebagai jumlah byte 32-bit diikuti oleh byte sendiri. `QDataStream` juga dapat digunakan untuk membaca dan menulis byte, tanpa menghitung byte header, menggunakan `readRawBytes()` dan `writeRawBytes()`.

Penanganan kesalahan ketika membaca dari `QDataStream` cukup mudah. Arus memiliki `status()` nilai yang dapat `QDataStream::Ok`, `QDataStream::ReadPastEnd`, atau `QDataStream::ReadCorruptData`. Setelah kesalahan telah terjadi, operator >> selalu membaca nol atau nilai kosong. Ini berarti bahwa kita dapat membaca seluruh berkas tanpa khawatir tentang potensi kesalahan dan memeriksa `status()` Nilai akhir untuk melihat apakah apa yang kita baca.

`QDataStream` menangani berbagai C++ dan tipe data Qt; daftar lengkap tersedia di http://doc.qt.nokia.com/4.7/datastreamformat.html. Kita juga dapat menambahkan dukungan untuk jenis kita sendiri kustom oleh overloading operator `<<and>>`. Berikut definisi dari tipe data kustom yang dapat digunakan dengan QDataStream:

```
class Painting
{
public:
    Painting() { myYear = 0; }
    Painting(const QString &title, const QString &artist, int
year) {
        myTitle = title;
        myArtist = artist;
        myYear = year;
    }
    void setTitle(const QString &title) { myTitle = title; }
    QString title() const { return myTitle; }
    ...
private:
    QString myTitle;
    QString myArtist;
    int myYear;
};
QDataStream &operator<<(QDataStream &out, const Painting
&painting);
QDataStream &operator>>(QDataStream &in, Painting &painting);
```

Berikut ini bagaimana kita akan melaksanakan operator <<:

```
QDataStream &operator<<(QDataStream &out, const Painting
&painting)
{
    out << painting.title() << painting.artist()
        << quint32(painting.year());
    return out;
}
```

Untuk output oainting, kita hanya ada dua output `QStrings` dan `quint32`. Pada akhir fungsi, kita kembali stream. Ini adalah C++ idiom umum yang memungkinkan kita untuk menggunakan rantai operator << dengan output stream. Sebagai contoh:

```
out <<painting1 <<painting2 <<painting3;
```

Pelaksanaan `operator>>()` adalah sama `dengan operator<<()`:

```
QDataStream &operator>>(QDataStream &in, Painting &painting)
{
    QString title;
    QString artist;
    quint32 year;
    in >> title >> artist >> year;
    painting = Painting(title, artist, year);
    return in;
}
```

Ada beberapa keuntungan untuk operator streaming menyediakan untuk tipe data kustom. Salah satunya adalah bahwa hal itu memungkinkan kita untuk wadah aliran yang menggunakan jenis kustom. Sebagai contoh:

```
QList<Painting> paintings = ...;
out << paintings;
```

Kita dapat membaca dalam kontainer dengan mudah:

```
QList<Painting> paintings;
in >> paintings;
```

Ini akan menghasilkan kesalahan kompiler jika painting tidak mendukung `<<or>>`. Manfaat lain dari operator menyediakan streaming untuk jenis data adalah bahwa kita dapat menyimpan nilai dari tipe sebagai `QVariants`, yang membuat mereka lebih luas digunakan-misalnya, dengan `QSettings`. Ini bekerja dengan ketentuan bahwa kita mendaftar tipe menggunakan `qRegisterMetaTypeStreamOperators<T>()`.

Ketika kita menggunakan `QDataStream`, Qt membaca dan menulis masing-masing tipe, termasuk wadah dengan jumlah item. Hal ini membebaskan kita dari kebutuhan untuk struktur apa yang kita tulis dan dari melakukan setiap jenis parsing pada apa yang kita baca. Hanya kewajiban kita adalah untuk memastikan bahwa kita membaca semua jenis dalam urutan yang sama persis seperti yang kita tulis.

`QDataStream` berguna baik bagi kita format file aplikasi khusus dan untuk format biner standar. Kita dapat membaca dan menulis format biner standar menggunakan operator streaming pada tipe dasar (seperti `quint16` atau `float`) atau menggunakan `readRawBytes()` dan `writeRawBytes()`. Jika `QDataStream` sedang digunakan murni untuk membaca dan menulis dasar tipe data C++, kita bahkan tidak perlu memanggil `setVersion()`.

Sejauh ini, kita memuat dan menyimpan data dengan versi aliran hard-code sebagai `QDataStream::Qt_4_3`. Pendekatan ini sederhana dan aman, tetapi memiliki satu kekurangan kecil: Kita tidak dapat mengambil keuntungan dari format baru atau diperbarui. Misalnya, jika versi yang lebih baru Qt menambahkan atribut baru untuk `QFont` (selain ukuran point, family, dll) dan atribut tidak akan disimpan atau diambil. Ada dua solusi. Pendekatan pertama adalah untuk melekatkan nomor versi `QDataStream` di file:

```
QDataStream out(&file);
out << quint32(MagicNumber) << quint16(out.version());
```

(`MagicNumber` adalah konstanta yang secara unik mengidentifikasi jenis file.) Pendekatan ini memastikan bahwa kita selalu menulis data menggunakan versi terbaru `QDataStream`, apa pun yang kebetulan. kita membaca versi stream:

```
quint32 magic;
quint16 streamVersion;
QDataStream in(&file);
in >> magic >> streamVersion;
if (magic != MagicNumber) {
    std::cerr << "File is not recognized by this application"
              << std::endl;
} else if (streamVersion > in.version()) {
    std::cerr << "File is from a more recent version of the "
              << "application" << std::endl;
    return false;
}
in.setVersion(streamVersion);
```

Kita dapat membaca data selama versi aliran kurang dari atau sama dengan versi yang digunakan oleh aplikasi, jika tidak, akan melaporkan kesalahan.

Jika format file berisi nomor versi sendiri, kita dapat menggunakannya untuk menyimpulkan nomor versi aliran bukannya menyimpannya secara eksplisit. Sebagai contoh, mari kita anggap bahwa format file untuk versi 1.3 dari aplikasi kita. Kemudian kita bisa menulis data sebagai berikut:

```
QDataStream out(&file);
out.setVersion(QDataStream::Qt_4_3);
out << quint32(MagicNumber) << quint16(0x0103);
```

Ketika kita membaca kembali, kita menentukan versi `QDataStream` digunakan berdasarkan nomor versi aplikasi:

```
QDataStream in(&file);
in >> magic >> appVersion;

if (magic != MagicNumber) {
    std::cerr << "File is not recognized by this application"
              << std::endl;
    return false;
} else if (appVersion > 0x0103) {
    std::cerr << "File is from a more recent version of the "
              << "application" << std::endl;
    return false;
}

if (appVersion < 0x0103) {
    in.setVersion(QDataStream::Qt_3_0);
} else {
    in.setVersion(QDataStream::Qt_4_3);
}
```

Dalam contoh ini, kita menetapkan bahwa setiap file yang disimpan dengan versi sebelum 1.3 aplikasi tersebut menggunakan aliran data versi 4 (Qt_3_0), dan file yang disimpan dengan versi 1.3 versi aplikasi data menggunakan aliran 9 (Qt_4_3).

Singkatnya, ada tiga kebijakan untuk menangani versi `QDataStream`: hard-coding, secara eksplisit menulis dan membaca nomor versi, dan menggunakan nomor versi yang berbeda tergantung pada versi aplikasi. Setiap kebijakan ini dapat digunakan untuk memastikan bahwa data yang ditulis oleh versi lama dari suatu aplikasi dapat dibaca dengan versi yang baru, bahkan jika link versi baru terhadap versi yang lebih baru dari Qt. Setelah kita memilih kebijakan untuk menangani versi QDataStream, membaca dan menulis data biner menggunakan Qt ini sederhana dan dapat diandalkan.

Jika kita ingin membaca atau menulis file dalam satu waktu, kita dapat menghindari menggunakan `QDataStream` dan sebagai gantinya menggunakan QIODevice `write()` dan fungsi `readAll()`. Sebagai contoh:

```
bool copyFile(const QString &source, const QString &dest)
{
    QFile sourceFile(source);
    if (!sourceFile.open(QIODevice::ReadOnly))
        return false;
    QFile destFile(dest);
    if (!destFile.open(QIODevice::WriteOnly))
        return false;
    destFile.write(sourceFile.readAll());
    return sourceFile.error() == QFile::NoError
           && destFile.error() == QFile::NoError;
}
```

Pada baris dimana `readAll()` dipanggil, seluruh isi dari file input yang dibaca ke dalam `QByteArray`, yang kemudian diteruskan ke fungsi `write()` yang akan ditulis ke file output. Setelah semua data dalam `QByteArray` memerlukan memori lebih dari membaca item dengan item, tapi ini menawarkan beberapa keuntungan. Sebagai contoh, kita kemudian dapat menggunakan `qCompress()` dan `qUncompress()` untuk kompres dan uncompress data. Sebuah alternatif memori kurang untuk menggunakan `qCompress()` dan `qUncompress()` adalah `QtIOCompressor` dari Qt Solutions. Sebuah `QtIOCompressor` kompres stream itu menulis dan dekompresi tanpa menyimpan seluruh file dalam memori.

Ada skenario lain di mana `QIODevice` mengakses secara langsung adalah lebih tepat daripada menggunakan `QDataStream`. QIODevice memberikan fungsi `peek()` yang mengembalikan byte data berikutnya tanpa memindahkan posisi perangkat serta fungsi `ungetChar()` yang "unreads" byte. Ini bekerja baik untuk perangkat random-access (seperti file) dan untuk perangkat sekuensial (seperti soket jaringan). Ada juga fungsi `seek()` yang menentukan posisi perangkat, untuk perangkat yang mendukung akses random.

# Qt Timer

Kelas QTimer menyediakan repetitif dan single-shot timer. Kelas QTimer menyediakan antar muka tingkat tinggi pemrograman untuk manajemen waktu. Untuk menggunakannya, membuat QTimer terhubung sinyal `timeout()` ke slot yang sesuai, dan panggil fungsi `start()`.

## Jam Digital dengan Qt LCD Number

Contoh Jam Digital menunjukkan cara menggunakan QLCDNumber untuk menampilkan nomor dengan digit seperti LCD.

Kelas DigitalClock menyediakan widget jam menunjukkan waktu dengan jam dan menit yang dipisahkan oleh tanda titik berkedip. Subclass `QLCDNumber` menerapkan slot `showTime()` untuk memperbarui tampilan jam:

```
DigitalClock::DigitalClock(QWidget *parent)
     : QLCDNumber(parent)
 {
     setSegmentStyle(Filled);
     QTimer *timer = new QTimer(this);
     connect(timer, SIGNAL(timeout()), this, SLOT(showTime()));
     timer->start(1000);
     showTime();
     setWindowTitle(tr("Digital Clock"));
     resize(150, 60);
}
```

Untuk menampilkan waktu realtime Anda dapat menggunakan `QTime::currentTime();` untuk mengambil waktu saat ini pada device.

Implementasinya cukup mudah, Anda cukup menaruh kode program berikut pada file header.

```
#ifndef DIGITALCLOCK_H
#define DIGITALCLOCK_H
#include <QtGui/QWidget>
#include <QLCDNumber>
class DigitalClock : public QLCDNumber
{
    Q_OBJECT
public:
    DigitalClock(QWidget *parent = 0);
private slots:
    void showTime();
};
#endif // DIGITALCLOCK_H
```

Jangan lupa untuk menyertakan library QLCDNumber. Kemudian pada file cpp masukan kode program berikut ini :

```
#include "digitalclock.h"
#include <QtGui>
 DigitalClock::DigitalClock(QWidget *parent)
     : QLCDNumber(parent)
 {
     setSegmentStyle(Filled);
     QTimer *timer = new QTimer(this);
     connect(timer, SIGNAL(timeout()), this,
SLOT(showTime()));
     timer->start(1000);
     showTime();
     setWindowTitle(tr("Digital Clock"));
     resize(150, 60);
 }
 void DigitalClock::showTime()
 {
     QTime time = QTime::currentTime();
     QString text = time.toString("hh:mm");
     if ((time.second() % 2) == 0)
         text[2] = ' ';
     display(text);
 }
```

Jika kode program dijalankan maka akan tampil seperti berikut.

## Jam Analog dengan QPainter

Contoh ini juga memperlihatkan bagaimana fitur transformasi dan skala dari QPainter dapat digunakan untuk membuat gambar custom widget lebih mudah.

Kelas AnalogClock menyediakan widget jam dengan jarum jam dan menit yang akan diperbarui secara otomatis setiap beberapa detik. Subclass QWidget dan reimplement `paintEvent()` berfungsi untuk menggambar tampilan jam:

```
class AnalogClock : public QWidget
 {
     Q_OBJECT
 public:
     AnalogClock(QWidget *parent = 0);
 protected:
     void paintEvent(QPaintEvent *event);
};
```

Berikut implementasi dari QPainter untuk membuat sebuah aplikasi jam analog.

Masukan kode program berikut ini pada file header.

```
#ifndef ANALOGCLOCK_H
#define ANALOGCLOCK_H

#include <QtGui/QWidget>

class AnalogClock : public QWidget
{
    Q_OBJECT
public:
    AnalogClock(QWidget *parent = 0);

protected:
    void paintEvent(QPaintEvent *event);
};

#endif // ANALOGCLOCK_H
```

Pada file header terdapat QPaintEvent yang berfungsi untuk menampilkan tampilan jam. Kemudian masukan kode program pada file cpp berikut ini.

```
#include <QtGui>
#include "analogclock.h"
AnalogClock::AnalogClock(QWidget *parent)
     : QWidget(parent)
{
     QTimer *timer = new QTimer(this);
     connect(timer, SIGNAL(timeout()), this, SLOT(update()));
     timer->start(1000);
     setWindowTitle(tr("Analog Clock"));
     resize(200, 200);
}
```

```
void AnalogClock::paintEvent(QPaintEvent *)
 {
     static const QPoint hourHand[3] = {
         QPoint(7, 8),
         QPoint(-7, 8),
         QPoint(0, -40)
     };
     static const QPoint minuteHand[3] = {
         QPoint(7, 8),
         QPoint(-7, 8),
         QPoint(0, -70)
     };
     QColor hourColor(127, 0, 127);
     QColor minuteColor(0, 127, 127, 191);
     int side = qMin(width(), height());
     QTime time = QTime::currentTime();
     QPainter painter(this);
     painter.setRenderHint(QPainter::Antialiasing);
     painter.translate(width() / 2, height() / 2);
     painter.scale(side / 200.0, side / 200.0);
     painter.setPen(Qt::NoPen);
     painter.setBrush(hourColor);
     painter.save();
     painter.rotate(30.0 * ((time.hour() + time.minute() /
60.0)));
     painter.drawConvexPolygon(hourHand, 3);
     painter.restore();
     painter.setPen(hourColor);
     for (int i = 0; i < 12; ++i) {
         painter.drawLine(88, 0, 96, 0);
         painter.rotate(30.0);
     }
     painter.setPen(Qt::NoPen);
     painter.setBrush(minuteColor);
     painter.save();
     painter.rotate(6.0 * (time.minute() + time.second() /
60.0));
     painter.drawConvexPolygon(minuteHand, 3);
     painter.restore();
     painter.setPen(minuteColor);
     for (int j = 0; j < 60; ++j) {
         if ((j % 5) != 0)
             painter.drawLine(92, 0, 96, 0);
         painter.rotate(6.0);
     }
 }
```

Kemudian jalankan program, maka akan tampil pada simulator :


N8
NOKIA
Simulator
Exit

# View Local Directory

Dir View menunjukkan tampilan struktur pohon ke sistem pengarsipan lokal. Ia menggunakan kelas QDirModel(obsolete) untuk menyediakan informasi file pada direktori.

Untuk melihat isi direktori lokal pada sebuah system kita cukup menggunakan `QFileSystemModel` ditambah `QTreeView` untuk memberikan tampilah pohon pada suatu direktori.

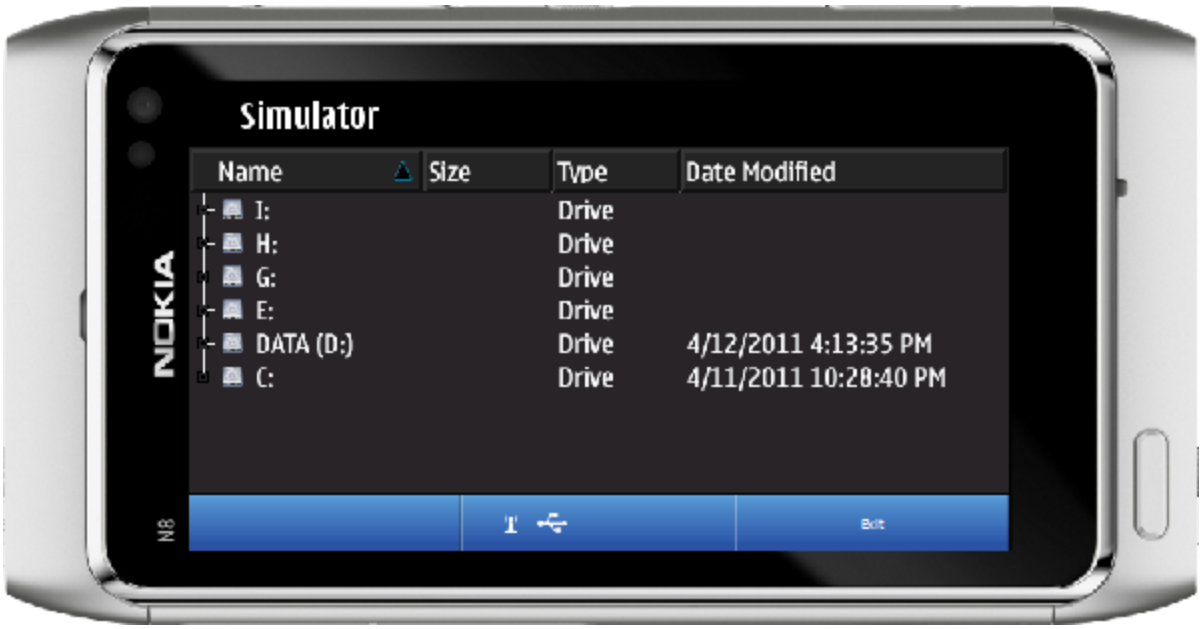
Contoh sederhananya, cukup menambahkan beberapa baris kode pada file utama.

```
#include <QtGui/QApplication>
#include <QFileSystemModel>
#include <QTreeView>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QFileSystemModel model;
    model.setRootPath("");
    QTreeView tree;
    tree.setModel(&model);

    // Tampilan Tree
    tree.setAnimated(false);
    tree.setIndentation(20);
    tree.setSortingEnabled(true);
    tree.setWindowTitle(QObject::tr("Dir View"));
    tree.resize(640, 480);
    tree.show();
    return app.exec();
}
```

Karena kita menggunakan kelas `QFileSystemModel`, maka kita juga hrus menambahkan library `QFileSystemModel`. Begitu juga dengan kelas QTreeView.

Pada kode program diatas, kita tidak menentukan default root folder ketika aplikasi dijalankan, ini terlihat pada `model.setRootPath("");`

Jika kode program dijalankan, akan tampil sebagai berikut:


Simulator
Name
Size
Type
Date Modified
I:
Drive
H:
Drive
G:
Drive
E:
Drive
DATA (D:)
Drive
4/12/2011 4:13:35 PM
C:
Drive
4/11/2011 10:28:40 PM
Exit
NOKIA
N8

# Qt Splash Screen

Splash Screen  adalah sebuah widget yang biasanya ditampilkan ketika aplikasi sedang dimulai. Splash layar sering digunakan untuk aplikasi yang lama pada saat start up (misalnya aplikasi database atau jaringan yang membutuhkan waktu untuk membangun hubungan) untuk memberikan informasi kepada pengguna bahwa aplikasi sedang loading.

Untuk membuat splash screen pada Qt, Anda cukup menggunakan kelas `QSplashScreen`. Berikut contoh kode program sederhana menggunakan QSPlashScreen.

```
#include <QtGui/QApplication>
#include <QSplashScreen>
#include <QPixmap>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
     QApplication app(argc, argv);
     QPixmap pixmap(":/images/splash.png");
     QSplashScreen splash(pixmap);
     splash.show();

     return app.exec();
}
```

Anda juga bisa menambahkan pesan loading saat splash screen berjalan, dengan menggunakan :

```
splash.showMessage("Please Wait...");
```

Untuk membuat splash screen menunggu sampai mainWin widget ditampilkan, Anda bisa menggunakan:

```
splash.finish(&window);
```

Splash screen ini akan tampil hanya sampai aplikasi utama benar-benar terbuka, jadi tergantung berapa lama aplikasi utama memulai start-up.

Berikut implememtasi dari penggabungan kode program:

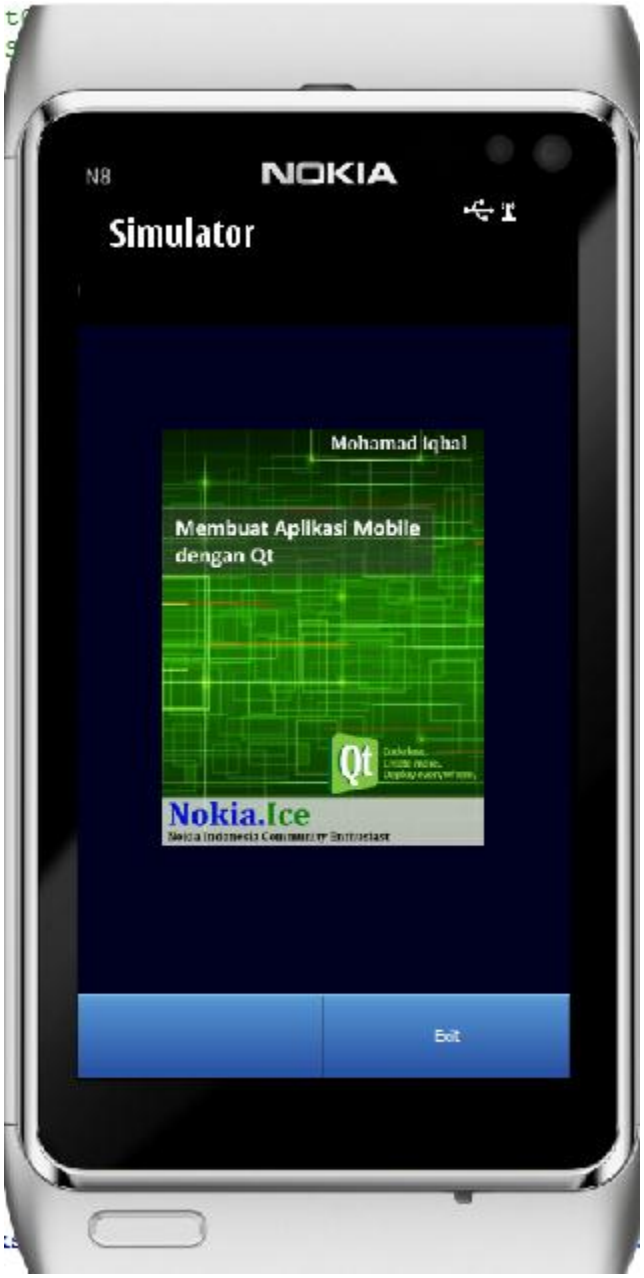
```
#include <QtGui/QApplication>
#include <QSplashScreen>
#include <QPixmap>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
     QApplication app(argc, argv);
     QPixmap pixmap(":/images/splash.png");
     QSplashScreen splash(pixmap);
     splash.show();

     splash.showMessage("Wait...");
     qApp->processEvents();//This is used to accept a click on
the screen so that user can cancel the screen

     QMainWindow window;
     window.setStyleSheet("* { background-color:rgb(199,147,88);
padding: 7px}");

     window.show();
     splash.finish(&window);
     return app.exec();
}
```

Pada kode program di atas akan tampil sebuah gambar splash.png denga tulisan Wait.. dan ketika aplikasi utama terbukua maka fungsi `QMainWindow` akan bekerja.

Terdapat baris kode `qApp->processEvents();` yang memungkinkan pengguna untuk menghentikan splash screen dan menutup aplikasi.

Anda juga bisa membuat image splash.png tampil full screen dengan menambahkan `showFullScreen()` (pembahasan ini telah di bahas pada bab layout)

Ketika kode program dijalankan, maka akan tampil :


N8
NOKIA
Simulator
Mohamad Iqbal
Membuat Aplikasi Mobile
dengan Qt
Nokia.Ice
Exit

# Database Qt

Fitur Qt diperkaya dengan fitur database yang dapat melakukan pengolahan data pada berbagai database server.

Salah satu fitur Qt adalah fitur pengolahan database yang sudah tersedia kelas-kelas untuk pengolahan database, berikut adalah beberapa kelas yang disediakan oleh Qt khusus untuk pengolahan database menggunakan Qt.

| Nama Kelas | Keterangan |
|---|---|
| **NQSql** | Digunakan untuk hal-hal umum Qt SQL |
| **QSqlDatabase** | Digunakan untuk melakukan koneksi ke database |
| **QSqlDriver** | Kelas abstrak untuk mengakses spesifik database |
| **QSqlDriverCreator** | Kelas template yang menyediakan SQL driver pada tipe spesifik database. |
| **QSqlDriverCreatorBase** | Kelas Dasar untuk semua SQL driver |
| **QSqlError** | Informasi error pada SQL database |
| **QSqlField** | Untuk manipulasi field didalam SQL database termasuk table dan view |
| **QSqlIndex** | Untuk manipulasi database index |
| **QSqlQuery** | Untuk melakukan query SQL statement |
| **QSqlQueryModel** | Data model yang bersifat read-only untuk hasil SQL query. |
| **QSqlRecord** | Enkapsulasi database record |
| **QSqlRelationalTableModel** | Data Model yang dapat diedit pada sebuah table database yang support foreign key |
| **QSqlResult** | Abstract interface untuk mengakses data dari spesifik SQL database |
| **QSqlTableModel** | Data model yang dapat diedit untuk sebuah table database. |

Secara umum. Library Qt sudah support untuk tiga layer untuk menangani pengolahan data database. Tiga layer tersebut meliputi:

**Driver layer**. Tujuan layer ini menyediakan penghubungan spesifik database dan SQL API layer. Kelas-kelas yang masuk dalam layer ini meliputi:

- QSqlDriver
- QSqlDriverCreator<T>
- QSqlDriverCreatorBase

- QSqlDriverPlugin
- QSqlResult

**SQL API layer**. Layer yang berfungsi untuk melakukan koneksi ke database dan berinteraksi. Kelas-kelas pada layer ini meliputi:

- QSIDDatabase
- QSqlQuery
- QSqlError
- QSqlField
- QSqlIndex
- QSqlRecord

**Antar muka layer**. Layer ini berfungsi untuk menghubungkan data dari database ke widget. Kelas-kelas pada layer ini meliputi:

- QSqlQueryModel
- QSqlTableModel
- QSqlRelationalTableModel

## Database yang Di-Support

Kita dapat menentukan database yang akan digunakan pada aplikasi Qt pada kelas QSqlDatabase. Beberapa database yang disupport oleh Qt dapat dilihat pada table di bawah ini :

| Tipe Driver | Keterangan |
|---|---|
| **QDB2** | IBM DB2 |
| **QIBASE** | Borlan Interbase Driver |
| **QMYSQL** | MySql Driver |
| **QOCI** | Oracle Call Interface Driver |
| **QODBC** | ODBC |
| **QPSQL** | PostgreSQL Driver |
| **QSQLITE** | SQLite versi 3 keatas |
| **QSQLITE2** | SQLite versi 2 |
| **QTDS** | Sybase Adaptive Server |

Contoh penggunaan dalam kode program untuk koneksi ke database MySql seperti di bawah ini:

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
```

Khusus untuk aplikasi Qt mobile, sebaiknya kita menggunakan database SQLite.

## Melakukan Koneksi ke Database

Kita dapat melakukan koneksi ke database melalui Qt memanfaatkan kelas QSqlDatabase. Pada saat instalasi objek QSqlDatabase, kita harus menentukan driver database yang digunakan. Secara umum, langkah-langkah untuk melakukan koneksi ke database sebagai berikut.

1. Menentukan tipe driver database yang digunakan
2. Mengisi parameter sesuai tipe database seperti nama database server, nama database, user name, dan password untuk mengakses database tersebut.
3. Panggil fungsi open() dari objek QSqlDatabase. Jika bernilai true maka kita dapat melakukan koneksi ke database dan sebaliknya.

Sebagai contoh, ilustrasi bagaimana kita melakukan koneksi ke database MySQL dapat dilihat pada kode program berikut ini.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");
db.setDatabaseName("namadatabase");
db.setUserName("User");
db.setPassword("Password");
if(!db.open())
{
    qDebug() << db.lastError();
    qFatal("Gagal melakukan koneksi.");
}
```

Contoh lain, apabila kita menggunakan database SQLite, misalkan nama file SQLite yaitu databaseku.db, ilustrasi kode program untuk melakukan koneksi ke database sebagai berikut.

```
QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
db.setDatabaseName("databaseku.db");
if(!db.open())
{
    qDebug() << db.lastError();
    qFatal("Gagal melakukan koneksi.");
}
```

### Melakukan Query

Setelah kita berhasil melakukan koneksi ke database, langkah selanjtnya kita akan melakukan query. Pengertian query disini adalah kita dapat mengambil, membuat, mengedit ataupun menghapus data dari atau ke database.

Contoh ilustrasi kode program ini misalnya bagaimana kita melakukan query data firstname dan lastname dari table employee sebagai berikut.

```
QSqlQuery query(db);
query.prepare("SELECT firstname, lastname FROM employees");

if(!query.exec())
    qCritical() << query.lastError();
```

Contoh lain, untuk kode program yang menghapus data yang nilai country-nya dalah UK.

```
QSqlQuery query(db);
query.prepare("DELETE employees WHERE country='UK'");

if(!query.exec())
    qCritical() << query.lastError();
```

Khusus untuk query yang tujuannya mengambil data dari database, kita harus menangkap kursor hasil query dengan memanfaatkan kelas QSqlRecord.

Contoh ilustrasi untuk menangkap hasil query dan menampilkan ke Qt debug (qDebug).

```
QSqlRecord rec = qry.record();
int jumlah_kol = rec.count();
QString lineData;
for(int i=0 c<jumlah_kol i++)
    lineEdit += rec.fieldName(i) + ((i<jumlah_kol-1)?"\t":"");
qDebug() << lineData;
while(qry.next())
{
    lineData = "";
    for(int i=0; c<cols; i++)
        lineData += qry.value(i).toString()+((i,jumlah_kol-
1)?"\t":"");

    qDebug() << lineData;
}
```

### Menampilkan Data

Untuk menampilkan data ke dalam aplikasi Qt, kita disediakan beberapa opsi, antara lain:

- List View
- Tree View
- Table View
- Column View

Pendekatan di atas dengan memanfaatkan Model-Based. Sedangkan untuk pendekatan Item-Based, kita memanfaatkan komponen Qt antara lain:

- List Widget
- Tree Widget
- Table Widget

Untuk ilustrasinya, misalkan kita menampilkan daftar kontak nama hanya ada satu table pada database SQLite yaitu contacts. Disini kita menampilkan data dengan memanfaatkan pendekatan View-Based melalui kelas QTreeView, misalkan nama variabelnya treeView. Mula-mulai kita deklarasikan objek model pada file header.

```
QSqlTableModel * model;
```

Disisi lain, untuk menampilkan ke QTreeView kita bisa melakukan query *select*. Berikut ini contoh kode programnya.

```
QTreeView *view = ui->treeView;
QSqlQuery query(db);
query.exec("CREATE TABLE contacts (name VARCHAR(32), phone
VARCHAR(16)");
model = new QSqlTableModel(this,db);
model->setTable("contacts");
model->setEditStrategy(QSqlTableModel::onFieldChange);
model->select();
view->setModel(model);
```

## Bekerja dengan Gambar

Kita juga dapat menyimpan gambar ke dalam database melalui Qt database. Untuk keperluan ini, misalakan variable gambar atau image bertipe QImage adalah sebuah gambar berformat PNG dan kita ingin melakukan penyimpanan ke dalam database PNG dan kita ingin pada field data kita dapat memanfaatkan QBuffer dan QImageWriter. Berikut ini contoh kode program.

```
QImage image;
QBuffer buffer;
QImageWriter writer(&buffer, "PNG");
writer.writer(image);
QSqlQuery qry;
int id;
qry.prepare("SELECT COUNT{*} FROM image");
qry.exec();
qry.next();
id = qry.value(0).toInt() +1;
qry.prepare("INSERT INTO images(id, data") VALUES (:id, :data)");
qry.bindValue(":id", id);
qry.bindvalue(":data", buffer.data());
qry.exec();
```

Sedangkan, untuk mengambil gambar dari table, kita dapat memanfaatkan QByteArray dan QImageReader untuk mengambil data BLOB gambar. Berikut ini contoh program implementasinya untuk mengambil data pada table images dengan id tertentu.

```
QSqlQuery qry;
qry.prepare("SELECT data FROM images WHERE id = :id");
qry.bindvalue(":id", id);
if(!qry.exec())
    qFatal("Failed to get image");
if(!qry.next())
    qFatal("Failed to get image");
QByteArray array = qry.value(0).toByteArray();
QBuffer buffer(&array);
buffer.open(QIODevice::ReadOnly);
QImageReader reader(&buffer, "PNG");
QImage image = reader.read();
```

# Webkit

WebKit adalah sebuah web open source content rendering dan editing yang pada awalnya diciptakan oleh KDE ('K' Desktop Environment) developer. WebKit sekarang digunakan sebagai dasar untuk browser web, termasuk Google Chrome, KDE Konqueror, dan Mac OSX's Safari, dan juga digunakan oleh sebagian web-enabled perangkat mobile.

QtWebKit bertujuan untuk menjadi standar compliant, dan mendukung semua teknologi standar web dan menyediakan Web browser engine yang mempermudah pembuatan konten embed dari World Wide Web ke dalam aplikasi Qt Anda. QtWebKit menyediakan fasilitas untuk rendering dari dokumen HyperText Markup Language (HTML), Extensible HyperText Markup Language (XHTML) dan Scalable Vector Graphics (SVG), ditata dengan menggunakan Cascading Style Sheets (CSS) dan ditulis dengan JavaScript.

Terdapat sebuah jembatan antara lingkungan eksekusi JavaScript dan model Qt object memungkinkan untuk kostum QObjects. Integrasi dengan modul Qt networking memungkinkan halaman Web dimuat dari Web, file sistem lokal atau bahkan sistem sumber daya Qt.

## Kelas Utama Webkit

| Kelas | Keterangan |
|---|---|
| **QWebElement** | Sebuah kelas untuk mengakses dan mengedit QWebFrame elemen DOM dengan jQuery seperti API (Qt 4.6) |
| **QWebFrame** | Sebuah objek data yang merupakan frame di halaman web |
| **QWebHistory** | History dari link yang dikunjungi terkait dengan QWebPage |
| **QWebHistoryItem** | Sebuah objek yang mewakili satu link dikunjungi dalam sebuah QWebHistory |
| **QWebPage** | Sebuah objek data yang merupakan halaman web |
| **QWebSettings** | Sebuah objek data yang memegang pengaturan yang digunakan oleh QwebFrame atau QWebPage |

| | |
|---|---|
| **QWebView** | Sebuah widget visualisasi dari sebuah QWebPage |

Selain menyediakan fitur rendering murni, dokumen HTML dapat dibuat dan diedit sepenuhnya oleh pengguna melalui penggunaan atribut `contenteditable` pada elemen HTML.

QtWebKit juga didasarkan pada Open Source WebKit engine.. Informasi lebih lanjut tentang WebKit sendiri dapat ditemukan pada situs web http://webkit.org/

Untuk menyertakan definisi kelas modul, gunakan perintah berikut:

```
#include <QtWebKit>
```

Untuk menghubungkan terhadap modul, tambahkan baris ini ke file .pro Anda:

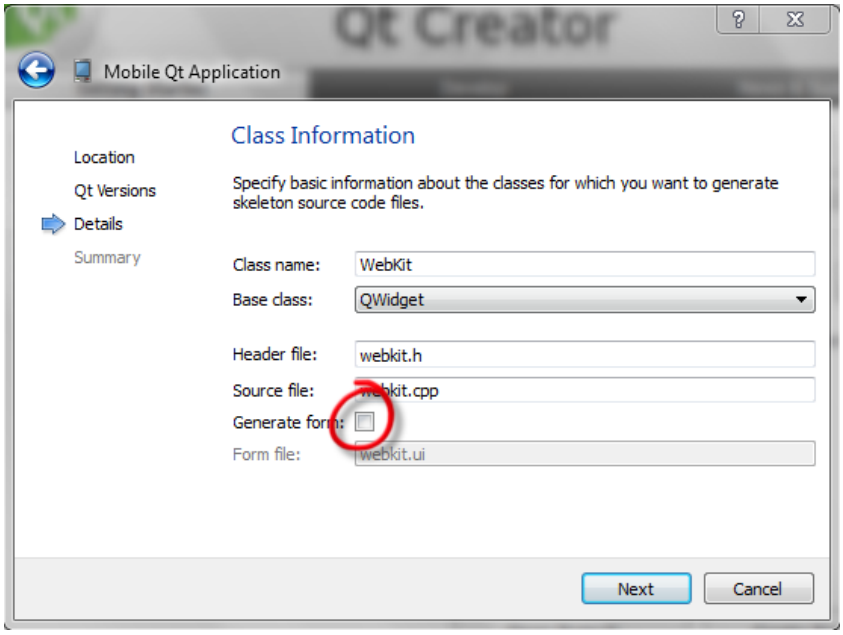
```
QT += webkit
```

Qt menyarankan untuk build modul hanya dalam modus rilis bukan debugging karena akan menimbulkan masalah dibeberapa platform.

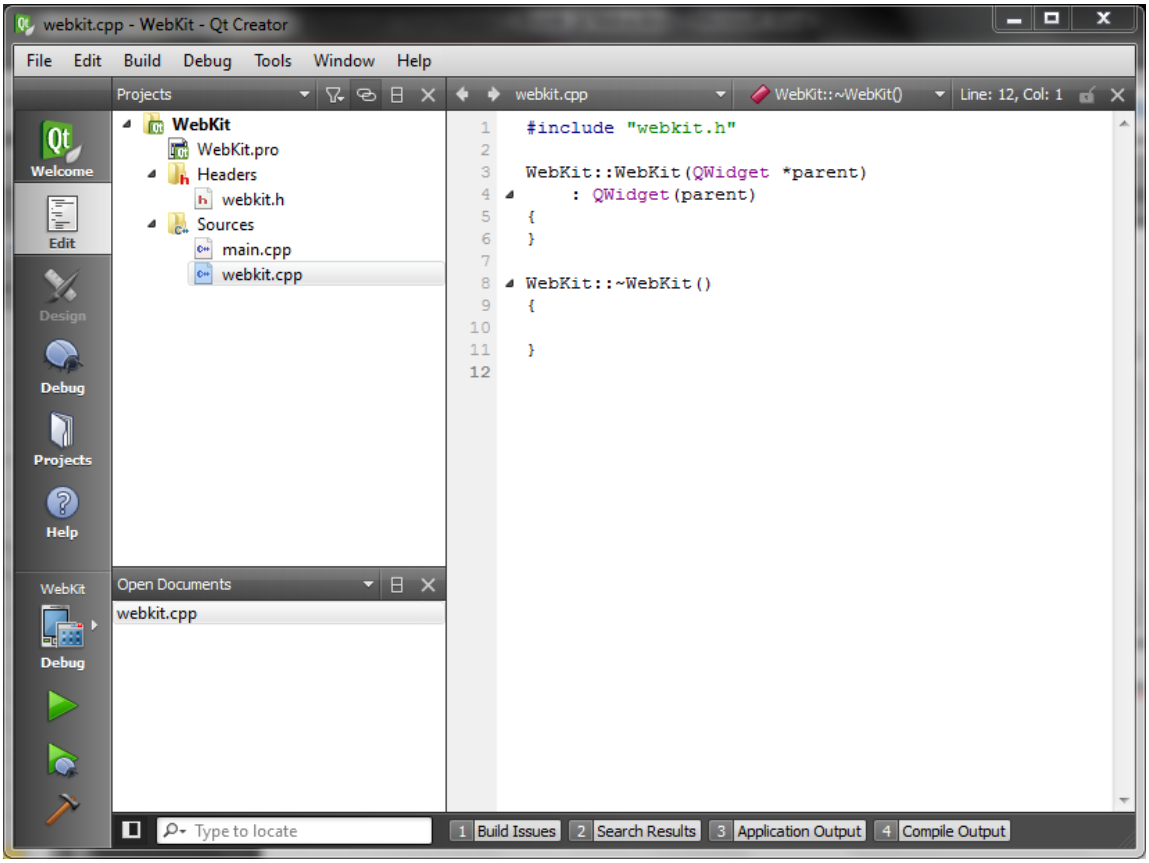
## Aplikasi Sederhana WebKit

Mari kita coba membuat sebuah project menggunakan WebKit.

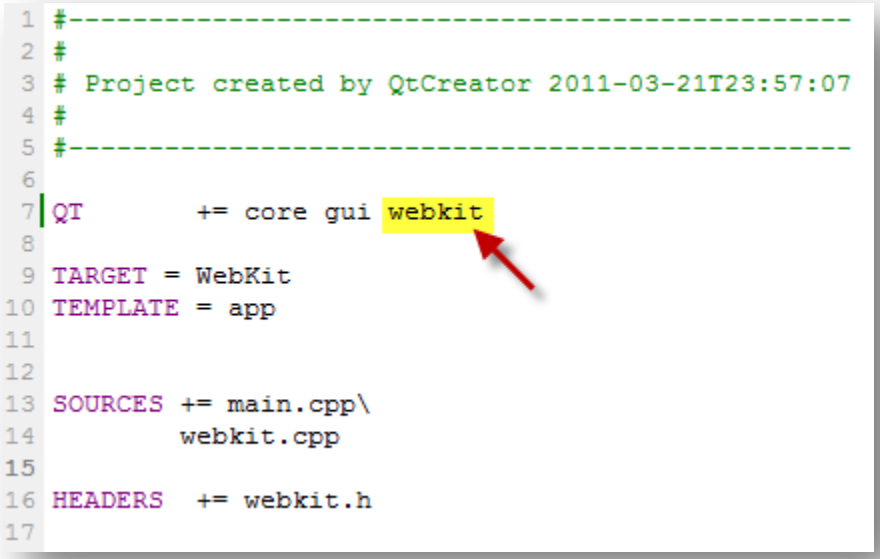
1. Buat project seperti biaya, namun pada Class Information, pada Base Class pilih QWidget (Qwidget adalah base class untuk semua user interface di Qt). Selanjutnya uncheck pada Generate form, karena kita akan memanfaatkan Qt form standard.




2. Sekarang Anda dapat melihat proyek Anda dalam Qt Creator, yang sekarang dalam mode Edit, seperti yang disorot di bawah ini:



3. Karena kita akan menggunakan widget QWebView, kita perlu menambahkan modul QtWebKit ke file proyek - WebKit.pro.



```
#-------------------------------------------------
#
# Project created by QtCreator 2011-03-21T23:57:07
#
#-------------------------------------------------

QT       += core gui webkit

TARGET = WebKit
TEMPLATE = app


SOURCES += main.cpp\
        webkit.cpp

HEADERS  += webkit.h

```

4. Anda juga akan melihat bahwa Qt Creator telah menciptakan bagian untuk pengaturan Symbian-spesifik. Mari kita kembali ke main.cpp, sehingga seluruh sumber aplikasi pertama kami akan berada di fungsi utama:

Jangan lupa untuk menambahkan Class QtWebkit/QWebView karena kita akan menggunakan object QWebView.

```
#include <QtGui/QApplication>
#include "webkit.h"
#include <QtWebKit/QWebView>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QWebView view;
        view.load(QUrl("http://www.google.co.id"));
#if defined(Q_WS_S60)
    view.showMaximized();
#else
    view.show();
#endif
    return a.exec();
}
```

Makro di atas memastikan bahwa jendela aplikasi akan tampil maksimal ketika kita build aplikasi untuk perangkat.

5. Yah, kita sudah selesai. Mari kita menjalankan proyek WebKit dan melihat apa yang tampak seperti di simulator:


Kita sudah berhasil membuat project webkit yang sangat sederhana, sekarang kita coba modifikasi sedikit dengan menambahkan URL bar agar pengguna dapat menginput alamat situs web yang diinginkan layaknya sebuah browser.

Kita akan mulai dengan mengubah fungsi utama: dalam contoh sebelumnya, semua kode sumber kami terletak dalam fungsi ini, kali ini kita akan membuat fungsi utama menampilkan widget utama dari aplikasi kita. Ini adalah pendekatan standar: fungsi utama hanyalah entry point aplikasi Anda dan semua logika yang sebenarnya diimplementasikan di widget atau di main window.

Buka main.cpp kembalikan seperti semua.

```
#include <QtGui/QApplication>
#include "webkit.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    WebKit w;
#if defined(Q_WS_S60)
    w.showMaximized();
#else
    w.show();
#endif
    return a.exec();
}
```

Selanjutnya pada webkit.h kita menambahkan beberapa baris kode untuk mendeklarasikan kelas-kelas yang terdapat pada project agar dapat menjalankan signals dan slots.

```
#ifndef WEBKIT_H
#define WEBKIT_H

#include <QtGui/QWidget>
class QWebView;
class QLineEdit;
class QVBoxLayout;

class WebKit:public QWidget
{
    Q_OBJECT
public:
    WebKit(QWidget *parent = 0);
    ~WebKit();
private slots:
    void openUrl();
    void onLoadFinished(bool finished);
private:
    QWebView* m_view;
    QLineEdit* m_lineEdit;
    QVBoxLayout* m_layout;
};
#endif // WEBKIT_H
```

Jadi, sekarang WebKit dapat menampilkan sebuah tampilan web, line edit (yang digunakan untuk URL bar) dan menggunakan QVBoxLayout untuk manajemen tata letap program.

Selanjutnya pada file webkit.cpp tambahkan beberapa baris kode berikut ini.

```
#include "webkit.h"
#include <QtGui/QLineEdit>
#include <QtWebKit/QWebView>
#include <QtGui/QVBoxLayout>

WebKit::WebKit(QWidget *parent)
    : QWidget(parent)
{
    m_lineEdit = new QLineEdit(this);
    m_lineEdit->setStyleSheet("background-color:white; padding: 6px;
color:blue");
    m_lineEdit->setPlaceholderText("Enter url ...");
    m_view = new QWebView(this);
    m_view->load(QUrl("http://blog.symbian.org"));
    m_layout = new QVBoxLayout;
    m_layout->addWidget(m_lineEdit);
    m_layout->addWidget(m_view);
    m_layout->insertSpacing(1,10);
    setLayout(m_layout);
    connect(m_lineEdit,SIGNAL(editingFinished()),SLOT(openUrl()));

connect(m_view,SIGNAL(loadFinished(bool)),SLOT(onLoadFinished(bool))
);
}
void WebKit::openUrl()
{
 QString url(m_lineEdit->text());
 if(!url.contains("http://",Qt::CaseInsensitive))
     url.prepend("http://");
 m_view->load(QUrl(url));
}
void WebKit::onLoadFinished(bool finished)
{
  if(finished){
     m_lineEdit->clear();
     m_lineEdit->setPlaceholderText(tr("Enter url ..."));
  }
}
WebKit::~WebKit()
{
}
```

Pada Line edit memiliki fungsi yang menarik (diperkenalkan pada Qt 4.7) untuk mengatur teks pengganti, jadi kita akan menggunakannya. Seperti yang Anda lihat, kita dapat merubah gaya line edit menggunakan sintaks CSS langsung dalam kode. Dalam contoh kita menggunakan warna latar belakang "white" dan warna teks untuk "blue". Untuk memperlajari fungsi “Styling ” (Kelas QStyle) anda bisa mempelajarinya di http://doc.qt.nokia.com/4.7/qstyle.html

Pada kode program di atas juga menujukan bahwa home page ketika aplikasi dijalankan akan me-load situs web http://nice.or.id.

Terkahir, kita ingin menempatkan kedua widget dalam tata letak vertikal, jadi kita bisa menambahkannya ke layout manager, kita juga menambahkan sedikit ruang antara dua widget dengan fungsi *insertSpacing*.

Dalam contoh, line edit memungkinkan pengguna untuk mengetikkan URL. Untuk melakukan ini, kita menggunakan fungsi *QObject::connect* dan terhubung ke slot di WebKit (*openUrl()*) ke sinyal di QLineEdit (*editingFinished ()*). Setiap line edit tersambung ke signals, WebKit akan menjalankan fungsi *openURL().*

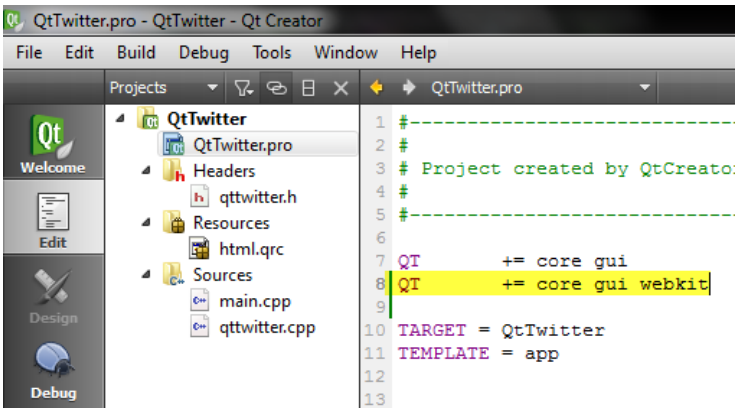
Demikian pula, kita juga menghubungkan WebKit:: onLoadFinished kesignals loadFinished yang sambungkan oleh QWebView.

Jika program dijalankan, maka akan tampil pada simulator seperti berikut.

## Aplikasi Twitter Sederhana dengan Webkit

Mari kita sedikit memodifikasi browser dengan membuat aplikasi twitter yang memungkinkan kita melihat timeline twitter seseorang. Kita cukup memanfaatkan widget timeline dari twitter dan disimpan pada sebuah file .html yang berisi script timeline twitter. ☺

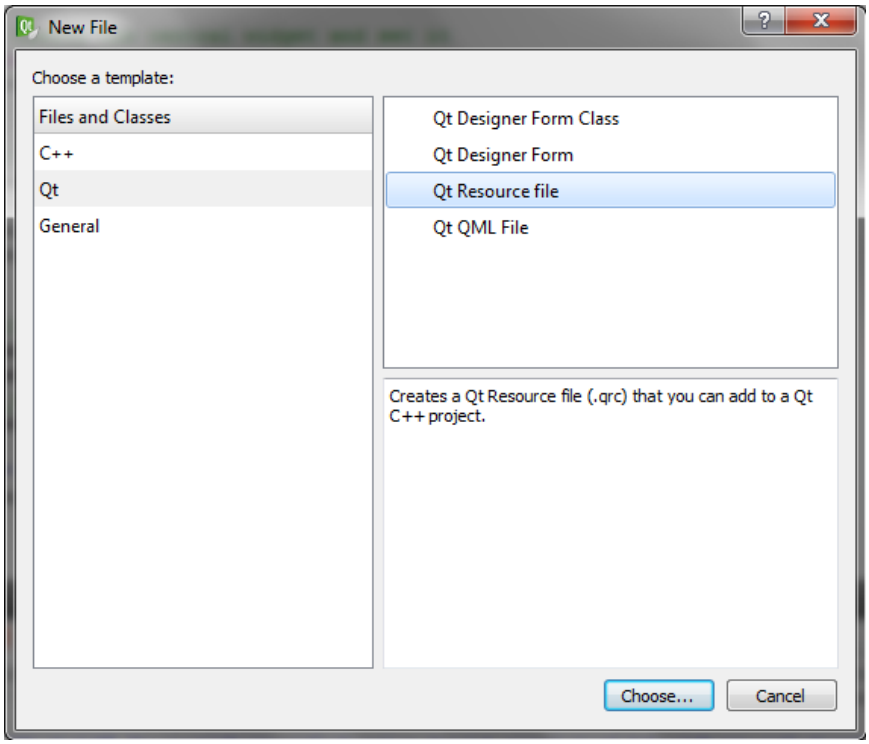
Ok mari kita mulai, buat proyek baru dengan name QtTwitter tanpa UI, kemudian pastikan file project Anda menembahkan modul QtWebkit sama seperti contoh sebelumnya.



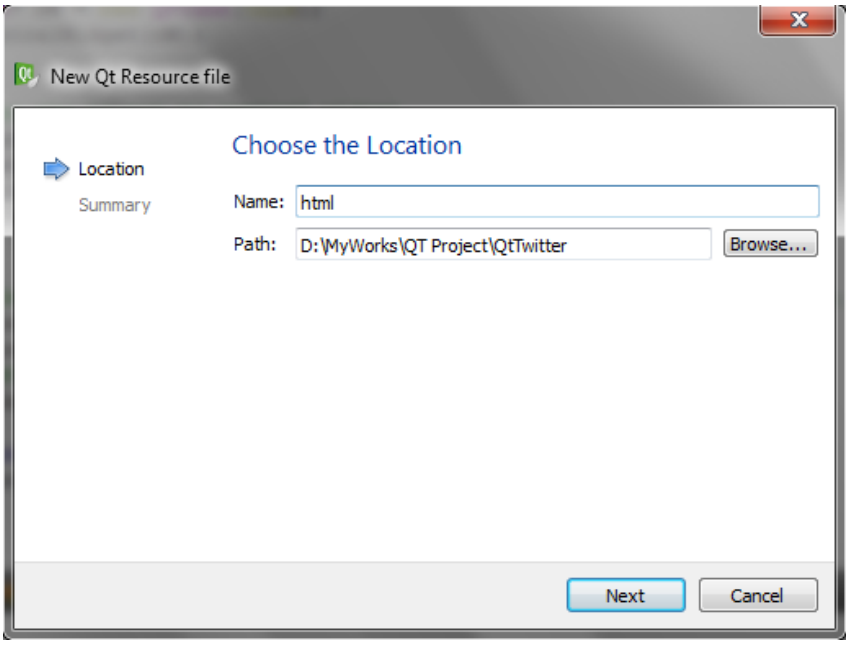

Selanjutnya pada file header cukup menambahkan kode program berikut ini.

```
#ifndef QTTWITTER_H
#define QTTWITTER_H

#include <QtCore/QPointer>
#include <QtWebKit/QWebView>
#include <QtWebKit/QWebPage>
#include <QtWebKit/QWebFrame>
#include <QtGui/QApplication>
#include <QtGui/QMainWindow>
#include <QtGui/QFrame>
#include <QtGui/QVBoxLayout>

class QtTwitter : public QMainWindow
{
    Q_OBJECT
public:
    QtTwitter(QWidget *parent = 0);
    ~QtTwitter();
private:
    QPointer<QWebView> _webView;
    QPointer<QVBoxLayout> _layout;
    void setupTwitter();
};

#endif // QTTWITTER_H
```

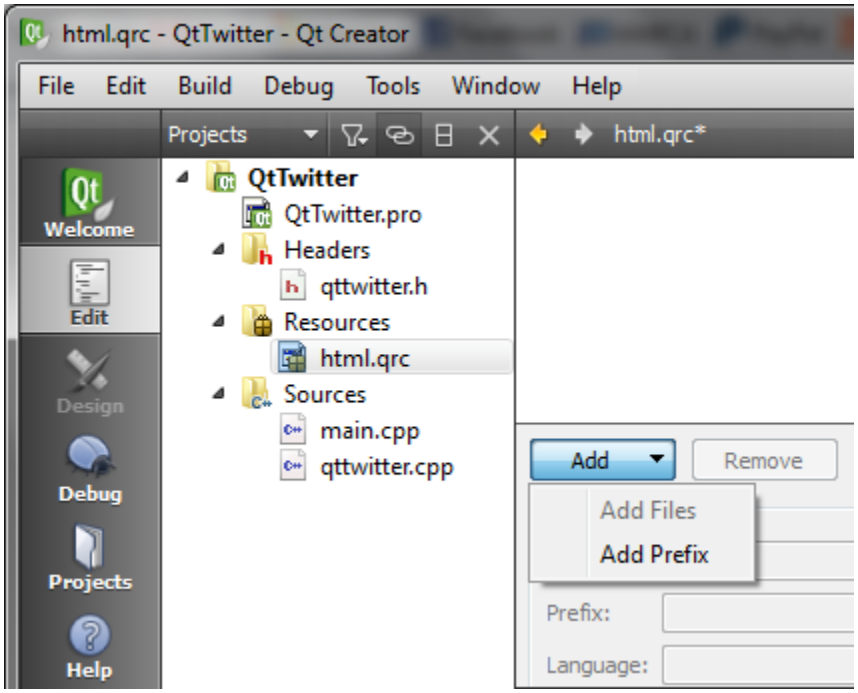
Kemudian kita buat Resource file, dengan meng-klik kanan pada project, kemudian Add New (caranya sama seperti membuat kelas baru pada project)



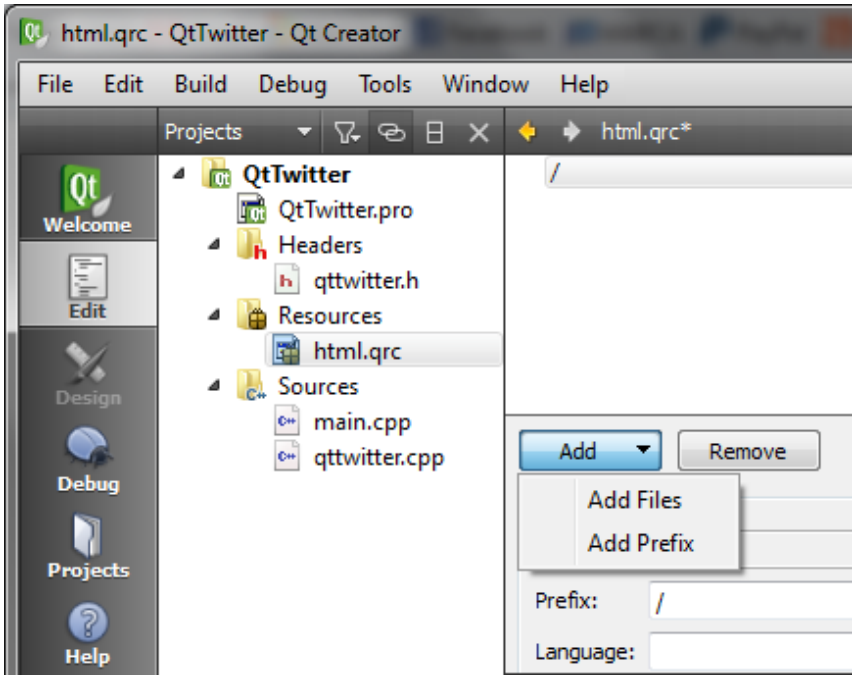

Kemudian pilih template Qt kemudian pilih Qt Resource file. Tentukan folder tempat file akan disimpan. Kemudian klik next.

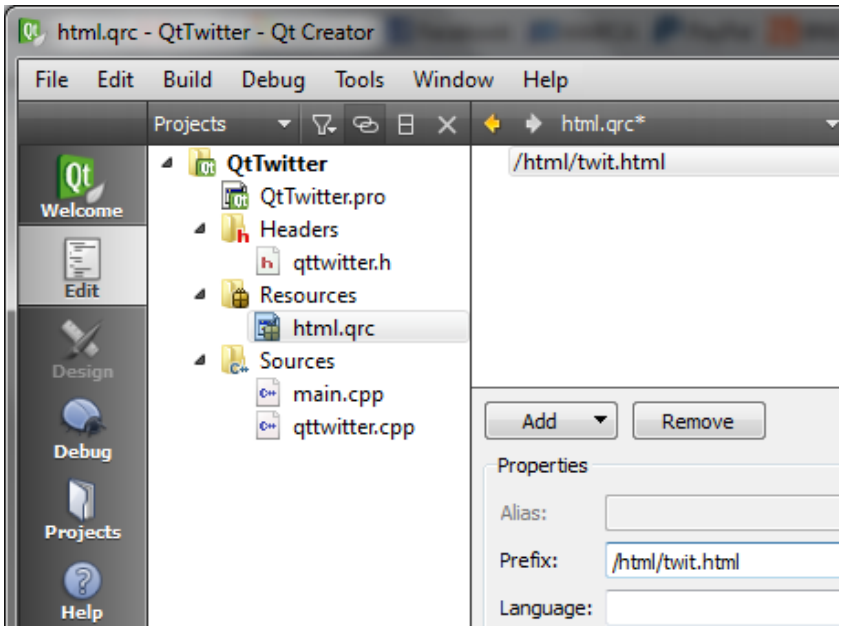


Jika sudah selesai, buat prefix baru.




Kemudian tambahkan file yang telah kita html yang telah kita buat.






Kemudian pada file .cpp kita masukan kode program untuk memanggil file dan membuat objek WebView baru.

```
#include "qttwitter.h"

QtTwitter::QtTwitter(QWidget *parent)
    : QMainWindow(parent)
{
    setupTwitter();
}

QtTwitter::~QtTwitter()
{
}

void QtTwitter::setupTwitter()
{
    // Membuat Central Widget
    QFrame* cW = new QFrame(this);
    setCentralWidget(cW);

    _layout = new QVBoxLayout(this);
    cW->setLayout(_layout);
    _layout->setMargin(0);
    _layout->setSpacing(0);

      // Me-load file .html
    _webView = new QWebView(this);
    _webView->load(QUrl("qrc:///html/twit.html"));
    _webView->show();

    _layout->addWidget(_webView);
}
```

`QUrl` berfungsi untuk memanggil file berisi situs web yang akan kita panggil. Pastikan path folder berisi file benar, karena jika salah kita tidak bisa memanggil file html tersebut.

Terakhir kita buat file html berisi script timeline twitter, saya ambil script original dari situs web http://twitter.com/about/resources/widgets/widget_profile

```
<script>
    new TWTR.Widget({
        version: 2,
        type: 'profile',
        rpp: 8,
        interval: 5000,
        width: 'auto',
        height: 640,
        theme: {
            shell: {
                background: '#2dc70e',
                color: '#f2f2f2'
            },
            tweets: {
                background: '#f0edf0',
                color: '#050405',
                links: '#eb1307'
            }
        },
        features: {
            scrollbar: false,
            loop: false,
            live: false,
            hashtags: true,
            timestamp: true,
            avatars: true,
            behavior: 'all'
        }
    }).render().setUser('ciebal').start();
</script>
```

Untuk mempercantik halaman web, saya memiliki template yang memungkinkan refresh timeline. Buat sebuah file .html dengan nama **twit.html** kemudian masukan kode program berikut ini.

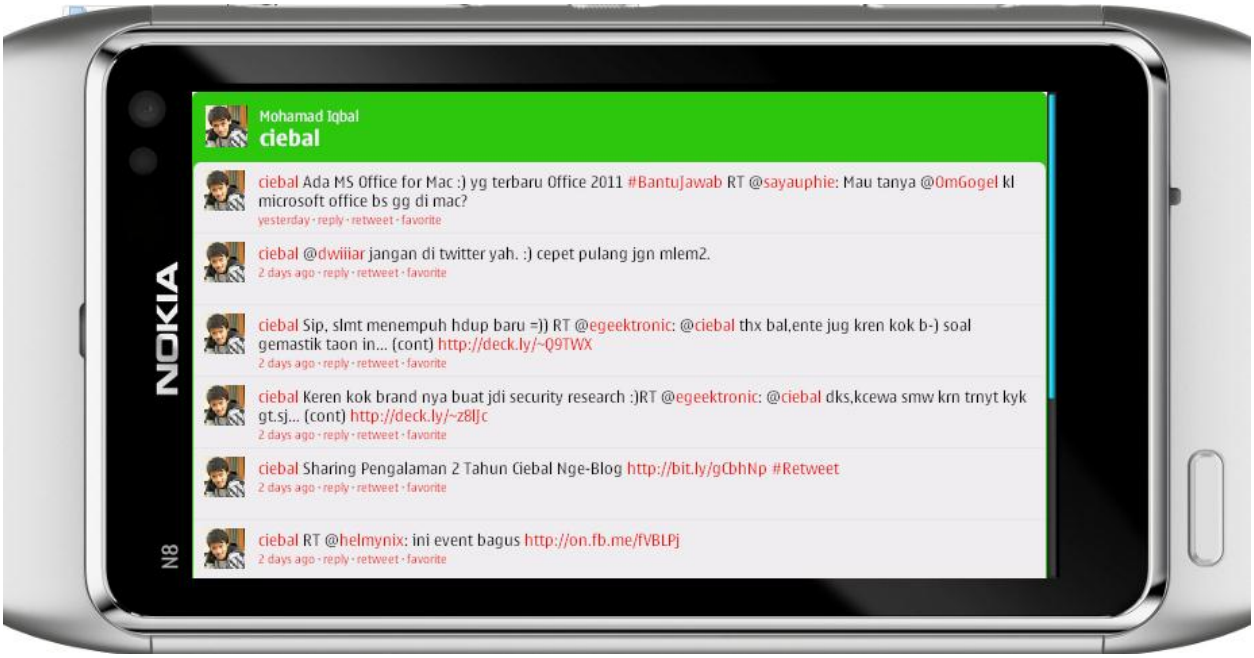
Catatan : Untuk memilih user yang ingin ditampilkan pada timeline, Anda cukup merubah script diatas pada baris `render().setUser('ciebal').start();`

Ubah **ciebal** menjadi nama user twitter sesuai keinginan Anda.

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no"
/>
<meta http-equiv="content-type" content="text/html; charset=UTF-
8"/>
<style type="text/css">
  html { height: 100% }
  body { height: 100%; margin: 0px; padding: 0px;font-size:medium;
}
#Button1
{
    width: 132px;
    height: 26px;
}
</style>
<title>Twitter</title>
</head>
<body>
<script src="http://widgets.twimg.com/j/2/widget.js"></script>
<script>
    new TWTR.Widget({
        version: 2,
        type: 'profile',
        rpp: 8,
        interval: 5000,
        width: 'auto',
        height: 640,
        theme: {
            shell: {
                background: '#2dc70e',
                color: '#f2f2f2'
            },
            tweets: {
                background: '#f0edf0',
                color: '#050405',
                links: '#eb1307'
            }
        },
        features: {
            scrollbar: false,
            loop: false,
            live: false,
            hashtags: true,
            timestamp: true,
            avatars: true,
            behavior: 'all'
        }
    }).render().setUser('ciebal').start();
</script>
<div align="center"><input id="Button1" type="button"
value="Refresh" onClick="window.location.reload()" />
</div>
</body>
</html>
```

Untuk menampilkan aplikasi secara Full Screen, seperti biasaya kita cukup menambahkan `showFullScreen()` pada file main.cpp

Mari kita coba jalankan aplikasi Twitter ini.




Terlihat cukup menarik di device nokia N8 atau touch screen, bagaimana dengan non touch screen?


Tidak begitu buruk. ☺

# Sistem Threading

Untuk mempercepat proses pekerjaan diperlukan pengerjaan secara parallel. Salah satu solusinya adalah dengan threading.

Teknologi Qt tidak hanya menyediakan solusi cross-platform dan tampilan GUI yang cantik tetapi Qt juga menyediakan solusi system threading.

| Nama Kelas | Keterangan |
|---|---|
| **NQAtomicInt** | Operasi integer yang atomic platform independent. |
| **QAtomicPointer** | Template class untuk menyediakan operasi pointer yang atomic platform independent |
| **QFuture** | Digunakan untuk menampung hasil komputasi asinkronus. |
| **QFutureSynchronizer** | Digunakan untuk sinkronisasi QFuture |
| **QFutureWatcher** | Untuk melakukan monitoring QFuture dengan memanfaatkan signal dan slots. |
| **QMutex** | Untuk mengakses serialization antar thread |
| **QMutexLocker** | Untuk melakukan locking dan unlocking |
| **QReadLocker** | Untuk melakukan locking dan unlocking untuk akses read |
| **QReadWriteLocker** | Untuk locking read dan write |
| **QRunnable** | Kelas untuk semua objek runnable |
| **QSemaphore** | Untuk semaphore |
| **QThread** | Untuk membuat thread yang platform independent |
| **QThreadPool** | Mengatur koleksi QThreads |
| **QWaitCondition** | Variabel kondisi untuk sinkronisasi thread |
| **QWaitLocker** | Untuk locking dan unlocking proses read-write locks |
| **QConcurrent** | High-level API untuk membuat program multi-thread |

Sistem threading bisa digunakan ketika kita ingin menjalankan suatu proses secara bersamaan sehingga dapat meningkatkan produktivitas.

Bahasa pemrograman C/C++ atau lainnya umumnya sudah men-support system threading. Sedangkan, pada teknologi Qt kita disediakan beberapa kelas yangd apat digunakan untuk menangani system threading. Berikut ini beberapa kelas yang digunakan untuk support system threading.

Untuk membuat thread pada Qt, kita dapat membuat suatu kelas turunan dari kelas QThread. Qt Creator ini dapat dilakukan ketika menambahkan C++ Class dan langsung menyebutkan turunan kelas QThread.

Setelah kita membuat kelas ini, kita akan memperoleh kode skeleton sebagai berikut.

```
#ifndef MYTHREAD_H
#define MYTHREAD_H
#include <QThread>

class MyThread : public QThread
{
    Q_OBJECT
public:
    explicit MyThread(QObject *parent = 0);

signals:

public slots:
};

#endif // MYTHREAD_H
```

Untuk ilustrasi ini, kita menambahkan fungsi baru bernama run(). Fungsi run() ini merupakan implementasi fungsi run() pada kelas QThread. Contoh realisasinya sebagai berikut.

```
#ifndef MYTHREAD_H
#define MYTHREAD_H
#include <QThread>

class MyThread : public QThread
{
    Q_OBJECT
public:
    explicit MyThread(QObject *parent = 0);
    void run();

signals:

public slots:
};

#endif // MYTHREAD_H
```

Sedangkan implementasi fungsi run() ditulis di file mythread.cpp. berikut ini realisasinya.

```
#include "mythread.h"
#include <QDateTime>
#include <iostream>
using namespace stf;

bool isStop = false;
MyThread::MyThread(QObject *parent) :
    QThread(parent)
{
}
void MyThread::run()
{
    while(!isStop)
    {
        QDateTime now = QDateTime::currentDateTime();
        cout << "Ini dari Run" << now.toString("dd.MM.yyyy
hh:mm:ss").toAscii().data() <<end1;
        sleep(1);
    }
}
```

Pada kode program di atas, kita membuat aplikasi yang menampilkan tulisan secara terus-menerus atau background.

Selanjutnya, untuk menggunakan program utama (main.cpp) cukup mudah. Berikut contoh ilustrasi kode programnya.

```
#include <QtCore/QCoreApplication>
#include "qthread.h"
#include <mythread.h>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    MyThread obj;
    obj.start();
    return a.exec();
}
```

Untuk menakhiri aplikasi ini, kita harus menutup aplikasi tersebut. Cara lebih halus adalah dengan memanggil fungsi wait() sepertic ontoh berikut ini.

```
#include <QtCore/QCoreApplication>
#include "qthread.h"
#include <mythread.h>
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    MyThread obj;
    obj.start();
    // berhenti
    obj.wait();
    return a.exec();
}
```

## Sinkronisasi Thread

Kadang kala, kita mempunyai beberapa thread yang salah satunya tergantung dari thread lain dan thread ini mengeksekusi sesuatu setelah thread lain melakukan hal tertentu. Dalam hal ini, prosesnya dikenal dengan sinkronisasi thread.

Pada Qt, kita dapat memanfaatkan mutex untuk melakukan locking an unlocking. Ilustrasi contoh sebelumnya kita dapat memanfaatkan mutec untuk sinkronisasi, berikut contoh modifilasi kode program.

```
#include "mythread.h"
#include <QDateTime>
#include <iostream>
using namespace stf;

bool isStop = false;
MyThread::MyThread(QObject *parent) :
    QThread(parent)
{
}
void MyThread::run()
{
    while(!isStop)
    {
        mutex.lock();
        if(isStop)
        {
            mutex.unlock();
            return;
        }

        QDateTime now = QDateTime::currentDateTime();
        cout << "Ini dari Run" << now.toString("dd.MM.yyyy
hh:mm:ss").toAscii().data() <<end1;
        sleep(1);

        mutex.unlock();
    }
}
```

Berdasarkan kode program di atas, mula-mula akan dilakukan locking dan pengecekan apakah nilai isStop bernilai TRUE. Jika bernilai TRUE, program akan berhenti.

# Qt Mobility APIs

Qt Mobility tidak hanya memungkinkan Qt developer untuk menyertakan fungsionalitas ponsel standar dalam aplikasi mereka, tetapi juga menyediakan fungsionalitas aplikasi yang berguna di seluruh platform desktop.

- Membuat aplikasi navigasi dinamis dengan lokasi API
- Dukungan mobile aplikasi standar seperti pesan, kontak dan multimedia
- Aktifkan jaringan roaming menggunakan manajemen pembawa

Proyek Qt Mobility mencakup beragam fitur dan teknologi. Ini bukan API tunggal tetapi kumpulan API atau kerangka kerja.

| API | Keterangan |
|---|---|
| **Bearer Management** | API untuk kontrol sistem konektivitas lokasi. |
| **Contacts** | API memungkinkan klien untuk meminta data kontak dari lokal atau remote. |
| **Document Gallery** | API untuk menavigasi dan query dokumen menggunakan meta-data. |
| **Feedback** | API memungkinkan klien untuk mengontrol getaran perangkat (bila vibrator yang digunakan) atau umpan balik dari layar. |
| **Location** | API Lokasi menyediakan library untuk penentuan posisi lokasi, pemetaan dan navigasi. |
| **Messaging** | API memungkinkan akses ke layanan pesan. |
| **Mobility QML Plugins** | Satu set QML plugin yang kompatibel untuk Proyek Mobilitas. |
| **Multimedia** | Menyediakan satu set API untuk bermain dan merekam media, dan mengelola koleksi konten media. |
| **Organizer** | API memungkinkan klien untuk permintaan kalender, jadwal dan data pribadi dari lokal atau remote. |
| **Publish and Subscribe** | Publish and Subscribe API, mengandung Nilai Space, |

| | |
|---|---|
| | memungkinkan aplikasi untuk membaca nilai item, menavigasi dan berlangganan untuk mengubah pemberitahuan. |
| **Qt Service Framework** | Satu set Qt API untuk yang memungkinkan klien untuk menemukan dan instantiate layanan sewenang-wenang. |
| **Sensors** | Sensor API memberikan akses ke sensor. |
| **System Information** | Satu set API untuk menemukan sistem informasi terkait. |
| **Versit** | Sebuah API untuk impor dan ekspor ke format vCard dan iCalendar. |

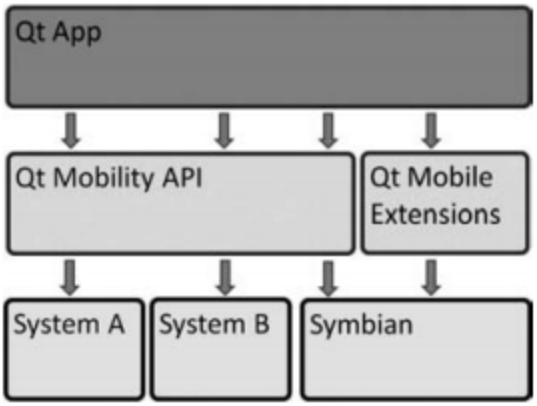
API ini memberikan berbagai fungsi pengembang yang diinginkan untuk platform mobile, tapi sekarang fungsi-fungsi ini menjadi mungkin pada platform tidak tradisional dihubungkan dengan beberapa fitur. Sebuah perangkat mobile yang menggunakan Proyek Qt Mobility API akan mampu melakukan :

- **Cari layanan on-board atau jarak jauh**. Cari koneksi optimal untuk jaringan untuk layanan tertentu.
- **Layanan dibangun di atas API** ini akan meliputi aplikasi internet seperti email dan browsing web.
- **Fitur multimedia** memungkinkan capture gambar dan video dengan audio, rekaman audio, dan memainkan musik atau klip video.
- **Lokasi** akan membuat perangkat tersebut tahu lokasi geografis untuk mendukung aplikasi GIS.
- **Publish-Subscribe** memungkinkan komunikasi antara objek terpisah apakah lokal atau jauh dari perangkat.

Qt API Ini akan digunakan oleh pengembang untuk membangun berbagai aplikasi dan layanan mobile yang mampu atau akrab bagi pengguna ponsel, terlepas dari apakah platform ini bergerak atau tidak.

## Kompatibilitas

Untuk peralatan berbasis Symbian dan Maemo 5, Qt Mobility sudah men-support semua fitur kecuali fitur feedback. Sedangkan platformnya menggunakan Qt 4.7 ke atas.




**Tabel Platform compatibility**

| | S60 3rdE FP1 S60 3rdE FP2 S60 5thE | Maemo 5 | Maemo 6 | Windows Mobile | linux | Mac |
|---|---|---|---|---|---|---|
| **Service Framework** | Yes | Yes | Yes | Yes | Yes | Yes |
| **Messaging** | Yes | No | Yes | Yes | Yes | No |
| **Bearer Management** | Yes | No | Yes | Yes | Yes | Yes |
| **Publish and Subscribe** | Yes | No | Yes | Yes | Yes | Yes |
| **Contacts** | Yes | No | Yes | Yes | No | No |
| **Location** | Yes | No | No | Yes | No | No |
| **Multimedia** | | No | No | | Yes | No |
| **System Information** | Yes | No | No | Yes | Yes | Yes |

## Instalasi

Library Qt mobility dapat diunduh secara gratis pada website Nokia dengan alamat :

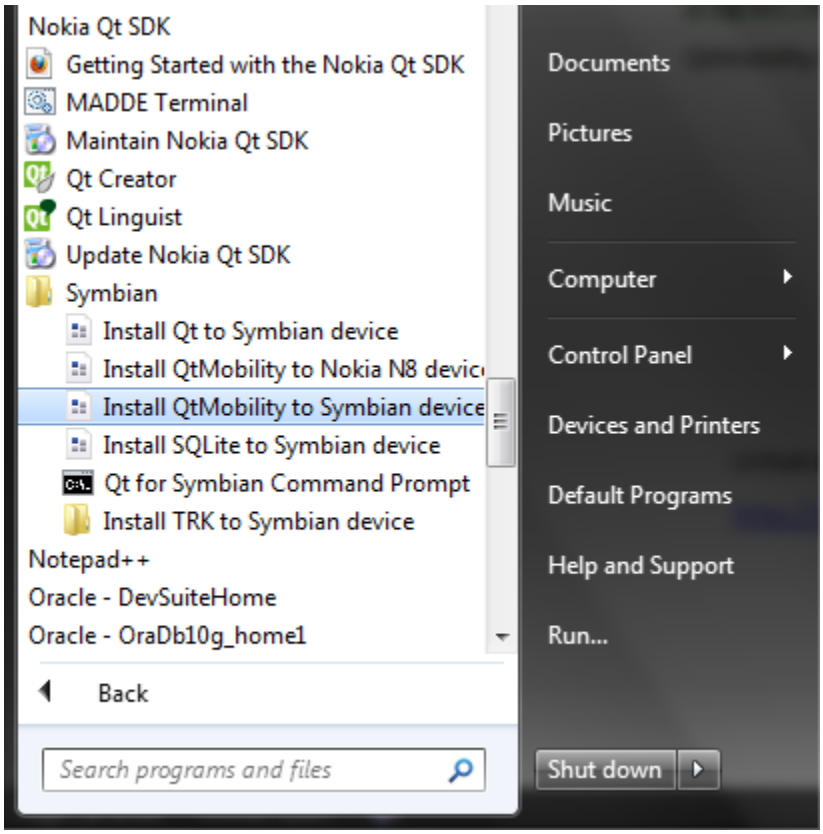
http://qt.nokia.com/products/qt-addons/mobility

Jika Anda menggunakan Qt SDK, Qt Mobility harus diinstal manual, namun apabila Anda menggunakan Nokia Qt SDK, Qt mobility API sudah terinstal didalamnya.

Pada Nokia Qt SDK 1.0, Nokia sudah menyediakan Qt Mobility versi 1.0.2 didalamnya.

Peralatan mobile yang akan dijalankan aplikasi berbasis Qt Mobility harus diinstal Qt Mobility. Caranya cukup mudah, peralatan mobile cukup dipasangkan melalui USB dan pilih salah satu platformnya melalui menu Nokia SDK - > Symbian -> Install QtMobility to Symbian device pada program menu utama. Untuk

Nokia N8, kita dapat memilihnya melalui Nokia Qt SDK -> Symbian -> QtMobility to Nokia N8 device. Contohnya lihat pada gambar berikut.



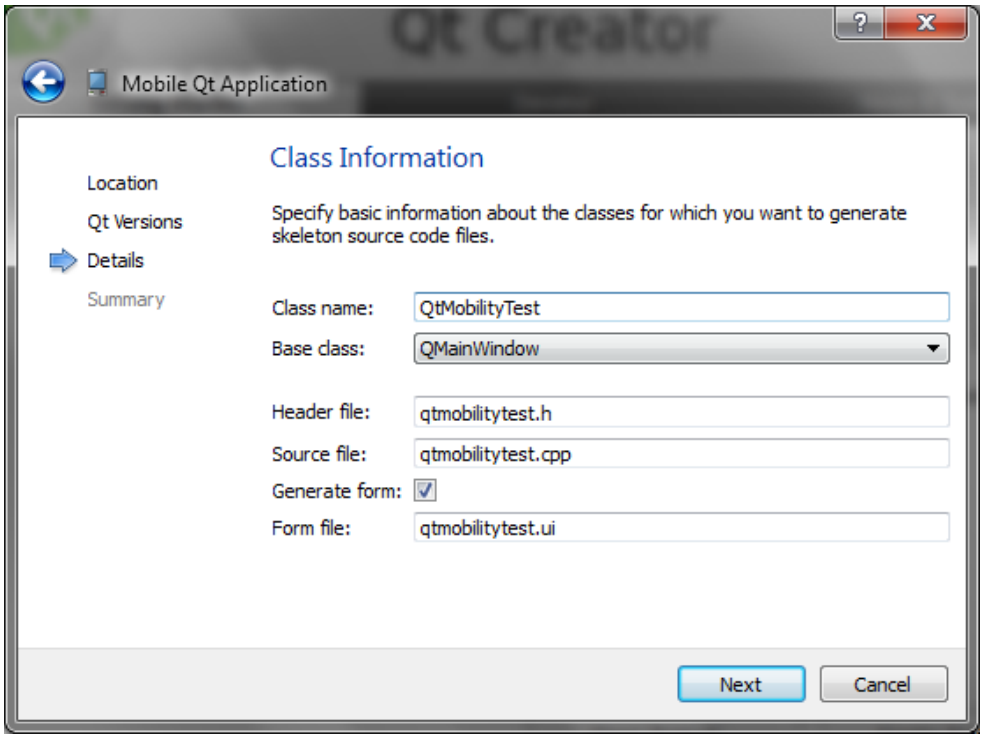
## API Sistem Informasi

Pada Qt Mobility untuk system informasi, sudah disediakan kelas-kelas yang dapat membantu Andauntuk memperoleh informasi mengenai suatu system. Kelas-kelas dalam kategori system informasi dapat dilihat pada table di bawah ini.

| Nama Kelas | Keterangan |
|---|---|
| **NQSystemDeviceInfo** | Memberikan informasi perangkat mobile |
| **QSystemDisplayInfo** | Memberikan informasi display |
| **QsystemInfo** | Memberikan informasi umum suatu system |
| **QSystemNetworkInfo** | Memberikan informasi jaringan |
| **QSystemScreenSaver** | Mengakses screen saver dan blanking |
| **QSystemStorageInfo** | Mengakses informasi memory dan disk |

Selanjutnya, saya akan memaparkan bagaimana memulai aplikasi Qt Mobility sederhana dengan menggunakan Nokia Qt SDK. fitur Qt Mobility yang digunakan adalah API system information yaitu QSystemInfo.

Untuk keperluan ini, kita membuat project baru bertipe Mobile Qt Application. Misalkan nama projectnya QtMobilityTest, nama kelas QtMobilityTest dan target pada Symbian dan Maemo.

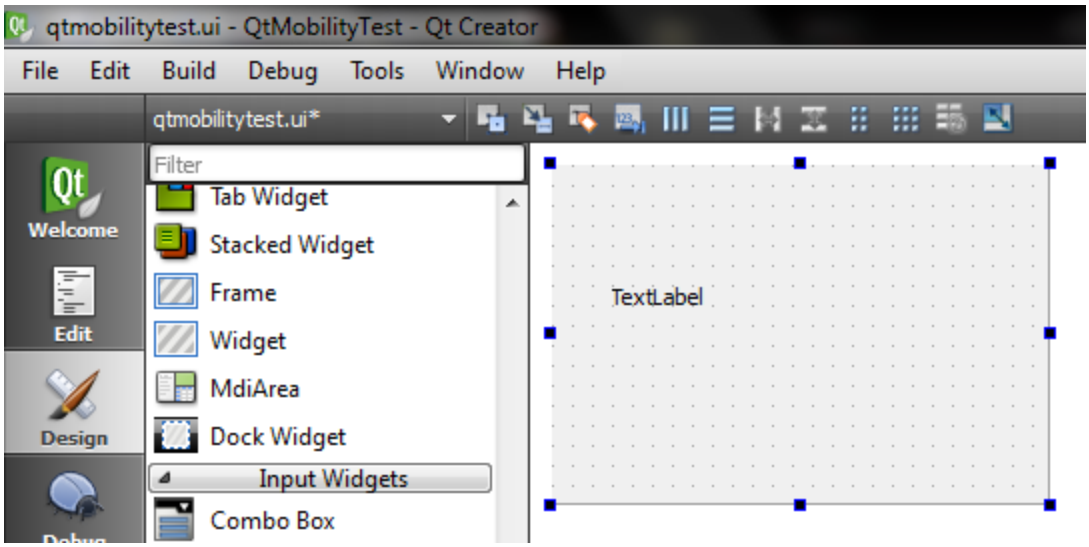


Setelah project terbuat, langkah pertama kita harus menambahkan systeminfo pada MOBILITY dibagian property projectnya.

Dengan menambahkan ini, kita dapat mengakses semua kelas Qt Mobility untuk system informasi. Berikut contoh file project QtMobilityTest.pro.

```
#-------------------------------------------------
#
# Project created by QtCreator 2011-03-29T19:40:34
#
#-------------------------------------------------

QT       += core gui

TARGET = QtMobilityTest
TEMPLATE = app


SOURCES += main.cpp\
        qtmobilitytest.cpp

HEADERS  += qtmobilitytest.h

FORMS    += qtmobilitytest.ui

CONFIG += mobility
MOBILITY = systeminfo

symbian {
    TARGET.UID3 = 0xe0db4899
    # TARGET.CAPABILITY +=
    TARGET.EPOCSTACKSIZE = 0x14000
    TARGET.EPOCHEAPSIZE = 0x020000 0x800000
}

```

Dibagian GUI, kita hanya menggunakan label. Misalkan nama variabelnya adalah label. Nilai label diambil dari informasi bahasa yang digunakan. Informasi ini diperoleh melalui kelas QSystemInfo.




Untuk realisasinya, berikut ini kode program pada bagian file qtmobilitytest.cpp

```
#include "qtmobilitytest.h"
#include "ui_qtmobilitytest.h"
#include <QtSystemInfo/QSystemInfo>
using namespace QtMobility;
QtMobilityTest::QtMobilityTest(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::QtMobilityTest)
{
    ui->setupUi(this);
    QSystemInfo* systemInfo;
    systemInfo = new QSystemInfo();
    ui->label->setText("Bahasa :" + systemInfo-
>currentLanguage());
}
QtMobilityTest::~QtMobilityTest()
{
    delete ui;
}
```

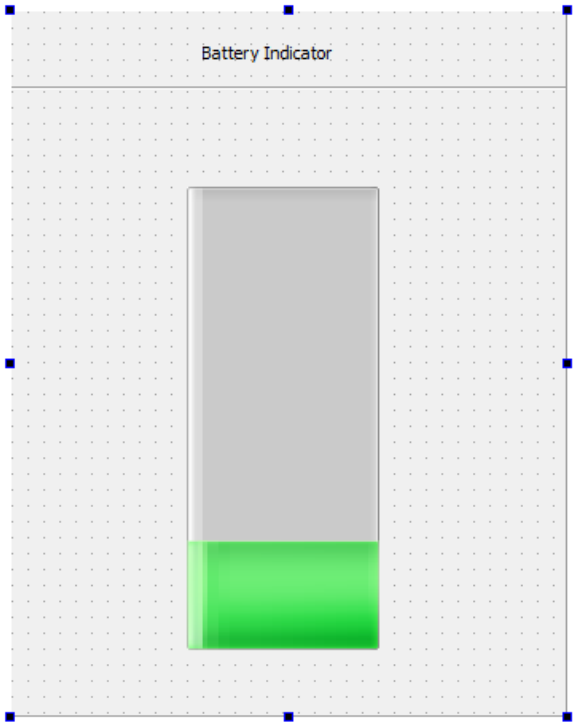
Pada kode program di atas, kita menampilkan nilai label dengan nilai bahasa yang digunakan melalui fungsi currentLanguage().

Selanjutnya lakukan kompilasi dan jalankan program.

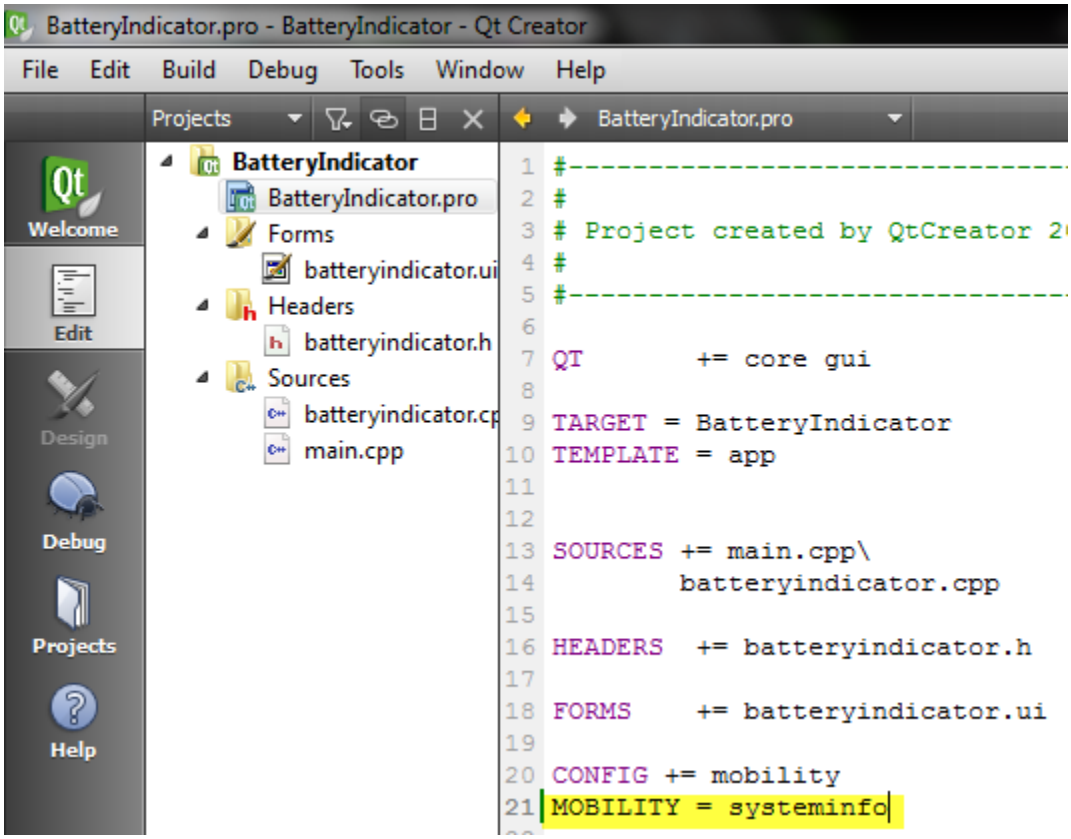

Pada gambar di atas terlihat, bahasa yang digunakan pada simulator adalah English.

Contoh lain kita akan mengambil indicator battery dan ditampilkan pada aplikasi kita dengan menggunakan API systeminfo.

Buat desain seperti berikut, dengan menggunakan Label, Horizontal Line, dan Progress Bar



Kemudian jangan lupa untuk menambahkan API mobility systeminfo pada file project.




Selanjutnya tambahkan kode program pada file header dengan kelas `QSystemDeviceInfo`, dan manambahkan library `QSystemInfo`.

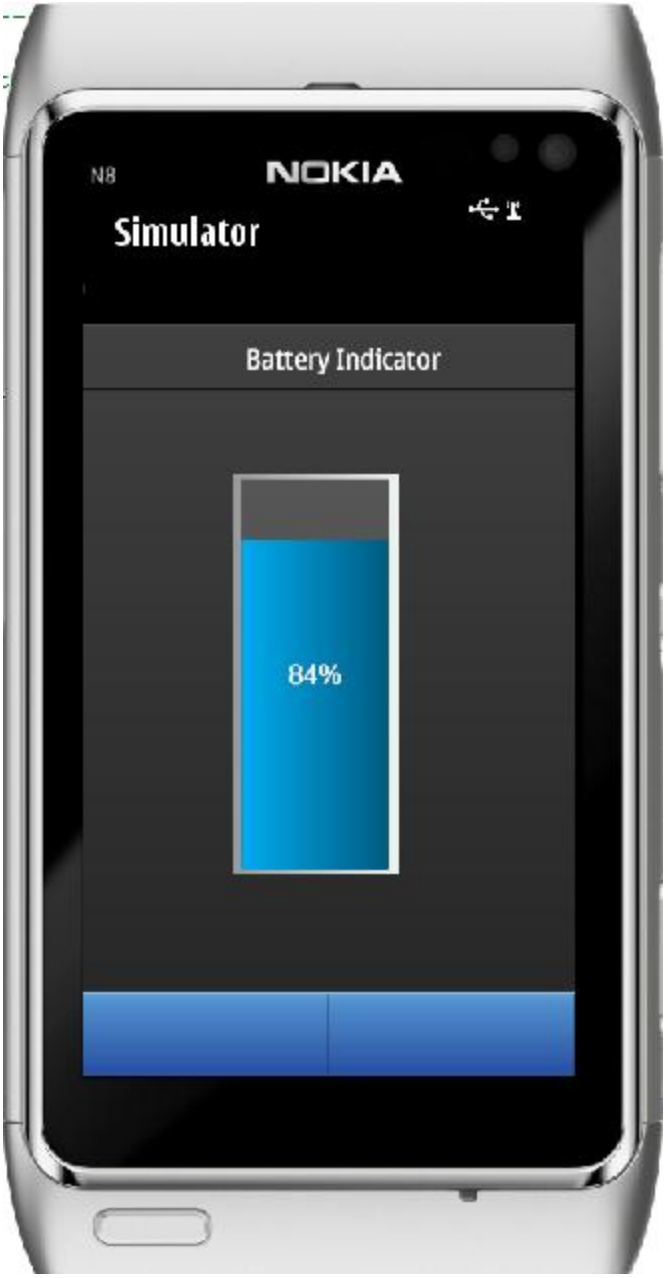
```
#ifndef BATTERYINDICATOR_H
#define BATTERYINDICATOR_H
#include <QDialog>
#include <QSystemInfo>
QTM_USE_NAMESPACE
namespace Ui {
    class BatteryIndicator;
}
class BatteryIndicator : public QDialog
{
    Q_OBJECT
public:
    explicit BatteryIndicator(QWidget *parent = 0);
    ~BatteryIndicator();
private:
    Ui::BatteryIndicator *ui;
    void setupGeneral();
    QSystemDeviceInfo *deviceInfo;
};
#endif // BATTERYINDICATOR_H
```

Selanjutnya pada file cpp masukan kode program berikut ini.

```
#include "batteryindicator.h"
#include "ui_batteryindicator.h"

BatteryIndicator::BatteryIndicator(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::BatteryIndicator),
    deviceInfo(NULL)
{
    ui->setupUi(this);
    setupGeneral();
}
BatteryIndicator::~BatteryIndicator()
{
    delete ui;
}

void BatteryIndicator::setupGeneral()
{
    // Buat objek QSystemDeviceInfo dan set nilainya
    deviceInfo = new QSystemDeviceInfo(this);
    ui->batteryLevelBar->setValue(deviceInfo->batteryLevel());
    // terhubung sinyal ke slot setValue dari progress bar
    connect(deviceInfo, SIGNAL(batteryLevelChanged(int)),
            ui->batteryLevelBar, SLOT(setValue(int)));
}
```

Pada file cpp ini kita membuat objek baru `QSystemDeviceInfo` dan membuat slot dan signal pada batteryLevelBar (yang merupakan nama objek progress bar)

Jalankan kode program di atas, maka akan tampil pada simulator yang menujukan indicator battery pada device nokia.


N8
NOKIA
Simulator
Battery Indicator
84%

## API Kontak

API ini mendefinisikan struktur dan pengambilan data kontak dari lokal atau remote. API menawarkan operasi seperti membuat, mengedit, daftar, menghapus dan pencarian informasi kontak, terlepas dari apakah data yang disimpan secara lokal maupun remote.

Kelas yang tersedia untuk API Kontak :

| Kelas | Keterangan |
|---|---|
| QContact | Kontak |
| QContactDetail | Rincian dari QContact |
| QContactManager | Akses ke kontak yang tersimpan di backend khususnya |
| QContactFilter | Digunakan untuk memilih kontak melalui QContactManager |
| QContactAction | Antarmuka untuk melakukan tindakan untuk menghubungi (seperti "Kirim email" atau "Dial") |

Kontak dapat dibuat dengan menciptakan sebuah instance dari object QContact, menambahkan rincian kontak dan menyimpannya ke database menghubungi melalui kelas QContactManager. Ini dilakukan dalam potongan kode berikut ini:

```
QContactManager* contactManager = new QContactManager( this );
QContact homer;

// Create name detail
QContactName name;
name.setFirst("Dwi");
name.setLast("Ariani");
name.setCustomLabel("Chuwey");
homer.saveDetail(&name);

// Create phone number detail
QContactPhoneNumber number;
number.setContexts(QContactDetail::ContextHome);
number.setSubTypes(QContactPhoneNumber::SubTypeMobile);
number.setNumber("62812342342");
homer.saveDetail(&number);
homer.setPreferredDetail("DialAction", number);

// Create address detail
QContactAddress address;
address.setCountry("ID");
address.setRegion("Bandung);
address.setPostCode("40134");
homer.saveDetail(&address);

// Save the contact to the contacts database
contactManager->saveContact(&dwi);
}
```

Contoh kode berikut menggambarkan cara untuk mendapatkan nomor telepon dari kontak tertentu. ID dari kontak pertama diambil sebagai QList dari ID kontak. Contoh mengambil kontak pertama dalam database kontak dan mendapatkan nomor telepon untuk itu:

```
QContactManager* contactManager = new QContactManager( this );
QList<QContactLocalId> contactIds = contactManager->contacts();
QContact firstContact = contactManager->contact( contactIds.first()
);
QString phoneNumber = firstContact.detail(QContactPhoneNumber::
DefinitionName).value(QContactPhoneNumber::FieldNumber);
```

Kontak dapat diedit dengan terlebih dahulu mengambil kontak, mengubah atau menambahkan rincian yang diperlukan, dan kemudian menyimpan kontak diperbarui dalam database kontak. Hal ini dapat dicapai dengan potongan kode berikut:

```
QContactManager* contactManager = new QContactManager( this );
QList<QContactLocalId> contactIds = contactManager->contacts();
QContact firstContact = contactManager->contact( contactIds.first()
);

// Change the phone number
QList<QContactDetail> numbers =
firstContact.details(QContactPhoneNumber::DefinitionName);
QContactPhoneNumber phoneNumber = numbers.value(0);
phoneNumber.setNumber("62812342342");

// Add an email address
QContactEmailAddress email;
email.setEmailAddress("email@ciebal.com");
email.setContexts(QContactDetail::ContextWork);
email.setValue("Label", "Dwi's work email");

// Save the details
firstContact.saveDetail(&phone);
firstContact.saveDetail(&email);

// Save the updated contact to the database
contactManager->saveContact(&firstContact);
```

## Bearer Management

API ini berguna untuk mengontrol kondisi koneksi system peralatan mobile dimana user dapat melakukan start dan stop interface atau roaming secara transparan.

Bearer Management memungkinkan untuk:

- Mengelola konektivitas ke jaringan
  - Memungkinkan pengguna untuk memulai atau menghentikan interface jaringan
  - Informasi jika perangkat sedang online dan berapa banyak yg tersedia interface
- Memungkinkan perbandingan dan prioritas akses dan penggunaan jalur akses
- Menempatkan koneksi terbaik
  - Pengguna membuat pilihan berdasarkan situasi
  - Transparan seleksi
- Otomatis roaming antara jaringan selular dan WLAN

API Bearer ini memanfaatkan HTTP level roaming yang disediakan dikelas QNetworkAccessManager. Dalam hal ini, kita melakukan optimalisasi bagaimana mengatur suatu interface. Ada tiga kelas yang disediakan bearer management, yaitu:

| Nama Kelas | Keterangan |
|---|---|
| **QNetworkConfiguration** | Abstraksi dari satu atau lebih konfigurasi access point |
| **QNetworkConfigurationManager** | Mengatur konfigurasi jaringan yang disediakan oleh system |
| **QNetworkSession** | Mengkontrol access point system dan mengaktifkan session management ketika banyak client yang mengakses access point yang sama. |

Untuk menggunakan bearer management kita harus mengkonfigurasi file project Qt dengan menambahkan:

```
CONFIG += mobility
MOBILITY = bearer
```

Contoh ilustrasi, misalkan kita ingin memperoleh informasi konfigurasi jaringan, kita dapat menggunakan kelas QNetworkConfigurationManager.

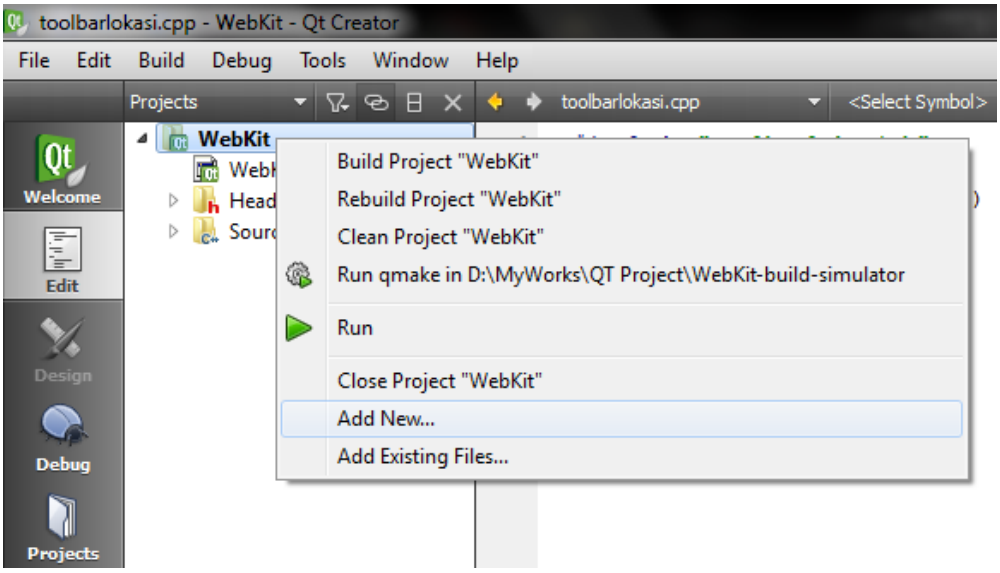
```
#include <qnetworkconfmanager.h>
...
QNetworkConfigurationManager manager;
QList<QNetworkConfiguration> allConfigurations = manager
allConfiguration();
```

## API Lokasi

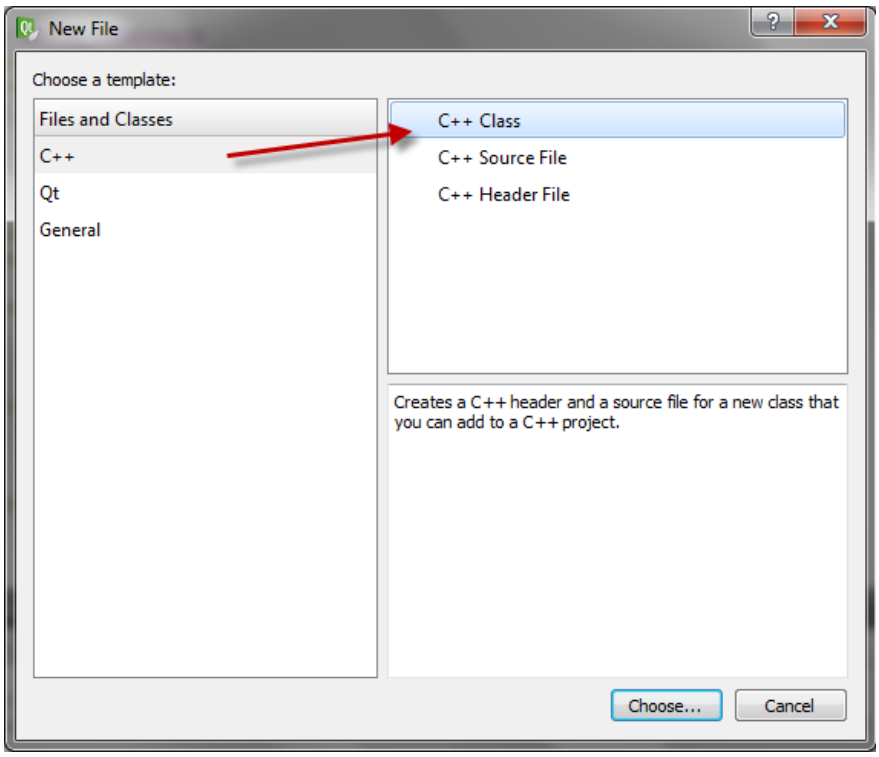
Pada bab sebelumnya kita sudah membuat sebuah aplikasi browser sederhana menggunakan WebKit, Kali ini kita coba membuat browser disertai informasi lokasi pengguna dengan menggunakan Location API (QGeoPositionInfoSource) yang sudah tersedia pada Qt Mobility.

Pada aplikasi ini, kita akan menambahkan toolbar berisi informasi lokasi. Layout yang digunakan untuk widget toolbar adalah horisontal, jadi kita juga akan menggunakan manajer layout QHBoxLayout.

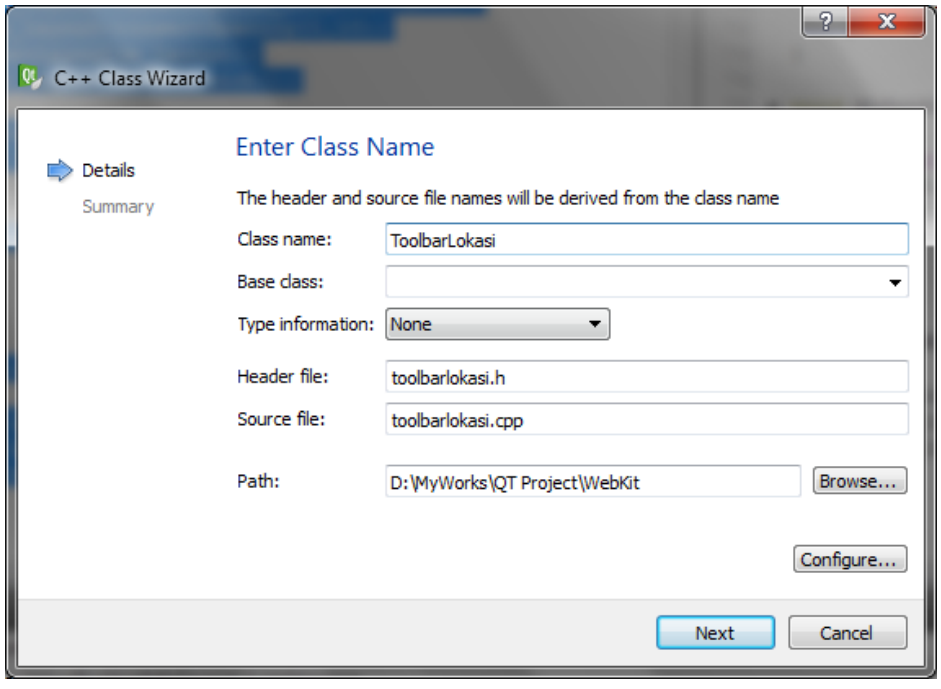
Sekarang buka project WebKit yang telah kita buat. Kemudian buat kelas baru. Klik kanan pada Project, kemudian Add New.



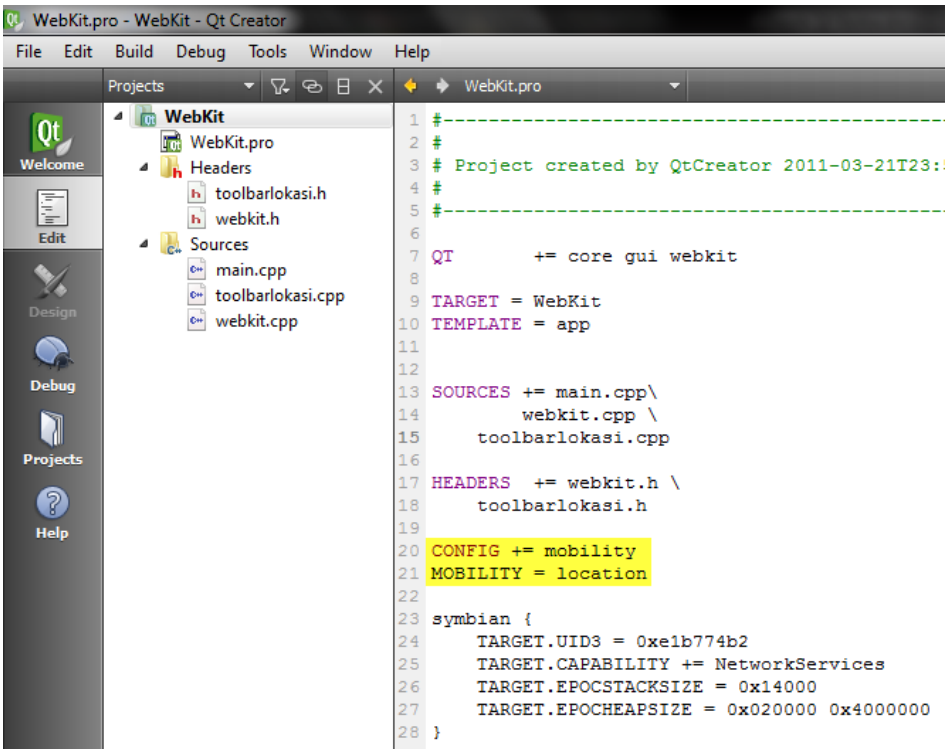

Pada template pilih C++ kemudian C++ Class.




Buat kelas baru dengan nama ToolbarLokasi




Kelas ToolbarLokasi ini akan diisi kode program API Lokasi dan pembentukan tata letak toolbar. Selanjutnya pada WebKit.pro jangan lupa menambahkan MOBILITY Location pada property proyek.




Buka file toolbarlokasi.h, dan ubah beberapa baris kode berikut ini.

```
#ifndef ToolbarLokasi_H
#define ToolbarLokasi_H
#include <qmobilityglobal.h>
#include <QtGui/QWidget>

QTM_BEGIN_NAMESPACE
class QGeoPositionInfo;
class QGeoPositionInfoSource;
QTM_END_NAMESPACE

QTM_USE_NAMESPACE
class QHBoxLayout;
class QLabel;
class ToolbarLokasi : public QWidget
{
    Q_OBJECT
public:
    ToolbarLokasi(QWidget *parent = 0);
    ~ToolbarLokasi();
signals:
    void showLocation();
private slots:
    void onPositionUpdated(const QGeoPositionInfo& posInfo);
private:
    QLabel* m_locationLabel;
    QHBoxLayout* m_layout;
    QGeoPositionInfoSource* m_location;
};
#endif // ToolbarLokasi_H
```

Seperti yang Anda lihat, ToolbarLokasi memiliki signals sendiri. Jangn lupa untuk menambahkan kelas ToolbarLokasi, QGeoPositionInfoSource karena kita membutuhkannya. Mari kita lihat implementasi dari kelas ini (toolbarlokasi.cpp):

```
#include <QtGui/QLabel>
#include <QtGui/QHBoxLayout>
#include <QtGui/QAction>
#include <QtLocation/QGeoPositionInfoSource>
#include <QtLocation/QGeoPositionInfo>
#include <QtGui/QApplication>
#include "ToolbarLokasi.h"

const QString locString("My Location: ")
;
ToolbarLokasi::ToolbarLokasi(QWidget* parent)
    :QWidget(parent)
{
    m_locationLabel = new QLabel(this);
    m_location = QGeoPositionInfoSource::createDefaultSource(this);
    m_location->setUpdateInterval(20000);
    m_location->startUpdates();
    m_layout = new QHBoxLayout;
    m_layout->addWidget(m_locationLabel);
    m_layout->insertSpacing(1,10);
    setLayout(m_layout);
    m_layout->addStretch();

connect(m_location,SIGNAL(positionUpdated(QGeoPositionInfo)),SLOT(o
nPositionUpdated(QGeoPositionInfo)));
}

ToolbarLokasi::~ToolbarLokasi()
{
}

void ToolbarLokasi::onPositionUpdated(const QGeoPositionInfo&
posInfo)
{
    QGeoCoordinate coordinate = posInfo.coordinate();
    QString label_coordinate = coordinate.toString();
    m_locationLabel->setText(locString+" "+label_coordinate);
}
```

Kode program di atas kita membuat child widget dan mengatur interval update setiap 20 detik. Selanjutnya mari kita lihat kelas WebKit, untuk melihat bagaimana hal itu berubah. Dalam constructor, kita perlu menciptakan widget toolbar dan menambahkannya ke tata letak (vertikal) yang ada:

```
// file webkit.cpp
#include <QtGui/QLineEdit>
#include <QtWebKit/QWebView>
#include <QtGui/QVBoxLayout>
#include <QtGui/QLabel>
#include <QtCore/QUrl>
#include <QtGui/QAction>
#include "toolbarlokasi.h"
#include "WebKit.h"

WebKit::WebKit(QWidget *parent)
    : QWidget(parent)
{
    m_softkeyAction = new QAction( tr("Options"), this );
    m_softkeyAction->setSoftKeyRole(QAction::PositiveSoftKey);
    addAction(m_softkeyAction);

    m_lineEdit = new QLineEdit(this);
    m_lineEdit->setStyleSheet("background-color:white; padding: 6px ;
color:blue");
    m_lineEdit->setPlaceholderText("Enter url ...");
    m_view = new QWebView(this);
    m_view->load(QUrl("http://nice.or.id"));
    m_layout = new QVBoxLayout();
    m_layout->addWidget(m_lineEdit);
    m_layout->addWidget(m_view);
    m_layout->insertSpacing(1,10);
    //add toolbar
    m_toolbar = new ToolbarLokasi(this);
    m_layout->addWidget(m_toolbar);

    setLayout(m_layout);
    m_layout->addStretch();

    connect(m_lineEdit,SIGNAL(editingFinished()),SLOT(openUrl()));

connect(m_view,SIGNAL(loadFinished(bool)),SLOT(onLoadFinished(bool)));
}

WebKit::~WebKit()
{
}
void WebKit::openUrl()
{
    QString url(m_lineEdit->text());
    if(!url.contains("http://",Qt::CaseInsensitive))
        url.prepend("http://");
    m_view->load(QUrl(url));
}

void WebKit::onLoadFinished(bool finished)
{
    if(finished){
        m_lineEdit->clear();
        m_lineEdit->setPlaceholderText(tr("Enter url ..."));
    }
}
```

Yang terakhir, kita ubah sedikit baris kode pada file webkit.h

```
#ifndef WEBKIT_H
#define WEBKIT_H
#include <QtGui/QWidget>
class QWebView;
class QLineEdit;
class QVBoxLayout;
class QLabel;
class QAction;
class QGeoPositionInfoSource;
class ToolbarLokasi; // Tambah Kelas ToolbarLokasi
class WebKit : public QWidget
{
    Q_OBJECT
public:
    WebKit(QWidget *parent = 0);
    ~WebKit();
private slots:
    void openUrl();
    void onLoadFinished(bool finished);
private:
    QAction* m_softkeyAction;
    QWebView* m_view;
    QLineEdit* m_lineEdit;
    QLabel* m_label;
    QVBoxLayout* m_layout;
    ToolbarLokasi* m_toolbar;
};
#endif // WEBKIT_H
```

Kita perlu menambahkan kelas-kelas yang digunakan dalam pembuatan aplikasi terutama kelas QGeoPositionInfoSource dan ToolbarLokasi. Setelah aplikasi dijalankan, maka akan tampil seperti berikut ini.

## Network Session

`QNetworkConfigurationManager` adalah kelas yang mengelola konfigurasi jaringan yang disediakan oleh sistem. Kelas `QNetworkConfiguration` menyediakan sebuah abstraksi dari konfigurasi jalur akses. Contoh kode berikut menggambarkan bagaimana sesi jaringan dapat dibentuk tanpa adanya interaksi pengguna:

```
QNetworkConfigurationManager configurationManager;
const bool canStartAccessPoint = (configurationManager.capabilities()
                                  &
QNetworkConfigurationManager::BearerManagement);
QNetworkConfiguration configuration = manager.defaultConfiguration();
if ( configuration.isValid() || !canStartAccessPoint )
    return;
switch( configuration.type() ) {
case QNetworkConfiguration::InternetAccessPoint:
    // System starts the IAP immediately
    break;
case QNetworkConfiguration::ServiceNetwork:
    // System determines the best IAP available and starts it immediately
    break;
case QNetworkConfiguration::UserChoice:
    // The access point is resolved by asking the user
    break;
}
QNetworkSession* session = new QNetworkSession( configuration );
session->open();
```

Pada file project Qt Anda jangan lupa tambahkan *network* pada mobility. Pembahasan lebih lengkap akan dibahas pada bab selanjutnya.

## API Messaging

Sebuah interface umum untuk penanganan SMS, MMS dan pesan email yang diberikan oleh API Messaging. API memberikan akses ke berbagai kegiatan terkait dengan pesan. Hal ini memungkinkan layanan pesan ke pesan pencarian dan menyortir, memberitahukan perubahan pesan yang disimpan, mengirim pesan dengan atau tanpa lampiran, mengambil data untuk menampilkan pesan yang ada, atau untuk menulis pesan.

`QMessage` merupakan obyek pesan dari berbagai tipe, seperti email, MMS atau SMS. Feld message yang dibutuhkan adalah, body, attachments dan data lain yang dapat ditambahkan tergantung pada jenis pesan. Potongan kode berikut membuat pesan email baru dan mengirimkannya ke alamat email tertentu. Hal ini dilakukan oleh kelas `QMessageServiceAction`, yang juga dapat digunakan untuk mengambil pesan dan data pesan dan pesan-terkait tindakan yang sesuai:

```
QMessageServiceAction* serviceAction = new QMessageServiceAction(
this );
// Create a new email message

QMessage message;
message.setType(QMessage::Email);

// Add required fields
message.setTo(QMessageAddress("myfriend.bestis@emailaddress",QMessage
Address::Email));
message.setSubject("Pictures from our holidays :)");

// Set message body
message.setBody("Here you go!");

// Add attachments
QStringList attachments;
attachments.append("Picture1.jpg");
attachments.append("Picture2.jpg");
message.appendAttachments(paths);

// Send the message
serviceAction->send(message);
```

## API Multimedia

Multimedia telah menjadi fitur standar di ponsel. Qt Mobility API menyediakan akses ke library multimedia, yang menawarkan cara mudah untuk memainkan dan merekam audio dan video dalam berbagai format.

Selain bermain dan merekam, fitur lainnya juga dapat diakses dengan menggunakan API. Sebagai contoh, FM radio dapat digunakan melalui API ini. Selain itu, tampilan slide dapat diakses dengan menggunakan API ini.

Potongan kode berikut ini memainkan lagu MP3 di situs remote. API mengirimkan signal dari pemutar file media. Signal `positionChanged()` dihubungkan dengan parameter untuk menentukan posisi pemutaran audio dalam milidetik dari awal file audio. Durasi `method()` mengembalikan total waktu pemutaran file objek media:

```
QMediaPlayer* mediaPlayer = new QMediaPlayer;
connect(mediaPlayer, SIGNAL(positionChanged(qint64)), this,
SLOT(myPositionChangedHandler(qint64)));
mediaPlayer-
>setMedia(QUrl::fromLocalFile("http://musik.com/lagu.mp3"));
mediaPlayer->setVolume(50);
mediaPlayer->play();
```

Kelas `QMediaPlaylist` dapat digunakan untuk membuat playlist untuk beberapa jenis media. `QVideoWidget` merupakan widget khusus untuk pemutaran video. Contoh ini membuat playlist video dan menggunakan media player untuk memutar ulang video pada sebuah widget video:

```
QMediaPlayer* mediaPlayer = new QMediaPlayer( this );
QMediaPlaylist* playlist = new QMediaPlaylist(player);
playlist->append(QUrl("/MyVideos/video1.mp4"));
playlist->append(QUrl("/MyVideos/video2.mp4"));
playlist->append(QUrl("/MyVideos/video3.mp4"));
QVideoWidget* widget = new QVideoWidget( mediaPlayer, parentWindow
);
widget->show();
player->play();
```

## API Publish and Subscribe

Publish and Subscribe adalah paradigma pesan banyak digunakan, di mana komunikasi dipisahkan dari kedua produsen (penerbit) dan konsumen (pelanggan) dari data. Komunikasi berlangsung asynchronous, paling sering menggunakan objek data tambahan.

API Publish and Subscribe memungkinkan akses konteks-sensitif terhadap informasi dalam rilis yang akan datang. Sebagai perubahan konteks, nilai-nilai perubahan obyek tetapi hubungan tetap sama. Hal ini dimaksudkan untuk menjadi sebuah teknologi yang memungkinkan, dasar dari berbagai aplikasi di masa depan.

## Service Framework

Hal ini biasa untuk perangkat mobile untuk memberikan layanan perangkat khusus. Oleh karena itu, salah satu ekstensi yang paling menarik Qt mobile adalah sebuah platform independent method untuk menemukan services.

Dalam lingkup QtMobility API, Service framework API mendefinisikan cara bersatu untuk menemukan, menerapkan dan mengakses layanan di seluruh platform. Dalam hal framework, layanan merupakan komponen independen yang memungkinkan klien untuk melakukan operasi yang jelas. Service diimplementasikan sebagai plug-in diinstal pada perangkat dan dapat mencari service pendukung eksternal yang berjalan pada server pusat. Selain itu, karena framework dasarnya adalah layanan lapisan abstraksi, aplikasi tidak perlu khawatir dengan protokol yang mendasari, di mana server, kejanggalan hardware dengan jaringan dan rincian low-level.

## Webkit dan API Lokasi

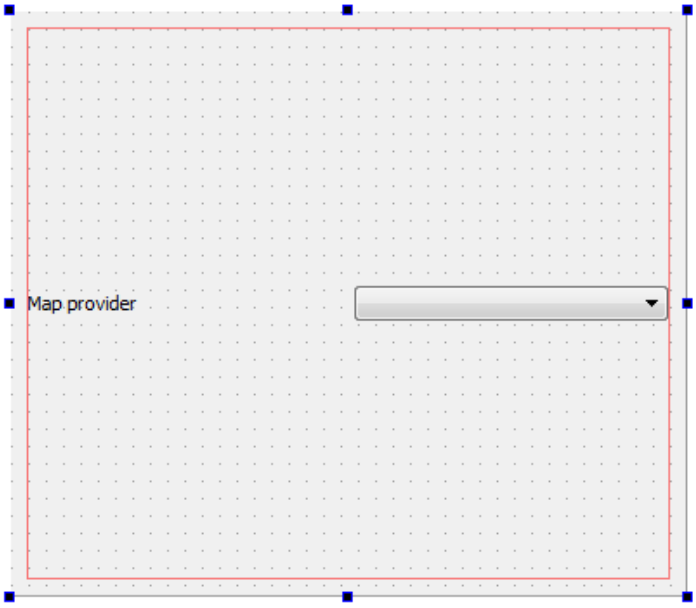
Kali ini kita coba bagaimana dapat menggunakan web map dan data lokasi dalam aplikasi Qt Anda. Kita juga melihat bagaimana mengintegrasikan kode CPP, QML Kode dan JS berjalan di webkit. Kita juga melihat bagaimana mudahnya menggunakan API web. Gambar dibawah menunjukkan aplikasi UI.


Kali ini kita coba menggunakan Qml sebagai implementasi dasar dari UI dengan menggunakan elemen WebView, dan menggunakan HTML sebagai file yang berisi script peta.

Sekarang mari kita mulai membuat aplikasi map sederhana. Buat sebuah project QWidget baru dengan nama project **qmlwebmaplocation** disertai file UI dan nama kelas dan header biarkan widget.

Desain UI sederhana dengan menambahkan label dan comboBox yang nanti berfungsi untuk memilih provider map OVI, BING, atau Google Map.

Dan buat menjadi Horizontal Layout.




Mari kita mulai dengan file header.

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QGeoPositionInfo>
#include <QGeoPositionInfoSource>
QTM_USE_NAMESPACE
class QDeclarativeView;
namespace Ui {
    class Widget;
}
class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    Q_PROPERTY(QString mapProvider READ mapProvider WRITE
setMapProvider NOTIFY mapProviderChanged)
    QString mapProvider();
    void setMapProvider(QString newMapProvider);
private slots:
    void mapProviderSelected(int);
    void positionUpdated(QGeoPositionInfo);
signals:
    void mapProviderChanged(QString);
    void posUpdated(double lat, double lon);
private:
    Ui::Widget *ui;
    QDeclarativeView* m_view;
    QGeoPositionInfoSource* m_geoSource;
    QString m_mapProvider;
};
#endif // WIDGET_H
```

Disini kita menggunakan kelas QGeoPosition untuk menunjukan lokasi update dari GPS, kemudian kita lihat kode program CPP.

```
#include <QDeclarativeView>
#include <QDeclarativeEngine>
#include <QDeclarativeContext>
#include <QtDebug>
#include "widget.h"
#include "ui_widget.h"
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    m_mapProvider = "init.html";
    ui->setupUi(this);
    m_view = new QDeclarativeView(this);
    m_view->rootContext()->setContextProperty("cppEngine", this);
    m_view->setSource(QUrl("qrc:/qmls/main.qml"));
    ui->verticalLayout->addWidget(m_view);
    m_view->setResizeMode(QDeclarativeView::SizeRootObjectToView);
    ui->comboBox->addItem("Select Map Provider",
QVariant("init.html"));
    ui->comboBox->addItem("OVI Maps", QVariant("ovimaps.html"));
    ui->comboBox->addItem("Google Maps", QVariant("gmaps.html"));
    ui->comboBox->addItem("Bing Maps", QVariant("bingmaps.html"));
    m_geoSource = QGeoPositionInfoSource::createDefaultSource(this);
    if (m_geoSource)
    {
        connect(m_geoSource,
SIGNAL(positionUpdated(QGeoPositionInfo)), this,
SLOT(positionUpdated(QGeoPositionInfo)));
        m_geoSource->setUpdateInterval(10000);
        m_geoSource->startUpdates();
    }
    connect(ui->comboBox, SIGNAL(currentIndexChanged(int)),
            this, SLOT(mapProviderSelected(int)));
}
Widget::~Widget()
{
    delete ui;
}
QString Widget::mapProvider()
{
    return m_mapProvider;
}
void Widget::mapProviderSelected(int index)
{
    QString htmlFileName = ui->comboBox->itemData( index
).toString();
    setMapProvider( htmlFileName );
}
```

```
void Widget::setMapProvider(QString newMapProvider)
{
    if (m_mapProvider == newMapProvider)
        return;
    else
    {
        m_mapProvider = newMapProvider;
        emit mapProviderChanged(m_mapProvider);
    }
}
void Widget::positionUpdated(QGeoPositionInfo info)
{
    emit posUpdated( info.coordinate().latitude(),
info.coordinate().longitude());
}
```

Kita memiliki kelas yang mewarisi QWidget. Instance dari kelas ini akan menjadi top-level widget. Pada bagian public kita mendefinisikan sebuah properti (mapProvider) yang digunakan dari sisi QML. Ini berisi nama file html yang digunakan dalam webView.

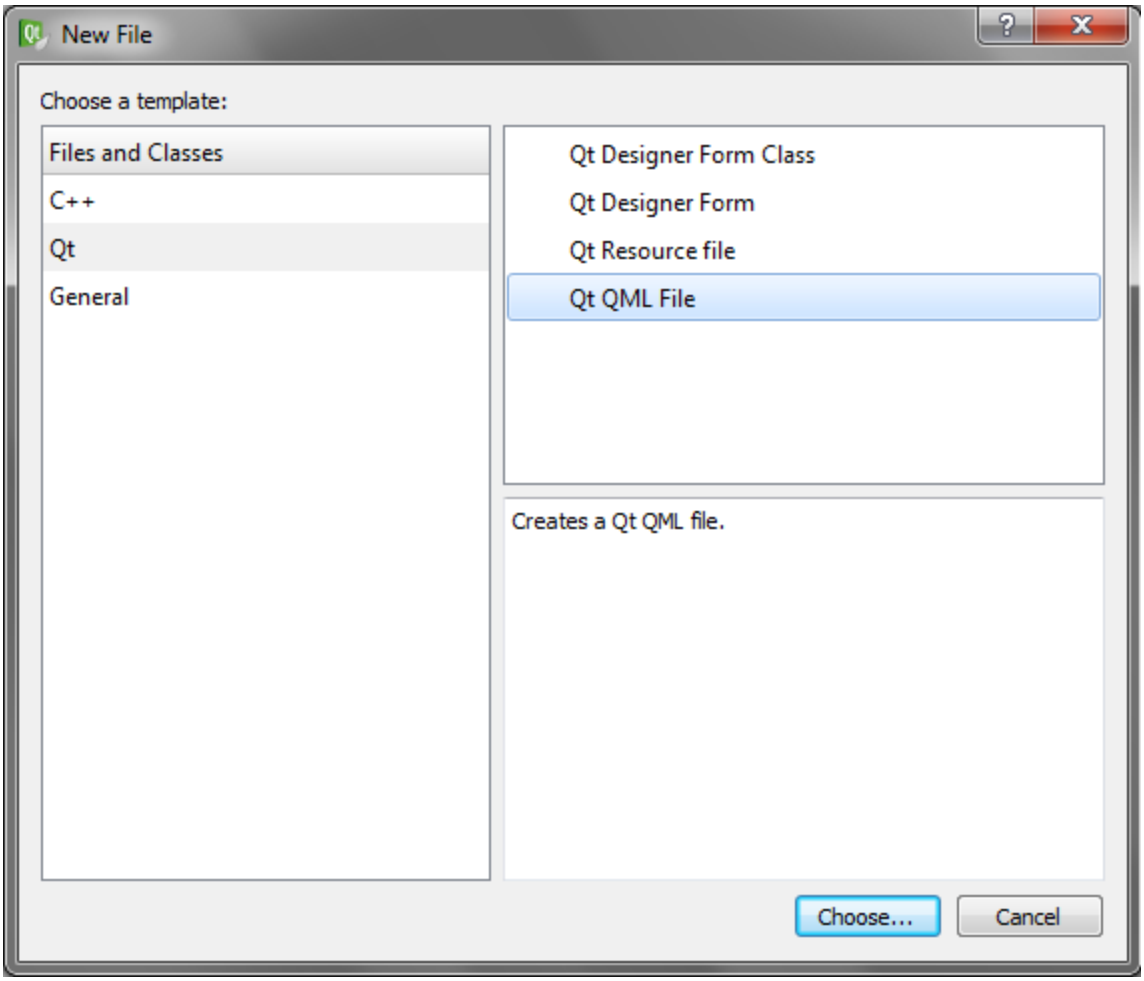
Kemudian kita mendefinisikan dua slot. Pertama adalah digunakan ketika pengguna memilih provider map dari combo box dan yang kedua dihubungkan dengan sinyal `QGeoPositionInfoSources` 'positionUpdated. Sinyal ini dipancarkan ketika perubahan lokasi device.

Pada bagian sinyal kita mendefinisikan sinyal sendiri untuk mengubah provider map & update.

Di konstruktor kita membuat aplikasi UI dan load file qml untuk `QDeclarativeView`. kita juga menerbitkan instance dari kelas (setContextProperty) ke sisi QML, karena kita ingin menggunakan sinyal, slot dan properti dari kelas. Kita juga mengatur Modus mengubah ukuran untuk `SizeRootObjectToView` dan elemen root secara otomatis akan memiliki ukuran maksimum. kita juga membuat sumber info posisi geografis dan mulai mencari update lokasi.

Dalam metode `setMapProvider` kita menetapkan provider peta baru dan memancarkan sinyal tentang perubahan itu. Dalam slot `positionUpdated` kita memancarkan sinyal baru yang digunakan di sisi QML.

Mari kita buat file QML dalam project ini, caranya sama seperti memuat kelas baru pada Qt. Namun pada template pilih Qt Qml File dengan nama **mail.qml** karena pada file CPP kita memanggil file main.qml, namun Anda bisa sesuaikan sendiri.




Selanjutnya kita masukan kode program pada file QML.

```
import QtWebKit 1.0
import Qt 4.7
Item {
    function mapHtmlFileName() {
         var url = "/htmls/" + cppEngine.mapProvider;
         return url;
         }
    function positionUpdated(lat, lon)
        {
        mapView.evaluateJavaScript("pan("+lat+","+lon+");");
        }
    Connections {
        target: cppEngine;
        onPosUpdated: positionUpdated(lat, lon)
    }
    WebView {
        id: mapView
        url: mapHtmlFileName()
        preferredWidth: parent.width
        preferredHeight: parent.height
        scale: 1
        smooth: false
    }
}
```

Dalam Koneksi objek kita terhubung posUpdated sinyal untuk metode JavaScript positionUpdated. Dalam metode yang di sebut metode webView evaluateJavaScript. Ini akan mengeksekusi kode di lingkungan weViews JavaScript. Kita juga menggunakan properti mapProvider cppEngine untuk menentukan file yang diambil untuk webView.

Selanjutnya kita buat file html untuk me-load masing-masing file maps, file ini terdiri dari *init.html, ovimaps.html, bingmaps.html, dan gmaps.html*. semua file html tersebut diletakan pada folder htmls yang diletakan bersamaan pada folder project. Untuk mempercantik tampilan siapkan juga gambar loading yang bisa Anda dapatkan di internet.

Kode html untuk init.html

```
<html>
<head>
</head>
<body bgcolor="#000000" text="#FFFFFF">
<br>
<br>
<br>
<center>Select map provider...
<br>
<br>
<img src="loader.gif"/>
</center>
</body>
</html>
```

Kode ovimaps.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
  <style type="text/css">
    html { height: 100% }
    body { height: 100%; margin: 0px; padding: 0px  ; background:
url("loader.gif");
    background-repeat: no-repeat;
    background-position: center;
}
    #map { height: 100% }
  </style>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
    <title>Ovi Maps API Example</title>
    <script src="http://api.maps.ovi.com/jsl.js" type="text/javascript"
charset="utf-8"></script>
    <script type="text/javascript">
    var map;
    function initialize()
    {
```

```
map = new ovi.mapsapi.map.Display(document.getElementById("mapDiv"), {
     components: [ new ovi.mapsapi.map.component.Behavior(), //behavior
collection
                                new ovi.mapsapi.map.component.ZoomBar(),
                                new ovi.mapsapi.map.component.Overview(),
                                new
ovi.mapsapi.map.component.TypeSelector(),
                                new ovi.mapsapi.map.component.ScaleBar()
],
        'zoomLevel': 10, //zoom level for the map
        'center': [52.51, 13.4] //center coordinates
        });
        map.addComponent( new ovi.mapsapi.map.component.panning.Drag()
);
        map.addComponent( new
ovi.mapsapi.map.component.panning.Kinetic() );
    }
     function pan( lat, lon ) {
     map.setCenter(new ovi.mapsapi.geo.Coordinate(lat,lon));
    }
    </script>
  </head>
  <body  onload="initialize()" bgcolor="#000000" text="#FFFFFF" >
    <div id="mapDiv" style="width:100%; height:100%;"></div>
    <script type="text/javascript">
    </script>
  </body>
</html>
```

Isi dari gmaps.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<style type="text/css">
  html { height: 100% }
  body { height: 100%; margin: 0px; padding: 0px ; background:
url("loader.gif");
  background-repeat: no-repeat;
  background-position: center; }
  #map_canvas { height: 100% }
</style>
<script type="text/javascript"
    src="http://maps.google.com/maps/api/js?sensor=false">
</script>
<script type="text/javascript">
  var map;

  function initialize() {
    setTimeout("qtcpp.posUpdated.connect(pan)",1000);
```

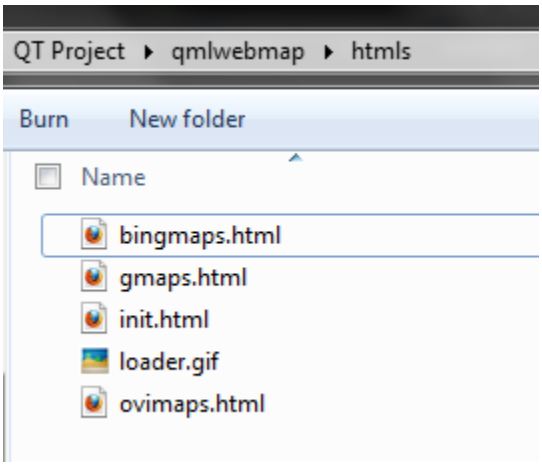
```
var latlng = new google.maps.LatLng(60.15, 25.03);
    var myOptions = {
      zoom: 16,
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    map = new google.maps.Map(document.getElementById("map_canvas"),
        myOptions);
  }
  function pan( lat, lon ) {
     var latlng = new google.maps.LatLng(lat, lon);
     map.panTo(latlng);
  }
</script>
</head>
<body onload="initialize()">
  <div id="map_canvas" style="width:100%; height:100%"></div>
</body>
</html>
```

Isi dari bingmaps.html

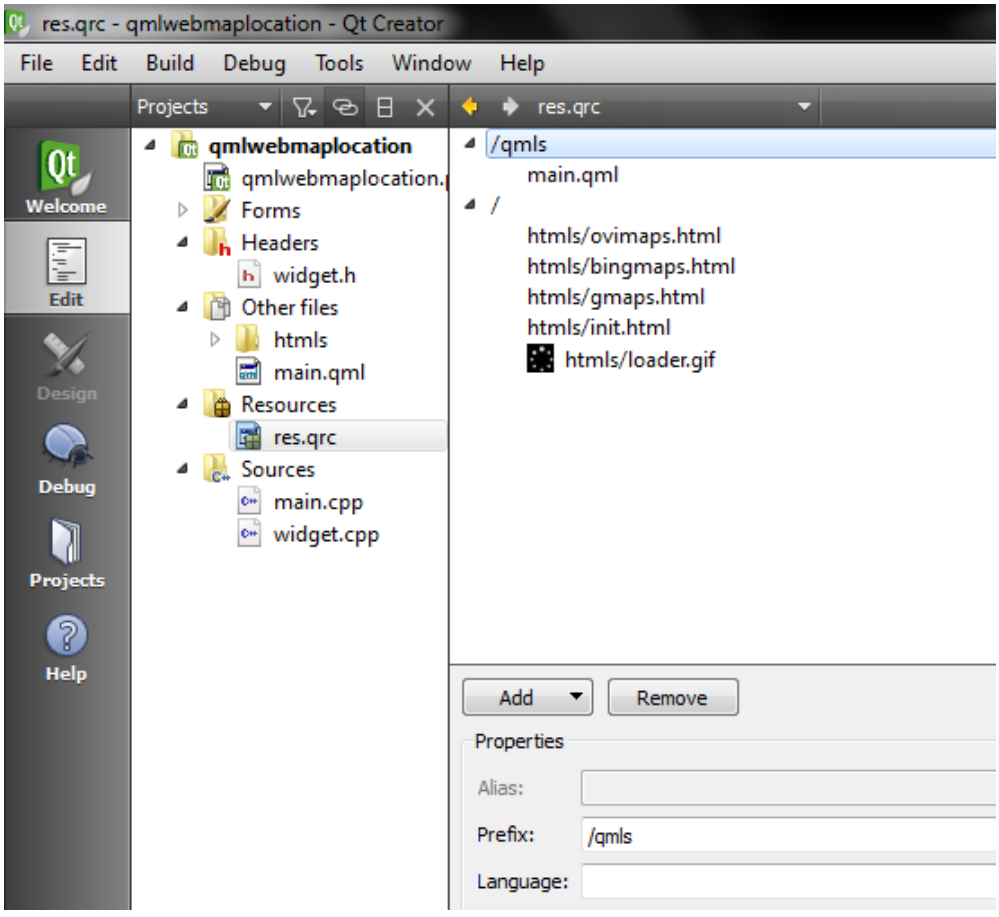
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
   <style type="text/css">
     html { height: 100% }
     body { height: 100%; margin: 0px; padding: 0px ; background:
url("loader.gif");
     background-repeat: no-repeat;
     background-position: center; }
     #myMap { height: 100% }
   </style>
      <title></title>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-
8">
      <script type="text/javascript"
src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=6.2"><
/script>
      <script type="text/javascript">
         var map = null;
         function GetMap()
         {
            map = new VEMap('myMap');
            map.LoadMap(new VELatLong(65.01, 25.29), 10 ,'h' ,false);
         }
         function pan(lat, lon)
         {
         var latlon = new VELatLong(lat, lon);
         map.setCenter(lat,lon);
         }
      </script>
   </head>
   <body onload="GetMap();">
      <div id='myMap' style="position:relative; width:100%;
height:100%;"></div>
   </body>
</html>
```

Kita menggunakan map control BING versi 6, karena ketika saya mencoba menggunakan versi 7 sulit untuk me-load pada simulator. Jadi Anda bisa coba sendiri nanti. ☺

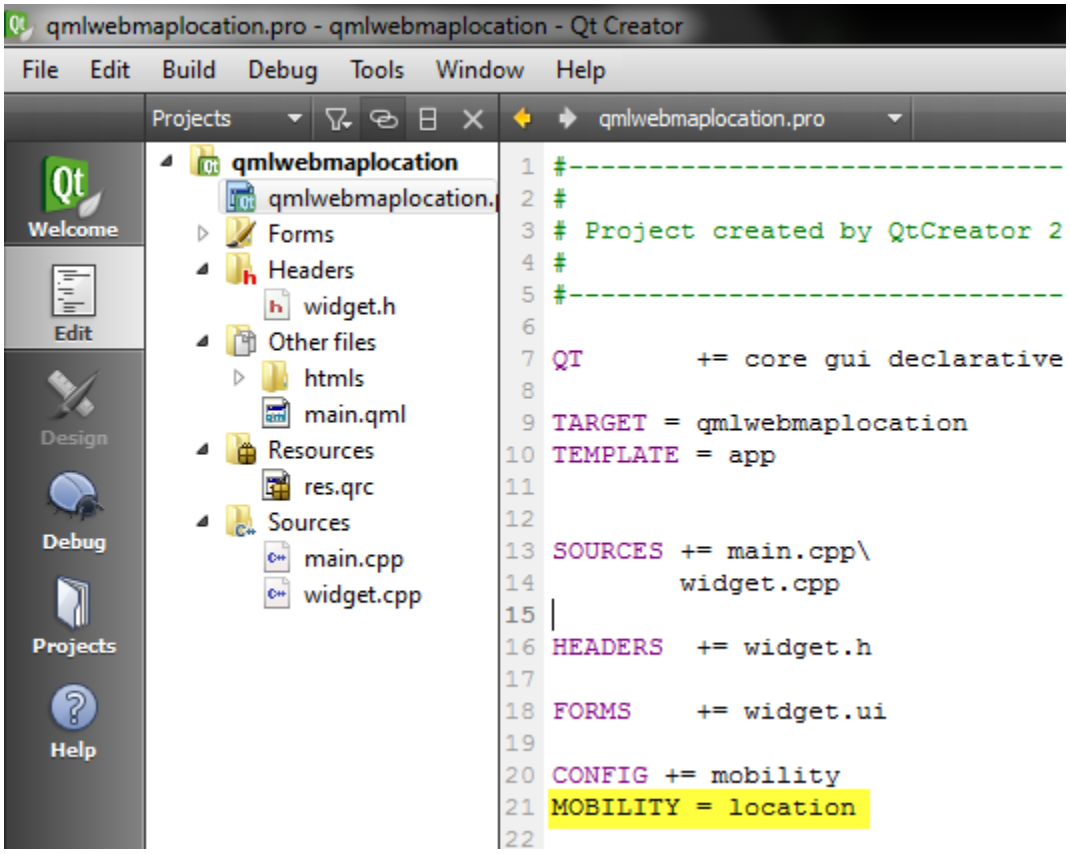
Sekarang kita sudah memiliki 5 file yaitu init.html, ovimaps.html, gmaps.html, bingmaps.html, dan loader.gif, seperti berikut:




Kita buat resource baru pada project ini, caranya sama seperti pada bab Webkit, yang nantinya akan jadi seperti ini.



Satu lagi, jangan lupa menambahkan modul mobility lokasi pada file project.




Mari kita coba jalankan kode program ini.


Simulator
Map provider
Bing Maps
2D
3D
Road
Aerial
Labels
Oulu
bing
Exit
NOKIA
N8

# Qt Network

Modul QtNetwork menawarkan kelas-kelas yang memungkinkan Anda untuk menulis TCP/IP klien dan server. Hal ini menawarkan kelas-kelas seperti QFtp yang mengimplementasikan protokol tertentu, kelas bawah level seperti `QTcpSocket`, `QTcpServer` dan `QUdpSocket` yang mewakili konsep-konsep level jaringan rendah, dan kelas tingkat tinggi seperti `QNetworkRequest`, `QNetworkReply` dan `QNetworkAccessManager` untuk melakukan operasi jaringan dengan menggunakan protokol yang umum . Ini juga menawarkan kelas-kelas seperti `QNetworkConfiguration`, `QNetworkConfigurationManager` dan `QNetworkSession` yang menerapkan bearer management.

## Kelas Qt Network

| Nama Kelas | Keterangan |
|---|---|
| **QAbstractSocket** | Fungsi umum untuk semua jenis soket dasar |
| **QAuthenticator** | Otentikasi objek |
| **QFtp** | Implementasi sisi klien protokol FTP |
| **QHostAddress** | Alamat IP |
| **QHostInfo** | Fungsi statis untuk pencarian nama host |
| **QNetworkAccessManager** | Memungkinkan aplikasi untuk mengirim permintaan jaringan dan menerima balasan |
| **QNetworkAddressEntry** | Satu alamat IP didukung oleh antarmuka jaringan, bersama dengan netmask yang terkait dan alamat broadcast |
| **QNetworkConfiguration** | Abstraksi dari satu atau lebih konfigurasi jalur akses |
| **QNetworkConfigurationManager** | Mengelola konfigurasi jaringan yang disediakan oleh sistem |
| **QNetworkInterface** | Daftar alamat IP host dan antarmuka jaringan |
| **QNetworkProxy** | Network layer proxy |
| **QNetworkProxyFactory** | Untuk membuat kebijakan untuk penggunaan proxy |
| **QNetworkReply** | Berisi data dan header untuk meminta dikirim dengan QNetworkAccessManager |
| **QNetworkRequest** | Menahan permintaan yang akan dikirim dengan QNetworkAccessManager |
| **QNetworkSession** | Kontrol atas jalur akses sistem dan memungkinkan manajemen sesi untuk kasus-kasus ketika beberapa klien mengakses jalur akses yang sama |
| **QSocketNotifier** | Dukungan untuk memantau aktivitas di file descriptor |
| **QSsl** | Deklarasikan enums umum untuk semua kelas SSL di QtNetwork |
| **QSslCertificate** | Convenient API untuk sertifikat X509 |
| **QSslCipher** | Merupakan cipher kriptografi SSL |
| **QSslConfiguration** | Memegang konfigurasi dan keadaan koneksi SSL |

| | |
|---|---|
| **QSslError** | Kesalahan SSL |
| **QSslKey** | Interface untuk kunci pribadi dan publik |
| **QSslSocket** | SSL terenkripsi soket untuk kedua klien dan server |
| **QTcpServer** | Server berbasis TCP |
| **QTcpSocket** | TCP socket |
| **QUdpSocket** | UDP socket |
| **QUrl** | Convenient interface untuk bekerja dengan URL |
| **QUrlInfo** | Menyimpan informasi tentang URL |

## Jaringan Operasi Tingkat Tinggi untuk HTTP dan FTP

Network Access API adalah kumpulan kelas untuk melakukan operasi jaringan yang umum. API menyediakan lapisan abstraksi atas operasi-operasi tertentu dan protokol yang digunakan (misalnya, mendapatkan dan mengirim data melalui HTTP), dan hanya mengekspos kelas, fungsi, dan sinyal untuk konsep tingkat umum atau tinggi.

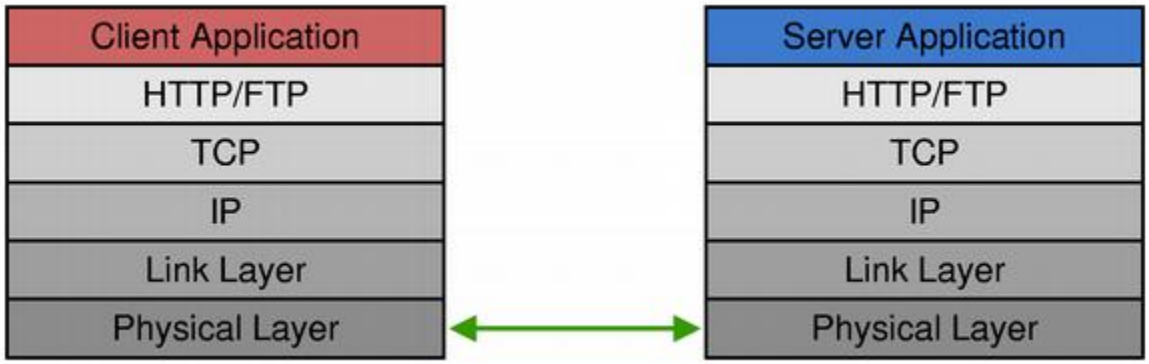
Request jaringan diwakili oleh kelas `QNetworkRequest`, yang juga bertindak sebagai wadah umum untuk informasi yang terkait dengan permintaan, seperti informasi header dan enkripsi yang digunakan. URL yang ditentukan jika permintaan dibangun menentukan protokol yang digunakan untuk permintaan. Saat ini HTTP, FTP dan URL file lokal yang didukung untuk upload dan download.

Koordinasi operasi jaringan dilakukan oleh kelas `QNetworkAccessManager`. Setelah permintaan telah dibuat, kelas ini digunakan untuk mengirimkan dan memancarkan sinyal untuk melaporkan prosesnya. Manajer juga mengkoordinasikan penggunaan cookie untuk menyimpan data pada klien, permintaan otentikasi, dan penggunaan proxy.

Balasan untuk permintaan jaringan diwakili oleh kelas `QNetworkReply`; ini diciptakan oleh `QNetworkAccessManager` bila ada permintaan yang dikirim. Sinyal yang diberikan oleh `QNetworkReply` dapat digunakan untuk memantau membalas setiap individu, atau pengembang dapat memilih untuk menggunakan sinyal manajer untuk tujuan ini bukan dan membuang referensi untuk balasan. Karena `QNetworkReply` adalah subclass dari `QIODevice`, balasan dapat ditangani secara synchronous atau asynchronous, yakni sebagai memblokir atau non-blocking operasi.

Setiap aplikasi atau library dapat membuat satu atau lebih instances dari `QNetworkAccessManager` untuk menangani komunikasi jaringan.

## Menulis FTP Klien dengan QFtp




FTP menggunakan dua koneksi jaringan, satu untuk perintah pengiriman dan satu untuk mentransfer data. Protokol FTP memiliki sebuah tempat dan membutuhkan klien untuk mengirim beberapa perintah sebelum transfer file berlangsung. Sambungan klien FTP tetap terbuka sepanjang sesi. Dalam setiap sesi, transfer ganda dapat terjadi.

Kelas `QFtp` menyediakan dukungan sisi klien untuk FTP. Ini memiliki karakteristik sebagai berikut:

- *Non-blocking behavior*. `QFtp` adalah asynchronous. Anda dapat menjadwalkan serangkaian perintah yang dieksekusi kemudian, ketika kontrol kembali ke loop event Qt.
- *Command IDs*. perintah Masing-masing memiliki nomor ID unik yang dapat Anda gunakan untuk mengikuti pelaksanaan perintah. Sebagai contoh, `QFtp` memancarkan sinyal `commandStarted()` dan `commandFinished()` dengan ID perintah untuk setiap perintah yang dijalankan.
- *Data transfer progress indicators*. `QFtp` memancarkan sinyal setiap kali data ditransfer (`QFtp::dataTransferProgress()`, `QNetworkReply::downloadProgress()`, dan `QNetworkReply::uploadProgress()`). Anda bisa terhubung ini sinyal ke `QProgressBar::setProgress()` atau `QProgressDialog::setProgress()`, misalnya.
- *QIODevice support*. Kelas mendukung upload dan download dari/ke `QIODevices`, selain API QbyteArray-based.

Ada dua cara utama menggunakan `QFtp`. Pendekatan yang paling umum adalah untuk melacak Command ID dan mengikuti pelaksanaan setiap perintah dengan menghubungkan ke sinyal-sinyal yang tepat. Pendekatan lain adalah untuk menjadwalkan semua perintah sekaligus dan hanya terhubung ke sinyal `done()`, yang dipancarkan ketika semua perintah terjadwal telah dilaksanakan. Pendekatan pertama membutuhkan lebih banyak pekerjaan, tetapi memberikan Anda kontrol lebih besar atas pelaksanaan perintah individu dan memungkinkan Anda untuk memulai perintah baru berdasarkan hasil dari perintah sebelumnya. Hal ini juga memungkinkan Anda untuk memberikan umpan balik secara rinci kepada pengguna.

Contoh FTP menggambarkan bagaimana menulis sebuah klien FTP. Menulis FTP Anda sendiri (atau HTTP) server adalah mungkin menggunakan kelas low-level `QTcpSocket` dan `QTcpServer`.

Kita akan mulai dengan sebuah contoh yang menunjukkan cara untuk mengambil satu file menggunakan `get()`. Contohnya adalah sebuah aplikasi konsol `ftpget` disebut yang download file remote ditentukan pada baris perintah. Mari kita mulai dengan `main()` fungsi:

```
int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv);
    QStringList args = QCoreApplication::arguments();
    if (args.count() != 2) {
        std::cerr << "Usage: ftpget url" << std::endl
                  << "Example:" << std::endl
                  << "    ftpget ftp://ftp.trolltech.com/mirrors"
                  << std::endl;
        return 1;
    }
    FtpGet getter;
    if (!getter.getFile(QUrl(args[1])))
        return 1;
    QObject::connect(&getter, SIGNAL(done()), &app,
SLOT(quit()));
    return app.exec();
}
```

Kita membuat QCoreApplication daripada subclass yang `QApplication` untuk menghindari menghubungkan di library *QtGui*. `QCoreApplication::argumen()` mengembalikan fungsi argumen baris perintah sebagai `QStringList`, dengan item pertama adalah nama program tersebut, dan setiap argumen Qt-spesifik seperti gaya dihapus. Jantung dari fungsi `main()` adalah konstruksi objek `FtpGet` dan `getFile()`. Jika panggilan berhasil, kita membiarkan loop event berjalan hingga proses download selesai.

Semua pekerjaan dilakukan oleh subclass FtpGet, yang didefinisikan sebagai berikut:

```
class FtpGet : public QObject
{
    Q_OBJECT
public:
    FtpGet(QObject *parent = 0);
    bool getFile(const QUrl &url);
signals:
    void done();
private slots:
    void ftpDone(bool error);
private:
    QFtp ftp;
    QFile file;
};
```

Kelas memiliki fungsi publik, `getFile()`, yang mengambil file ditentukan oleh URL. Kelas QUrl menyediakan antarmuka tingkat tinggi untuk mengekstraksi berbagai bagian URL, seperti nama file, path, protokol, dan port.

`FtpGet` memiliki slot pribadi, `ftpDone()`, yang disebut ketika transfer file selesai, dan sinyal `done()` yang memancarkan ketika file telah di-download. Kelas ini juga memiliki dua variabel pribadi: variabel `ftp`, tipe `QFtp`, yang menyatukan koneksi ke server FTP, dan variabel `file` yang digunakan untuk menulis file yang didownload ke disk.

```
FtpGet::FtpGet(QObject *parent)
    : QObject(parent)
{
    connect(&ftp, SIGNAL(done(bool)), this,
SLOT(ftpDone(bool)));
}
```

Dalam constructor, kita menghubungkan signal QFtp::done(bool) ke private slot `ftpDone(bool`. QFtp memancarkan `done(bool)` ketika telah selesai memproses semua permintaan. Parameter bool menunjukkan apakah sebuah kesalahan terjadi atau tidak.

```
bool FtpGet::getFile(const QUrl &url)
{
    if (!url.isValid()) {
        std::cerr << "Error: Invalid URL" << std::endl;
        return false;
    }
    if (url.scheme() != "ftp") {
        std::cerr << "Error: URL must start with 'ftp:'" <<
std::endl;
        return false;
    }
    if (url.path().isEmpty()) {
        std::cerr << "Error: URL has no path" << std::endl;
        return false;
    }
    QString localFileName = QFileInfo(url.path()).fileName();
    if (localFileName.isEmpty())
        localFileName = "ftpget.out";
    file.setFileName(localFileName);
    if (!file.open(QIODevice::WriteOnly)) {
        std::cerr << "Error: Cannot write file "
                  << qPrintable(file.fileName()) << ": "
                  << qPrintable(file.errorString()) <<
std::endl;
        return false;
    }
    ftp.connectToHost(url.host(), url.port(21));
    ftp.login();
    ftp.get(url.path(), &file);
    ftp.close();
    return true;
}
```

Fungsi `getFile()` dimulai dengan memeriksa URL yang disahkan jika masalah ditemui, fungsi akan mencetak pesan kesalahan ke `cerr` dan kembali `false` untuk menunjukkan bahwa download gagal.

Daripada memaksa pengguna untuk membuat nama file lokal, kita mencoba untuk membuat nama yang masuk akal menggunakan URL sendiri, dengan fallback dari `ftpget.out`. Jika kita gagal untuk membuka file, kita mencetak pesan kesalahan dan kembali palsu.

Selanjutnya, kita menjalankan urutan empat perintah FTP menggunakan obyek `QFtp`. `url.port(21)` memanggil kembali nomor port tertentu dalam URL, atau port 21 jika tidak ada yang ditentukan dalam URL itu sendiri. Karena tidak ada nama pengguna atau sandi diberikan untuk fungsi `login()`, sebuah login anonim dicoba. Argumen kedua untuk `get()` menentukan output I/O device.

Perintah-perintah FTP yang antri dan dijalankan dalam loop event Qt. Penyelesaian semua perintah yang ditunjukkan oleh sinyal QFtp `done(bool)`, yang terhubung ke `ftpDone(bool)` di konstruktor.

```
void FtpGet::ftpDone(bool error)
{
    if (error) {
        std::cerr << "Error: " << qPrintable(ftp.errorString())
                  << std::endl;
    } else {
        std::cerr << "File downloaded as "
                  << qPrintable(file.fileName()) << std::endl;
    }
    file.close();
    emit done();
}
```

Setelah perintah FTP telah dilaksanakan, kita tutup file dan memancarka signal `done()`. Ini mungkin tampak aneh bahwa kita menutup file tersebut di sini, daripada setelah `ftp.close()` panggilan pada akhir fungsi `getFile()`, tapi ingat bahwa perintah FTP dijalankan asynchronous dan baik mungkin sedang dalam proses setelah fungsi `getFile()` telah kembali. Hanya ketika objek QFtp signal `done()` yang dipancarkan kita tahu bahwa download selesai dan bahwa aman untuk menutup file tersebut.

QFtp menyediakan beberapa perintah FTP, termasuk `connectToHost()`, `login()`, `close()`, `list()`, `cd()`, `get()`, `put()`, `remove()`, `mkdir()`, `rmdir()`, dan `rename()`. Semua fungsi ini jadwal perintah FTP dan mengembalikan nomor ID yang mengidentifikasi perintah. Hal ini juga memungkinkan untuk mengontrol mode transfer (defaultnya adalah pasif) dan jenis transfer (defaultnya adalah biner).

Sewenang-wenang perintah FTP dapat dijalankan dengan menggunakan `rawCommand()`. Sebagai contoh, berikut adalah cara untuk mengeksekusi perintah `SITE CHMOD`:

```
ftp.rawCommand("SITE CHMOD 755 fortune");
```

QFtp memancarkan sinyal commandStarted(int) ketika mulai menjalankan perintah, dan memancarkan sinyal commandFinished(int, bool) ketika perintah selesai. Parameter int adalah nomor ID yang

mengidentifikasi perintah. Melacak nomor ID memungkinkan kita untuk memberikan umpan balik rinci kepada pengguna. Sebagai contoh:

```
bool FtpGet::getFile(const QUrl &url)
{
    ...
    connectId = ftp.connectToHost(url.host(), url.port(21));
    loginId = ftp.login();
    getId = ftp.get(url.path(), &file);
    closeId = ftp.close();
    return true;
}
void FtpGet::ftpCommandStarted(int id)
{
    if (id == connectId) {
        std::cerr << "Connecting..." << std::endl;
    } else if (id == loginId) {
        std::cerr << "Logging in..." << std::endl;
    ...
}
```

Cara lain untuk memberikan umpan balik untuk terhubung ke sinyal `QFtp stateChanged()`, yang dipancarkan setiap kali koneksi memasuki sesi baru (`QFtp::Connecting`, `QFtp::Connected`, `QFtp::LoggedIn`, dll).

Dalam sebagian besar aplikasi, kita hanya tertarik pada nasib urutan perintah secara keseluruhan dan bukan dalam perintah tertentu. Dalam kasus tersebut, kita hanya dapat terhubung ke sinyal `done(bool)`, yang dipancarkan apabila antrian perintah menjadi kosong.

Bila terjadi kesalahan, `QFtp` secara otomatis menghapus antrian perintah. Ini berarti bahwa jika koneksi atau gagal login, perintah yang mengikuti antrian tidak pernah dieksekusi.

Pada file .pro aplikasi, kita perlu menambahkan baris berikut untuk menghubungkan terhadap library QtNetwork:

```
QT += network
```

Sekarang kita akan mencoba membuat contoh lain. `spider` command-line program download semua file terletak di direktori FTP, rekursif download dari semua subdirektori pada direktori. Logika jaringan terletak di kelas `Spider`:

```
class Spider : public QObject
{
    Q_OBJECT
public:
    Spider(QObject *parent = 0);
    bool getDirectory(const QUrl &url);
signals:
    void done();
private slots:
    void ftpDone(bool error);
    void ftpListInfo(const QUrlInfo &urlInfo);
private:
    void processNextDirectory();
    QFtp ftp;
    QList<QFile *> openedFiles;
    QString currentDir;
    QString currentLocalDir;
    QStringList pendingDirs;
};
```

Direktori mulai ditentukan sebagai `QUrl` dan ditetapkan menggunakan fungsi `getDirectory()`.

```
Spider::Spider(QObject *parent)
    : QObject(parent)
{
    connect(&ftp, SIGNAL(done(bool)), this,
SLOT(ftpDone(bool)));
    connect(&ftp, SIGNAL(listInfo(const QUrlInfo &)),
            this, SLOT(ftpListInfo(const QUrlInfo &)));
}
```

Dalam konstruktor, kita mendirikan dua koneksi signal-slot. Signal `listInfo(const QUrlInfo &)` dipancarkan oleh `QFtp` ketika kita meminta daftar direktori (dalam `getDirectory()`) untuk setiap file yang mengambil. Sinyal ini dihubungkan ke slot `ftpListInfo()`, yang mendownload file yang berhubungan dengan URL itu diberikan.

```
bool Spider::getDirectory(const QUrl &url)
{
    if (!url.isValid()) {
        std::cerr << "Error: Invalid URL" << std::endl;
        return false;
    }
    if (url.scheme() != "ftp") {
        std::cerr << "Error: URL must start with 'ftp:'" <<
std::endl;
        return false;
    }
    ftp.connectToHost(url.host(), url.port(21));
    ftp.login();
    QString path = url.path();
    if (path.isEmpty())
        path = "/";
    pendingDirs.append(path);
    processNextDirectory();
    return true;
}
```

Ketika fungsi `getDirectory()` ini disebut, dimulai dengan melakukan beberapa pemeriksaan, dan jika semuanya baik-baik saja, ia mencoba untuk membangun koneksi FTP. Ia mencatat proses `processNextDirectory()` untuk mulai men-download direktori root.

```
void Spider::processNextDirectory()
{
    if (!pendingDirs.isEmpty()) {
        currentDir = pendingDirs.takeFirst();
        currentLocalDir = "downloads/" + currentDir;
        QDir(".").mkpath(currentLocalDir);
        ftp.cd(currentDir);
        ftp.list();
    } else {
        emit done();
    }
}
```

Fungsi `processNextDirectory()` mengambil isi direktori pertama jauh dari daftar `pendingDirs` dan menciptakan sebuah direktori yang sesuai pada sistem file lokal. Ia kemudian memberitahu objek QFtp untuk mengubah ke direktori dan ke daftar file-nya. Untuk setiap file yang proses `list()`, itu memancarkan signal `listinfo()` yang menyebabkan slot `ftpListInfo()` untuk dipanggil.

Jika tidak ada direktori lagi untuk diproses, fungsi memancarkan sinyal `done()` untuk menunjukkan bahwa download sudah selesai.

```
void Spider::ftpListInfo(const QUrlInfo &urlInfo)
{
    if (urlInfo.isFile()) {
        if (urlInfo.isReadable()) {
            QFile *file = new QFile(currentLocalDir + "/"
                                    + urlInfo.name());
            if (!file->open(QIODevice::WriteOnly)) {
                std::cerr << "Warning: Cannot write file "
                          << qPrintable(QDir::toNativeSeparators(
                                        file->fileName()))
                          << ": " << qPrintable(file-
>errorString())
                          << std::endl;
                return;
            }
            ftp.get(urlInfo.name(), file);
            openedFiles.append(file);
        }
    } else if (urlInfo.isDir() && !urlInfo.isSymLink()) {
        pendingDirs.append(currentDir + "/" + urlInfo.name());
    }
}
```

slot `ftpListInfo()` parameter `urlInfo` menyediakan informasi terinci tentang file remote. Jika file adalah file biasa (bukan sebuah direktori) dan dapat dibaca, kita panggil `get()` untuk

mendownloadnya. Objek QFile digunakan untuk mendownload dialokasikan menggunakan pointer baru dan untuk itu disimpan dalam daftar `openedFiles`.

Jika QUrlInfo memegang rincian direktori remote yang bukan link simbolis, kita menambahkan direktori ini ke daftar `pendingDirs`. kita melewatkan link simbolik karena mereka dapat dengan mudah menyebabkan rekursi tak terbatas.

```
void Spider::ftpDone(bool error)
{
    if (error) {
        std::cerr << "Error: " << qPrintable(ftp.errorString())
                  << std::endl;
    } else {
        std::cout << "Downloaded " << qPrintable(currentDir) << "
to "
                  << qPrintable(QDir::toNativeSeparators(

QDir(currentLocalDir).canonicalPath()));
    }
    qDeleteAll(openedFiles);
    openedFiles.clear();
    processNextDirectory();
}
```

Slot `ftpDone()` dipanggil saat semua perintah FTP sudah selesai atau jika kesalahan terjadi. kita menghapus objek QFile untuk mencegah kebocoran memori dan untuk menutup setiap file. Akhirnya, kita panggil `processNextDirectory()`.

Jika tidak ada kesalahan, urutan dan sinyal perintah FTP adalah sebagai berikut:

```
connectToHost(host, port)
login()
cd(directory_1)
list()
    emit listInfo(file_1_1)
        get(file_1_1)
    emit listInfo(file_1_2)
        get(file_1_2)
    ...
emit done()
...
cd(directory_N)
list()
    emit listInfo(file_N_1)
        get(file_N_1)
    emit listInfo(file_N_2)
        get(file_N_2)
    ...
emit done()
```

Jika terjadi kesalahan jaringan saat men-download, katakanlah, 20 file dalam sebuah direktori, file yang tersisa tidak akan diunduh. Jika kita ingin men-download sebagai file sebanyak mungkin, salah satu solusi akan jadwal operasi `GET` satu per satu dan untuk menunggu sinyal `done(bool)` sebelum penjadwalan operasi GET baru. Dalam `listinfo()`, kita hanya akan menambahkan nama file untuk `QStringList`, bukannya memanggil `get(`, dan `done(bool)` kita panggil `get()` pada file selanjutnya untuk men-download di QStringList tersebut. Urutan eksekusi maka akan terlihat seperti ini:

```
connectToHost(host, port)
login()
cd(directory_1)
list()
...
cd(directory_N)
list()
    emit listInfo(file_1_1)
    emit listInfo(file_1_2)
    ...
    emit listInfo(file_N_1)
    emit listInfo(file_N_2)
    ...
emit done()
get(file_1_1)
emit done()
get(file_1_2)
emit done()
...
get(file_N_1)
emit done()
get(file_N_2)
emit done()
...
```

Solusi lain adalah dengan menggunakan salah satu objek QFtp per file. Hal ini akan memungkinkan kita untuk men-download file secara paralel, melalui koneksi FTP terpisah.

```
int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv);
    QStringList args = QCoreApplication::arguments();
    if (args.count() != 2) {
        std::cerr << "Usage: spider url" << std::endl
                  << "Example:" << std::endl
                  << "   spider ftp://dl2.foss-id.web.id/"
                  << "leafnode" << std::endl;
        return 1;
    }
    Spider spider;
    if (!spider.getDirectory(QUrl(args[1])))
        return 1;
    QObject::connect(&spider, SIGNAL(done()), &app, SLOT(quit()));
    return app.exec();
}
```

Fungsi `main()` melengkapi program. Jika pengguna tidak menetapkan URL pada baris perintah, kita memberikan pesan kesalahan dan mengakhiri program.

Dalam kedua contoh FTP, data diambil menggunakan `get()` ditulis ke `QFile`. Ini tidak perlu terjadi. Jika kita ingin data dalam memori, kita bisa menggunakan `QBuffer`, subclass `QIODevice` yang membungkus sebuah `QByteArray`. Sebagai contoh:

```
QBuffer *buffer = new QBuffer;
buffer->open(QIODevice::WriteOnly);
ftp.get(urlInfo.name(), buffer);
```

Kita juga bisa menghilangkan I/O device argumen ke get (). Kelas QFtp kemudian memancarkan signal `readyRead()` setiap kali data baru tersedia, dan data dapat dibaca menggunakan `read()` atau `readAll()`.

## HTTP Client

Kelas QHttp mengimplementasikan sisi klien dari protokol HTTP di Qt. Ini menyediakan berbagai fungsi untuk melakukan operasi HTTP yang paling umum, termasuk `get()` dan `post()`, dan menyediakan sarana untuk mengirim permintaan HTTP. Jika Anda membaca bagian sebelumnya tentang QFtp, Anda akan menemukan bahwa ada banyak kesamaan antara QFtp dan QHttp.

Kelas QHttp bekerja secara asynchronous. Ketika kita memanggil fungsi seperti fungsi `get()` atau `post()`, dan transfer data yang terjadi kemudian, ketika kontrol kembali ke loop event Qt. Hal ini memastikan bahwa antarmuka pengguna aplikasi tetap responsif saat HTTP request sedang diproses.

Kita akan meninjau contoh aplikasi konsol `httpget` disebut yang menunjukkan cara men-download file menggunakan protokol HTTP. Hal ini sangat mirip dengan contoh `ftpget` dari bagian sebelumnya, baik dalam fungsi dan pelaksanaan, jadi kami tidak akan menampilkan file header.

```
HttpGet::HttpGet(QObject *parent)
    : QObject(parent)
{
      connect(&http, SIGNAL(done(bool)), this,
SLOT(httpDone(bool)));
}
```

Dalam constructor, kita menghubungkan objek QHttp signal `done(bool)` ke private slot httpDone(bool).

Perhatikan kode program berikut ini.

```
bool HttpGet::getFile(const QUrl &url)
{
    if (!url.isValid()) {
        std::cerr << "Error: Invalid URL" << std::endl;
        return false;
    }
    if (url.scheme() != "http") {
        std::cerr << "Error: URL must start with 'http:'" <<
std::endl;
        return false;
    }
    if (url.path().isEmpty()) {
        std::cerr << "Error: URL has no path" << std::endl;
        return false;
    }
    QString localFileName = QFileInfo(url.path()).fileName();
    if (localFileName.isEmpty())
        localFileName = "httpget.out";
    file.setFileName(localFileName);
    if (!file.open(QIODevice::WriteOnly)) {
        std::cerr << "Error: Cannot write file "
                  << qPrintable(file.fileName()) << ": "
                  << qPrintable(file.errorString()) << std::endl;
        return false;
    }
    http.setHost(url.host(), url.port(80));
    http.get(url.path(), &file);
    http.close();
    return true;
}
```

Fungsi `getFile()` yang sama melakukan pengecekan kesalahan sebagai `FtpGet::getFile()` ditampilkan sebelumnya dan menggunakan pendekatan yang sama untuk memberikan file nama lokal. Ketika mengambil dari situs web, login tidak diperlukan, jadi kita hanya mengatur host dan port (menggunakan port standar HTTP 80 jika tidak ada yang ditentukan dalam URL) dan men-download data ke file tersebut, karena argumen kedua untuk `QHttp::get()` menentukan output I/O device.

Penyelesaian permintaan ditunjukkan oleh QHttp signal `done(bool)`, yang telah terhubung ke `httpDone(bool)` di konstruktor.

```
void HttpGet::httpDone(bool error)
{
    if (error) {
        std::cerr << "Error: " << qPrintable(http.errorString())
                  << std::endl;
    } else {
        std::cerr << "File downloaded as "
                  << qPrintable(file.fileName()) << std::endl;
    }
    file.close();
    emit done();
}
```

Setelah permintaan HTTP selesai, kita menutup file, memberitahu pengguna jika kesalahan terjadi.

Fungsi `main()` ini sangat mirip dengan yang digunakan oleh `ftpget`:

```
int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv);
    QStringList args = QCoreApplication::arguments();
    if (args.count() != 2) {
        std::cerr << "Usage: httpget url" << std::endl
                  << "Example:" << std::endl
                  << "   httpget
http://doc.trolltech.com/index.html"
                  << std::endl;
        return 1;
    }
    HttpGet getter;
    if (!getter.getFile(QUrl(args[1])))
        return 1;
    QObject::connect(&getter, SIGNAL(done()), &app, SLOT(quit()));
    return app.exec();
}
```

Kelas QHttp menyediakan banyak operasi, termasuk `setHost()`, `get()`, `post()`, dan `head()`. Jika situs memerlukan otentikasi, `setUser()` dapat digunakan untuk menyediakan nama pengguna dan sandi. QHttp dapat menggunakan soket yang disediakan oleh programmer daripada QTcpSocket internal sendiri. Hal ini memungkinkan untuk menggunakan secure `QSslSocket` untuk mencapai HTTP melalui SSL atau TLS.

Untuk mengirim daftar “name = value" pasangan untuk skrip CGI, kita bisa menggunakan `post()`:

```
http.setHost("www.nice.or.id");
http.post("/cgi/somescript.py", "x=200&y=320", &file);
```

Kita bisa melewati data baik sebagai string 8-bit atau dengan menyediakan suatu QIODevice terbuka, seperti QFile. Untuk lebih banyak kontrol, kita dapat menggunakan fungsi `request()`, yang menerima header HTTP. Sebagai contoh:

```
QHttpRequestHeader header("POST", "/search.html");
header.setValue("Host", "www.nice.or.id");
header.setContentType("application/x-www-form-urlencoded");
http.setHost("www.nice.or.id");
http.request(header, "qt-interest=on&search=opengl");
```

QHttp memancarkan signal `requestStarted(int)` ketika mulai melaksanakan permintaan, dan memancarkan signal `requestFinished(int,bool)` bila permintaan tersebut telah selesai.

Parameter int merupakan nomor ID yang mengidentifikasi permintaan. Melacak nomor ID memungkinkan kita untuk memberikan umpan balik rinci kepada pengguna.

Dalam sebagian besar aplikasi, kita hanya ingin tahu apakah urutan seluruh permintaan selesai dengan sukses atau tidak. Hal ini mudah dicapai dengan menghubungkan ke signal `done(bool)`, yang dipancarkan ketika antrian permintaan menjadi kosong.

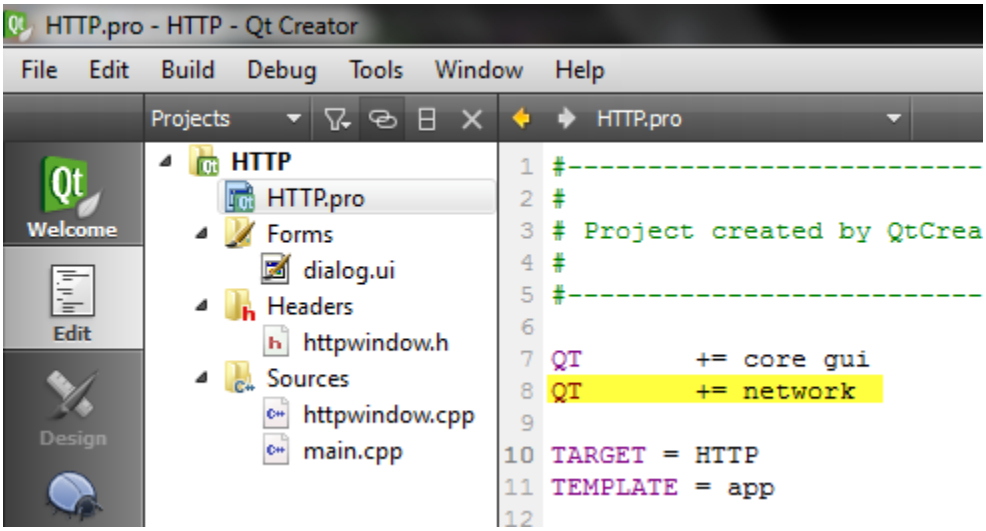
Ketika suatu kesalahan terjadi, antrian permintaan secara otomatis dihapus. Tetapi jika permintaan baru setelah kesalahan telah terjadi menggunakan objek QHttp yang sama, permintaan ini akan masuk dalam antrian dan dikirim seperti biasa.

Seperti QFtp, QHttp menyediakan signal `readyRead()` serta fungsi `read()` dan `readAll()` yang dapat kita gunakan bukan menentukan I / O device.

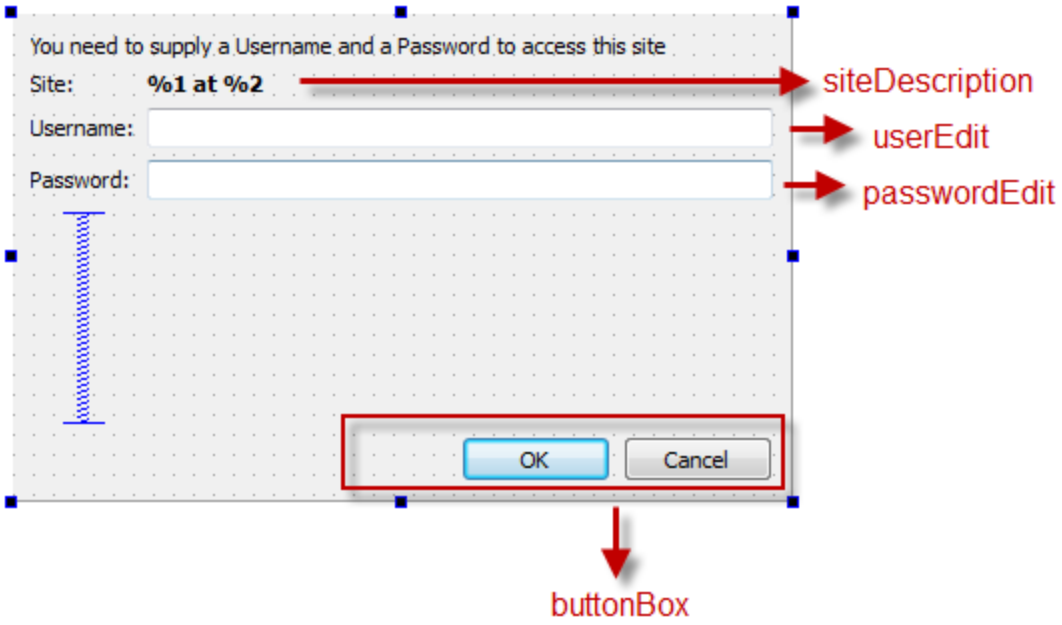
Sebagai contoh kita akan membuat sebuah aplikasi downloader, contoh ini menunjukkan klien HTTP sederhana yang menunjukkan bagaimana mendownload file ditentukan oleh URL dari host remote.

Buat sebuah project Dialog dengan nama dialog untuk file .ui nama project **HTTP** dan class **HttpWindow**.

1. Jangan lupa menambahkan modul *QtNetwork* ke file proyek - http.pro.

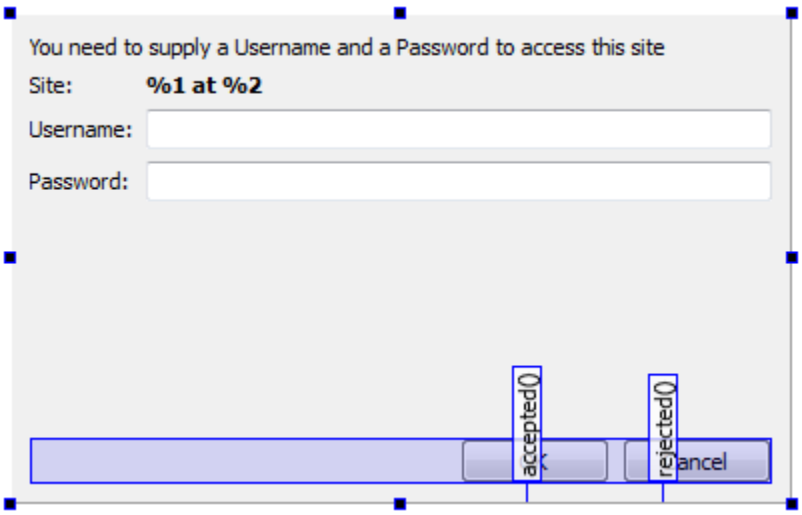



2. Desain ui seperti berikut.




Nantinya UI ini akan tampil ketika membutuhkan autentifikasi untuk mendownload sebuah file. %1 berfungsi untuk mengambil nama file dan %2 untuk mengambil url tempat file berada.

3. Buat slot `accepted()` pada tombol OK dan `rejected()` pada tombol Cacel.



4. Pada file header tambahkan kode program berikut :

```
#ifndef HTTPWINDOW_H
#define HTTPWINDOW_H

#include <QDialog>
#include <QNetworkAccessManager>
#include <QUrl>

class QDialogButtonBox;
class QFile;
class QLabel;
class QLineEdit;
class QProgressDialog;
class QPushButton;
class QSslError;
class QAuthenticator;
class QNetworkReply;
class HttpWindow : public QDialog
{
    Q_OBJECT

public:
    HttpWindow(QWidget *parent = 0);
    void startRequest(QUrl url);

private slots:
    void downloadFile();
    void cancelDownload();
    void httpFinished();
    void httpReadyRead();
    void updateDataReadProgress(qint64 bytesRead, qint64 totalBytes);
    void enableDownloadButton();
    void slotAuthenticationRequired(QNetworkReply*,QAuthenticator *);
#ifndef QT_NO_OPENSSL
    void sslErrors(QNetworkReply*,const QList<QSslError> &errors);
#endif

private:
    QLabel *statusLabel;
    QLabel *urlLabel;
    QLineEdit *urlLineEdit;
    QProgressDialog *progressDialog;
    QPushButton *downloadButton;
    QPushButton *quitButton;
    QDialogButtonBox *buttonBox;
    QUrl url;
    QNetworkAccessManager qnam;
    QNetworkReply *reply;
    QFile *file;
    int httpGetId;
    bool httpRequestAborted;
};
#endif // HTTPWINDOW_H
```

5. Pada file httpwindow.cpp masukan kode program berikut ini.

```
#include <QtGui>
#include <QtNetwork>
#include "httpwindow.h"
#include "ui_dialog.h"
HttpWindow::HttpWindow(QWidget *parent)
    : QDialog(parent)
{
#ifndef QT_NO_OPENSSL
    urlLineEdit = new QLineEdit("https://nice.or.id/");
#else
    urlLineEdit = new QLineEdit("http://nice.or.id/");
#endif
    urlLabel = new QLabel(tr("&URL:"));
    urlLabel->setBuddy(urlLineEdit);
    statusLabel = new QLabel(tr("Please enter the URL of a file you
want to download."));
    downloadButton = new QPushButton(tr("Download"));
    downloadButton->setDefault(true);
    quitButton = new QPushButton(tr("Quit"));
    quitButton->setAutoDefault(false);
    buttonBox = new QDialogButtonBox;
    buttonBox->addButton(downloadButton,
QDialogButtonBox::ActionRole);
    buttonBox->addButton(quitButton, QDialogButtonBox::RejectRole);
    progressDialog = new QProgressDialog(this);
     connect(urlLineEdit, SIGNAL(textChanged(QString)),
             this, SLOT(enableDownloadButton()));
    connect(&qnam,
SIGNAL(authenticationRequired(QNetworkReply*,QAuthenticator*)),
             this,
SLOT(slotAuthenticationRequired(QNetworkReply*,QAuthenticator*)));
#ifndef QT_NO_OPENSSL
    connect(&qnam,
SIGNAL(sslErrors(QNetworkReply*,QList<QSslError>)),
             this,
SLOT(sslErrors(QNetworkReply*,QList<QSslError>)));
#endif
    connect(progressDialog, SIGNAL(canceled()), this,
SLOT(cancelDownload()));
    connect(downloadButton, SIGNAL(clicked()), this,
SLOT(downloadFile()));
    connect(quitButton, SIGNAL(clicked()), this, SLOT(close()));
    QHBoxLayout *topLayout = new QHBoxLayout;
    topLayout->addWidget(urlLabel);
    topLayout->addWidget(urlLineEdit);
    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout->addLayout(topLayout);
    mainLayout->addWidget(statusLabel);
    mainLayout->addWidget(buttonBox);
    setLayout(mainLayout);
    setWindowTitle(tr("HTTP"));
    urlLineEdit->setFocus();
}
```

```
void HttpWindow::startRequest(QUrl url)
{
    reply = qnam.get(QNetworkRequest(url));
    connect(reply, SIGNAL(finished()),
            this, SLOT(httpFinished()));
    connect(reply, SIGNAL(readyRead()),
            this, SLOT(httpReadyRead()));
    connect(reply, SIGNAL(downloadProgress(qint64,qint64)),
            this, SLOT(updateDataReadProgress(qint64,qint64)));
}
void HttpWindow::downloadFile()
{
    url = urlLineEdit->text();
    QFileInfo fileInfo(url.path());
    QString fileName = fileInfo.fileName();
    if (fileName.isEmpty())
        fileName = "index.html";
    if (QFile::exists(fileName)) {
        if (QMessageBox::question(this, tr("HTTP"),
                                  tr("There already exists a file
called %1 in "
                                     "the current directory.
Overwrite?").arg(fileName),
                                  QMessageBox::Yes|QMessageBox::No,
QMessageBox::No)
            == QMessageBox::No)
            return;
        QFile::remove(fileName);
    }
    file = new QFile(fileName);
    if (!file->open(QIODevice::WriteOnly)) {
        QMessageBox::information(this, tr("HTTP"),
                                 tr("Unable to save the file %1:
%2.")
                                 .arg(fileName).arg(file-
>errorString()));
        delete file;
        file = 0;
        return;
    }
    progressDialog->setWindowTitle(tr("HTTP"));
    progressDialog->setLabelText(tr("Downloading
%1.").arg(fileName));
    downloadButton->setEnabled(false);
    // schedule the request
    httpRequestAborted = false;
    startRequest(url);
}
void HttpWindow::cancelDownload()
{
    statusLabel->setText(tr("Download canceled."));
    httpRequestAborted = true;
    reply->abort();
    downloadButton->setEnabled(true);
}
```

```
void HttpWindow::httpFinished()
{
    if (httpRequestAborted) {
        if (file) {
            file->close();
            file->remove();
            delete file;
            file = 0;
        }
        reply->deleteLater();
        progressDialog->hide();
        return;
    }
    progressDialog->hide();
    file->flush();
    file->close();
    QVariant redirectionTarget = reply-
>attribute(QNetworkRequest::RedirectionTargetAttribute);
    if (reply->error()) {
         file->remove();
        QMessageBox::information(this, tr("HTTP"),
                                 tr("Download failed: %1.")
                                 .arg(reply->errorString()));
        downloadButton->setEnabled(true);
    } else if (!redirectionTarget.isNull()) {
        QUrl newUrl = url.resolved(redirectionTarget.toUrl());
        if (QMessageBox::question(this, tr("HTTP"),
                                  tr("Redirect to %1
?").arg(newUrl.toString()),
                                  QMessageBox::Yes | QMessageBox::No)
== QMessageBox::Yes) {
            url = newUrl;
            reply->deleteLater();
            file->open(QIODevice::WriteOnly);
            file->resize(0);
            startRequest(url);
            return;
        }
    } else {
        QString fileName = QFileInfo(QUrl(urlLineEdit-
>text()).path()).fileName();
        statusLabel->setText(tr("Downloaded %1 to current
directory.").arg(fileName));
        downloadButton->setEnabled(true);
    }
     reply->deleteLater();
     reply = 0;
     delete file;
     file = 0;
 }
```

```
void HttpWindow::httpReadyRead()
{
    if (file)
        file->write(reply->readAll());
}
void HttpWindow::updateDataReadProgress(qint64 bytesRead, qint64
totalBytes)
{
    if (httpRequestAborted)
        return;
    progressDialog->setMaximum(totalBytes);
    progressDialog->setValue(bytesRead);
}
void HttpWindow::enableDownloadButton()
{
    downloadButton->setEnabled(!urlLineEdit->text().isEmpty());
}
void
HttpWindow::slotAuthenticationRequired(QNetworkReply*,QAuthenticator
*authenticator)
{
    QDialog dlg;
    Ui::Dialog ui;
    ui.setupUi(&dlg);
    dlg.adjustSize();
    ui.siteDescription->setText(tr("%1 at %2").arg(authenticator-
>realm()).arg(url.host()));
    // Did the URL have information? Fill the UI
    // This is only relevant if the URL-supplied credentials were
wrong
    ui.userEdit->setText(url.userName());
    ui.passwordEdit->setText(url.password());
    if (dlg.exec() == QDialog::Accepted) {
        authenticator->setUser(ui.userEdit->text());
        authenticator->setPassword(ui.passwordEdit->text());
    }
}
#ifndef QT_NO_OPENSSL
void HttpWindow::sslErrors(QNetworkReply*,const QList<QSslError>
&errors)
{
    QString errorString;
    foreach (const QSslError &error, errors) {
        if (!errorString.isEmpty())
            errorString += ", ";
        errorString += error.errorString();
    }
    if (QMessageBox::warning(this, tr("HTTP"),
                             tr("One or more SSL errors has occurred:
%1").arg(errorString),
                             QMessageBox::Ignore | QMessageBox::Abort)
== QMessageBox::Ignore) {
        reply->ignoreSslErrors();
    }
}
#endif
```

6. Jalankan kode program di atas.


Pada kode program diatas semua kelas di letakan di file header dan semua entry point dan logika diletakan pada file httpwindow.cpp

Ketika ada file yang sudha diunduh dengan nama yang sama, maka akan tampil peringatan bahwa file yang didownload sudah ada, apakah ingin di timpah (tampil message dialog Yes/No). Isian default lineedit adalah nice.or.id dan pada file httpwindow.cpp terdapat QNetworkReply yang Berisi data dan header untuk request dikirim dengan QNetworkAccessManager.

Untuk keterangan kelas-kelas dari kode program di atas bisa Anda ketaui dengan melihat table kelas-kelas Qt Network.

## Menggunakan TCP dengan QTcpSocket dan QTcpServer

TCP (Transmission Control Protocol) adalah low-level network protocol yang paling banyak digunakan oleh protokol Internet, termasuk HTTP dan FTP, untuk transfer data. Ini berorientasi stream, connection-oriented transport protocol. Hal ini sangat cocok untuk transmisi data kontinu.

Kelas `QTcpSocket` menyediakan sebuah antarmuka untuk TCP. Anda dapat menggunakan `QTcpSocket` untuk mengimplementasikan protokol jaringan standar seperti POP3, SMTP, dan NNTP, serta protokol kustom.

Koneksi TCP harus ditetapkan untuk sebuah host remote dan port sebelum transfer data dapat dimulai. Setelah koneksi telah ditetapkan, alamat IP dan port dari peer yang tersedia melalui `QTcpSocket::peerAddress()` dan `QTcpSocket::peerPort()`. Setiap saat, peer bisa menutup sambungan, dan data transfer kemudian akan segera berhenti.

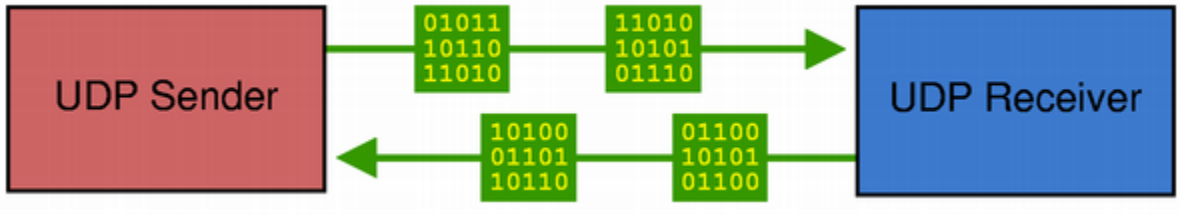
`QTcpSocket` bekerja asynchronous dan memancarkan sinyal untuk melaporkan perubahan status dan kesalahan, seperti `QNetworkAccessManager` dan `QFtp`. Hal ini bergantung pada loop event untuk mendeteksi data yang masuk dan secara otomatis flush data keluar. Anda dapat menulis data ke soket menggunakan `QTcpSocket::write()`, dan membaca data menggunakan `QTcpSocket::read()`. `QTcpSocket` merupakan dua independen aliran data: satu untuk membaca dan satu untuk menulis.

Sejak QIODevice mewarisi QTcpSocket, Anda dapat menggunakannya dengan QTextStream dan QDataStream. Ketika membaca dari QTcpSocket, Anda harus memastikan bahwa data cukup tersedia dengan memanggil QTcpSocket:: bytesAvailable () terlebih dahulu.

Jika Anda perlu menangani koneksi TCP yang masuk (misalnya, dalam aplikasi server), menggunakan kelas `QTcpServer`. Call `QTcpServer::mendengarkan()` untuk mengatur server, dan terhubung ke `QTcpServer::newConnection()` sinyal, yang dipancarkan sekali untuk setiap klien yang terhubung. Dalam slot Anda, hubungi `QTcpServer::nextPendingConnection()` untuk menerima koneksi dan menggunakan QTcpSocket kembali untuk berkomunikasi dengan klien.

Fortune client dan Fortune Server Fortune contoh yang menunjukkan bagaimana menggunakan `QTcpSocket` dan `QTcpServer` untuk menulis TCP aplikasi client-server.

## Menggunakan UDP dengan QudpSocket




Kelas `QUdpSocket` memungkinkan Anda untuk mengirim dan menerima datagrams UDP. Ini mewarisi `QAbstractSocket`, dan karena itu saham sebagian besar antarmuka QTcpSocket. Perbedaan utama adalah bahwa QUdpSocket transfer data datagrams bukan sebagai arus kontinu data. Singkatnya, datagram adalah paket data dengan ukuran terbatas (biasanya lebih kecil dari 512 bytes), yang berisi alamat IP dan port pengirim dan penerima datagram di samping data yang ditransfer.

`QUdpSocket` mendukung IPv4 broadcasting. Broadcastingsering digunakan untuk mengimplementasikan protokol penemuan jaringan, seperti menemukan host pada jaringan yang memiliki ruang disk paling kosong. Satu host menyiarkan datagram ke jaringan yang semua host lain terima. Setiap host yang menerima permintaan kemudian mengirimkan jawaban kembali ke pengirim dengan jumlah ruang disk kosong. originator yang menunggu sampai menerima balasan dari semua host, dan kemudian dapat memilih server dengan ruang bebas yang paling untuk menyimpan data. Untuk broadcast datagram, cukup kirimkan ke alamat khusus `QHostAddress::Broadcast` (255.255.255.255), atau untuk alamat broadcast jaringan lokal Anda.

`QUdpSocket::bind()` menyiapkan soket untuk menerima datagram yang masuk, seperti `QTcpServer::listen()` untuk server TCP. Setiap kali satu atau lebih datagrams tiba, QUdpSocket memancarkan `readyRead()`. Panggil `QUdpSocket::readDatagram()` untuk membaca datagram.

## Dukungan untuk Proxy Jaringan

Jaringan komunikasi dengan Qt dapat dilakukan melalui proxy, yang langsung atau filter lalu lintas jaringan antara koneksi lokal dan remote.

Masing-masing proxy diwakili oleh kelas `QNetworkProxy`, yang digunakan untuk menggambarkan dan mengkonfigurasi koneksi ke proxy. jenis proxy yang beroperasi di tingkat yang berbeda dari komunikasi jaringan yang didukung, dengan dukungan SOCKS 5 proxy yang memungkinkan lalu lintas jaringan pada tingkat rendah, dan HTTP dan FTP proxy bekerja pada tingkat protokol. Untuk menggunakannya Anda cukup menambahkan library `<QNetworkProxy>`.

Proxy dapat diaktifkan pada basis per-socket atau untuk semua komunikasi jaringan dalam sebuah aplikasi. Sebuah socket yang baru dibuka dapat dibuat untuk menggunakan proxy dengan memanggil fumgsi `QAbstractSocket::setProxy()` sebelum tersambung. Aplikasi proxy bisa diaktifkan untuk semua koneksi soket berikutnya melalui penggunaan fungsi `QNetworkProxy:: setApplicationProxy ()`.

Proxy Factory digunakan untuk membuat kebijakan untuk penggunaan proxy. `QNetworkProxyFactory` proxy menyediakan berdasarkan permintaan untuk jenis proxy tertentu. Permintaan sendiri dikodekan dalam obyek `QNetworkProxyQuery` yang memungkinkan proxy akan dipilih berdasarkan kriteria utama, seperti tujuan proxy (TCP, UDP, TCP server, URL request), port lokal, remote host dan port, dan protokol di menggunakan (HTTP, FTP, dll).

`QNetworkProxyFactory::proxyForQuery()` digunakan untuk query factory secara langsung. Kebijakan aplikasi untuk proxy dapat diimplementasikan dengan melewatkan factory untuk `QNetworkProxyFactory::setApplicationProxyFactory()` dan kebijakan proxy kustom dapat dibuat dengan subclassing `QNetworkProxyFactory`.

## Dukungan Bearer Management

Bearer Management mengontrol konektivitas perangkat sehingga aplikasi dapat mulai atau berhenti antarmuka jaringan dan berkelana transparan antara titik akses.

Kelas `QNetworkConfigurationManager` mengelola daftar konfigurasi jaringan yang dikenal ke perangkat. Suatu konfigurasi jaringan menggambarkan set parameter yang digunakan untuk memulai sebuah antarmuka jaringan dan diwakili oleh kelas `QNetworkConfiguration`.

Sebuah antarmuka jaringan dimulai oleh sebuah openning `QNetworkSession` berdasarkan konfigurasi jaringan. Dalam situasi yang paling membuat sesi jaringan berbasis pada platform konfigurasi jaringan tertentu default adalah tepat. Konfigurasi jaringan default dikembalikan oleh fungsi `QNetworkConfigurationManager::defaultConfiguration()`.

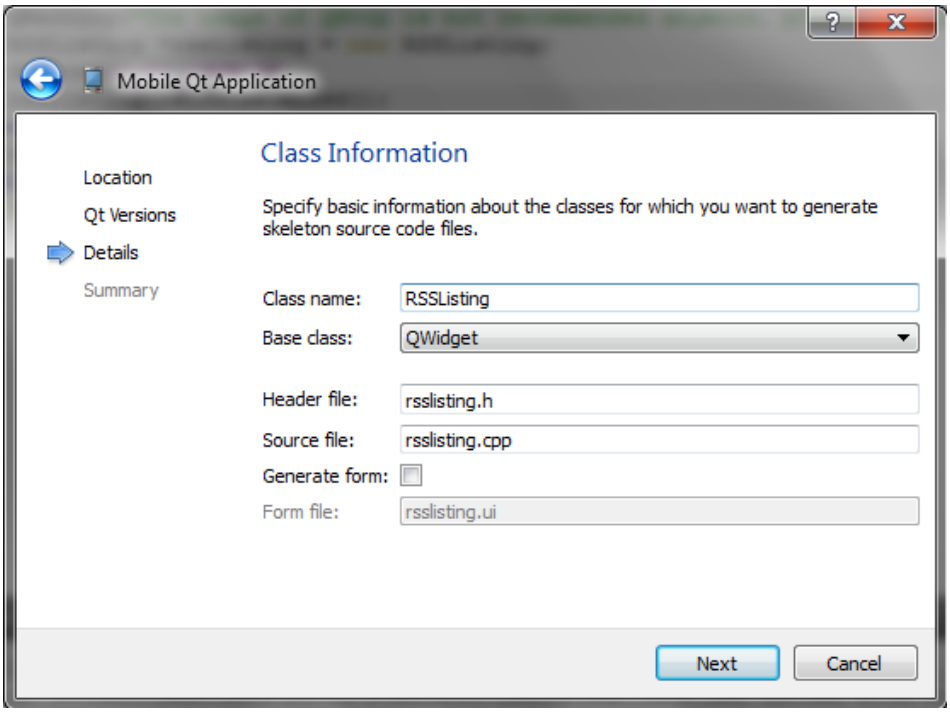
Pada beberapa platform ini merupakan persyaratan platform untuk aplikasi membuka sesi jaringan sebelum operasi jaringan dapat dilakukan. Hal ini dapat diuji oleh QNetworkConfigurationManager::NetworkSessionRequired nilai yang dikembalikan oleh fungsi `QNetworkConfigurationManager::capabilities()`.

# Membaca XML dengan QxmlStreamReader

Menggunakan QXmlStreamReader adalah cara tercepat dan termudah untuk membaca XML di Qt. Karena parser bekerja secara bertahap, sangat berguna untuk menemukan semua kejadian tag yang diberikan dalam dokumen XML, untuk membaca file yang besar yang mungkin tidak cocok di memori, dan untuk mengisi struktur data kustom untuk mencerminkan isi dokumen XML. menggunakan QXmlStreamReader memungkinkan peng-kode-an sederhana dan lebih cepat.

Kali ini kita coba untuk membuat aplikasi Ambil RSS dari sebuah situs web. Contoh ini menunjukkan cara membuat widget yang menampilkan berita dari sumber berita (RSS).

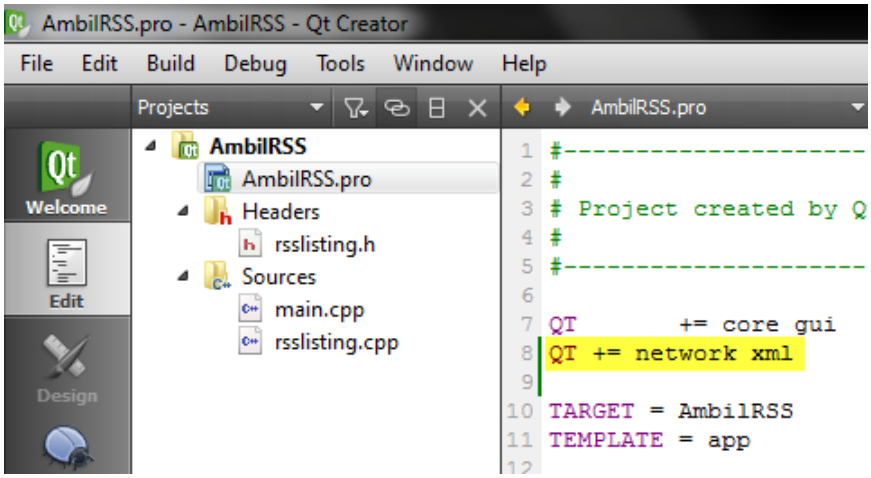
Buat project Qt Mobile baru dengan nama AmbilRSS dan Kelas RSSListing dengan Base Class QWidget.




Setelah membuat project, ada 4 buah file yang akan dibuat, yaitu AmbilRSS.pro, rsslisting.h, rsslisting.cpp, dan main.cpp.

Tambahkan network xml pada file project (AmbilRSS.pro)

QT += network xml




Pada file rsslisting.h tambah beberapa baris kode berikut ini:

```
#ifndef RSSLISTING_H
#define RSSLISTING_H

#include <QHttp>
#include <QWidget>
#include <QBuffer>
#include <QXmlStreamReader>

#ifdef Q_OS_SYMBIAN
// Bearer
#include <QNetworkConfigurationManager>
#include <QNetworkSession>
#include <QPointer>
// QtMobility namespace
QTM_USE_NAMESPACE
#endif

QT_BEGIN_NAMESPACE
class QLineEdit;
class QTreeWidget;
class QTreeWidgetItem;
class QPushButton;
QT_END_NAMESPACE

class RSSListing : public QWidget
{
    Q_OBJECT
public:
    RSSListing(QWidget *widget = 0);
public slots:
    void fetch();
    void finished(int id, bool error);
    void readData(const QHttpResponseHeader &);
    void itemActivated(QTreeWidgetItem * item);
```

```
private:
    void parseXml();
    QXmlStreamReader xml;
    QString currentTag;
    QString linkString;
    QString titleString;
    QHttp http;
    int connectionId;

    QLineEdit *lineEdit;
    QTreeWidget *treeWidget;
    QPushButton *abortButton;
    QPushButton *fetchButton;

#ifdef Q_OS_SYMBIAN
    // for bearer management
    QPointer<QNetworkSession> m_session;
#endif
};
```

Pada kode program di atas terdapat library <QXmlStreamReader> yang berfungsi untuk parsing XML dan terdapat kelas-kelas untuk GUI.

Selanjutnya pada file rsslisting.cpp tambahkan beberapa baris kode berikut ini.

```
#include <QtCore>
#include <QtGui>
#include <QtNetwork>
#include "rsslisting.h"
RSSListing::RSSListing(QWidget *parent)
    : QWidget(parent)
{
#ifdef Q_OS_SYMBIAN
    // Set Internet Access Point
    QNetworkConfigurationManager manager;
    const bool canStartIAP = manager.capabilities() &
QNetworkConfigurationManager::CanStartAndStopInterfaces;
    // Is there default access point, use it
    QNetworkConfiguration cfg = manager.defaultConfiguration();
    if (!cfg.isValid() || !canStartIAP) {
        // Available Access Points not found
        QMessageBox::warning(this, "Error", "No access point");
        return;
    }
    m_session = new QNetworkSession(cfg);
    m_session->open();
    m_session->waitForOpened();
#endif
    lineEdit = new QLineEdit(this);
    lineEdit->setText("http://nice.or.id/activity/feed/");
    fetchButton = new QPushButton(tr("Ambil"), this);
    abortButton = new QPushButton(tr("Batal"), this);
    abortButton->setEnabled(false);
    treeWidget = new QTreeWidget(this);
    connect(treeWidget,
```

```
SIGNAL(itemActivated(QTreeWidgetItem*,int)),
            this, SLOT(itemActivated(QTreeWidgetItem*)));
    QStringList headerLabels;
    headerLabels << tr("Judul") << tr("Link");
    treeWidget->setHeaderLabels(headerLabels);
    treeWidget->header()-
>setResizeMode(QHeaderView::ResizeToContents);
    connect(&http, SIGNAL(readyRead(QHttpResponseHeader)),
             this, SLOT(readData(QHttpResponseHeader)));
    connect(&http, SIGNAL(requestFinished(int,bool)),
             this, SLOT(finished(int,bool)));
    connect(lineEdit, SIGNAL(returnPressed()), this, SLOT(fetch()));
    connect(fetchButton, SIGNAL(clicked()), this, SLOT(fetch()));
    connect(abortButton, SIGNAL(clicked()), &http, SLOT(abort()));
    QVBoxLayout *layout = new QVBoxLayout(this);
    QHBoxLayout *hboxLayout = new QHBoxLayout;
    hboxLayout->addWidget(lineEdit);
    hboxLayout->addWidget(fetchButton);
    hboxLayout->addWidget(abortButton);
    layout->addLayout(hboxLayout);
    layout->addWidget(treeWidget);
    setWindowTitle(tr("Ambil RSS"));
#if !defined(Q_OS_SYMBIAN) && !defined(Q_WS_MAEMO_5)
    resize(640,480);
#endif
}
void RSSListing::fetch()
{
    lineEdit->setReadOnly(true);
    fetchButton->setEnabled(false);
    abortButton->setEnabled(true);
    treeWidget->clear();
    xml.clear();
    QUrl url(lineEdit->text());
    http.setHost(url.host());
    connectionId = http.get(url.path());
}
void RSSListing::readData(const QHttpResponseHeader &resp)
{
    if (resp.statusCode() != 200)
        http.abort();
    else {
        xml.addData(http.readAll());
        parseXml();
    }
}
void RSSListing::finished(int id, bool error)
{
    if (error) {
        QMessageBox::warning(this, "Error", http.errorString());
        lineEdit->setReadOnly(false);
        abortButton->setEnabled(false);
        fetchButton->setEnabled(true);
    }
    else if (id == connectionId) {
        lineEdit->setReadOnly(false);
        abortButton->setEnabled(false);
        fetchButton->setEnabled(true);
    }
}
```

```
void RSSListing::parseXml()
{
    while (!xml.atEnd()) {
        xml.readNext();
        if (xml.isStartElement()) {
            if (xml.name() == "item")
                linkString =
xml.attributes().value("rss:about").toString();
            currentTag = xml.name().toString();
        } else if (xml.isEndElement()) {
            if (xml.name() == "item") {

                QTreeWidgetItem *item = new QTreeWidgetItem;
                item->setText(0, titleString);
                item->setText(1, linkString);
                treeWidget->addTopLevelItem(item);
                titleString.clear();
                linkString.clear();
            }
        } else if (xml.isCharacters() && !xml.isWhitespace()) {
            if (currentTag == "title")
                titleString += xml.text().toString();
            else if (currentTag == "link")
                linkString += xml.text().toString();
        }
    }
    if (xml.error() && xml.error() !=
QXmlStreamReader::PrematureEndOfDocumentError) {
        qWarning() << "XML ERROR:" << xml.lineNumber() << ": " <<
xml.errorString();
        http.abort();
    }
}
void RSSListing::itemActivated(QTreeWidgetItem * item)
{
    QDesktopServices::openUrl(QUrl(item->text(1)));
}
```

Pada kode program di atas terdapat semua fungsi dari aplikasi AmbilRSS, pada lineEdit ditetapkan url http://nice.or.id/activity/feed/ akan muncul ketika aplikasi dijalankan, namun tidak akan membaca RSS secara otomatis.

Yang terakhir adalah memanggil beberapa fungsi di file main.cpp

```
#include <QtGui/QApplication>
#include "rsslisting.h"
int main(int argc, char **argv)
{
    QApplication app(argc, argv);
    qWarning("The usage of QHttp is not recommended anymore,
please use QNetworkAccessManager.");
    RSSListing *rsslisting = new RSSListing;
#if defined(Q_OS_SYMBIAN)
    rsslisting->showMaximized();
#else
    rsslisting->show();
#endif
    return app.exec();
}
```

Pada kode program di atas terdapat qWarning jika menggunakan kelas QHttps. Kelas QHttp menyediakan antarmuka langsung ke HTTP yang memungkinkan Anda untuk men-download dan upload data dengan protokol HTTP. Namun, untuk aplikasi baru, disarankan untuk menggunakan QNetworkAccessManager dan QNetworkReply, API sederhana namun lebih kuat dan implementasi protokol yang lebih modern.

Jalankan aplikasi, maka akan tampil seperti berikut.


Sebetulnya kode program tersebut adalah example code yang bisa Anda gunakan sebagai bahan pembelajaran dari Nokia Qt SDK. ☺

## Membaca XML dengan DOM

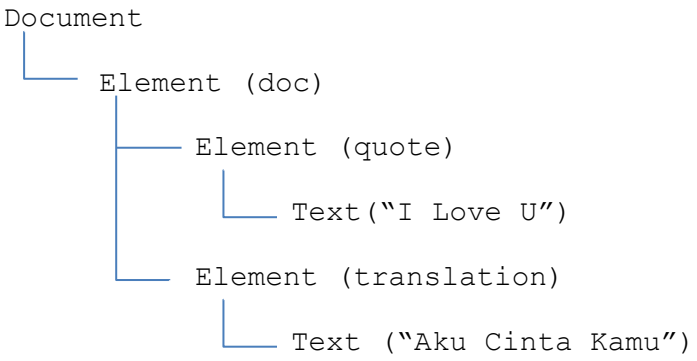
DOM merupakan API standar untuk parsing XML dikembangkan oleh W3C. Qt menyediakan non-validating DOM Level 2 implementasi untuk membaca, memanipulasi, dan menulis dokumen XML.

DOM merupakan file XML sebagai tree dalam memori. Kita dapat menavigasi melalui tree DOM sebanyak yang kita inginkan, dan kita dapat memodifikasi pohon dan simpan kembali ke disk sebagai file XML.

Mari kita perhatikan dokumen XML berikut:

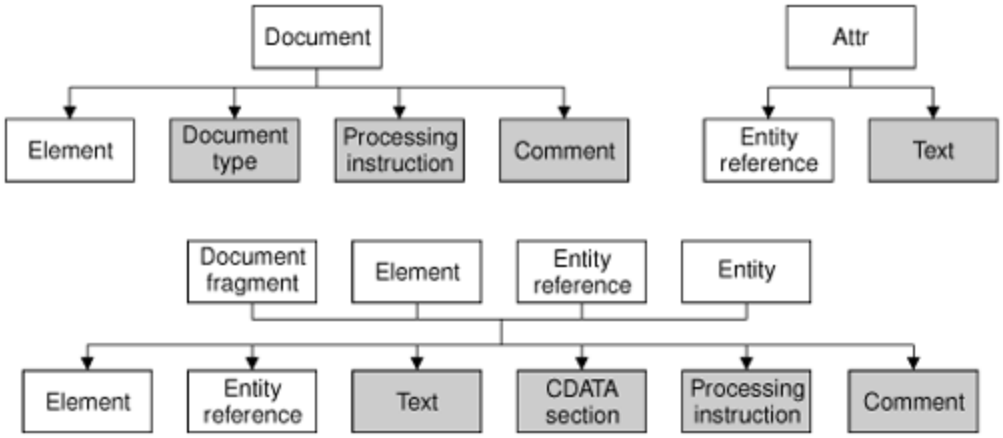
```
<doc>
    <quote>I Love U</quote>
    <translation>I Cinta Kamu</translation>
</doc>
```

Hal ini sesuai dengan tree DOM sebagai berikut:




Tree DOM berisi node dari berbagai jenis. Sebagai contoh, sebuah node `Element` sesuai dengan tag pembuka dan tag penutup yang sesuai. Materi yang jatuh di antara tag muncul sebagai node anak dari simpul `Element`. Dalam Qt, jenis node (seperti semua kelas DOM-terkait lainnya) memiliki awalan `QDom`, dengan demikian, `QDomElement` merupakan node `Element`, dan `QDomText` merupakan node Teks.

Berbagai jenis node dapat memiliki berbagai jenis node anak. Sebagai contoh, sebuah node `Element` dapat berisi node lain `Element`, serta node `EntityReference`, `Teks`, `CDATASection`, `ProcessingInstruction`, dan `Comment`. Gambar di bawah menunjukkan yang mana node dapat memiliki jenis node anak. Simpul ditampilkan dalam warna abu-abu tidak bisa memiliki node anak pun mereka sendiri.




Untuk menggambarkan cara menggunakan DOM untuk membaca file XML, kita akan menulis parser.

```
class DomParser
{
public:
    DomParser(QTreeWidget *tree);
    bool readFile(const QString &fileName);
private:
    void parseBookindexElement(const QDomElement &element);
    void parseEntryElement(const QDomElement &element,
                           QTreeWidgetItem *parent);
    void parsePageElement(const QDomElement &element,
                          QTreeWidgetItem *parent);
    QTreeWidget *treeWidget;
};
```

Kita mendefinisikan sebuah kelas disebut `DomParser` yang akan mengurai indeks buku XML file dan menampilkan hasilnya dalam `QTreeWidget`.

```
DomParser::DomParser(QTreeWidget *tree)
{
    treeWidget = tree;
}
```

Dalam constructor, kita hanya menetapkan widget tree diberikan kepada anggota variabel. Semua parsing dilakukan dalam fungsi `readFile()`.

```
bool DomParser::readFile(const QString &fileName)
{
    QFile file(fileName);
    if (!file.open(QFile::ReadOnly | QFile::Text)) {
        std::cerr << "Error: Cannot read file " <<
qPrintable(fileName)
                  << ": " << qPrintable(file.errorString())
                  << std::endl;
        return false;
    }
    QString errorStr;
    int errorLine;
    int errorColumn;
    QDomDocument doc;
    if (!doc.setContent(&file, false, &errorStr, &errorLine,
                        &errorColumn)) {
        std::cerr << "Error: Parse error at line " << errorLine <<
", "
                  << "column " << errorColumn << ": "
                  << qPrintable(errorStr) << std::endl;
        return false;
    }
    QDomElement root = doc.documentElement();
    if (root.tagName() != "bookindex") {
        std::cerr << "Error: Not a bookindex file" << std::endl;
        return false;
    }
    parseBookindexElement(root);
    return true;
}
```

Dalam `readFile()`, kita mulai dengan mencoba membuka file yang namanya disahkan masuk Jika kesalahan terjadi, kami output pesan kesalahan dan kembali false untuk menandakan kegagalan. Jika tidak, kita mendirikan beberapa variabel untuk menyimpan informasi kesalahan parse, mereka harus diperlukan, dan kemudian membuat `QDomDocument`. Ketika kita panggil `setContent()` pada dokumen DOM, dokumen XML seluruh disediakan oleh `QIODevice` itu dibaca dan diuraikan. Fungsi `setContent()` secara otomatis membuka perangkat jika tidak sudah terbuka. Argumen `false` untuk `setContent()` Menonaktifkan pengolahan namespace, lihat dokumentasi referensi QtXml untuk pengenalan XML namespaces dan bagaimana menangani mereka dalam Qt.

Jika kesalahan terjadi, Output pesan kesalahan dan kembali `false` untuk mengindikasikan kegagalan. Jika parse ini berhasil, kita sebut `documentElement()` pada QDomDocument untuk memperoleh anak tunggal QDomElement, dan kita memeriksa bahwa itu adalah unsur `<bookindex>`. Jika kita memiliki sebuah `<bookindex>`, kita panggil `parseBookindexElement()` untuk mengurai itu. Seperti pada bagian sebelumnya, parsing dilakukan menggunakan keturunan rekursif.

```
void DomParser::parseBookindexElement(const QDomElement
&element)
{
    QDomNode child = element.firstChild();
    while (!child.isNull()) {
        if (child.toElement().tagName() == "entry")
            parseEntryElement(child.toElement(),
                              treeWidget->invisibleRootItem());
        child = child.nextSibling();
    }
}
```

Dalam `parseBookindexElement()`, kita iterate atas semua node anak. Kita berharap setiap node untuk menjadi elemen `<entry>`, dan untuk setiap satu yang kita sebut `parseEntry()` untuk mengurai itu. Kita mengabaikan node yang tidak diketahui, untuk memungkinkan format indeks buku untuk diperpanjang di masa depan tanpa mencegah parser lama dari bekerja. Semua node `<entry>` yang adalah anak-anak langsung dari node `<bookindex>` adalah top-level node dalam widget tree kita mengisi untuk mencerminkan tree DOM, sehingga ketika kita ingin mengurai masing-masing kita melewati kedua elemen node dan pohon item akar tak terlihat menjadi parent item tree widget.

Kelas `QDomNode` dapat menyimpan semua jenis node. Jika kita ingin mengolah node lebih lanjut, pertama kita harus mengkonversinya ke jenis data yang benar. Dalam contoh ini, kita hanya peduli node Elemen, jadi kita `sebut toElement()` pada `QDomNode` untuk mengubahnya menjadi `QDomElement` dan kemudian memanggil `Tagname()` untuk mengambil nama tag `element`. Jika node tidak dari Elemen mengetik, `toElement()` fungsi mengembalikan sebuah objek `QDomElement` null, dengan nama tag kosong.

```
void DomParser::parseEntryElement(const QDomElement &element,
                                  QTreeWidgetItem *parent)
{
    QTreeWidgetItem *item = new QTreeWidgetItem(parent);
    item->setText(0, element.attribute("term"));
    QDomNode child = element.firstChild();
    while (!child.isNull()) {
        if (child.toElement().tagName() == "entry") {
            parseEntryElement(child.toElement(), item);
        } else if (child.toElement().tagName() == "page") {
            parsePageElement(child.toElement(), item);
        }
        child = child.nextSibling();
    }
}
```

Dalam `parseEntryElement()`, kita membuat sebuah item widget tree. Item parent yang disahkan pada adalah salah item tak terlihat tree root (jika ini adalah entri top-level) atau entri lain (jika ini adalah sub-entry). Kita Panggil `setText()` untuk mengatur teks yang ditampilkan pada kolom pertama item dengan nilai atribut istilah tag `<entry>` itu.

Setelah kita memiliki diinisialisasi `QTreeWidgetItem`, kita iterate selama node anak dari simpul `QDomElement` sesuai dengan tag `<entry>` saat ini. Untuk setiap elemen anak yang merupakan tag `<entry>`, kita sebut `parseEntryElement()` secara rekursif dengan item saat ini sebagai argumen kedua. `QTreeWidgetItem` Setiap anak maka akan dibuat dengan entry ini sebagai induknya. Jika elemen anak `<page>`, kita panggil `parsePageElement()`.

```
void DomParser::parsePageElement(const QDomElement &element,
                                 QTreeWidgetItem *parent)
{
    QString page = element.text();
    QString allPages = parent->text(1);
    if (!allPages.isEmpty())
        allPages += ", ";
    allPages += page;
    parent->setText(1, allPages);
}
```

Dalam `parsePageElement()`, kita panggil `teks()` pada elemen untuk mendapatkan teks yang terjadi antara `tag <page>` dan `</page>`, kemudian kita menambahkan teks ke daftar dipisahkan koma-nomor halaman pada kolom kedua `QTreeWidgetItem` . Fungsi `QDomElement::teks()` menavigasi melalui node anak elemen dan merangkai semua teks yang tersimpan dalam teks dan node CDATA.

Mari kita sekarang melihat bagaimana kita dapat menggunakan class `DomParser` untuk mengurai file:

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QStringList args = QApplication::arguments();
    ...
    QTreeWidget treeWidget;
    ...
    DomParser parser(&treeWidget);
    for (int i = 1; i < args.count(); ++i)
        parser.readFile(args[i]);
    return app.exec();
}
```

Kita mulai dengan mendirikan sebuah `QTreeWidget`. Lalu kita membuat `DomParser`. Untuk setiap file yang terdaftar pada baris perintah, kita sebut `DomParser::readFile()` untuk membuka dan mengurai file masing-masing dan mengisi widget pohon.

Seperti contoh sebelumnya, kita perlu baris berikut dalam file pro pada aplikasi untuk menghubungkan terhadap library QtXml.:

```
QT += xml
```

Sebagai contoh menggambarkan, navigasi melalui tree DOM sangat mudah, meskipun tidak cukup nyaman karena menggunakan `QXmlStreamReader`. Programmer yang menggunakan banyak DOM sering menulis fungsi wrapper untuk menyederhanakan operasi biasanya diperlukan.

## Membaca XML dengan SAX

SAX adalah domain publik de facto API standar untuk membaca dokumen XML. Kelas Qt SAX dimodelkan setelah SAX2 Java, dengan beberapa perbedaan dalam penamaan agar sesuai dengan konvensi Qt. Dibandingkan dengan DOM, SAX lebih rendah dan biasanya lebih cepat. Tapi karena kelas `QXmlStreamReader` disajikan sebelumnya dalam bab ini menawarkan API Qt lebih cepat daripada SAX parser, penggunaan utama dari SAX parser adalah untuk porting kode yang menggunakan API SAX ke Qt. Untuk informasi lebih lanjut mengenai SAX, lihat http://www.saxproject.org/.

Qt menyediakan parser non-validating SAX berbasis XML yang disebut `QXmlSimpleReader`. Parser ini mengakui well-formed XML dan mendukung XML namespaces. Ketika parser berjalan melalui dokumen, itu memanggil fungsi virtual dalam kelas handler untuk menunjukkan parsing event. (Ini "event parsing" tidak ada hubungannya dengan event Qt, seperti event tombol dan mouse.) Sebagai contoh, mari kita asumsikan parser sedang menganalisa dokumen XML berikut:

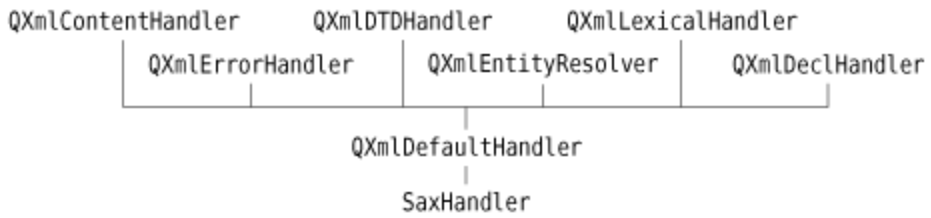
```
<doc>
    <quote>Gnothi seauton</quote>
</doc>
```

Program pendeteksi kombinasi tombol akan memanggil handler parsing acara berikut:

```
startDocument()
startElement("doc")
startElement("quote")
characters("Gnothi seauton")
endElement("quote")
endElement("doc")
endDocument()
```

Fungsi sebelumnya semua dideklarasikan dalam `QXmlContentHandler`. Untuk mempermudah, kita menghilangkan beberapa argumen untuk `startElement()` dan `endElement()`.

`QXmlContentHandler` hanyalah salah satu dari kelas handler banyak yang dapat digunakan bersama dengan `QXmlSimpleReader`. Yang lain `QXmlEntityResolver`, `QXmlDTDHandler`, `QXmlErrorHandler`, `QXmlDeclHandler`, dan `QXmlLexicalHandler`. Kelas-kelas ini hanya menyatakan fungsi virtual murni dan memberikan informasi tentang berbagai jenis peristiwa parsing. Untuk sebagian besar aplikasi, `QXmlContentHandler` dan `QXmlErrorHandler` adalah hanya dua yang dibutuhkan.




Untuk kenyamanan, Qt juga menyediakan `QXmlDefaultHandler`, sebuah kelas yang berasal dari semua kelas handler dan yang menyediakan implementasi trivial untuk semua fungsi. Ini desain, dengan banyak kelas penangan abstrak dan satu subclass trivial, tidak lazim untuk Qt, melainkan diadopsi untuk selalu mengikuti penerapan model Java.

Perbedaan yang paling signifikan antara menggunakan API SAX dan QXmlStreamReader atau API DOM adalah bahwa API SAX mengharuskan kita untuk secara manual melacak tempat parser menggunakan anggota variabel, sesuatu yang tidak diperlukan dalam dua pendekatan lain, yang keduanya memungkinkan keturunan rekursif .

Untuk menggambarkan cara menggunakan SAX untuk membaca file XML. Di sini kita akan mengurai menggunakan `QXmlSimpleReader` dan subclass `QXmlDefaultHandler` disebut SaxHandler.

Langkah pertama untuk melaksanakan parser adalah subclass `QXmlDefaultHandler`:

```
class SaxHandler : public QXmlDefaultHandler
{
public:
    SaxHandler(QTreeWidget *tree);
    bool readFile(const QString &fileName);
protected:
    bool startElement(const QString &namespaceURI,
                      const QString &localName,
                      const QString &qName,
                      const QXmlAttributes &attributes);
    bool endElement(const QString &namespaceURI,
                    const QString &localName,
                    const QString &qName);
    bool characters(const QString &str);
    bool fatalError(const QXmlParseException &exception);
private:
    QTreeWidget *treeWidget;
    QTreeWidgetItem *currentItem;
    QString currentText;
};
```

Kelas SaxHandler berasal dari QXmlDefaultHandler dan reimplements empat fungsi: `startElement()`, `endElement()`,`characters()`, dan `fatalError()`. Tiga fungsi pertama dinyatakan dalam QXmlContentHandler, fungsi terakhir ini dideklarasikan pada QXmlErrorHandler.

```
SaxHandler::SaxHandler(QTreeWidget *tree)
{
    treeWidget = tree;
}
```

Konstruktor SaxHandler menerima `QTreeWidget` kita ingin mengisi dengan informasi yang tersimpan dalam file XML.

```
bool SaxHandler::readFile(const QString &fileName)
{
    currentItem = 0;
    QFile file(fileName);
    QXmlInputSource inputSource(&file);
    QXmlSimpleReader reader;
    reader.setContentHandler(this);
    reader.setErrorHandler(this);
    return reader.parse(inputSource);
}
```

Fungsi ini dipanggil saat kita memiliki nama file yang akan dipecah. Kami membuat objek `QFile` untuk file dan membuat `QXmlInputSource` untuk membaca isi file. Lalu kita membuat `QXmlSimpleReader` mengurai file. Kami mengatur konten pembaca dan penjamah kesalahan untuk kelas ini (`SaxHandler`), dan kemudian kita panggil `parse()` pada reader untuk melakukan parsing tersebut. Dalam SaxHandler, kami hanya reimplement fungsi dari `QXmlContentHandler` dan kelas `QXmlErrorHandler`, jika kita telah melaksanakan fungsi dari kelas handler lainnya, ini juga akan diperlukan untuk memanggi fungsil `setXxxHandler`().

Alih-alih melewati objek QFile sederhana ke fungsi `parse()`, kita melewati `QXmlInputSource`. Kelas ini membuka file yang diberikan, membacanya (dengan mempertimbangkan setiap pengkodean karakter yang ditentukan dalam deklarasi `<? xml?>`), dan menyediakan sebuah antarmuka dimana parser membaca file.

```
bool SaxHandler::startElement(const QString & /* namespaceURI */,
                              const QString & /* localName */,
                              const QString &qName,
                              const QXmlAttributes &attributes)
{
    if (qName == "entry") {
        currentItem = new QTreeWidgetItem(currentItem ?
                currentItem : treeWidget->invisibleRootItem());
        currentItem->setText(0, attributes.value("term"));
    } else if (qName == "page") {
        currentText.clear();
    }
    return true;
}
```

Fungsi `startElement()` ini dipanggil saat reader bertemu dengan tag pembuka baru. Parameter ketiga adalah nama tag (atau lebih tepatnya, "nama yang memenuhi syarat"). Parameter keempat adalah daftar atribut. Dalam contoh ini, kita mengabaikan parameter pertama dan kedua. Mereka berguna untuk file XML yang menggunakan mekanisme namespace XML, sebuah topik yang dibahas secara rinci dalam dokumentasi referensi.

Jika tag `<entry>`, kita membuat `QTreeWidgetItem` baru. Jika tag bersarang dalam tag lain `<entry>`, tag baru mendefinisikan sebuah sub-entri dalam indeks, dan `QTreeWidgetItem` baru diciptakan sebagai anak dari `QTreeWidgetItem` yang mewakili entri menyeluruh. Jika tidak, kami menciptakan `QTreeWidgetItem` sebagai bagian tingkat atas, menggunakan item akar pohon tak terlihat widget sebagai induknya. Kita panggil `setText()` untuk mengatur teks yang ditampilkan pada kolom 0 dengan nilai atribut istilah tag `<entry>` itu.

Jika tag `<page>`, kita menetapkan variabel currentText menjadi string kosong. Variabel berfungsi sebagai akumulator untuk teks terletak antara tag `<page>` dan `</page>`.

Pada akhirnya, kita kembali `true` untuk memberitahu SAX untuk melanjutkan parsing file. Jika kita ingin melaporkan tag tidak dikenal sebagai kesalahan, kita akan kembali `false` dalam kasus-kasus. Kita kemudian juga reimplement `errorString()` dari `QXmlDefaultHandler` untuk mengembalikan pesan kesalahan yang sesuai.

```
bool SaxHandler::characters(const QString &str)
{
    currentText += str;
    return true;
}
```

Fungsi `characters()` dipanggil untuk melaporkan data karakter dalam dokumen XML. Kita hanya menambahkan karakter pada variabel `currentText`.

```
bool SaxHandler::endElement(const QString & /* namespaceURI */,
                            const QString & /* localName */,
                            const QString &qName)
{
    if (qName == "entry") {
        currentItem = currentItem->parent();
    } else if (qName == "page") {
        if (currentItem) {
            QString allPages = currentItem->text(1);
            if (!allPages.isEmpty())
                allPages += ", ";
            allPages += currentText;
            currentItem->setText(1, allPages);
        }
    }
    return true;
}
```

Fungsi `endElement()` ini dipanggil saat reader bertemu dengan sebuah tag penutup. Sama seperti dengan `startElement()`, parameter ketiga adalah nama tag.

Jika tag `</entry>`, kita update variabel `currentItem` untuk menunjuk ke parent `QTreeWidgetItem`. (Untuk alasan hostory, top-level item kembali 0 sebagai parent mereka daripada item root invisible.) Hal ini menjamin bahwa variabel `currentItem` dikembalikan ke nilai yang diadakan sebelum tag `<entry>` terkait dibacakan.

Jika tag `</page>`, kita menambahkan nomor halaman yang ditentukan atau jangkauan halaman ke daftar dipisahkan koma-item dalam teks berjalan di kolom 1.

```
bool SaxHandler::fatalError(const QXmlParseException &exception)
{
    std::cerr << "Parse error at line " <<
exception.lineNumber()
              << ", " << "column " << exception.columnNumber()
<< ": "
              << qPrintable(exception.message()) << std::endl;
    return false;
}
```

Fungsi `fatalError()` ini dipanggil saat pembaca gagal untuk mengurai file XML. Jika hal ini terjadi, kita hanya mencetak pesan ke konsol, memberikan nomor baris, jumlah kolom, dan teks kesalahan parser.

Ini melengkapi pelaksanaan `SaxHandler`. Fungsi `main()` yang menggunakan hampir identik dengan yang kita dibahas dalam bagian sebelumnya untuk DomParser, perbedaan adalah bahwa kita menggunakan SaxHandler bukan DomParser.

## Menulis XML

Kebanyakan aplikasi yang dapat membaca file XML juga perlu menulis file tersebut. Ada tiga pendekatan untuk menghasilkan file XML dari aplikasi Qt:

- Kita dapat menggunakan sebuah `QXmlStreamWriter`.
- Kita bisa membangun tree DOM dan panggilan `save()` di atasnya.
- Kita bisa menghasilkan XML dengan manual.

Pilihan antara pendekatan ini sebagian besar tergantung pada apakah kita menggunakan `QXmlStreamReader`, `DOM`, atau `SAX` untuk membaca dokumen XML, walaupun jika data yang diadakan di tree DOM sering masuk akal untuk menyelamatkan tree secara langsung.

Menulis XML menggunakan kelas `QXmlStreamWriter` sangat mudah. Jika kita ingin output data indeks buku dari `QTreeWidget` menggunakan `QXmlStreamWriter`, kita bisa melakukannya dengan hanya menggunakan dua fungsi. Fungsi pertama akan mengambil nama file dan `QTreeWidget*`, dan akan iterate atas semua top-level item dalam tree:

```
bool writeXml(const QString &fileName, QTreeWidget *treeWidget)
{
    QFile file(fileName);
    if (!file.open(QFile::WriteOnly | QFile::Text)) {
        std::cerr << "Error: Cannot write file "
                  << qPrintable(fileName) << ": "
                  << qPrintable(file.errorString()) << std::endl;
        return false;
    }
    QXmlStreamWriter xmlWriter(&file);
    xmlWriter.setAutoFormatting(true);
    xmlWriter.writeStartDocument();
    xmlWriter.writeStartElement("bookindex");
    for (int i = 0; i < treeWidget->topLevelItemCount(); ++i)
        writeIndexEntry(&xmlWriter, treeWidget->topLevelItem(i));
    xmlWriter.writeEndDocument();
    file.close();
    if (file.error()) {
        std::cerr << "Error: Cannot write file "
                  << qPrintable(fileName) << ": "
                  << qPrintable(file.errorString()) << std::endl;
        return false;
    }
    return true;
}
```

Jika kita mengaktifkan auto-format, XML adalah output dalam gaya manusia yang lebih ramah, dengan lekukan digunakan untuk menunjukkan struktur rekursif data itu. Fungsi `writeStartDocument()` menulis baris header XML

```
<?xml version="1.0" encoding="UTF-8"?>
```

Fungsi `WriteStartElement()` menghasilkan tag awal yang baru dengan teks tag yang diberikan. Fungsi `writeEndDocument()` menutup tag terbuka. Untuk setiap top-level, kita panggil `writeIndexEntry()`, `QXmlStreamWriter`, dan item ke output. Berikut adalah kode untuk `writeIndexEntry ()`:

```
void writeIndexEntry(QXmlStreamWriter *xmlWriter, QTreeWidgetItem
*item)
{
    xmlWriter->writeStartElement("entry");
    xmlWriter->writeAttribute("term", item->text(0));
    QString pageString = item->text(1);
    if (!pageString.isEmpty()) {
        QStringList pages = pageString.split(", ");
        foreach (QString page, pages)
            xmlWriter->writeTextElement("page", page);
    }
    for (int i = 0; i < item->childCount(); ++i)
        writeIndexEntry(xmlWriter, item->child(i));
    xmlWriter->writeEndElement();
}
```

Fungsi menciptakan elemen `<entry>` sesuai dengan `QTreeWidgetItem` yang diterimanya sebagai parameter. The `writeAttribute()` fungsi menambahkan atribut tag yang baru saja ditulis, misalnya, mungkin berubah `<entry>` menjadi `<entry term="sidebearings">`. Jika ada nomor halaman, mereka terbagi pada koma-spasi, dan untuk setiap halaman `<page>...</page>` tag ditulis terpisah dengan teks halaman di antara keduanya. Ini semua dicapai dengan memanggil `writeTextElement()` dan passing nama tag dan teks untuk menempatkan antara awal dan akhir tag. Dalam semua kasus, `QXmlStreamWriter` mengurus sendiri karakter khusus XML, jadi kita tidak perlu khawatir tentang hal ini.

Jika item telah memiliki item anak, kita secara rekursif memanggil `writeIndexEntry()` pada masing-masing. Akhirnya, kita panggil `writeEndElement()` untuk output `</entry>`.

Menggunakan `QXmlStreamWriter` untuk menulis XML adalah pendekatan yang paling mudah dan aman, tetapi jika kita sudah memiliki XML di tree DOM, kita hanya bisa meminta tree untuk output XML yang relevan dengan memanggil `save()` pada objek `QDomDocument`. Secara default, `save()` menggunakan `UTF-8` sebagai pengkodean untuk file yang dihasilkan. Kita dapat menggunakan pengkodean lain dengan mengawali suatu deklarasi `<xml??>` seperti

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Ke tree DOM. Potongan kode berikut ini menunjukkan cara melakukan ini:

```
const int Indent = 4;
QDomDocument doc;
...
QTextStream out(&file);
QDomNode xmlNode = doc.createProcessingInstruction("xml",
                            "version=\"1.0\" encoding=\"ISO-
8859-1\"");
doc.insertBefore(xmlNode, doc.firstChild());
doc.save(out, Indent);
```

Dimulai dengan Qt 4.3, alternatif adalah untuk menetapkan pengkodean pada `QTextStream` menggunakan `setCodec()` dan untuk `QDomNode:: EncodingFromTextStream` sebagai parameter ketiga untuk `save()`.

Pembangkit XML file dengan manual tidak jauh lebih sulit daripada menggunakan DOM. Dapat menggunakan `QTextStream` dan menulis string seperti yang kita akan melakukan dengan file teks lainnya. Bagian paling sulit adalah untuk melarikan diri karakter khusus dalam nilai-nilai teks dan atribut. Fungsi `Qt:: escape()` karakter `'<'`, `'>'`, dan `'&'`. Berikut beberapa kode yang menggunakan ini:

```
QTextStream out(&file);
out.setCodec("UTF-8");
out << "<doc>\n"
    << "    <quote>" << Qt::escape(quoteText) << "</quote>\n"
    << "    <translation>" << Qt::escape(translationText)
    << "</translation>\n"
    << "</doc>\n";
```

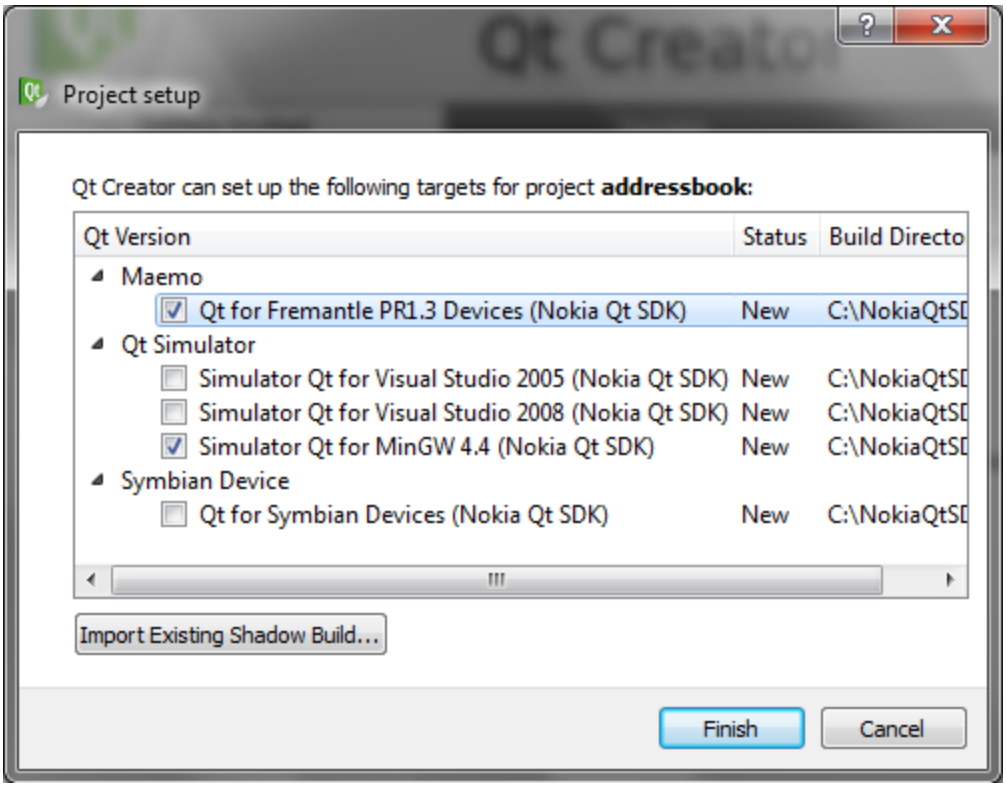
Saat membuat file XML seperti ini, selain harus menulis yang benar deklarasi dan `<xml??>` pengaturan penyandian yang tepat, kita juga harus ingat untuk meninggalkan dari teks kita menulis, dan jika kita menggunakan atribut kita harus meninggalkan tanda kutip tunggal atau nilai ganda. Menggunakan `QXmlStreamWriter` jauh lebih mudah karena menangani semua ini untuk kita.

# Menggunakan Template Aplikasi Qt

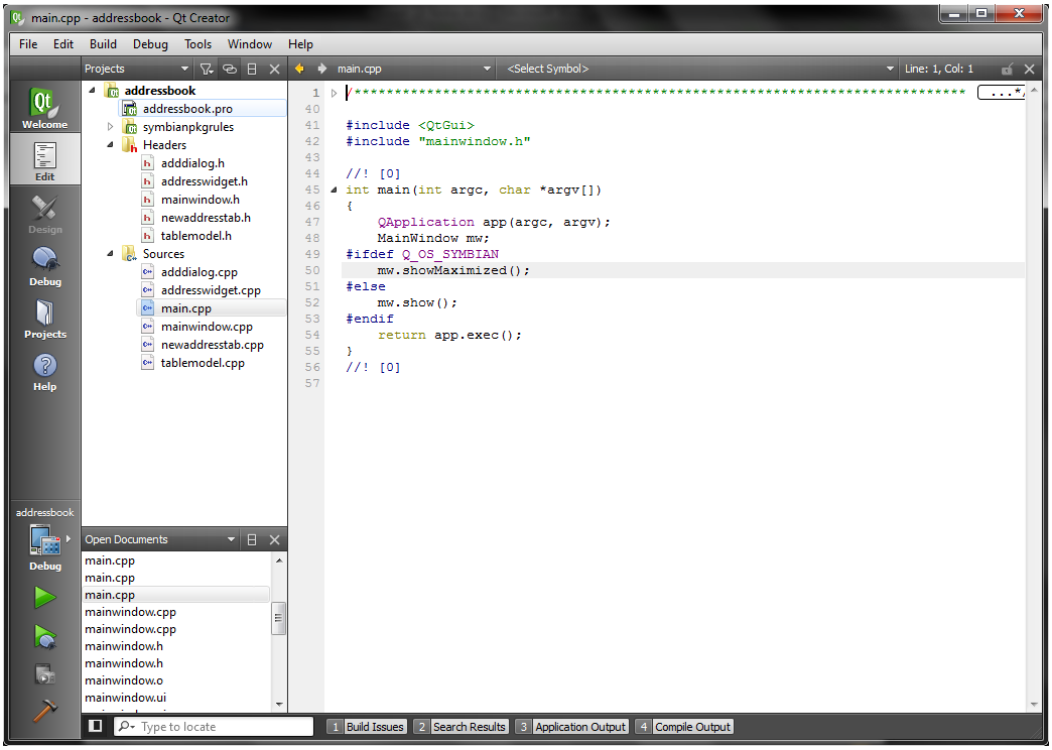
Qt Creator menyediakan banyak template aplikasi yang bisa Anda manfatkan sebagai bahan pembelajaran. Anda bisa ambil beberapa baris kode program dan digabunkan untuk dijadikan suatu aplikasi mobile yang powerful. ☺

Template ini berada pada halaman depan Qt ketika kita membuka aplikasi Qt Creator.

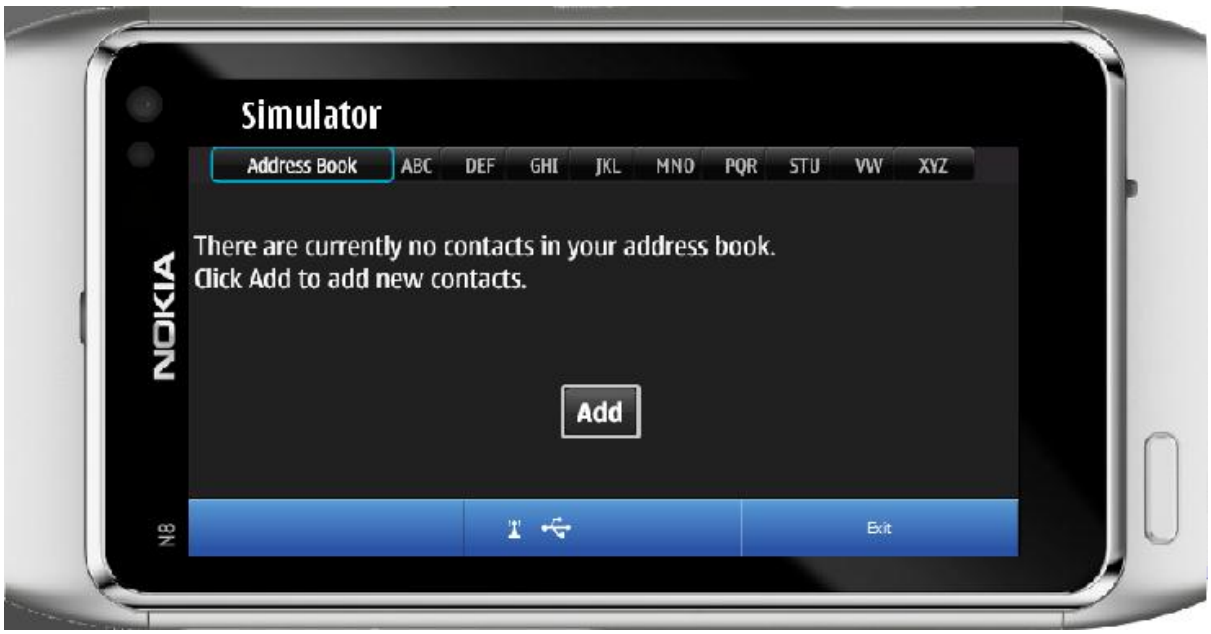

Terdapat beberapa kategori template dengan full kode. Pilih salah satu template yang tersedia, kemudian setelah memilih, Anda diminta untuk menentukan simulator yang ingin Anda gunakan.




Selanjutnya pada modus edit akan tampil beberapa source code yang bisa Anda gunakan.




Mari kita coba jalankan aplikasi dari template ini.




Cukup menarik bukan. ☺

# Sekilas Tentang Qt Quick

Qt Quick singkatan dari Qt User Interface Creation Kit adalah teknologi antarmuka tingkat atas (high-level user interface technology) yang diklaim secara dramatis memudahkan para pengembang untuk mendisain antarmuka (UI) yang cantik, 'pixel perfect' dan ringan disamping 'touch-enabled' menggunakan basis pemrograman Qt, tanpa perlu pengetahuan tentang pemrograman C++. Qt Quick diperkenalkan bersama rilis Qt 4.7 bulan Maret lalu dan selanjutnya menjadi bagian dari rilis Qt berikutnya.

Yang penulis amati, Qt Quick tidak tersedia pada Qt Creator versi 2.0, hanya tersedia pada versi 2.1 ke atas, Anda bisa download Qt Creator 2.1 di halaman download qt.nokia.com sebesar 49MB.


Bagi para pengembang, Qt Quick menawarkan kelebihan-kelebihan antara lain:

- Deklaratif yang mudah digunakan dengan bahasa mirip dengan skrip QML (Qt Meta-Object Language);

- Tools didalam Qt Creator IDE termasuk sebuah visual editor yang membantu para pengembang dan disainer menggabungkan elemen-elemen sederhana menjadi sebuah antarmuka animasi;
- Kelebihan lainnya adalah pendekatan pemrograman deklaratif yang membantu pembuatan antarmuka-antarmuka menjadi gampang, hanya dengan memerintahkan apa yang harus dibuat dan tidak harus menuliskan bagaimana cara membuatnya.

Sementara Qt saat ini masih melayani pengembang C++, Qt Quick telah memperluas jangkauan pengguna Qt dengan kemampuan antarmuka. Hal ini membuka peluang emas untuk para disainer antarmuka termasuk pengembang yang terbiasa dengan bahasa skrip untuk memanfaatkan Qt dalam membuat antarmuka yang keren dan aplikasi-aplikasi untuk perangkat atau komputer yang mendukung platform Qt seperti halnya device Nokia.

Pada ebook ini saya belum sempat untuk membahas secara teknis tentang Qt Quick ini, namun Insya Allah jika ada kesempatan untuk menulis lagi, saya akan coba membahas tentang Qt Quick. ☺

# Penutup

Demikian ebook yang sederhana ini saya buat. Isinya mungkin masih jauh dari apa yang namanya bagus apalagi sempurna, namun mudah-mudahan apa yang saya tulis ini bisa bermanfaat, terutama bagi teman-teman yang belum mengenal teknologi mobile Qt juga untuk memberikan wawasan tentang Pemrograman. Apabila ada pertanyaan seputar teknologi mobile lainnya, Anda dapat bertanya melalui:

- Email: ciebal745@gmail.com
- Facebook : http://www.facebook.com/ciebal
- Twitter : http://twitter.com/ciebal

Untuk update mengenai Teknologi Mobile Qt, silahkan kunjungi situs:

- Nokia Indonesia Community Enthusiast (NICE)
  http://nice.or.id
- Blog Penulis:
  http://www.ciebal.web.id
- Blog Penulis @Nice
  http://nice.or.id/ciebal

# Referensi


- http://doc.qt.nokia.com
- http://forum.nokia.com
- Buku Saku Pemrograman Mobile dengan Qt (Agus Kurniawan)
- Pengantar Pemrograman Qt dan KDE (Ariya Hidayat, ilmukomputer.com)
- C++ GUI Programming with Qt 4, Second Edition