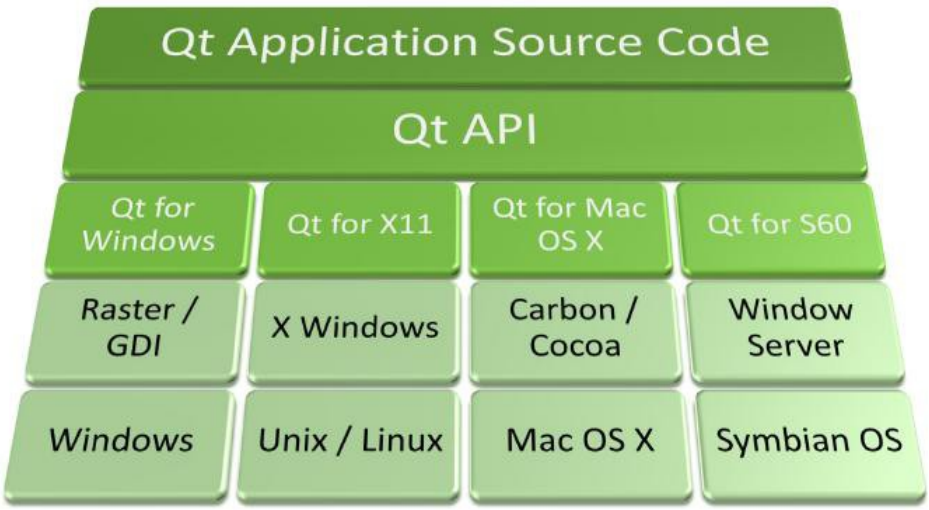
Qt adalah framework pengembangan aplikasi *cross-platform* yang komperhensif dengan bahasa C++ dan menawarkan solusi pemrograman dengan konsep pemrograman berorientasi objek (OOP, *object oriented programming)* yang kuat: di mana masalah pemrograman diselesaikan dalam instans objek dengan atribut, fungsi, dan interaksi antar objek.

Qt memilki *tools* yang digunakan untuk menyederhanakan proses pengembangan program dan antar muka grafis pada desktop, embedded system dan perangkat *mobile*, antara lain:

- **Qt Framework** – seperangkat API intuitif dalam bahas C++ dan Qt Quick dengan sintaks yang mirip dengan JavaScript/CSS untuk *prototyping* UI,
- **Qt Creator IDE** – lingkungan pengembangan terpadu (Integrated Development Environment) yang *cross-platform,*
- **Tools and toolchains** – *cross compiler toolchain* dan perangkat lokalisasi bahasa.

Qt adalah mengimplementasikan API (application programming interface) secara mandiri di atas API bawaan sistem operasi, sehingga performa dan kompabilitas pada masing-masing platform bekerja secara optimal. Sementara fungsi-fungsi yang tidak tergantung terhadap sistem operasi (misal: pengolahan string atau operasi bilangan) menggunakan class internal Qt yang diimplementasikan se-universal mungkin antar platform (sistem operasi dan arsitektur prosesor).



Gambar 1. Arsitektur Qt

Untuk interaksi antar objek, selain menggunakan pemanggilan metode atribut public/protected secara sinkron, Qt juga memiliki mekanisme SIGNAL dan SLOT untuk interaksi antar objek secara asinkron. Mekanisme ini ditangani moc (*meta object compiler*) yang menerjemahkan kode yang kita tulis menjadi kode yang optimal dalam bahasa C++ dan sesuai dengan pustaka Qt. moc (dan class QMetaObject) juga menangani stuktur inheritans, penerjemahan bahasa (jika aplikasi dirancang multi bahasa), dan hal-hal lain.

Qt juga memiliki *language binding* untuk bahasa pemrograman C++ yang dikenal kaku dan kompleks, antara lain **PyQt** untuk Python, **Qyoto** untuk C#, **Jambix** untuk Java, dan *language binding* lainnya. Solusi ini menungkinkan pengembangan program dalam berbagai bahasa pemrograman berbeda sehingga memberikan fleksibilitas bagi pemrogram.

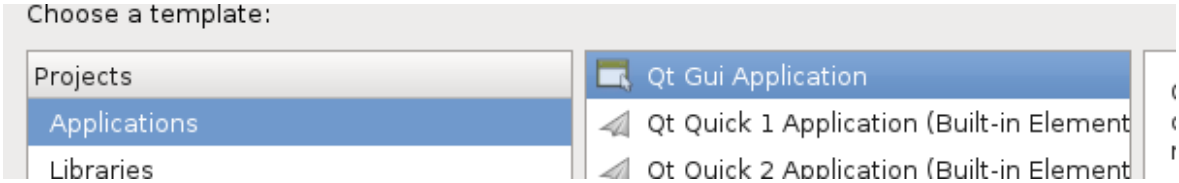
Untuk desain form (tampilan) GUI, Qt dapat menggunakan designer UI yang menggunakan layout XML atau QML (Qt Markup Language) yang mirip dengan JavaScript di samping dengan *class-class widget* yang lebih konsisten namun kurang praktis. Hal ini menungkinkan desain GUI yang lebih produktif, interaktif dan menarik di saat tampilan menjadi nilai tambah aplikasi, serta konsistensi dan efesiensi tampilan di saat lain.

Qt diimplementasikan tidak hanya pada sistem dekstop: Linux, Windows dan MacOS, melainkan termasuk embedded system antara lain: Montavista Linux (mobilinux), Android, WindowsCE, Symbian S60 dan lainnya. Dengan Qt, kita dapat menggunakan ulang kode untuk program secara efesien untuk berbagai platform komputer berbeda dengan hanya satu kode program (dengan pengubahan minor jika diperlukan).

Bagian 1: Hello, Qt!

Hello!

Buka IDE Qt Creator, buat sebuah proyek baru bernama hello dengan tipe Qt Gui Application.



Gambar 2a. Pemilihan Templat Program

Ubah file **hello.pro** sehingga berisi seperti berikut:

```
QT       += gui widgets
TARGET = hello
SOURCES += main.cpp
```

sehingga struktur project akan berubah seperti berikut:

Gambar 2b. Stuktur Source Program **hello**

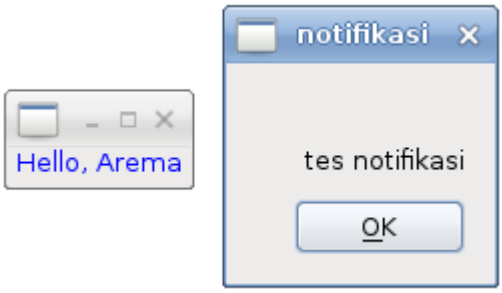
Ubah file main.cpp sehingga berisi seperti:

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QLabel x("<font color=blue>Hello, Arema</font>");
    x.show();

    QMessageBox msgBox;
    msgBox.setWindowTitle("notifikasi");
    msgBox.setInformativeText("tes notifikasi");
    msgBox.setModal(true);
    msgBox.show();

    return a.exec();
}
```

Jalankan program dengan klik pada tombol Run atau hotkey Ctrl+R:

Gambar 2c. Kompilasi dan Eksekusi Program

maka akan muncul aplikasi dengan tampilan sebagai berikut:



Gambar 2d. Tampilan Program **hello**

Pengenalan Struktur Source Program

Dalam sebuah project Qt, terdapat beberapa jenis kode terpisah: secara umum berkas *project* (***.pro**), berkas *header* (***.h**), berkas *source*(***.cpp**), pada program yang lebih kompleks memungkinkan penggunaan berkas konfigurasi pustaka (***.prf**) dengan penambahan konfigurasi *project*.

Berkas *project* dengan berisisi deklarasi mengenai file-file yang tercantum dalam *project*, penyertaan pustaka, konfigurasi program (GUI widget, Qt Quick, atau berbasis terminal), nama build program dan lainnya.

Berkas header (*.h) berisi deklarasi class dan properti yang dimiliki: metode dan atribut (termasuk SiGNAL dan SLOT).

Berkas (*.cpp) berisi implementasi metode yang telah dijabarkan pada berkas header. Berkas *source* berisi berbagai implementasi metode-metode yang sebelumnya dideklarasikan di

berkas *header*. Berkas *source* berisi metode-metode dan operasi, juga event dari SIGNAL. (Mekanisme SIGNAL dan SLOT dijelaskan di Bagian 2)

Berikut adalah contoh berkas project (**efly.pro**):

```
include(/usr/local/qwt/features/qwt.prf)
QT       += core gui
TARGET = lomba
TEMPLATE = app
SOURCES += main.cpp \
    flygambar.cpp \
    flykontrol.cpp \
    flygrafik.cpp
HEADERS += \
    flygambar.h \
    flykontrol.h \
    flygrafik.h
```

Berikut adalah contoh berkas header (**flygambar.h**):

```
#ifndef FLYGAMBAR_H
#define FLYGAMBAR_H
#define IMAGE_WIDTH 200
#define IMAGE_HEIGHT 200
#include <QLabel>
#include <QPixmap>
#include <QPainter>
#include <QImage>
#include <QFileDialog>
class flyGambar : public QLabel
{
    Q_OBJECT
public:
    explicit flyGambar(QWidget *parent = 0);

signals:
    void newPixel();
    void newCurve();
public slots:
    void drawPixel(int x, int y, char grayscale);
    void drawLine(const char *pixels);
    void saveImage();
private:
    QImage *canvas;
    QPainter painter;
    short int baris;
};
#endif // FLYGAMBAR_H
```

Berikut adalah contoh berkas source (**flygambar.cpp**) :

```
#include "flygambar.h"
flyGambar::flyGambar(QWidget *parent) : QLabel(parent){
    canvas =  new QImage(IMAGE_WIDTH,IMAGE_HEIGHT,QImage::Format_RGB32);
    canvas->fill(IMAGE_WIDTH*IMAGE_HEIGHT);
    setPixmap(QPixmap::fromImage(*canvas));
    baris=0;
}
void flyGambar::drawPixel(int x, int y, char grayscale){
    painter.begin(canvas);
    painter.setPen(qRgb(grayscale,grayscale,grayscale));
    painter.drawPoint(x,y);
    painter.end();
    if (x==(IMAGE_WIDTH-1)) update(0,y,IMAGE_WIDTH,1);
    setPixmap(QPixmap::fromImage(*canvas));
}
void flyGambar::drawLine(const char *pixels){
    qDebug("baris terima %d",baris);
    painter.begin(canvas);
    for (int i=0; i<IMAGE_WIDTH; i++){
        painter.setPen(qRgb(pixels[i],pixels[i],pixels[i])) ;
        painter.drawPoint(i,baris);
    }
    painter.end();
    update(0,baris,IMAGE_WIDTH,1);
    setPixmap(QPixmap::fromImage(*canvas));
    baris++;
}
void flyGambar::saveImage(){
    canvas->save(QFileDialog::getSaveFileName(),"PNG",0);
}
```

Bagian 2: Aplikasi Spinbox, Slider, Button

Interaksi antar Objek: SIGNAL dan SLOT

Mekanisme SIGNAL dan SLOT adalah mekanisme komunikasi antar objek di Qt yang memungkinkan pemanggilan fungsi secara asinkron (bebas/acak). Kelebihan mekanisme ini dibanding *function call* biasa adalah type-safe, keterjaminan hubungan antar SIGNAL dan SLOT, kemudahan desain *class* dan interaksi antar objek, dan kemudahan.

Class yang mengakomodasi fitur objek Qt (QObject atau QMetaObject) harus dideklarasikan dengan makro `Q_OBJECT` dan fungsi SIGNAL/SLOT yang akan digunakan harus dideklarasikan di berkas *header*.

```
…
class flyGambar : public QLabel
{
    Q_OBJECT
public:
    explicit flyGambar(QWidget *parent = 0);

signals:
    void newPixel();
    void newCurve();
public slots:
    void drawPixel(int x, int y, char grayscale);
    void drawLine(const char *pixels);
    void saveImage();
…
```

SIGNAL selalu bernilai kembalian void dan diimplementasikan pada berkas *source* dengan makro `emit`.

```
…
emit newCurve();
emit newPixel();
…
```

SLOT diimplementasikan seperti metode biasa dalam berkas source. SLOT dapat bersifat public, protected atau private untuk menyesuaikan dengan lingkup enkapsulasi data.

```
void flyGambar::saveImage(){
    canvas->save(QFileDialog::getSaveFileName(),"PNG",0);
}
```

Fungsi dan operasi dalam SLOT dapat dijalankan dengan pemanggilan fungsi biasa atau saat SIGNAL yang bersesuaian dan telah tersambung dijalankan. Penyambungan dijalankan dengan fungsi `QObject::connect()` dalam lingkup global (**main.cpp**)

```
QObject::connect(buttonSave,SIGNAL(clicked()),foto,SLOT(saveImage()));
```

Widget

Widget adalah entitas dari jenis-jenis class yang “ditampilkan” ke layar dengan kegunaan yang spesifik, sebagai contoh: QLabel untuk menampilkan teks label, QPushButton untuk membuat *push button*, dan QTextEdit untuk kotak manipulasi teks. Ciri khas *widget* adalah metode `show()` untuk menampilkan *widget* ke tampilan.

Kita dapat membuat sebuah *widget* sendiri dengan menurunkan sebuah *class* dari salah satu *class widget* dan memanggil metode `parent()` ke QWidget.

```
class buttonDewa : public QPushButton{
Q_OBJECT
public:
explicit buttonDewa(QWidget *parent = 0); }
```

Layout

Layout adalah mekanisme penataan letak widget. Terdapat berbagai macam *class* layout: QHBoxLayout, QVBoxLayout, QGridLayout, dan lainnya. Masing-masing *class* memilki metode penganturan yang spesifik, namun secara garis besar seragam dalam metode penambahan *widget* dengan metode `addWidget()`.

Agar layout dapat ditampilkan, sebelumnya layout harus diletakkan sebagai sebuah widget.

```
QWidget widget;
QHBoxLayout layout;
layout.addWidget(&spinbox
layout.addWidget(&slider);
layout.addWidget(&button);
widget.setWindowTitle("signal-slot");
widget.setLayout(&layout);
widget.show();
```

Aplikasi Spinbox dan Slider

Berikut ini adalah contoh aplikasi sederhana dengan spinbox dan slider yang berkomunikasi dengan SIGNAL dan SLOT yang bawaan.

sigslot.pro

```
QT       += core gui widgets
TARGET = sigslot
TEMPLATE = app
SOURCES += main.cpp
```

main.cpp

```
#include <QtGui/QApplication>
#include <QHBoxLayout>
#include <QSpinBox>
#include <QSlider>
#include <QPushButton>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QSpinBox spinbox;
    QSlider slider;
    QPushButton button("quit");
    QWidget widget;
    QHBoxLayout layout;
    layout.addWidget(&spinbox);
    layout.addWidget(&slider);
    layout.addWidget(&button);
    widget.setWindowTitle("signal-slot");
    widget.setLayout(&layout);
    widget.show();
    Qobject::connect(&spinbox,SIGNAL(valueChanged(int)),\
    &slider,SLOT(setValue(int)));
    Qobject::connect(&slider,SIGNAL(valueChanged(int)),\
    &spinbox,SLOT(setValue(int)));
    QObject::connect(&button,SIGNAL(clicked()),&app,SLOT(quit()));
    return app.exec();
}
```

Berikut adalah tampilan program signal-slot, dengan sebuah *spinbox*, *slider* dan *button*.

Gambar 3. Tampilan Aplikasi Spinbox, Slider dan Button

Bagian 3: Aplikasi Diary

Operasi Berkas (File)

Penanganan berkas (file) di Qt dilakukan dengan class QFile, terdapat beberapa tahap untuk melakukan akses ke berkas: pengaturan nama berkas (filename), pembukaan berkas dengan mode tertentu, akhirnya operasi baca tulis, dan penutupan berkas. Penutupan berkas dapat dilakukan secara arbiter atau secara otomatis ketika objek yang bersangkutan dihancurkan, termasuk saat program selesai dijalankan.

Pembukaan berkas dilakukan dengan fungsi `open()`

```
QFile fileku;
fileku.setFileName("readme.txt");
fileku.open(QFile::ReadWrite);
```

Nilai yang digunakan sebagai argumen pembukaan file adalah angka yang dioperasikan dengan operator OR ('|') untuk melakukan beberapa mode sekaligus, yang berarti dapat menggunakan operasi bitwise lain untuk menyesuaikan mode. Nilai enumerasi dapat dilihat pada dokumentasi QFile.

Constant	Value	Description
QIODevice::NotOpen	0x0000	The device is not open.
QIODevice::ReadOnly	0x0001	The device is open for reading.
QIODevice::WriteOnly	0x0002	The device is open for writing.
QIODevice::ReadWrite	ReadOnly | WriteOnly	The device is open for reading and writing.
QIODevice::Append	0x0004	The device is opened in append mode, so that all data is written to the end of the file.
QIODevice::Truncate	0x0008	If possible, the device is truncated before it is opened. All earlier contents of the device are lost.
QIODevice::Text	0x0010	When reading, the end-of-line terminators are translated to '\n'. When writing, the end-of-line terminators are translated to the local encoding, for example '\r\n' for Win32.
QIODevice::Unbuffered	0x0020	Any buffer in the device is bypassed.

Tabel 1. Kontanta Enumerasi untuk Pembukaan Berkas

Operasi baca dan tulis dapat dilakukan dengan “gaya” C maupun C++. Dengan metode `read()` dan `write()` kita dapat menuliskan data sesuai *pointer* yang diberikan sebagai argumen (gaya C). Sementara dengan menggunakan QTextStream kita dapat membaca dan menulis ke berkas dengan operator “<<” dan “>>” (gaya C++).

```
fileku.write("aku imut");
fileku.read(pointer,jumlahkarakter);
QTextStream stream(&fileku);
stream << "aku imut";
stream >> pointer;
```

Operasi Tanggal dan Waktu

Operasi tangal dan waktu dilayani oleh *class* QTime yang melingkupi waktu dalam sehari, QDate yang meliputi tanggal, dan QDateTime yang melingkupi waktu dalam bentuk absolut (*epoch*). Ketiga *class* tersebut memiliki metode statik untuk mendapatkan waktu sekarang: `QDate::currentDate()`, `QTime::currentTime()` dan `QDateTime::currentDateTime()` dan nilai kembaliannya berupa *class* tersebut sendiri.

Dari objek yang menampung referensi waktu, dapat dikonvesi langsung menjadi *string* dengan metode `toString`(...) dengan menyertakan argumen sesuai format ekspresi atau menggunakan templat format.

Berikut adalah format ekspresi tanggal dan waktu.

Expression	Output
d	the day as number without a leading zero (1 to 31)
dd	the day as number with a leading zero (01 to 31)
ddd	the abbreviated localized day name (e.g. 'Mon' to 'Sun'). Uses QDate::shortDayName().
dddd	the long localized day name (e.g. 'Monday' to 'Qt::Sunday'). Uses QDate::longDayName().
M	the month as number without a leading zero (1-12)
MM	the month as number with a leading zero (01-12)
MMM	the abbreviated localized month name (e.g. 'Jan' to 'Dec'). Uses QDate::shortMonthName().
MMMM	the long localized month name (e.g. 'January' to 'December'). Uses QDate::longMonthName().
yy	the year as two digit number (00-99)
yyyy	the year as four digit number

Tabel 2. Format Ekspresi Tanggal

Expression	Output
h	the hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display)
hh	the hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display)
m	the minute without a leading zero (0 to 59)
mm	the minute with a leading zero (00 to 59)
s	the second without a leading zero (0 to 59)
ss	the second with a leading zero (00 to 59)
z	the milliseconds without leading zeroes (0 to 999)
zzz	the milliseconds with leading zeroes (000 to 999)
AP	use AM/PM display. *AP* will be replaced by either "AM" or "PM".
ap	use am/pm display. *ap* will be replaced by either "am" or "pm".

Tabel 3. Format Ekspresi Waktu

Program Diary

diary.pro

```
QT       += core gui widgets
TARGET = diary
TEMPLATE = app
SOURCES += main.cpp \
    catatan.cpp
HEADERS += \
    catatan.h
```

catatan.h

```
#ifndef CATATAN_H
#define CATATAN_H
#include <QObject>
#include <QTextEdit>
#include <QFile>
#include <QDateTime>
#include <QMessageBox>
class catatan : public QObject
{
    Q_OBJECT
public:
    explicit catatan(QObject *parent = 0);
    catatan(QFile *berkas, QTextEdit *teks);
signals:
public slots:
    void simpan();
private:
    QFile *fileku;
    QTextEdit *teksku;
};
#endif // CATATAN_H
```

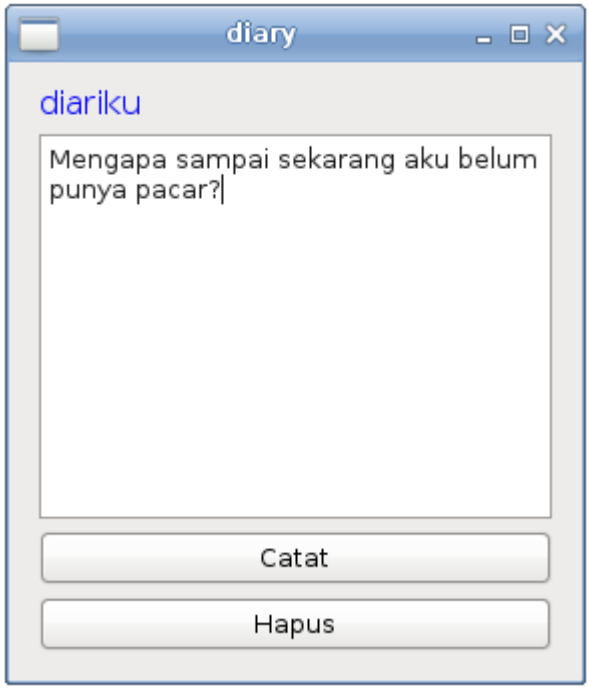
catatan.cpp

```
#include "catatan.h"
#include <QMessageBox>
catatan::catatan(QObject *parent) :
    QObject(parent)
{
}
catatan::catatan(QFile *berkas, QTextEdit *teks){
    fileku = berkas;
    teksku = teks;
}
void catatan::simpan(){
    fileku->open(QFile::WriteOnly | QIODevice::Append | QIODevice::Text);
    fileku->write(QDateTime::currentDateTime().toString("dd MMM yyyy,
hh:mm").toStdString().c_str());
    fileku->write("\n");
    fileku->write(teksku->toPlainText().toStdString().c_str());
    fileku->write("\n\n");
    fileku->close();
    QMessageBox::information(0, "Diariku", "Catatan telah disimpan");
    teksku->setText("");
}
```

main.cpp

```
#include <QApplication>
#include <QLabel>
#include <QFile>
#include <QPushButton>
#include <QBoxLayout>
#include <QTextEdit>
#include <catatan.h>
#include <QMessageBox>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QFile file(“diariku.txt”);
    QLabel judul("<font color=blue size=4>diariku</font>");
    QTextEdit kotakteks;
    QPushButton catat("Catat");
    QPushButton hapus("Hapus");
    QVBoxLayout layout;
    layout.addWidget(&judul);
    layout.addWidget(&kotakteks);
    layout.addWidget(&catat);
    layout.addWidget(&hapus);
    catatan hehe(&file,&kotakteks);
    QObject::connect(&hapus,SIGNAL(clicked()),&kotakteks,SLOT(clear()));
    QObject::connect(&catat,SIGNAL(clicked()),&hehe,SLOT(simpan()));
    QWidget tampil;
    tampil.setLayout(&layout);
    tampil.show();
    return a.exec();
}
```

Tampilan program **diary**:



Gambar 4. Tampilan Program **diary**

Bagian 4: Realtime Plotting

Qwt: QwtPlot

Plot grafik di Qt dilakukan dengan menggunakan library tambahan **Qwt.** Untuk sebelum digunakan Qwt harus dikompilasi terlebih dahulu. Langkah yang dilakukan untuk melakukan kompilasi Qwt secara garis besar adalah:

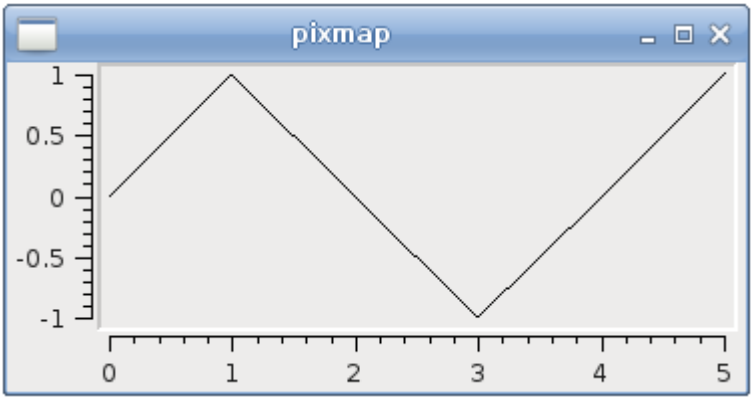
1) pastikan `qmake` dan `make` (atau ekivalen, eg: ming32-make) terakses oleh PATH sistem operasi,
2) ekstrak arsip Qwt, eg: qwt-6.1-rc3.tar.gz,
3) pindah direktori kerja ke direktori ekstraksi qwt-6.1-rc3.tar.gz,
4) jalankan qmake, make, make install

Setelah dikompilasi, Qwt dapat digunakan dengan menyertakan *project feature file* (***.prf**) ke dalam berkas *project.*

```
include (/usr/local/qwt/features/qwt.prf)
include (C:/qwt/features/qwt.prf)
```

Pembuatan sebuah plot sederhana dilakukan dengan objek dari *class* QwtPlot dan QwtPlotCurve, dua buah larik dengan panjang sama dengan tipe double.

```
#include <qwt_plot.h>
#include <qwt_plot_curve.h>
…
double x[6]={0,1,2,3,4,5};
double y[6]={0,1,0,-1,0,1};
QwtPlot plot;
QwtPlotCurve kurva;
kurva.setRawSamples(x,y,6);
kurva.attach(&plot);
plot.show();
```



Gambar 5. Plot Sederhana dari 2 Larik dengan 6 Elemen

Thread

Thread adalah sebuah eksekusi independen dalam sebuah program, memiliki waktu eksekusinya sendiri sehingga dapat berjalan secara *pseudo* real-time. Karakteristik thread mirip seperti rutin prosedur pada pemrograman berlogika sekuensial.

Qt mengimpelentasikan *thread* dengan Qthread. Secara garis besar, hal yang harus dilakukan untuk menjalankan *thread* adalah:

1) membuat *class* inheritans dari QThread,
2) mendefinisikan rutin thread pada fungsi `run(),`
3) memulai *thread* dengan fungsi `start().`

Berikut adalah contoh implementasi *thread* di berkas *header*:

```
#include <QThread>
class foothread : public QThread
{
    Q_OBJECT
public:
    explicit foothread(QObject *parent = 0);
private:
    void run();
};
```

sementara berkas source berisi:

```
void foothread::run(){
    while(1){
        // do lame things here
    }
}
```

Di rutin lain (eg: **main.cpp**), kita memulai *thread*:

```
foothread underdamp;
underdamp.start(QThread::NormalPriority);
```

Program osilasi

osilasi.pro

```
include (/usr/local/qwt/features/qwt.prf)
QT       += core gui widgets
TARGET = osilasi
TEMPLATE = app
SOURCES += main.cpp \
    logika.cpp \
    grafik.cpp
HEADERS += \
    logika.h \
    grafik.h
```

grafik.h

```
#ifndef GRAFIK_H
#define GRAFIK_H
#include <QWidget>
#include <qwt_plot.h>
#include <qwt_plot_curve.h>
#define SAMPLE 20
#define PANJANG 500
#define TINGGI 200
class grafik : public QwtPlot
{
    Q_OBJECT
public:
    explicit grafik(QwtPlot *parent = 0);
signals:
public slots:
    void newValue(double x, double y);
private:
    double dataX[SAMPLE];
    double dataY[SAMPLE];
    QwtPlotCurve kurva;
    int samplecount;
};
#endif // GRAFIK_H
```

logika.h

```
#ifndef LOGIKA_H
#define LOGIKA_H
#include <QThread>
class logika : public QThread
{
    Q_OBJECT
public:
    explicit logika(QObject *parent = 0);
signals:
    void newValue(double X, double Y);
public slots:
    void suspendThread();
    void continueThread();
private:
    void run();
    double arus, waktu;
};
#endif // LOGIKA_H
```

grafik.cpp

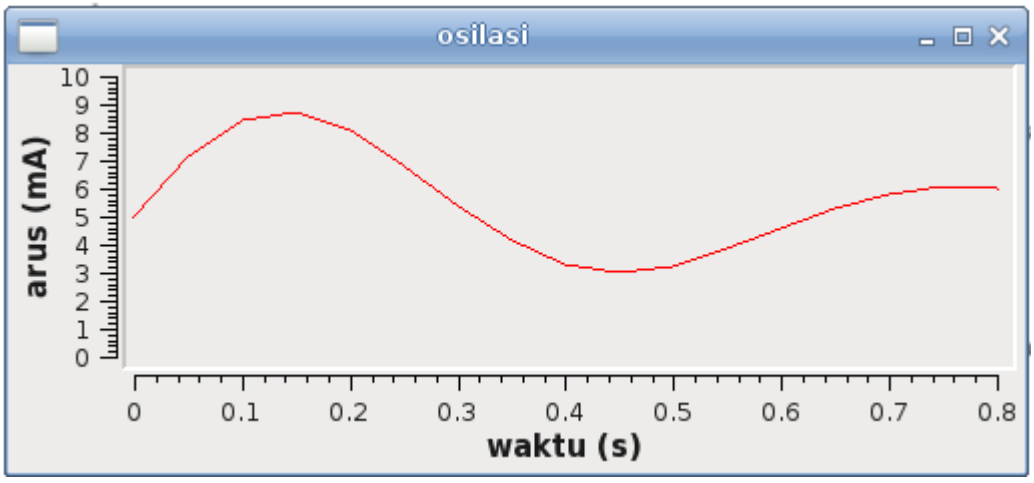
```
#include "grafik.h"
grafik::grafik(QwtPlot *parent) : QwtPlot(parent){
    setAxisTitle(yLeft,"arus (mA)");
    setAxisTitle(xBottom,"waktu (s)");
    setMinimumSize(PANJANG,TINGGI);
    kurva.setRawSamples(dataX,dataY,0);
    kurva.setPen(QColor(Qt::red));
    kurva.attach(this);
    setAutoReplot(true);
    setAxisScale(yLeft,0,10,1);
}
void grafik::newValue(double x, double y){
    int i;
    if (samplecount<SAMPLE-1){
        dataX[samplecount] = x;
        dataY[samplecount] = y;
        qDebug("%d %f
%f\n",samplecount,dataX[samplecount],dataY[samplecount]);
        samplecount++;
        kurva.setRawSamples(dataX,dataY,samplecount);
    }
    else {
        for (i=0; i<SAMPLE-1; i++){
            dataX[i]=dataX[i+1];
            dataY[i]=dataY[i+1];
        }
        dataX[SAMPLE-1] = x;
        dataY[SAMPLE-1] = y;
        qDebug("akhir %d %f
%f\n",samplecount,dataX[samplecount],dataY[samplecount]);
        kurva.setRawSamples(dataX,dataY,samplecount);
    }
}
```

logika.cpp

```
#include "logika.h"
#include <math.h>
#include <stdio.h>
// underdamp dengan persamaan i(t) = I + B e-at sin(wt)

logika::logika(QObject *parent) :
    QThread(parent)
{
    waktu=0;
}
void logika::run(){
    while(1){
        arus = 5 + (5*exp(-2*waktu)*sin(10*waktu));
        emit newValue(waktu,arus);
        waktu+=0.05;
        msleep(200);
    }
}
void logika::suspendThread(){
    sleep(~0);
}
void logika::continueThread(){
    start(QThread::NormalPriority);
}
```

main.cpp

```
#include <QApplication>
#include <QThread>
#include <grafik.h>
#include <logika.h>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    grafik *grafikku=new grafik();
    grafikku->show();
    logika *underdamp = new logika();

QObject::connect(underdamp,SIGNAL(newValue(double,double)),grafikku,SLOT(ne
wValue(double,double)));
    underdamp->start(QThread::NormalPriority);
    return a.exec();
}
```



Gambar 6. Tampilan Program **osilasi**

Referensi

Jasmin Blanchette; Mark Summerfield; in association with Trolltech Press.2008. *C++ GUI Programming with Qt 4, Second Edition*. Massachusetts: Prentice Hall.

Andreas Jakl. 2009. *Basics of Qt*.

Scott Collins. 2005. *A Deeper Look at Signals and Slots*.

Digia Plc and/or its subsidiaries. 2013. *Qt 5.0 Reference Documentation*. QtDoc 5.0. *http://qt-project.org/doc/qt-5.0/qtdoc*

Nokia Corporation and/or its subsidiary(-ies). 2009. *Qt 4.6 Whitepaper*

Nokia. 2013 *Archived:Porting iPhone native (Objective-C) applications to Qt for Symbian. http://www.developer.nokia.com/Community/Wiki/index.php?title=Archived:Porting_iPhone_native_%28Objective-C%29_applications_to_Qt_for_Symbian&oldid=175319*