# Программирование на языке C++

*Основы работы с QT*

Штанюк А.А.

27 апреля 2015 г.

# Введение


Что такое QT (кьют)?

- Кроссплатформенный фреймворк для разработки приложений на C++
- Набор классов для GUI, XML, Network и др.
- Библиотека для множества программ (KDE, Skype, VLC Player, VirtualBox, Mathematica, Google Earth)

Поддержка: **Unix, Linux, Windows, Mac OS X, Android, iOS, Embedded....**

# Введение

Поддержка IDE:

- QTCreator
- Eclipse
- MS Visual Studio

Программы могут компилироваться из командной строки.

# Введение

Модульная архитектура:

- Qt Core - базовая, non-GUI функциональность
- Qt GUI - основы разработки приложений с GUI
- Qt Multimedia - работа с мультимедиа (аудио, видео, радио, камеры и пр.)
- Qt Network - поддержка сети
- Qt QML - поддерка языков **QML** и **JavaScript**
- Qt SQL - поддержка интерфейса с СУБД
- Qt Test - моддержка модульного тестирования
- Qt WebKit - обеспечение браузерного ядра
- Qt Widgets - расширение библиотеки виджетов

# Введение

Модульная архитектура:

- Qt Bluetooth - операции с протоколом Bluetooth
- Qt OpenGL - поддержка работы с OpenGL
- Qt Sensor - работа с сенсорными устройствами
- Qt XML - поддержка работы с XML-документами

Статистика: более 500 классов и 10000 функций.

Последняя версия на начало 2015 года: **5.4**

# Введение

Web- сайт:http://www.qt.io/

Основные компании, владевшие торговой маркой QT:

- Trolltech
- Nokia
- Digia

Лицензии:

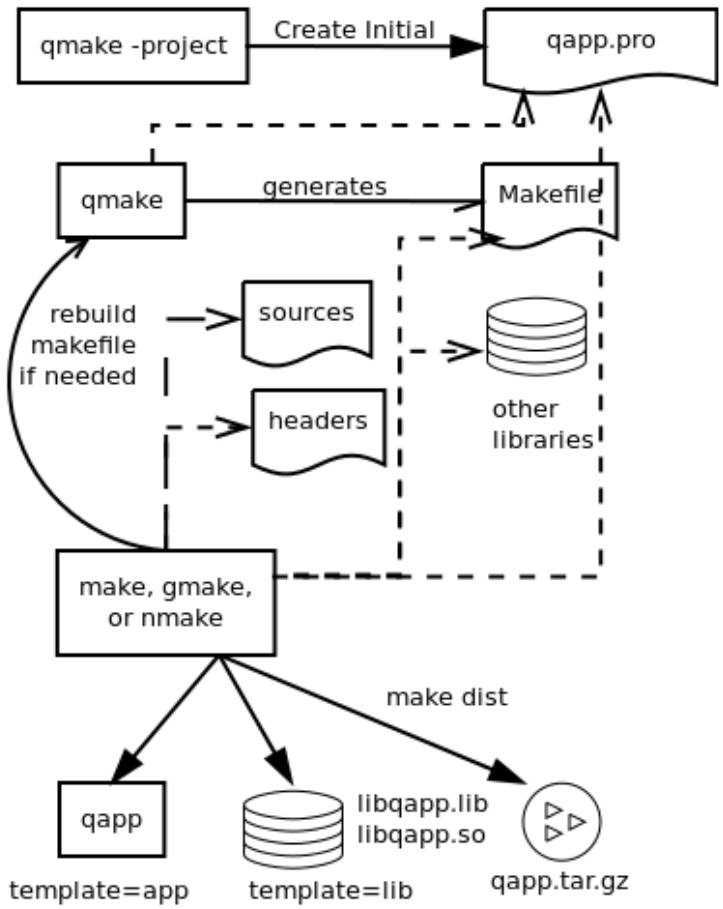
1. **Коммерческая** - полный контроль над кодом, доп. поддержка, инструменты.
2. **Свободная** - используется GPL.

# Установка в Linux

1. Загружается установочный файл с сайта `https://www.qt.io/ru/download`
2. Установочному файлу даются права на выполнение, например **chmod +x qt-install.run**
3. Запускается online-установка, например **./qt-install.run**
4. Установить SDK (для правильной работы qmake): **apt-get install qt-sdk** (команда установки в linux ubuntu)

# Установка в Linux

Схема построения приложения:



# Первая программа

Создаем файл **first.cpp** в каталоге **first**:

```
#include <QApplication>
#include <QTextEdit>
int main(int argv, char **args)
{
    QApplication app(argv, args);

    QTextEdit textEdit;
    textEdit.show();

    return app.exec();
}
```

# Первая программа

Для создания проекта запускаем в каталоге **first**

```
qmake -project
```

Образуется файл **first.pro**:

```
TEMPLATE = app
TARGET = first
INCLUDEPATH += .
SOURCES += first.cpp
```

Добавляем строку (после *INCLUDEPATH*):

```
QT += widgets
```

# Первая программа

Создаем **Makefile**:

```
qmake
```

Компилируем:

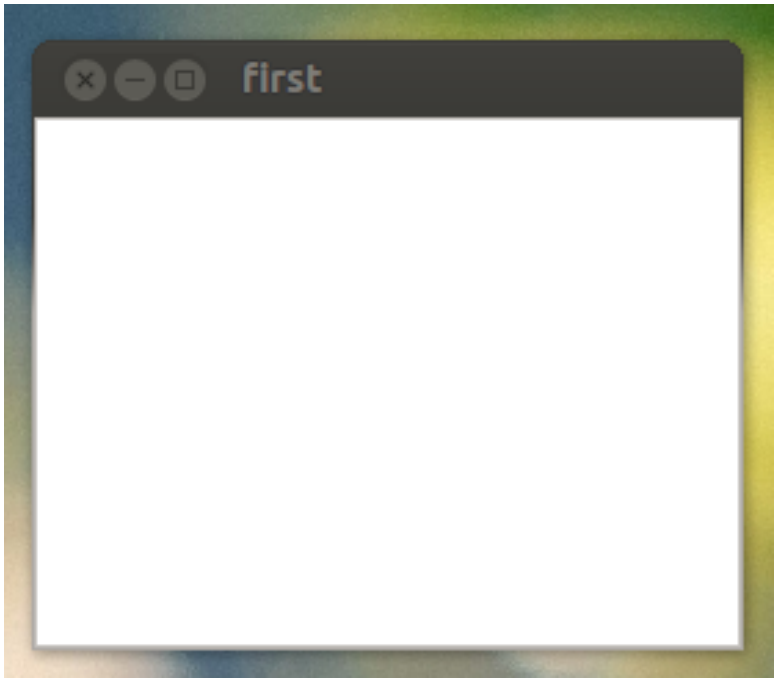
```
make
```

Запускаем:

```
./first
```

# Первая программа

Результат (в **linux ubuntu**):



# Консольное приложение

В QT можно создавать консольные приложения:

```
#include <QCoreApplication>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    cout<<" hello world"<<endl;
    return a.exec();
}
```

Каркас приложения **QCoreApplication** не предполагает графического окна, поэтому редактировать проектный файл не нужно

# Консольное приложение

Альтернатива:

```
#include <QDebug>
int main(int argc, char *argv[])
{
    qDebug() <<"Hello World"<< endl;
    return 0;
}
```

# Сигналы и слоты

Сигналы и слоты используются для связи между объектами.

## Сигналы

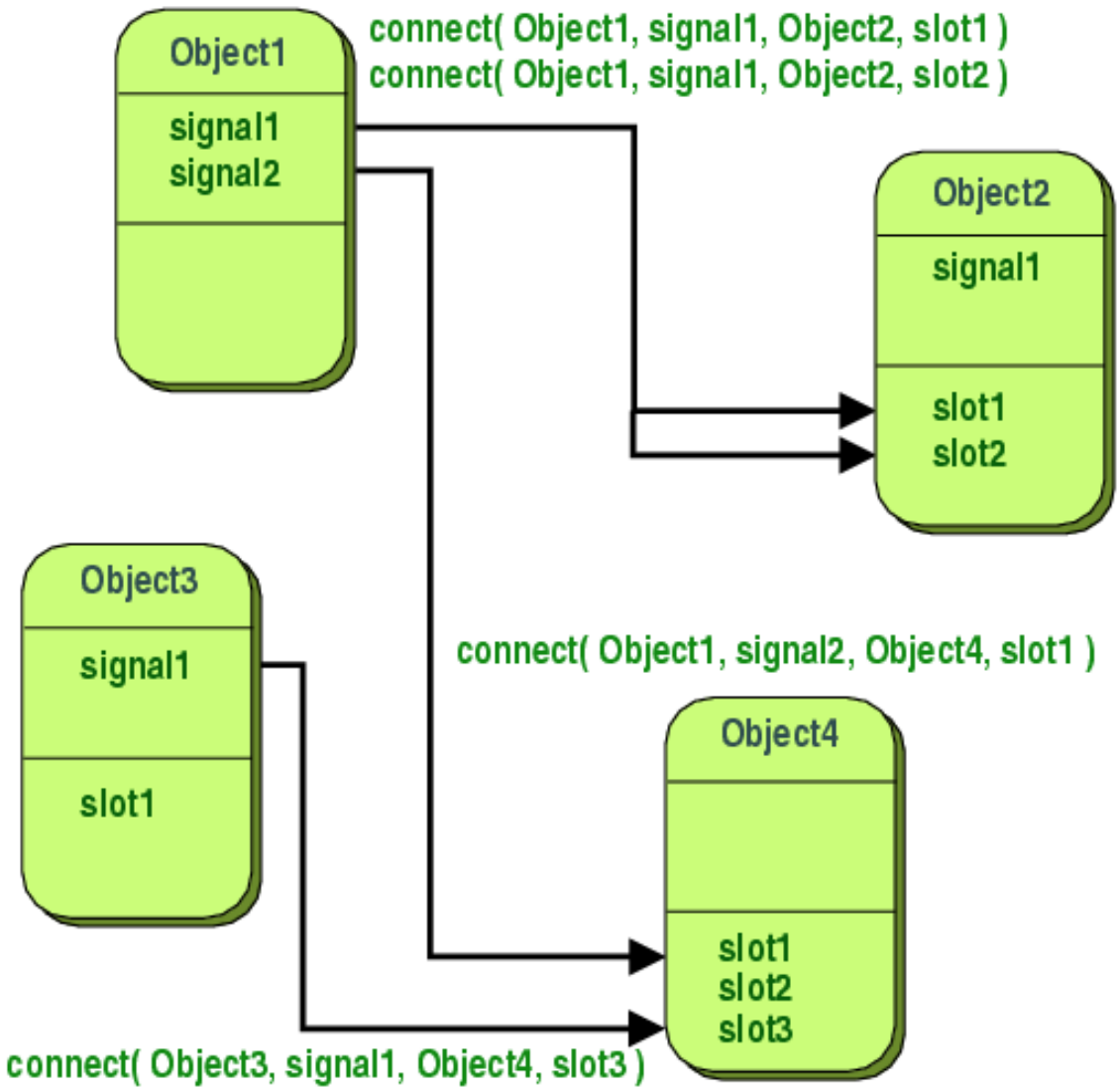
Сигналы - методы, осуществляющие пересылку сообщений от одного объекта к другому

## Слоты

Слоты - методы, принимающие сигналы

Слот вызывается когда вырабатывается сигнал, с которым он связан

# Сигналы и слоты



# Приложение Notepad

Рассмотрим приложение Notepad

Вначале разработаем заголовочный файл окна:

```
#include <QWidget>
#include <QTextEdit>
#include <QPushButton>
class Notepad : public QWidget
{
    Q_OBJECT
    public:
        Notepad(QWidget *parent = 0);
    private slots:
        void quit();
    private:
        QTextEdit *textEdit;
        QPushButton *quitButton;
};
```

# Приложение Notepad

Файл **notepad.cpp** с реализацией методов:

```
#include "notepad.h"
#include <QVBoxLayout>
#include <QApplication>
Notepad::Notepad(QWidget *parent)  {
    textEdit = new QTextEdit;
    quitButton = new QPushButton(tr("Quit"));
    connect(quitButton, SIGNAL(clicked()), this, SLOT(quit()));
    QVBoxLayout *layout = new QVBoxLayout;
    layout->addWidget(textEdit);
    layout->addWidget(quitButton);
    setLayout(layout);
    setWindowTitle(tr("Notepad"));
}
void Notepad::quit() {
   QApplication::exit();
}
```

# Приложение Notepad

Файл **main.cpp**:

```
#include "notepad.h"
#include <QApplication>
int main(int argc, char*argv[])
{
    QApplication a(argc,argv);
    Notepad note;
    note.show();
    return a.exec();
}
```

# Приложение Notepad

Файл проекта **notepad.pro** создаем вручную:

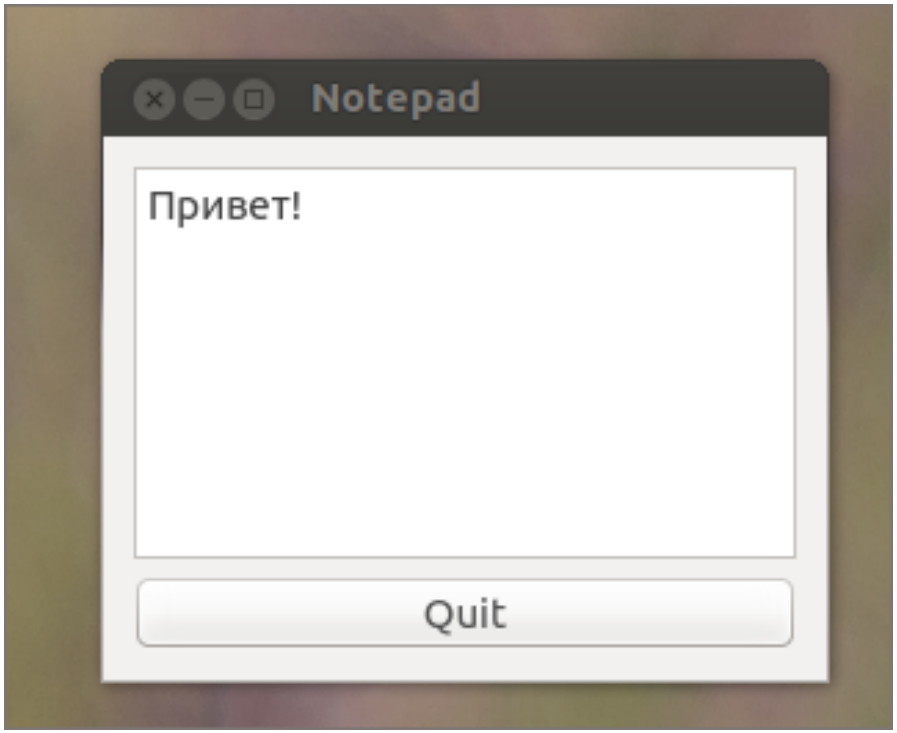
```
HEADERS = notepad.h
SOURCES = notepad.cpp \
          main.cpp
QT += widgets
```

Компилируем проект:

```
qmake
make
```

# Приложение Notepad

Результат:



# MOC

## MOC - метаобъектный компилятор

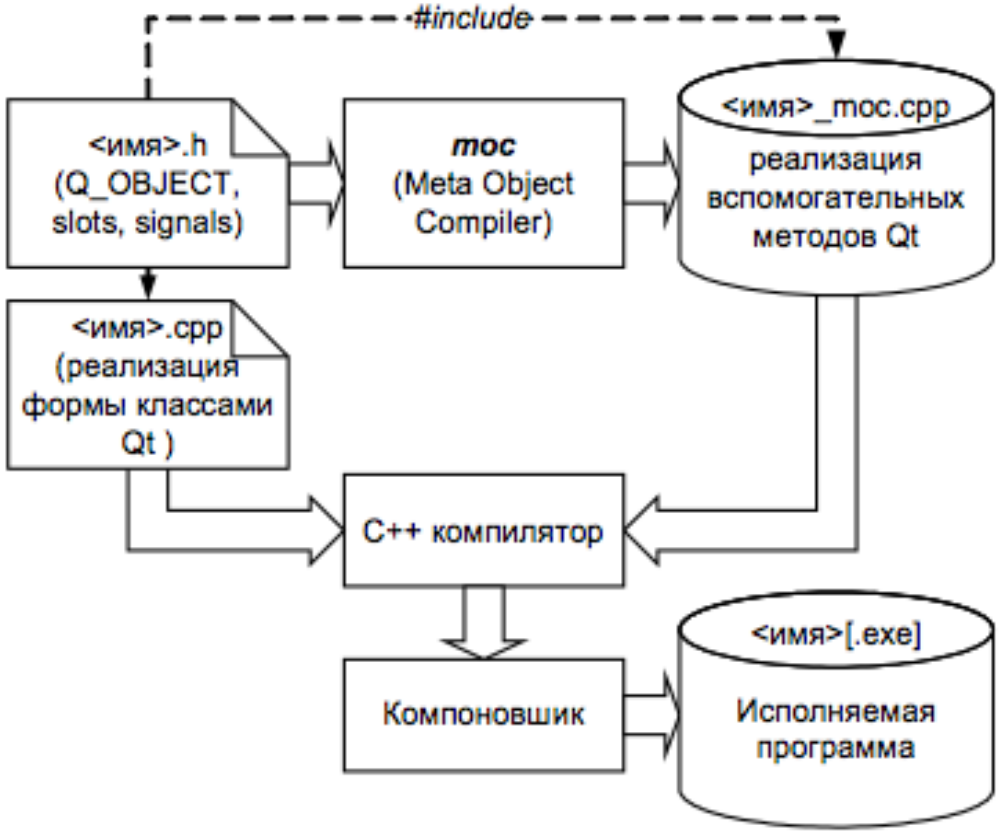
Мета-объектный компилятор, moc, - программа, которая обрабатывает расширения C++ от Qt.

Инструмент moc читает заголовочный файл C++. Если он находит одно или более объявлений классов, которые содержат макрос Q_OBJECT, то он порождает файл исходного кода C++, содержащий мета-объектный код для этих классов.

Мета-объектный код требуется механизму сигналов и слотов, информации о типе времени выполнения и системы динамических свойств.

# MOC

С учетом использования **MOC**, схема построения приложения немного меняется:



# Компоновка GUI

Для компоновки виджетов используются специальные классы компоновки

- QHBoxLayout - горизонтальное расположение.
- QVBoxLayout - вертикальное расположение.
- QGridLayout - двумерное расположение.
- QFormLayout - расположение в два столбца (как на бланках).

# Компоновка GUI

Пример кода с компоновкой:

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

QHBoxLayout *layout = new QHBoxLayout;
layout->addWidget(button1);
layout->addWidget(button2);
layout->addWidget(button3);

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

# Компоновка GUI

Пример использования **QGridLayout**:

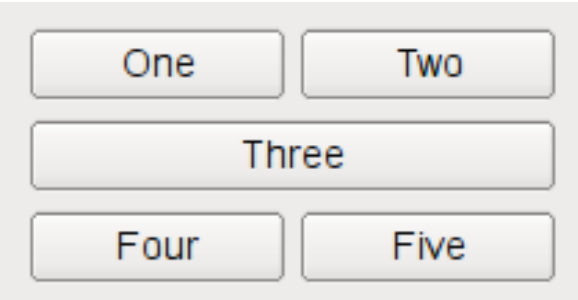
```
QWidget *window = new QWidget;
QPushButton *button1 = new QPushButton("One");
QPushButton *button2 = new QPushButton("Two");
QPushButton *button3 = new QPushButton("Three");
QPushButton *button4 = new QPushButton("Four");
QPushButton *button5 = new QPushButton("Five");

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();
```

# Компоновка GUI

Результат:



# Компоновка GUI

Пример использования **QFormLayout**:

```
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();
```

# Компоновка GUI

Результат:

# Стандартные диалоги

Стандартные диалоги используются для выбора параметров: указания имен файлов и т.п., а также для вывода на экран информационных сообщений.

Диалог открытия файла (на чтение и запись):

```
QString fname_open = QFileDialog::getOpenFileName(this,
        "Список станций", "~/Data", "TXT Files (*.txt)");
QString fname_save = QFileDialog::getSaveFileName(this,
        "Список станций", "~/Data", "TXT Files (*.txt)");
```

Диалог для открытия файлов возвращает имя *существующего* файла.

# Стандартные диалоги

Среди вариантов диалогов для файлов:

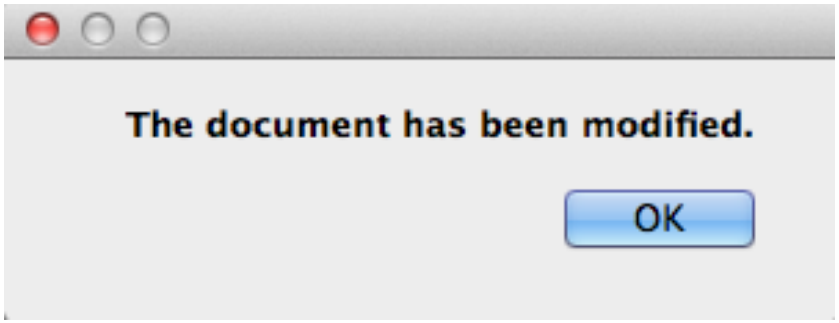
- getExistingDirectory
- getExistingDirectoryUrl
- getOpenFileNames
- getOpenFileUrl
- getOpenFileUrls
- getSaveFileUrl

Основное отличие функции с суффиксом *Url* - возможность работы с сетевыми ресурсами.

# Стандартные диалоги

Класс диалога для вывода информационных сообщений QMessageBox

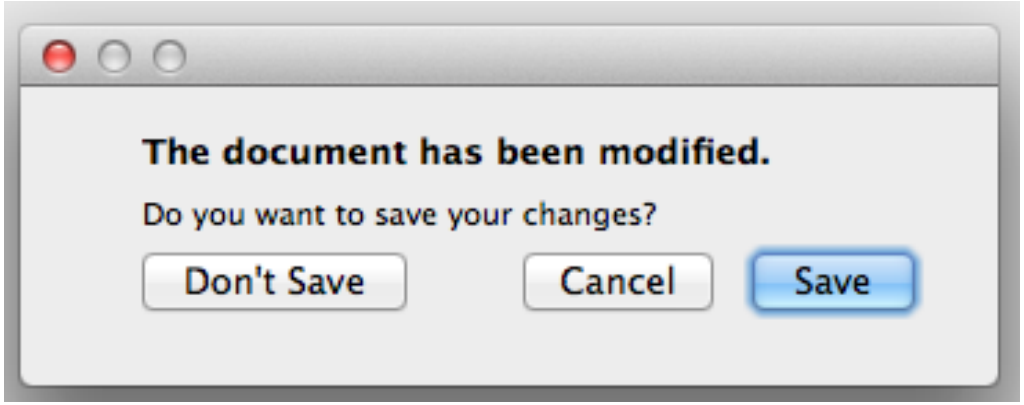
```
QMessageBox msgBox;
msgBox.setText("The document has been modified.");
msgBox.exec();
```



# Стандартные диалоги

Более сложные случаи требуют настройки кнопок и анализ результата

```
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();
```



# Стандартные диалоги

Статические члены класса QMessageBox используются для вывода сообщений без создания экземпляра

- critical
- information
- question
- warning

Например:

```
QMessageBox::information(this, "Заголовок", "Текст сообщения");
```

# Добавление формы Qt Designer

После добавления в проект новой формы, для ее активизации нужно сделать:

Добавить заголовочный файл новой формы, например:

```
#include "ui_dialog.h"
```

Создать экземпляр класса **QDialog** и связать с описанием формы, а потом активизировать:

```
Ui::Dialog dialog;
QDialog *qdialog=new QDialog();
dialog.setupUi(qdialog);
qdialog->exec();
```

# Пользовательские диалоги

Пользовательские диалоги представляют из себя формы, создаваемые и вызываемые в программе помимо главного окна.

В QCreator существуют 2 режима добавления новой формы:

1. Класс формы Qt Designer
2. Форма Qt Designer

Основная разница в том, что в первом случае в программу добавляется полноценный класс для ручного редактирования.

# Пользовательские диалоги

Если после заполнения виджетов, к ним нужно получить доступ из главного окна:

```
Ui::Dialog dialog;
...
QString data=dialog.lineEdit->text();
```

# Пользовательские диалоги

Второй способ: через добавление класса

```
// dialog.h
#include <QDialog>
namespace Ui {
class Dialog;
}
class Dialog : public QDialog
{
    Q_OBJECT
public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();
private:
    Ui::Dialog *ui;
};
```

# Пользовательские диалоги

```
// dialog.cpp
#include "dialog.h"
#include "ui_dialog.h"
Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
}
Dialog::~Dialog()
{
    delete ui;
}
```

# Пользовательские диалоги

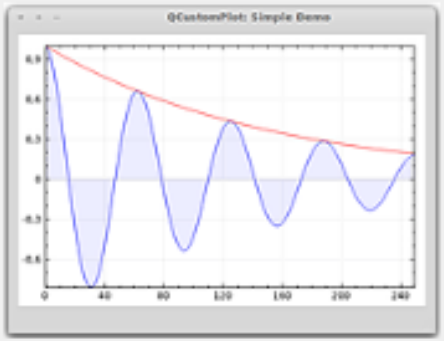
Для активизации такой формы в главном окне:

```
Dialog *dialog=new Dialog();
dialog->exec()
```

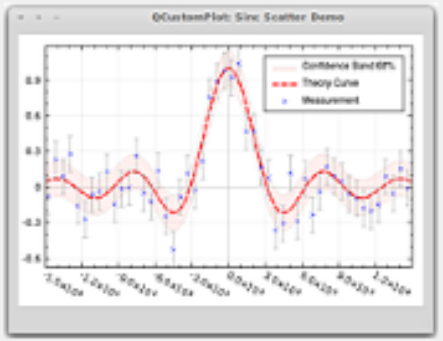
# Преобразование виджетов

Рассмотрим преобразование виджетов на примере создания формы с графиком функции.

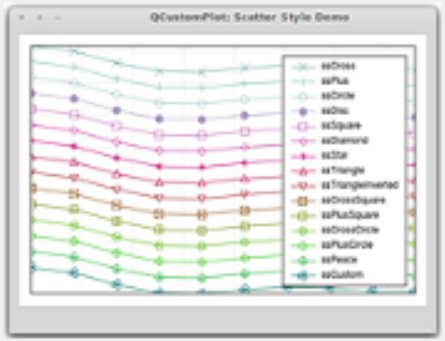
Будем использовать виджет с сайта `www.qcustomplot.com`



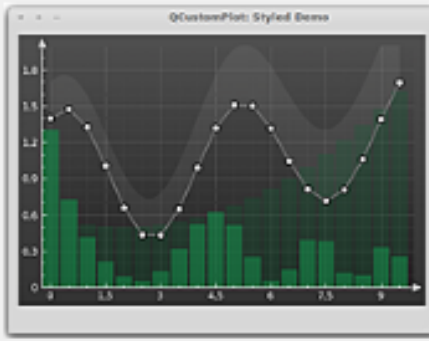
A simple decaying sine function with fill and its exponential envelope in red



sinc function with data points, corresponding error bars and a 2-sigma confidence band



A demonstration of several scatter point styles



Demonstrating QCustomPlot's versatility in styling the plot

# Преобразование виджетов

В проект добавляем **qcustomplot.h** и **qcustomplot.cpp**.

В файле проекта добавляем (для QT 5)

```
QT += printsupport
```

На форму помещаем **Widget**, а потом преобразуем его к типу **QCustomPlot**.

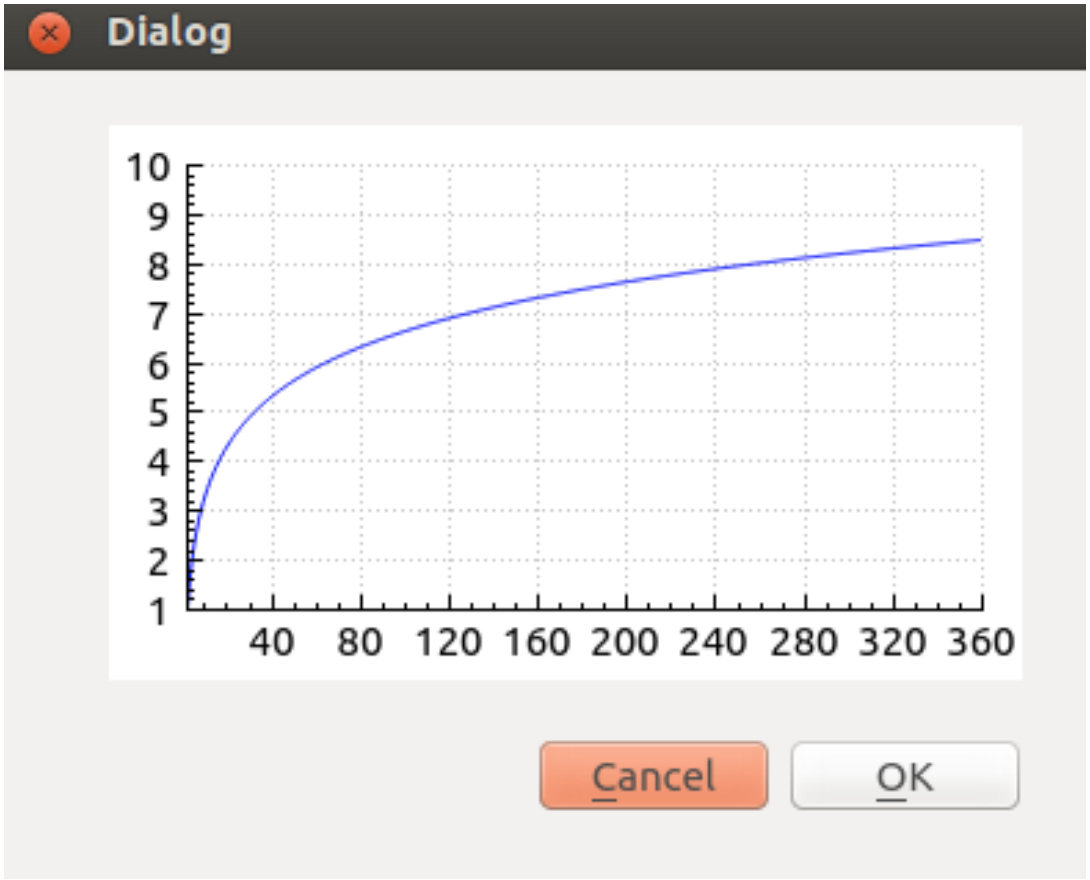
# Преобразование виджетов

В конструктор формы с графиком добавляем:

```
QVector<double> x(500),y(500);
for(int i=0;i<500;i++) {
 x[i]=i+2;
 y[i]=log10(x[i])/log10(2);
}
ui->widget->addGraph();
ui->widget->graph(0)->setData(x,y);
ui->widget->xAxis->setRange(2,360);
ui->widget->xAxis->setRange(1,10);
ui->widget->replot();
```

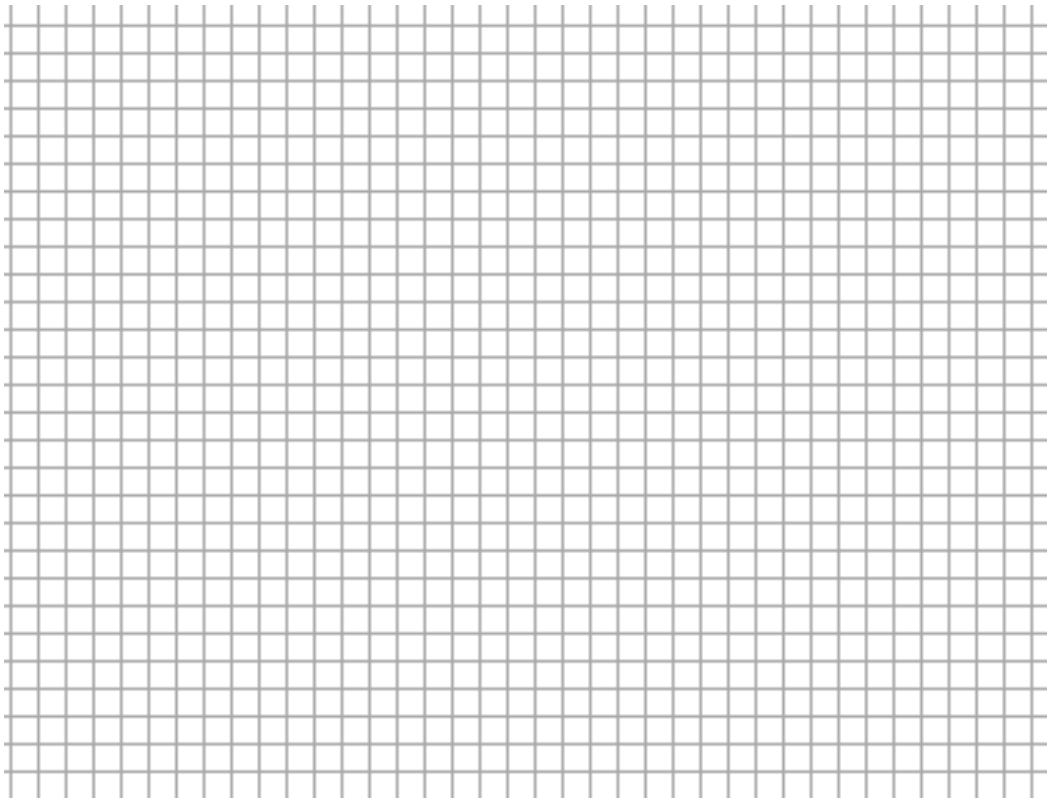
# Преобразование виджетов

Результат:



# Табличный виджет

```
QTableWidget *table=new QTableWidget(N,M,this);
```



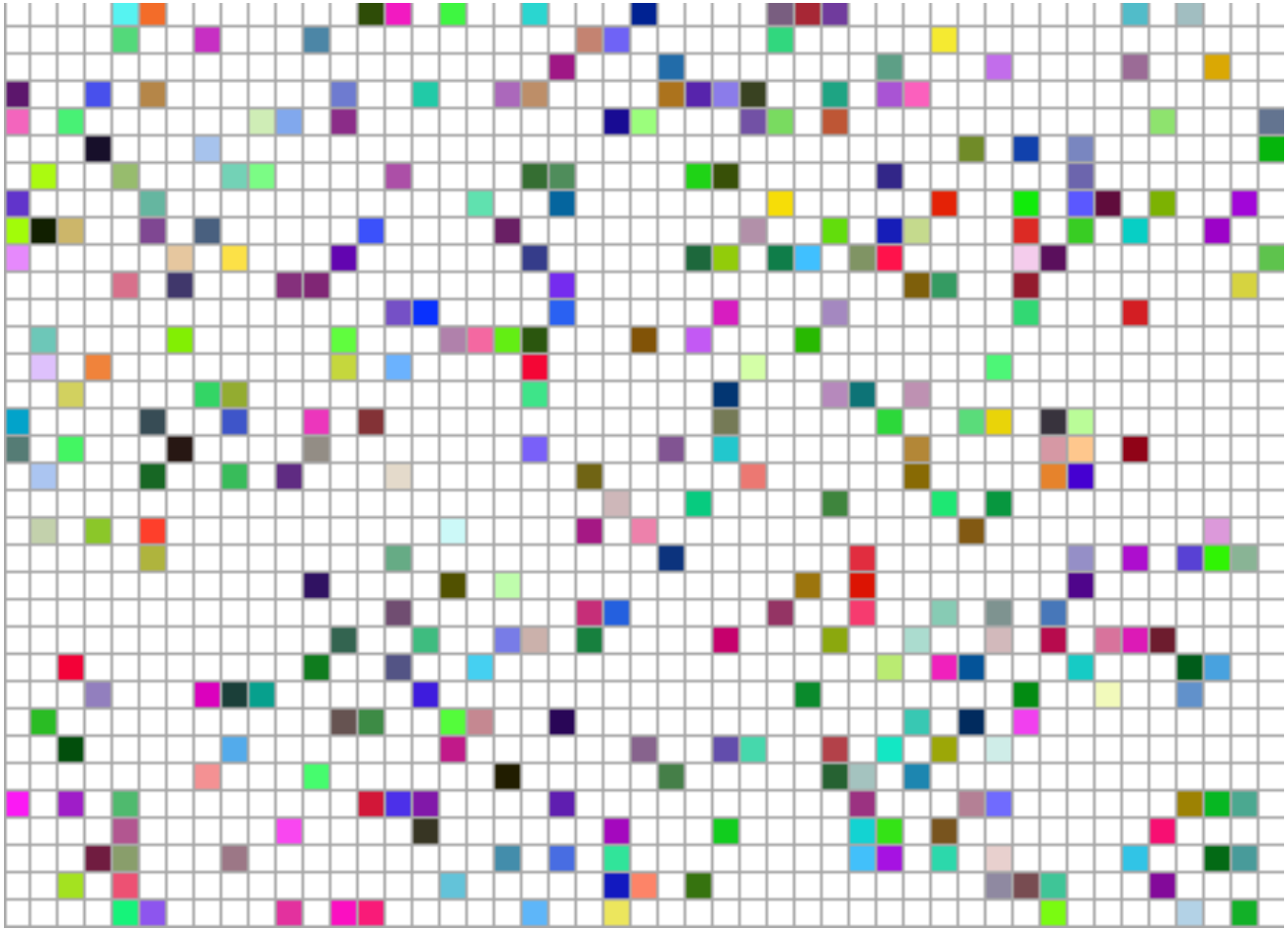
# Табличный виджет

Настраиваем виджет (размер 80x100, без заголовков строк и столбцов). Создаем содержимое ячеек (**QTableWidgetItem**) и устанавливаем случайный цвет фона

```
for(int n=0;n<100;n++)
  {
     int x=rand()%100;
     int y=rand()%80;
     QTableWidgetItem *item=new QTableWidgetItem();
     item->setBackgroundColor(QColor(rand()%256,rand()%256,rand()%256));
     ui->tableWidget->setItem(y,x,item);
  }
```

# Табличный виджет

Результат:



# Библиографический список I



Википедия
*https://ru.wikipedia.org/wiki/Qt*


Бланшет, Саммерфилд
*Qt4 Программирование GUI на C++ 2ed*
2008


Шлее Макс
*Qt 4.8. Профессиональное программирование на C++*
2012


Марк Саммерфилд
*Qt Профессиональное программирование (High tech)*
2011


Андрей Боровский
*Qt 4.7+. Практическое программирование на C++*
2012


Alan Ezust, Paul Ezust
*Introduction to Design Patterns in C++ with Qt*
2011

# Библиографический список II


Список примеров программ на QT
*http://doc.qt.io/qt-5/all-examples.html*