GUI Development with QT

Alex X. Liu

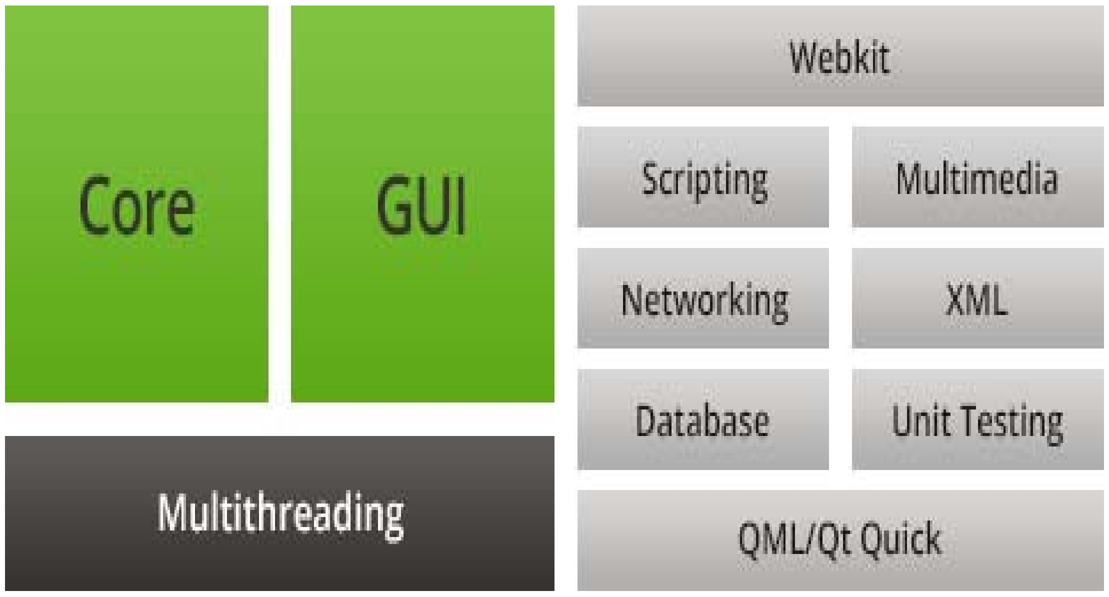
QT

- QT is a GUI development framework
- Two reasons that we choose QT:
  - Cross platform application development framework for GUI
  - Widely used: Skype, Google Earth, Adobe Photoshop Album
- Work with C++, Java, phthon, C#, Ruby, PHP, Perl, Pascal, Ada
  - QT for Java: QT Jambi
- Software: You need to download and install QT from https://www.qt.io/download-open-source/
  - You need license if you sell your software.
- Documentation: http://doc.qt.io/
- Tutorial: https://www.youtube.com/watch?v=6KtOzh0StTc
  - 152 lectures
- For C++ applications with GUI, we will use QtCreator.

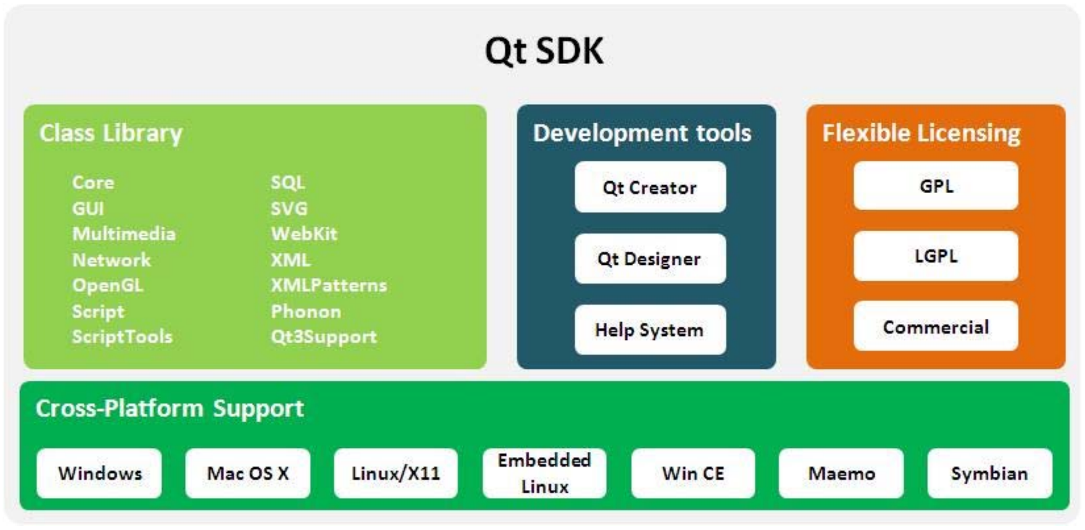
Tools

- Qt Creator : Cross platform IDE
- Qt Designer: GUI layout and forms builder
- Qt Linguest: Internationalization toolset
- Qt Assist: Customizable documentation reader
- Qt Qmake: Cross platform build tool
- Plugin for other IDE: Integration with Visual Studio and Eclipse
- Configure: Tool to configure Qt on any specific platform
- Qt SDK: Rich C++ library

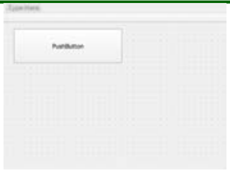
Qt Modular Class Library



Qt Architecture



QT Example 1: Button, Signal, Slot



```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
   QMainWindow(parent),
   ui(new Ui::MainWindow){
   ui->setupUi(this);
   ui->AlexButton->setText("Close");
   connect(ui->AlexCloseButton, SIGNAL(clicked()), this, SLOT(close()));
}
MainWindow::~MainWindow(){
   delete ui;
}
```

Q_OBJECT macro

- Qt's meta-object system provides the signals and slots mechanism for inter-object communication, run-time type information, and the dynamic property system.
- The meta-object system is based on three things:
  - The QObject class provides a base class for objects that can take advantage of the meta-object system.
  - The Q_OBJECT macro inside the private section of the class declaration is used to enable meta-object features, such as dynamic properties, signals, and slots.
  - The Meta-Object Compiler (moc) supplies each QObject subclass with the necessary code to implement meta-object features.
- The moc tool reads a C++ source file. If it finds one or more class declarations that contain the Q_OBJECT macro, it produces another C++ source file which contains the meta-object code for each of those classes. This generated source file is either #include'd into the class's source file or, more usually, compiled and linked with the class's implementation.

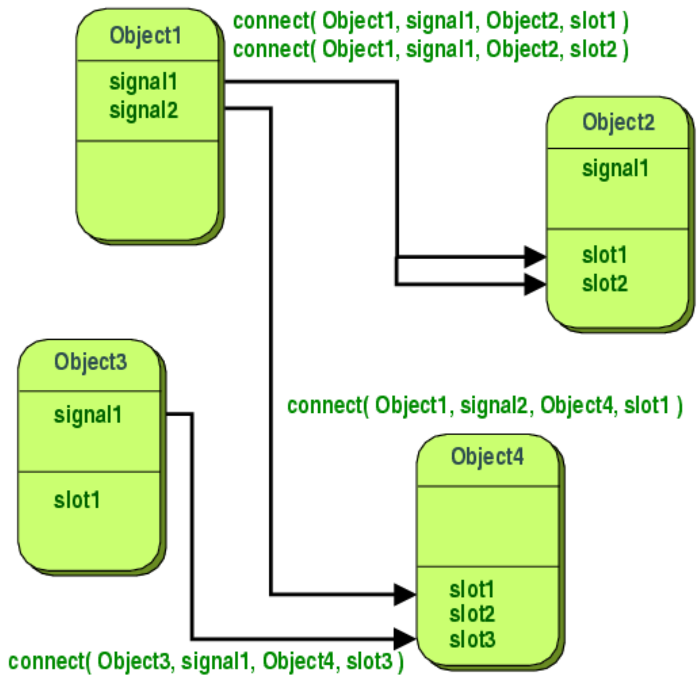
Signals and Slots

- Used for communication between objects.
- A signal is emitted when a particular event occurs.
- A slot is a function that is called in response to a particular signal.
- Qt's widgets have many pre-defined signals and slots.
  - You can subclass widgets and add your own slots.
- The signature of a signal must match the signature of the receiving slot.
- All classes that inherit from QObject or one of its subclasses (e.g., QWidget) can contain signals and slots.
- Use the “connect” method to join signal and slots

```
connect( slider,  SIGNAL(valueChanged(double)) ,
                  plot,  SLOT(setIntervalTime(double)));
```

- Signal is sent, or emitted using the keyword emit (i.e. Emit someSignal(“Hello”)).
- Parsed by QT’s Meta Object Compiler (moc)

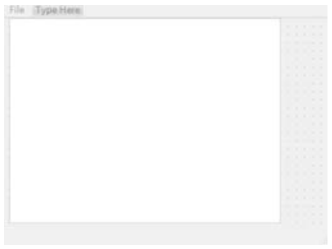
Signals and Slots

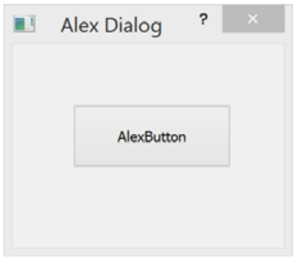

Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

QT Example 2: Slider, ProgressBar

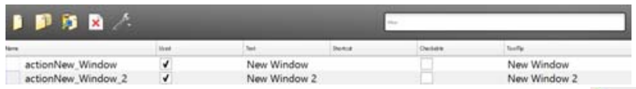
```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) :
   QMainWindow(parent),
   ui(new Ui::MainWindow){
   ui->setupUi(this);
   connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),
         ui->progressBar,SLOT(setValue(int)));
   connect(ui->horizontalSlider,SIGNAL(valueChanged(int)),
         ui->progressBar_2,SLOT(setValue(int)));
   //disconnect(ui->horizontalSlider,SIGNAL(valueChanged(int)),
   //          ui->progressBar,SLOT(setValue(int)));
}
```

QT Example 3: Menu, New Window



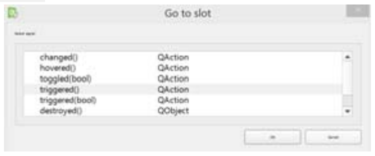


- Forms->Add New->QT->QT Designer Form Class
- On the mainwindow designer, File->Type Here->New Window
- On the bottom action designer, it shows

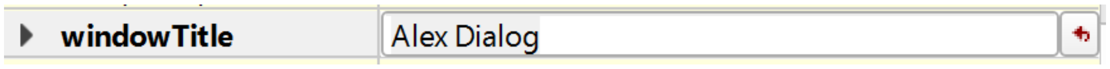


Name	Used	Text	Shortcut	Checkable	ToolTip
actionNew_Window	✓	New Window		☐	New Window
actionNew_Window_2	✓	New Window 2		☐	New Window 2

- Right click actionNew_Window->Go to slot
- Edit window title:



changed()	QAction
hovered()	QAction
toggled(bool)	QAction
triggered()	QAction
triggered(bool)	QAction
destroyed()	QObject



windowTitle: Alex Dialog

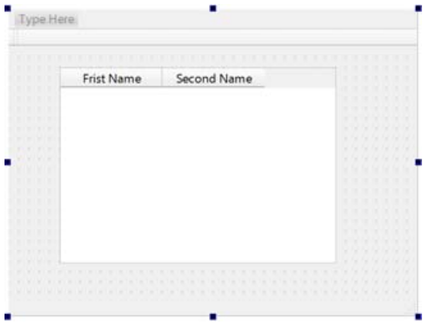
Use heap, Not stack

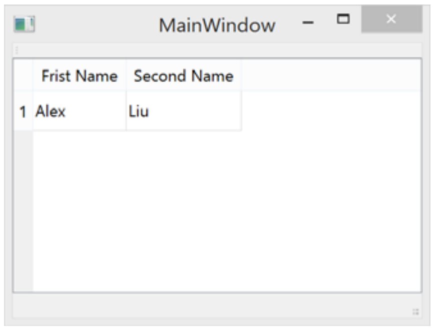
- Stack: new dialog will display and then disappear immediately

```
void MainWindow::on_actionNew_Window_triggered(){
   MyDialog Dialog;
   Dialog.show();
}
```

- Heap: new dialog will display forever until being killed
  - mainwindow.h: MyDialog *mDialog;
  - mainwindow.cpp:
    - void MainWindow::on_actionNew_Window_triggered(){
      mDialog = new MyDialog(this);
      mDialog->show();}
    - MainWindow::~MainWindow(){
      delete ui;
      delete alexDialog;}
- Default: no modal, can operate mainwindow after dialog is shown
- mDialog->setModal(true): cannot operate mainwindow after dialog is shown

QT Example 4: Table





	Frist Name	Second Name
1	Alex	Liu

MainWindow constructor:

```
ui->tableWidget->insertRow(ui->tableWidget->rowCount());
QTableWidgetItem *FirstNameCell = new QTableWidgetItem;
ui->tableWidget->setItem(ui->tableWidget->rowCount()-1,0,FirstNameCell);
FirstNameCell->setText("Alex");
QTableWidgetItem *LastNameCell = new QTableWidgetItem;
ui->tableWidget->setItem(ui->tableWidget->rowCount()-1,1,LastNameCell);
LastNameCell->setText("Liu");
ui->tableWidget->resizeColumnsToContents();
setCentralWidget(ui->tableWidget);
```