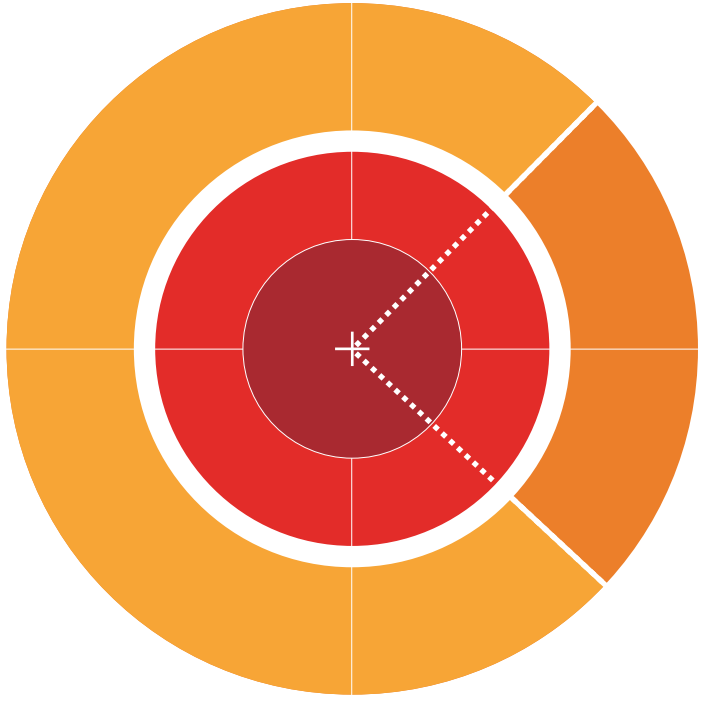
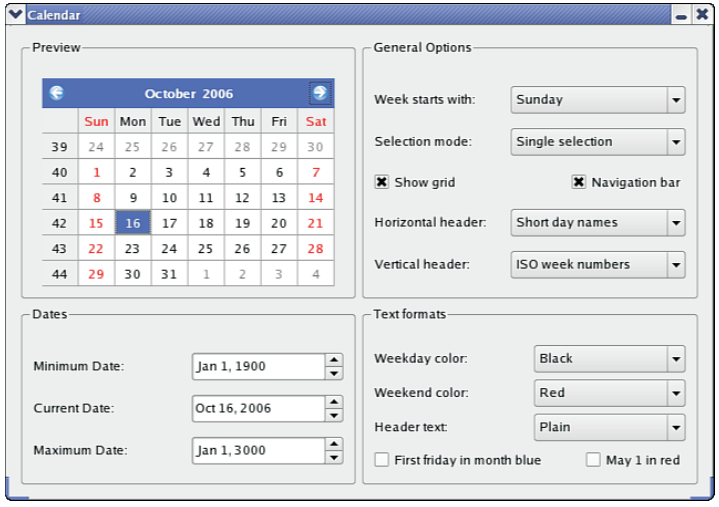


Multiplatformní GUI knihovna

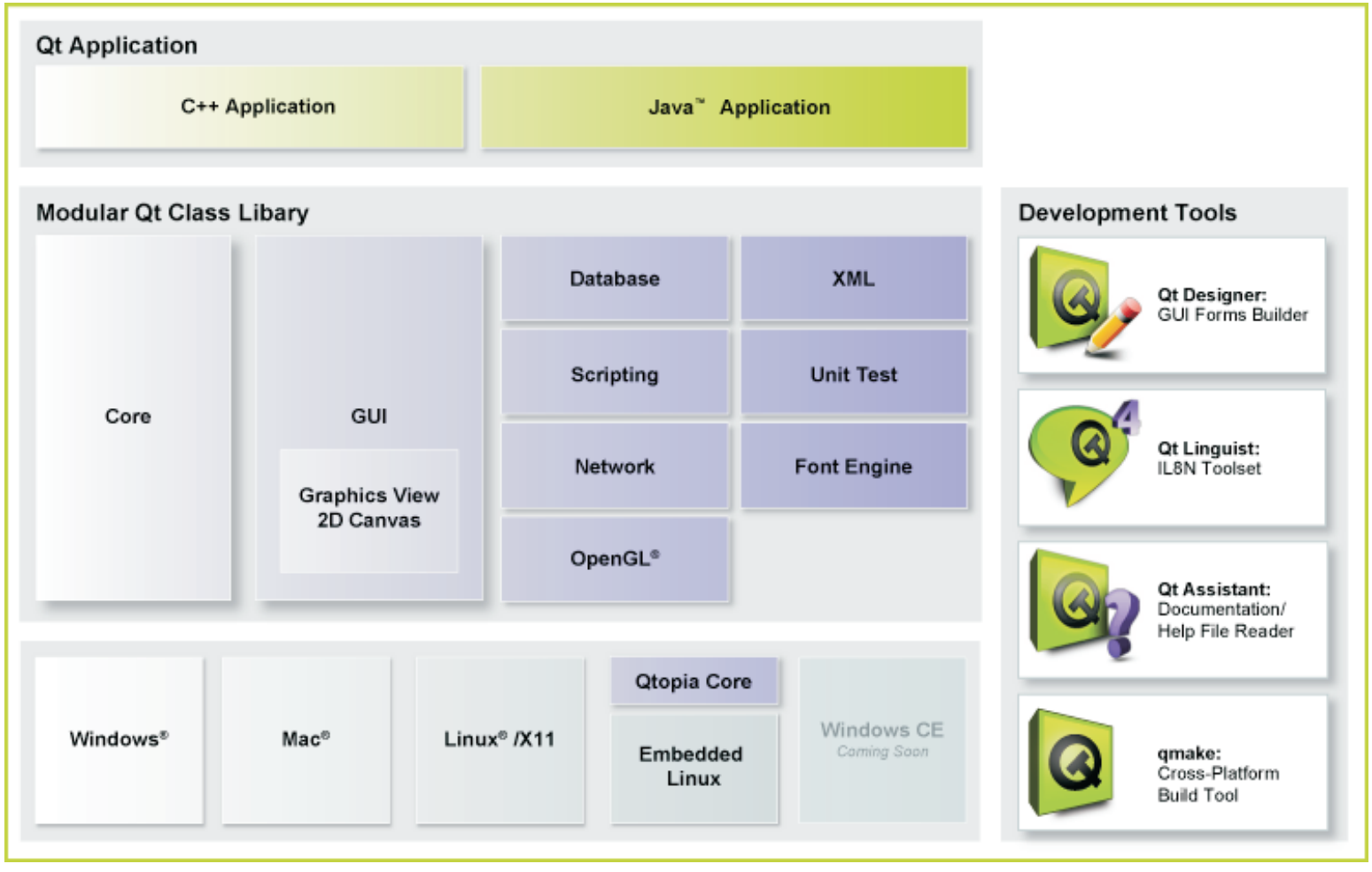


Qt = jedna ze dvou nejpopulárnějších multiplatformních knihoven na tvorbu programů s grafickým uživatelským rozhraním.

Pomocí **Qt** jsou naprogramovány např. : Adobe Photoshop Album, Skype, Google Earth, Opera, KDE a mnoho dalších.

- knihovna Qt existuje pro C++, C, C#, Python, Ruby, Perl, Pascal a Javu na Win32, UNIX/Linux, Mac OS, aj.
- podporuje lokalizaci aplikací a také SQL, zpracování XML, správu vláken a přístup k souborům

Architektura knihovny Qt



detaily na: `http://trolltech.com/products/qt/features`

Získání a instalace knihovny Qt (pro C++)

- knihovna má několik verzí - jedna z nich je k dispozici **zdarma jako Open Source pod GPL**; lze stáhnout z

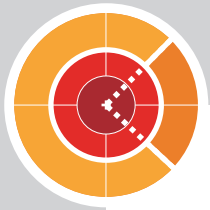
`http://trolltech.com/downloads/opensource`

- podporuje pouze překladač MinGW, je třeba jej nainstalovat (což umí udělat instalátor Qt, má-li přístup k i-netu).
- úplná instalace Qt má asi 236 MB, MinGW 95 MB
- je nutné přidat **`/qt/4.3.3/bin`** a **`/mingw/bin`** do PATH

- online dokumentace: **`http://doc.trolltech.com/`**

Pokud je vše správně nainstalováno, mělo by fungovat:

```
E:\Work\QtTest\qmake
E:\Work\QtTest\mingw32-make
```



Vytvoření projektu a překlad

- v pracovním adresáři libovolný zdrojový soubor **.cpp**, např. **hello.cpp** (obsah viz dále)

```
X:\>qmake -project
X:\>ren xx.pro hello.pro
X:\>qmake
X:\>mingw32-make
```

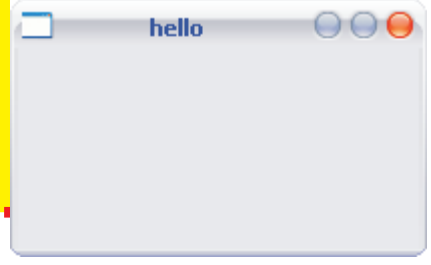
není nutné, funguje to i s **xx.pro** ;)

- v případě, že bude zdrojový kód ve více souborech, je samozřejmě nutné je průběžně doplňovat do **projektového souboru**...
- pokud se objeví při **make** chyby, je většinou problém s cestami...

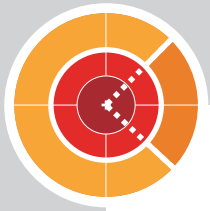
```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += hello.cpp
```

Minimální aplikace v Qt - hello.cpp

```
#include <QApplication>
#include <QMainWindow>

int main(int argc, char *argv[]) {
     QApplication app(argc, argv);
     QMainWindow mainWin;

     mainWin.show();

     return app.exec();
}
```



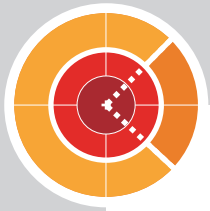
- třída **QApplication** vytváří jen kontext aplikace, není přímo vázaná na vizuální prvky, proto není s **QMainWindow** nijak spojená...



Přidávání dalších prvků

```
#include <QApplication>
#include <QMainWindow>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QMainWindow mainWin;
    QPushButton button("&Quit");

    mainWin.setWindowTitle("Qt Test App");
    mainWin.setCentralWidget(&button);

    mainWin.show();

    return app.exec();
}
```

mít vše staticky je velmi nepraktické až nepoužitelné...



Praktický způsob programování aplikace

hello.cpp

```
#include <QApplication>
#include "main.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    MainWindow mainWin;

    mainWin.show();

    return app.exec();
}
```

main.h

```
#ifndef MAIN_H
#define MAIN_H

#include <QMainWindow>

class MainWindow:
  public QMainWindow {
public:
    MainWindow();
};

#endif
```

main.cpp

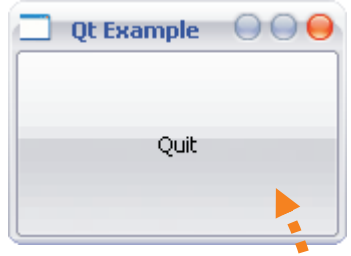
```
#include <QMainWindow>
#include "main.h"

MainWindow::MainWindow() {
    setWindowTitle("Qt Example
}
```

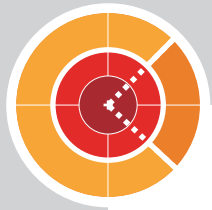
Přidávání ovládacích prvků (reálný případ)

```
#ifndef MAIN_H
#define MAIN_H
#include <QMainWindow>

class QPushButton;
class MainWindow : public QMainWindow {
    private:
        QPushButton *quitButton;
    public:
        MainWindow();
};

#endif
```

```
#include <QMainWindow>
#include <QPushButton>
#include "main.h"

MainWindow::MainWindow() {
    quitButton = new QPushButton("&Quit");
    setWindowTitle("Qt Example");
    setCentralWidget(quitButton);
}
```

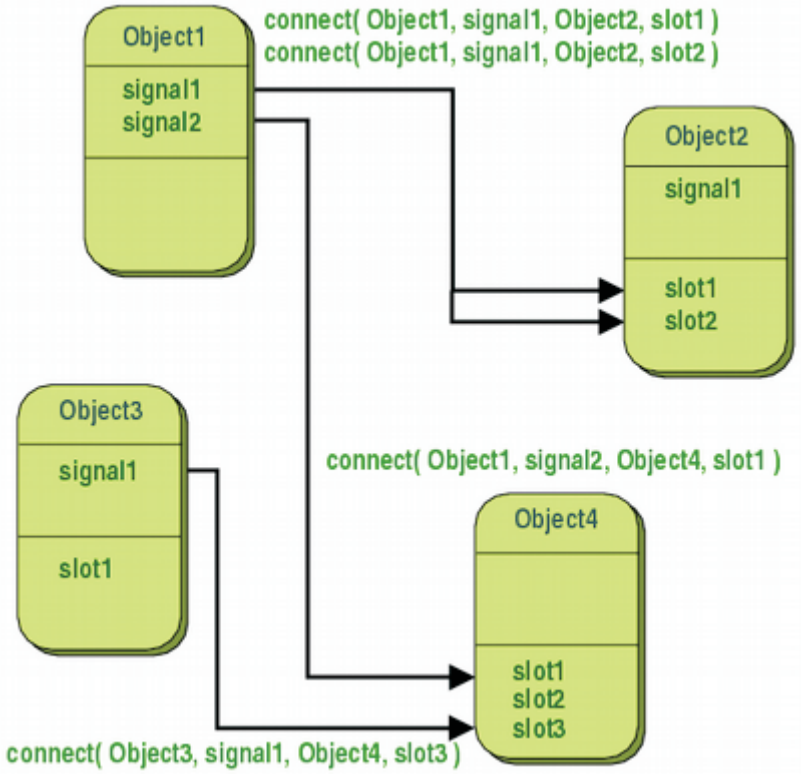
dynamicky





Jak to udělat, aby to něco dělalo: Signály a sloty

- nastane-li **událost** (stisk myšítka, klávesy, atp.), vyšle odpovědný objekt **signál**, jiný objekt může tento signál zachytit některým ze svých **slotů**; **Qt** definuje stovky signálů a slotů:



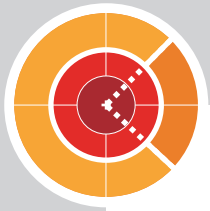
QWidget

Public Slots

```
bool close()
void hide()
void lower()
void raise()
void repaint()
void setDisabled(bool)
void setEnabled(bool)
void setFocus()
void setHidden(bool)
virtual void setVisible(bool)
void setWindowModified(bool)
void setWindowTitle(const QString &)
void show()
void showFullScreen()
void showMaximized()
void showMinimized()
void showNormal()
void update()
```

Public Signals

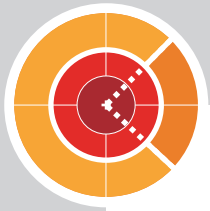
```
void customContextMenuRequested(
  const QPoint &)
void destroyed(QObject *obj = 0)
```



Programové propojení signálu se slotem

```
#include <QtGui>
#include <QMainWindow>
#include <QPushButton>
#include "main.h"

MainWindow::MainWindow() {
  quitButton = new QPushButton("&Quit");

  setWindowTitle("Qt Example");
  setCentralWidget(quitButton);

  QObject::connect(quitButton, SIGNAL(clicked()),
    qApp, SLOT(quit()));
}
```

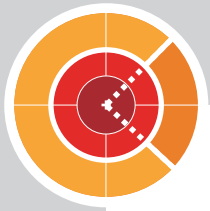
globální proměnná (z knihovny **QtGui**) typu ukazatel na aktivní instanci třídy **QApplication**, tj. právě běžící Qt-aplikaci...



Reakce aplikace na události: Vlastní sloty

```
class QPushButton;
class QLineEdit;

class MainWindow : public QMainWindow {
  Q_OBJECT
  private:
    QPushButton *quitButton;
    QPushButton *clearButton;
    QPushButton *showButton;
    QLineEdit *nameEdit;
  public:
    MainWindow();
  public slots:
    void showDialogue();
};
```

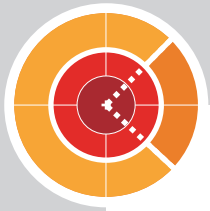
tohle pochopitelně není konstrukce C++, je třeba zavolat **moc** (Meta-Object Compiler), který to přeloží do C++; zahrneme-li jméno headeru do proměnné **HEADERS** v projektovém souboru **.pro**, zavolá ho **qmake** za nás...



Implementace slotu jako metody

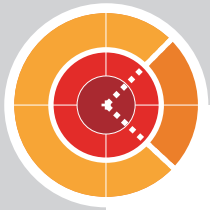
```
...
  QObject::connect(showButton, SIGNAL(clicked()),
    this, SLOT(showDialogue()));
}

void MainWindow::showDialogue() {
  QString *str = new QString("Hello ");

  str->append(nameEdit->text());
  QMessageBox::warning(this, "Greeting",
    *str, QMessageBox::Ok, QMessageBox::Ok);

  delete str;
}
```

- metodu lze normálně volat, může vracet hodnotu
- jako slot je však **vždy deklarovaná** jako **`void`**

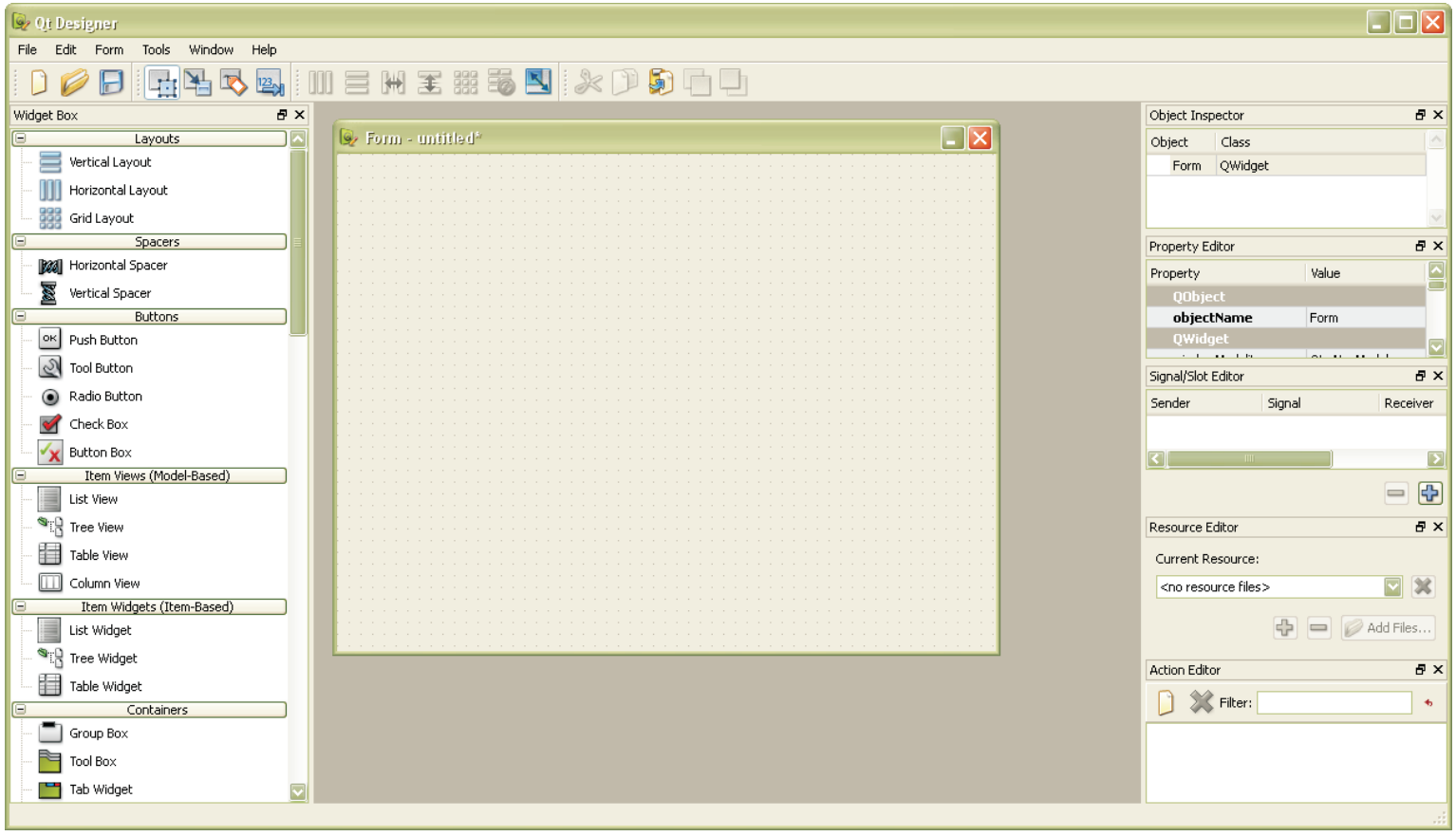


Rozmisťování prvků GUI: Skupiny a layouty

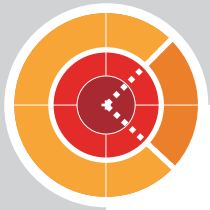
```
MainWindow::MainWindow() {
    QGridLayout *mainLayout = new QGridLayout;
    QGroupBox *group = new QGroupBox("Greeting");

    quitButton = new QPushButton("&Quit");
    showButton = new QPushButton("&Greet me...");
    clearButton = new QPushButton("&Clear");
    nameEdit = new QLineEdit();

    mainLayout->addWidget(nameEdit, 0, 0);
    mainLayout->addWidget(clearButton, 1, 0);
    mainLayout->addWidget(showButton, 1, 1);
    mainLayout->addWidget(quitButton, 1, 2);

    group->setLayout(mainLayout);

    setWindowTitle("Qt Example");
    setCentralWidget(group);
}
```



Rozmisťování prvků GUI: Qt Designer



- aplikace **Qt Designer** usnadňuje tvorbu uživatelského rozhraní, spouští se **`/qt/4.3.3/bin/designer`**



Zdroje informací, příkladů a nápověda

- aplikace **Qt Assistant** obsahuje kompletní dokumentaci celé knihovny Qt s řadou příkladů a ukázek, spouští se `/qt/4.3.3/bin/assistant`

- aplikace **Qt Examples and Demos** obsahuje ukázky, co se dá s knihovnou Qt realizovat - u každé ukázky je uveden odkaz na zdrojový kód a dokumentaci v Assistantu, spouští se `/qt/4.3.3/bin/qtdemo`