

## 9. Qt intro

B2B99PPC – Praktické programování v C/C++

Stanislav Vítek

Katedra radioelektroniky  
Fakulta elektrotechnická  
České vysoké učení v Praze

Část I

QT intro

# Co je to Qt?

---

- multiplatformní knihovna (framework) pro tvorbu aplikací
- podpora řady jazyků (C++, Python, Java, ...)
- komplexní funkcionalita (pokrývá většinu běžných IT problémů)
- součástí frameworku jsou i vývojové nástroje
- Qt Creator, Qt Designer, Qt Linguist, qmake, moc
- portována na mobilní a embedded platformy
- komerční vs. free licence
- bohatá historie, konzistence, vývoj, dokumentace, komunita

# Proč si vybrat Qt?

---

## Pro

- Jednoduchý a rychlý vývoj aplikací
- Množství knihoven
- Multiplatformní

## Proti

- Instalace a nastavení prostředí
- Distribuce programu s knihovnamí
- Komplikovanější překlad
- Některé zajímavé knihovny pouze s komerční licencí

# Čím se budeme zabývat?

---

# Hello Qt!

```
1  #include <QApplication>
2  #include <QPushButton>
4  int main(int argc, char *argv[])
5  {
6      QApplication app(argc, argv);
7      QPushButton b("Hello Qt!");
8      b.show();
9      return app.exec();
10 }
```

lec09/01-button

## QApplication ↗

- hlavní třída, kontroluje defaultní chování GUI a spravuje zdroje
- každá Qt aplikace s GUI musí mít právě jednu instanci `QApplication`
- instance vytvořena před použitím dalších prvků
- parsuje argumenty spuštění
- pomocí makra `qApp` ↗ lze získat globální pointer na aplikaci

# Hello Qt!

---

```
1  #include <QApplication>
2  #include <QPushButton>
4  int main(int argc, char *argv[])
5  {
6      QApplication app(argc, argv);
7      QPushButton b("Hello Qt!");
8      b.show();
9      return app.exec();
10 }
```

lec09/01-button

## QPushButton

- GUI prvek, na který se dá kliknout
- můžeme ho vytvořit v okamžiku, kdy už je instancionována QApplication
- vlastnosti tlačítka:
  - velikost,
  - text,
  - barva,
  - okraje

# Hello Qt!

---

```
1  #include <QApplication>
2  #include <QPushButton>
4  int main(int argc, char *argv[])
5  {
6      QApplication app(argc, argv);
7      QPushButton b("Hello Qt!");
8      b.show();
9      return app.exec();
10 }
```

lec09/01-button

## .show()

- widgety jsou defaultně neviditelné
- je třeba zavolat metodu show()
- zavolání metody show() zobrazí i potomky
- pozn. některé widgety slouží jako kontejnery pro ostatní (tj. např. určují zobrazování v určité mřížce)



# Hello Qt!

```
1  #include <QApplication>
2  #include <QPushButton>
4  int main(int argc, char *argv[])
5  {
6      QApplication app(argc, argv);
7      QPushButton b("Hello Qt!");
8      b.show();
9      return app.exec();
10 }
```

lec09/01-button

## .exec()

- funkce main() předává řízení QT frameworku
- metoda exec() neskončí, dokud není uzavřeno okno aplikace
- event-driven flow
  - objekty mají funkce / metody, které jsou automaticky volány při události (onClick(), resize(), ...)

# I. QT intro

---

Qt třídy a objekty

Datové typy

Programovací model GUI

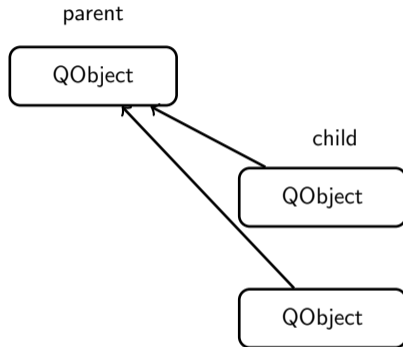
Stylování komponent

# QObject

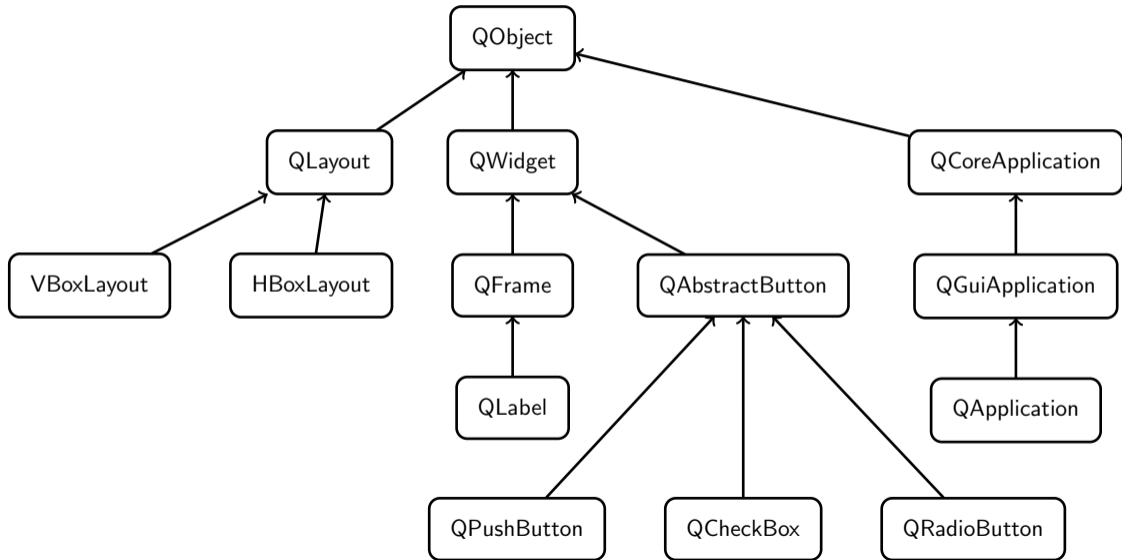
- Bázová třída téměř všech tříd v QT
- QWidget dědí z QObject (is a)
- Třídy dědicí z QObject
  - mohou využívat signal / slot mechanismus
  - dědí atributy a metody
  - memory management
  - Meta-Object Compiler (moc)

## Vztahy mezi objekty

- instance QObject se automaticky řadí do stromu
  - založeno na parent-child
  - nejedná se o dědění ve smyslu C++
- `QObject(QObject *parent = 0)`
  - Parent vloží objekt to seznamu potomků
  - Parent vlastní potomka



# Hierarchie tříd



```
1  #include <QApplication>
2  #include <QPushButton>
4  int main(int argc, char ** argv)
5  {
6      QApplication app (argc, argv);
7      // objekt může být vytvořen v zásobníku i na haldě
8      QPushButton *b = new QPushButton("&Quit");
9      // propojení signálu emitovaného objektem se slotem aplikace
10     QObject::connect(b, SIGNAL(clicked()), &app, SLOT(quit()));
11     // aktivace objektu
12     b->show();
13     //
14     return app.exec();
15 }
```

- V dynamické paměti (haldě) – QObject s rodičem
  - QTimer\* timer = new QTimer(this);
- V zásobníku – QObject bez rodiče
  - QFile,
  - QApplication
  - top-level widgety – QMainWindow
- V zásobníku – proměnné datových typů s hodnotami
  - QString,
  - QStringList,
  - QColor
- Zásobník nebo halda
  - QDialog – záleží na době života

# I. QT intro

---

Qt třídy a objekty

Datové typy

Programovací model GUI

Stylování komponent

# QVariant

---

- Union pro většinu datových typů v QT
  - uživatelské datové typy musí být registrovány v Meta-Object Systému (MOS)
- Návrátový typ nebo parametr ve funkcích, kde není znám předem datový typ

```
1   QVariant v1 (42);
2   int value = v1.toInt();    // read back
3   qDebug() << v1.typeName(); // int
5   QVariant v2 = QVariant::fromValue(QColor(Qt::red));
6   QColor color = v2.value<QColor>(); // read back
7   qDebug() << v2.typeName();      // QColor
```

lec07/03-variant/main.cpp

- Pozn.: jak na textový výstup bez `qDebug()`?

```
1   #include <QTextStream>
3   QTextStream out(stdout);
4   out << "nazdar" << endl;
```



```
1 // Contact.h
2 #include <QMetaType>
3
4 class Contact
5 {
6 public:
7     void setName(const QString & name);
8     QString name() const;
9     ...
10 };
11
12 Q_DECLARE_METATYPE(Contact);
```

lec07/04-contact/contact.h

```
1  #include <QDebug>
2  #include <QVariant>
4  #include "contact.h"
6  int main(int argc, char* argv[])
7  {
8      Contact contact;
9      contact.setName("Peter");
10     const QVariant variant = QVariant::fromValue(contact);
11     const Contact otherContact = variant.value<Contact>();
13     qDebug() << otherContact.name(); // "Peter"
14     qDebug() << variant.typeName(); // prints "Contact"
16     return 0;
17 }
```

- Konverzní konstruktor a operátor přiřazení

```
1 | QString str("abc");  
2 | str = "def";
```

- Statická funkce, zde převod z čísla

```
1 | QString n = QString::number(1234);
```

- Statická funkce, zde ukazatel na char

```
1 | QString text = QString::fromLatin1("Hello Qt");  
2 | QString text = QString::fromUtf8(inputText);  
3 | QString text = QString::fromLocal8Bit(cmdLineInput);  
4 | QString text = QStringLiteral("Literal string"); // (UTF-8)
```

- Překlad (translation – ne UTF-8)

```
1 | QString text = tr("Hello Qt");
```

- Operátory

```
1 | QString str = str1 + str2;  
2 | fileName += ".txt";
```

- Odstranění duplicitních bílých znaků

```
1 | simplified()
```

- Část textového řetězce

```
1 | left(), mid(), right()
```

- Zarovnání

```
1 | leftJustified(), rightJustified()  
2 | QString s = "apple";  
3 | QString t = s.leftJustified(8, '.'); // t == "apple..."
```

- Převod na čísla

```
1 | QString text = ...;  
2 | int value = text.toInt();  
3 | float value = text.toFloat();
```

- Převod ze stringu

```
1 | QString text = ...;  
2 | QByteArray bytes = text.toLatin1();  
3 | QByteArray bytes = text.toUtf8();  
4 | QByteArray bytes = text.toLocal8Bit();
```

## QString – formátovaný výstup

---

```
1  int i = ...;
2  int total = ...;
3  QString fileName = ...;
5  QString status = tr("Processing file %1 of %2: %3")
6      .arg(i).arg(total).arg(fileName);
8  double d = 12.34;
10 QString str = QString::fromLatin1("delta: %1").arg(d, 0, 'E', 3);
11 // str == "delta: 1.234E+01"
```

## Alternativy k STL kontejnerům – QList, QMap

---

- možno dále využívat STL kontejnery, existují metody pro konverzi Qt-STL

```
1 QList<QString> list;
2 list << "one" << "two" << "three";
3 QString item1 = list[1]; // "two"
5 for(int i=0; i<list.count(); i++) const QString &item2 = list.at(i);
7 int index = list.indexOf("two");
```

```
1 QMap<QString, int> map;
2 map["Norway"] = 5;
3 map["Italy"] = 48;
5 int value = map["France"];
7 if(map.contains("Norway")) int value2 = map.value("Norway");
```

## Alternativy k STL kontejnerům – foreach

---

### foreach (variable, container)

```
1  foreach (const QString& str, list) {  
2      if (str.isEmpty())  
3          break;  
4      qDebug() << str;  
5  }
```



# I. QT intro

---

Qt třídy a objekty

Datové typy

Programovací model GUI

Stylování komponent

# Hlavní kategorie komponent

---

- **Widget**

- prvky GUI (tlačítka, kontejnery, menu, ...)
- zpracovávají vstupy
- emitují signály, které mohou být zachytávány jinými prvky (objekty) pomocí tzv. slotů
- vykreslují grafickou podobu prvku, každý prvek může být stylizován do zcela jiné podoby

- **Layout**

- řídí rozmístění GUI prvků
- používá jednoduchá pravidla pro řazení prvků (vertikální, horizontální, ...)
- zajišťují správné škálování prvků při změně geometrie

- **Signal & Slot**

- propojuje komponenty na úrovni zasílání zpráv
- signál – emitován jako v návaznosti na nějakou událost
- slot – metoda instance třídy, která signál zpracovává

Ve skutečnosti je to daleko složitější, ale pro začátek si s tím vystačíme.

# QWidget

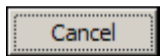
---

- Prvky GUI, které umožňují:
  - zpracovat vstup(y)
  - emitovat signál(y)
  - vykreslit grafickou podobu prvku
- Prvek může být stylizován do zcela jiné podoby
- Většina prvků emituje **signály**, které mohou být zachytávány jinými prvky (objekty) pomocí tzv. **slotů**
- Předdefinované prvky: tlačítka, kontejnery, menu, ...
- Widget bez rodiče → *okno*

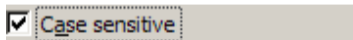
```
1 | new QWidget(0)
```

QPushButton [↗](#)

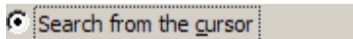
```
1 | QPushButton *button = new QPushButton("&Download", this);
```

QCheckBox [↗](#)

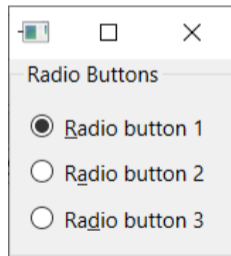
```
1 | QCheckBox *checkbox = new QCheckBox("C&ase sensitive", this);
```

QRadioButton [↗](#)

```
1 | QRadioButton *b = new QRadioButton("Search from the &cursor", this);
```



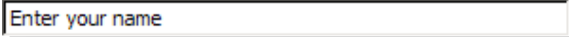
```
1  QGroupBox *groupBox = new QGroupBox(tr("Exclusive Radio Buttons"));
3  QRadioButton *radio1 = new QRadioButton(tr("&Radio button 1"));
4  QRadioButton *radio2 = new QRadioButton(tr("R&adio button 2"));
5  QRadioButton *radio3 = new QRadioButton(tr("Ra&dio button 3"));
7  radio1->setChecked(true);
9  QVBoxLayout *vbox = new QVBoxLayout;
10 vbox->addWidget(radio1);
11 vbox->addWidget(radio2);
12 vbox->addWidget(radio3);
13 vbox->addStretch(1);
14 groupBox->setLayout(vbox);
```



- Další: [QTabWidget](#), [QFrame](#), [QToolBox](#)

[QLineEdit](#) , [QTextEdit](#)


```
1 | QLineEdit *lineEdit = new QLineEdit(this);  
2 | lineEdit->setText("Enter your name");
```



Enter your name

[QDateEdit](#) , [QDateTimeEdit](#)


```
1 | QDateTimeEdit *dateEdit = new QDateTimeEdit(QDate::currentDate());
```



01-Jan-00

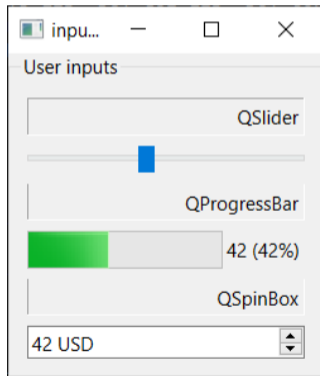
[QComboBox](#)

```
1 | QComboBox *comboBox = QComboBox (this);  
2 | comboBox->addItem("Windows style");
```



Windows style

```
1 // QSlider
2 QSlider *slider = new QSlider(Qt::Horizontal);
3 slider->setRange(0, 99);
4 slider->setValue(42);
5
6 //QProgressBar
7 QProgressBar *progress = new QProgressBar;
8 progress->setRange(0, 99);
9 progress->setValue(42);
10 progress->setFormat("%v (%p%)");
11
12 //QSpinBox
13 QSpinBox *spin = new QSpinBox;
14 spin->setRange(0, 99);
15 spin->setValue(42);
16 spin->setSuffix(" USD");
```



# Rozložení komponent – Qt Layout

---

- Popisuje, jak je GUI prvek umístěn v rámci UI
- Automaticky volí vhodnou velikost prvku
- Řeší `resize()` eventy a změny obsahu (přidávání / mizení prvků)
- Vztahy mezi komponentami:
  - widget může obsahovat jiné widgety
  - widget může obsahova rozložení
  - rozložení může obsahovat widgety
  - rozložené může obsahovat jiná rozložení
- Složitá uspořádání se řeší jako kombinace elementárních
- Horizontální (vodorovné) rozložení: `QHBoxLayout` ↗
- Vertikální (svislé) rozložení: `QVBoxLayout` ↗
- Rozložení do mřížky: `QGridLayout` ↗



## Příklad QHBoxLayout

---

```
1  #include <QApplication>
2  #include <QPushButton>
3  #include <QHBoxLayout>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication app(argc, argv);
8      QWidget *window = new QWidget;
9      QHBoxLayout *layout = new QHBoxLayout;
10     layout->addWidget(new QPushButton("One"));
11     layout->addWidget(new QPushButton("Two"));
12     layout->addWidget(new QPushButton("Three"));
13     window->setLayout(layout);
14     window->show();
15     return app.exec();
16 }
```

lec09/08-horizontal

## Příklad QGridLayout

---

```
1  #include <QApplication>
2  #include <QPushButton>
3  #include <QGridLayout>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication app(argc, argv);
8      QWidget* window = new QWidget;
9      QPushButton* one = new QPushButton("One"); // *two, *three
10     QGridLayout* layout = new QGridLayout;
11     layout->addWidget(one, 0, 0); // row:0, col:0
12     layout->addWidget(two, 0, 1); // row:0, col:1
13     layout->addWidget(three, 1, 0, 1, 2); // r:1, c:0, rSpan:1, cSpan:2
14     window->setLayout(layout)
15     return app.exec();
16 }
```

lec09/09-grid

# I. QT intro

---

Qt třídy a objekty

Datové typy

Programovací model GUI

Stylování komponent

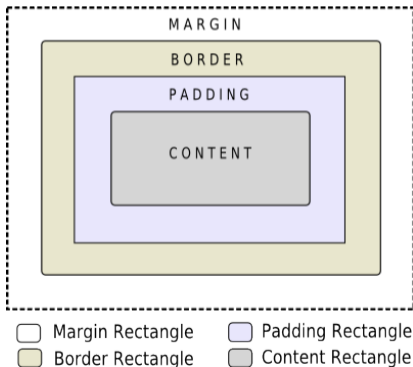
# Stylování komponent

- Mechanismus pro změnu vizuální podoby widgetů
- Doplněk k podtřídě `QStyle` ↗
- Inspirováno HTML CSS, kompletní [reference](#) ↗
- Textová specifikace stylu

## Stylování komponent – CSS pravidla

```
.selector { property : value; }
```

- Selector: specifikace widgetu
- Páry `property/value`
- Stylovatelné elementy
  - Colors, fonts, pen style, alignment.
  - Background images.
  - Position, size
  - Border, padding



## Stylování komponent – příklad

```
1 // celá aplikace QApplication::setStyleSheet(const QString & style)
2 qApp->setStyleSheet("QLineEdit {background-color: yellow}");
3 // komponenty uvnitř jiné komponenty
4 myD->setStyleSheet("QLineEdit {background-color: yellow}");
5 // pomocí ID selektoru
6 myD->setStyleSheet("QLineEdit#nameEdit {background-color: yellow}");
7 // přímo komponenta - QWidget::setStyleSheet(const QString & style)
8 nameEdit->setStyleSheet("background-color: yellow");
```

```
1 QPushButton {
2     border-width: 2px;
3     border-radius: 10px;
4     padding: 6px;
5     // ...
6 }
```

