

10. Qt – event driven programování, události, signály

B2B99PPC – Praktické programování v C/C++

Stanislav Vítek

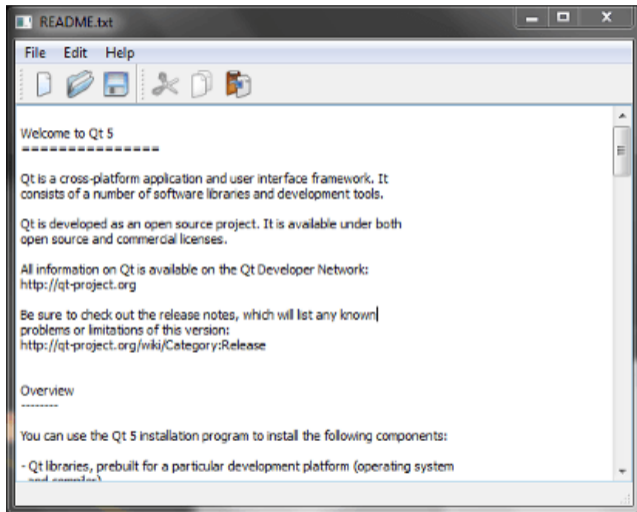
Katedra radioelektroniky
Fakulta elektrotechnická
České vysoké učení v Praze

Část I

Aplikace v Qt

Uspořádání typické GUI aplikace

- Menu bar
- Tool bar, Status bar
- Central widget
- Často má dokovací okno
- Settings (uložení stavu)
- Resources (ikony, styly, ...)
- Translation
- Load/Save documents



I. Aplikace v Qt

QMainWindow

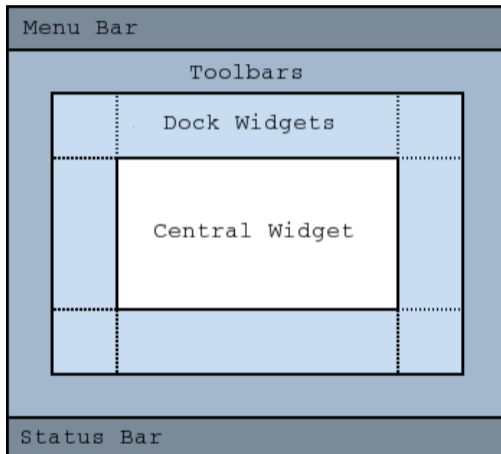
QSettings

Internacionalizace

Tvorba UI

QMainWindow

- QMainWindow
- hlavní aplikační okno
- má svůj vlastní layout
 - Central Widget
 - QMenuBar
 - QToolBar
 - QDockWidget
 - QStatusBar
- Central Widget
 - QMainWindow::setCentralWidget
 - libovolný widget



QMainWindow – menu

- QMenuBar – a horizontální menu bar
- QMenu – reprezentuje menu (top-level)
- QAction – položky přidané do QMenu

```
1 QMenuBar * bar = menuBar();
2 QMenu* menu = bar->addMenu("&File");
3 menu->addAction(action);
4 menu->addSeparator();
5 QMenu* subMenu = menu->addMenu("Sub Menu");
```

lec10/01-main-window

QMainWindow – akce

- abstraktní UI příkaz
- při spuštění emituje signál
- signál je možné ošetřit slotem (více později v této přednášce)
- použitelné u menu, toolbar, key shortcuts

```
1 QAction* action = new QAction("Open ...");
2 action->setIcon(QIcon(":/images/open.png"));
3 action->setShortcut(QKeySequence::Open);
4 action->setStatusTip("Open file");
5 connect(action, SIGNAL(triggered()), this, SLOT(onOpen()));
6 menu->addAction(action);
7 toolbar->addAction(action);
```

QMainWindow – status bar

- Horizontální bar – ideální pro prezentaci statusu aplikace
- Metody:
 - showMessage (message, timeout)
 - zobrazí dočasnou zprávu po dobu specifikovanou v ms
 - clearMessage()
 - odstraní všechny dočasné zprávy
 - addWidget() nebo addPermanentWidget()
 - normální, permanentní zpráva

```
1 QStatusBar * bar = statusBar();  
2 bar->showMessage("Ready");  
3 bar->addWidget(new QLabel("Label on StatusBar"));
```


I. Aplikace v Qt

QMainWindow

QSettings

Internacionalizace

Tvorba UI

QSettings

- Unix: INI files
- Windows: System registry
- MacOS: CFPreferences API
- hodnoty jsou uloženy jako QVariant
- hierarchie
 - použitím '/'
 - nebo beginGroup(prefix) / endGroup()
- value() očekává defaultní hodnotu

```
1 | settings.value("group/value", 68).toInt()
```

QSettings – příklad

```
1 void MainWindow::writeSettings() {
2     QSettings settings;
3     settings.setValue("MainWindow/size", size());
4     settings.setValue("MainWindow/pos", pos());
5 }
6 //--
7 void MainWindow::readSettings() {
8     QSettings settings;
9     settings.beginGroup("MainWindow");
10    resize(settings.value("size", QSize(400, 400)).toSize());
11    move(settings.value("pos", QPoint(200, 200)).toPoint());
12    settings.endGroup();
13 }
```

I. Aplikace v Qt

QMainWindow

QSettings

Internacionalizace

Tvorba UI

Co je internacionalizace

- Internacionalizace, **i18n**
 - internacionalizace je proces návrhu aplikace tak, aby bylo možné ji adaptovat s ohledem na různé jazyky a regiony bez změn kódu aplikace
- Lokalizace, **l10n**
 - lokalizace je proces adaptace aplikace pro specifický jazyk přidáním komponent s ohledem na region a předklady textu

Co všechno se lokalizuje?

- Formát data a času
- Měna, formátování čísel, formát a velikosti papíru
- Fonty
- Ikony, obrázky, obecně multimediální obsah
- Kódování textu, směr textu

Lokalizace v Qt

- Lokalizace je obecně komplikace pro vývoj, Qt komplikace minimalizuje
- Podpora Unicode (pozor, není to UTF [↗](#))
 - Je dobré používat datové typy `QChar` a `QString`
 - Je dobré naopak nepoužívat `char` a `std::string`
 - Pro převody mezi kódováními lze využít `QString::utf8()`, `QString::toUtf8()` a další
- Lokalizační třída `QTranslator` [↗](#)
- Texty určené pro lokalizaci jsou argumenty funkce `QObject::tr()` [↗](#)
- Nástroj `Qt Linguist` [↗](#) pro správu textů v UI
 - `.ts` – XML soubory, lze editovat i ručně, pokud je potřeba
 - `.qm` – binární soubory, rychlejší práce za běhu aplikace
- Workflow:
 - extrakce – `lupdate` [↗](#)
 - překlad – `linguist` [↗](#)
 - kompilace – `lrelease` [↗](#)
 - deployment – `lconvert` [↗](#)

```
1  #include <QApplication>
2  #include <QPushButton>
3  #include <QTranslator>
5  int main(int argc, char ** argv)
6  {
7      QApplication app(argc, argv);
9      QTranslator translator;
11     if (!translator.load("trans_cz"))
12     return 1;
14     app.installTranslator(&translator);
16     QPushButton b1(QPushButton::tr("Hello"));
17     b1.show();
19     return app.exec();
20 }
```

lec10/04-translation

- Výběr jazyka podle nastavení OS

```
1 | QApplication app(argc, argv);  
3 | QTranslator t;  
4 | QLocale loc = QLocale::system();  
5 | t.load (QString("app_%1").arg(loc.name()));  
6 | app.installTranslator(&t);
```

- Nalezení správné cesty (lze využít i resources)

```
1 | t.load (QString("app_%1").arg(loc.name()), QLibraryInfo::location(  
    |     QLibraryInfo::TranslationsPath);
```


I. Aplikace v Qt

QMainWindow

QSettings

Internacionalizace

Tvorba UI

Qt Designer

- Qt Designer generuje popis UI v XML formátu
- Tento popis je automaticky během překladau přeložen pomocí `uic`
- Výsledkem překladau je třída, která se použije v hlavní programu

```
1  #include "ui_formular.h"
3  int main(int argc, char ** argv)
4  {
5      QApplication app(argc, argv);
7      QWidget widget;
8      Ui_Form ui;
9      ui.setupUi(&widget);
11     widget.show();
13     return app.exec();
14 }
```

[lec10/designer-ui](#)

Část II

Event driven programování v Qt

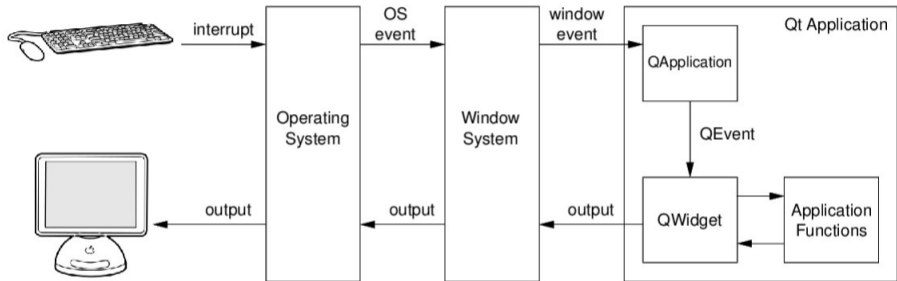
II. Event driven programování v Qt

Události

Signály

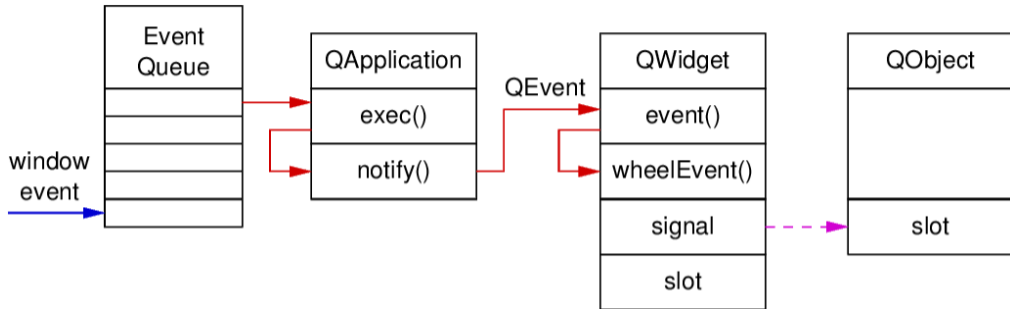
Události v Qt

- Události jsou základem komunikace Qt aplikace s prostředím, ve kterém běží



- Události jsou založeny na třídě `QEvent` [↗](#), např. `QKeyEvent` [↗](#) nebo `QTimerEvent` [↗](#)
- Existují dva hlavní způsoby posílání událostí
 - přímé posílání komponentám pomocí `sendEvent` [↗](#)
 - posílání přes frontu událostí pomocí `postEvent` [↗](#), fronta je zpracována `QEventLoop` [↗](#)

Event loop



- `QCoreApplication::exec()` spustí event loop
 - Vybírá z fronty události generované nativními okny
 - Překládá události na instance `QEvent` (nebo její potomky)
 - Posílá `QEvent` instancím `QObject` voláním `QObject::event()`
 - `QCoreApplication` pro aplikace bez GUI (→ `QGuiApplication` → `QApplication`)
- `QObject::event()`
 - Hlavní funkce pro zpracování událostí, přeposílá události specializovaným handlerům

Zpracování událostí

1. Vlastní implementace specifickýh handlerů (funkcí) pro zpracování událostí
 - Tím se změní jejich chování
2. Vlastní implementace hlavního handleru `QObject::event()`
 - Zachytávání událostí před tím, než jsou poslány specifickým handlerům
3. Použití filtru události instance `QObject`
 - Události určená pro objekt jsou nejprve poslány filtru
 - `QObject::installEventFilter` ↗
4. Použití filtru události instance `QApplication`
 - Všechny události pro všechny objekty projdou tímto filtrem
 - `QCoreApplication::installNativeEventFilter` ↗
5. Vlastní implementace funkce `QCoreApplication::notify`
 - Lze zachytit všechny události ještě předtím, než jsou poslány do filtrů

Handlery událostí

```
1  bool MyWidget::event (QEvent * event)
2  {
3      /* https://doc.qt.io/qt-6/qevent.html#Type-enum */
4      if (event->type() == QEvent::Paint)
5          qDebug() << event;
6      return QWidget::event(event);
7  }
9  void MyWidget::closeEvent (QCloseEvent * event)
10 {
11     if (okToContinue())
12         event->accept();
13     else
14         event->ignore();
15 }
```

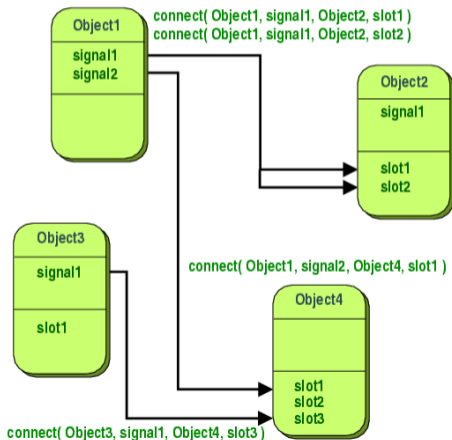

II. Event driven programování v Qt

Události

Signály

Signály a sloty

- Zatímco události jsou instance `QEvent`, signály jsou podobné systému **callbacků**
 - při použití callbacku je třeba dát vědět příjemci
 - signál stačí
- signál – informace o eventu
 - speciální druh funkce
 - `QPushButton::clicked()`
 - `QTimer::timeout()`
- slot – funkce, která ošetří signál
 - normální členská funkce třídy
 - `QTimer::start()`
 - `QTimer::stop()`x;
- programátor definuje propojení
- obecně N-to-M relace



Signály – varianty propojení

- Qt4 styl (stále funkční)

```
1 | QSlider s1;  
2 | QSpinBox s2;  
3 | connect (s1, SIGNAL(valueChanged(int)), s2, SLOT(setValue(int)));
```

- Ukazatele na funkce

```
1 | connect (s1, &QSlider::valueChanged, s2, &QSpinBox::setValue);
```

- Nečlenské funkce

```
1 | static void printValue(int value) {  
2 |     //...  
3 | }  
4 | connect (s1, &QSignal::valueChanged, &printValue);
```

Signály a sloty ve vlastní třídě

```
1  class Counter : public QObject
2  {
3      Q_OBJECT
4
5      int m_value;
6
7  public slots:
8      void setValue (int value) {
9          if (value != m_value) {
10             m_value;
11             emit valueChanged (value);
12         }
13     }
14
15  signals:
16      void valueChanged (int value);
17  };
```