PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Učebnice o knihovně Qt

2013

Martin Rotter

Vedoucí práce: Mgr. Tomáš Kühr

Studijní obor: Aplikovaná informatika

Anotace

Qt patří mezi nejkomplexnější multiplatformní knihovny, a proto je široce využíváno. Je velmi dobře dokumentováno a existuje pro něj hned několik učebnic. Drtivá většina těchto publikací uchopuje Qt z pohledu uživatelských rozhraní, což je sice relativně zajímavé, nicméně to znemožňuje lepší pochopení principů Qt, které bývají často opomíjeny. V dostupných knižních materiálech o Qt rovněž nenajdeme popis tvorby libovolné komplexnější aplikace, jenž by čtenáře provedl od prvotních idejí autora softwaru až po finální nasazení hotového produktu. V této práci je popsán pokus o vytvoření učebnice knihovny Qt s důrazem na principy, důležité vlastnosti a proces tvorby aplikací.

Poděkování náleží vedoucímu této bakalářské práce panu Mgr. Tomáši Kührovi za vstřícnost a pohotové postřehy. Rád bych také poděkoval svým nejbližším za neutuchající podporu a sdílení mého (leckdy živelného) entusiasmu.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

1 Úvod

Qt je rozsáhlá kolekce knihoven, která nabízí velmi rozmanitou funkcionalitu. To vede k vysoké penetraci Qt do mnoha oblastí, počínaje mobilními zařízeními a konče stolními počítači. Qt se využívá jak pro veliké balíky aplikací, tak pro drobné jednoúčelové nástroje. Získává si široké publikum pro svou jednoduchost, robustnost a velmi důmyslný návrh struktury komponent.

Qt jako takové je velmi dobře dokumentováno [1, 2]. Existuje pro něj i několik učebnic, které se mu obvykle věnují z pohledu uživatelských rozhraní a dalších důležitých součástí, mezi které patří podpora databázových systémů, síťování nebo například 3D grafika.

K dispozici jsou různé tutoriály a doporučené postupy, které lze najít v oficiální dokumentaci projektu. Nicméně chybí text, který by vysvětloval principy a smysl základních stavebních kamenů knihovny Qt jakožto celku, který je třeba dobře pochopit, aby se dal efektivně a správně používat. Žádná z obecně známých učebnic, mezi něž patří [3, 4, 5], nepopisuje fundamentální principy knihovny Qt a pokud ano, tak se jimi zabývá velmi okrajově a abstraktně. V uvedených učebnicích chybí hlubší technický (případně filosofický) pohled na věc. Pořizovací náklady těchto učebnic jsou relativně vysoké. Proto by přišla vhod existence textu, který by byl dostupný zdarma.

Drtivá většina učebnic představuje pásmo faktů a menších ukázkových příkladů, jež ilustrují práci s jednotlivými komponentami knihovny Qt. Žádná učebnice čtenáři nenabízí možnost pozorovat vývoj větší aplikace od začátku (tedy od počátečních myšlenek autora aplikace) až po její dokončení, publikaci či údržbu. Přitom právě na vývoji ukázkové aplikace lze lépe pochopit souvislosti mezi jednotlivým částmi knihovny Qt.

1.1 Cíle práce

V rámci této bakalářské práce bylo vytyčeno hned několik cílů, které hrají určitou roli při pochopení knihovny Qt uživatelem, který ji předtím nepoužíval nebo ji používal jen velmi zřídka. Mezi hlavní cíle spjaté s touto bakalářskou prací patří:

1. Poskytnout úvod do problematiky knihovny Qt.

2. Napsat publikaci vhodných měřítek, která pojednává o knihovně Qt. Napsaný text by měl výstižně a logicky správně popisovat jednotlivé principy knihovny Qt a tím čtenáři poskytnout kvalitní základ pro její pochopení a správné použití.

3. Naprogramovat větší ukázkovou aplikaci nad knihovnou Qt a popsat některé pasáže postupu vytváření takové aplikace ve zmíněné publikaci.

4. Popsat postupy, které se využívají při vytváření, publikování a údržbě aplikací s přihlédnutím ke knihovně Qt, k otevřenému softwaru a k zásadám, typickým pro každý operační systém.

2 Knihovna Qt

Před dalším postupem je rozumné představit si hlavního aktéra této bakalářské práce – knihovnu Qt. Qt je tu s námi od roku 1995, kdy byla po dvou letech privátního vývoje vydána první veřejná verze 1.0. [4, str. 16] Nutno dodat, že ačkoliv Qt označuji jako „knihovnu“, tak představuje celý softwarový balík, ve kterém je mimo jiné zahrnuto následující:

- sada knihoven pro různé účely,
- kolekce překladačů (meta-objektový překladač a interpret pro QtScript, postavený na bázi JavaScriptu),
- nástroje pro lokalizaci,
- aparáty pro grafické uživatelské rozhraní[1], jeho tvorbu i testování,
- vývojové prostředí Qt Creator,
- sada doplňkových skriptů pro různé sestavovací systémy (například pro CMake [6] či pro pkg-config).

Qt pravděpodobně používá téměř každý uživatel stolního počítače, aniž by o tom věděl. Řadí se totiž mezi nerozšířenější a nejvíce využívané multiplatformní knihovny a používá jej mnoho obecně známých aplikací, mezi něž patří:

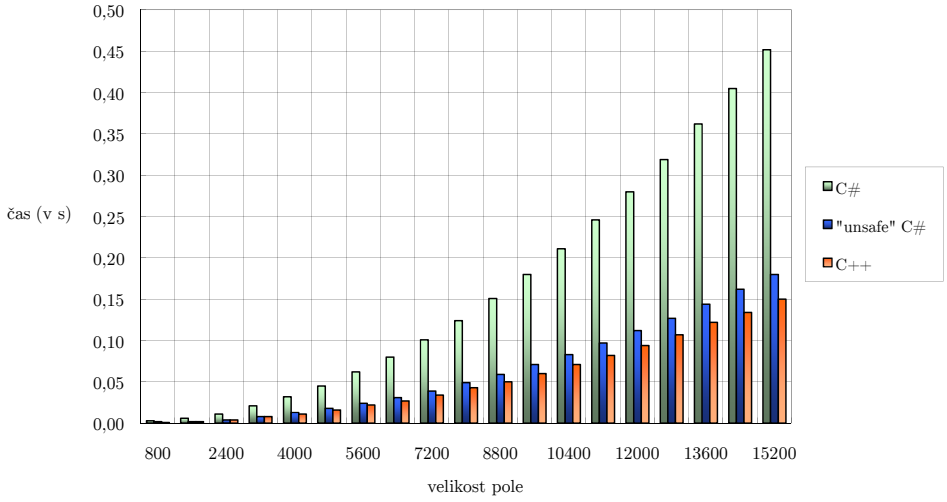
- pracovní prostředí K Desktop Environment (KDE) [7] nebo razor-qt,
- matematický softwarový balík Mathematica,
- aplikace pro internetovou komunikaci Skype,
- nástroj pro sdílení souborů Transmission.

Knihovna Qt byla od počátku vyvíjená pod licencí Q Public License [8]. Po různých peripetiích došlo ke změně licence na GNU Lesser General Public License (GNU LGPL) [9]. To se stalo především z jednoho důvodu – licence Q Public License nebyla kompatibilní s GNU General Public License (GNU GPL) [10], což se ukázalo být velikou překážkou ve vývoji svobodného softwaru založeného na Qt. Qt je mimo GNU LGPL dostupné také v komerčních licencích, za které musí koncový zákazník zaplatit. Odměnou mu je kvalitnější podpora ze strany výrobce. Majitel komerční licence pro Qt navíc může linkovat svou aplikaci vůči Qt staticky, viz [11, kapitola Compilers, linkers, assemblers, ...].

[1] Původní anglický výraz pro grafické uživatelské rozhraní je Graphical User Interface (GUI).

2.1 Podporované programovací prostředky

Qt bylo od počátku vyvíjeno v programovacím jazyku C++. Tento programovací nástroj má mnoho přívrženců, ale i odpůrců. O jeho výhodách a nevýhodách lze diskutovat. Autoři zvolili tak, jak zvolili. C++ zůstává hlavním programovacím prostředkem pro vývoj Qt i dnes, kdy se do popředí dostávají jiné jazyky, například jazyky deklarativního typu. Obecně největší výhodou jazyka C++ je rychlost vykonávání kódu. Uvažujme jednoduché srovnání rychlosti (obrázek 1[2]) vzhledem k jazyku C# založené na třídění metodou QuickSort. Vidíme, že i ve vykonávání elementárního kódu mohou vzniknout podstatné časové rozdíly. Představme si, jaké deficity se mohou objevit při komplexnějších výpočtech. Více o uvedeném srovnání nabízí [11, kap. C plus plus as base stone].



Obrázek 1: Srovnání rychlosti jazyků C++ a C#

Vzhledem k popularitě Qt na sebe nenechaly dlouho čekat takzvaná *bindings*, tedy navázání určitého vysokoúrovňového jazyka na jazyk C++. Velmi zjednodušeně lze říct, že takové navázání určitým způsobem převádí volání z jednoho programovacího jazyka do druhého. Prostřednictvím navázání je Qt k dispozici například v těchto jazycích:

- C#,
- Java,
- Python,
- Common Lisp,

[2] „Unsafe“ C# představuje aplikaci, jejíž zdrojový kód byl přeložen s dodatečnými optimalizacemi, mezi které patří například možnost použít ukazatele pro zrychlení práce s proměnnými.

- Scheme.

Je známo, že jazyk Common Lisp nedisponuje prostředky pro modelování skutečně unifikovaných GUI. Pomocí Qt lze tento nedostatek efektivně a zároveň efektně řešit na všech podporovaných platformách.

Nutno dodat, že kvalita některých navázání není vysoká, nicméně ta nejvíce používaná jsou zpracována na vysoké úrovni.

2.2 Podporované platformy a operační systémy

Jelikož je Qt svobodným softwarem, je jeho rozšiřování na nejpoužívanější platformy rychlejší, než je tomu u uzavřených aplikací, protože se na portování podílí více programátorů. V důsledku toho dnes máme oficiálně podporované takřka všechny majoritní platformy. Rozhodneme-li se vyvíjet Qt aplikaci, pak máme jistotu, že bude fungovat na nepoužívanějších operačních systémech. Mezi oficiálně podporované operační systémy patři:

- operační systémy rodiny Windows (od verze XP výše),
- GNU/Linux,
- Mac OS X,
- Android (a další mobilní platformy).

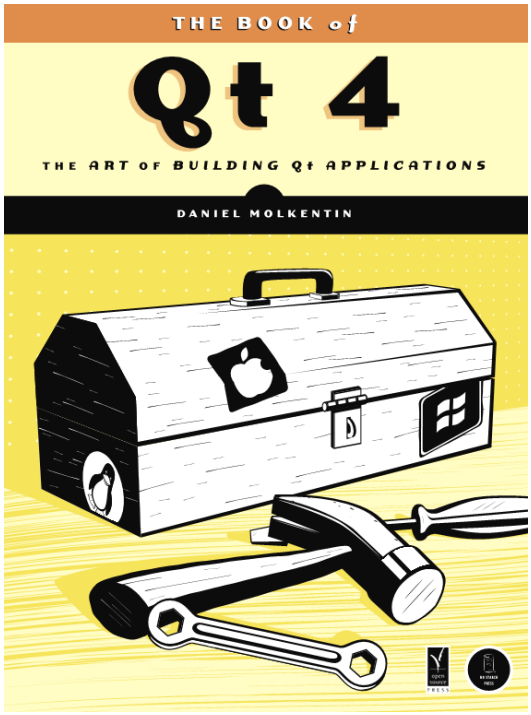
Podpora operačního systému Android je zajištěna projektem Necessitas [12]. Tento projekt je zajímavou nadstavbou nad Qt, Android SDK [13] a Android NDK [14], která využívá dvoufázovou kompilaci:

1. Nejdříve je zkompilováno jádro aplikace, které je psané v C++.
2. Zkompilovaný objektový kód je následně doplněn řídícím bajtkódem, který je generován z kódu v programovacím jazyce Java, a vše je zabaleno do finálního archivu s příponou *apk*, se kterým pracuje telefon. Telefon tento archiv použije jako instalační balíček. Ten je během instalace rozbalen do jeho paměti.

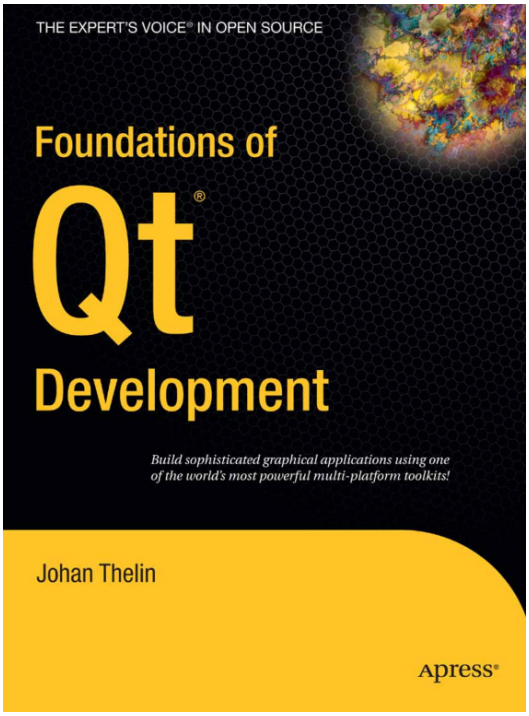
Neoficiální cestou jsou podporovány i další operační systémy, mezi které patří například iOS nebo eComStation, což je následník známého operačního systému OS/2.

3 Učebnice o knihovně Qt

Jelikož je Qt oblíbenou a široce používanou knihovnou, tak se očekává kvalitní dokumentace a široká paleta učebnic a návodů. Qt skutečně disponuje dobrou dokumentací [1], která popisuje takřka všechny jeho fragmenty.



(a) *The Book of Qt 4*



(b) *Foundations of Qt Development*

Obrázek 2: Přebaly dvou oblíbených knih o knihovně Qt

3.1 Dostupná literatura

Jak již bylo zmíněno, tak Qt nabízí docela rozsáhlou oficiální dokumentaci. Kromě toho existuje několik učebnic, které se zabývají knihovnou Qt a souvisejícími oblastmi. Mezi neznámější knihy patří:

- *The Book of Qt 4*, autor MOLKENTIN [4],
- *Foundations of Qt Development*, autor THELIN [5].

Tyto dvě učebnice (viz obrázek 2) představují klasickou šablonu pro učení Qt prostým naučením se práce s jeho komponentami. Například programátora 3D her bude zajímat jen ta část učebnic, která popisuje práci s 3D (případně s 2D) grafikou, databázový specialista bude hledat podporu pro databázové systémy a tak podobně. Tento přístup je vhodný, pokud je daný cílový uživatel s Qt obeznámen a někdy jej používal.

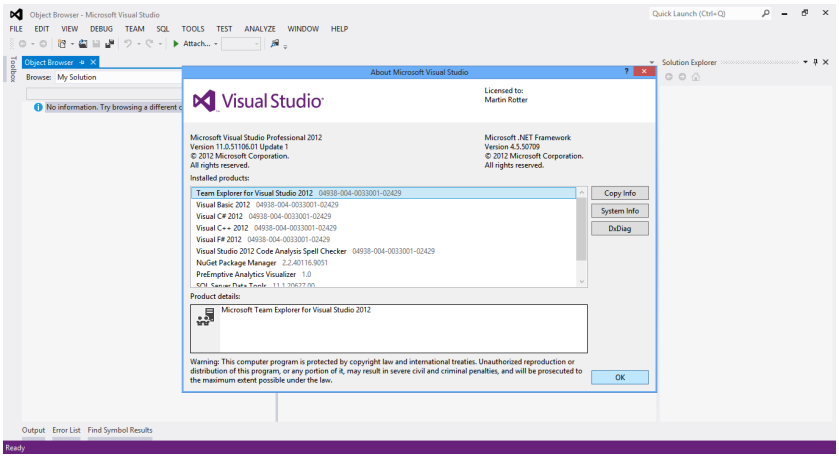
Předchozí přístup naopak nemusí být vhodný pro uživatele, který Qt předtím nikdy nepoužíval. Takový uživatel se bude spíše zajímat o pochopení struktury knihovny a jejích principů, které vedou k psaní kvalitních programů. Cílem jeho zájmu bude pochopit, proč byly určité segmenty knihovny navrhnuty tím nebo onakým způsobem. Pro takového uživatele nemá valný smysl komponovat návody na používání jednotlivých tříd do knižní publikace, protože většina informací o třídách a procedurách, které jsou obsažené v Qt, je k nalezení v oficiální dokumentaci. Rozumný uživatel raději sáhne po dokumentaci, kde nalezne pravdivé

a nezkreslené informace o té dané třídě či proceduře, a v knize bude hledat postřehy někoho, kdo už aplikaci v Qt psal a setkal se s některými problémy.

Pokud uživatel produkuje svobodný software, tak mu v tom může pomoci komunita. Je-li software využívající Qt napsán systematicky správně, tak je vyšší šance, že jej bude vyvíjet více osob, což může vést k vyšší kvalitě výsledného produktu, jak tvrdí [15].

3.2 Důvody vzniku další učebnice

Vyšel jsem z úvah, jež nastiňuje kapitola 3.1. Při psaní softwaru založeném na Qt jsem míval nutkání zjišťovat „doporučené“ postupy pro použití jednotlivých tříd a hledal jsem šablony pro typický design Qt aplikací. V Qt platí, že jedna věc jde provést hned několika způsoby, což se projevuje například u GUI.



Obrázek 3: Modální dialog aplikace Visual Studio 2012

Uvažujme modální dialog. Typický modální dialog, tak jak vypadá na operačních systémech rodiny Windows, ukazuje obrázek 3. Hlavní potíž tkví v tom, že modální dialogy se mají chovat na každé platformě jinak. Qt některé nuance řeší samo. Berme v potaz například vzhled modálních dialogů. Je obvyklé, že ve Windows nelze měnit velikost těchto dialogů, avšak v prostředí KDE i v jiných prostředích ano. Jak tedy dialogy s fixní velikostí naimplementovat tak, aby byl kód dobře platformově přenositelný?

Neznalý uživatel by udělal první věc, která jej napadne, to jest nastavil by velikost okna na fixní hodnotu (zdrojový kód 1), pokud by aktuálním operačním systémem byly Windows. Avšak v dokumentaci lze nalézt i jiný, a dle mého lepší, způsob, tak jak jej ukazuje zdrojový kód 2. Uživatel by tento postup mohl lehce přehlédnout, stejně tak jako desítky dalších drobných informací v dokumentaci.

Díky své mohutnosti je dokumentace poněkud roztříštěná, a proto je dobré některé vlastnosti Qt shrnout do jednoho dokumentu.

```
#if defined(Q_OS_WIN32)
QDialog dialog;
dialog.setFixedSize(size());
#endif
```

Zdrojový kód 1: Nevhodné nastavení fixní velikosti pro dialog na operačním systému Windows

```
QDialog dialog;
dialog.setWindowFlags(
  Qt::MSWindowsFixedSizeDialogHint |
  Qt::Dialog
);
```

Zdrojový kód 2: Správné nastavení fixní velikosti pro dialog na operačním systému Windows

3.2.1 Ekonomická specifika

Existuje hned několik učebnic [3, 4, 5], které se zabývají knihovnou Qt. Pro tyto učebnice platí, že nejsou k dispozici zdarma a mnohdy se jedná o finančně náročnou záležitost. Drtivá většina uživatelů si přitom vystačí s „průvodcem“ knihovnou Qt a s oficiální dokumentací. Mým záměrem je, aby má publikace suplovala roli onoho průvodce a potenciální čtenář si tak vystačil se zdarma dostupným duem dokumentace & učebnice.

3.3 Struktura učebnice

Moje učebnice je poněkud specifická, protože neobsahuje kompletní souhrn informací o všemožných komponentách Qt. Nelze ji tedy rozdělit do oblastí podle diskutované komponenty, například „Databáze a Qt“, „Uživatelská rozhraní“ a tak podobně.

Nastalo tedy dilema, jak učebnici strukturovat, protože členění jednotlivých částí obecně jakéhokoliv dokumentu je důležité. Čtenář učebního textu obvykle hledá:

- určitou nepříliš rozšířenou informaci teoretického typu,
- praktický postup realizace programovacího problému.

Je tedy nanejvýš vhodné rozdělit učebnici do dvou hlavních částí:

TEORETICKÉ QT
: Tato část se zabývá teorií, která souvisí s knihovnou Qt. Řeší kompilační proces Qt aplikací, strukturu knihovny jako takové nebo některé hlavní mechanismy. Mezi tyto mechanismy patří například meta-objektový systém.

PRAKTICKÉ QT

Tato část staví na první části a ukazuje čtenáři, jak vyprodukovat komplexnější aplikaci pomocí nástrojů poskytnutých knihovnou Qt. Uživatel nahlédne do procesu programování ukázkové aplikace a následně bude seznámen s nástroji, které se používají pro poloautomatické sestavování aplikace na klientských počítačích. Uživatel zde rovněž najde doporučení ohledně správy zdrojových kódů aplikace nebo způsoby, jak balíčkovat software v nejpoužívanějších operačních systémech. Tato část učebnice je doplněna ukázkovou aplikací, kterou popisuje kapitola 4. Vyprodukovaný software je zcela otevřený a očekává se, že čtenář bude fragmenty zdrojového kódu používat pro své vlastní potřeby.

3.3.1 Forma a sazba

Učebnice je situována do samostatného dokumentu [11] a je vysázena pomocí typografického systému LaTeX. Vzhledem k existenci potenciálně větší cílové skupiny byl jako jazyk dokumentu zvolen jazyk anglický.

3.3.2 Dostupnost dokumentu

Jedním z hlavních záměrů je poskytnout dokument, který je zdarma a je otevřeným softwarem. Uživatelé mohou tento dokument volně šířit.

3.4 Další informace o učebnici

Pro další informace o učebnici odkazuji k publikaci [11] samotné. Lze v ní nalézt všechny ideje a pohnutky, které stály za jejím vytvořením a další související informace, jež jsem se zde rozhodl neduplikovat.

4 Aplikace Qonverter

Druhá část učebnice [11, část Real-world Qt] je doplněna ukázkovou aplikací. Tato aplikace není ani triviální ani příliš komplexní, což je záměr. Čtenář učebnice tak v aplikaci najde zajímavé fragmenty kódu a zároveň nebude odrazen jeho celkovým množstvím.

Jako ukázková aplikace byla zvolena kalkulačka. V segmentu softwarových kalkulaček je určitá konkurence, v této oblasti je tedy malá šance získat majoritní postavení. Nicméně kalkulačka s některými specifickými funkcemi na trhu chybí.

Kalkulačka není typem aplikace, který by vyžadoval vysoce komplexní přístup a mnoho času na vývoj a zároveň se nejedná o primitivní software, který by nebyl užitečný. Vyjděme z předinstalovaných kalkulaček na jednotlivých operačních systémech:

1. Operační systémy rodiny Windows obsahují kalkulačku, která je jednoduchá a velmi dobře plní základní funkce, mezi které patří elementární matematické funkce nebo práce s pamětí.

2. Rovněž desktopové prostředí KDE v operačním systému GNU/Linux nabízí kalkulačku. Jmenuje se KCalc [16]. KCalc nabízí pokročilou práci s konstantami a podobnou množinu funkcí jako kalkulačka z Windows. Není od věci říci, že KCalc disponuje poněkud nezvyklým GUI (obrázek 4).

Obrázek 4: Vzhled hlavního okna aplikace KCalc [16]

Jak vidíme, tak výchozí kalkulačky u některých operačních systémů resp. desktopových prostředí poskytují spíše základní funkce a některým uživatelům budou dostačovat. Avšak někteří uživatelé hledají pokročilou funkcionalitu, která zahrnuje například operace s komplexními čísly, práci s řetězci nebo možnost použít komparační operátory.

4.1 Základní přehled funkcí

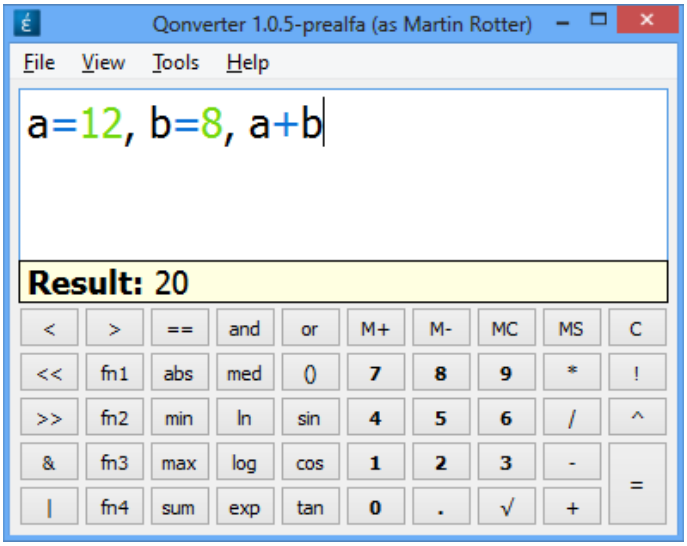
Moje kalkulačka disponuje standardní sadou funkcí, která se od kalkulačky přirozeně očekává:

- základní matematické operace jako $+, -, \times, /$,
- práce s desetinnými čísly,
- trigonometrické a další elementární funkce,
- paměť pro výsledek výpočtu,
- vzhledově neutrální GUI.

Qonverter navíc nabízí paletu funkcí, které nejsou až tak obvyklé:

- podpora pokročilých funkci s proměnným počtem argumentů (medián, průměr, maximum, minimum),
- komparační operátory,
- neomezený počet pojmenovaných proměnných a předdefinované konstanty,
- dvouúrovňový systém výsledků předchozích výpočtů,
- kaskádové výpočty (obrázek 5),
- automatické oznamování mezivýsledků,
- uživatelské vzhledy,
- integrace do systémového panelu,
- převodník jednotek s bohatou databází veličin,
- převodník měn s možností on-line aktualizace kurzů z Evropské centrální banky.

Podrobnější popis některých funkcí nabídne kapitola 4.3 a také [11].



Obrázek 5: Kaskádové výpočty

4.2 Programátorská dokumentace

Aplikace Qonverter se co do velikosti řadí ke spíše menším projektům. Zdrojový kód aplikace sestává z několika tisíců řádků a bylo definováno několik desítek tříd. Drtivá většina zdrojového kódu je napsána v programovacím jazyce C++. Byl zvolen právě tento jazyk, neboť je v něm napsáno i samotné Qt.

4.2.1 Struktura projektu

Jak již bylo zmíněno, Qonverter obsahuje desítky tříd, což znamená, že projekt sám o sobě zahrnuje desítky až stovku souborů, protože jedna třída je zpravidla rozdělena mezi hlavičkový soubor s koncovkou „h“ a zdrojový soubor s koncovkou „cpp“. Kořenový adresář projektu Qonverter obsahuje tyto elementy:

soubor CMakeLists.txt
: Jedná se o klíčový soubor projektu, který obsahuje několik skriptů, které se postarají o překlad aplikace a její následné balíčkování. Obsaženy jsou také skripty pro generování překladů a instalaci aplikace.

adresář .git
: Obsahuje nastavení repozitáře pro Qonverter a další informace potřebné pro korektní chod verzovacího systému Git [17].

adresář localization
: Tento adresář obsahuje zdrojové soubory pro překlad aplikace a knihovny Qt do jiných jazyků. Lokalizace pro knihovnu Qt nemusí být zahrnuta, protože bývá součástí instalace Qt. Avšak na některých operačních systémech se vyskytují problémy s načtením předinstalovaných lokalizačních souborů, a proto Qonverter spoléhá na vlastní soubor.

adresář resources
: V tomto adresáři lze nalézt záložní sadu grafiky, kterou používá Qonverter, pokud se nepodaří načíst systémové ikony. Dále se zde nachází výchozí vzhledy aplikace, skript pro inicializaci databáze a další podpůrné soubory.

adresář src
: Zde najdeme samotné zdrojové kódy aplikace, včetně všech kompilačních závislostí, mezi které patří knihovna muParserX (viz kapitola 4.2.2). Jednotlivé komponenty aplikace jsou strukturovány do podadresářů.

adresář ui
: Tento adresář zahrnuje soubory grafických návrhů některých dialogů programu Qonverter. Tyto soubory dokáže otevřít vývojové prostředí Qt Creator nebo jednoúčelový nástroj pro návrh GUI – Qt Designer.

4.2.2 Knihovna muParserX

Aplikace Qonverter sestává z několika částí, které tvoří jednolitý celek. Výpočetní jádro je formováno knihovnou muParserX [18], což je otevřený projekt, který si klade za cíl vyprodukovat dobře použitelnou knihovnu pro výpočty matematických formulí. Knihovna se mi velice líbila, ale obsahovala několik velmi nepříjemných vlastností a nedostatků. Proto jsem kontaktoval jejího autora a přidal se k vývoji. Více o tomto obsahuje [11, kap. muParserX library].

4.2.3 Obalení knihovny muParserX

Jedním z výchozích kroků bylo obalení potřebných rutin knihovny muParserX tak, aby byly dobře použitelné v Qt aplikaci. Tato komponenta kalkulačky byla implementována skrze třídu `Calculator`.

4.2.4 Oddělení výpočetní logiky od zbytku aplikace

Pilířem návrhu aplikace bylo oddělení výpočetní logiky kalkulačky od GUI tak, aby nebyl uživatel obtěžován právě probíhajícím výpočtem a mohl s rozhraním aplikace nakládat volně a nerušeně. Výpočetní jádro je separováno pomocí třídy `CalculatorWrapper` (zdrojový kód 3). Tato třída tvoří v podstatě most mezi třídou `Calculator` a zbytkem komponent, které ji využívají.

```
1  class Calculator;
2  class QThread;
3
4  class CalculatorWrapper : public QObject {
5     Q_OBJECT
6
7   public:
8     explicit CalculatorWrapper(QObject *parent = 0);
9     ~CalculatorWrapper();
10
11    Calculator *getCalculator();
12
13    static CalculatorWrapper &getInstance();
14
15  private:
16    Calculator *m_calculator;
17    QThread *m_thread;
18
19    static QScopedPointer<CalculatorWrapper> s_instance;
20 };
```

Zdrojový kód 3: Prototyp třídy CalculatorWrapper

`CalculatorWrapper` používá samostatné vlákno (zdrojový kód 4) pro obalení třídy `Calculator`. Všechen kód (tedy všechny metody) třídy `Calculator` se vykonávají v tomto odděleném vlákně. Při destrukci instance této třídy dojde k ukončení vlákna (zdrojový kód 5) s čekáním na ukončení v délce jedné vteřiny. Třída `CalculatorWrapper` implementuje návrhový vzor Singleton [19, str. 168–190], protože v rámci aplikace je vyžadována právě jedna výpočetní komponenta, která obstará výpočty pro všechny funkce aplikace.

```
1  CalculatorWrapper::CalculatorWrapper(QObject *parent) : QObject(
       parent) {
2    // Create calculator.
3    m_calculator = new Calculator();
4
5    // Create separate thread for calculator.
6    m_thread = new QThread();
7
8    // Prepare calculator for usage in separate thread.
9    m_calculator->moveToThread(m_thread);
10   connect(m_thread, &QThread::started, m_calculator, &Calculator::
       initialize);
11   connect(m_thread, &QThread::finished, m_thread, &QThread::
       deleteLater);
12 }
```

Zdrojový kód 4: Přesun instance třídy Calculator do samostatného vlákna

```
1  CalculatorWrapper::~CalculatorWrapper() {
2    qDebug("Deleting calculator wrapper.");
3
4    m_thread->quit();
5    m_thread->wait(1000);
6
7    delete m_calculator;
8  }
```

Zdrojový kód 5: Ukončení vlákna s instancí třídy Calculator

4.2.5 Konstanty, proměnné a funkce

Kalkulačka dokáže pracovat s uživatelskými proměnnými a předdefinovanými konstantami. Jádro kalkulačky (tedy knihovna muParserX) poskytuje základní programové rozhraní, neboli Application Programming Interface (API), které umožňuje pojmenovávat proměnné (konstanty) a ty ukládat pro další použití. muParserX k tomu využívá třídu `std::map<T,Y>`, kde `T` je názvem proměnné (konstanty) a `Y` je ukazatelem na objekt, který představuje hodnotu proměnné (konstanty).

Každá proměnná (konstanta) obsahuje unikátní textový identifikátor a hodnotu. To může být pro náročnějšího uživatel málo. Představme si například, že uživatel definuje proměnnou „moje_promenna“ a bude chtít vědět, jaký je její účel. Ve zmíněné implementaci není žádná možnost, jak ukládat k proměnné (konstantě) dodatečné textové informace. Proto byl tento koncept rozšířen ve třídě `Calculator`.

4.2.5.1 Reprezentace proměnných, konstant a funkcí

Jako základní datová struktura byla opět použita mapa, tentokrát mapa z knihovny Qt. Její prototyp je `QMap<QString, MemoryPlace*>`.

Každá proměnná disponuje unikátním identifikátorem a je představována objektem struktury `MemoryPlace` (zdrojový kód 6).

```
1  struct MemoryPlace {
2    enum Type {
3      CONSTANT = 0,
4      IMPLICIT_VARIABLE = 1,
5      EXPLICIT_VARIABLE = 2,
6      SPECIAL_VARIABLE = 3,
7      FUNCTION = 4
8    };
9
10   QString m_name;
11   QString m_description;
12   Value *m_value;
13   Type m_type;
14
15   MemoryPlace(const QString &name);
16   MemoryPlace(const QString &name, const QString &description,
17               const Value &value, const Type &type);
18   ~MemoryPlace();
19 }
```

Zdrojový kód 6: Fragment deklarace struktury MemoryPlace

Rozšířené proměnné (konstanty) tedy mohou obsahovat textový popis a navíc lze u každé proměnné určit, zda je explicitně či implicitně definována (viz strana 33) či zda jde o speciální proměnnou.

Z praktických důvodů byly do mapy proměnných a konstant přidány také zabudované matematické funkce. Ty jsou také reprezentovány objektem třídy `MemoryPlace`, avšak ten nemá žádnou hodnotu. Každá funkce je tedy popsána jen svým názvem a popiskem. Rozhodnutí zahrnout funkce do mapy proměnných (konstant) se může jevit jako nekoncepční, nicméně, jak uvidíme dále, není tomu tak.

4.2.5.2 Perzistentní ukládání proměnných

Pro zvýšení uživatelského pohodlí jsem se rozhodl ukládat proměnné perzistentně. Jako uložiště byla zvolena SQLite databáze [20], a to z několika důvodů:

- přístup k datům v databázi je velmi rychlý,
- databázový systém SQLite je distribuovaný a databáze je situována v souboru,

- Qt 5 samotné SQLite nativně podporuje.

Správu připojení k SQLite databázi obstarává třída `Database`. Ta obsahuje několik metod, jež popisuje zdrojový kód 7. Podívejme se blíže na metodu `QSqlDatabase addDatabaseConnection(const QString &name)` (viz zdrojový kód 8). Ta řeší vytváření nového připojení k databázi, ale nejen to. Metoda musí zajistit také inicializaci databáze, pokud doposud neexistuje. Inicializace spočívá v načtení zdrojového SQL skriptu ze souboru a jeho následné provedení databázovým strojem. Navíc je třeba nastavit parametry databáze, viz zdrojový kód 8, řádky 16 – 19.

Existence databáze se zjišťuje pomocí testovacího dotazu (zdrojový kód 8, řádek 21). Databáze je inicializována, pokud vyhodnocení daného dotazu selže. Další informace o inicializačním skriptu databáze pro Qonverter lze nalézt ve skriptu samotném. Ten je součástí zdrojových kódů aplikace a je k nalezení v adresáři Qonverter-src/resources/database.

```
1  class Database {
2    public:
3      // Establishes new SQLite connection and returns its handle.
4      static QSqlDatabase addDatabaseConnection(const QString &name);
5
6      // Returns handle of the existing connection.
7      static QSqlDatabase getDatabaseConnection(const QString &name);
8
9      // Removes already existing SQLite connection.
10     static void removeDatabaseConnection(const QString &name);
11
12     // Removes all established connections.
13     static void removeAllConnections();
14
15   private:
16     // Returns standard path to store application database file.
17     static QString getDatabasePath();
18
19     // Initalizes database with initial data - creates necessary
         tables.
20     static void initializeDatabase(QSqlDatabase &db, QSqlQuery &q);
21 };
```

Zdrojový kód 7: Metody třídy Database

4.2.5.3 Seznam konstant, proměnných a funkcí

Konstanty, proměnné a funkce lze zobrazit v seznamu, což je velmi důležité z hlediska přehledu. Pro zobrazování se používá schéma Model-View-Controller (MVC), který obsahuje tři komponenty:

Tabulka 1: Uspořádání prvků mapy do tabulky

název konstanty	hodnota	typ hodnoty	popis
pi	3.14159	float	Archimédova konstanta
e	2.7	float	Eulerovo číslo
⋮	⋮	⋮	⋮
sin	-	-	funkce sinus
⋮	⋮	⋮	⋮

MODEL
: Definuje abstrakci mezi datovými strukturami zdrojových dat a zbytkem aplikační logiky. V tomto případě tvoří rozhraní pro mapu konstant (a proměnných). Existují různé typy modelů. Pro kalkulačku se nejlépe hodí model, který abstrahuje 2D mapu to 2D tabulky, a to tak, že každý prvek mapy představuje řádek tabulky a každý element prvku mapy představuje buňku řádku. Lepší vysvětlení podá tabulka 1. Model poskytuje metody, které berou jako argument souřadnici prvku tabulky a případně další doplňující informace. Ostatní komponenty aplikace následně volají tyto metody pro získání potřebných informací o konstantách, proměnných a funkcích.

VIEW
: Zobrazuje data, která jsou obsažena v modelu. Data jsou zobrazena například v tabulce či ve stromu.

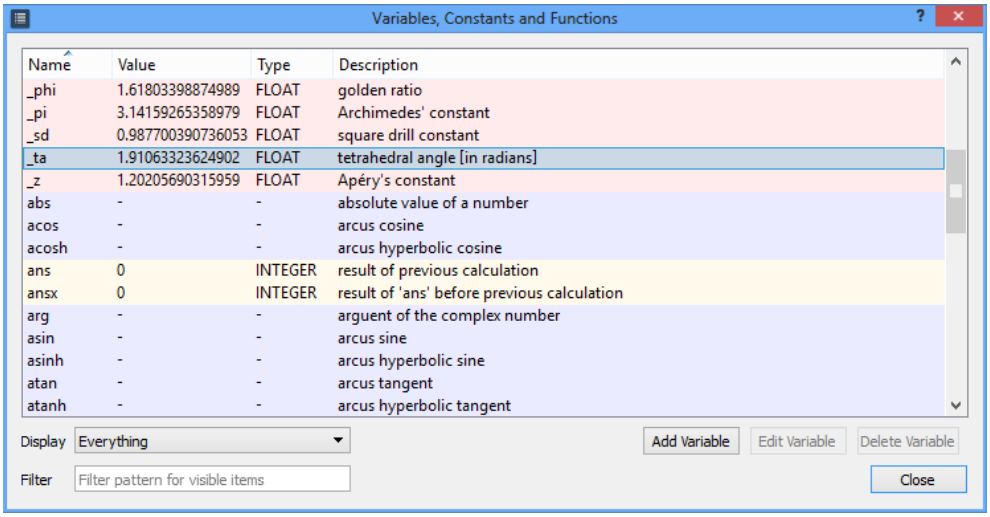
CONTROLLER
: Poskytuje možnost perzistentní úpravy dat poskytovaných modelem.

Seznam konstant, proměnných a funkcí je implementován pomocí komponent *model* a *view*. *Controller* není použit, protože není potřeba. Úpravu konstant a proměnných obstarává samostatný dialog.

Model je implementován třídou `ConstantsModel`. Ta obsahuje stěžejní metodu `data(const QModelIndex &index, int role)`. Tato metoda bere jako argument souřadnice elementu, který potřebujeme vrátit a tzv. *roli*.

Role představují účel, na jaký budou data použita. Jednou z nejvíce používaných rolí je `Qt::DisplayRole`, která se používá pro získávání tisknutelných (tedy textových) dat. Logickou strukturu metody `data(const QModelIndex &index, int role)` popisuje zdrojový kód 9.

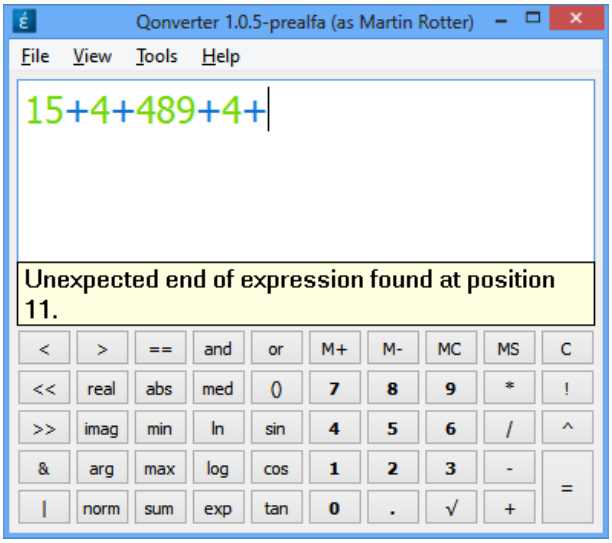
Model je použit především pro zobrazování přehledného seznamu konstant, proměnných a funkcí (obrázek 6). Ten je implementován dialogem, jenž je instancí třídy `FormVariables`. Ta obsahuje grafický seznam, což je objekt třídy `ConstantsView`. Tento seznam představuje uživatelsky viditelnou část MVC, tedy view.



Obrázek 6: Přehled konstant a proměnných

4.2.6 Plovoucí štítek

GUI kalkulačky obsahuje specifické elementy, mezi které patří plovoucí štítky. Plovoucí štítek je obdélníkový panel, jenž obsahuje viditelný text. Ten se na určitou dobu zobrazí a poté automaticky skryje. Panel je „plovoucí“, jelikož překrývá jiný prvek GUI. V případě kalkulačky je překryto vstupní textové pole. Plovoucí štítek ukazuje obrázek 7.



Obrázek 7: Plovoucí štítek

Plovoucí štítek je implementován třídou `FloatingLabel` (viz zdrojový kód 10). Štítek se automaticky umisťuje na spodní hranu nadřazeného objektu a přejímá jeho šířku (viz zdrojový kód 11).

Automatické skrývání štítku je zajištěno časovačem. Ten obstará skrytí štítku

po uplynutí určitého časového intervalu. Propojení časovače a štítku má na starost konstruktor třídy `FloatingLabel` (viz zdrojový kód 12).

4.2.7 Sestavování aplikace

Qonverter používá kompilační systém CMake [6]. CMake dokáže generovat sestavovací skripty, tzv. „makefiles“. To podstatně usnadňuje sestavování aplikace ze zdrojových kódů. Qonverter podporuje hned několik typů „makefiles“:

- Minimalistic GNU for Windows (MinGW) makefiles (pro Windows),
- GNU Compiler Collection (GCC) makefiles (pro Linux).

Qonverter patrně podporuje i další sestavovací skripty, například skripty pro Mac OS X nebo OS/2, avšak nebylo v mých silách to ověřit. Technicky nic nebrání tomu, aby aplikace fungovala na těchto operačních systémech:

- Windows (XP a vyšší),
- GNU/Linux,
- Mac OS X[3]
- Android[2],
- OS/2[2].

Pro sestavení aplikace je třeba vlastnit kompletní zdrojové kódy aplikace. Ty lze získat buď z on-line Git repozitáře nebo ze zabaleného archivu stabilních verzí programu. Vše lze nalézt na adrese [21] nebo na přiloženém DVD.

Dále je třeba mít nainstalováno Qt 5 [1]. Pro operační systém Windows se doporučuje použít verzi pro kompilátor MinGW. Sestavení aplikace vyžaduje překladač, který podporuje standard C++ 11. Dalším nástrojem nutným pro sestavení aplikace je sestavovací systém CMake [6]. Navíc může být nutné přidat cesty s binárními soubory všech prerekvizit do systémové proměnné PATH.

4.2.7.1 Sestavování z vývojového prostředí

Je vhodné použít vývojové prostředí Qt Creator, který je součástí Qt. V něm stačí nakonfigurovat cestu k spustitelnému souboru z CMake a následně otevřít soubor CMakeLists.txt, který je dostupný v kořenovém adresáři zdrojových kódů aplikace. Dále se stačí řídit grafickým průvodcem.

[3]Na tomto operačním systému nelze zajistit bezproblémový chod aplikace, protože jej nemám k dispozici. Může se tak stát, že aplikace bude mít nekonzistentní GUI nebo bude trpět jinými neduhy.

4.2.7.2 Sestavování v textovém prostředí

Aplikaci Qonverter lze pohodlně sestavit s použitím příkazového interpretu. Opět je třeba mít nakonfigurovány všechny prerekvizity. Stačí přejít do kořenového adresáře a následně spustit sekvenci příkazů, tak jak ji zobrazuje zdrojový kód 13.

4.2.8 Balíček pro AUR

Uživatelé GNU/Linux distribuce Archlinux mají k dispozici instalační skript. Ten je nabízen skrze AUR [22], což je speciální repozitář, do kterého může každý uživatel umístit svůj software. Ostatní uživatelé si jej následně mohou velmi jednoduše nainstalovat.

Vývojář, který chce publikovat svůj software skrze AUR musí respektovat podmínky, které stanovuje [23]. Následně stačí vypracovat jednoduchý skript (viz zdrojový kód 14), který poté použijí uživatelé k instalaci softwaru. Pro bližší informace lze nahlédnout do [11, kap. Publishing Qonverter for Archlinux].

4.3 Uživatelská dokumentace

Podívejme se na aplikaci Qonverter z pohledu uživatele. Předpokládejme, že používáme operační systém Windows a již disponujeme sestavenou verzí aplikace. Informace v uživatelské příručce jsou rámcově platné pro všechny podporované operační systémy.

4.3.1 Instalace

Předem upozorňuji, že tato kapitola se týká pouze operačních systémů Windows, a to z toho důvodu, že instalaci aplikace na počítačích s Mac OS X nemám možnost otestovat a pro GNU/Linux doporučuji instalaci aplikace sestavením ze zdrojových kódů (viz kapitola 4.2.7).

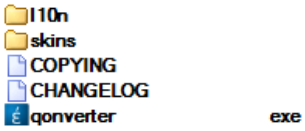
Předpokládejme tedy, že nechceme sestavovat aplikaci Qonverter ze zdrojových kódů, ale chceme nainstalovat již sestavený program v některé podporované verzi operačního systému Windows (viz kapitola 4.2.7). K tomu potřebujeme:

- binární distribuci aplikace Qonverter, kterou najdete v adresáři „Qonverter-bin“ na přiloženém DVD,
- knihovny pro běhové prostředí jazyka C++, které najdete v adresáři „VC-redist“ na přiloženém DVD. Tyto knihovny bývají na operačních systémech Windows obvykle předinstalovány, nicméně pokud tomu tak není, aplikace Qonverter se nemusí správně inicializovat, což se projeví běhovou chybou. Ta je signalizována chybovým dialogovým oknem, které v záhlaví obsahuje text „Microsoft Visual C++ Runtime“.

Knihovny pro běhové prostředí jazyka C++ je třeba instalovat jen v ojedinělých případech, neboť Windows 7 i 8 je standardně obsahují. Nyní nám nic nebrání v prvním spuštění aplikace.

4.3.2 První spuštění

Soubory předkompilované verze programu Qonverter jsou strukturovány tak, jak ukazuje obrázek 8.



Obrázek 8: Struktura souborů aplikace Qonverter

Adresář „l10n“ obsahuje binární lokalizační soubory aplikace a knihovny Qt. Naproti tomu adresář „skins“ disponuje textovými soubory vzhledu a volitelně mohou být obsaženy i další soubory, například bitmapová grafika. Nechybí ani textový soubor obsahující popis změn nebo licenční ujednání. Samozřejmostí je spustitelný soubor aplikace. Kořenový adresář aplikace může obsahovat další dodatečné knihovny. Jedná se například o použité Qt moduly.

Činnost programu lze zahájit poklepáním na spustitelný soubor „qonverter.exe“. Při prvním spuštění aplikace se zobrazí úvodní dialog s několika základními informacemi (obrázek 9). Po potvrzení úvodního dialogu se zobrazí hlavní okno aplikace (obrázek 10).

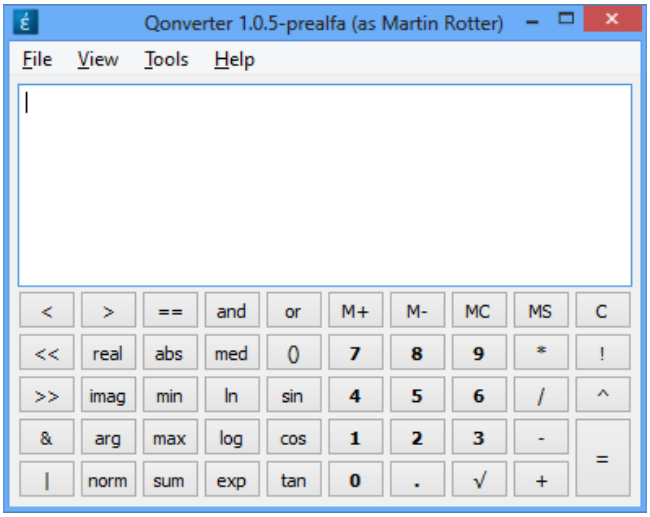
Obrázek 9: Úvodní dialog aplikace Qonverter

4.3.3 Nastavení aplikace

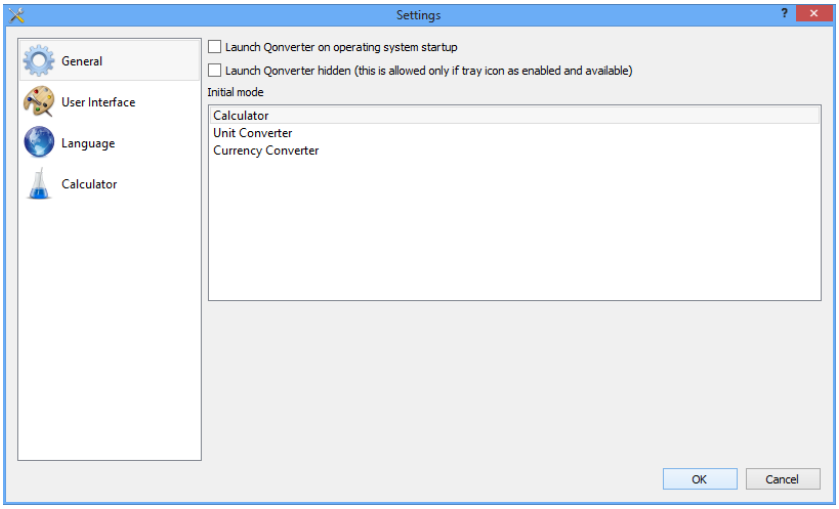
Nastavení (obrázek 11) je dostupné v hlavním menu pod položkou „Tools → Settings“. Formulář s nastavením obsahuje čtyři základní sekce:

GENERAL

Tato sekce nabízí základní nastavení chování aplikace. Je možno zvolit,



Obrázek 10: Hlavní formulář aplikace Qonverter



Obrázek 11: Nastavení aplikace Qonverter

zda se bude aplikace spouštět po startu systému. Tato sekce dále nabízí možnost volby výchozího módu aplikace nebo možnost spouštět Qonverter ve skrytém režimu (viz kapitola 4.3.7).

USER INTERFACE

V této kategorii nastavení lze objevit možnost přepínání vzhledu aplikace a nastavení notifikační ikony.

LANGUAGE

Tato sekce poskytuje jazykové nastavení aplikace.

CALCULATOR

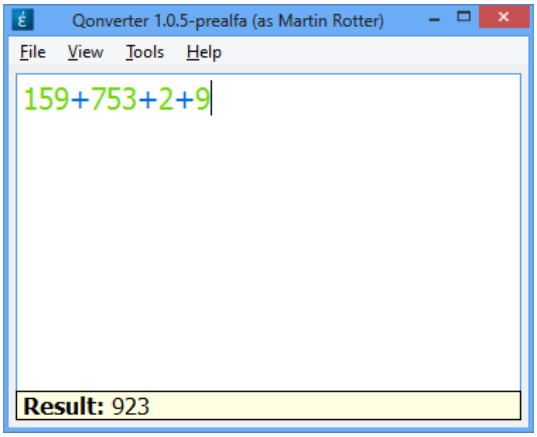
Zde najdeme nastavení, která se týkají kalkulačky. Je zde mj. možnost nastavení písma pro vstup kalkulačky a pro on-the-fly mód.

Některá konkrétní nastavení budou představena v rámci dalších kapitol.

4.3.4 Používání kalkulačky

Kalkulačka je fundamentální komponentou aplikace Qonverter. Některé její funkce ji činí zvláštní a odlišují ji od ostatních kalkulaček. GUI kalkulačky obsahuje dva základní elementy: vstupní textové pole a softwarovou klávesnici. Vzhled textového pole lze konfigurovat pomocí externích vzhledů (viz „Tools → Settings → User Interface“) nebo pomocí dodatečných nastavení (viz „Tools → Settings → Calculator“, sekce „Input Text Box Colors & Font“). Tato možnost umožňuje měnit velikost a barvy písma textového pole.

Uživatel používá kalkulačku tak, že zadává vstupní matematickou formuli skrze softwarovou klávesnici nebo přímo pomocí klávesnice hardwarové. Uživatel může oba typy klávesnic libovolně střídat, protože po stisknutí kteréhokoliv tlačítka softwarové klávesnice je přesunuto zaměření[4] na vstupní textové pole. Softwarovou klávesnici lze vypnout (viz obrázek 12) v menu „Tools → Display Calculator Keypad“. Qonverter si toto nastavení pamatuje i pro další spuštění.



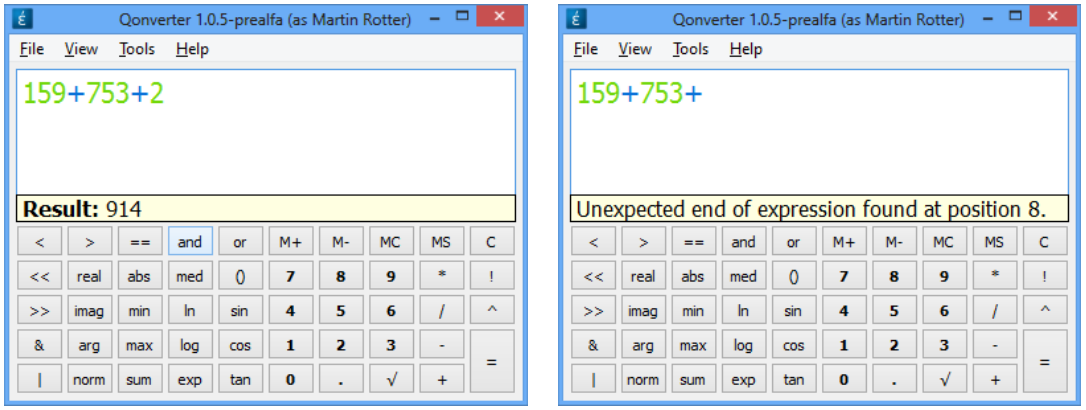
Obrázek 12: Qonverter s vypnutou softwarovou klávesnicí

Kalkulačka podporuje práci s celými i desetinnými čísly s přesností datového typu `double`, dále jsou zahrnuta komplexní čísla, booleovské hodnoty nebo řetězce.

4.3.4.1 On-the-fly mód

Typickou vlastností kalkulaček je, že vstup je zároveň i výstupem. Uživatel typicky zadá výraz a potvrdí jej stisknutím klávesy Enter. Výsledek se objeví ve vstupním textovém poli, což nemusí být nejlepší volba. Pro uživatele je lepší, když je průběžně informován o situaci v průběhu psaní výrazu. Proto Qonverter používá tzv. *on-the-fly* mód (viz obrázek 13). Ten informuje uživatele o mezivýsledku formule a případně o chybách.

[4]Zaměření je český výraz, který ne zcela přesně popisuje daný děj. Obvykle se pro popis

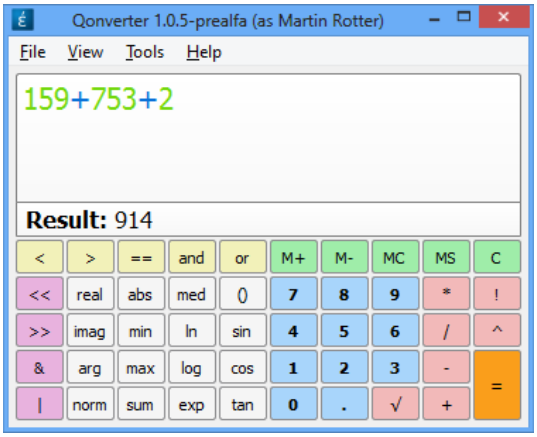


(a) Oznámení mezivýsledku výpočtu (b) Oznámení chyby výpočtu

Obrázek 13: On-the-fly mód

On-the-fly mód zobrazí panel (viz kapitola 4.2.6) s informativním textem, ze kterého se uživatel dozví vše potřebné. Panel se po určitém časovém intervalu sám skryje. Panel zároveň reaguje beze zpoždění, což je pro výsledný rozhodující. Interval zobrazování panelu a další drobnosti lze ladit v nastavení aplikace.

Panel on-the-fly módu lze vizuálně ladit prostřednictvím uživatelských stylů a skinů. Tak se může dosáhnout například toho, že panel lépe zapadne do vstupního textového pole (viz obrázek 14).

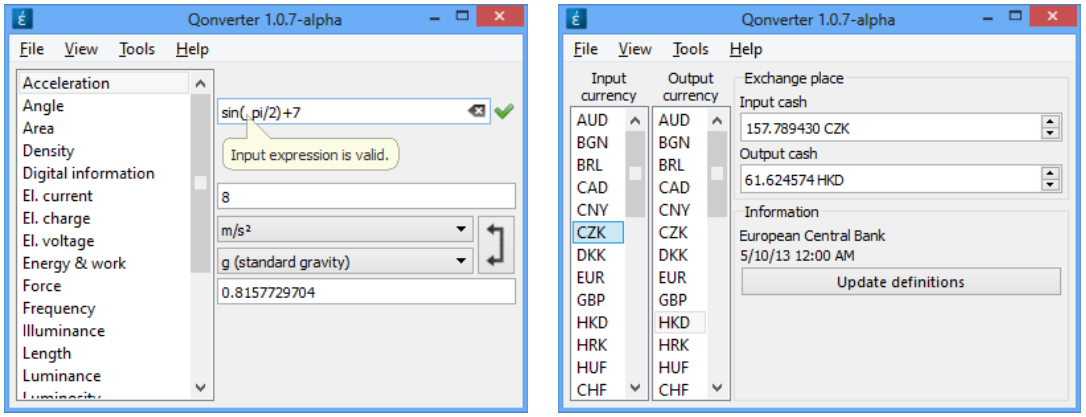


Obrázek 14: Jiný vzhled on-the-fly módu

4.3.5 Používání převodníků jednotek a měn

Menu „View“ nabízí možnost přepínat mezi třemi základními komponentami Qonverteru. S kalkulačkou jsme se již seznámili. Qonverter dále nabízí převodník jednotek a převodník měn. Obě tyto komponenty disponují očekávaným GUI (viz obrázek 15).

tohoto děje používá původní anglický výraz „focus“.



(a) Převodník jednotek (b) Převodník měn

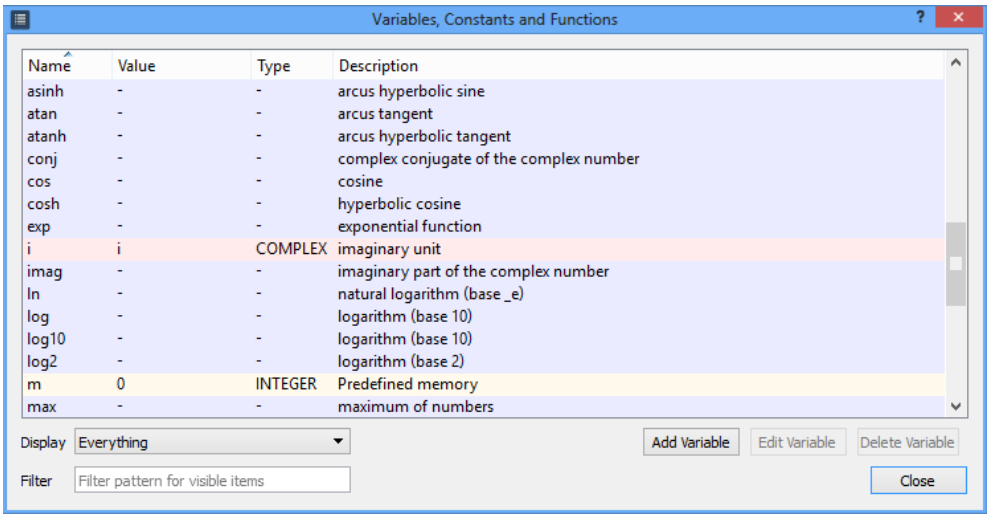
Obrázek 15: Převodník měn a převodník jednotek

Jak převodník jednotek tak převodník měn obsahují jedno vstupní textové pole, kam uživatel zadává data. Převodník jednotek dokáže zpracovat libovolný matematický výraz, protože používá stejné výpočetní jádro jako kalkulačka. Naproti tomu převodník měn akceptuje jako vstup jen celá a desetinná čísla.

Převodník měn nabízí možnost aktualizovat směnné kurzy z internetového zdroje. Jsou využívány kurzy Evropské centrální banky, jelikož se jedná po jednu z největších autorit v bankovnictví.

4.3.6 Proměnné, konstanty a funkce

Aplikace Qonverter nabízí možnost definovat uživatelské proměnné nebo použít předdefinované konstanty. Přehled proměnných, konstant a funkcí (obrázek 16) lze zobrazit volbou „Tools → Display Variables and Functions“ v hlavním menu.



Obrázek 16: Přehled proměnných a konstant

Tento dialog je základním nástrojem pro správu konstant, proměnných nebo funkcí. Qonverter disponuje hned několika typy entit, které tento dialog zobrazí:

KONSTANTY
: Jedná se o zabudované konstanty, obvykle známé číselné výrazy jako například číslo π atp. Hodnoty těchto konstant nelze měnit a rovněž nelze přidat žádnou novou hodnotu tohoto typu.

PROMĚNNÉ
: Proměnné může uživatel libovolně vytvářet a může upravovat jejich hodnotu, a to explicitně nebo implicitně:

  1. Explicitní úprava proměnné se provádí v již zmíněném dialogu se seznamem proměnných, konstant a funkcí.
  2. Implicitně lze proměnné pouze vytvářet, a to pouze ze vstupního textového pole kalkulačky. Proměnné se vytvářejí pomocí operátoru přiřazení. Výraz „$abc = 15$“ vytvoří novou implicitní proměnnou a přiřadí jí hodnotu 15. Tvorbu implicitních proměnných je třeba povolit v nastavení aplikace, konkrétně v záložce „Calculator“.

KRITICKÉ PROMĚNNÉ
: Toto jsou speciální proměnné, se kterými uživatel pracuje jako s jinými proměnnými s tím rozdílem, že nemohou být smazány. Do této kategorie patří pouze tři proměnné, a sice „ans“, „ansx“ a „m“.

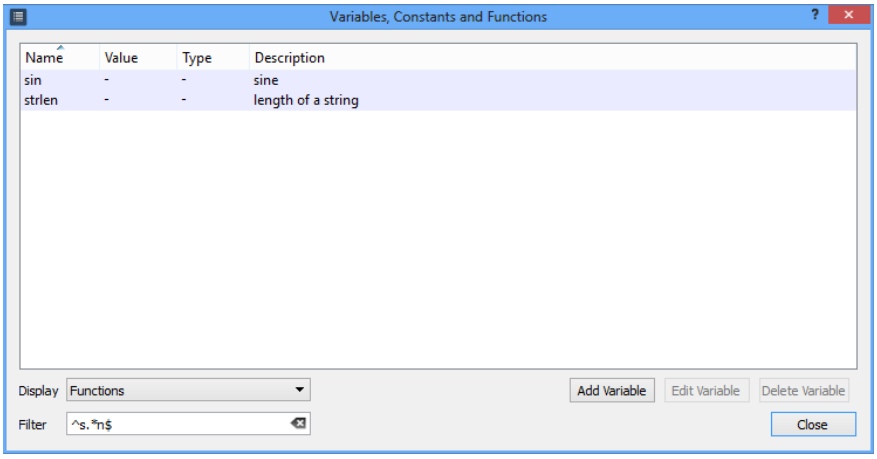
FUNKCE
: Jedná se o matematické funkce, které jsou definované buď knihovnou muParserX nebo Qonverterem jako rozšíření.

Běžné proměnné mají perzistentní charakter. To znamená, že při vypnutí aplikace jsou trvale uloženy na pevný disk počítače. Při spuštění aplikace jsou opětovně načteny. Kritické proměnné mají rozsah jen pro jedno sezení aplikace, jinak řečeno nejsou perzistentní a při dalším spuštění aplikace jsou automaticky vynulovány. Konstanty mají neměnitelné hodnoty, které jsou definovány přímo ve zdrojovém kódu aplikace a nejsou tedy ukládány na pevný disk.

Dialog funkcí, proměnných a konstant o každém zmíněném elementu zobrazí i dodatečný textový popis. Dále je možno filtrovat zobrazené položky, a to vybráním typu položky a filtrováním pomocí regulárního výrazu. Pro ilustraci si představme, že chceme zobrazit všechny funkce, které začínají znakem „s“ a končí znakem „n“. K tomu potřebujeme regulární výraz `^s.*n$`. Výsledek našeho snažení zobrazuje obrázek 17.

4.3.7 Notifikační ikona

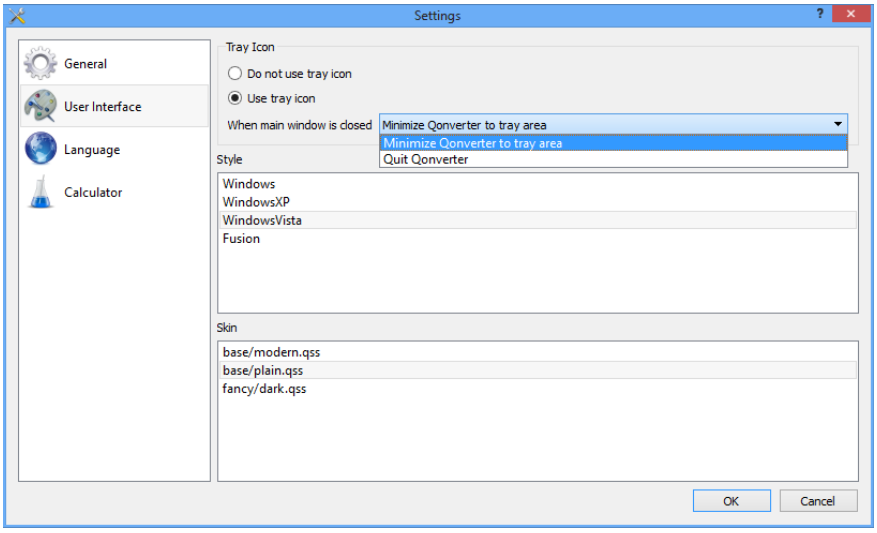
Drtivá většina softwarových kalkulaček podporuje pouze režim hlavního okna bez notifikační ikony. Uživatel takovou aplikaci spustí, provede výpočty a následně ji



Obrázek 17: Filtrování funkcí

ukončí. To je nevýhodné, protože uživatel musí při dalším použití aplikaci opět spustit.

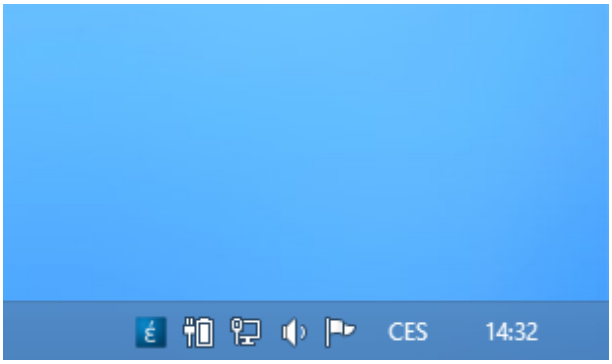
Qonverter nabízí jak režim jednoho okna tak režim s notifikační ikonou. Mezi režimy lze libovolně přepínat (viz obrázek 18) za předpokladu, že cílový operační systém notifikační ikonu podporuje.



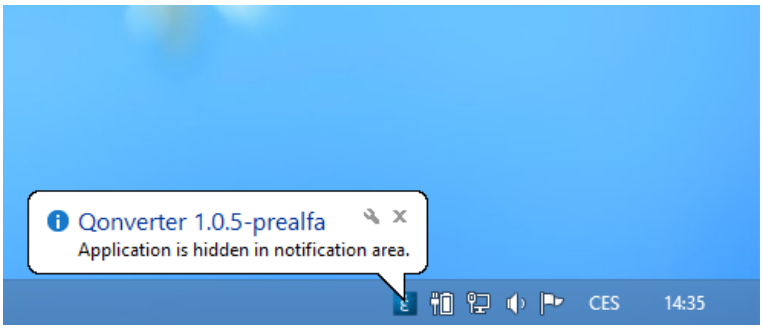
Obrázek 18: Nastavení notifikační ikony

Notifikační ikona se umisťuje do standardní části hlavního panelu. V operačním systému Windows 8 se jedná o pravou část hlavního panelu (viz obrázek 19).

Nutno dodat, že notifikační ikonu lze snadno přehlédnou a leckteří uživatelé, jenž kalkulačku testovali, si ztěžovali, že nebyli upozorněni na to, že Qonverter byl skryt do notifikační oblasti. Proto pokud dojde k prvnímu skrytí Qonverteru zavřením hlavního okna klepnutím na „křížek“ v rohu okna, tak notifikační ikona zobrazí informativní bublinovou informaci (viz obrázek 20).



Obrázek 19: Vzhled notifikační ikony



Obrázek 20: Bublinové oznámení notifikační ikony

4.4 Qonverter a jeho budoucnost

Vývoj aplikace Qonverter touto bakalářskou prací rozhodně nekončí. Mám v plánu přidat hodně nových funkcí, například historii výpočtů nebo export výsledků do souboru. Mnoho plánovaných funkcí jsem odložil na později, protože by bylo náročné je důkladně otestovat. Preferoval jsem tedy stabilitu před výčtem funkcí.

Vzhledem k živelnému rozvoji knihovny Qt ve verzi 5 se dají očekávat změny v kódu aplikace, které budou souviset s úpravami Qt API.

Budoucnost Qonverteru se bude odvíjet od úrovně penetrace Qt 5 do hlavních linuxových distribucí. Knihovna Qt 5 je zatím (ke konci března 2013) balíčkována pouze pro distribuci Archlinux nebo pro Ubuntu. Tato situace se bude v průběhu roku 2013 nepochybně měnit k lepšímu. Až se tak stane, tak budou k dispozici balíčky Qonverteru pro všechny důležité distribuce.

V ostatních operačních systémech je situace nepoměrně lepší. Pro Windows bude k dispozici předkompilovaný Qonverter, pro který bude použit překladač MinGW. Pro Mac OS X může být rovněž k dispozici binární aplikace, najde-li se zájemce, který Qonverter optimalizuje a bude obstarávat její překlad.

Qonverter bude komunitou průběžné lokalizován do potřebných jazyků. Vždy bude k dispozici český i anglický překlad.

Po implementaci všech plánovaných funkcí očekávám zvýšení počtu stažení aplikace (v jakékoliv formě) na stovky měsíčně, což je reálný cíl.

Má účast v projektu muParserX bude také pokračovat.

```
1  QSqlDatabase Database::addDatabaseConnection(const QString &name) {
2    QString db_path = QDir::toNativeSeparators(getDatabasePath());
3    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE", name);
4    QString folder = db_path.left(db_path.lastIndexOf(QDir::separator
         ()) + 1);
5
6    if (!db.isValid()) {
7      qFatal("QSQLITE database server was NOT found. Make sure that
           application and its dependencies are installed correctly.");
8      return db;
9    }
10
11   db.setDatabaseName(db_path);
12   qDebug("Opening database '%s'.", qPrintable(QDir::
         toNativeSeparators(db.databaseName())));
13   QDir().mkpath(folder);
14
15   if (db.open()) {
16     db.exec("PRAGMA synchronous = OFF");
17     db.exec("PRAGMA journal_mode = MEMORY");
18     db.exec("PRAGMA count_changes = OFF");
19     db.exec("PRAGMA temp_store = MEMORY");
20
21     QSqlQuery q = db.exec("SELECT q_information.value FROM
           q_information WHERE q_information.key = 'schema_version'");

22     if (q.lastError().isValid()) {
23       qWarning("Error occurred. Database is not fully initialized.
             Initializing now.");
24       initializeDatabase(db, q);
25     }
26     else {
27       q.next();
28       qDebug("Database connection '%s' to file %s seems to be loaded.
             ",
29           qPrintable(name),
30           qPrintable(QDir::toNativeSeparators(db.databaseName())));
31       qDebug("Version of database schema is '%s'.",
32           qPrintable(q.value(0).toString()));
33     }
34     q.finish();
35   }
36   else {
37     qFatal("Database was NOT opened. Error occurred: %s",
38         qPrintable(db.lastError().databaseText()));
39   }
40   return db;
41 }
```

Zdrojový kód 8: Přidávání připojení k databázi

```
1  QVariant ConstantsModel::data(const QModelIndex &index, int role)
       const {
2    switch (role) {
3      case Qt::ToolTipRole:
4      case Qt::DisplayRole:
5        switch (index.column()) {
6          case (int) Calculator::VARCONST_DESC: {
7            // Vrať textový popis proměnné či konstanty.
8          }
9          case (int) Calculator::VARCONST_VAL_TYPE : {
10           // Vrať typ hodnoty proměnné či konstanty.
11           QChar type = m_calculator->queryVariable(index.row(),
12                                                    static_cast<Calculator::
                                                       VarConstData>(index.column
                                                       ())).toChar();
13           switch (type.toLatin1()) {
14             case 'i':
15               return "INTEGER";
16             case 'f':
17               return "FLOAT";
18             case 'b':
19               return "BOOLEAN";
20             case 's':
21               return "STRING";
22             case 'c':
23               return "COMPLEX";
24             default:
25               return "VOID";
26           }
27         }
28         .
29         .
30         .
31       }
32     case Qt::BackgroundRole:
33       // Vrať barvu pozadí pro proměnnou či konstantu na daných souř
            adnicích.
34       .
35       .
36       .
```

Zdrojový kód 9: Struktura metody data()

```
1  class FloatingLabel : public QLabel {
2     Q_OBJECT
3
4    public:
5     explicit FloatingLabel(QWidget *parent = 0);
6     ~FloatingLabel();
7
8     void adjust();
9     void initialize();
10
11   public slots:
12    void showText(const QString &text, int duration = 1000);
13
14   private:
15    QTimer *m_timer;
16 };
```

Zdrojový kód 10: Deklarace plovoucího štítku

```
1  // Nastavuje velikost štítku a jeho pozici a šířku.
2  void FloatingLabel::adjust() {
3    adjustSize();
4
5    setFixedWidth(parentWidget()->width());
6
7    move(pos().x(),
8        parentWidget()->pos().y() + parentWidget()->height() - height()
             );
9  }
10
11 // Zobrazí štítek na určitý počet milisekund.
12 void FloatingLabel::showText(const QString &text, int duration) {
13   setText(text);
14   adjust();
15   show();
16
17   m_timer->stop();
18   m_timer->start(duration);
19 }
```

Zdrojový kód 11: Nastavení umístění a velikosti štítku a jeho automatické skrývání

```
1  FloatingLabel::FloatingLabel(QWidget *parent) : QLabel(parent) {
2    setVisible(false);
3    setPalette(QToolTip::palette());
4    setAutoFillBackground(true);
5    setFrameShape(QFrame::Box);
6    setWordWrap(true);
7
8    m_timer = new QTimer(this);
9    m_timer->setSingleShot(true);
10
11   // Propojení časovače a štítku.
12   connect(m_timer, &QTimer::timeout, this, &FloatingLabel::hide);
13 }
```

Zdrojový kód 12: Konstruktor třídy FloatingLabel

```
1 mkdir build
2 cd build
3 cmake ../ -DCMAKE_BUILD_TYPE=release -DCMAKE_INSTALL_PREFIX=/usr
```

Zdrojový kód 13: Kompilace aplikace Qonverter z textového prostředí

```
# Maintainer: Martin Rotter <rotter.martinos@gmail.com>

pkgname=qonverter-git
pkgver=20130412
pkgrel=1
pkgdesc='Very simple and easy-to-use desktop calculator with unusual
    functions.'
arch=('i686' 'x86_64')
url="http://code.google.com/p/qonverter"
license=('GPL3')
depends=(qt5-base)
makedepends=(gcc git cmake)

_gitname=qonverter
_gitroot=https://Rotter.Martinos@code.google.com/p/qonverter/

build() {
  cd ${srcdir}
  msg "Cloning " ${_gitname} " repository..."

  if [ -d ${_gitname} ] ; then
    cd ${_gitname} && git pull origin master
    msg "The local files are updated."
  else
    git clone ${_gitroot}
  fi

  msg "Git checkout done or server timeout."
  cd ${srcdir}/${_gitname}

  if [ ! -d "build" ]; then
    mkdir build
  fi

  cd build

  msg "Preparing files with cmake..."
  cmake ../ -DCMAKE_INSTALL_PREFIX=/usr -DCMAKE_BUILD_TYPE=release
}

package() {
  cd ${srcdir}/${_gitname}/build

  msg "Compiling files..."
  make DESTDIR=${pkgdir} install || return 1

  msg "All files were successfully compiled."
}
```

Zdrojový kód 14: Kompilace aplikace Qonverter skrze AUR

Závěr

V rámci této bakalářské práce jsem vytvořil učebnici, která pojednává o některých důležitých aspektech knihovny Qt. Učebnice se také zaobírá souvisejícími tématy, která by mohla potenciálního čtenáře zajímat, a je doplněna sadou miniaturních a plně funkčních programů, které slouží pro ilustraci témat.

Byla naprogramována větší ukázková aplikace, která doplňuje učebnici a ukazuje čtenáři příklad, jak použít různé komponenty knihovny Qt v jednom programu. Součástí řešení je také sada skriptů, které realizují pohodlné sestavení aplikace. Ukázková aplikace je plně funkční software, který je pro určitou cílovou skupinu uživatelů užitečný.

Nastudováním učebnice, referencí a přidruženého softwaru může čtenář získat základní vhled do možností tvorby otevřeného softwaru pomocí Qt.

Conclusions

In terms of this bachelor thesis, I created the textbook which deals with some important principles of the Qt framework. The textbook also describes related topics which potential reader might be interested in. Moreover, the textbook is supplemented with the set of small and fully-functional sample programs.

I programmed a larger application which shows a way of using the Qt framework. Application is replenished with some scripts for comfortable deployment. Application is fully-featured and could be interesting for certain group of users.

Potential textbook reader can gain some knowledge of the open-source software development by reading the textbook, its references and by browsing related source code.

A Obsah přiloženého DVD

Přiložené DVD obsahuje následující položky:

adresář Qt
: Zde můžeme nalézt instalační soubor pro Qt řady 5 pro operační systém Windows, které je třeba pro sestavení Qonverteru ze zdrojových kódů.

adresář CMake
: Obsahuje instalační program pro sestavovací nástroj CMake, který je třeba pro sestavení Qonverteru ze zdrojových kódů.

adresář Qonverter-src
: Tento adresář obsahuje kompletní zdrojové kódy aplikace Qonverter.

adresář Qonverter-bin
: Tento adresář obsahuje binární distribuci aplikace Qonverter. Funkčnost tohoto sestavení byla otestována v operačním systému Windows 8.

adresář Učebnice
: V tomto adresáři se nachází elektronická verze učebnice o knihovně Qt spolu se zdrojovými kódy ukázkových programů. Zdrojové kódy učebnice lze najít na [11].

adresář VC-redist
: Zde najdete instalační soubory pro instalaci základních běhových C++ knihoven.

Seznam zkratek

API	Application Programming Interface
AUR	Arch User Repository
GCC	GNU Compiler Collection
GNU GPL	GNU General Public License
GNU LGPL	GNU Lesser General Public License
GUI	Graphical User Interface
KDE	K Desktop Environment
MinGW	Minimalistic GNU for Windows
MVC	Model-View-Controller

Literatura

1 QT-PROJECT. *Qt 5 Online Reference Documentation* [online], 2013 [cit. 2013-01-14]. Dostupný z WWW: ⟨http://www.qt-project.org/doc/qt-5.0/⟩.

2 QT-PROJECT. *Qt Online Wikipedia* [online], 2013 [cit. 2013-01-14]. Dostupný z WWW: ⟨http://www.qt-project.org/wiki/⟩.

3 EZUST, Alan; EZUST, Paul. *An Introduction to Design Patterns in C++ with Qt*. Druhé vyd. Westford, MA : Prentice Hall, 2011. Prentice Hall Open Source Software Development Series. ISBN 978-0-13-282645-7.

4 MOLKENTIN, Daniel. *The Book of Qt 4*. 555 De Haro Street, San Francisco, CA 94107 : No Starch Press, 2007. ISBN 978-3-937514-12-3.

5 THELIN, Johan. *Foundations of Qt Development*. 2007. Expert's Voice in Open Source. ISBN 1-59059-831-8.

6 KITWARE, INC. *CMake* [online], 2013 [cit. 2013-04-12]. Dostupný z WWW: ⟨http://www.cmake.org/⟩.

7 PROJECT KDE. *KDE Project Homepage* [online], 2013 [cit. 2013-03-26]. Dostupný z WWW: ⟨http://www.kde.org/⟩.

8 OPEN SOURCE INITIATIVE. *Q Public License 1.0* [online], 2013 [cit. 2013-04-20]. Dostupný z WWW: ⟨http://opensource.org/licenses/QPL-1.0⟩.

9 OPEN SOURCE INITIATIVE. *GNU Lesser General Public License 2.1* [online], 2013 [cit. 2013-04-20]. Dostupný z WWW: ⟨http://opensource.org/licenses/lgpl-2.1.php⟩.

10 OPEN SOURCE INITIATIVE. *GNU General Public License 3.0* [online], 2013 [cit. 2013-04-20]. Dostupný z WWW: ⟨http://opensource.org/licenses/GPL-3.0⟩.

11 ROTTER, Martin. *Qt: Internals and Principles* [online], 2013 [cit. 2013-03-01]. Dostupný z WWW: ⟨http://github.com/Martin-Rotter/qt-internals-and-principles/blob/master/qt-internals-and-principles.pdf⟩.

12 PROJECT KDE. *Necessitas* [online], 2012 [cit. 2013-03-01]. Dostupný z WWW: ⟨http://necessitas.kde.org⟩.

13 GOOGLE, INC. *Android SDK* [online], 2013 [cit. 2013-04-12]. Dostupný z WWW: ⟨http://developer.android.com/sdk/index.html⟩.

14 GOOGLE INC. *Android NDK* [online], 2013 [cit. 2013-04-12]. Dostupný z WWW: ⟨http://developer.android.com/tools/sdk/ndk/index.html⟩.

15 RAYMOND, Eric S. *The Cathedral and the Bazaar* [online], 2002 [cit. 2013-03-10]. Dostupný z WWW: ⟨http://www.catb.org/esr/writings/homesteading/cathedral-bazaar/cathedral-bazaar.ps⟩.

16 PROJECT KDE. *KCalc* [online], 2013 [cit. 2013-04-12]. Dostupný z WWW: ⟨http://utils.kde.org/projects/kcalc/⟩.

17 TORVALDS, Linus. *Git* [online], 2013 [cit. 2013-04-20]. Dostupný z WWW: ⟨http://git-scm.com/⟩.

18 BERG, INGO. *muParserX library* [online], 2013 [cit. 2013-05-12]. Dostupný z WWW: ⟨https://code.google.com/p/muparserx/⟩.

19 FREEMAN, Elisabeth; FREEMAN, Eric; BATES, Bert; SIERRA, Kathy; ROBSON, Elisabeth. *Head First Design Patterns*. První vyd. 2004. ISBN 978-0596007126.

20 SQLITE CONSORTIUM. *SQLite Database System Homepage* [online], 2013 [cit. 2013-05-12]. Dostupný z WWW: ⟨http://www.sqlite.org/⟩.

21 ROTTER, Martin. *Git repozitář projektu Qonverter* [online], 2013 [cit. 2013-04-12]. Dostupný z WWW: ⟨https://code.google.com/p/qonverter/source/⟩.

22 ARCHLINUX PROJECT. *Arch User Repository* [online], 2013 [cit. 2013-01-14]. Dostupný z WWW: ⟨https://wiki.archlinux.org/index.php/Arch_User_Repository⟩.

23 ARCHLINUX PROJECT. *Arch Packaging Standards* [online], 2013 [cit. 2013-01-14]. Dostupný z WWW: ⟨https://wiki.archlinux.org/index.php/Arch_Packaging_Standards⟩.