Vysoká škola báňská – Technická univerzita Ostrava
Faculty of Eletrical Engineering and Computer Science

Programmer documentation

Summary

This application is created in C++ using Qt framework for creating nice and intuitive GUI.

In this program there is used object-oriented programming and MVC architecture. This means that program is well-structured, it's divided into three separated parts (model, view, controller) that communicate with each other.

Because of OOP, there is no own memory management. Due to constructors and destructors, programmer doesn't have to allocate and deallocate memory on his own, implicit destructors creater by compiler do it. Qt has also own memory management, so there is almost no possibility of memory leak.

Interesting solutions

There are many interesting solutions of different problems. Program can import/export data from/to **CSV** files, **XML** Files (import is handled by Qt xml parser), program can also load and save data from/to binary file **BCF** (Business cards file), extension was creates specially for this project.

There is also possibility to export file to **SQL** (MySQL) format to create remote database and export to **HTML** file. Both of these exports are connected to special dialog for filtering exporting data and other settings.

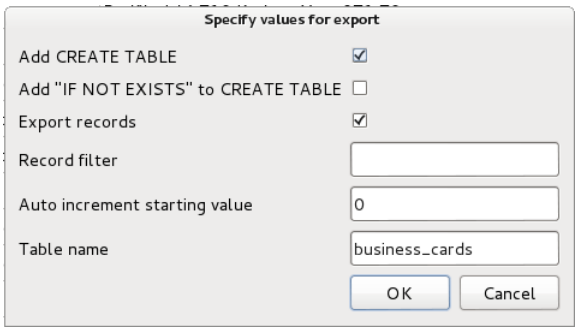

Illustration 1: Export data to SQL

It's worth mentioning that whole program is user-friendly. User can use **keyboard shortcuts** described in user documentation, there are also mnemotic shortcuts for easy menu access. Program is also **intuitive**, if you double-click on record (selected row), there will pop-up edit window for changing data in selected record.

User can also sort records inside the program (sorting is achieved by internal Qt objects). There is also filtering by user name. There are tabs „*A-Z*“ for filtering records by first letter in Name, for non-english, non-traditional names there is tab „*Others*“ (for example record with name „*Šárka*“ won't display under tab „*S*“) and there is also tab „*All*“ that displays all records.

Another nice feature is checking for editting data. This feature interferes with default closing event, so user won't loose unsaved data by an accident. Same piece of code also handles default window title. If the project is unsaved, program will have star (*) in the title to mark unsaved data. In the

Illustration 2: Unsaved project marking

title, there is also shown opened projects filename. If there is no file opened, default title is "Untitled".

As i said, the program is intuitive – There are implemented **drag&drop** methods from Qt, so instead of pressing open button, you can drag and drop the file inside main window. Sadly, it only works for own binary file. Since it's not proprietary, It doesnt have custom mime type, so only way, how to check for correct file is by extension or by passing it in DataWidget method inside try/catch. Both ways are uncomfortable and there is chance for misinterpreting input file. So if you drop another file than binary (BCF) file, it throws warning window. Of course it's not 100% percent safe. Only way, how to secure whole program is to use try/catch block on main application and overriding Qapplication::notify() method, but since we assume users will be dealing with given data and the application won't be used for big important data, It's not implemented.

Last interesting feature is multiplatformness. Because of Qt framework, program can be compiled under Linux, MacOS, Windows, even under Windows mobile and Android platform. But compiling it for mobile use will need some modifications because version of Qt that supports it (5.0) has been released on 20. 12. 2012. This program has been tested, programmed and compiled under Qt 4.8.4.

Code example

```
QXmlStreamReader xml(&file); // open stream

// start reading file from start, ends if error occurs
while(!xml.atEnd() && !xml.hasError()) {
  QXmlStreamReader::TokenType token = xml.readNext(); // obtain current token
  if(token == QXmlStreamReader::StartDocument) { // if it is header token, just read next token
    continue;
  }
  if(token == QXmlStreamReader::StartElement) { // parent element
    if(xml.name() == "records") { // if it is tag "records", just continue, we need child elements
        continue;
    }
    if(xml.name() == "card") { // handle correct data
        xml.readNext();

        while(!(xml.tokenType() == QXmlStreamReader::EndElement && xml.name() == "card")) { // and browse through all child elements
            if(xml.tokenType() == QXmlStreamReader::StartElement) {
                if(xml.name() == "name") {
                    xml.readNext();
                    name = xml.text().toString();
                }

                if(xml.name() == "profession") {
                    xml.readNext();
                    profession = xml.text().toString();
                }
```

Illustration 3: Part of import XML parser