SMS Booklet v1.0 Technical Documentation

Development Platform

We've decided to go with the first official preview release of Qt for Android (5.1.x / gcc 4.8.x) due to two major points:

- Staying cross-platform and minimizing platform-dependent code to make future ports to iOS, Windows Phone and BlackBerry easier.
- We needed page-flip animation which is easier to implement in QML rather than writing OpenGL ES.

Right now our code runs without any major issues on both Microsoft Windows and Android platforms by building from the same source code.

Devices

We support all Android devices with 16:9, 16:10, 4:3 screen ratios. On devices with ratio other the mentioned ones we stretch the screen a bit.

Major Issues

The biggest issue that we've encountered with, is the countless amount of critical/blocker bugs in Qt for Android 5.1.x. Unfortunately this release has a lot of known and unknown bugs which made our development cycle longer. We'll address some of them and the possible workarounds that we figured-out for most of them in the rest of this document.

Source Code

We kept our code from the beginning of the development under Git-SCM. So, it's possible to keep track of the development process and switch between different releases. Also, as a general rule of thumb

we've kept all QML code as resource inside our binary due to security concerns and other assets such as images, fonts and database as simple android assets. Other codes such as C++ and Java will be compiled into binary form. So, there should be no concern about them.

Code Hierarchy

Well, our code consists of three major parts which are implemented in C++, QML (Qt Quick 2.1) and Java (for native API access), and collaboration between Java/C++ parts was made possible by using Android JNI which is a part of Android NDK. Also we implemented a custom method for interaction between C++ and QML.

C++ Part

Our C++ chunk of the code is made of the following classes and files:

About	This is about window.
Android	This class provides the C++ part of the C++/Java bridge using JNI which is part of the Android NDK.
Application	Manages application life-time.
DB	Provides access to the Database.
DBTables	Defines database tables and fields.
defs.hpp	Custom definitions in our project.
MainWindow	Main window of the application. Both splash screen, main categories and favorites are handled here.
make_unique.hpp	Provides the proposed std::make_unique for C++14 which makes memory management easier by using AAA Style (Almost Always Auto).
MessageBrowser	This class is the window for browsing messages. It's possible to favorite a SMS or share it with other apps from here.
PageModel	This class implements a QML ListModel-like model in C++ for being used by our FlipBook component. It implements QML models memory model and also required methods to interact with it from QML. We needed this class to read data from the database and represent it as QML visual items on screen. Each element in this model is a Page component instance in QML and is a Page class instance in the C++ part which is also implemented in the same file and header as PageModel itself.

RT	This class name, RT, stands for **R**un**T**ime, it's an abbreviation due to high usage, because we wanted to make the name shorter for easier typing. This class holds all instances of commonly used objects (such as database, Android interaction bridge, etc.) in program in a singleton-like fashion which provides efficiency and security in their usage. This class deals with multi-threading, thread-safety and synchronization as well, too.
SubCategoryBrowser	This class is the window for browsing sub-categories.
Window	This is base class for all windows in the apps.

QML Part

We have implemented QML part using Qt Quick version 2.1. Core part of the code is a component which is called FlipBook with some sub-components such as a Page and a behavior controller. Also, there is another component called Notification for handling animated cross platform user notifications. And, the rest of the code is based on this component.

This is the hierarchy of our FlipBook component:

FlipBook.qml	This is the actual code for our flippable book component.
FlipBookBehavior.qml	This QML file is responsible for handling user input and interactions, and so decision making which determines our book and pages behavior.
Page.qml	This QML file is implementing each page of our flippable book.

And, this is the hierarchy of our Notification component:

Notification.qml	This only QML file responsible for implementing Notification system.

Also, this is the rest of the QML files:

mainwindow.qml	Main window actual UI in Qt Quick.
messagebrowser.qml	Message browser actual UI in Qt Quick.
subcategorybrowser.qml	Subcategory browser actual UI in Qt Quick.

Java Part

Our Java bit of the code resides inside the *android folder* of the source code:

org/qtproject/qt5/android/bindings/QtActivity.java	We've been forced to modify this file which is generated by Qt Cretor itself, in order to overcome some issues such as C++/Java interoperability for features such as sharing text with other apps or handling tablet/mobile keyPressEvents from C++ side. Qt 5.2.x will provide AndroidExtras which makes these changes unnecessary and obsolete.
smsdb/Android.java	This class provides the Java part of the C++/Java bridge.

Known issues and workarounds

Bug: A crash happens when you emit a signal after hiding or closing the current window.

Workaround: Emit the signal before hiding or closing the form.

Bug: Screen flicks while animating (e.g. page flips) the front window objects. It also shows the behind window while flicking.

Workaround: After showing the front window, emit a signal to the behind one to hide itself. And, before closing the front window emit a signal to the behind window to show itself.

Bug: A crash happens when you close a window by calling *close()* method while the parent window is shown right before closing the current window.

Workaround: We hide the form on Android. But this workaround creates another barrier which is the hidden form still catches the input. We solved this by passing a std::function (similar to function pointers but more flexible) in the constructor from the parent window which handles the input when the child is hidden.

Bug: Transition between windows of the application causes a nasty flick on Android.

Workaround: We were not able to overcome this issue. Also, this is a known issue on there's a bug-report on qt-project.org mailing lists about it. We should probably wait for next major Qt for Android

release.