

---

**Writing and Running Qmltestrunner Tests**  
**Case: Jolla**




Thesis

Häme University of Applied Sciences

Degree programme in Business Information Technology

Visamäki, Autumn 2015

Lauri Johansson



VISAMÄKI  
Tietojenkäsittelyn koulutusohjelma  
Systeemityö

---

<b>Tekijä</b>	Lauri Johansson	<b>Vuosi</b> 2015
<b>Työn nimi</b>	Writing and Running Qmltestrunner Tests	

---

## TIIVISTELMÄ

Tämän opinnäytetyön toimeksiantaja oli Jolla Oy. Työn tarkoituksena oli opastaa kuinka Sailfish OS-ohjelmoijat ja yhteisön jäsenet voivat itse testata luomansa sovelluksen automatisointiin tarkoitettua työkalua käyttäen. Aihe on nyt olennainen, sillä Sailfish OS-käyttäjärjestelmän käyttö on kasvamassa maailmalla.

Työllä oli kaksi pääaihetta: Sovelluksen tekeminen Sailfish OS-laitteelle sekä automaatiotestaus, jossa perehdytään enimmäkseen qmltestrunnerin käyttöön. Työssä esitellään Sailfish OS-ohjelmointiympäristö, jossa käytetään itsetehtyä applikaatiota esimerkkinä. Tämän jälkeen esitellään automaatiotestausta lyhyesti ja esitellään kaksi automaatiotestaukseen kuuluvaa työkalua. Opinnäytetyön toiminnallisena osuutena laadittiin valmis applikaatio ja ohjeistus testityökalun käyttämisestä. Työhön käytetty aineisto on kerätty erilaisista internetlähteistä. Lisäksi omakohtaista työkokemusta on hyödynnetty opinnäytettä tehdessä. Opinnäytetyössä hyödynnettiin kokeellista tutkimusta jonka pohjalta luotiin ohjeistus qmltestrunnerin käyttöön.

Lopputuloksena saatiin valmis demosovellus jossa on testit valmiiksi laitettuna. Testit onnistuttiin ajamaan onnistuneesti laitteella ja tulokset pystyttiin lukemaan. Demosovelluksen pohjalta rakennettiin ohjeistus mer-wiki-sivustolle. Seuraava kehitysidea on sovelluksen edelleen kehittäminen monimutkaisemmaksi. Samalla voidaan lisätä uusia testejä uusia ominaisuuksia varten.

**Avainsanat** Jolla, Sailfish OS, Qmltestrunner, Test automation, Qt, QML

**Sivut** 26 s. + liitteet 6 s.

VISAMÄKI  
Business Information Technology  
Software Engineering

---

<b>Author</b>	Lauri Johansson	<b>Year</b> 2015
<b>Subject of Bachelor's thesis</b>	Writing and Running Qmltestrunner Tests	

---

ABSTRACT

The client for this thesis work was Jolla Ltd. The idea was to provide information to the Jolla Community members and third party developers about testing Sailfish OS applications with a test automation tool called qmltestrunner. This subject is currently a highly relevant topic because of the growing use of Sailfish OS in the world.

This thesis has two main subjects. The first part is the Sailfish OS and how to make applications for it. The second part contains information about test automation and introduces two of the tools used in automation. The thesis introduces the Sailfish OS programming environment by using a self-made application. The test automation is introduced first generally and then with two tools testrunner-lite and qmltestrunner, the second more specifically. The functional part of the thesis is the demo application and instructions on how to use the qmltestrunner testing tool. All the material is gathered from different internet sources, and work experience was used while creating this thesis. Experimental research was used while doing the thesis. Based on the results found from the experimental research, introduction for the qmltestrunner usage was made.

As a result of this thesis project we got a working demo application and tests running on the demo application. The instructions about how to run automated tests with qmltestrunner were made to mer-wiki page. These instructions should help the developers to test their own applications, and this would benefit Jolla Store tester for not spending much time in the UI side issues.

Further development would consist more features for the demo application and new tests. Also implementation for the Jolla Tablet.

**Keywords** Jolla, Sailfish OS, Qmltestrunner, Test automation, Qt, QML

**Pages** 26 p. + appendices 6 p.

---

## VOCABULARY

<b>Qt</b>	also said as “cute”, is a cross-platform application framework
<b>QML</b>	Qt Meta Language or in other words Qt Modeling Language is a user interface markup language
<b>Qmltestrunner</b>	Automation testing tool for unit testing.
<b>Testrunner-lite</b>	Automation testing tool. Using xml-files.
<b>Harbour</b>	Place where Sailfish developers can submit their applications
<b>OS</b>	Operating System
<b>Qt Creator</b>	A cross-platform IDE (integrated development environment)
<b>Sailfish Silica</b>	QML-module for developing Sailfish applications
<b>Mer</b>	Free and open-source software distribution. Serve hardware vendors as a middleware for Linux kernel-based mobile-oriented OS.
<b>Nemo Mobile</b>	Linux distribution for mobile devices
<b>Qt Quick</b>	Module which contains all the essential types for creating QML-based user interfaces
<b>XML</b>	Extensible Markup Language.
<b>API</b>	Application programming interface.
<b>Git</b>	Distributed revision control system.
<b>SDK</b>	Software development kit. Usually a set of software development tools.
<b>UI</b>	User interface.
<b>IDE</b>	Integrated development environment. Software application which provides comprehensive facilities.
<b>ssh</b>	Secure Shell. Targeted for secure telecommunications.
<b>Emulator</b>	Hardware or software which enables a host to behave like another computer system.
<b>Qt Test</b>	Provides classes for unit testing.
<b>Jolla</b>	New Finnish mobile phone maker.

---

---

<b>Sailfish OS</b>	New Finnish operating system based on MeeGo project.
<b>MeeGo</b>	Old project started by Nokia and Intel for Linux based mobile phone operating system.
<b>RPM</b>	RPM Package Manager (RedHat Package Manager) is Linux based package management system.
<b>CSS</b>	Cascading Style Sheets, styling document mainly for WWW-documents.
<b>VM</b>	Virtual Machine
<b>TOH</b>	The Other Half is a back cover for Jolla Phone.

---

## CONTENTS

1	INTRODUCTION.....	1
2	SUMMARY OF JOLLA LTD. ....	2
2.1	Summary of Sailfish OS.....	2
2.2	Jolla Harbour.....	3
3	MAKING APPLICATION FOR SAILFISH OS.....	5
3.1	Summary of Qt.....	5
3.2	Summary of QML.....	5
3.3	Sailfish Silica.....	6
3.4	SailfishSDK.....	7
3.4.1	RPM-Validator.....	7
3.5	MerSDK.....	8
3.6	Other Sailfish Development Kits.....	9
3.7	The Demo Application.....	9
3.7.1	Building and Installing the Demo Application to the Device.....	12
3.8	Sailfish Application Lifecycle.....	14
3.9	Summary of “libsailfishapp”.....	15
4	TEST AUTOMATION.....	17
4.1	Unit Testing.....	17
4.2	Testrunner-lite.....	18
4.3	Qmltestrunner.....	19
4.3.1	Writing Qmltestrunner Tests.....	19
4.3.2	Executing the Qmltestrunner Tests on Sailfish OS Device.....	20
4.3.3	Executing the Qmltestrunner Tests on Sailfish OS Emulator.....	22
4.3.4	Reading the results.....	23
5	SUMMARY.....	25
	SOURCES.....	26

Appendage 1	Qmltestrunner usage
Appendage 2	Invoker usage
Appendage 3	Testrunner-lite usage

---

## 1 INTRODUCTION

This thesis work is about test automation and how to use one of the test automation tools for automation work. The tool used for this is qmltestrunner. As a side of the main work this thesis also provides information about QML language, Sailfish OS and how to do applications to Sailfish OS. The client for this thesis work was Jolla Ltd. The reason for choosing this subject is because I personally work daily with qmltestrunner and there are no good instructions on how to use it, therefore making the instructions were needed.

The main purpose for this thesis is to give information to the Sailfish OS community developers about how they can test with the qmltestrunner tool. The documentation of the qmltestrunner usage is going to the mer-wiki page. In addition to documentation, this thesis also provides a demo application as an example for qmltestrunner usage.

The thesis will provide answer on how to write and run qmltestrunner tests, what is test automation, what is Sailfish OS and how to write applications for Sailfish OS devices. Most of the sources are gathered from the internet because documentation of Sailfish OS and Qt are found mostly there.

The thesis provides information about Sailfish OS development and also introduces automation test tool, qmltestrunner.

In this thesis work the demo application and the tests for the demo application were made ready and functional. Also the documentation of qmltestrunner for the wiki-page was done.

## 2 SUMMARY OF JOLLA LTD.

Jolla Ltd. is mobile phone Company established on 2011 in Finland. Currently the company has 125 employees working in Helsinki, Tampere and Hong Kong offices. Jolla Ltd. is developing its own mobile devices and a new independent operating system called Sailfish OS. The first Jolla device, Jolla Phone, was released to the public late November 2013 and is currently available all over Europe, Hong Kong, India and Russia. Jolla's next product, Jolla Tablet, was introduced November 2014 and the goal is for it to be available in autumn 2015. Jolla's main aim is to open, transparent and independent in everything they do. (Jolla Oy 2015.)

### 2.1 Summary of Sailfish OS

Like Linux distributions, the Sailfish operating system is build in the same way. The core of the OS is based on Mer Project which is an open, mobile-optimized core distribution. Jolla has developed the Sailfish UI with QML which is a powerful user experience design language provided by Qt framework. Sailfish OS has many capabilities because of the rich UI elements, like to create animated, touch-enabled UIs and lightweight applications. Jolla has created Sailfish Silica which is UI building blocks with custom components for native applications. (Sailfish OS 2015a.)

Sailfish OS has also the capability to run Android™ applications because of the Android libraries included (Sailfish OS 2015a).

By nature Sailfish OS and integration is designed to be modular because of easy support for multiple hardware targets (Sailfish OS 2015a). Image 1 shows the Sailfish OS modular system.



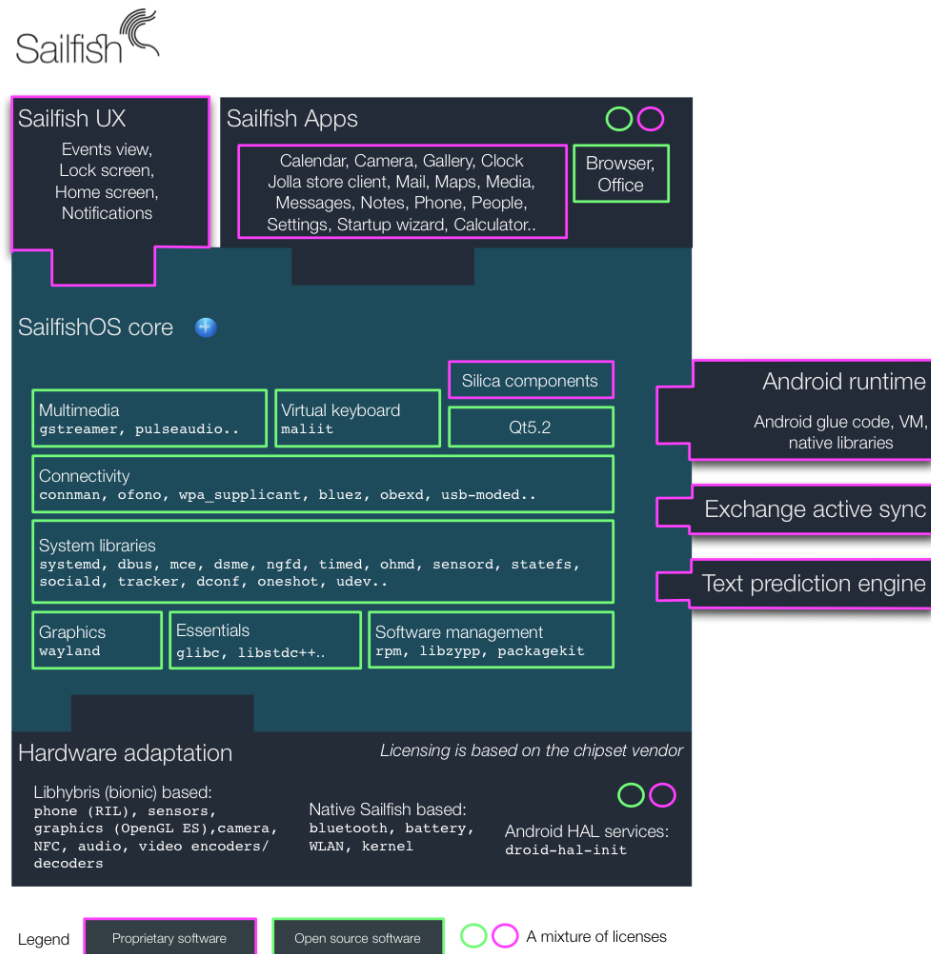


Image 1. Sailfish OS modular system.

## 2.2 Jolla Harbour

Jolla Harbour is the place where developers can submit their ready-made Sailfish OS applications or Android™ applications. The submit form can be seen on image 2.

Jolla Harbour is free to join and free to submit applications. Harbour QA will manually check the applications and see if it's compatible with all the rules that Jolla Harbour has. Also some automatic tests are done with RPM-validator. All the written rules are found in Harbours FAQ-page. (Sailfish OS. 2015b.)

The screenshot displays the 'Submit' page on Jolla Harbour. On the left, a teal sidebar contains a list of navigation items: Title, Details, Categorization, Binary, Visual assets, Contact details, and Publish settings. A 'Submit to QA' button is located at the bottom of this sidebar. Below the sidebar, the word 'Saved' is visible. The main content area is titled 'English' and includes a '+ Add a language' button. It features four form sections: 'Title\*' (with a character limit of up to 30), 'Description\*' (with a character limit of up to 4000), 'Summary\*' (with a character limit of up to 200), and 'Recent changes' (with a character limit of up to 2000). Each section has a text input field and a checkmark icon on the right.

Image 2. Submit-page on Jolla Harbour.

### 3 MAKING APPLICATION FOR SAILFISH OS

This part focuses on Qt, QML and Sailfish application development. Demo application was made and it is used to demonstrate QML-language. Because of the demo application, two SDK's is introduced SailfishSDK and MerSDK. SailfishSDK is used for developing to Sailfish OS platform. MerSDK is part of the development to Sailfish OS. This demo application can be further develop much more complicated program and also made available to the Jolla Tablet.

#### 3.1 Summary of Qt

Qt is not a programming language, it is cross-platform application development framework for desktops, embedded and mobile. Supported platforms are Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry and Sailfish OS. Qt framework is written in C++. (The Qt Company Ltd. 27.06.2015a.)

Qt has its own IDE (Integrated Development Environment) which is named QtCreator. Qt Creator runs on Linux, OS X and Windows machines. It offers intelligent code completion, syntax highlighting, an integrated help system, debugger and profiler integration. Also it includes all major version control system integrations like git. (The Qt Company Ltd. 27.06.2015a.)

#### 3.2 Summary of QML

When saying something about QML, Qt Quick needs to be introduced. Qt Quick is used for building user interfaces (UIs) for set-top boxes, tablet devices, in-vehicle infotainment systems, e-readers and mobile phones. QML, declarative language comes from Qt Quick. QML is Qt Meta-object Language. It is based on CSS and JavaScript. Most of the QML code is just CSS-style name-value pairs of properties (e.g., color: "black"). Behavioral parts like response to mouse click is written in JavaScript. Qt Quick has the access to full power of the Qt. All Qt's properties, signals and slots of a QObject-derived class are accessible from QML. (The Qt Company Ltd. 28.06.2015b.)

What makes Qt Quick and QML better than other options? Qt Quick closes the gap between UI designers and developers because it facilitates communication via one common medium. UI Designers has the access to Qt Quick through Quick Designer which is a visual design tool for QML applications. So basically designers create the UI and then developers adds the functionality to that current UI. This maintains close interaction with the designers. (The Qt Company Ltd. 28.06.2015b.)

```
35 Page {
36     id: main
37
38     ListModel {
39         id: pagesModel
40
41         ListElement {
42             page: "ButtonPage.qml"
43             title: "Buttons"
44             subtitle: "Assorted Button variants"
45             section: "Component tests"
46         }
47
48         ListElement {
49             page: "ComboPage.qml"
50             title: "Combo Box"
51             section: "Component tests"
52         }
53
54         ListElement {
55             page: "TextPage.qml"
56             title: "Text Inputs"
57             subtitle: "TextField and TextArea components"
58             section: "TextArea tests"
59         }
60
61         ListElement {
62             page: "special.qml"
63             title: "Special"
64             section: "Special tests"
65         }
66     }
67
68     SilicalistView {
69         id: listView
70         anchors.fill: parent
71         model: pagesModel
72         header: PageHeader { title: "QML Tester" }
73         section {
74             property: 'section'
75             delegate: SectionHeader {
76                 text: section
77             }
78         }
79         delegate: BackgroundItem {
80             width: listView.width
81             Label {
82                 id: firstName
83                 text: model.title
84                 color: highlighted ? Theme.highlightColor : Theme.primaryColor
85                 anchors.verticalCenter: parent.verticalCenter
86                 anchors.fill: parent.width
87             }
88             onClicked: pageStack.push(Qt.resolvedUrl(page))
89         }

```

Image 3. Picture about QML-language

### 3.3 Sailfish Silica

Sailfish Silica is included in Sailfish SDK. Sailfish Silica is a QML module for developing Sailfish applications. This module provides additional types specifically designed for Sailfish applications. While writing Sailfish applications with QML, it is must to use both modules, Sailfish Silica and QtQuick. (Sailfish OS. 2015c.)

Sailfish Silica module makes possible to write user interfaces which have Sailfish look and feel. Module gives use of unique Sailfish application features, such as pulley menus and application covers. Current release of Sailfish Silica is based on Qt 5.0 and QtQuick 2.0. All the Sailfish applications should import QtQuick 2.0 version of the module. (Sailfish OS. 2015c.)

### 3.4 SailfishSDK

SailfishSDK contains a collection of tools like: QtCreator (IDE), Mer build engine, The Sailfish OS Emulator, Tutorial, Design and API Documentation and repositories for additional libraries and open source code (Sailfish OS 2015d).

The Sailfish OS Emulator is an x86 VM image, containing a stripped down version of the target device software. The Emulator emulates most of the functions of Sailfish OS, such as gestures, task switching and ambience theming. (Sailfish OS 2015d.)

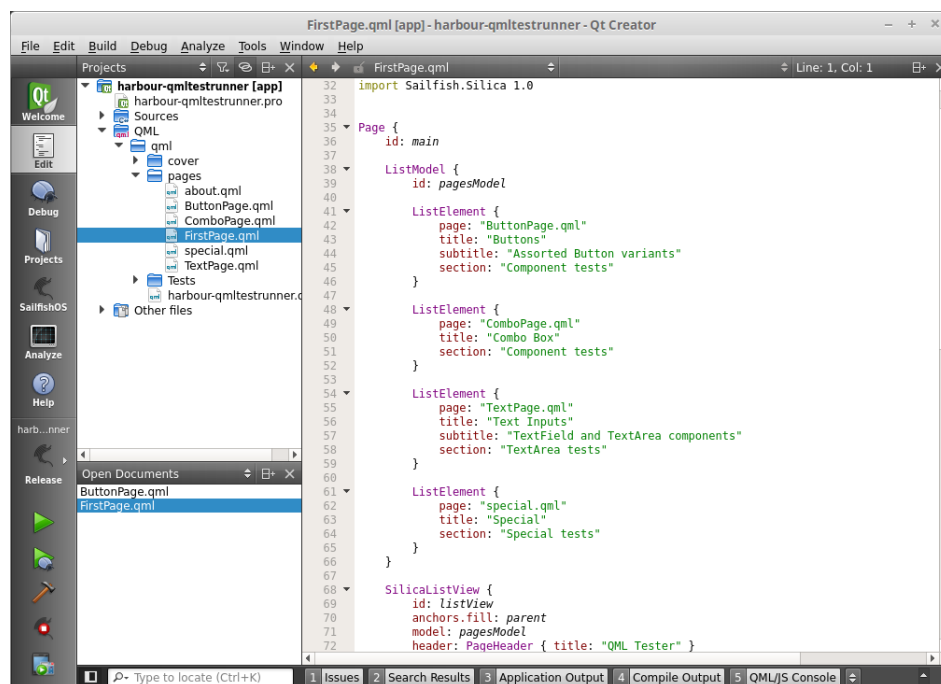


Image 4. SailfishSDK

#### 3.4.1 RPM-Validator

RPM-validator is a script made for checking quality criteria's quickly for an application. This tool runs similar checks to the Harbour package validation process and is automatically updated before each run. This tool can be found inside the Sailfish SDK, see Image 5.

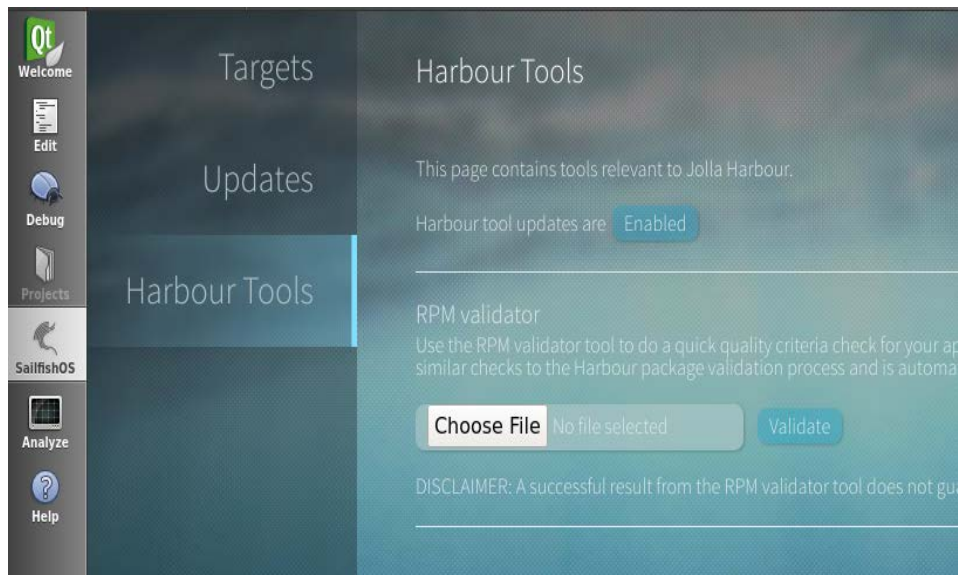


Image 5. RPM-validator tool in Sailfish SDK

### 3.5 MerSDK

MerSDK or in other words, Mer build engine, seen on image 6, is a virtual machine containing all the Mer development toolchains and tools. It also includes a Sailfish OS target for building and running Sailfish and QML applications. Mer build engine also supports additional build targets and cross-compilations toolchains. (Sailfish OS 2015d.)



Image 6. Mer build machine

### 3.6 Other Sailfish Development Kits

Besides the normal SailfishSDK, Jolla provides also two other development kits. These are The Other Half Developer Kit, which is meant for creating your own ambiances to Jolla device, and Hardware Adaptation Development Kit, which is used for creating your own SailfishOS image against and any Android™ device, e.g. Nexus 5.

Currently community has made many different kinds of The Other Half's. Most popular are TOH Keyboard, Solar Panel and Toholed.

### 3.7 The Demo Application

In this thesis, a demo application is made and built from the scratch. It contains most of the common features from the Sailfish applications. Because of the nature of this application, it is meant to be a dummy application. A Dummy application is a kind of application that does nothing particular. This kind of applications has the best way of showing the use of qmltestrunner tool. Other than showing how to write tests, it does not have any other use.

The demo application is named to match with the rules in Jolla Harbour. The rule is that the application name must start with harbour-`<your_app_name>` prefix, so the name of the demo application is harbour-qmltestrunner.

This demo application contains several common features. These features are pulley menus, buttons, combo boxes and text inputs. The main page of the application is shown in image 7. Tests are made to buttons, combo boxes and text inputs.

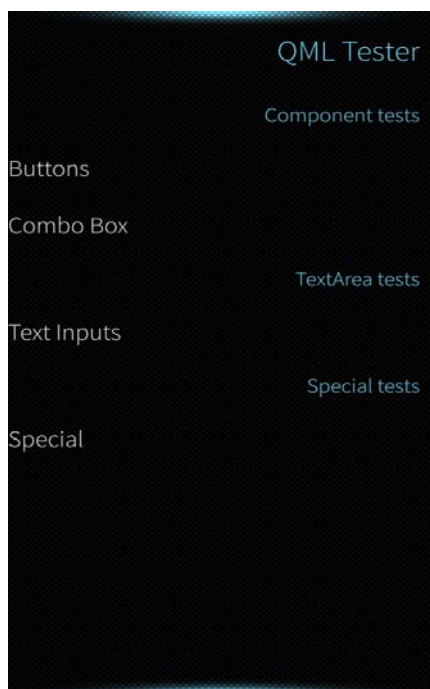


Image 7. First page of the application

On the buttons page there is switch button. Switch button is a TextSwitch type which provides a Sailfish-styled toggle button with a textual label. This button switches state to enabled and on disabled state. On the buttons page, there was also added two regular buttons that does math operations, sums and differences. Buttons were named “+” and “-“. A clear button was added so math operations can be set back to normal as shown in image 8.

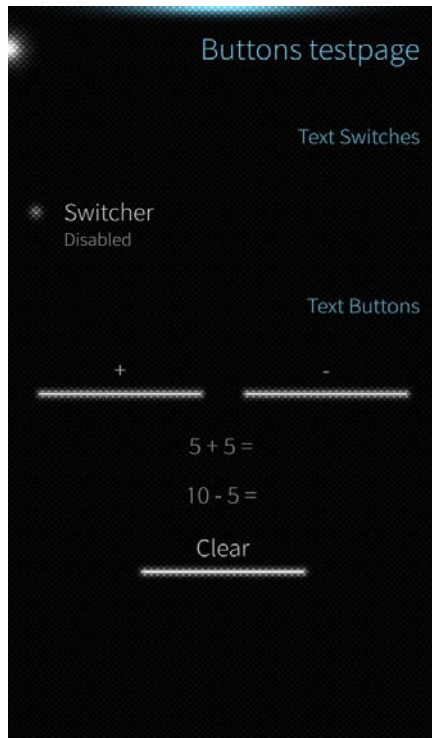


Image 8. Buttons page

In the combo box page, see image 9, different operations was added, like text, busy indicator and progress bar. When choosing one of the options UI shows the corresponding operation.

The ComboBox is a type which provides a label with an attached menu. This label enables the user to select a value from a list of options.



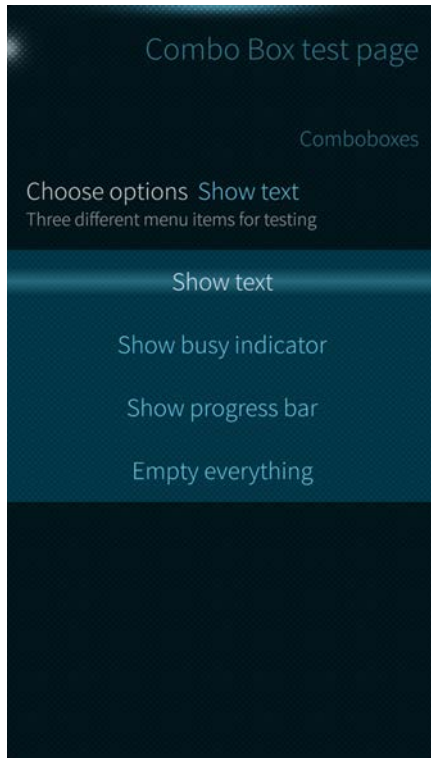


Image 9. Combo box page.

The last page was for text inputs as shown in image 10. On this page the user must add his name to the corresponding fields. After that the user presses the accept button and the name is showed in a new window.

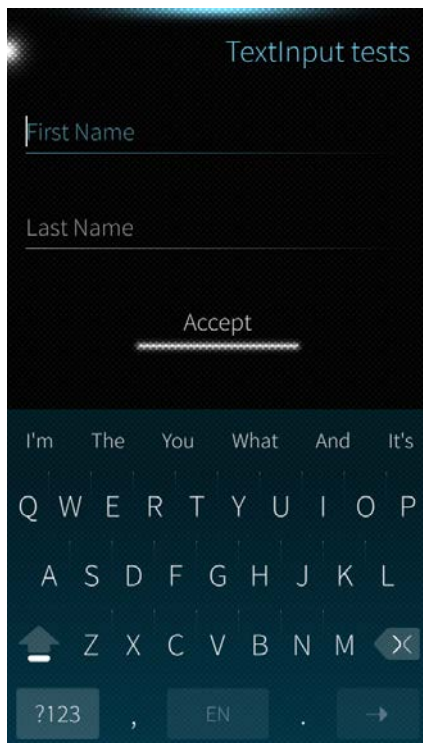


Image 10. Text input page.

### 3.7.1 Building and Installing the Demo Application to the Device

Once the application is ready, developer must build it and then deploy the RPM-package to the device. This is done by various ways. The first option is by choosing MerSDK-SailfishOS-armv7hl kit in project build configuration. In the Build option developer must choose the release, and in the Deploy option developer must choose “Deploy By Building An RPM Package”. For better understanding see image 11.

Project: harbour-qmltestrunner		
Run: harbour-qmltestrunner (on Mer Device)		
Kit	Build	Deploy
MerSDK-SailfishOS-armv7hl	Debug	Deploy As RPM Package
MerSDK-SailfishOS-i486	Release	Deploy By Building An RPM Package
		Deploy By Copying Binaries

Image 11. Project building options

After selecting the correct kit, user should go to Build menu and choose “Deploy Project “harbour-<your\_app\_name>” in Qt Creator as shown on image 12. After successful deploy a prompt window about the path where the application was deployed should be seen, see image 13. The final binary should be under the RPMS folder and should look like this: “harbour-<your\_app\_name>-0.1-1.armv7hl.rpm. This RPM can be submitted to the Harbour.

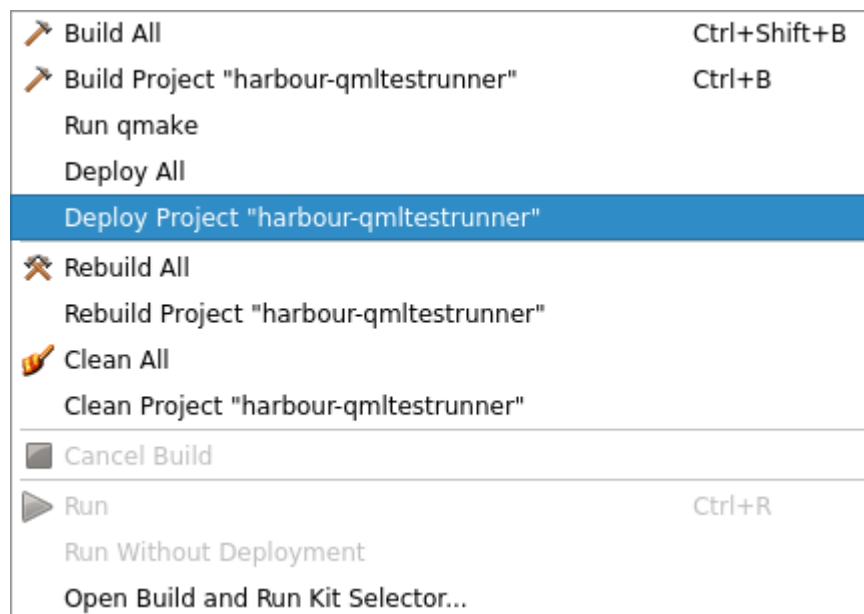


Image 12. Deploying the Project

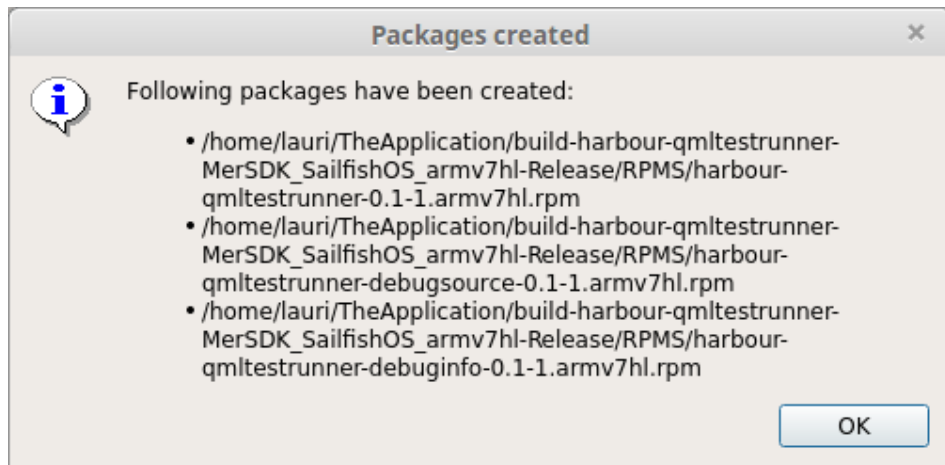


Image 13. Packages paths

A ready RPM can be copied to the device with the ssh-connection or with normal PC-connection. SSH-connection needs the developer-mode to be set and ssh-connection enabled on the device and also root access. To copy the RPM to the device you must use secure copy command, see example 1.

Example 1:

```
scp harbour-qmltestrunner-0.1-1.armv7hl.rpm  
nemo@192.168.2.15:~
```

This command will ask for the password of the device. The password is the same with the ssh connection password. Once the RPM is copied to the device, the ssh connection is needed. After successful connection the RPM can be installed locally. This is done by doing commands from example 2.

Example 2:

```
ssh nemo@192.168.2.15 (will ask the password)  
devel-su (and then enter password again)  
pkcon install-local harbour-qmltestrunner-0.1-1.armv7hl.rpm  
Choose yes by pressing y and then enter
```

Now the application is installed to the device and applications launch icon should be visible on the application grid view.

The second option for installing the RPM into to the device is by connecting your device to the PC and choosing “Deploy as RPM Package” option. For this to work correctly, device type is needed to be set for the MerSDK-SailfishOS-armv7hl. This is done by going to the Tools → Options → Devices within Qt Creator. The Mer ARM Device should be added to the list and not the local PC. See image 14.

When device is set and connected to the PC, green run option comes visible in Qt Creator and the RPM can be deployed directly to the device. This operation will install the application to the device and runs it.

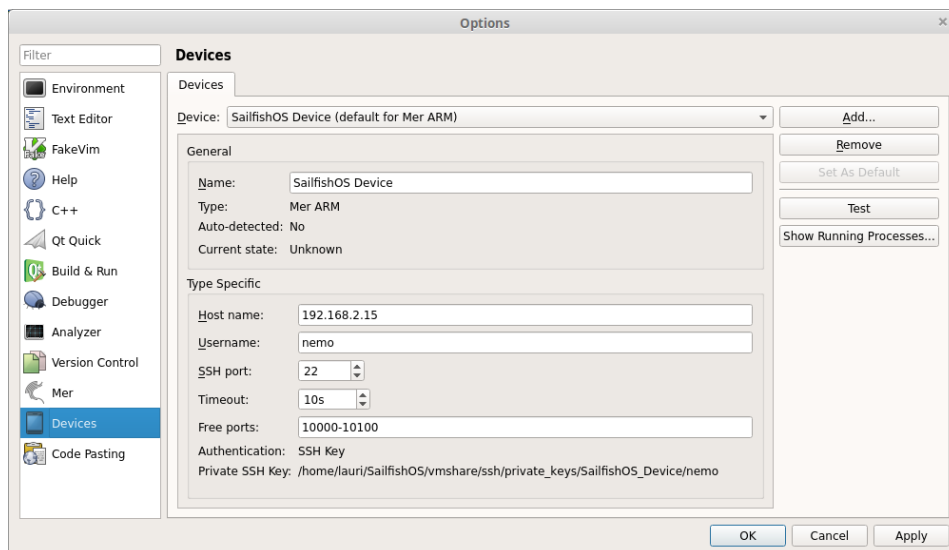


Image 14. Adding device to SailfishSDK

Third install option is by copy-pasting the RPM via normal PC-connection to the device. The device should contain file browser application. The file browser can be installed from the Jolla Store. With file browser, the RPM can installed normally to the device. Untrusted software installation must be allowed.

### 3.8 Sailfish Application Lifecycle

Sailfish is a true multiprocessing system and all of the applications should have two states supported, Active which consumes all the available screen real estate and Background which provides cover for home screen. The application determines which state it is on by using `Qt.application.active` property. The property is true when the application is running in foreground and false if the application is running in the background. If the application is running in the background it mandatory that application minimizes resource usage. All the animations must pause and unused resources released, if possible. (Sailfish OS. 2015e.)

While the application is pushed to the home screen, its cover will automatically be created and displayed. Covers are specified via the `ApplicationWindow.cover` property. When this property is set display displays the current status of the application. Applications may update the cover based on state changes but no animations should be displayed in the cover. When the application becomes active cover goes automatically hidden. (Sailfish OS. 2015e.)

Covers can be also Active Covers. These Active Covers provide instant access to common actions of the application. Via `CoverActionList` right and/or left gestures can be added to the cover. (Sailfish OS. 2015e.)

### 3.9 Summary of “libsailfishapp”

All the third party applications should use libsailfishapp. This eases the development of an application for sailfish, makes sure that paths for the application is correctly set and provides accelerated startup time. This library, libsailfishapp, provides functions to set up the project, installs all the files into right directories and gets important paths at runtime via convenience methods. (Sailfish OS. 2015f.)

To use this library on project, first is needed to add BuildRequires: pkgconfig(sailfishapp) to .spec-file. Then set TARGET = harbour-yourappname into the .pro file, and add CONFIG += sailfishapp into the .pro-file, include sailfishapp.h into the .cpp-file and use the SailfishApp:: methods in the main(). (Sailfish OS. 2015f.) See images 15, 16 and 17.

```

%{!?qtc_qmake:%define qtc_qmake %qmake}
%{!?qtc_qmake5:%define qtc_qmake5 %qmake5}
%{!?qtc_make:%define qtc_make make}
%{?qtc_builddir:%define _builddir %qtc_builddir}
Summary:    Dummy application for using qmltestrunner
Version:    0.1
Release:    1
Group:      Qt/Qt
License:    LICENSE
URL:        http://example.org/
Source0:    %{name}-%{version}.tar.bz2
Source100:  harbour-qmltestrunner.yaml
Requires:   sailfishsilica-qt5 >= 0.10.9
Requires:   qt5-qtdeclarative-import-qttest
Requires:   qt5-qtdeclarative-devel-tools
BuildRequires:  pkgconfig(sailfishapp) >= 1.0.2
BuildRequires:  pkgconfig(Qt5Core)
BuildRequires:  pkgconfig(Qt5Qml)
BuildRequires:  pkgconfig(Qt5Quick)
BuildRequires:  desktop-file-utils

```

Image 15. .spec-file

```
2 # The name of your application
3 TARGET = harbour-qmltestrunner
4
5 CONFIG += sailfishapp
6
7 SOURCES += src/harbour-qmltestrunner.cpp
8
9 OTHER_FILES += qml/harbour-qmltestrunner.qml \
10 qml/cover/CoverPage.qml \
11 qml/pages/FirstPage.qml \
12 rpm/harbour-qmltestrunner.changes.in \
13 rpm/harbour-qmltestrunner.spec \
14 rpm/harbour-qmltestrunner.yaml \
15 translations/*.ts \
16 harbour-qmltestrunner.desktop \
17 qml/pages/ButtonPage.qml \
18 qml/pages/TextPage.qml \
19 qml/pages/ComboPage.qml \
20 qml/pages/special.qml \
21 qml/pages/about.qml \
22 qml/Tests/tst_ButtonTest.qml \
23 qml/Tests/tst_ComboTests.qml \
24 qml/Tests/TestFunctions.qml \
25 qml/Tests/tst_TextInputtest.qml
```

Image 16. .pro-file

```
#ifndef QT_QML_DEBUG
#include <QtQuick>
#endif

#include <sailfishapp.h>

int main(int argc, char *argv[])
{
    // SailfishApp::main() will display "qml/template.qml", if you need more
    // control over initialization, you can use:
    //
    // - SailfishApp::application(int, char *[]) to get the QGuiApplication *
    // - SailfishApp::createView() to get a new QQuickView * instance
    // - SailfishApp::pathTo(QString) to get a QUrl to a resource file
    //
    // To display the view, call "show()" (will show fullscreen on device).

    return SailfishApp::main(argc, argv);
}
```

Image 17. .cpp-file

## 4 TEST AUTOMATION

Automation is manual testing that is done with a testing tool. First you need to have a working manual test environment before you can start automation work. The manual test environment must include detailed test cases and expected results. The expected results come from the requirement specifications and also from the test plans. Automation needs different test environments than manual testing which include test database that can be easily edited and maintained if the test target changes often. (Pohjolainen. 2003.)

With automation you have to get three benefiting factors, repeatability, leverage and accumulation. Repeatability means that test cases can be tested many times without changing the test case itself. Leverage means that the tests are not possible to do manually but can only be done with test automation. Accumulation means that automated testing performs testing with fewer test cases than manual testing needs. Especially when test target changes. (Pohjolainen. 2003.)

Automation is a much quicker way to test the product than manual testing. With automation you can rerun tests much faster and many times if needed at a lower cost. In the best scenario automation has lowered testing costs by about 80%. (Pohjolainen. 2003.)

What test cases are chosen for automation? Usually when starting to automate something, you should focus more on manually time consuming test cases first. (Pohjolainen. 2003.)

There are many different test automation tools. One of the methods is to do test scripts. Test scripts contains commands and instructions for the testing tool. With test scripts you can find problems in early state phase. (Pohjolainen. 2003.)

In this thesis project focus was on two testing tool, testrunner-lite and qmltestrunner. Testrunner-lite is introduced generally and qmltestrunner more detailed.

### 4.1 Unit Testing

Unit testing refers to testing certain functions, areas and units of the code. This gives the ability to verify what functions are working as expected. With unit testing we can determine if certain functions return the proper values while doing inputs. This kind of testing helps us to identify failures in algorithms and helps to improve the quality of the code. If approaching development from a unit testing perspective, you most likely write easy code which is easy to test also. Since unit testing requires code to be easily testable. This means you are most likely to write higher number of smaller and more focused functions in your code. Solid unit tests and well-tested code also gives an advantage for preventing future changes from breaking the functionality. The only disadvantage is that it comes at the expense of investing time to write a suite of tests early in development. (tuts+. 2015.)

### 4.2 Testrunner-lite

Testrunner-lite is a tool used for test executions, which reads xml-files as input. With this tool operations from table 1 can be done.

Execute automatic, semi-automatic and manual test cases
Execute tests locally or in host-based mode
Validate the used test plan file automatically
Use options and filters to select which test cases to execute

Table 1. Different uses for testrunner-lite

Testrunner-lite tests can be executed with following example command:

```
testrunner-lite -f /path/to/the/test/xml-file -o ~/re-
sults.xml -v
```

With testrunner-lite, manual test cases can also be executed. This is defined in the Test Definition in XML-file. Testrunner-lite goes the defined steps through and will ask from the user if the test is a pass or a fail. After the test case is done user can add additional comments about the test. (MerProject. 2015.)

Testrunner-lite has 3 different test case verdicts, Pass, Fail and N/A (MerProject. 2015).

All the test steps are executed in a separate shell. A new process is spawned for execution and waits for the step to be finished or timeout. A cleanup routine is executed once the test case has finished. In cleanup, testrunner-lite tries to kill all processes left running by the test steps. For more information about how-to use testrunner-lite see appendage 3. (MerProject. 2015.)

Example of the xml-file structure for the testrunner-lite:

```
<?xml version="1.0" encoding="UTF-8"?>
<testdefinition version="1.0">
  <suite name="name_for_testsuite" domain="ui">
    <description>Demo app testing</description>
    <set name="UI-test-for-demo-app" feature="Demo app">
      <pre_steps timeout="15">
        <step>"pre steps here"</step>
      </pre_steps>
      <description>Testing demo app on Jolla 1</description>
      <case manual="false" name="tst_ButtonTest" timeout="180">
        <step>"add qmltestrunner execution"</step>
      </case>
      <post_steps>
        <step>"all the post steps here"</step>
      </post_steps>
    </set>
  </suite>
</testdefinition>
```



### 4.3 Qmltestrunner

Qmltestrunner is a tool used for unit testing. This tool allows to execute QML files as test cases. These files should contain `test_functions`. Qmltestrunner is an open-source project and its source code can be found from the github.

#### 4.3.1 Writing Qmltestrunner Tests

There are many ways to write tests for applications and all of them are correct. In this thesis one of the options is chosen.

Once the application is ready, the tests can be added to a Tests-folder in projects working directory. Then with the Qt Creator, a new QML-file is added inside the Tests-folder. Once done, a new clean QML-file is opened on the screen. This created QML-file is going to be the file where all the helper functions is added. These functions are needed for actual tests.

To be able to create tests correct imports are needed. Example 3 shows what imports are needed.

Example 3:

```
import QtQuick 2.0
import QtTest 1.0
import Sailfish.Silica 1.0
```

After imports, it is needed to add the helper functions inside the `TestCase` type. This `TestCase` type represents a unit test case. JavaScript functions are used when creating functions within `TestCase` type. See the example 4. It is unpredictable how these test functions are found because of the JavaScript properties. Test framework assists the predictability by sorting the functions on ascending order of name. This helps if two test is needed to run in order. (`TestCase` QML Type. 2015.)

Example 4:

```
TestCase {
    function click_center(item) {
        var pos = main.mapFromItem(item, item.width/2,
item.height/2)
        testEvent.mouseClick(main, pos.x, pos.y, Qt.LeftButton,
0, 0)
    }
}
```

Once the helper functions are done, actual test can be added by creating QML-files inside the Test-folder. The best naming for these QML-files are adding “`tst_`” or “`test_`” front of the test name example “`tst_Button-Tests.qml`”. After the creation of the test file it is critical that the correct test target is opened for testing. This is done by adding a launch action inside the `ApplicationWindow` type. This type is used to create the top-level item in Sailfish application. Every Sailfish application must have this component

defined at the root of its hierarchy. This application window is the entry point for the loading of the app. Inside of this `ApplicationWindow` type, the user must add `initialPage` variant that specifies the page that is displayed when the application is opened. See example 5 for use of `ApplicationWindow` and `initialPage` on test file.

Example 5:

```
ApplicationWindow {
    //here we launch the application on certain page
    id: main
    initialPage: Qt.resolvedUrl("../pages/ButtonPage.qml")
}
```

Before doing any of the test actions we need to inherit the helper QML-file so helper functions can be used in test creation. This is done by adding the name of the helper function file to the test file without the `.qml` part. For example, if the helper filename is `TestFunctions.qml` then this is inherited by adding `TestFunctions { }` within the `ApplicationWindow`. See the example 6 for full start of the test file.

Example 6:

```
ApplicationWindow {
    //here we launch the application on certain page
    id: main
    initialPage: Qt.resolvedUrl("../pages/ButtonPage.qml")

    //Test are written always inside the TestFunctions object.
    //Name must be the same with the TestFunctions.qml file name.

    TestFunctions {
        name: "Button tests"
        when: windowShown

        //here is the test itself
        function test_buttons(){
        }
    }
}
```

Once all the mandatories are set, tests can be made as a functions within the `TestFunctions`, see example 4.

### 4.3.2 Executing the Qmltestrunner Tests on Sailfish OS Device

Before tests can be run, the `qmltestrunner` must be installed to the device. This is done by taking a `ssh` connection to the device and as root user installing two packages, `qt5-qtdeclarative-import-qttest` and `qt5-qtdeclarative-devel-tools` to the device, see the example 7. After a successful install `qmltestrunner` is installed under `/usr/lib/qt5/bin/`. Do note though these packages can be also installed with one command.

Example 7:

```
pkcon install qt5-qtdeclarative-import-qttest
pkcon install qt5-qtdeclarative-devel-tools
```

When executing tests, the user should go to the place where the tests are installed. On the demo app it is `/usr/share/harbour-qmltestrunner/qml/Tests/`.

Once at the Tests-folder next command will execute the test:

```
invoker --type=generic /usr/lib/qt5/bin/qmltestrunner -input
<name of the test file>
```

Or if wanted to execute all the tests at once:

```
invoker --type=generic /usr/lib/qt5/bin/qmltestrunner -input
/usr/share/<harbour-your_app_name>/qml/Tests/
```

Tests can be executed also in any place on the device. The command for this is:

```
cd /usr/share/harbour-<your_app_name>/qml/Tests/ && invoker
--type=generic /usr/lib/qt5/bin/qmltestrunner -input <name
of the test file>
```

Or if wanted to execute all of the tests at once:

```
cd /usr/share/harbour-<your_app_name>/qml/Tests/ && invoker
--type=generic /usr/lib/qt5/bin/qmltestrunner -input
/usr/share/harbour-<your_app_name>/qml/Tests/
```

More information about qmltestrunner usage can be found in appendage 1.

Invoker is a tool which is needed for launching the testing tool qmltestrunner. More information on how-to use invoker can be found on appendage 2.

Tests can be run as a nemo-user (normal user) or as a root user. It is recommended thought to run tests as a nemo user since the normal user runs applications as a nemo user. When executing the test progress can be seen on the devices screen and also in the command line (terminal). After full test run the results can be also seen in the terminal window as image 18 shows.

```
invoker: Invoking execution: '/usr/lib/qt5/bin/qmltestrunner'
[0] QWaylandEglClientBufferIntegration::QWaylandEglClientBufferIntegration:62 - Using Wayland-EGL
***** Start testing of qmltestrunner *****
Config: Using QTest library 5.2.2, Qt 5.2.2
PASS : qmltestrunner::Button tests::initTestCase()
QDEBUG : qmltestrunner::Button tests::test_buttons() [D] test_buttons:21 - Enabled
QDEBUG : qmltestrunner::Button tests::test_buttons() [D] test_buttons:33 - Disabled
PASS : qmltestrunner::Button tests::test_buttons()
PASS : qmltestrunner::Button tests::cleanupTestCase()
QWARN : qmltestrunner::UnknownTestFunc() [W] unknown:57 - file:///usr/share/harbour-qmltestrunner/qml
PASS : qmltestrunner::Combo Tests::initTestCase()
QDEBUG : qmltestrunner::Combo Tests::test_comboboxes() [D] test_comboboxes:18 - Show text
QDEBUG : qmltestrunner::Combo Tests::test_comboboxes() [D] test_comboboxes:19 - Show busy indicator
QDEBUG : qmltestrunner::Combo Tests::test_comboboxes() [D] test_comboboxes:20 - Show progress bar
QDEBUG : qmltestrunner::Combo Tests::test_comboboxes() [D] test_comboboxes:21 - Empty everything
PASS : qmltestrunner::Combo Tests::test_comboboxes()
PASS : qmltestrunner::Combo Tests::cleanupTestCase()
PASS : qmltestrunner::Text Input tests::initTestCase()
QDEBUG : qmltestrunner::Text Input tests::test_TextInputs() [D] test_TextInputs:37 - Tester Test
PASS : qmltestrunner::Text Input tests::test_TextInputs()
PASS : qmltestrunner::Text Input tests::cleanupTestCase()
Totals: 9 passed, 0 failed, 0 skipped
***** Finished testing of qmltestrunner *****
```

Image 18. Qmltestrunner results

### 4.3.3 Executing the Qmltestrunner Tests on Sailfish OS Emulator

If the tester does not have a device for testing, tests can be run on Sailfish OS Emulator which is provided with SailfishSDK. Also it is highly recommended to run tests with emulator rather than on real device. If done something wrong there is a change to damage the device you are using as a daily device.

First is needed to take a ssh connection to the emulator with following command:

```
ssh -p 2223 nemo@localhost
```

Then installing the two packages mentioned in chapter 4.3.2. These packages can be installed as a normal user, nemo in the emulator. It is critical to have the emulator running on the virtual machine before trying to take a ssh connection to it. Once at the emulator, the password is required to be set for the ssh connection. This is done by opening the settings application and going to the System settings>developer settings –page. The page should look the same as image 19.

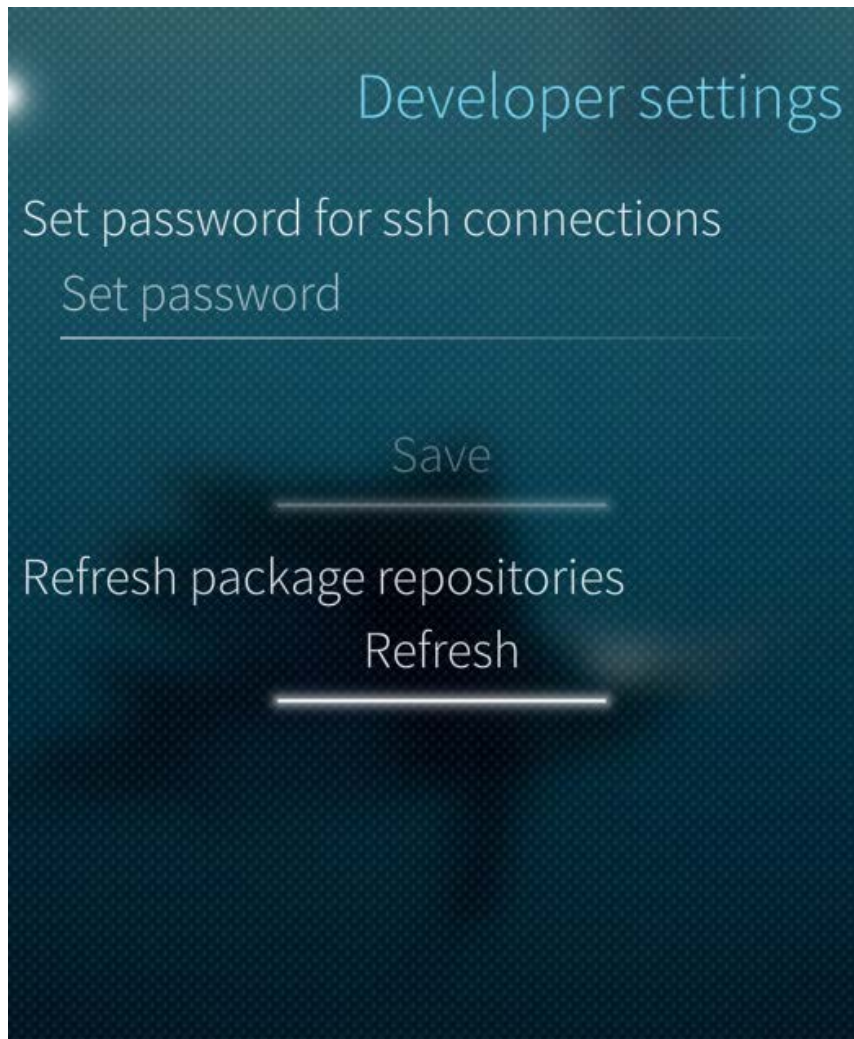


Image 19. Developer settings –page on emulator

Now the emulator is ready for deployment. The deployment is done by choosing MerSDK-SailfishOS-i486 as a kit option when choosing project build configuration, see image 10 for better understanding. After a successful deployment, the program can be run on the emulator. Testing can be started now with the same commands as used in real device. The progress can be seen in the emulator screen.

### 4.3.4 Reading the results

When all the test are run, the results can be seen in the terminal window. First lines in the results indicates that testing is successfully started. After successful start of the qmltestrunner, first test case is started. This is shown on example 8.

Example 8:

```
***** Start testing of qmltestrunner *****  
Config: Using QTest library 5.2.2, Qt 5.2.2  
PASS    : qmltestrunner::Button tests::initTestCase()
```

When the test case is run, qmltestrunner gives verdict and starts cleanup process for the test case, see the example 9.

Example 9:

```
PASS    : qmltestrunner::Button tests::test_buttons()
PASS    : qmltestrunner::Button tests::cleanupTestCase()
```

If debug items has been set inside the test script file, those are also shown in results, see the example 10.

Example 10:

```
QDEBUG  : qmltestrunner::Button tests::test_buttons() [D]
test_buttons:21 - Enabled
QDEBUG  : qmltestrunner::Button tests::test_buttons() [D]
test_buttons:33 - Disabled
```

When results of the test case is a fail. It shows like shown in the example 11. Loc entry in the example 11 indicates the test case file where the error is and also the number at the end of the “Loc” line points to corresponding line of the test case file.

Example 11:

```
FAIL!   : qmltestrunner::Button tests::test_buttons() Uncaught
exception:
Cannot read property 'width' of undefined
Loc:    [ /usr/share/harbour-qmltestrunner/qml/Tests/TestFunc-
tions.qml(46) ]
```

When the full testing run is done total results are shown and testing has been stopped, see the example 12.

Example 12:

```
Totals: 9 passed, 0 failed, 0 skipped
***** Finished testing of qmltestrunner *****
```

## 5 SUMMARY

This Thesis should help community members to build and run tests for their own applications. With this thesis work, UI-errors should be going down. This helps the store testers to test the applications a bit faster rather than pointing out all the minor UI-errors seen in the application. This also makes the quality of the application much higher.

Thesis contains two practical part. The first one was to create demo application for testing and second one was to make document for how to write and run qmltestrunner tests. The application was done and it is working correctly with the test. This application includes really basic features of the Sailfish applications. Currently the application is not submitted to the Jolla Store since there is import QtTest 1.0 which is not allowed according to Jolla Harbour rules. Also there is two dependencies which are not allowed to the Store, qt5-qtdeclarative-import-qttest and qt5-qtdeclarative-devel-tools. This was solved by adding about page inside the demo application saying how-to install these packages to the device. To get the demo application to Jolla Store, it is needed to talk with Jolla Store representative to allow and make an exception with this demo application. The second practical work, documentation about how to write and run qmltestrunner tests, was also made ready and it is going to the mer-wiki page when the correct technical language checks are made. Mer-wiki is website page where all the information about mer and how-to is gathered so community can understand what mer is and contribute to it. This documentation is going to be ported to mer-wiki page after this thesis project.

First hard blockers during this project was, what kind of application should be made to demonstrate automated testing. This was solved by doing a dummy application that contains most of the common features. After completing the demo application, second blocking issue was how to write the tests inside the demo application so that they can be run from command line successfully. This needed some additional understanding about launching the application before starting the tests. Solution was to add initialPage component inside the ApplicationWindow type. When the application was set to open on certain page before tests starts it was easy to write the tests for the demo application. Some difficulties was on finding the correct text labels from the UI. These text labels should match with the text set on tests. To find the correct text labels, self-made debug function was used to find the correct text label. The debug function listed all visible items from the current page to the terminal window. From the list it is easy to find all what is needed for creating the tests.

All in all, thesis project was successful. The demo application demonstrated the qmltestrunner usage in a basic way. This was the goal for this thesis work.

## SOURCES

- Sailfish OS. 2015a. About. Sailfish OS. Referenced 20.07.2015. <https://sailfishos.org/about/>
- The Qt Company Ltd. 2015a. About. Qt. Referenced 23.07.2015. [http://wiki.qt.io/About\\_Qt](http://wiki.qt.io/About_Qt)
- Sailfish OS. 2015b. Jolla. Harbour. Referenced 23.07.2015. <https://sailfishos.org/develop/harbour>
- The Qt Company Ltd. 2015b. Qt Quick. Tutorial. Referenced 24.07.2015. [http://wiki.qt.io/Qt\\_Quick\\_Tutorial](http://wiki.qt.io/Qt_Quick_Tutorial)
- Sailfish OS. 2015c. Creating applications with Sailfish Silica. Referenced 15.09.2015. <https://sailfishos.org/develop/docs/silica/>
- The Qt Company Ltd. 2015c. Qt Documentation. TestCase QML Type. Referenced 20.08.2015. <http://doc.qt.io/qt-5/qml-qttest-testcase.html>
- Sailfish OS. 2015d. Software Development Kit. Referenced 28.07.2015. <https://sailfishos.org/develop/sdk-overview>
- Sailfish OS. 2015e. Sailfish Application Lifecycle. Referenced 15.09.2015. <https://sailfishos.org/develop/docs/silica/sailfish-application-lifecycle.html/>
- Sailfish OS. 2015f. libsailfishapp documentation. Referenced 15.09.2015. <https://sailfishos.org/develop/docs/libsailfishapp/>
- Image 1. SailfishOS. SailfishOS modular system. Referenced 20.07.2015. Available at <https://sailfishos.org/about/>
- Image 2. SailfishOS. Submit-page on Jolla Harbour. Available at <https://sailfishos.org/develop/harbour>
- Jolla Oy. 2015. About. Jolla Ltd. Referenced 23.05.2015. <https://jolla.com/about/>
- MerProject. 2015. Mer-wiki. Referenced 17.08.2015 <https://wiki.merproject.org/wiki/Quality/QA-tools/Testrunner-lite>
- Pohjolainen, P. 2003. Ohjelmiston testauksen automatisointi. Kuopion yliopisto. Informaatioteknologian ja kauppatieteiden tiedekunta. Tietojenkäsittelytieteen laitos. Tietojenkäsittelytiede. Pro gradu -tutkielma.
- tuts+. 2015. What Is Unit Testing? Referenced 21.08.2015. <http://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728>



### QMLTESTRUNNER USAGE

Usage: /usr/lib/qt5/bin/qmltestrunner [options]  
[testfunction[:testdata]]...

By default, all testfunctions will be run.

New-style logging options:

-o filename,format :

Output results to file in the specified format

Use - to output to stdout

Valid formats are:

txt : Plain text

xunitxml : XML XUnit document

xml : XML document

lightxml : A stream of XML tags

\*\*\* Multiple loggers can be specified, but at most one can log to stdout.

Old-style logging options:

-o filename : Write the output into file

-txt : Output results in Plain Text

-xunitxml : Output results as XML XUnit document

-xml : Output results as XML document

-lightxml : Output results as stream of XML tags

\*\*\* If no output file is specified, stdout is assumed.

\*\*\* If no output format is specified, -txt is assumed.

Test log detail options:

-silent : Log failures and fatal errors only

-v1 : Log the start of each testfunction

-v2 : Log each QVERIFY/QCOMPARE/QTEST (implies -v1)

-vs : Log every signal emission and resulting slot invocations

\*\*\* The -silent and -v1 options only affect plain text output.

Testing options:

-functions : Returns a list of current testfunctions

-datatags : Returns a list of current data tags.

A global data tag is preceded by ' \_\_global\_\_ '.

-eventdelay ms : Set default delay for mouse and keyboard simulation to ms milliseconds

-keydelay ms : Set default delay for keyboard simulation to ms milliseconds

-mousedelay ms : Set default delay for mouse simulation to ms milliseconds

-maxwarnings n : Sets the maximum amount of messages to output. 0 means unlimited, default: 2000

-nocrashhandler : Disables the crash handler

## Writing and Running Qmltestrunner Tests

---

### Benchmarking options:

-callgrind : Use callgrind to time benchmarks  
-perf : Use Linux perf events to time benchmarks  
-perfcounter name : Use the counter named 'name'  
-perfcounterlist : Lists the counters available  
-eventcounter : Counts events received during benchmarks  
-minimumvalue n : Sets the minimum acceptable measurement value  
-minimumtotal n : Sets the minimum acceptable total for repeated executions of a test function

-iterations n : Sets the number of accumulation iterations.  
-median n : Sets the number of median iterations.  
-vb : Print out verbose benchmarking information.

### QmlTest options:

-import dir : Specify an import directory.  
-input dir/file : Specify the root directory for test cases or a single test case file.

-qtquick1 : Run with QtQuick 1 rather than QtQuick 2.  
-translation file : Specify the translation file.  
-help : This help

### INVOKER USAGE

Usage: invoker [options] [--type=TYPE] [file] [args]

Launch applications compiled as a shared library (-shared) or a position independent executable (-pie) through map-launcherd.

TYPE chooses the type of booster used. Qt-booster may be used to

launch anything. Possible values for TYPE:

qt5	Launch a Qt 5 application.
qtquick2	Launch a Qt Quick 2 (QML) application.
silica-qt5	Launch a Sailfish Silica application.
generic	Launch any application, even if it's not a library.

Options:

-d, --delay SECS

After invoking sleep for SECS seconds (default 0).

-r, --respawn SECS

After invoking respawn new booster after SECS seconds (default 1, max 10).

-w, --wait-term

Wait for launched process to terminate (default).

-n, --no-wait

Do not wait for launched process to terminate.

-G, --global-syms

Places symbols in the application binary and its libraries to the global scope. See RTLD\_GLOBAL in the dlopen manual page.

-s, --single-instance

Launch the application as a single instance. The existing application window will be activated if already launched.

-o, --keep-oom-score

Notify invoker that the launched process should inherit oom\_score\_adj from the booster. The score is reset to 0 normally.

-T, --test-mode

Invoker test mode. Also control file in root home should be in place.

-F, --desktop-file

Desktop file of the application.

-h, --help

Print this help.

Example: invoker --type=qt5 /usr/bin/helloworld

## TESTRUNNER-LITE USAGE

```
/usr/bin/testrunner-lite [options]
Options

-h, --help
Show this help message and exit.

-V, --version
Display version and exit.

-f FILE, --file=FILE
Input file with test definitions in XML (required).

-o FILE, --output=FILE
Output file for test results (required).

-r FORMAT, --format=FORMAT
Output file format. FORMAT can be xml or text. Default: xml

-e ENVIRONMENT, --environment=ENVIRONMENT
Target test environment. Default: hardware

-v, -vv, --verbose[={INFO|DEBUG}]
Enable verbosity mode; -v and --verbose=INFO are equivalent
outputting INFO, ERROR and WARNING messages.
Similarly -vv and --verbose=DEBUG are equivalent, outputting
also debug messages. Default behaviour is silent mode.

-L, --logger=URL
Remote HTTP logger for log messages. Log messages are sent to
given URL in a HTTP POST message.
URL format is [http://]host[:port][/path/], where host may be
a hostname or an IPv4 address.

-a, --automatic
Enable only automatic tests to be executed.

-m, --manual
Enable only manual tests to be executed.

-l FILTER, --filter=FILTER
Filtering option to select tests (not) to be executed. E.g.
'-testcase=bad_test -type=unknown' first disables test case
named as bad_test. Next, all tests with type unknown are
disabled. The remaining tests will be executed. (Currently
supported filter type are: testset, testcase, requirement,
feature and type)

-c, --ci
Disable validation of test definition against schema.

-s, --semantic
Enable validation of test definition against stricter (se-
mantics) schema.

-A, --validate-only
Do only input xml validation, do not execute tests.
```

## Writing and Running Qmltestrunner Tests

---

`-H, --no-hwinfo`  
Skip hwinfo obtaining.

`-P, --print-step-output`  
Output standard streams from programs started in steps

`-S, --syslog`  
Write log messages also to syslog.

`-M, --disable-measurement-verdict`  
Do not fail cases based on measurement data

`--measure-power`  
Perform current measurement with `hat_ctrl` tool during execution of test cases

`-u URL, --vcs-url=URL`  
Causes `testrunner-lite` to write the given VCS URL to results.

`-U URL, --package-url=URL`  
Causes `testrunner-lite` to write the given package URL to results.

`--logid=ID`  
User defined identifier for HTTP log messages.

`-d PATH, --rich-core-dumps=PATH`  
Save rich-core dumps. `PATH` is the location, where rich-core dumps are produced in the device. Creates UUID mappings between executed test cases and generated rich-core dumps. This makes possible to link each rich-cores and test cases in test reporting NOTE: This feature requires working `sp-rich-core` package to be installed in the Device Under Test.

Test commands are executed locally by default. Alternatively, one of the following executors can be used:

**Chroot Execution:**  
`-C PATH, --chroot=PATH`  
Run tests inside a chroot environment. Note that this doesn't change the root of the `testrunner` itself, only the tests will have the new root folder set.

**Host-based SSH Execution:**  
`-t [USER@]ADDRESS[:PORT], --target=[USER@]ADDRESS[:PORT]`  
Enable host-based testing. If given, commands are executed from test control PC (host) side. `ADDRESS` is the ipv4 address of the system under test. Behind the scenes, host-based testing uses the external execution described below with SSH and SCP.

`-R[ACTION], --resume[=ACTION]`  
Resume `testrun` when ssh connection failure occurs. The possible ACTIONS after resume are:  
`exit` Exit after current test set  
`continue` Continue normally to the next test set  
The default action is 'exit'.

`-i [USER@]ADDRESS[:PORT], --hwinfo-target=[USER@]ADDRESS[:PORT]`  
Obtain hwinfo remotely. Hwinfo is usually obtained locally or in case of host-based testing from target address. This option

## Writing and Running Qmltestrunner Tests

---

overrides target address when hwinfo is obtained. Usage is similar to -t option.

-k KEY, --ssh-key=KEY  
path to SSH private key file

Libssh2 Execution:

-n [USER@]ADDRESS, --libssh2=[USER@]ADDRESS  
Run host based testing with native ssh (libssh2) EXPERIMENTAL

External Execution:

-E EXECUTOR, --executor=EXECUTOR  
Use an external command to execute test commands on the system under test. The external command must accept a test command as a single additional argument and exit with the status of the test command. For example, an external executor that uses SSH to execute test commands could be "/usr/bin/ssh user@target".

-G GETTER, --getter=GETTER

Use an external command to get files from the system under test. The external getter should contain <FILE> and <DEST> (with the brackets) where <FILE> will be replaced with the path to the file on the system under test and <DEST> will be replaced with the destination directory on the host. If <FILE> and <DEST> are not specified, they will be appended automatically. For example, an external getter that uses SCP to retrieve files could be "/usr/bin/scp target:'<FILE>' '<DEST>'".