Developing with SailfishOS

a short introduction

Sven Putze, hardcodes.de

December 2, 2013

Contents

1 About

Hi, my name is Sven[hc01] and I am eager to develop for the Jolla smartphone. On my way some questions came up and I tried to answer them as good as I can. After a while I decided to write down what I've learned, so that I had a central place to come back to and maybe others can benefit from this small document, too. The latest and greatest version is to be found on Github[hc02].

This goes only for content I have written myself, I don't claim ownership or copyright/left on cited content. Content from other parties remains under its original license.

All mentioned trademarks and trade names are the property of their respective holders, I have not marked them with a ™, ® or © sign. Use some common sense here. Not all information is written by myself, I've tried to quote as responsible as possible and quite intensive if appropiate. Read these notes as a human being, not like a lawyer.

If you find typos, errors or quirks, have suggestions how to make this document better, please drop me a note. Or collaborate. `#jolla2gether`!

Don't panic if you are not comfortable with writing in LaTeX, I am happy to use your Libre/Open Office or even Word documents.

2 Download and install

You will need Virtual Box, because some components of the SailfishOS SDK come as virtual machines. Download for your development machine[vbox01].

Figure 1: Download VirtualBox from the virtualbox website.

If you already use VirtualBox, you don't need to load it again, just skip that step. Just make sure that you have the latest updates installed.

To take a dip, head over to the Sailfish Website[sailfishos01] and download the SDK for your operating system.

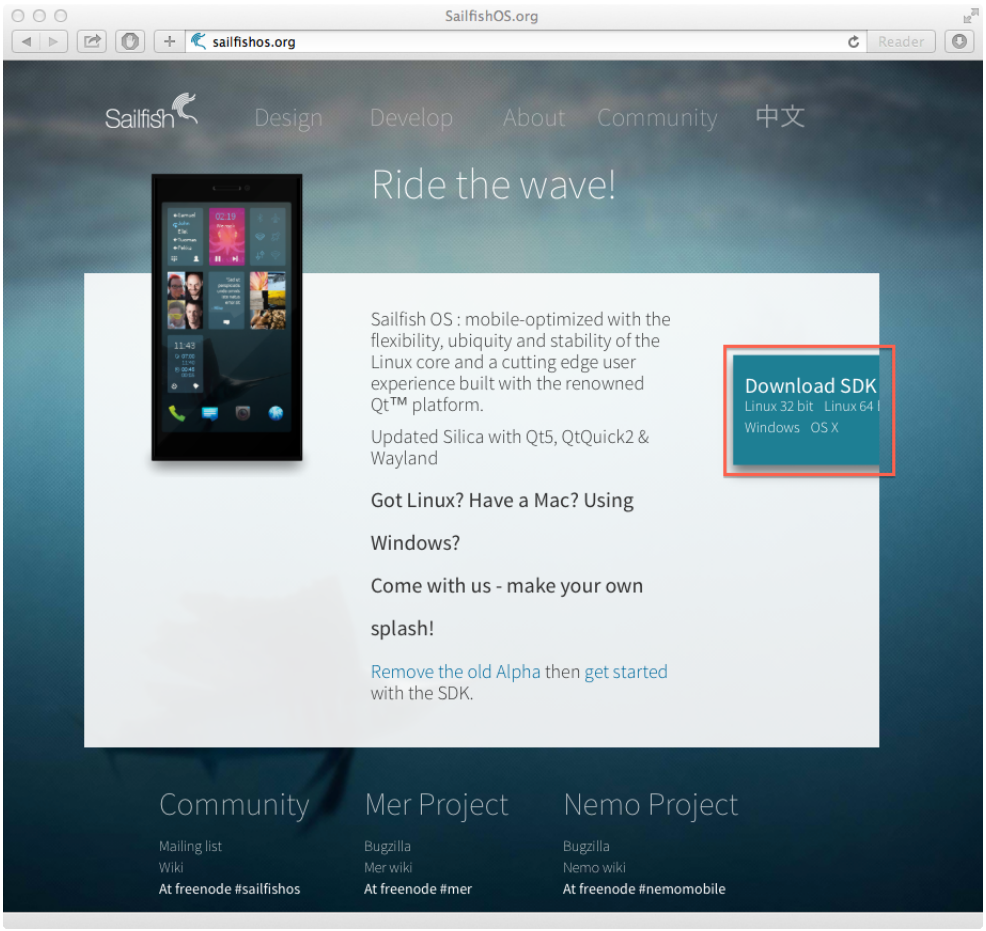


Figure 2: Download the SDK from the SailfishOS website.

2.1 OSX

Double click on the downloaded disk image for VirtualBox and run the installer application inside. Some file go into system folders and you must be or elevate to an admin account to install successfully.

Figure 3: Downloaded diskimage.

If you use VirtualBox just for the SailfishOS SDK you don't have to care about the VirtualBox application, although you can see which folders are shared inside the preferences.

Double click on the downloaded disk image for the SailfishOS SDK and run the installer app that's inside. The installed files will end in your user directory, you don't need to be an administrator to achieve that.

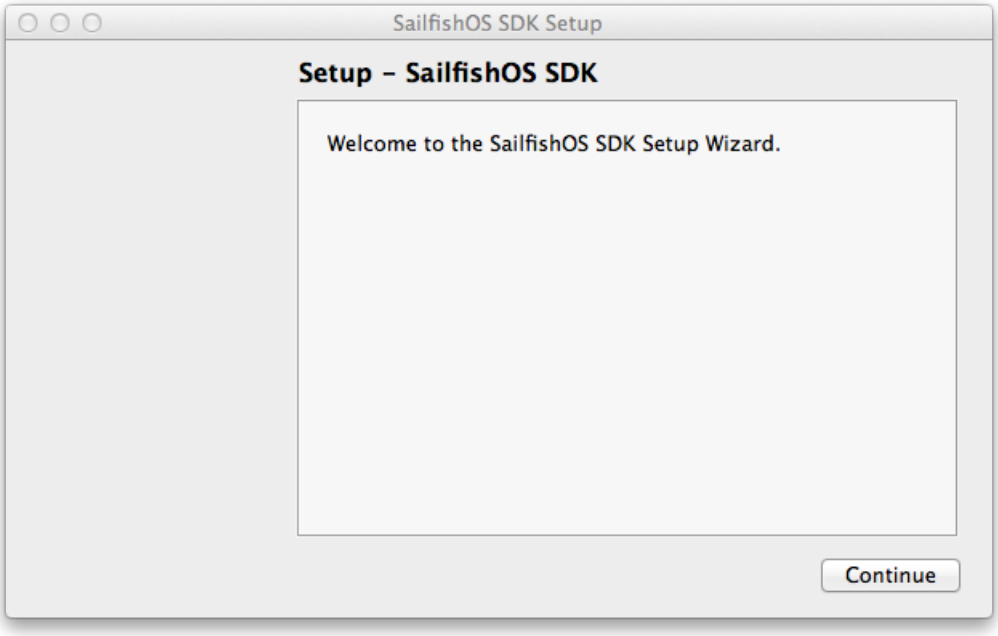


Figure 4: Install SailfishOS SDK, step 1.

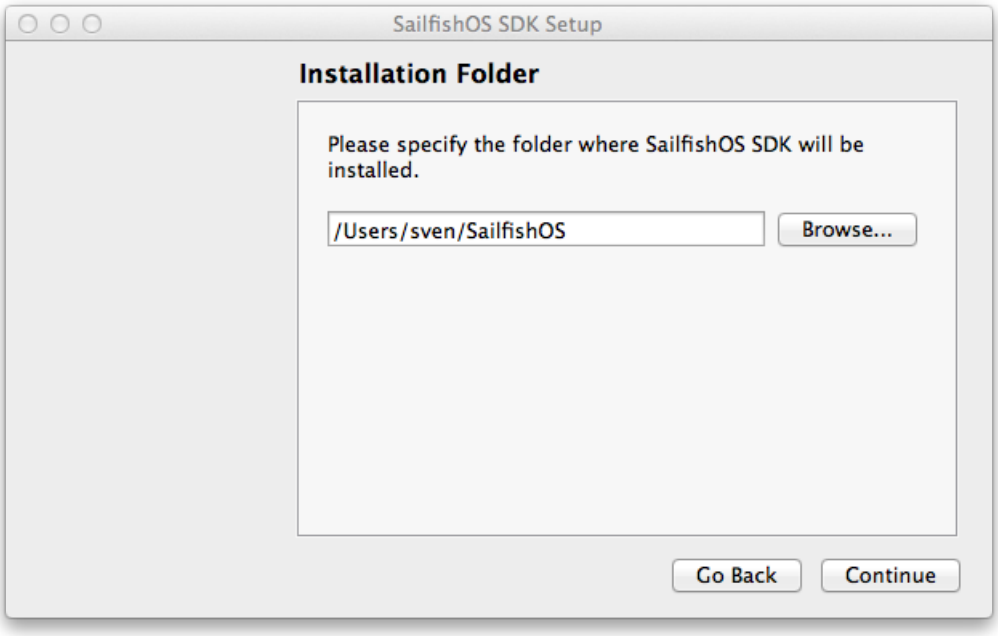


Figure 5: Install SailfishOS SDK, step 2.

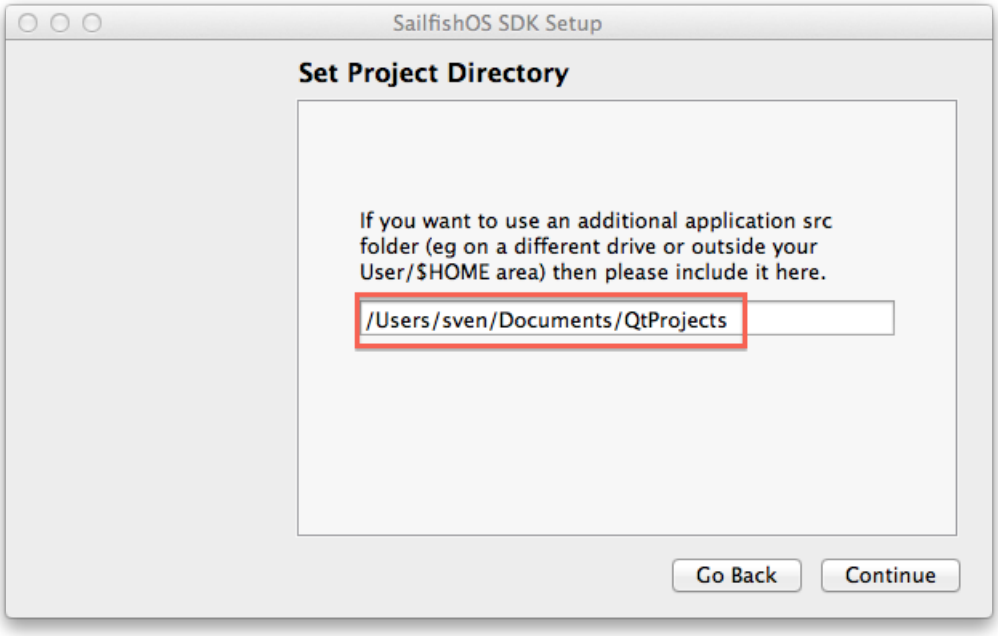


Figure 6: Install SailfishOS SDK, step 3. Enter the path to your source code. There is no folder selector dialog, you must enter it by hand.

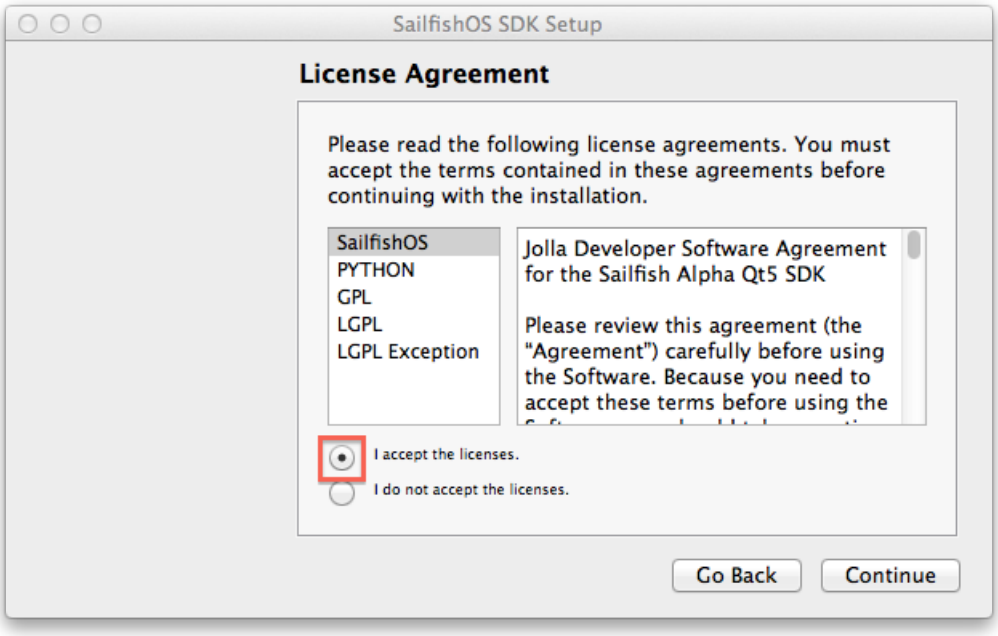


Figure 7: Install SailfishOS SDK, step 4.

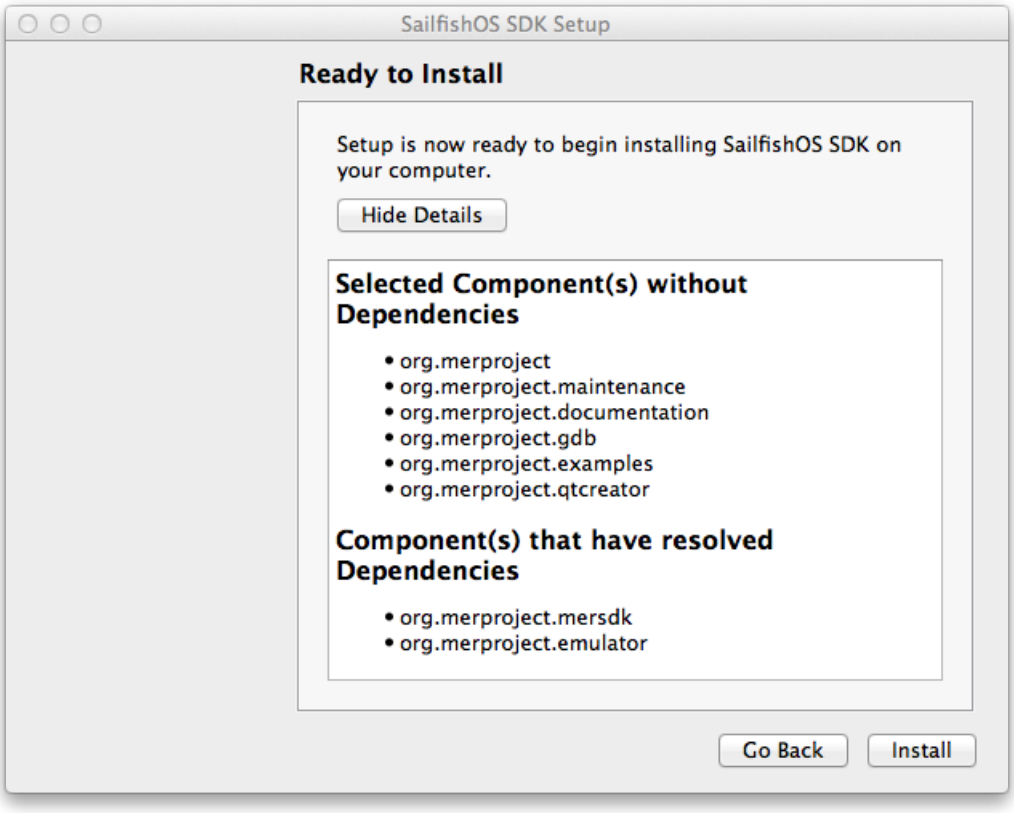


Figure 8: Install SailfishOS SDK, step 5.

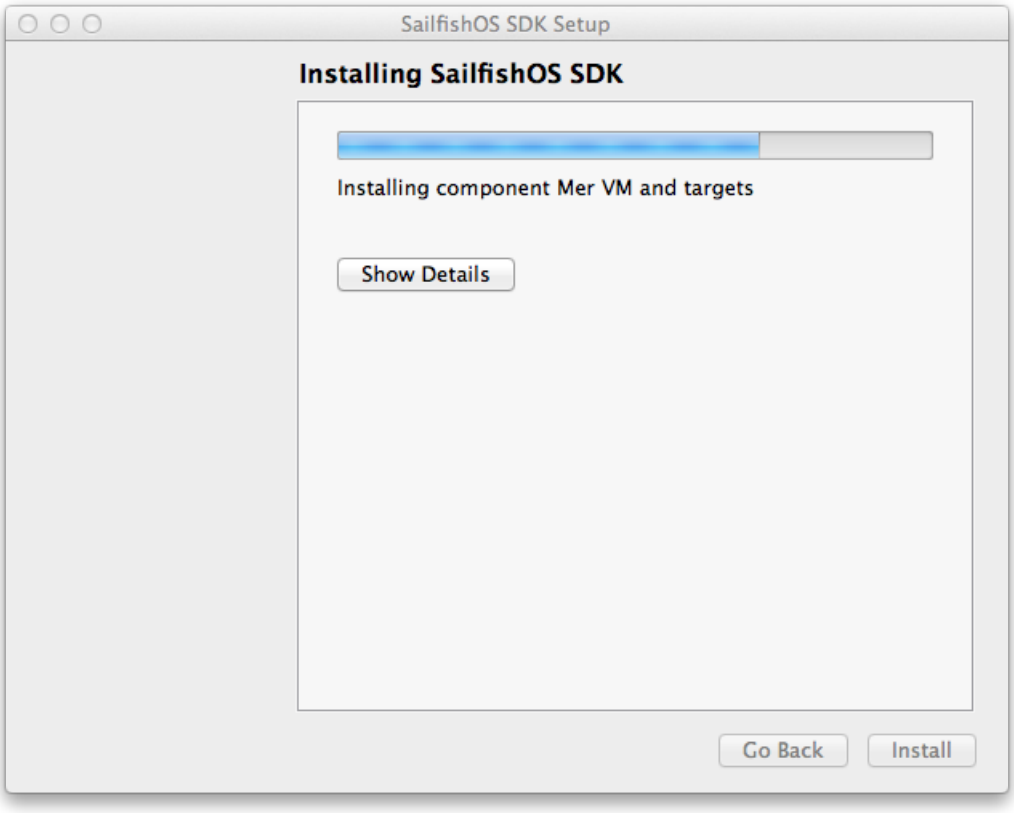


Figure 9: Install SailfishOS SDK, step 6.

Figure 10: Install SailfishOS SDK, step 7.

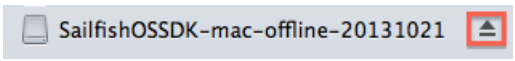


Figure 11: Unmount SailfishOS SDK disk image, step 8.

With the installation came two hidden directories, you should know about. More about those directories will follow later on.

```
$HOME/.config/SailfishAlpha2
$HOME/.scratchbox2
```

After you installed the SDK, you should immediately update the components. As of now the progress inside Jolla is at good pace, so it might be that there is some stuff slightly out of date in the installer (see figures 12 and 13).

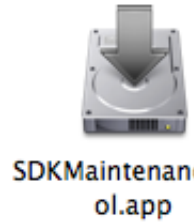


Figure 12: Inside the SailfishOS folder you find the maintenance application. Run it directly after the installation.

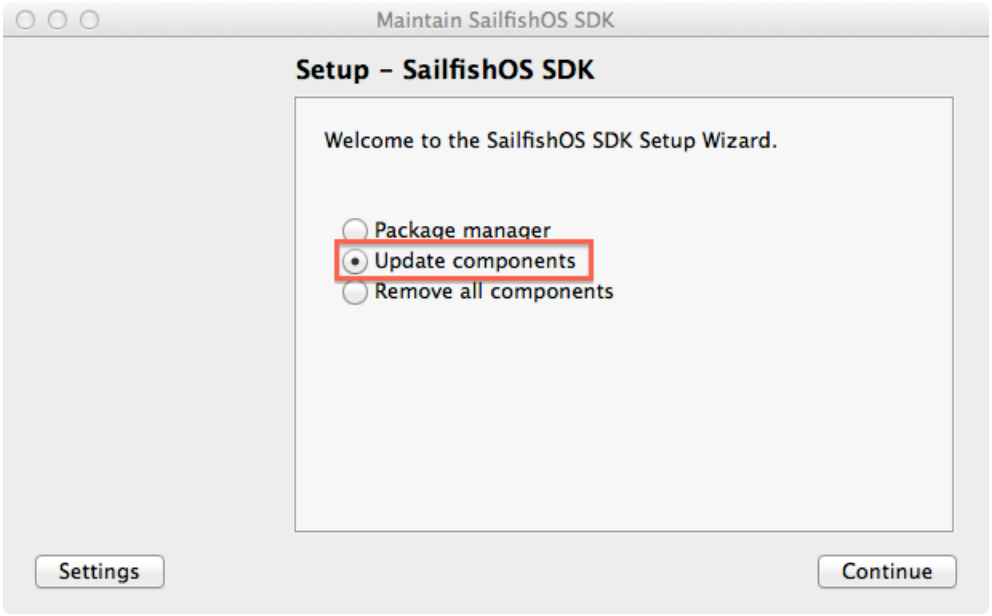


Figure 13: Update components, choose everything that is in store.

With the SDK comes QtCreator[1], a complete IDE for C++ development. This IDE is part of the Qt Framework[qt01] and is simply reused[2]by Jolla. Personally I use the QtCreator on my machines as well and for better differentiation I made a custom icon for the one in the SailfishOS SDK - feel free to download and use it, too[hc03].

[1]Look in in the "bin" folder of the SDK.
[2]Customized with some little tweaks to suite the SailfishOS development.

Figure 14: Alternative icon for the QtCreator inside the SDK.

2.1.1 Remove plugins

If you want to improve the startup time of QtCreator, you can deactivate plugins you don't need or want. Just don't shoot in your foot here.

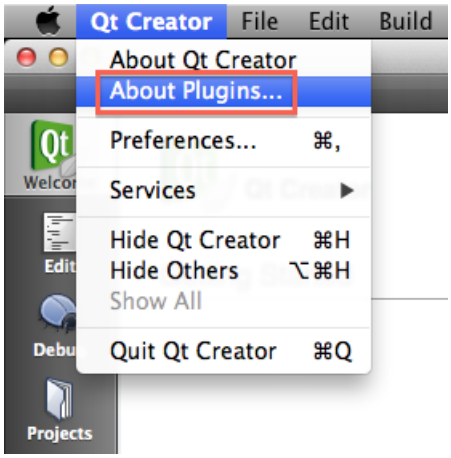


Figure 15: About plugins = manage plugins.

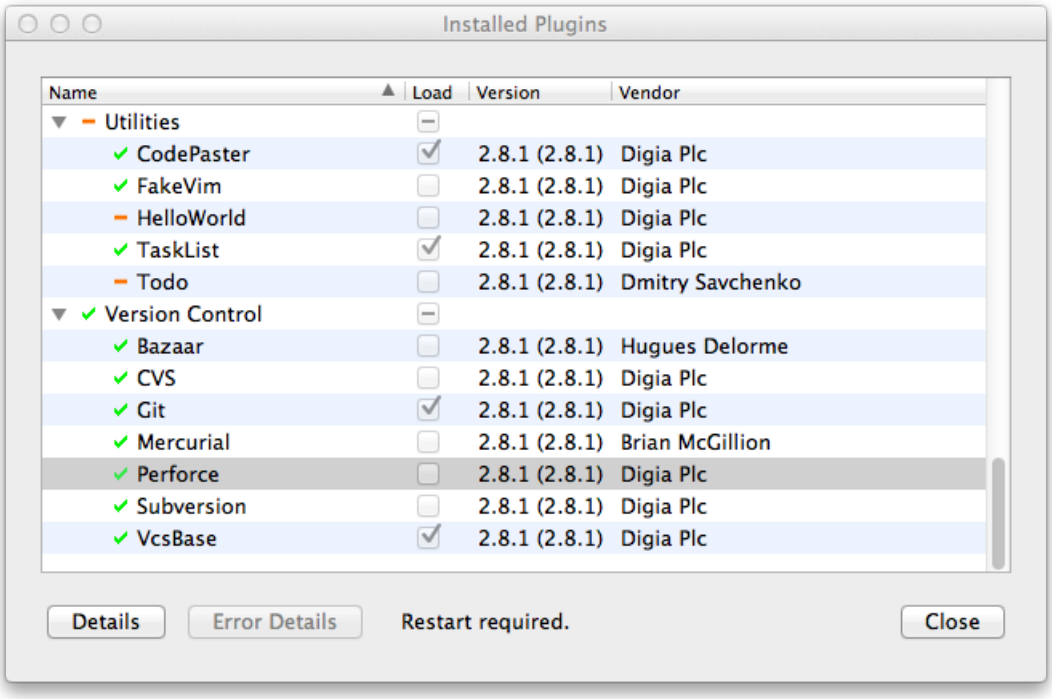


Figure 16: Deactivate every plugin you don't need.

2.2 Windows

Double click the executables that you have downloaded and follow the installer. From there on it should be more or less like the OSX install. Look in section *OSX* on page 5.

2.3 Linux

Since I have not installed the SDK on Linux yet, I can not provide any information here. Sorry!

Apart from that I have no physical Linux machine that is connected to a display and can be diverted for a test installation. A virtual machine might work but that would result in VMs inside a VM, not very promising.

Volunteers present?

3 Quickstart

Start the QtCreator from the fresh installed SDK.

"The SDK comes with a handy SailfishOS application template that gives you a quick way to create your very first Sailfish OS application. Just go to File-> New File or Project in the IDE"[sailfishos2]

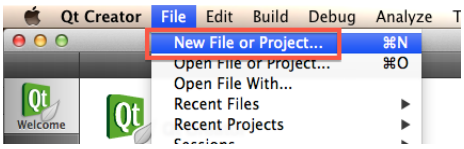


Figure 17: First example, step 1.

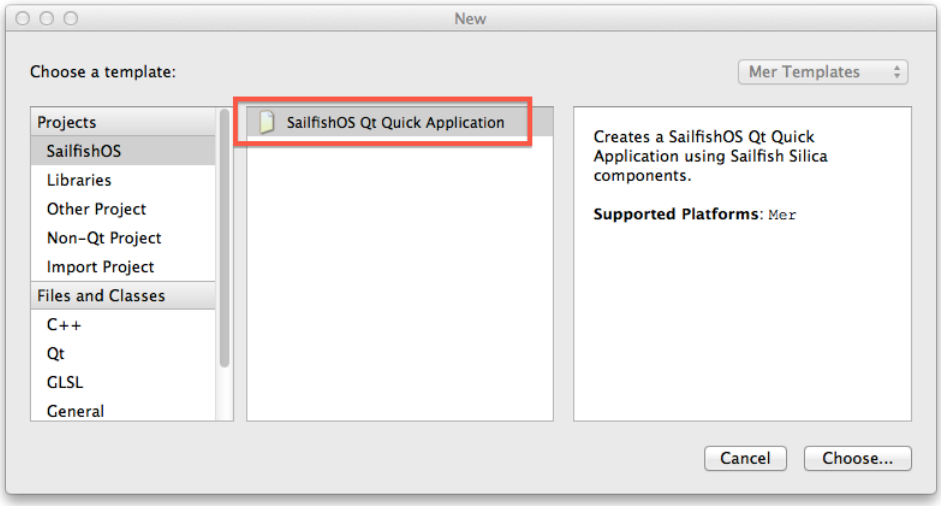


Figure 18: First example, step 2.

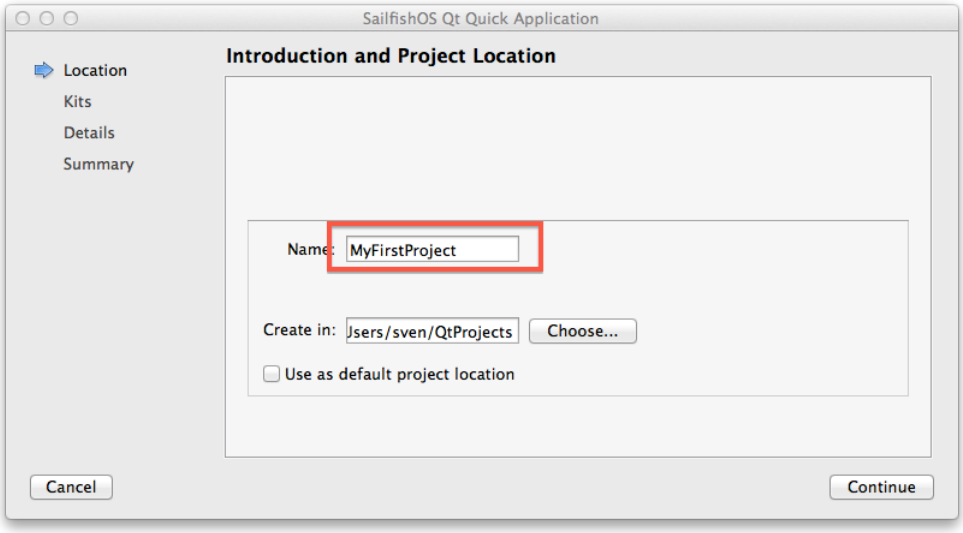


Figure 19: First example, step 3.

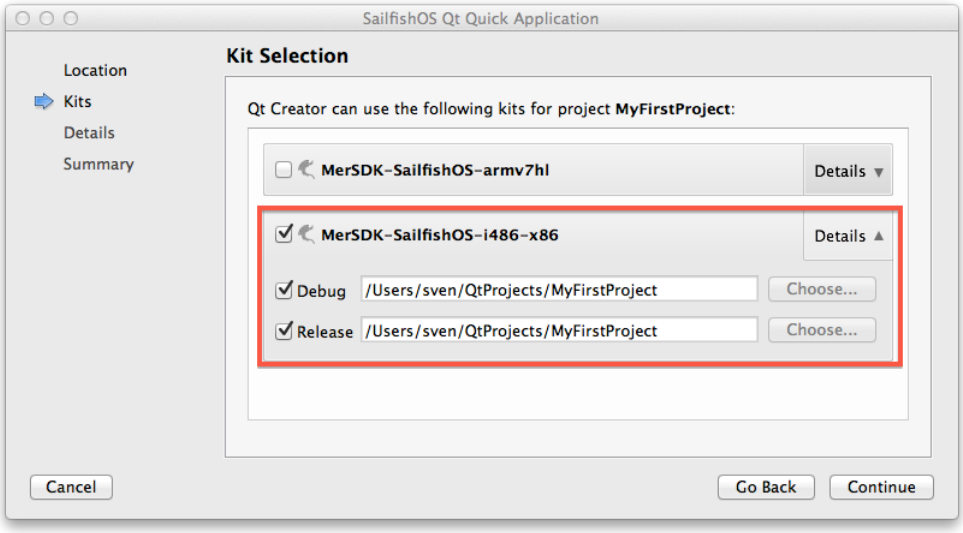


Figure 20: First example, step 4.

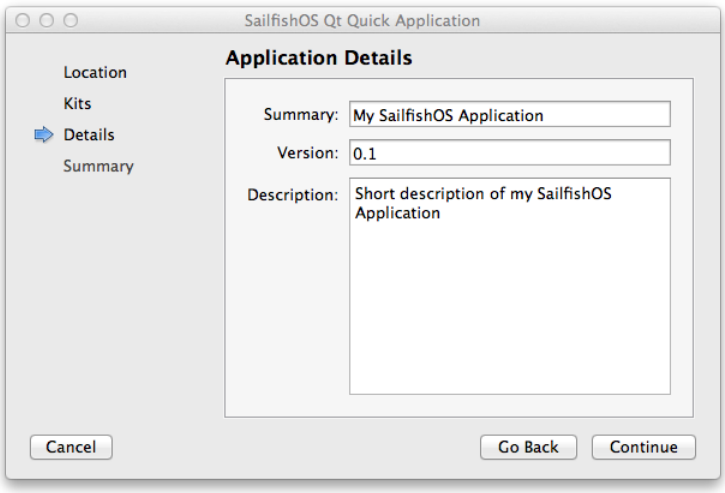


Figure 21: First example, step 5.

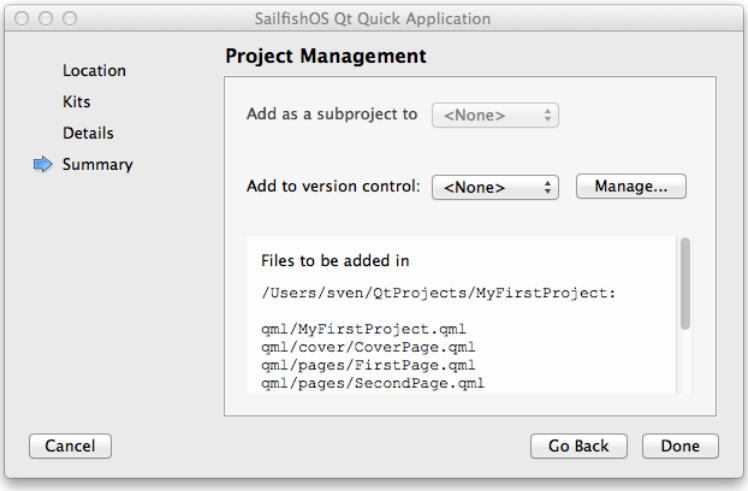


Figure 22: First example, step 1.

Start the SDK from inside the QtCreator.

Figure 23: Starting the SDK.

When the virtual machine with the SDK is running, apply updates if necessary.

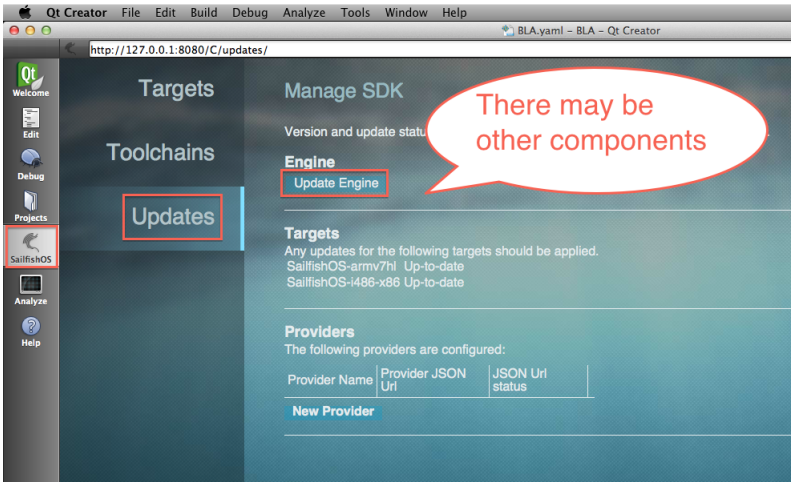


Figure 24: Updating the SDK.

Start the Emulator.

Figure 25: Starting the Emulator.

Compile and run your application.

Figure 26: Build and run the application.

Easy as pie, also see figure 47 on page 35. But what happens when and where if you click on those Icons? Look in section *Know your tools*.

A good starting point to look for a more sophisticated starting example is *The missing HelloWorld. Wizard included* by Artem Marchenko[gh01]. You should check that out!

4 Know your tools

Jolla chose the Qt framework to be part of their technology stack. “Qt is a cross-platform application and UI framework for developers using C++ or QML, a CSS

& JavaScript like language. Qt Creator is the supporting Qt IDE. Qt, Qt Quick and the supporting tools are developed as an open source project governed by an inclusive meritocratic model. Qt can be used under open source (LGPL v2.1) or commercial terms."[qt01]

"Qt - code once, deploy everywhere", that's the mantra of the Qt framework. If you have developed for more than one platform in the past, you know that this sounds like heaven. Maintaining different source code and technologies for each and every platform is a tedious task that can eat up all your developer resources. As if software development is not difficult enough if you stay on one platform[3].

So there were good reasons to choose Qt as framework, no doubt about that. As of now you can develop for Android, iOS, BlackBerry and of course SailfishOS. If you look into the documentation and examples of all these platforms, you will find that those examples assume, that you are developing for this platform only. Quite stupid, if you use the Qt framework. Understandable if you think about the effort that would be necessary to build a documentation that incorporates all other possible platforms. For some time now I was wondering how I should organize my code in such a way, that allows me to develop for more than one target platform at a time. It's not just compiling for another platform! Each platform has a unique UI that behaves in an own different way, e.g. SailfishOS is gesture based, other ones are touch based. In the long run you will create an UI for each of those targets. Period. Patterns like MVC[wiki01] will come to mind, using separate business logic, yada yada.

To cut a long story short, why am I writing about this stuff, this section is supposed to be about tools? When you prepare a software project for the use for more than one target platform, you will start organizing stuff differently. Maybe you use folders that have the name of the targets to differentiate stuff that's platform dependent. Maybe you even create a business logic that is really unique and capsulated in such clever way that it can be reused and does not know anything about the outside world. Such a business logic or model can be driven from tests, command line tools, web or different native UIs. Would be nice to have it in a separate folder or even subproject. If you start to move and/or rename things, your tools will break. Intentionally.

By examining those fractures you can learn a lot about your tools that otherwise work so silently in the background. So go on and break your tools![4]

Here is what I've learned so far.

[3] In my eyes software development is an art of craftsmanship and can not be done by Mr. Average and thus is a more or less complicated thing to do.

[4] Ok, not so short :-)

4.1 Technology stack

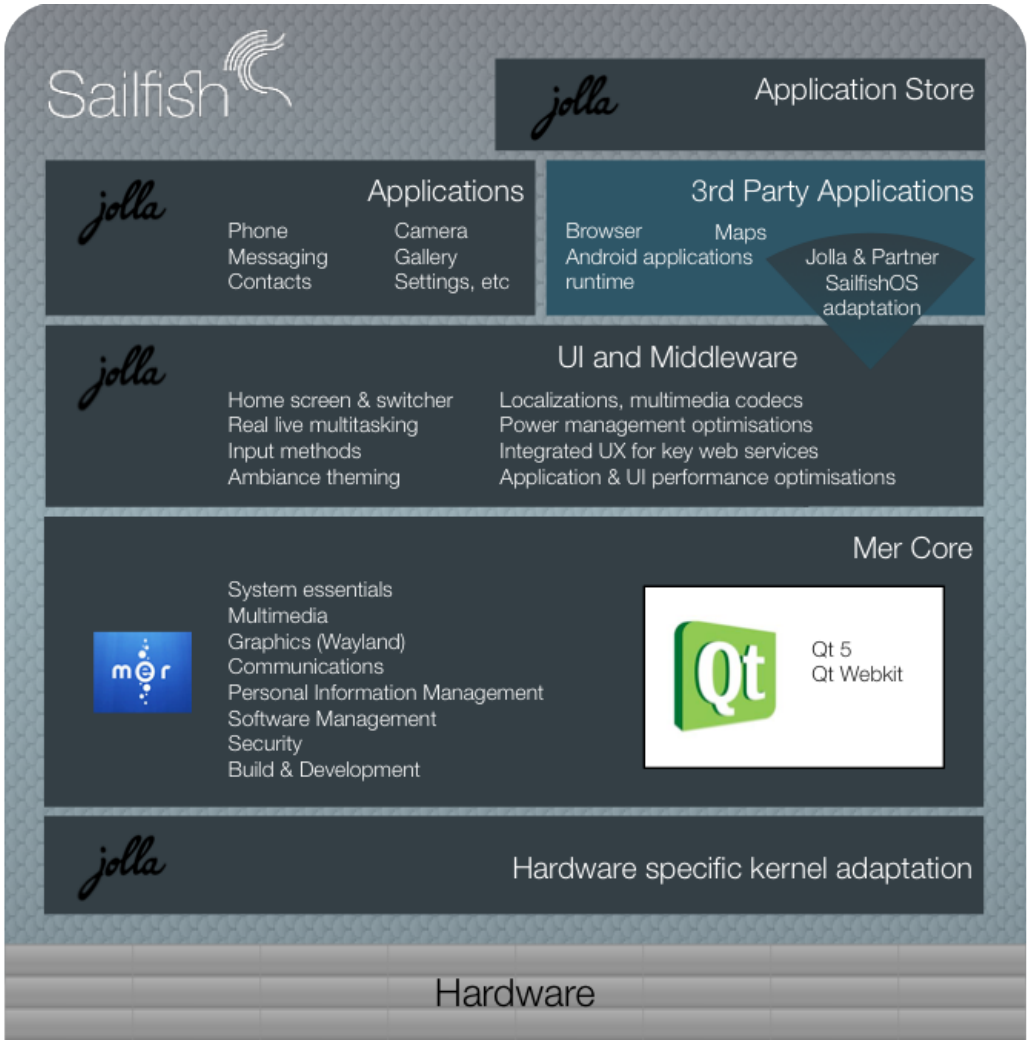


Figure 27: Sailfish architecture, taken from https://sailfishos.org/images/Sailfish_Architecture.png.

4.2 QtCreator integrated development environment (IDE)

"QtCreator is a cross platform integrated development environment (IDE) tailored to the needs of Qt developers. It has been extended to add support for Sailfish UI application development using Sailfish Silica components. It provides a sophisticated code editor with version control, project and build management system integration."[sailfishos3].

Reusing an existing open source IDE is a smart move from Jolla. Why should they waste resources on developing something that has already done by others? Or why should they burn up their staff for all those development solutions out there? Be it Visual Studio, Eclipse, Emacs or even Vi. If you really dive in the tools, you can also use those but I doubt that Jolla will provide you with support if something does not work. Working with QtCreator is also quite natural in the Qt universe albeit being a fast IDE. So have a look in the preferences[5].

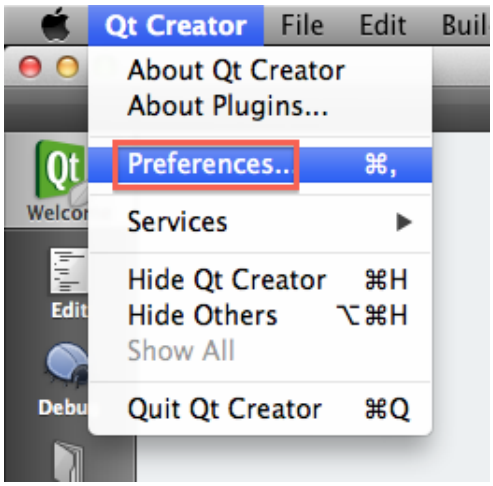


Figure 28: Open the QtCreator preferences.

I will not walk through every setting of QtCreator, [qt02] is a better place to start for basic questions. Also I will not use the order of tabs in the preferences, but try to follow the sequence in which those tools touch your source code.

4.2.1 kits

But before we do that, we must talk about *kits*. A kit is kind of an umbrella setting, which combines the information of the following bits and pieces, like `qmake`, `compiler`, and `device type`. This is the information hub that QtCreator uses to pull all information together and initiate its actions.

[5] On Windows and Linux they should be found in Extras/Options.

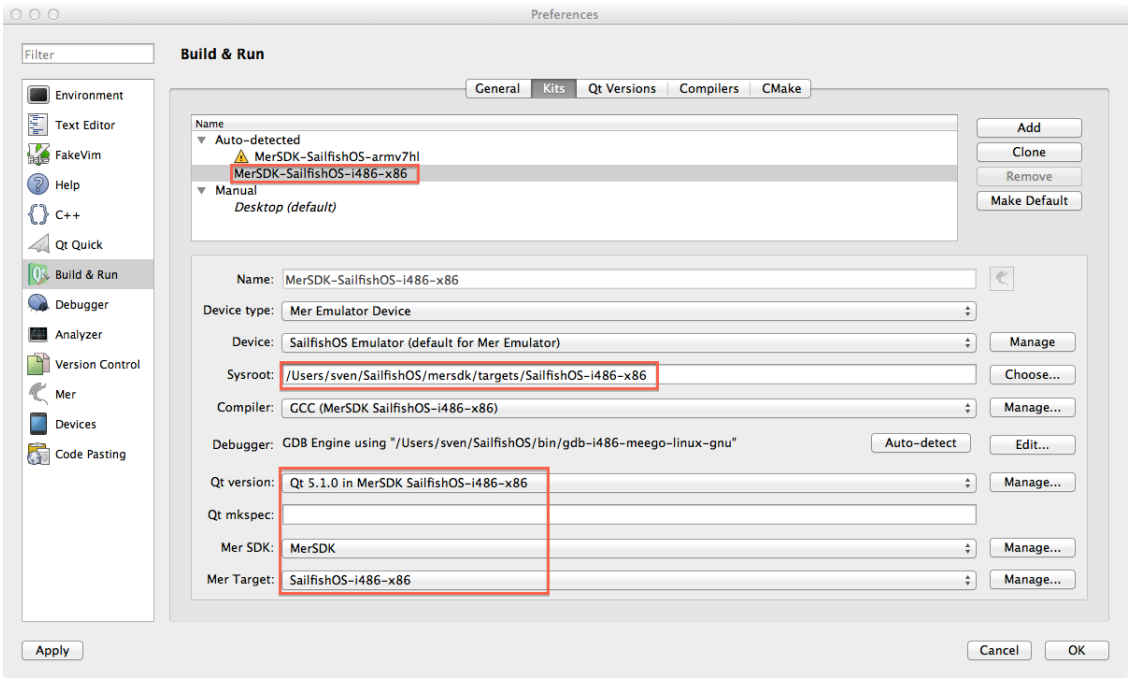


Figure 29: Preferences, kits tab.

4.2.2 qmake

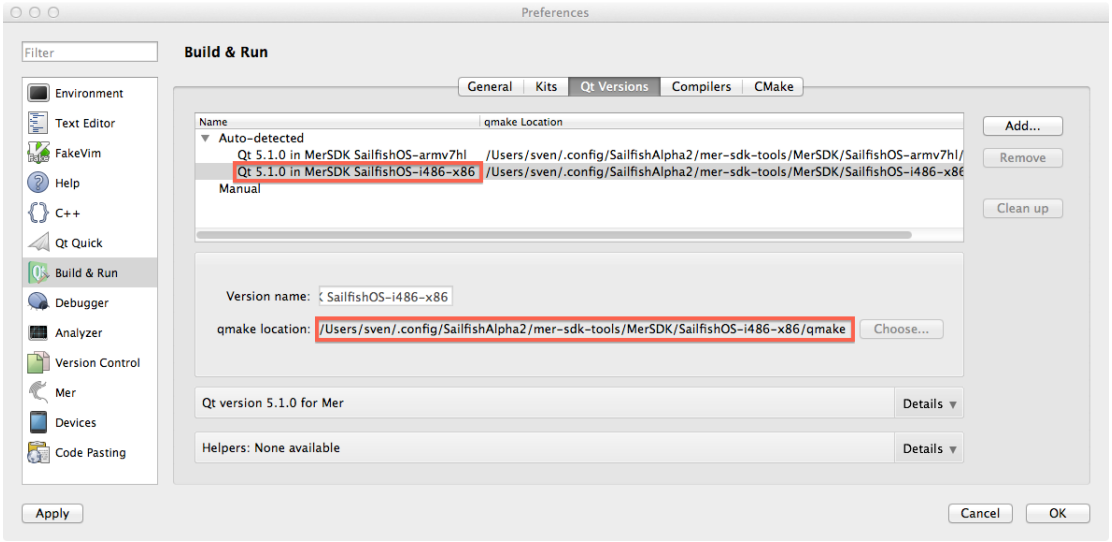


Figure 30: Preferences, QtVersions tab.

“qmake is a tool that helps simplify the build process for development project across different platforms. qmake automates the generation of Makefiles so that only a few lines of information are needed to create each Makefile. qmake can be used for any software project, whether it is written in Qt or not. qmake generates a Makefile based on the information in a project file. Project files are created by the developer, and are usually simple, but more sophisticated project files can be created for complex projects. qmake contains additional features to support development with Qt, automatically including build rules for moc and uic. qmake can also generate projects for Microsoft Visual studio without requiring the developer to change the project file.”[qt03]

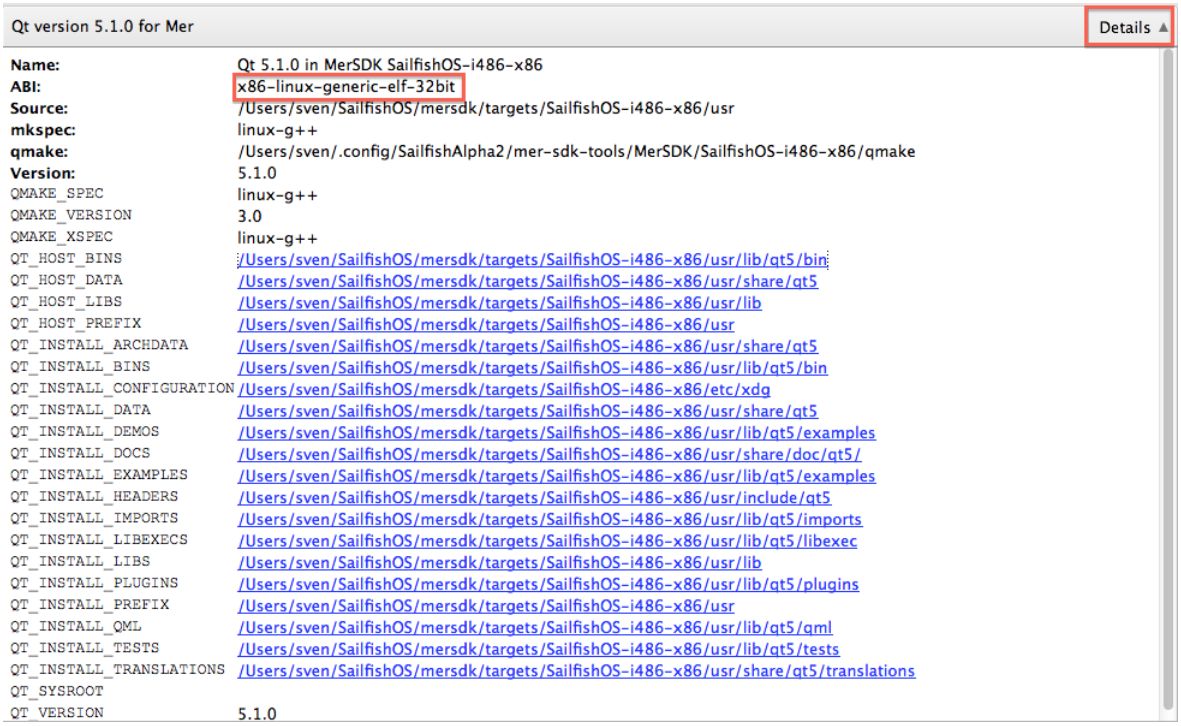


Figure 31: Preferences, QtVersions tab., qmake details

UIC is a tool that creates C++ classes from XML information generated with the UI designer inside QtCreator. This designer is for Qt widgets which should not be used with SailfishOS and is not further explained in this document.

MOC is the Meta-Object Compiler which “reads a C++ header file. If it finds one or more class declarations that contain the `Q_OBJECT` macro, it produces a C++ source file containing the meta-object code for those classes.”[qt04]. If you use `qmake` to produce your Makefile, you don’t have to worry about it, the rules are created automatically.

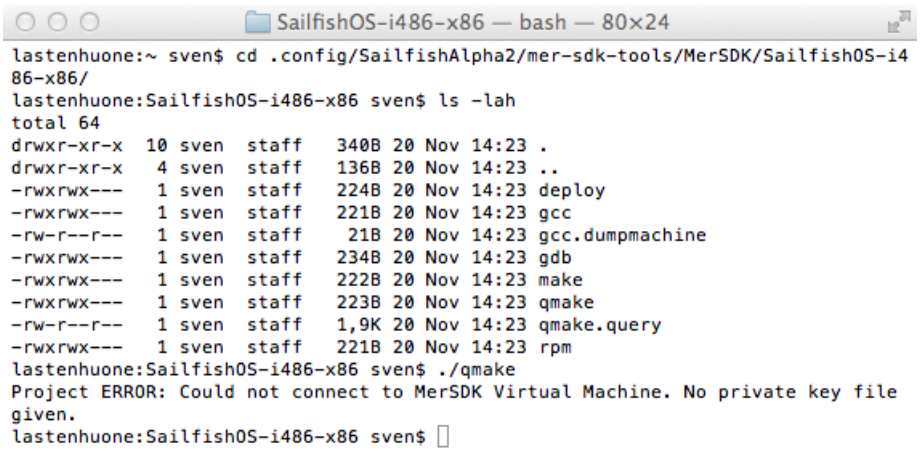


Figure 32: Running qmake from command line.

If you run qmake manually, you will find out that it tries to connect to the *Mer build engine for cross compilation*. The error also appears if the virtual machine is up and running. In the Projects settings you can see how `qmake` is invoked if started by QtCreator.

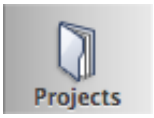


Figure 33: Project settings.

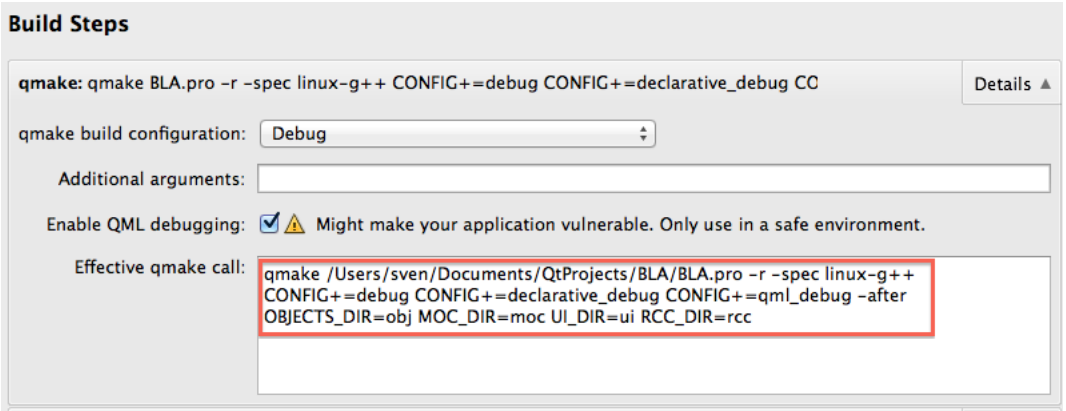


Figure 34: Build steps for qmake.

Using those parameters via command line does not work, too.

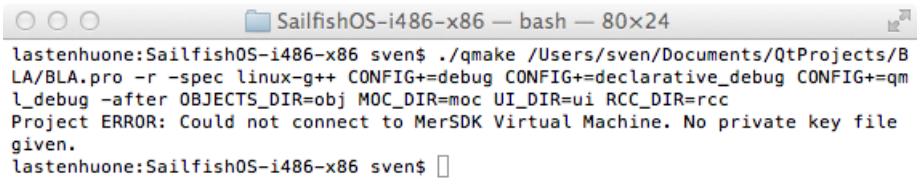


Figure 35: Running qmake from command line with parameters from build steps.

If `qmake` is invoked by the QtCreator it works just fine.

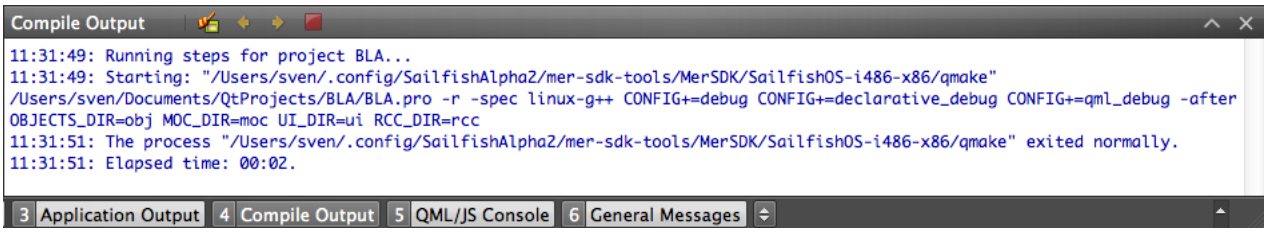


Figure 36: Running qmake from QtCreator (Build menu).

As a result you will find a `Makefile` in your project directory.

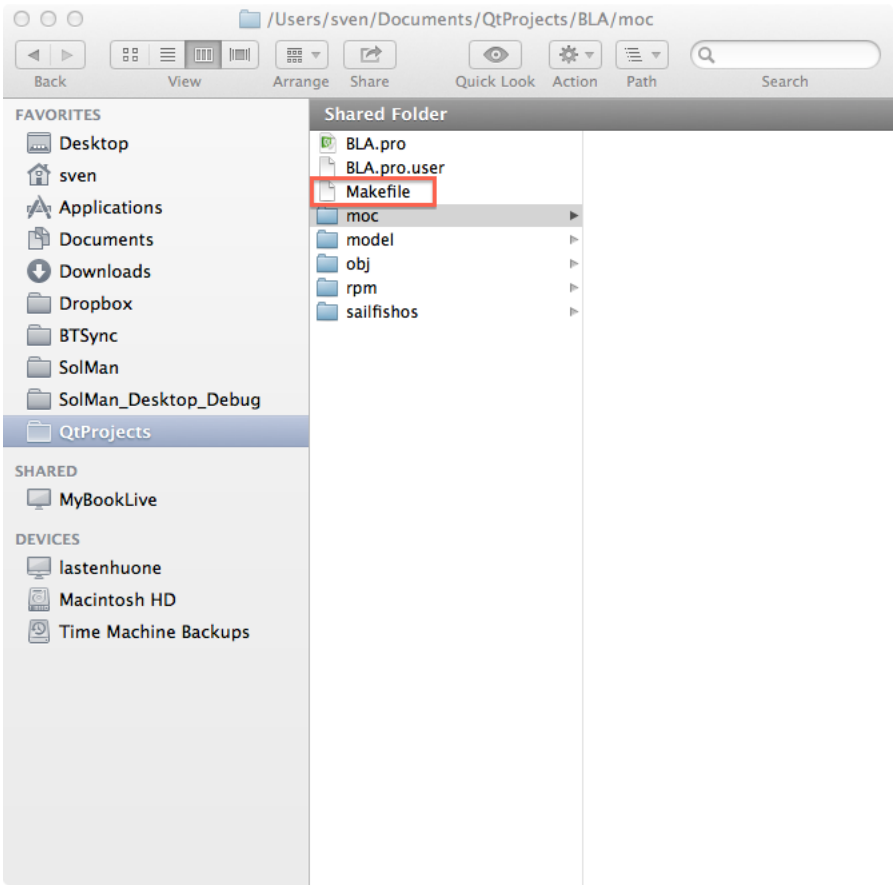


Figure 37: Result of running qmake from QtCreator (Build menu).

Here is the snippet about the MOC.

```
####### Sub-libraries

distclean: clean
	-$(DEL_FILE) $(TARGET)
	-$(DEL_FILE) Makefile


mocclean: compiler_moc_header_clean compiler_moc_source_clean

mocables: compiler_moc_header_make_all compiler_moc_source_make_all

check: first

compiler_rcc_make_all:
compiler_rcc_clean:
compiler_wayland-server-header_make_all:
compiler_wayland-server-header_clean:
```


```
compiler_wayland-client-header_make_all:
compiler_wayland-client-header_clean:
compiler_qtwayland-client-header_make_all:
compiler_qtwayland-client-header_clean:
compiler_qtwayland-server-header_make_all:
compiler_qtwayland-server-header_clean:
compiler_moc_header_make_all: moc/moc_qbusinesslogic.cpp
compiler_moc_header_clean:
	-$(DEL_FILE) moc/moc_qbusinesslogic.cpp
moc/moc_qbusinesslogic.cpp: /usr/include/qt5/QtCore/QObject \
    /usr/include/qt5/QtCore/qobject.h \
    /usr/include/qt5/QtCore/qobjectdefs.h \
    /usr/include/qt5/QtCore/qnamespace.h \
    /usr/include/qt5/QtCore/qglobal.h \
    /usr/include/qt5/QtCore/qconfig.h \
    /usr/include/qt5/QtCore/qfeatures.h \
    /usr/include/qt5/QtCore/qsystemdetection.h \
    /usr/include/qt5/QtCore/qcompilerdetection.h \
    /usr/include/qt5/QtCore/qprocessordetection.h \
    /usr/include/qt5/QtCore/qglobalstatic.h \
    /usr/include/qt5/QtCore/qatomic.h \
    /usr/include/qt5/QtCore/qbasicatomic.h \
    /usr/include/qt5/QtCore/qatomic_bootstrap.h \
    /usr/include/qt5/QtCore/qgenericatomic.h \
    /usr/include/qt5/QtCore/qatomic_msvc.h \
    /usr/include/qt5/QtCore/qatomic_integrity.h \
    /usr/include/qt5/QtCore/qoldbasicatomic.h \
    /usr/include/qt5/QtCore/qatomic_vxworks.h \
    /usr/include/qt5/QtCore/qatomic_power.h \
    /usr/include/qt5/QtCore/qatomic_alpha.h \
    /usr/include/qt5/QtCore/qatomic_armv7.h \
    /usr/include/qt5/QtCore/qatomic_armv6.h \
    /usr/include/qt5/QtCore/qatomic_armv5.h \
    /usr/include/qt5/QtCore/qatomic_bfin.h \
    /usr/include/qt5/QtCore/qatomic_ia64.h \
    /usr/include/qt5/QtCore/qatomic_mips.h \
    /usr/include/qt5/QtCore/qatomic_s390.h \
    /usr/include/qt5/QtCore/qatomic_sh4a.h \
    /usr/include/qt5/QtCore/qatomic_sparc.h \
    /usr/include/qt5/QtCore/qatomic_x86.h \
    /usr/include/qt5/QtCore/qatomic_cxx11.h \
    /usr/include/qt5/QtCore/qatomic_gcc.h \
    /usr/include/qt5/QtCore/qatomic_unix.h \
    /usr/include/qt5/QtCore/qmutex.h \
    /usr/include/qt5/QtCore/qlogging.h \
    /usr/include/qt5/QtCore/qflags.h \
    /usr/include/qt5/QtCore/qtypeinfo.h \
    /usr/include/qt5/QtCore/qtypetraits.h \
    /usr/include/qt5/QtCore/qsysinfo.h \
```

```
    /usr/include/qt5/QtCore/qobjectdefs_impl.h \
    /usr/include/qt5/QtCore/qstring.h \
    /usr/include/qt5/QtCore/qchar.h \
    /usr/include/qt5/QtCore/qbytearray.h \
    /usr/include/qt5/QtCore/qrefcount.h \
    /usr/include/qt5/QtCore/qarraydata.h \
    /usr/include/qt5/QtCore/qstringbuilder.h \
    /usr/include/qt5/QtCore/qlist.h \
    /usr/include/qt5/QtCore/qalgorithms.h \
    /usr/include/qt5/QtCore/qiterator.h \
    /usr/include/qt5/QtCore/qcoreevent.h \
    /usr/include/qt5/QtCore/qscopedpointer.h \
    /usr/include/qt5/QtCore/qmetatype.h \
    /usr/include/qt5/QtCore/qvarlengtharray.h \
    /usr/include/qt5/QtCore/qcontainerfwd.h \
    /usr/include/qt5/QtCore/qisenum.h \
    /usr/include/qt5/QtCore/qobject_impl.h \
    model/qt/qbusinesslogic.h
  /usr/lib/qt5/bin/moc $(DEFINES) $(INCPATH) -I/usr/lib/gcc/i486-
    meego-linux/4.6.4/../../../../include/c++/4.6.4 -I/usr/lib/gcc/
    i486-meego-linux/4.6.4/../../../../include/c++/4.6.4/i486-meego-
    linux -I/usr/lib/gcc/i486-meego-linux/4.6.4/../../../../include/c
    ++/4.6.4/backward -I/usr/lib/gcc/i486-meego-linux/4.6.4/include -I
    /usr/local/include -I/usr/include model/qt/qbusinesslogic.h -o moc
    /moc_qbusinesslogic.cpp
```

The `qmake` from the SailfishOS SDK is just a simple bash script, that invokes `merssh`.

```
#!/bin/bash
exec "/Users/sven/SailfishOS/bin/Qt Creator.app/Contents/MacOS/../
    Resources/merssh" -sdktoolsdir "/Users/sven/.config/SailfishAlpha2
    /mer-sdk-tools/MerSDK" -commandtype mb2 -mertarget SailfishOS-i486
    -x86 qmake $@oluhuone:SailfishOS-i486-x86
```

So that's the trick: `$@` is replaced with the `qmake` call parameters, `oluhuone` is just the name of one of my computers.

4.2.3 merssh

Looking with `top` showed a process called `merssh` when `qmake` was started via QtCreator. Interesting, what's that?

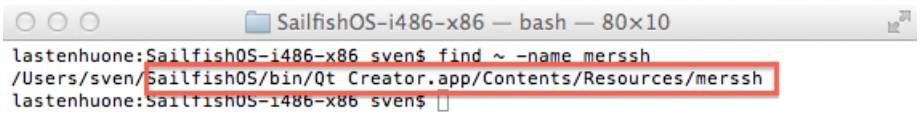


Figure 38: What is merssh?.

So it is part of the QtCreator that is shipped with the SailfishOS SDK.

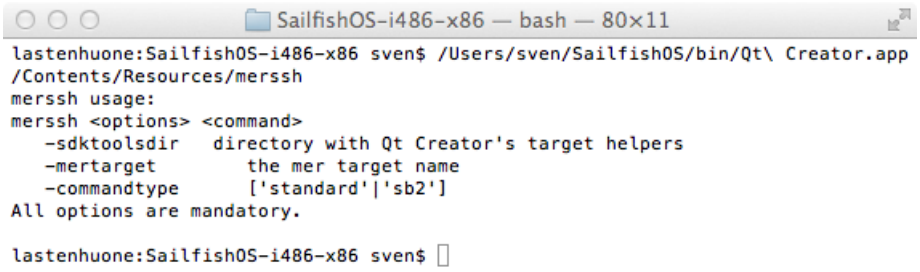


Figure 39: merssh invoked, what's it?.

The existence of `merssh` need further investigation, I can not tell more at the moment. Probably it corresponds to the settings of the Mer SDK, shown in figure 46 on page 34.

More than that, it calls `sb2` which is short for Scratchbox2, have a look at section Scratchbox2 on page 34, there are more details. For now let's just assume that "Scratchbox 2 is a cross-compilation engine, it can be used to create a highly flexible SDK."[sb2].

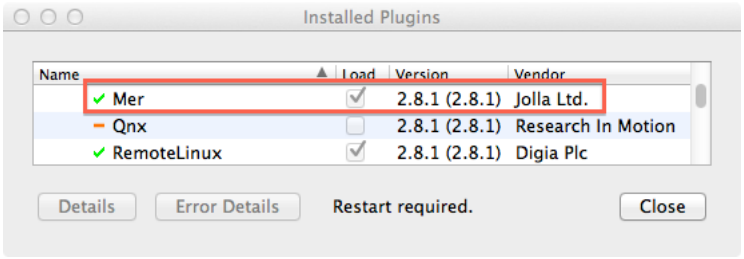


Figure 40: Mer plugin, maybe that's the source of merssh?.

I've grepped the command line for the `merssh`

```
$ ps -ef|grep "merssh"
```

And the result is:

```
/Users/sven/SailfishOS/bin/Qt Creator.app/Contents/MacOS/../Resources
   /merssh -sdktoolsdir /Users/sven/.config/SailfishAlpha2/mer-sdk-
   tools/MerSDK -commandtype mb2 -mertarget SailfishOS-i486-x86 qmake
    /Users/sven/QtProjects/TestSailfishOS/TestSailfishOS.pro -r -spec
    linux-g++ CONFIG+=debug CONFIG+=declarative_debug CONFIG+=
   qml_debug -after OBJECTS_DIR=obj MOC_DIR=moc UI_DIR=ui RCC_DIR=rcc
```

The picture is getting clearer now. QtCreator starts the SDK version of `qmake` which call `merssh` with all parameters needed to call `qmake` via `mb2` on the virtual machine.

4.2.4 Compilers

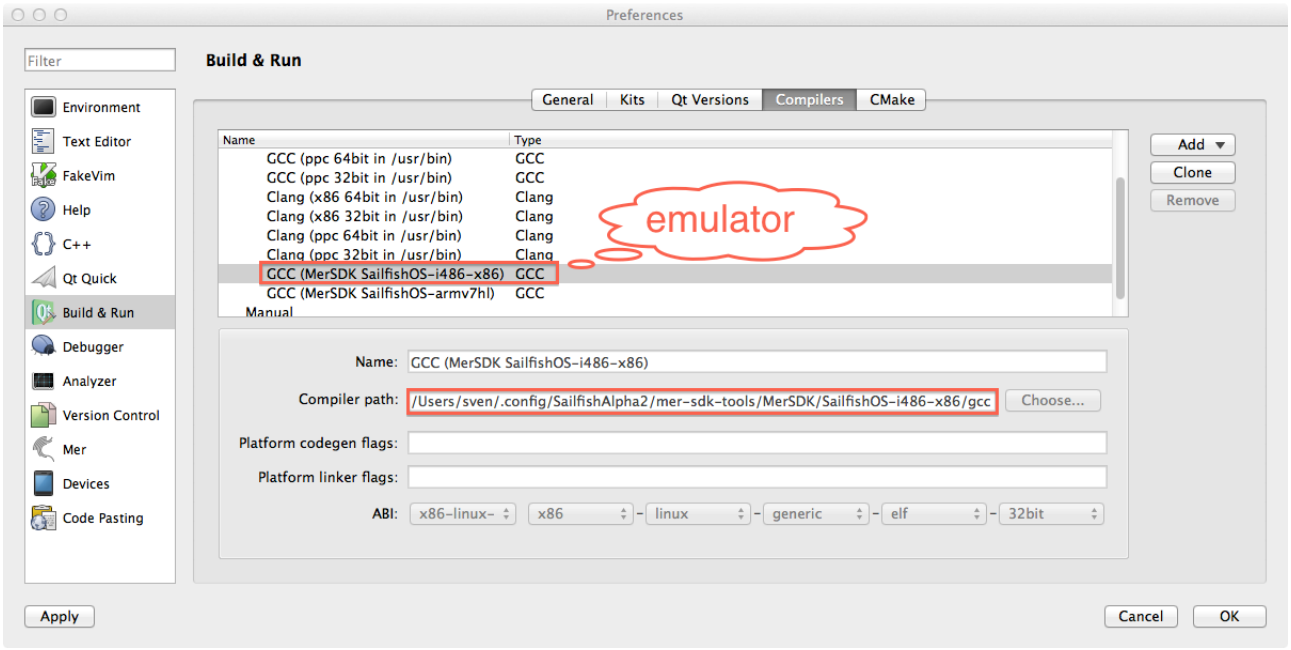


Figure 41: Preferences, compiler tab.

The SailfishOS SDK uses GCC as compiler. It is run inside the Mer build engine for cross compilation, see page 33. Stored on your development machine is only

a stub or proxy that wants to connect to the virtual machine and start compiling from there.

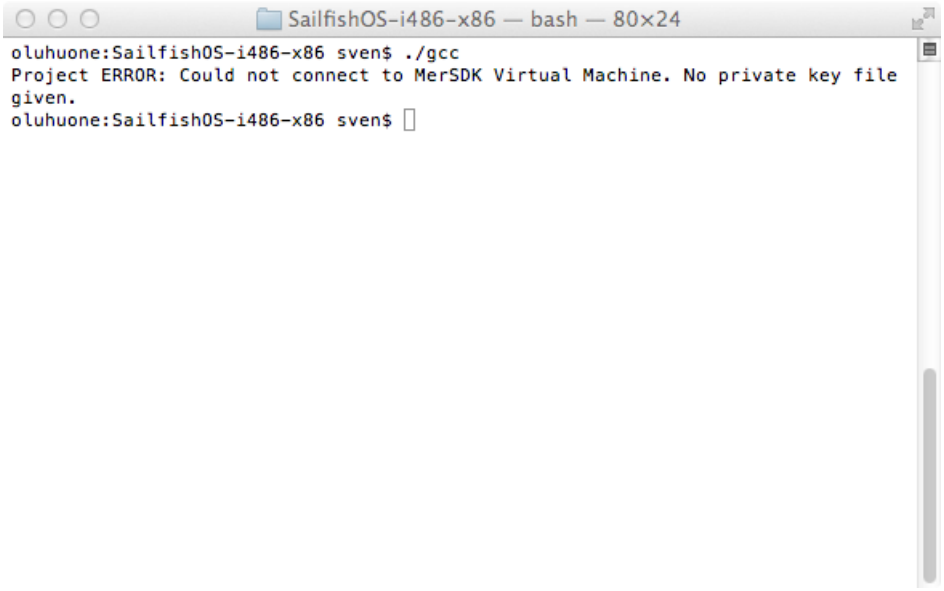


Figure 42: Running GCC from command line.

So far I don't know why this piece of software is not installed with the rest of the SDK, `~/.config` is not a directory where I would expect executables. The error message even shows up if the *Mer build engine for cross compilation* is up and running. Again this helper program is invoked via merssh, see page 28.

Looking inside `gcc` from the SDK I also find a bash script:

```
#!/bin/bash
exec "/Users/sven/SailfishOS/bin/Qt Creator.app/Contents/MacOS/../
    Resources/merssh" -sdktoolsdir "/Users/sven/.config/SailfishAlpha2
    /mer-sdk-tools/MerSDK" -commandtype sb2 -mertarget SailfishOS-i486
    -x86 gcc $@oluhuone:SailfishOS-i486-x86
```

`$@` is replaced with the `gcc` call parameters, `oluhuone` is just the name of my current machine.

4.2.5 make

Again, `make` is just a bash script, `$@` replaced, `oluhuone` my machine:

```
#!/bin/bash
```


```
exec "/Users/sven/SailfishOS/bin/Qt Creator.app/Contents/MacOS/../
    Resources/merssh" -sdktoolsdir "/Users/sven/.config/SailfishAlpha2
    /mer-sdk-tools/MerSDK" -commandtype mb2 -mertarget SailfishOS-i486
    -x86 make $@oluhuone:SailfishOS-i486-x86
```

I've grepped the command line for the `merssh` while building the application.

```
$ ps -ef|grep "merssh"
```

Resulting in

```
/Users/sven/SailfishOS/bin/Qt Creator.app/Contents/MacOS/../Resources
    /merssh -sdktoolsdir /Users/sven/.config/SailfishAlpha2/mer-sdk-
    tools/MerSDK -commandtype mb2 -mertarget SailfishOS-i486-x86 make
```

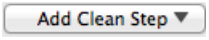
4.2.6 rpm

Guess what, a bash script:

```
#!/bin/bash
exec "/Users/sven/SailfishOS/bin/Qt Creator.app/Contents/MacOS/../
    Resources/merssh" -sdktoolsdir "/Users/sven/.config/SailfishAlpha2
    /mer-sdk-tools/MerSDK" -commandtype mb2 -mertarget SailfishOS-i486
    -x86 rpm $@oluhuone:SailfishOS-i486-x86
```

4.2.7 Project settings

4.2.8 Pimp the clean process

Every now and then you clean your project. What bugged my for some time using QtCreator[6] was that it left the Makefile after you cleaned the project. This way `qmake` is often not run after a `make clean`. No problem, just hit the button 

and create an extra step that is executed every time after the cleanup has been done.

[6]That has nothing to do with the SailfishOS SDK, the regular QtCreator does that, too.

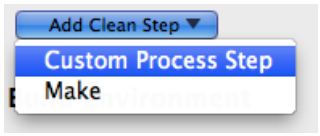


Figure 43: Create a custom process step.

Figure 44: Clean step: Remove the Makefile.

Here again for copy and paste:

```
rm
-f %{buildDir}/Makefile
%{buildDir}
```

4.3 Mer build engine for cross compilation

"The Mer build engine is a virtual machine (VM) containing the Mer development toolchains and tools. It also includes a SailfishOS target for building and running Sailfish and QML applications. The target is mounted as a shared folder to allow QtCreator to access the compilation target. Additionally, your home directory is shared and mounted in the VM, thus giving access to your source code for compilation. The build engine also supports additional build targets and cross-compilation toolchains. These can be managed from the SDK Control Centre interface within QtCreator which allows toolchains, targets and even individual target packages to be added and removed."[sailfishos3].

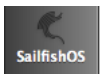


Figure 45: SailfishOS icon.

The VM runs headless, you can not see it running. For you as a developer there is a webpage served by this VM accessible through the SailfishOS icon inside QtCreator. See figure 24 on page 18.

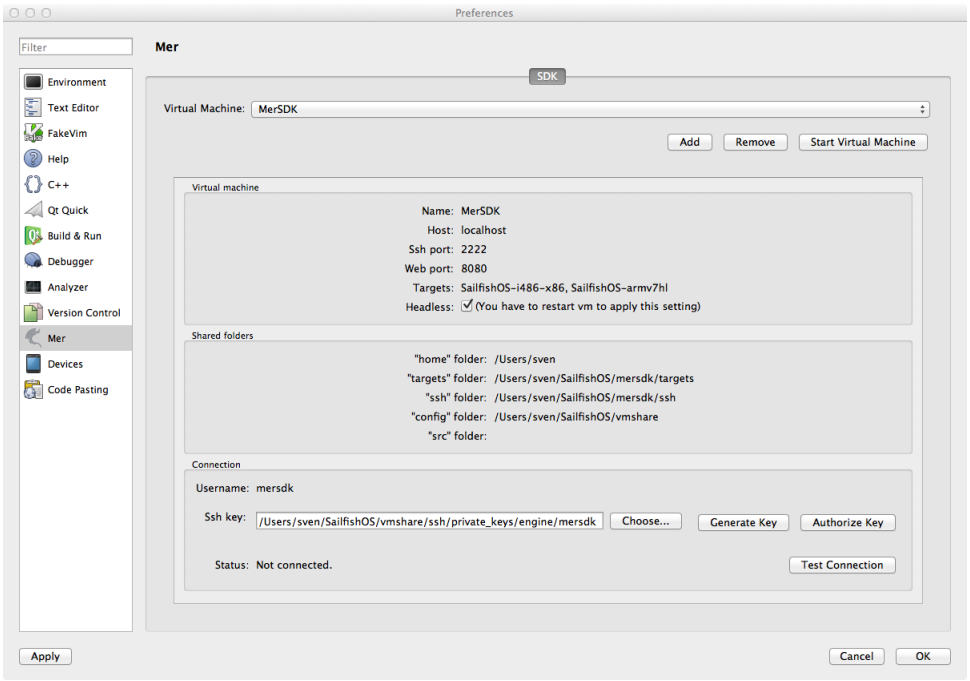


Figure 46: Preferences, Mer SDK - virtual machine.

4.4 Scratchbox2

“Scratchbox2 (sbox2 or sb2) is a cross-compilation toolkit designed to make embedded Linux application development easier. It also provides a full set of tools to integrate and cross-compile an entire Linux distribution.

In the Linux world, when building software, many parameters are auto-detected based on the host system (like installed libraries and system configurations), through autotools "./configure" scripts for example. But so, when one wants to build for an embedded target (cross-compilation), most of the detected parameters are incorrect (i.e. host configuration is not the same as the embedded target configuration).

Without Scratchbox2, one has to manually set many parameters and "hack" the "configure" process to be able to generate code for the embedded target.

At the opposite, Scratchbox2 allows one to set up a "virtual" environment that will trick the autotools and executables into thinking that they are directly running on the embedded target with its configuration.

Moreover, Scratchbox2 provides a technology called CPU-transparency that goes further in that area. With CPU-transparency, executables built for the host CPU or for the target CPU could be executed directly on the host with sbox2 handling the task to CPU-emulate if needed to run a program compiled for the target

CPU. So, a build process could mix the usage of program built for different CPU architectures. That is especially useful when a build process requires building the program X to be able to use it to build the program Y (Example: building a Lexer that will be used to generate code for a specific package)."[wiki02]

The Wiki page of the Mer project contains a exhaustive description how to compile a program on platform A for platform B[mer01].

4.5 The SailfishOS Emulator

"The emulator is an x86 VM image containing a stripped down version of the target device software. It emulates most of the functions of the target device running Sailfish operating system, such as gestures, task switching and ambience theming."[sailfishos3]. At least with the AlphaSDK2 the emulator can not simulate device rotations.

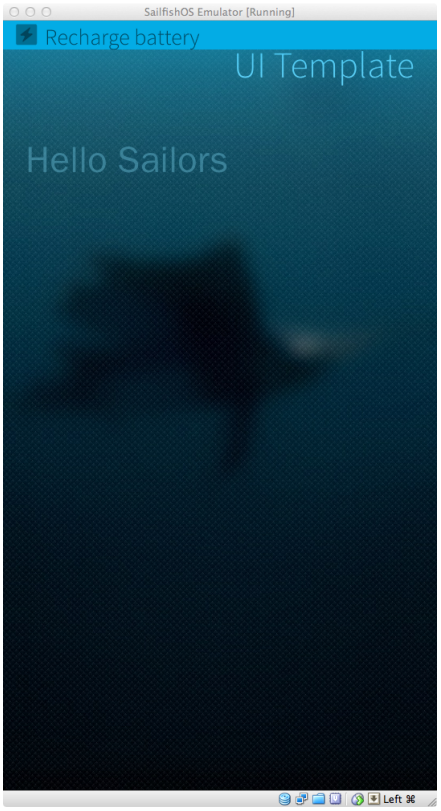


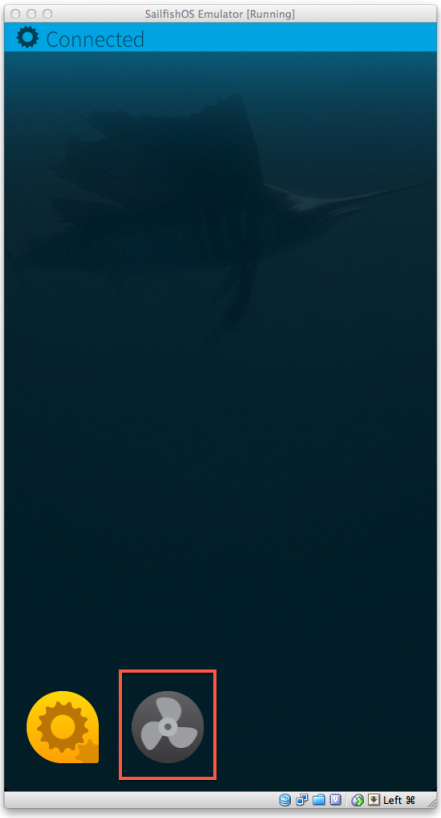
Figure 47: Emulator running the templated SailfishOS Qt Quick Application.

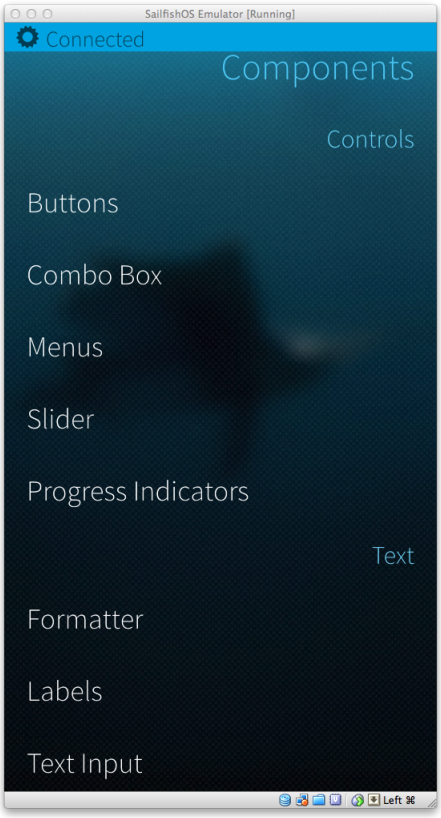
4.6 Sailfish Silica

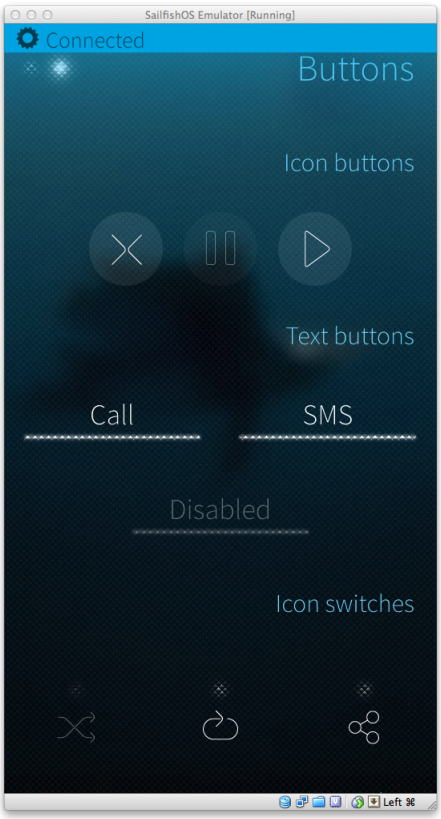
"Sailfish Silica is a QML module which provides Sailfish UI components for applications. Their look and feel fits with the Sailfish visual style and behavior and enables unique Sailfish UI application features, such as pulley menus and application covers."[sailfishos3].

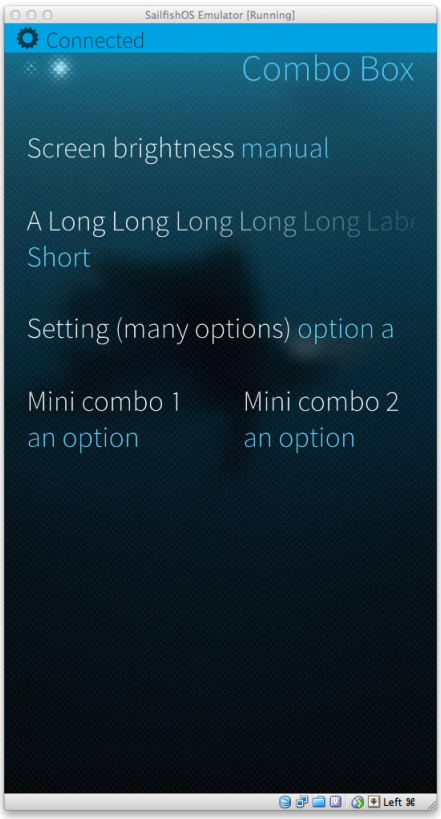
QML[qt05] is the Qt *Q*uick *M*arkup *L*anguage[qt06] that supersedes widgets for designing user interfaces. It is a declarative "language" that can contain a small subset of Javascript.

Also have a look at some open source examples on Github[sailfishos5]. The emulator comes with a demo application that shows the silica components.

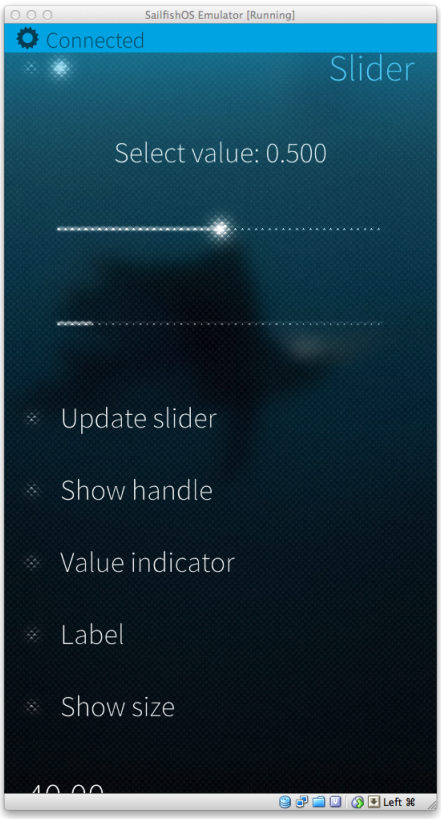


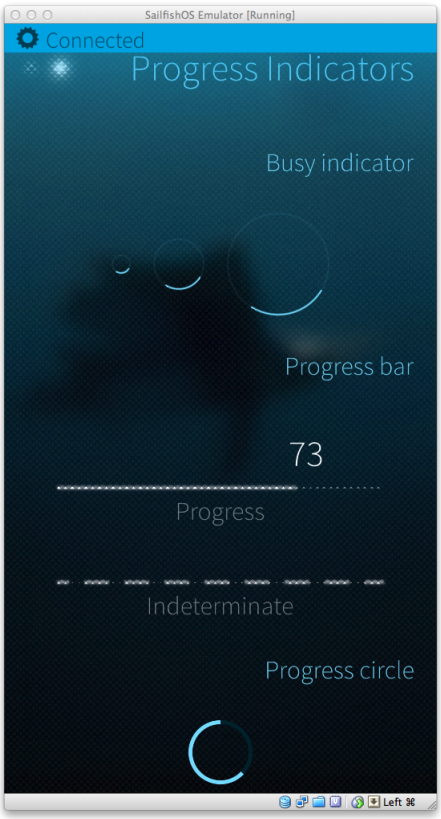


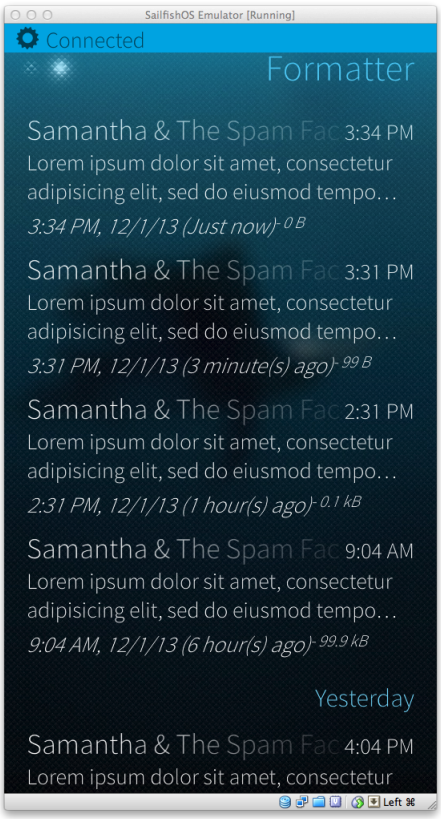

SailfishOS Emulator [Running]
Connected
Buttons
Icon buttons
Text buttons
Call
SMS
Disabled
Icon switches

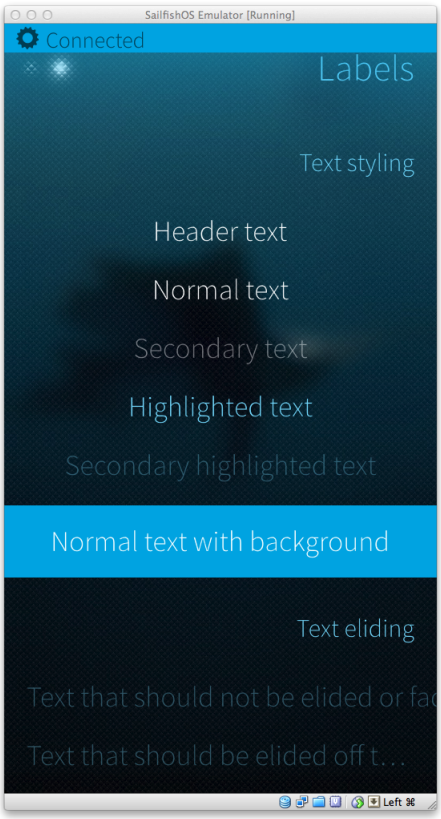

SailfishOS Emulator [Running]
Connected
Combo Box
Screen brightness manual
A Long Long Long Long Long Lab
Short
Setting (many options) option a
Mini combo 1
an option
Mini combo 2
an option

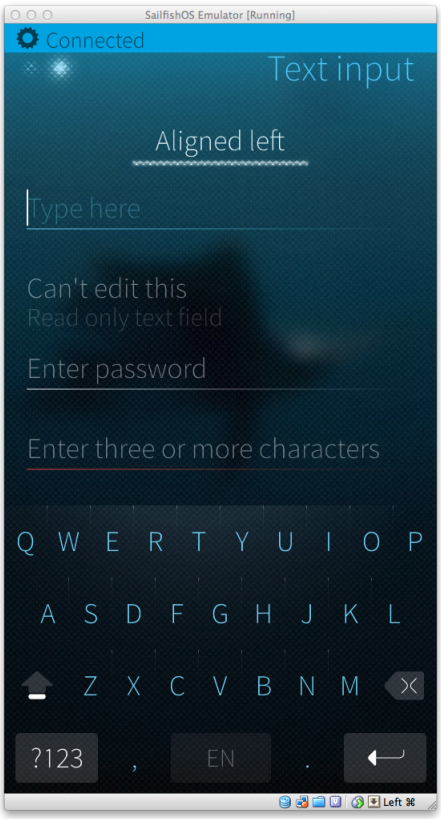

SailfishOS Emulator [Running]
Connected
Menus
1. Since Long
2. Run Every
3. Planetary Civilization
4. Endangered Impacts
5. Space Every
6. Surviving Civilization
7. Obliged Become
8. Spacefaring Because
9. Exploratory Romantic


SailfishOS Emulator [Running]
Connected
Slider
Select value: 0.500
Update slider
Show handle
Value indicator
Label
Show size


SailfishOS Emulator [Running]
Connected
Progress Indicators
Busy indicator
Progress bar
73
Progress
Indeterminate
Progress circle
Left ⌘


SailfishOS Emulator [Running]
Connected
Formatter
Samantha & The Spam Fac 3:34 PM
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo…
3:34 PM, 12/1/13 (Just now) - 0 B
Samantha & The Spam Fac 3:31 PM
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo…
3:31 PM, 12/1/13 (3 minute(s) ago) - 99 B
Samantha & The Spam Fac 2:31 PM
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo…
2:31 PM, 12/1/13 (1 hour(s) ago) - 0.1 kB
Samantha & The Spam Fac 9:04 AM
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo…
9:04 AM, 12/1/13 (6 hour(s) ago) - 99.9 kB
Yesterday
Samantha & The Spam Fac 4:04 PM
Lorem ipsum dolor sit amet, consectetur
Left ⌘





4.7 Tools chained up

Now that we have seen all the tools, bits and pieces, I will try to give an overview how everything works together, when you compile your code in QtCreator for SailfishOS.

TODO put some text and images here. TODO find out, how the packaging happens, Makefile?

5 Installing additional packages

You can use additional libraries for your code, so you don't have to write all functionality for yourself. Some of them are available on the emulator and the physical device later on. Sadly not all library packages that are available for Nemo/Mer are usable here. Have look in section Harbour on **??** for details about allowed packages.

5.1 Emulator

You can login from your development machine via terminal and `ssh`, the emulator should be running.

```
$ ssh -p 2223 nemo@localhost
nemo@localhost's password:
# the password is nemo
,---
| SailfishOS 0.98.0.67 (i486,testing)
'---
[nemo@SailfishEmul ~]$
```

As an alternative you can log in the running VM of the emulator. Press CMD+F2[7][8] to change to the login screen. User and password are of course the same.

5.1.1 zypper

Additional software and libraries come in `RPM` packages and the management tool on the emulator is `zypper`. It provides "functions like repository access, dependency solving, package installation"[suse01].

```
[nemo@SailfishEmul ~]$ zypper refresh
```

will update the meta data that is stored from the package repository.

```
[nemo@SailfishEmul ~]$ zypper search
```

will show you all packages that can be installed on the emulator.

```
[nemo@SailfishEmul ~]$ zypper search boost
```

[7] On OSX your function keys will probably not work as regular function keys, they provide OSX functionality as printed on then, e.g. volume up/down. Go to System Preferences"->Keyboard->Keyboard and check check "Use all F1, F2, etc. as standard function keys".

[8] On Windows and Linux use the CTRL Key instead of CMD.

will show you all package names that contain `boost`.

```
[nemo@SailfishEmul ~]$ sudo zypper install boost-filesystem
```

will install the library `boost-filesystem` and all its dependencies on the emulator. *Note*: `boost-filesystem` is not one of the currently available libraries[9].

```
[nemo@SailfishEmul ~]$ sudo zypper remove boost-filesystem
```

will remove the library `boost-filesystem` and all its dependencies from the emulator. Of course you can install additional software on the emulator[10] that helps you to edit files directly or manage the filesystem better.

Note: you do not need the development packages on the emulator of physical device.

6 Templates for QtCreator

The SailfishOS SDK comes with a template for a new SailfishOS Qt Quick Application project. On OSX those templates are stored inside the QtCreaor bundle, you can change those templates there or create new ones.

[9] So why do I use it as an example? Honi soit qui mal y pense.
[10] As long as this software is not part of your app.

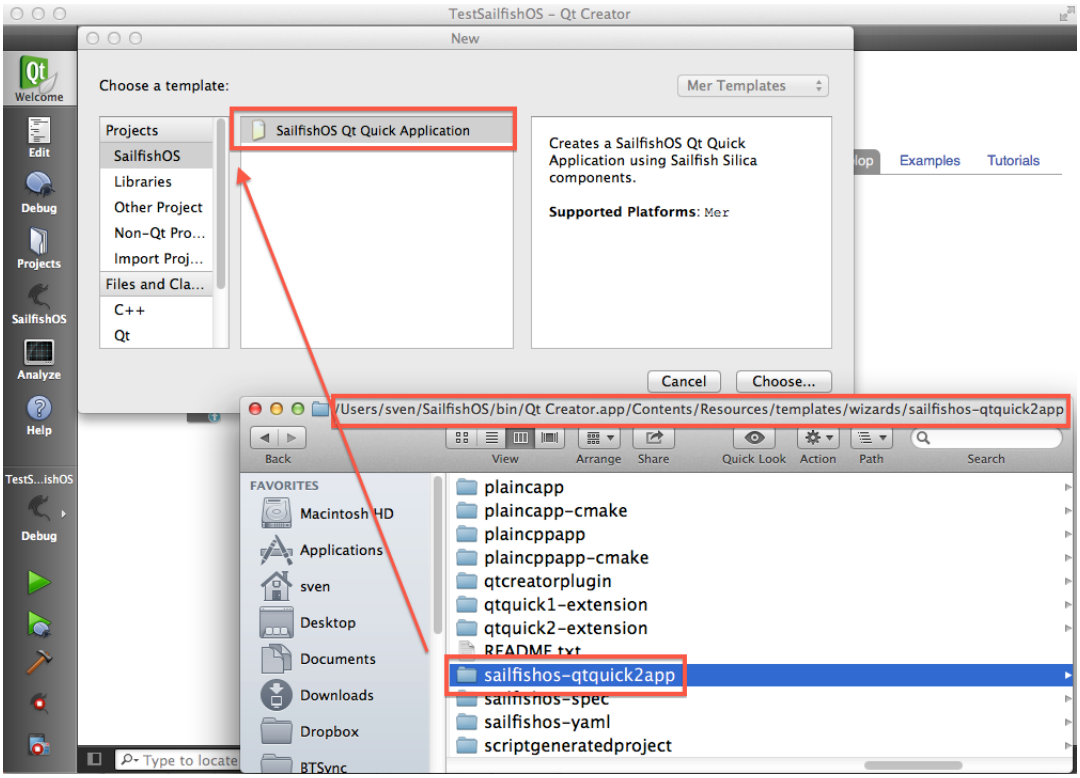


Figure 48: Template for a new SailfishOS Qt Quick Application.

Just make sure that you don't use the names `obj`, `moc`, `ui` or `rcc` inside your template. These are going to be used to store the compile results and temporaries if you compile a SailfishOS program.

Updates of the SDK may delete changed or new templates, you might want to create a backup in a safe place.

TODO example of a new template, figure out the caveats here!

7 Physical device

Developing with the emulator only will do you no good. You have to experience your program on a real device. Things that might look great on an emulator, may not even work on a real phone. It maybe just you fingers that are hiding the screen. Buttons are too small or too close.

7.1 How to connect to SSH over usb connection from PC

- the usb is either `usb_storage` or `usb_net`

- enable developer mode
- enable SSH (it's openssh, not dropbear)
- set password
- goto usb settings
- change that to developer mode
- reconnect usb cable
- you should see the ip address of the device on the UI
- you should be able to ssh to that address from PC (set an ip address first)

Taken from [ex01].

This section obviously needs a lot more information. Lacking a physical device or an SDK that enables me to interact with it, this has to be done in future.

8 Harbour

our wrote a fantastic app and now you want to bring it to the people. Head over to the Jolla Harbour[jolla02] and submit your app.

There are some things about naming an such to consider when you perpare your app, here is the Sailfish FAQ from the Jolla Harbour as of now = Dec. 2nd 2013. This is of course a moving target, which means there will be more libraries available in future. Even if you don't have an app ready to submit, you should look there once in a while. Ne need to re-invent the wheel here. If there is missing something, contact Jolla.

```
Naming
What should I name the application?
You have to use the prefix "harbour-" in front of the application
    name. Only lower case characters are allowed.
Why do I have to name my Application with a prefix?
The reason for this requirement is so that applications do not clash
    with other installed packages on the device. It also allows us to
    verify certain things automatically, e.g. imports of your own QML
    modules.
What is that '$NAME' you use here in the FAQ's?
That is your application name including the prefix "harbour-" (e.g.
    harbour-myawesomeapp).
How can I name the app in the application launch grid? I don't want
    that long name with prefix to appear there!
```

```
In the .desktop file, there is the "Name=" field. The string defined
    there is shown in the application launcher as the application name
    . That name does not have to be unique. So there might be more
    than one application called "HelloWorld!" in the application
    launcher grid. The package name of the application ($NAME) does
    not appear in the UI.
Where must this $NAME be used?
the executable binary: /usr/bin/$NAME
the .desktop file: /usr/share/applications/$NAME.desktop
the icon: /usr/share/icons/hicolor/86x86/apps/$NAME.png
the folder in /usr/share where you can install other application
    files: /usr/share/$NAME
The rpm package name, note that is not necessarily the same as the
    RPM file name!
The name you get with:
rpm -q --queryformat='%{NAME}\n' -p harbour-awesomeapp-1.0.0.armv7hl.
    rpm
That is what is set in .spec resp. .yaml file under "Name: "
Your own QML imports resp. modules, but with "-" replaced with "."
    due to QML grammar rules. e.g import harbour.myawesomeapp.
    MyQmlModule 1.0 as MyModule
Do I need to put that unique $NAME into the "Title" field when I
    upload an app to the Harbour?
No, in the "Title" field you can use a pretty name that will be shown
     to the user in the store client UI.
RPM-Packaging
In which locations can I install files?
You are allowed to install:
/usr/bin/$NAME <- the executable binary
/usr/share/applications/$NAME.desktop <- the desktop file
/usr/share/icons/hicolor/86x86/apps/$NAME.png <- the icon file
/usr/share/$NAME/* <- anything else (data files, private shared
    libraries, private QML imports, etc..) goes here
Why are you so restrictive? Why can't I install my libraries, images
    etc in places where I think it makes sense?
We have to ensure that rpms can be installed and do not conflict with
     other rpms. It will also allow us to install store applications
    under a different path (rpm --relocate) in the future.
What does the package name have to be? (.spec/.yaml "Name: foo")
Use "Name: $NAME". See Naming section in this FAQ.
Icons
Which size should the application icon be?
86x86. Older SDK versions contain a template which suggests a size of
     90x90. That is obsolete and will soon be updated with the next
    SDK version.
Where shall the icon be installed?
/usr/share/icons/hicolor/86x86/apps/$NAME.png. Older SDK versions
    contain a template which suggests a size of 90x90 and also a
    different install path. That is obsolete, not supported anymore,
```

```
   and will soon be updated with the next SDK version.
What file formats are supported for the icon?
The icon must be a PNG file.
How do I define an icon in the .desktop file, so it shows up in the
   application launcher?
Icon=$NAME

You must not use absolute path names. There was a bug in older SDK
   versions (lipstick < 0.18.6) so the absolute path was necessary
   and the template suggested it. That will soon be updated with the
   next SDK version. The reason to not use absolute path names is: it
    would allow us in the future to install store applications under
   a different path.

.desktop-Files
What do I have to put into the Exec= line?
Exec=$NAME (for Silica applications using C++ and QML) or Exec=
   sailfish-qml $NAME (for QML-only Silica applications without an
   application binary)
What do I have to put into the Icon= line?
Icon=$NAME
What do I have to put into the Name= line?
The name defined there is shown in the application launcher as the
   application name. That name does not have to be unique. So there
   might be more than one application called "HelloWorld!" in the
   application launcher grid.
What do I have to put into the X-Nemo-Application-Type= line and what
    is it good for?
For Silica Qt 5 applications, use "X-Nemo-Application-Type=silica-qt
   5" - this will make sure that the application is launched using
   the right booster, and will make startup faster.
How can I disable single-instance launching?
In general, you should use single-instance launching (tapping on the
   application icon will bring the existing window to the foreground)
   . If for some reason your application conflicts with single-
   instance launching, you can add "X-Nemo-Single-Instance=no" to
   your .desktop file to disable this behaviour.
QML API
Which QML modules (imports) are allowed?
Currently the following QML modules (imports) are allowed:
QtQuick 2.0
QtQuick 2.1
QtMultimedia 5.0
Sailfish.Silica 1.0
QtQuick.LocalStorage 2.0
QtQuick.XmlListModel 2.0
QtQuick.Particles 2.0
QtQuick.Window 2.0
Why do you not allow more QML modules from Qt/Nemo/Mer or other 3rd
```

```
   party?
Not all modules have a stable API. Before promising compatibility, we
    must first make sure that we can promise the API is of high
   quality and will not change (through a review process). We are
   open for suggestions to provide more APIs.
Can I use QML modules, which I ship together with the application?
Yes, you can do that. But you have to prefix the name of the imports
   with your $NAME, but with "-" replaced with "." due to QML grammar
    rules (e.g. harbour.myapp.myQmlObject, where harbour-myapp = $
   NAME). And you have to install them under /usr/share/$NAME (
   loadable QML plugins or the QML files) if the type is not built
   into the application (setContextProperty).
Shared Libraries
Which shared libraries can I link against?
Currently the following shared libraries are allowed:
libsailfishapp.so.1
libmdeclarativecache5.so.0
libQt5Quick.so.5
libQt5Qml.so.5
libQt5Network.so.5
libQt5Gui.so.5
libQt5Core.so.5
libGLESv2.so.2
libpthread.so.0
libstdc++.so.6
libm.so.6
libgcc_s.so.1
libc.so.6
ld-linux-armhf.so.3
libQt5Concurrent.so.5
libQt5Multimedia.so.5
libQt5Sql.so.5
libQt5Svg.so.5
libQt5XmlPatterns.so.5
libQt5Xml.so.5
librt.so.1
libz.so.1
libQt5DBus.so.5
Why do you allow just such a limited amount of shared libraries?
We can only whitelist libraries that have both a stable API and ABI,
   and which we are sure we can provide for the forseeable future.
   For now, the list is quite small, but as Sailfish OS matures, we
   expect to add more. We are open for suggestions to allow more
   stable shared libraries.
Can I link against shared libraries which I ship with the app
   together in the same rpm?
Yes, you can do that. But you have to install the library under /usr/
   share/$NAME/. You have to ensure yourself that the rpath in your
   executable is set correct, so the linker finds the library. Future
```

```
    versions of invoker might set the LD_LIBRARY_PATH to /usr/share/$
    NAME/, but that is not yet in place.
You do not allow QtOpenGL, what is the alternative?
QtGui in Qt 5 includes a number of classes to replace the QtOpenGL
    classes. In many cases, using the QtGui equivalents will just
    involve a renaming (QGL* -> QOpenGL*) and removing the linking
    against QtOpenGL. There are API changes involved in some cases,
    but these should not be too difficult.

Startup performance is better without using QtOpenGL, which is one
    reason we are disallowing it. This is due to its dependency on
    QtWidgets, which is quite a large library.

Why do you not allow QtWidgets?
QtWidgets is not optimized for (or well tested) on Sailfish OS.
    Furthermore, it is generally not going to result in a good user
    experience due to using non-touch-optimized controls, and software
     rendering (which will be much slower than rendering using OpenGL
    ES on Sailfish OS).
I think library XYZ would be useful for others too, I want to make
    that library available in the store. Can I submit a library only
    rpm?
No, the app store, is as the name says, an application store and not
    a shared library store. But if you think an important library is
    missing in SailfishOS and you want to see it available in the
    platform, then please join the Mer and Nemo project and make the
    library available in one of these projects. Then suggest the
    library to become a supported one (by making sure it has a stable
    API, and is well-maintained and supported).
Runtimes
Can I submit Python applications?
Currently not, there are some enablers missing for that. But we are
    working on it, to make that happen. You can support us with that
    effort, please ask in Nemo project how to help with Python.
Can I submit Perl/Ruby/$MY_FAVOURITE_LANGUAGE applications?
No, and we currently do not plan to support that. But feel free to
    request it, if there is enough interest, we might allow it in the
    future.
```

9 Community

9.1 Jolla

Jolla homepage: `http://jolla.com`.
Jolla on Twitter: `https://twitter.com/JollaHQ`.

9.1.1 SailfishOS

As a developer you should subscribe to the mailing list at `https://lists.sailfishos.org/cgi-bin/mailman/listinfo/devel`.

Have a look at the Wiki at `https://sailfishos.org/wiki/Main_Page`. At freenode: `#sailfishos`.

9.1.2 Mer

At freenode: `#mer`. Homepage: `http://merproject.org`.

9.1.3 Nemo mobile

At freenode: `#nemomobile`. Mer Wiki page about the Nemo project: `https://wiki.merproject.org/wiki/Nemo`.

10 Thanks

Of course a big thanks goes out to everybody at Jolla. You are `#unlike`!

References

[jolla01] Jolla ltd., based in Finland - the inventors of the Jolla smartphone `http://jolla.com`

[jolla02] Jolla Harbour - the place for your apps `https://harbour.jolla.com`

[sailfishos01] SailfisoS - the operating system driving the Jolla smartphone `https://sailfishos.org`

[sailfishos2] Quick introduction into development with the SailfishOS SDK `https://sailfishos.org/develop.html`

[sailfishos3] SailfishOS SDK overview `https://sailfishos.org/develop-overview-article.html`

[sailfishos4] SailfishOS open source code `http://releases.sailfishos.org/sdk/`

[sailfishos5] Unofficial Sailfish OS third party open source apps collection `https://github.com/sailfishapps`

[vbox01] VirtualBox, a virtualization platform from Oracle
https://www.virtualbox.org/wiki/Downloads

[hc01] hardcodes, that's my nickname and my website
http://www.hardcodes.de

[hc02] This document on Github
https://github.com/hardcodes/
developwithsailfishos.git

[hc03] Alternative Icon for the QtCreator inside of the SailfishOS SDK
http://blog.hardcodes.de/articles/68/
sailfish-os-icon

[qt01] The Qt Project.
https://qt-project.org

[qt02] QtCreator.
https://qt-project.org/doc/qtcreator-2.8/

[qt03] qmake manual.
http://qt-project.org/doc/qt-4.8/
qmake-manual.html

[qt04] Meta-Object Compiler (MOC)
http://qt-project.org/doc/qt-4.8/moc.html

[qt05] QML
http://qt-project.org/doc/qt-5.0/qtqml/
qtqml-index.html

[qt06] QtQuick
http://qt-project.org/doc/qt-5.0/qtquick/
qtquick-index.html

[qt07] QtCreator, Adding New Custom Wizards
http://qt-project.org/doc/qtcreator-2.8/
creator-project-wizards.html

[wiki01] Wikipedia, Model–view–controller (MVC)
http://en.wikipedia.org/wiki/Model\T1\
textendashview\T1\textendashcontroller

[wiki02] Wikipedia, Scratchbox2
http://en.wikipedia.org/wiki/Scratchbox2

[sb2] Scratchbox2 homepage
https://maemo.gitorious.org/scratchbox2

[mer01] Mer Wiki, Platform SDK and SB2
https://wiki.merproject.org/wiki/Platform_SDK_and_SB2

[ex01] Jolla details on eLinux
http://elinux.org/Jolla

[sdc01] SmartDevCon
http://smartdevcon.eu

[gh01] The missing HelloWorld. Wizard included by Artem Marchenko
https://github.com/amarchen/helloworld-pro-sailfish

[suse01] openSUSE Portal:Zypper
http://en.opensuse.org/Portal:Zypper