Developing with SailfishOS

a short introduction

Sven Putze, `hardcodes.de`

November 30, 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].

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

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.

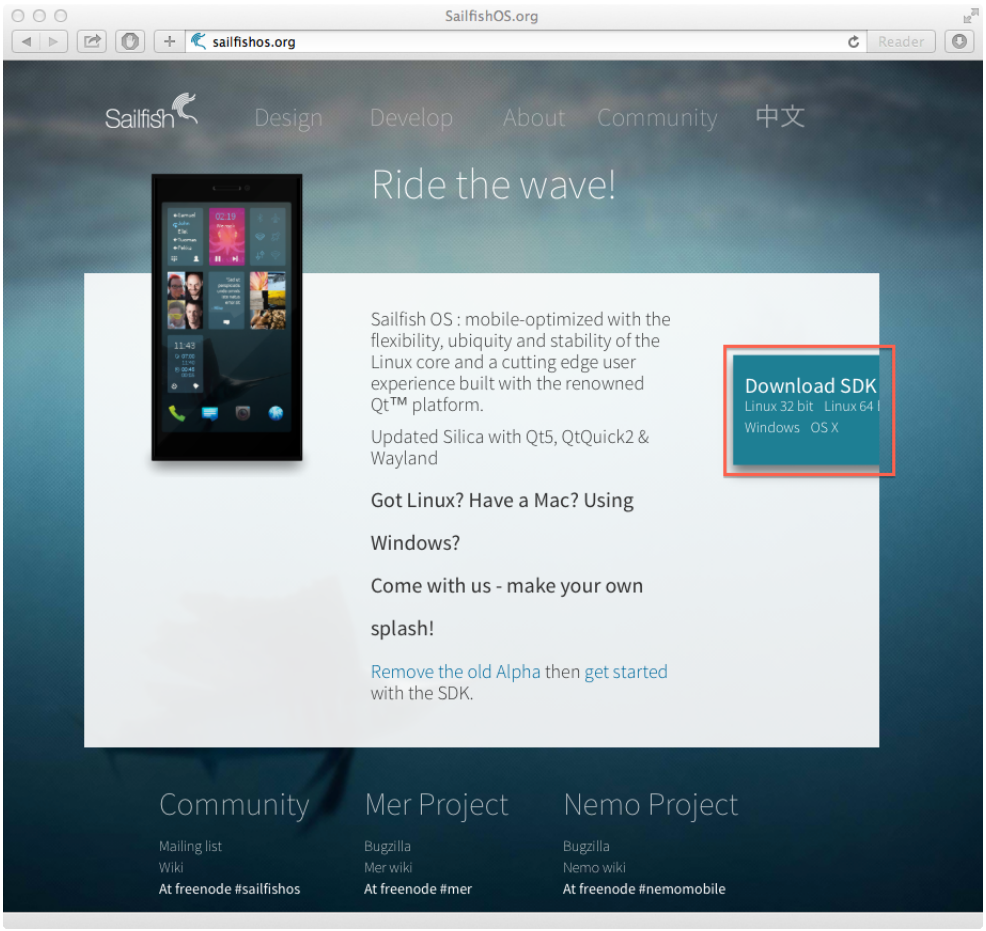

Figure 2: Download the SDK from the SailfishOS website.

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.

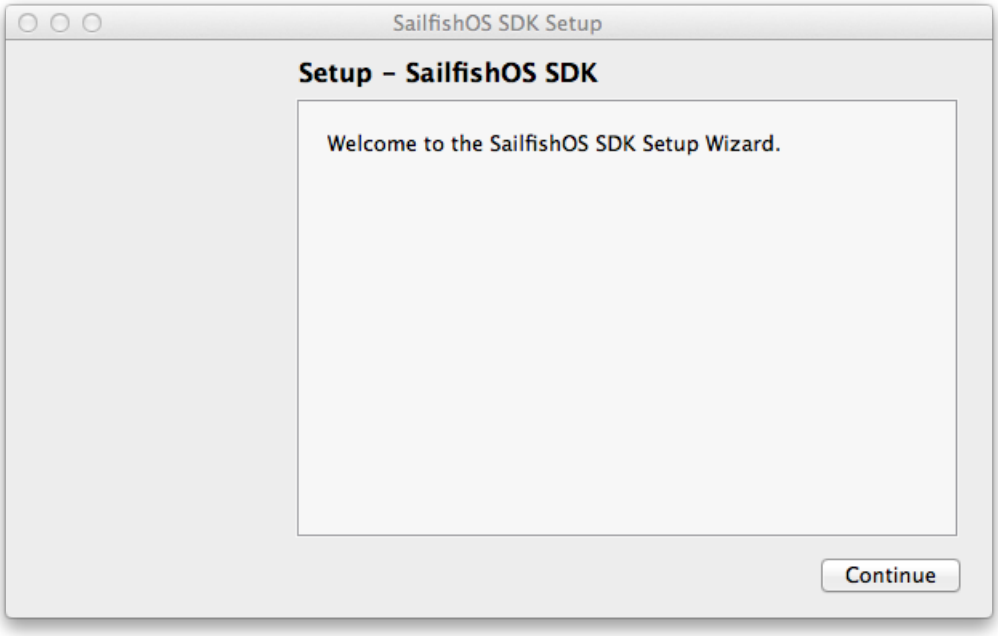

Figure 4: Install SailfishOS SDK, step 1.

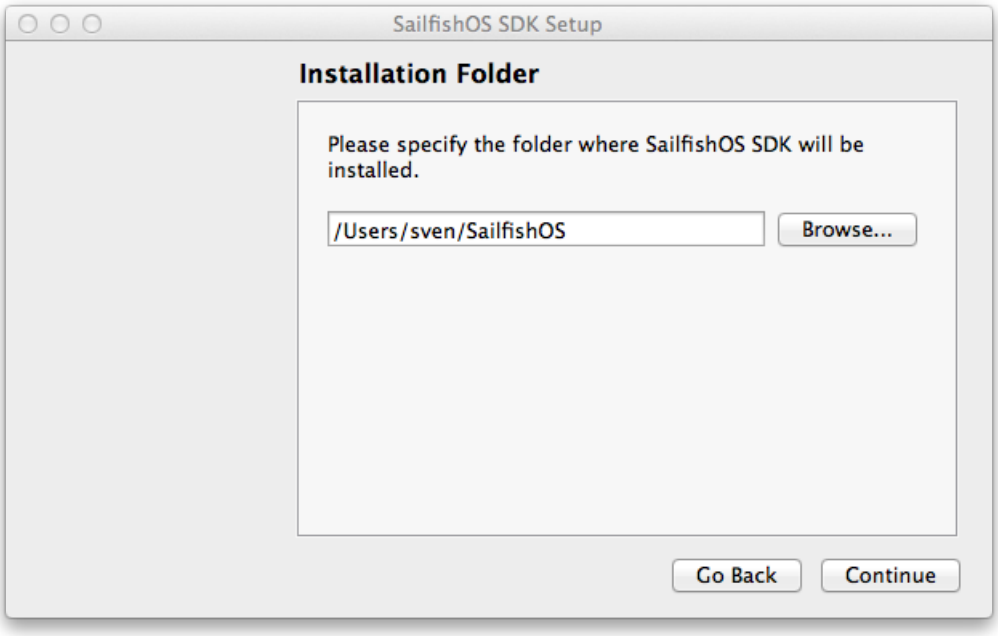

Figure 5: Install SailfishOS SDK, step 2.

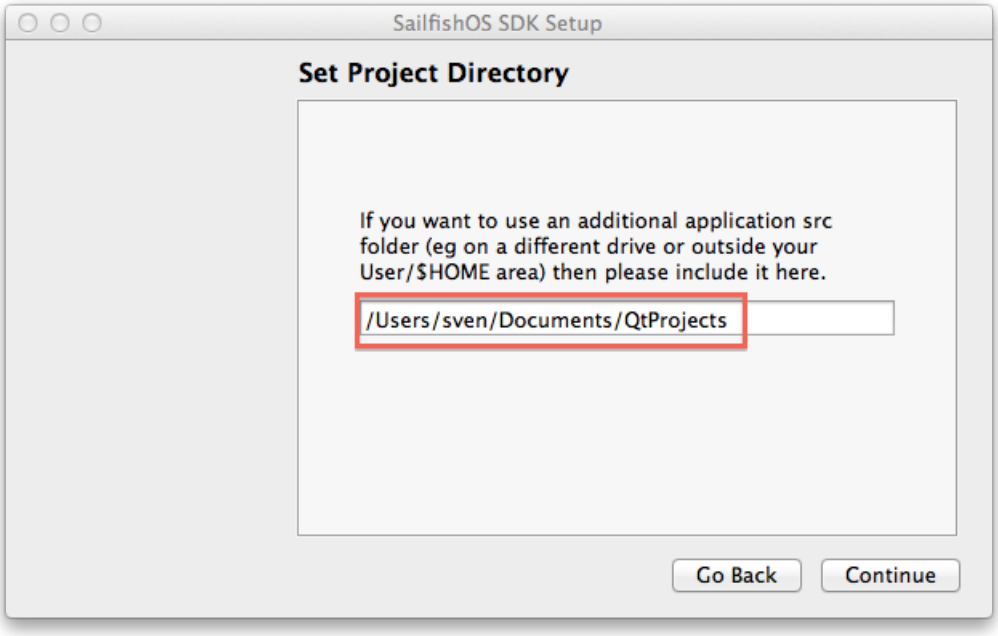

Figure 6: Install SailfishOS SDK, step 3. Enter the path to your source code. There is no folder selector dialog, you must enter it by hand.

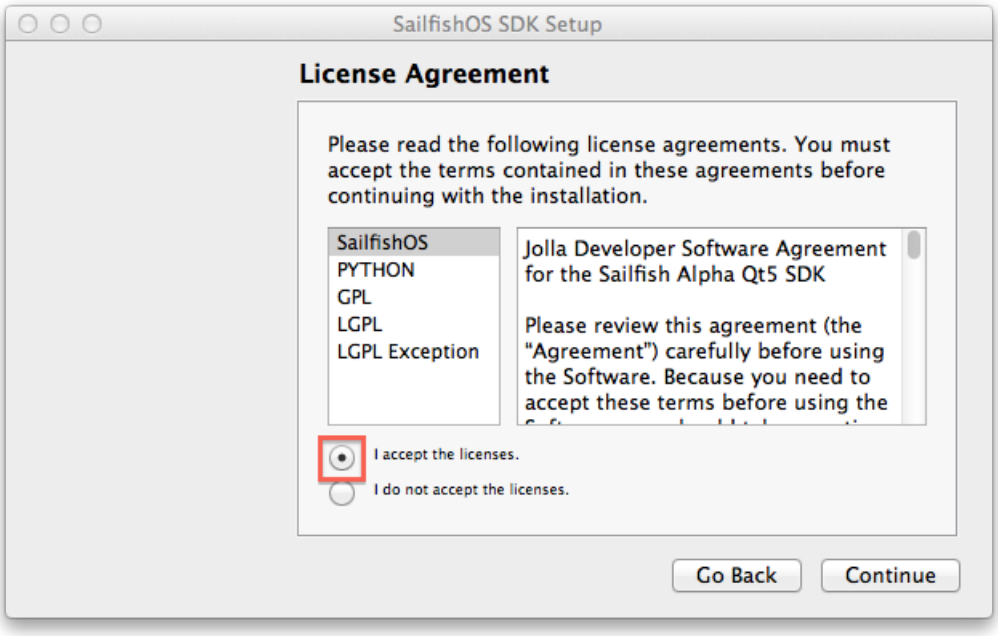

Figure 7: Install SailfishOS SDK, step 4.

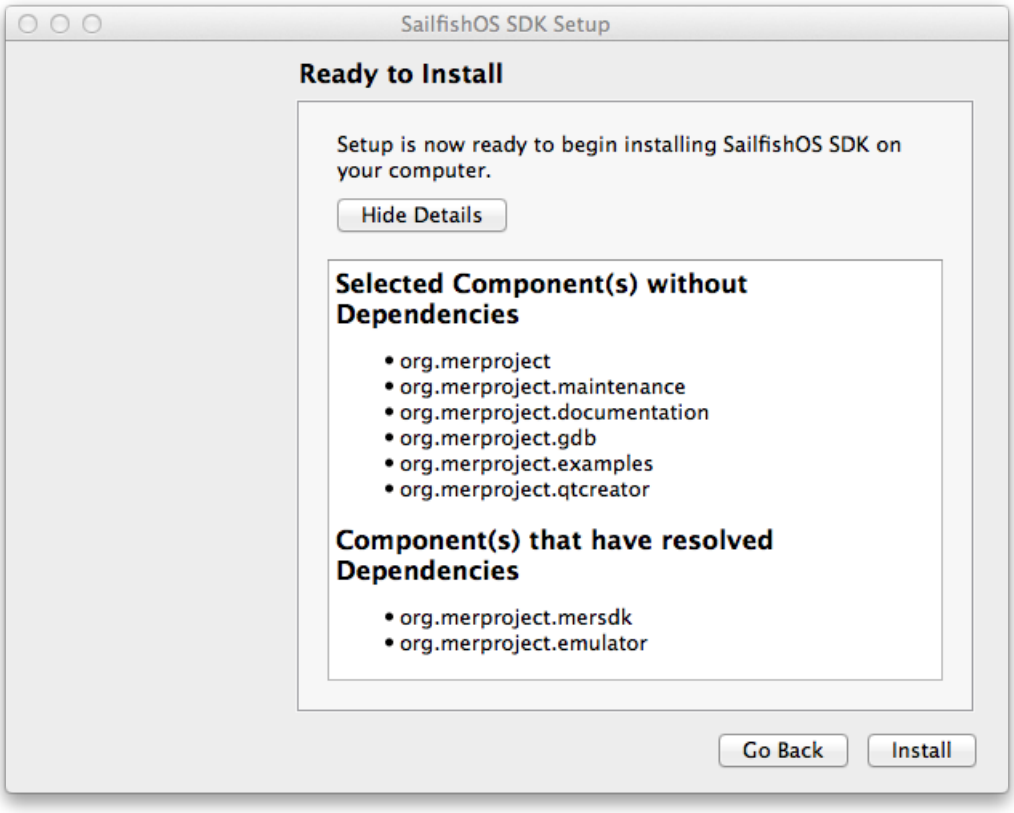

Figure 8: Install SailfishOS SDK, step 5.

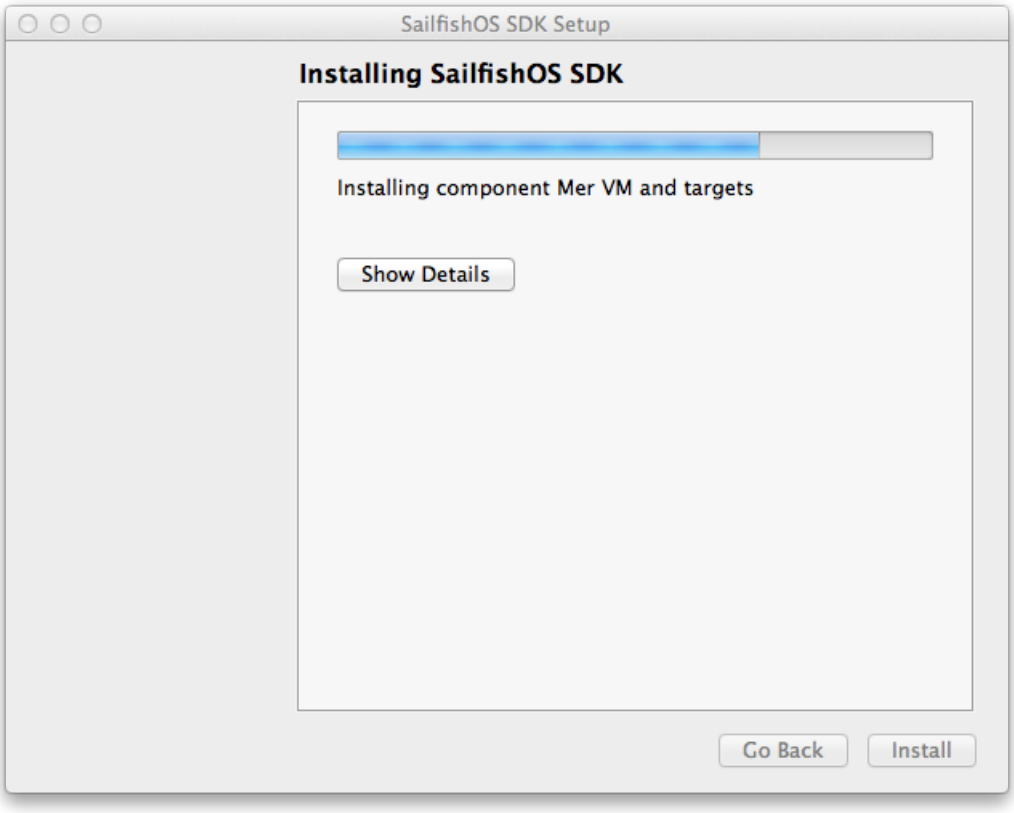

Figure 9: Install SailfishOS SDK, step 6.

Figure 10: Install SailfishOS SDK, step 7.

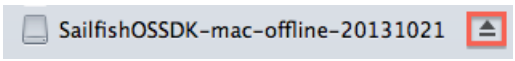

Figure 11: Unmount SailfishOS SDK disk image, step 8.

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).

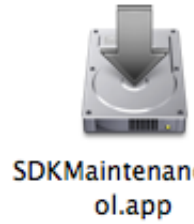

Figure 12: Inside the SailfishOS folder you find the maintenance application. Run it directly after the installation.

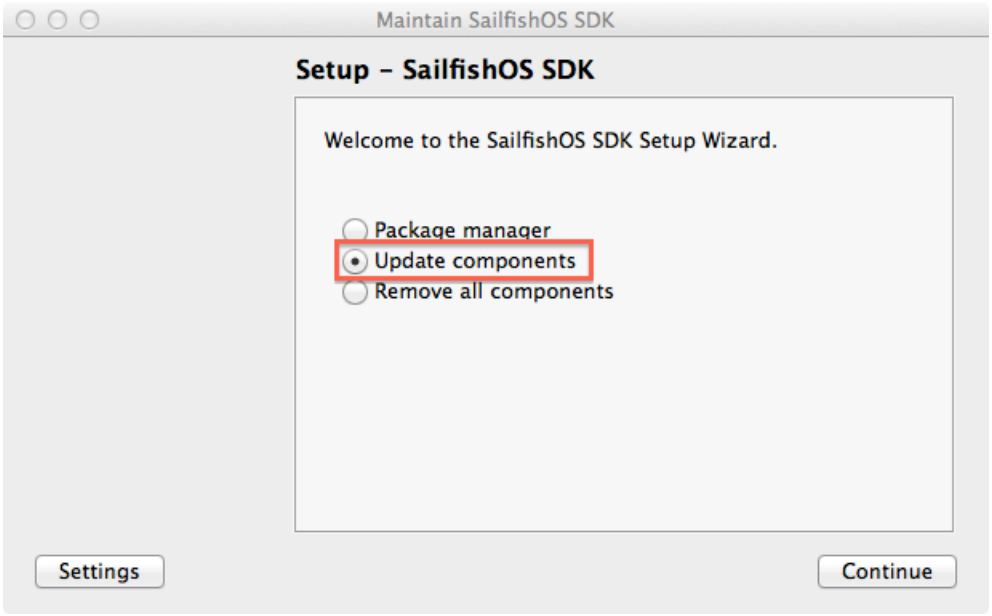

Figure 13: Update components, choose everything that is in store.

With the SDK comes QtCreator, a complete IDE for C++ development. This IDE is part of the Qt Framework[qt01] and is simply reused[1]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]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. Don't shoot in your foot here.

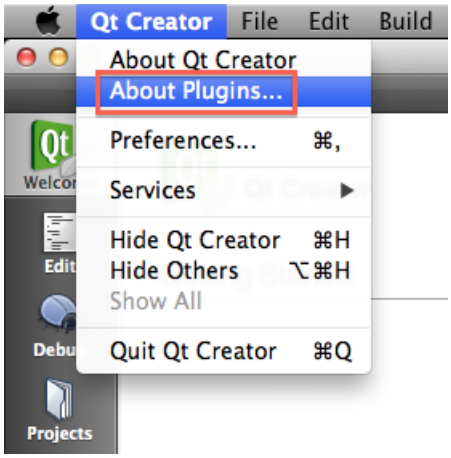

Figure 15: About plugins = manage plugins.

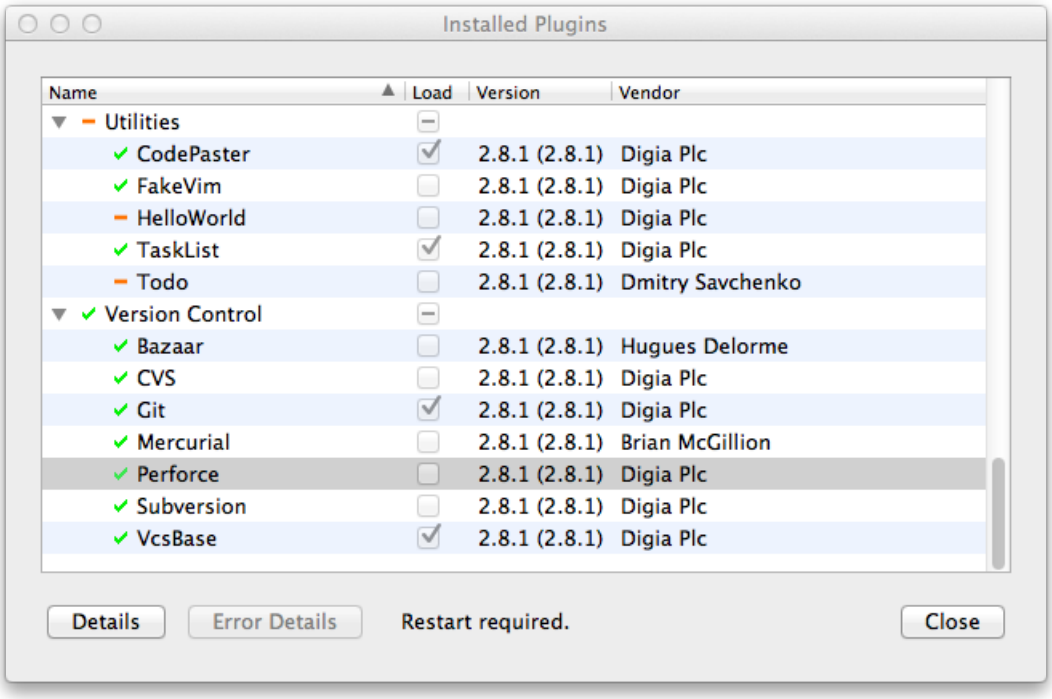

Figure 16: Deactivate every plugin you don't need.

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 4.

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]

Start the SDK from inside the QtCreator.

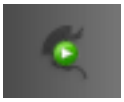

Figure 17: Starting the SDK.

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

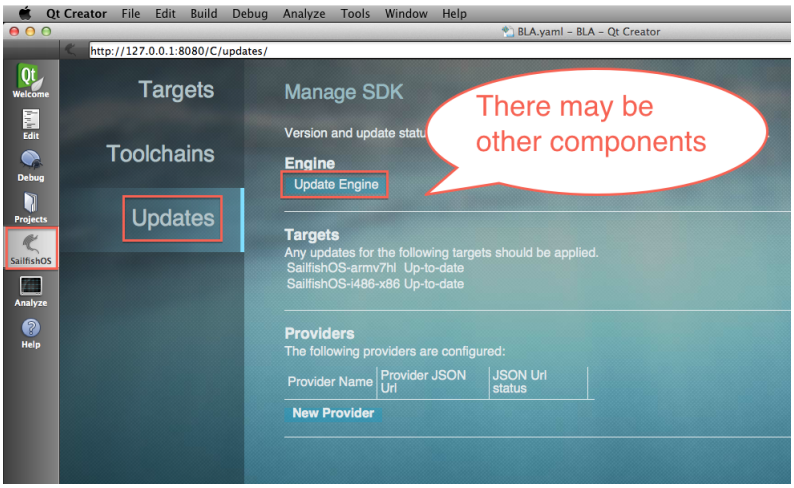

Figure 18: Updating the SDK.

Start the Emulator.

Figure 19: Starting the Emulator.

Compile and run your application.

Figure 20: Start the application.

Easy as pie, also see figure 39 on page 32. But what happens when and where if you click on those Icons? Look in section *Know your tools*.

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[2].

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

[2] 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.

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![3]

Here is what I've learned so far.

4.1 Technology stack

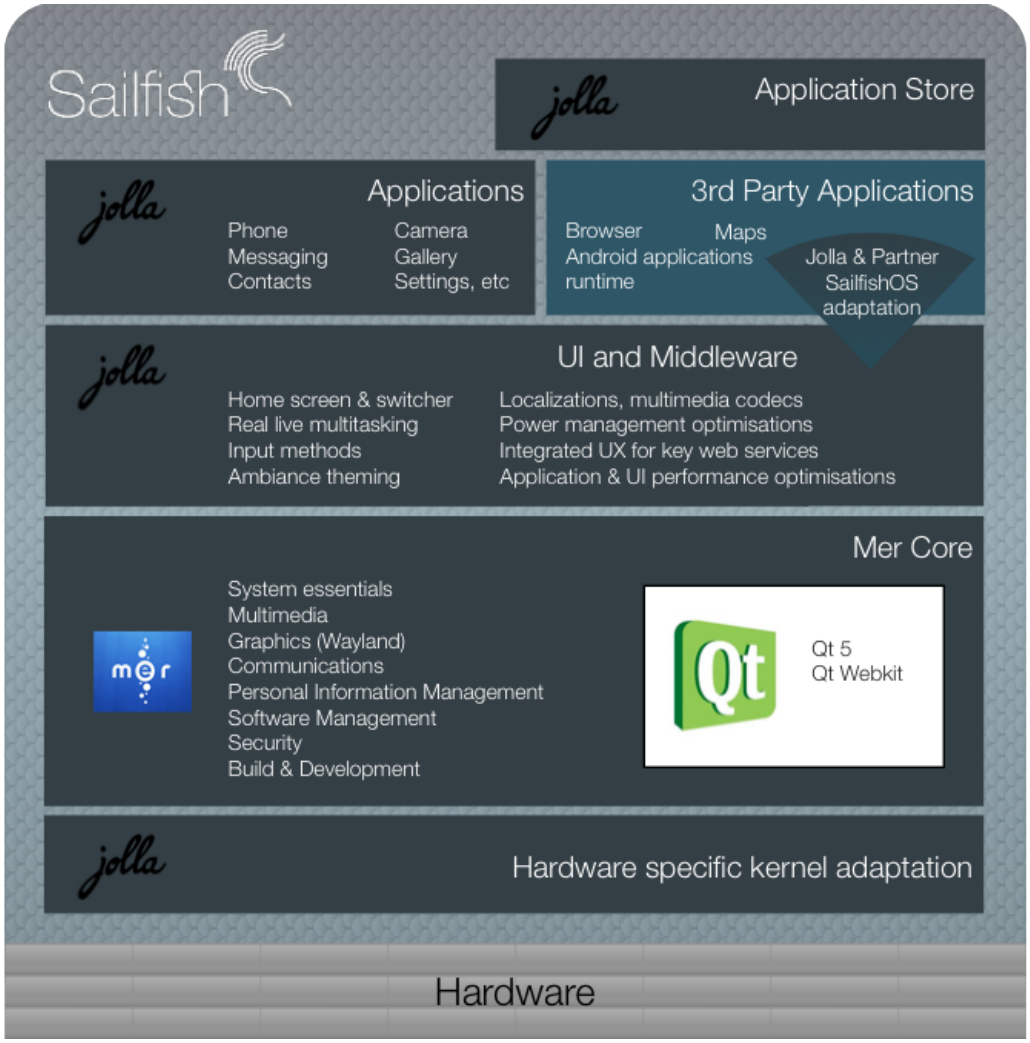

Figure 21: Sailfish architecture, taken from https://sailfishos.org/images/Sailfish_Architecture.png.

[3] Ok, not so short :-)

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[4].

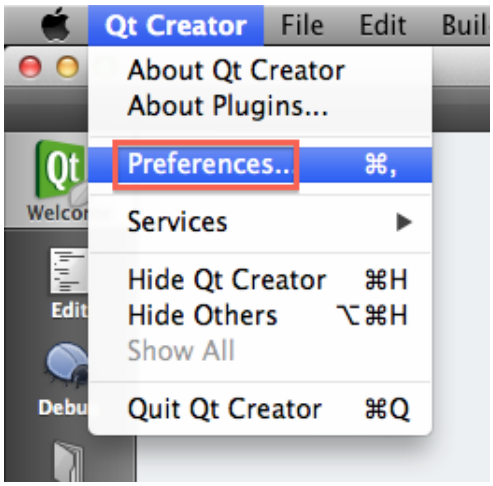

Figure 22: Open the QtCreator preferences.

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`. When we try to do some of the steps manually via command line, we will see that it often does not work but the same action succeeds

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

invoked by QtCreator. One of the reasons are those kits. This is the information hub that QtCreator uses to pull all information together and initiate its actions.

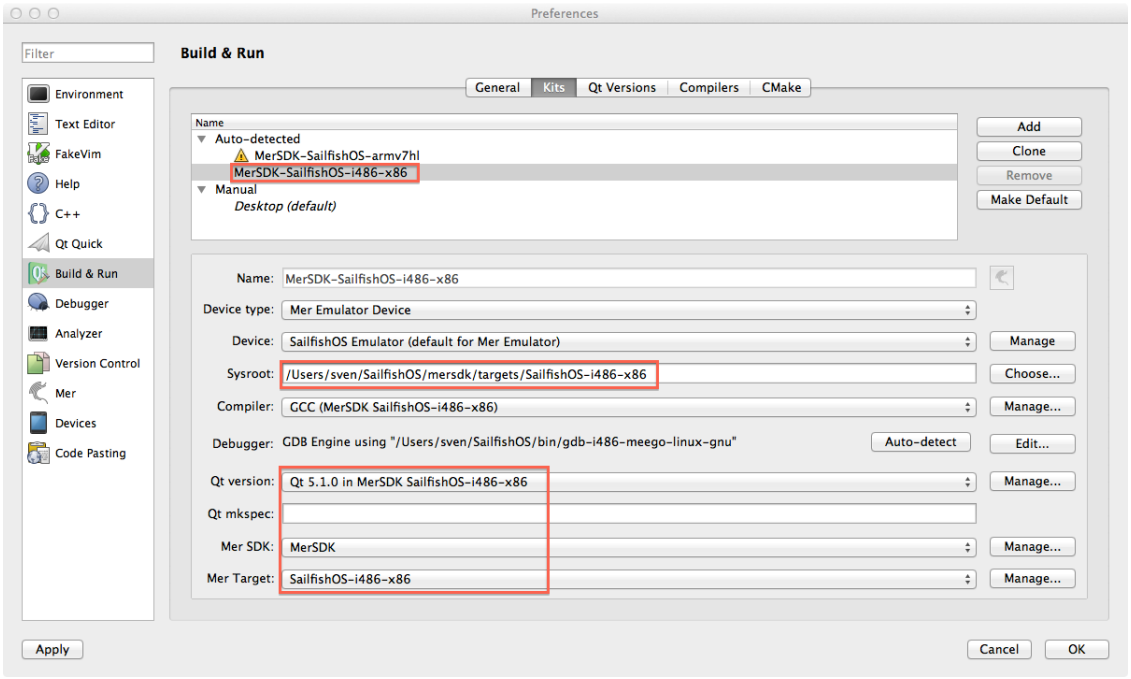

Figure 23: Preferences, kits tab.

4.2.2 qmake

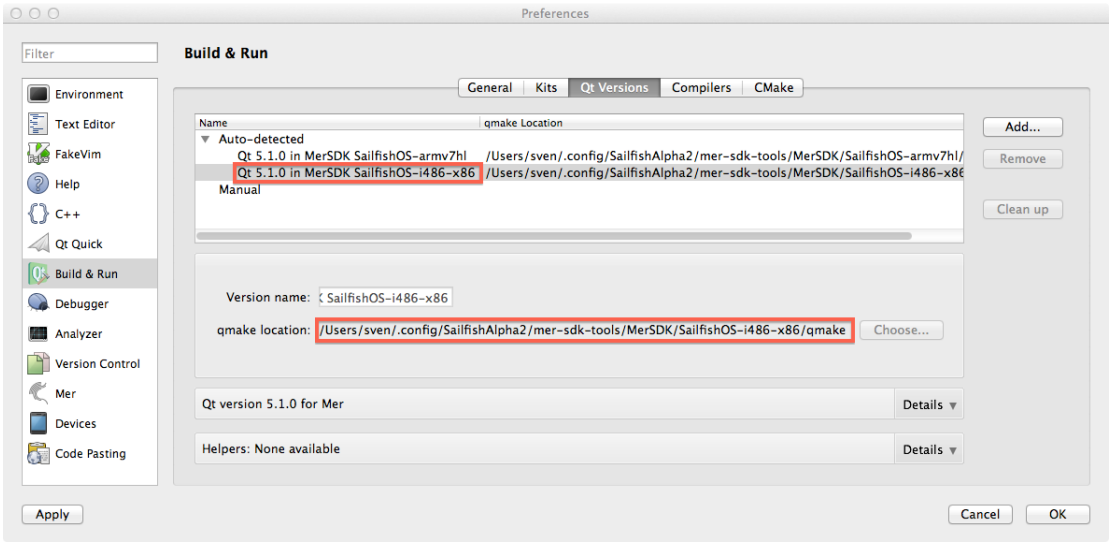

Figure 24: Preferences, QtVersions tab.

"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]

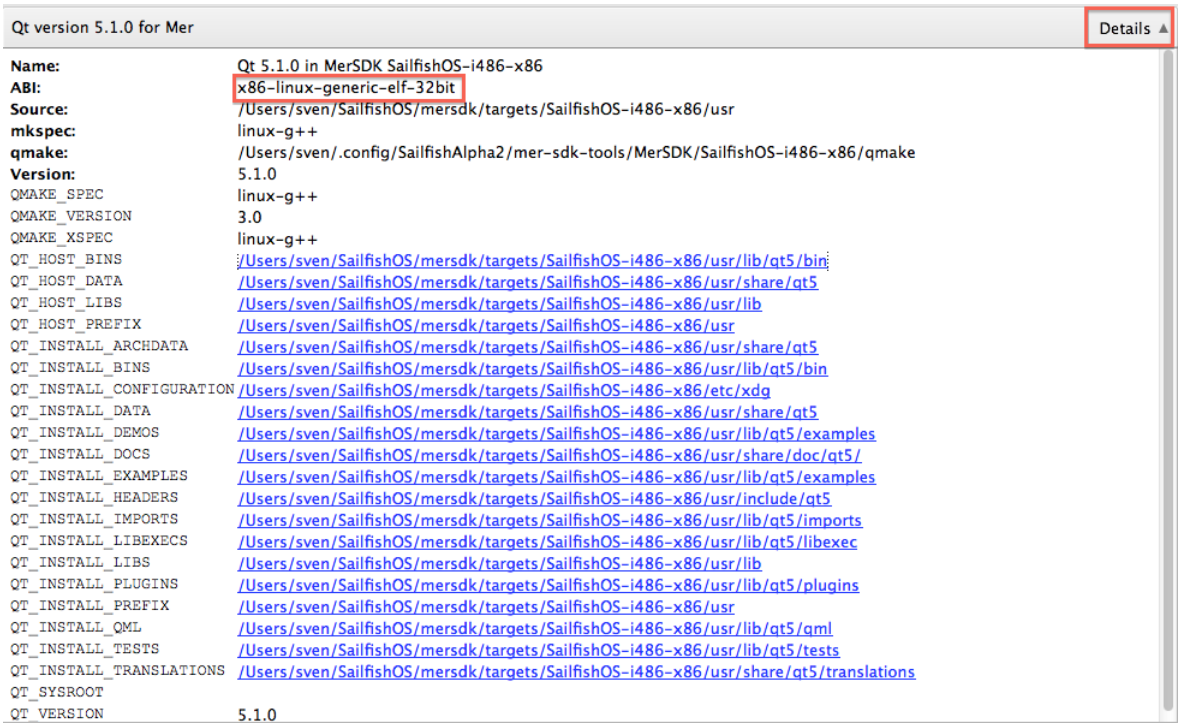

Figure 25: Preferences, QtVersions tab., qmake details

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.

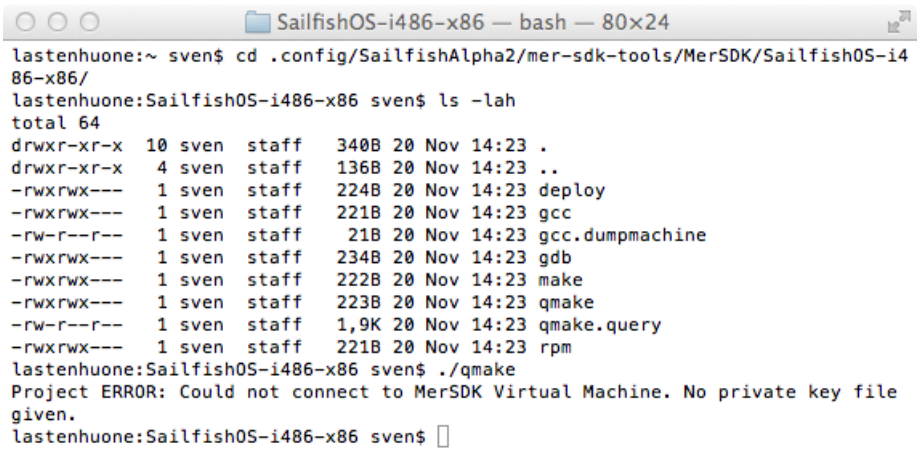

Figure 26: Running qmake from command line.

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.

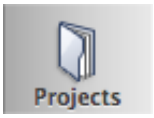

Figure 27: Project settings.

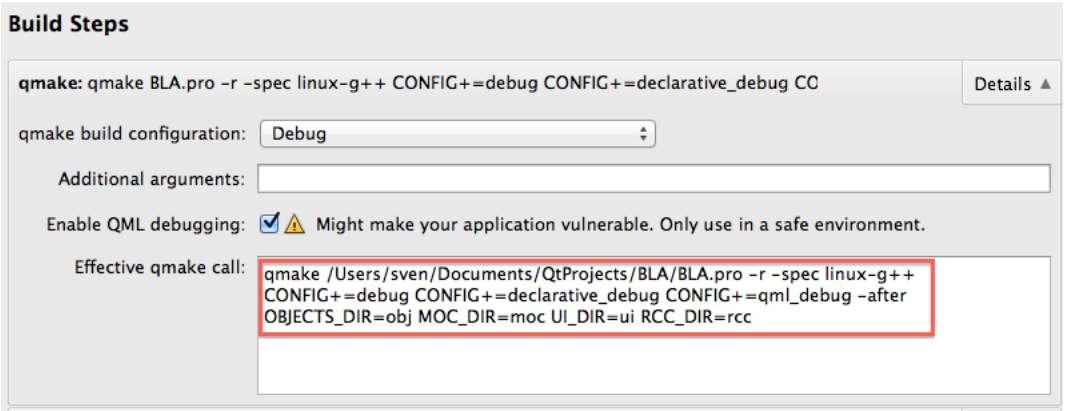

Figure 28: Build steps for qmake.

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

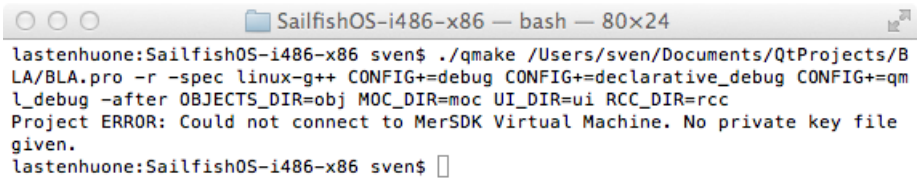

Figure 29: Running qmake from command line with parameters from build steps.

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

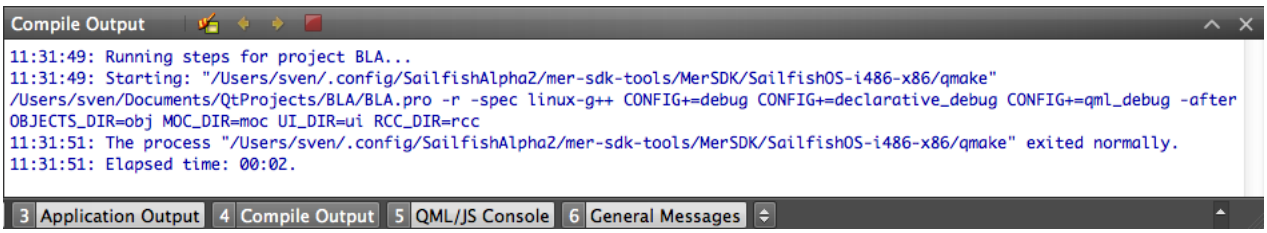

Figure 30: Running qmake from QtCreator (Build menu).

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

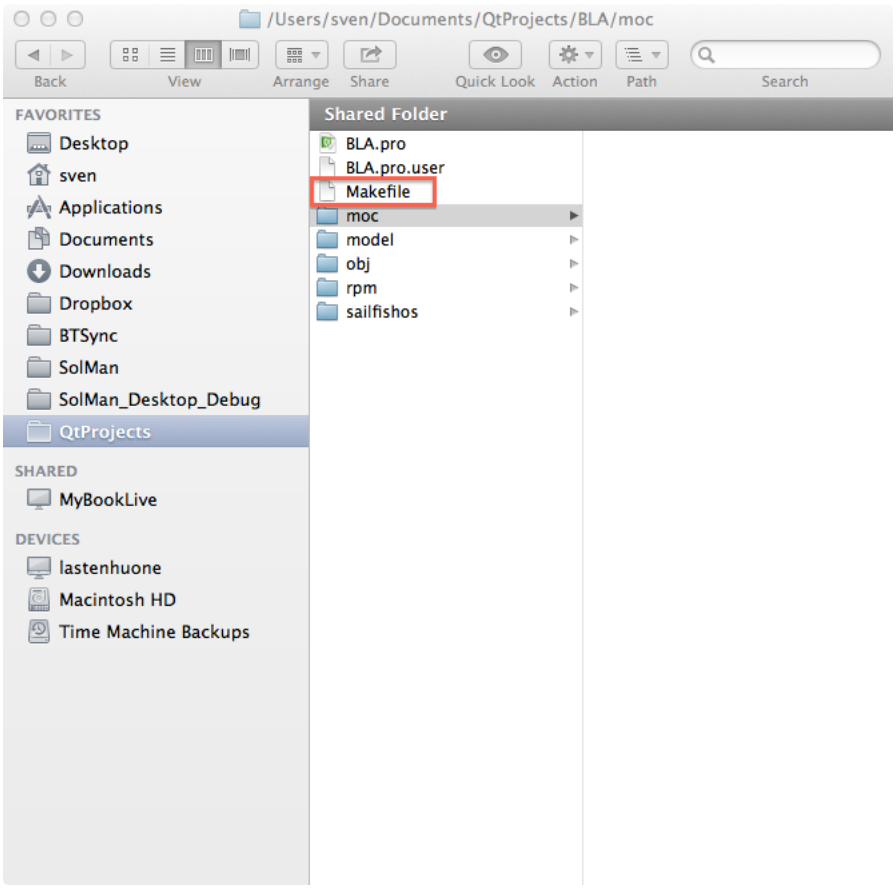

Figure 31: Result of running qmake from QtCreator (Build menu).

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?

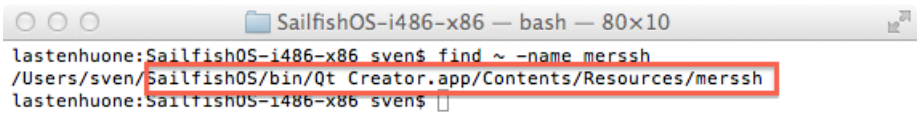

Figure 32: What is merssh?.

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

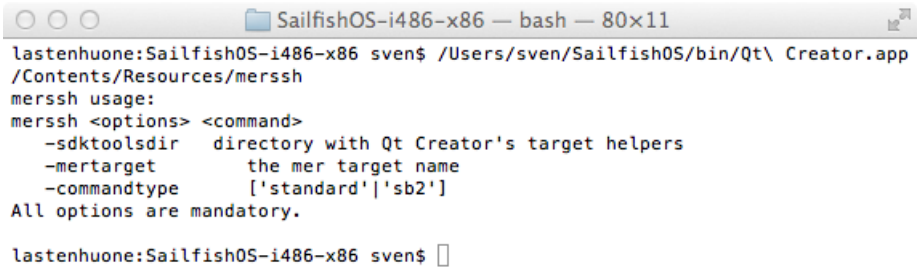

Figure 33: merssh invoked, what's it?.

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 38 on page 30.

More than that, it calls `sb2` which is short for Scratchbox2, have a look at section Scratchbox2 on page 30, 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].

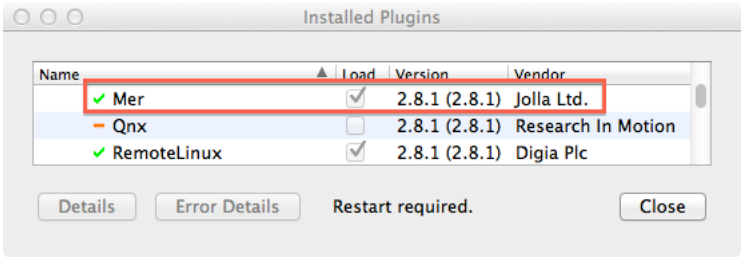

Figure 34: Mer plugin, maybe that's the source of merssh?.

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

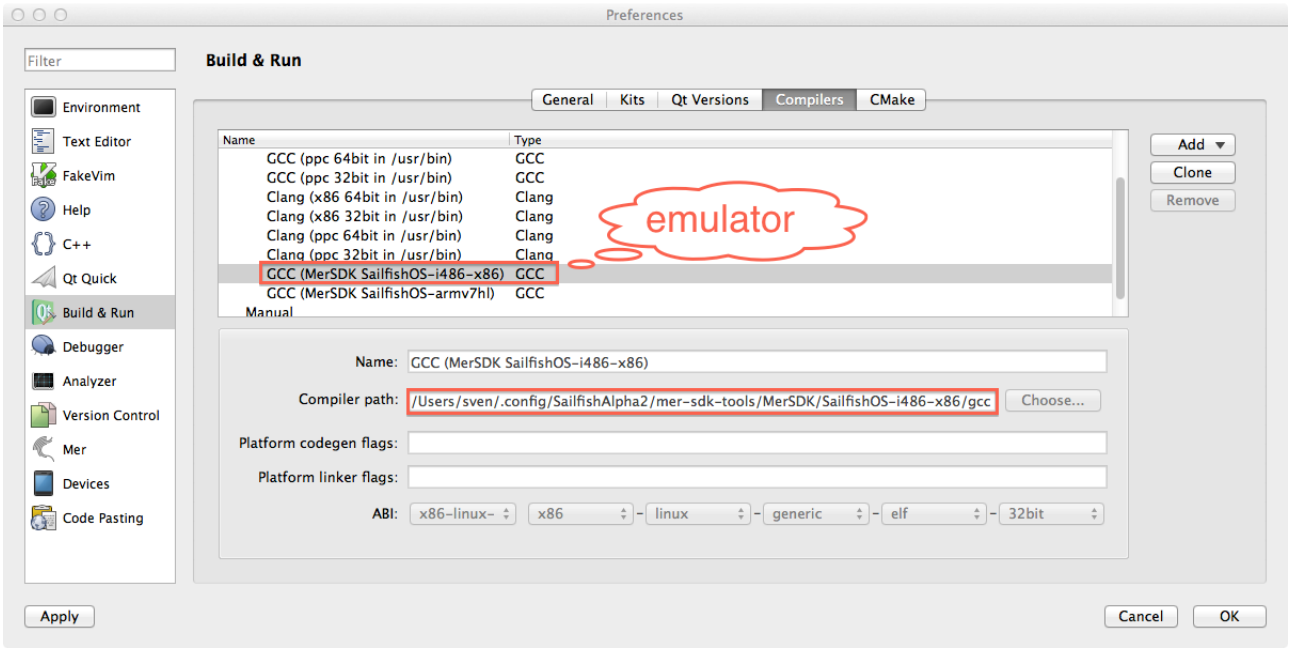

Figure 35: Preferences, compiler tab.

The SailfishOS SDK uses GCC as compiler. It is run inside the Mer build engine for cross compilation, see page 29. Stored on your development machine is only

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

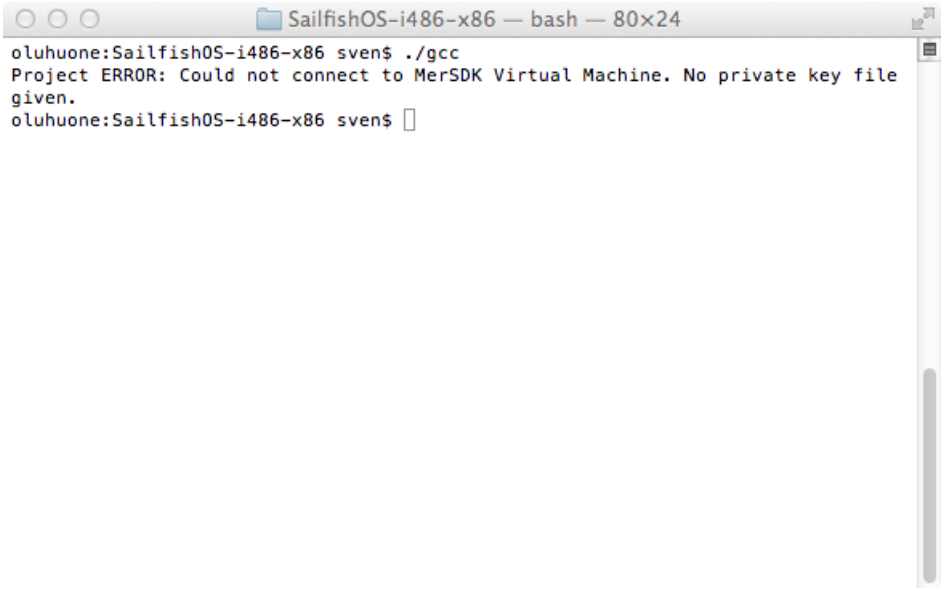

Figure 36: Running GCC from command line.

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 25.

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

```
1 #!/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
```

4.3 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 37: 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 18 on page 14.

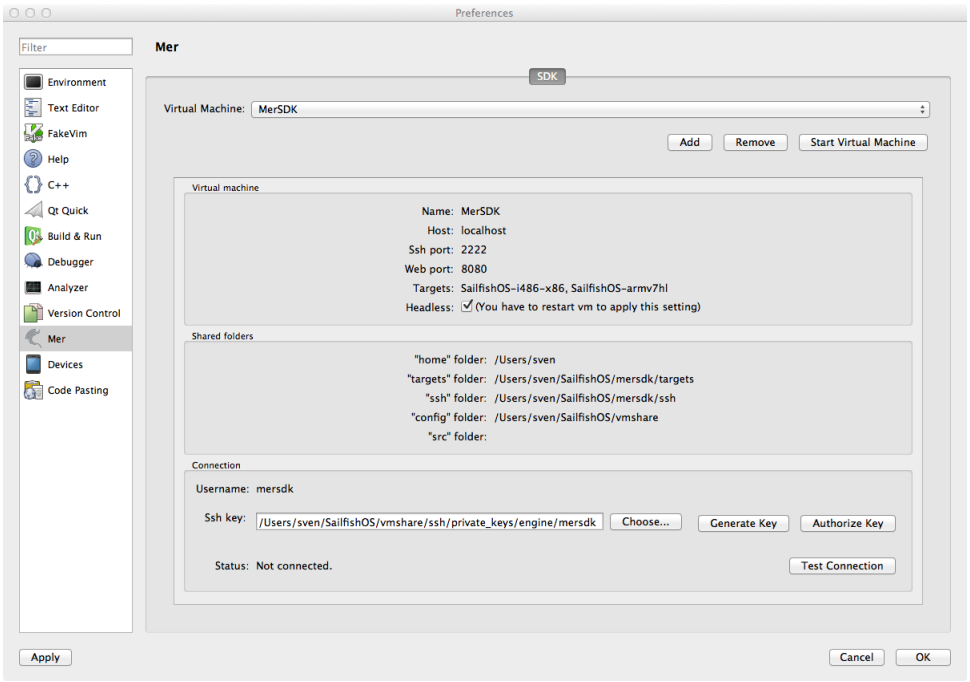

Figure 38: Preferences, Mer SDK - virtual machine.

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.

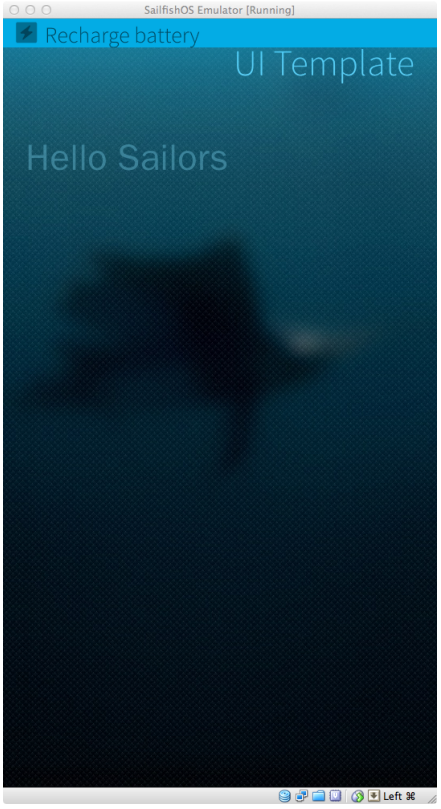

Figure 39: Emulator running the templated SailfishOS Qt Quick Application.

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].

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

TODO see also caveats on sailfish site

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.

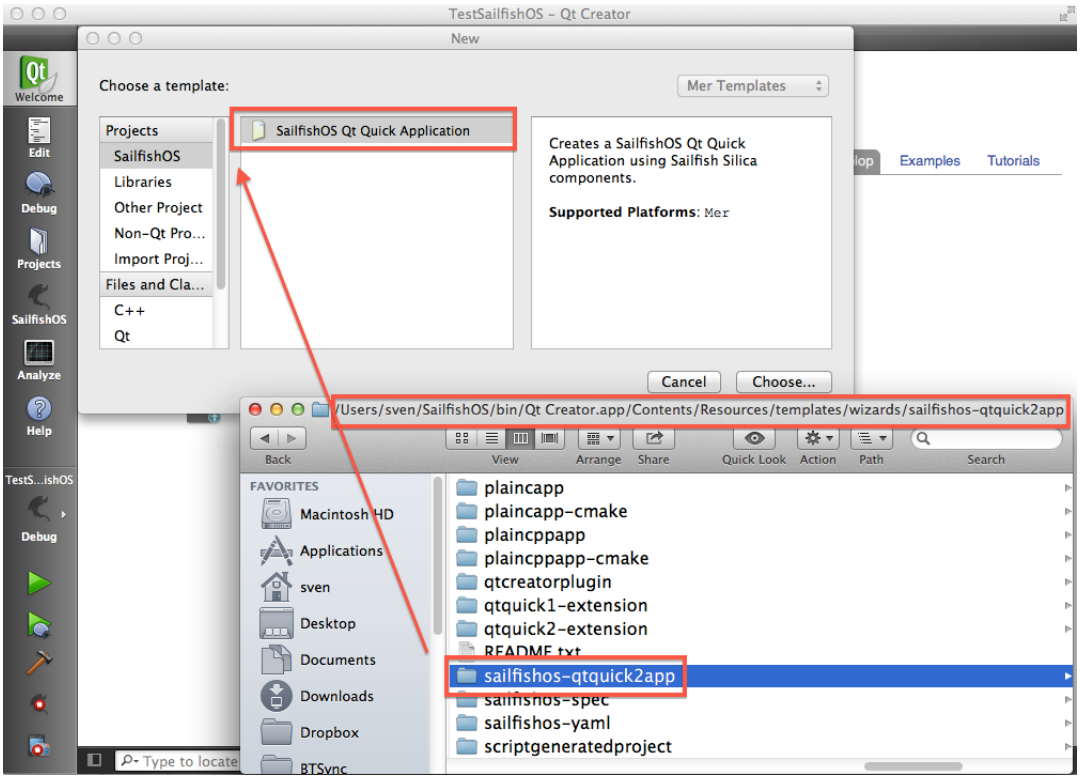

Figure 40: Template for a new SailfishOS Qt Quick Application.

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 for you on a real phone. It maybe just you fingers that are hiding the screen.

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 Community

8.0.1 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`.

8.0.2 Mer

At freenode `#mer`.

8.0.3 Nemo mobile

At freenode `#nemomobile`.

9 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`

[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
`TODO`

[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