Developing with Qt5 on iMX Developer's Kits

Embedded Artists

Embedded Artists AB
Davidshallsgatan 16
SE-211 45 Malmö
Sweden

http://www.EmbeddedArtists.com

Disclaimer

Feedback

We appreciate any feedback you may have for improvements on this document. Send your comments by using the contact form: www.embeddedartists.com/contact.

Trademarks

Table of Contents

1 Document Revision History

Revision	*Date*	*Description*
A	2017-01-02	First release

2 Introduction

Qt is a feature rich and cross-platform application framework developed by the Qt Company. Many associate Qt with GUI development, but it is possible to do much more than just the GUI using the Qt framework.

This document provides an overview of Qt and step-by-step instructions for setting up Qt Creator - the development environment for Qt. This is not a complete course in application development using the Qt framework. For this purpose please visit doc.qt.io.

Additional documentation you might need:

- The *Getting Started* document for the board you are using.
- The *Working with Yocto* document

2.1 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

`Constant width text` – is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the
development workstation, i.e., on the workstation where you edit,
configure and build Linux
```

```
# This field illustrates user input on the target hardware, i.e.,
input given to the terminal attached to the COM Board
```

```
This field is used to illustrate example code or excerpt from a
document.
```

This field is used to highlight important information

3 Overview of Qt

Qt is an application framework written in C++ and supported by many platforms such as Linux, Windows, and OS X. The application framework is mainly developed by the Qt Company, but other companies and individuals are also involved.

3.1 Licenses

Qt is available under both commercial **licenses** and open source licenses. The open source licenses are either GPL or LGPL3 depending on which part of Qt you are looking at. Most of the core libraries are under GPL while most of the APIs are LGPL3.

Please **note** that some modules and functionality is only available when using the commercial license.

More information about Qt licensing can be found at www.qt.io/faq/

3.2 User interfaces

Qt offers several ways of developing a user interface; Qt Widgets, Qt Quick, and Qt WebEngine. A short description of these technologies is available below, but for a more in-depth description and comparison please visit: doc.qt.io/qt-5/topics-ui.html.

3.2.1 Qt Widgets

The user interface is developed using the C++ programming language often in combination with the graphical tool Qt Designer. Qt Widgets is often used to create classic **desktop-style** user interfaces with a native look and feel. A native look and feel means that the GUI looks as it normally does on the platform, for example, as normal Windows applications when developing on a Windows platform.

Qt Widgets has not been designed to benefit from a Graphical Processing Unit (GPU) since it is using a raster paint engine (QPainter) when drawing its graphical elements.

3.2.2 Qt Quick

Qt Quick is described as a module you can use to develop fluid and dynamic user interfaces with **animations and effects** - the type of GUI you usually see in mobile devices.

With Qt Quick the user interface is developed using QML – a declarative language with a JSON-like syntax. The whole application can be written in QML, but normally only the GUI is written in QML and the other parts of the application is written in C++. As with Qt Widgets there is tool support in the form of Qt Quick Designer that helps you develop the GUI.

Qt Quick has been designed to be **hardware accelerated** and has because of this a dependency towards OpenGL.

NOTE: Not all of Embedded Artists COM boards support QtQuick when using the open source version of Qt. See section 4.3.1 for more information.

3.2.3 Qt WebEngine

Qt includes a web engine that lets you include web content into the application. This means that the entire user interface or parts of it can be developed using web technologies such as HTML, CSS, and JavaScript. For more information about developing with web content visit the link below.

doc.qt.io/qt-5/topics-web-content.html

3.3 Qt Essentials

The foundation of Qt is called Qt Essentials and includes the user interface functionality described above, but also a lot more. The table below contains a short description of some of the modules in Qt essentials. For a more complete list and description visit the link below.

doc.qt.io/qt-5/qtmodules.html

Module	Description
Qt Core	Non-graphical classes used by other modules. This module, for example, contains the object communication functionality called "signals and slots". It also defines the object model and property system.
Qt Multimedia	Audio, video, radio, and camera functionality. The exact support depends on the underlying platform.
Qt Network	This module contains classes that provide networking (TCP/IP) functionality.
Qt SQL	Database functionality is offered through this module.
Qt Test	This module provides classes for unit testing.

In addition to all the modules in Qt essentials there are also many **add-on** modules. Please visit the link above the table for more details.

4 Qt on iMX Developer's Kits

4.1 Yocto image

The Yocto image that adds Qt5 support to the target is fsl-image-qt5. See the document "Working with Yocto to build Linux" for more information about Yocto and how to build and deploy images.

Before building this image we need to add support for an SFTP server. The SFTP server allows us to download a Qt application to the target from within Qt Creator.

Open the file `<build dir>/conf/local.conf`

```
$ nano conf/local.conf
```

Locate the IMAGE_INSTALL_append variable and add the following two lines:

```
    openssh-sftp \
    openssh-sftp-server \
```

Now you can build the Qt5 image and an SFTP server will be included.

```
$ bitbake fsl-image-qt5
```

4.2 Qt version

Embedded Artists provides two Yocto branches (BSP versions). These are mentioned in the document "Working with Yocto to build Linux". It is Qt **version 5.5.0** that is used in the 3.14.52 branch and **version 5.5.1** that is used in the 4.1.15 branch.

There is a Qt layer called meta-qt5 in Yocto and the version that is used can be found in `meta-qt5/recipes-qt/qt5/qt5-git.inc`.

4.3 Supported modules

The table below lists Qt modules and if they are built for a COM board, or not. Even though a module is available, all functionality hasn't been tested by Embedded Artists. The table has been compiled by listing the Qt libraries available in the root file system by running the command below.

```
$ cd /usr/lib
$ ll | grep Qt
```

The second column (iMX6Q/D6DL/SX) in the table reflects the Qt support for iMX6 Quad, iMX6 Dual, iMX6 DualLite and iMX6 SoloX. The third column (iMX7D/iMX6UL) reflects the Qt support for iMX7 Dual COM, iMX7 Dual uCOM, and iMX6 UltraLite.

Module	iMX6Q/D/DL/SX	iMX7D/iMX6UL
Qt 3D	X	
Qt Bluetooth	X	
Qt Concurrent	X	X
Qt Core	X	X
Qt D-Bus	X	X

Qt GUI	X	X
Qt Location	X	
Qt Multimedia	X	
Qt Multimedia Widgets	X	
Qt Network	X	X
Qt NFC	X	
Qt OpenGL	X	
Qt Platform	X	
Qt Positioning	X	
Qt Print Support	X	X
Qt QML	X	
Qt Quick	X	
Qt Quick Widgets	X	
Qt Script	X	
Qt Sensors	X	
Qt Serial Port	X	
Qt SQL	X	X
Qt SVG	X	
Qt Test	X	X
Qt WebChannel	X	
Qt WebSockets	X	
Qt Widgets	X	X
Qt XML	X	X
Qt XML Patterns	X	

4.3.1 Qt Quick

As mentioned in section 3.2.2 Qt Quick is a module that lets you create nice looking user interfaces. Because of this many want to use Qt Quick. This module however depends on OpenGL and a graphical processing unit (GPU) and is therefore not supported on all Embedded Artists COM boards. It is not supported on iMX6 UltraLite COM, iMX7 Dual COM, and iMX7 Dual uCOM because the processors on these boards don't have a GPU.

Even though a board lacks a GPU it could be possible to use Qt Quick with a software rendered graphical engine. The performance would however be worse than on board that has a GPU. NXP has therefore decided not to include Qt Quick in the Qt recipes in Yocto.

If you would like to use Qt Quick on boards without a GPU you should have a look at Qt's module "**Qt Quick 2D Renderer**". This is an alternative renderer for Qt Quick that uses the raster paint engine instead of OpenGL. Please note that this module is only available in the commercial versions of Qt. For more information visit the link below.

doc.qt.io/QtQuick2DRenderer/

5 Setup QtCreator

The instructions in this section have been tested on a virtual machine running **lubuntu 16.04**. The document "Working with Yocto to build Linux" has a chapter that explains how to create a VMware based virtual machine running lubuntu.

If you are an experienced Linux user it shouldn't be a problem getting Qt Creator to run on another Linux distribution with the instructions below as a guideline.

5.1 Install toolchain

To be able to build Qt applications that will run on Embedded Artists iMX based COM boards a toolchain is needed. The toolchain contains cross compiler, linker, Qt headers and needed libraries.

The toolchain can be built in Yocto using the **meta-toolchain-qt5** image. See the document "Working with Yocto to build Linux" for more information about Yocto.

```
$ bitbake meta-toolchain-qt5
```

The build will result in a file located at `<build directory>/tmp/deploy/sdk`. The exact name of the file depends on several parameters, but in our example it is called:

```
fsl-imx-fb-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-vfp-neon-
toolchain-4.1.15-1.2.0.sh
```

Part of file name	Description
`fsl-imx-fb`	The distribution (DISTRO parameter) used when initializing the build.
`x86_64`	Architecture of the host computer. In this example a **64-bit** Intel x86 platform
`4.1.15-1.2.0`	BSP version

It is recommended to build this toolchain on your host computer where you will do the Qt development, but if you have a 64-bit Intel x86 based host computer you can download a pre-built version from imx.embeddedartsits.com.

```
$ wget imx.embeddedartists.com/common/fsl-imx-fb-glibc-x86_64-
meta-toolchain-qt5-cortexa9hf-vfp-neon-toolchain-4.1.15-1.2.0.sh
```

Install the toolchain. It is recommended to use the default settings (such as installation path) when installing.

```
$ sudo ./fsl-imx-fb-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-
vfp-neon-toolchain-4.1.15-1.2.0.sh
```

5.2 Install Qt Creator

Offline installers of Qt Creator are available at: download.qt.io/official_releases/qt/5.7/5.7.0/. The instructions below are based on instructions from wiki.qt.io/Install_Qt_5_on_Ubuntu.

Download installer

```
$ wget download.qt.io/official_releases/qt/5.7/5.7.0/qt-
opensource-linux-x64-5.7.0.run
```

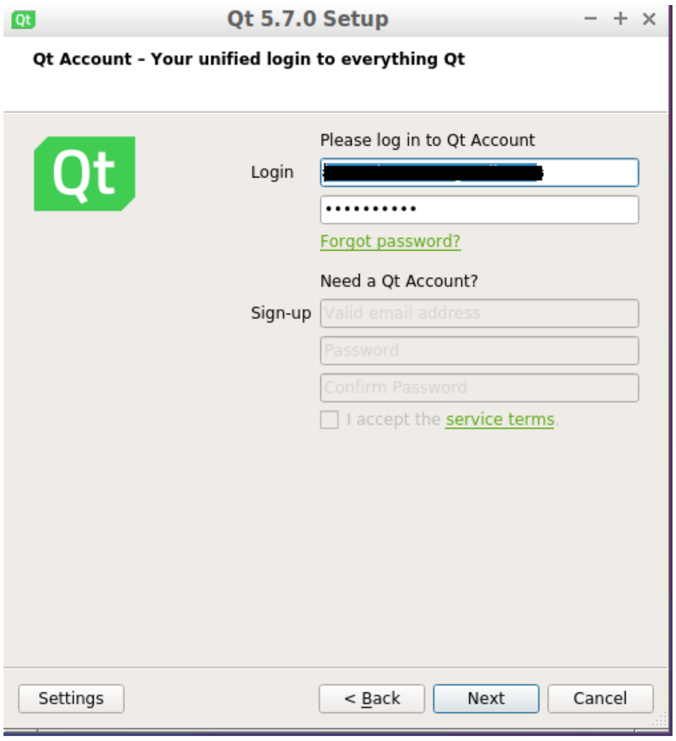
Adjust permission

```
$ chmod +x qt-opensource-linux-x64-5.7.0.run
```

Run installer

```
$ ./qt-opensource-linux-x64-5.7.0.run
```

In general the default settings have been used when installing Qt Creator. Please note that a Qt account is required when installing Qt Creator as shown in Figure 1. Choose an existing account or create a new account. There is no cost associated with a Qt account.


Qt 5.7.0 Setup

Qt Account - Your unified login to everything Qt

Please log in to Qt Account

Login [redacted]

••••••••••

Forgot password?

Need a Qt Account?

Sign-up Valid email address

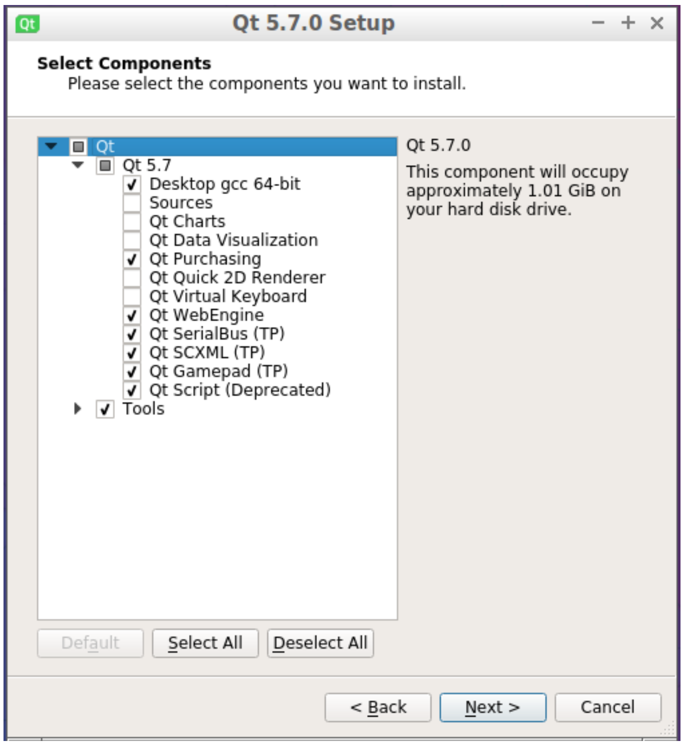
Password

Confirm Password

☐ I accept the service terms.

Figure 1 - Log in to a Qt account

The default components to install were selected as shown in Figure 2. If you plan to choose other components select these at this step. Please note that some components may only be available when using a commercial version of Qt.


Qt 5.7.0 Setup

Select Components
Please select the components you want to install.

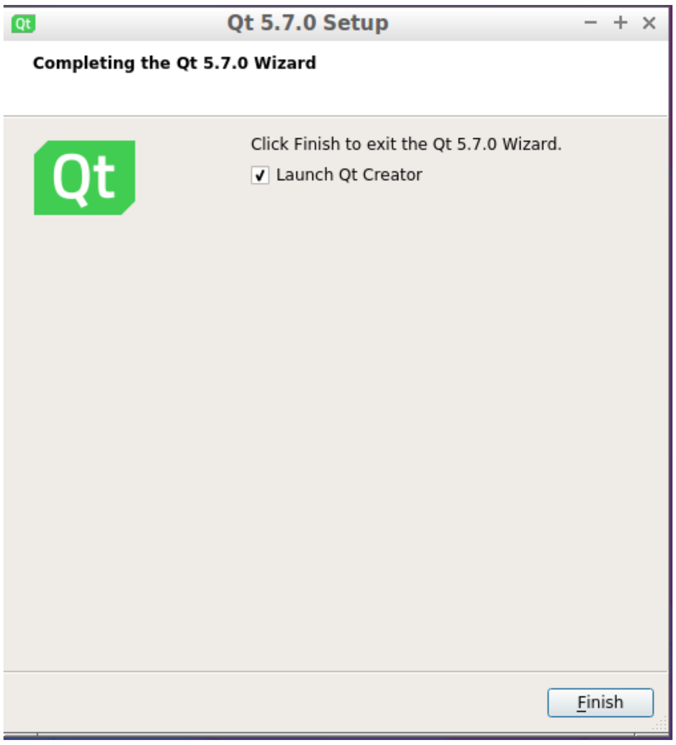
- [■] Qt
  - [■] Qt 5.7
    - [✓] Desktop gcc 64-bit
    - [ ] Sources
    - [ ] Qt Charts
    - [ ] Qt Data Visualization
    - [✓] Qt Purchasing
    - [ ] Qt Quick 2D Renderer
    - [ ] Qt Virtual Keyboard
    - [✓] Qt WebEngine
    - [✓] Qt SerialBus (TP)
    - [✓] Qt SCXML (TP)
    - [✓] Qt Gamepad (TP)
    - [✓] Qt Script (Deprecated)
  - [✓] Tools

Qt 5.7.0

This component will occupy approximately 1.01 GiB on your hard disk drive.

Figure 2 - Components to install

When Qt Creator has been installed select to launch it and click the "Finish" button and then continue to the next section of this document.


Qt 5.7.0 Setup

Completing the Qt 5.7.0 Wizard

Click Finish to exit the Qt 5.7.0 Wizard.

[✓] Launch Qt Creator

Figure 3 - Finish install of QtCreator

5.3 Configure Qt Creator

Qt Qreator needs to be configured to use the toolchain that was previously installed. Begin by opening the Options dialog: Tools → Options.

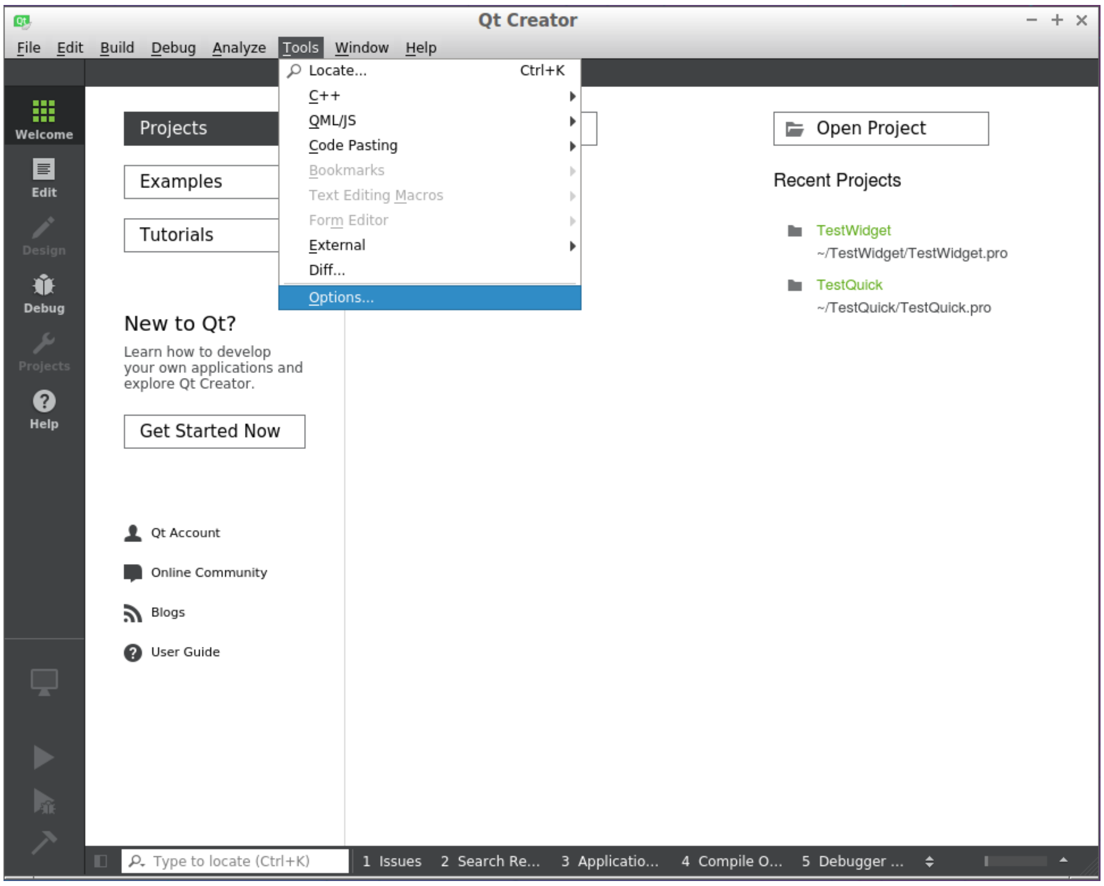
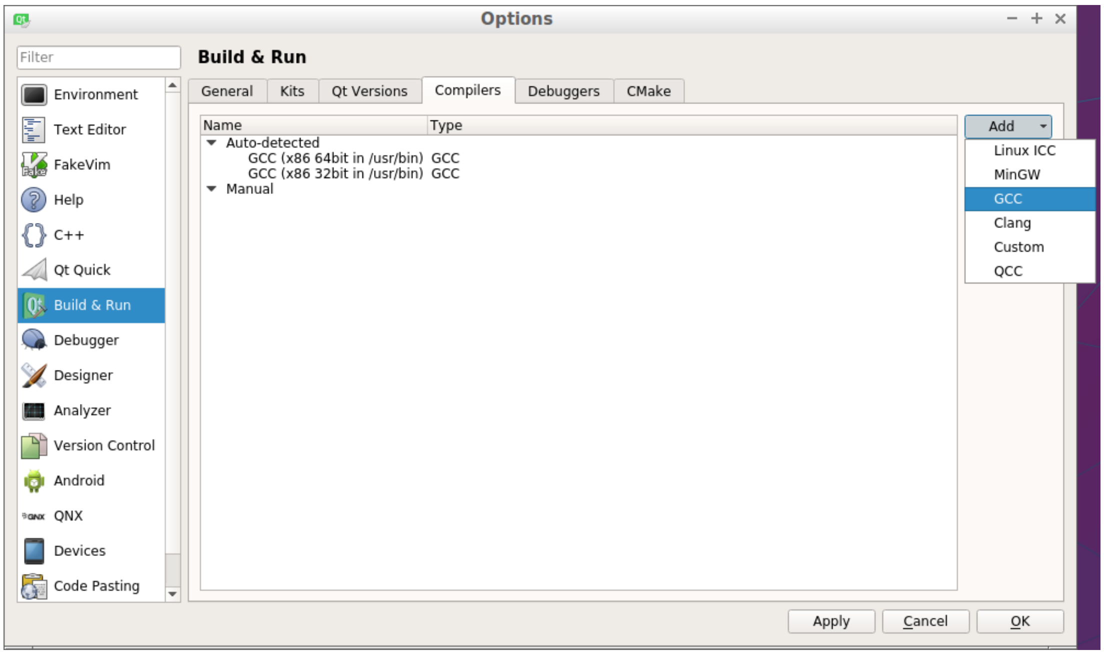


Figure 4 - Options under Tools menu

Select "Build & Run" in the left menu and then the "Compilers" tab. Click the "Add" button and then GCC as shown in Figure 5 below.

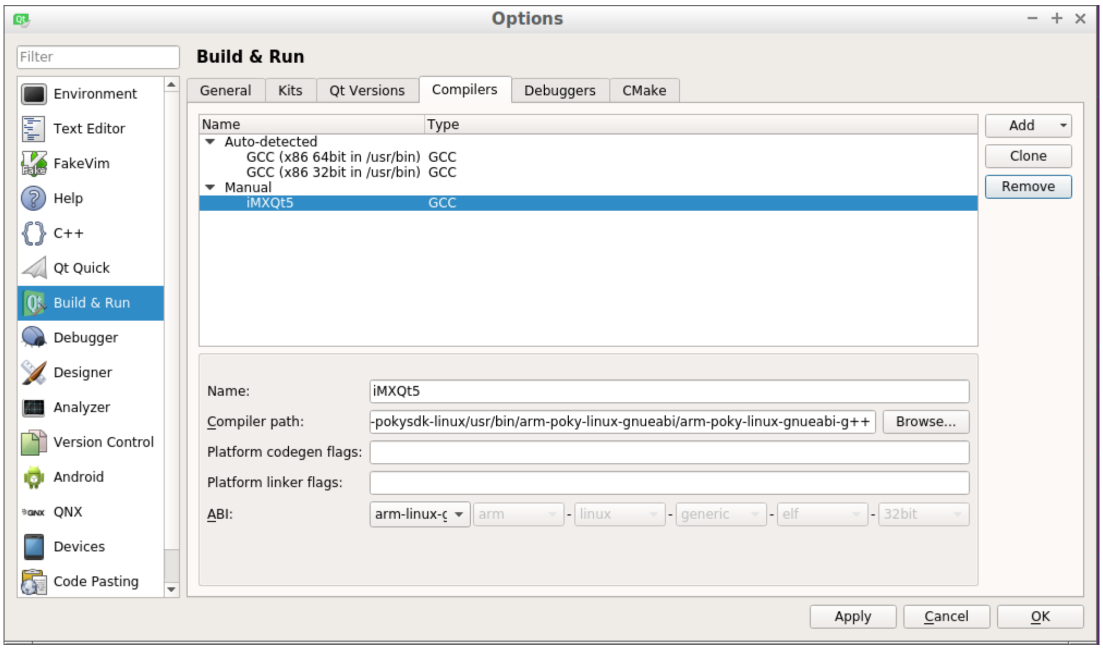


Name	Type
Auto-detected	
GCC (x86 64bit in /usr/bin)	GCC
GCC (x86 32bit in /usr/bin)	GCC
Manual	

Figure 5 - Add new compiler

Choose a name for the compiler and then select the path to the compiler. In this example the compiler is located at:

```
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-
linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++
```



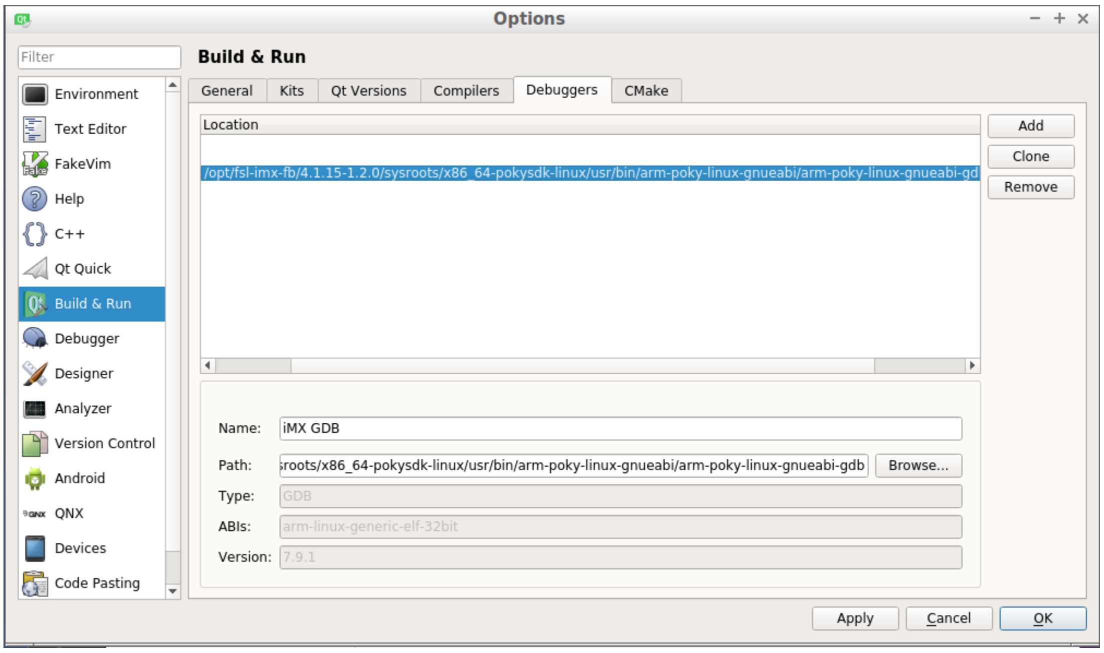
Name	Type
Auto-detected	
GCC (x86 64bit in /usr/bin)	GCC
GCC (x86 32bit in /usr/bin)	GCC
Manual	
iMXQt5	GCC

Name: iMXQt5
Compiler path: -pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++
Platform codegen flags:
Platform linker flags:
ABI: arm-linux-g arm - linux - generic - elf - 32bit

Figure 6 - Compilers tab

Go to the "Debuggers" tab as shown in Figure 7 and click the "Add" button. Select the path to the GDB debugger. In this example it is located at:

```
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-
linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
```



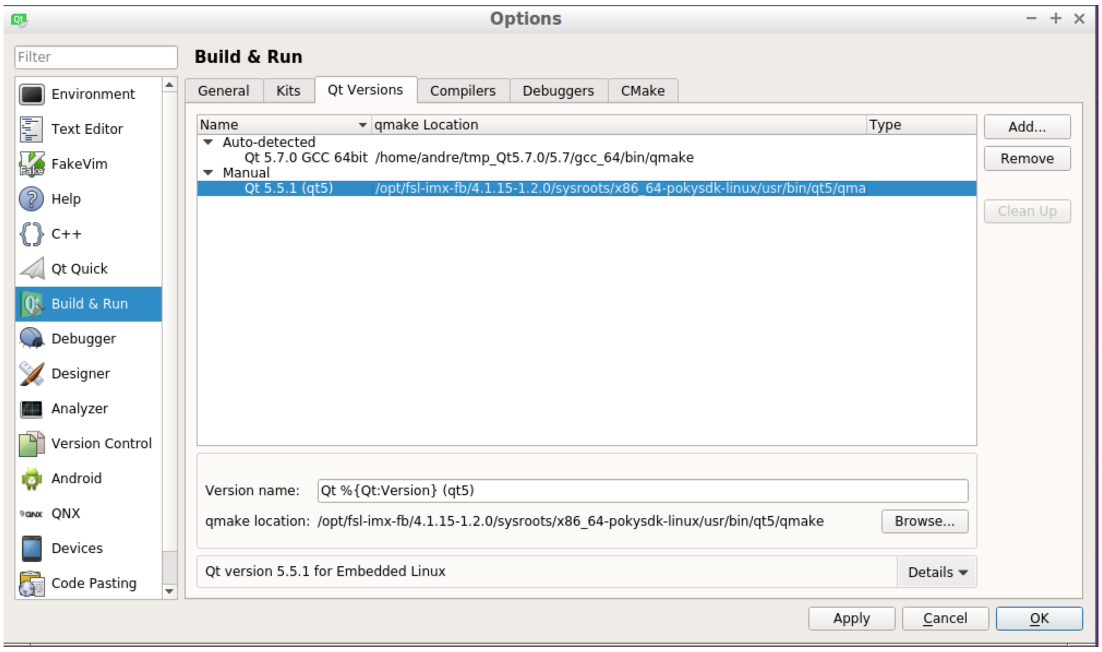
Location
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gd

Name: iMX GDB
Path: sroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
Type: GDB
ABIs: arm-linux-generic-elf-32bit
Version: 7.9.1

Figure 7 - Debuggers tab

Go to the "Qt Versions" tab as shown in Figure 8 and click the "Add" button. Select the path to qmake. In this example qmake is located at:

```
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-
linux/usr/bin/qt5/qmake
```



Name	qmake Location	Type
Auto-detected		
Qt 5.7.0 GCC 64bit	/home/andre/tmp_Qt5.7.0/5.7/gcc_64/bin/qmake	
Manual		
Qt 5.5.1 (qt5)	/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qma	

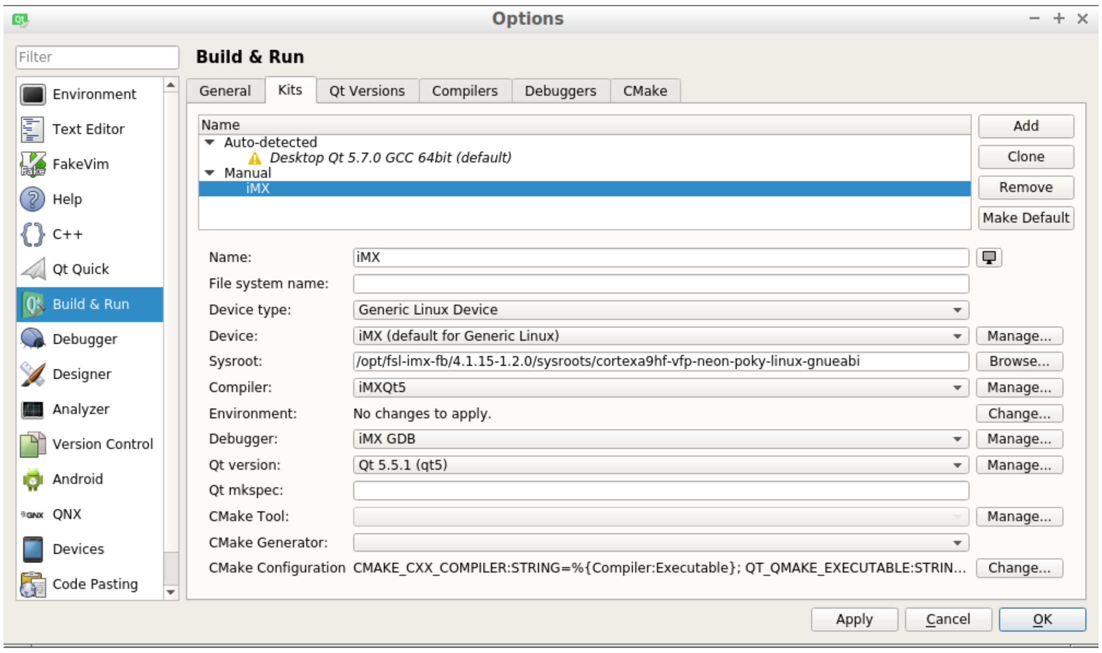
Version name: Qt %{Qt:Version} (qt5)
qmake location: /opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake
Qt version 5.5.1 for Embedded Linux

Figure 8 - Qt Versions tab

Go to the "Kits" tab and add a new kit by clicking the "Add" button. Choose a name for the kit and set "Device type", "Sysroot", "Compiler", "Debugger", and "Qt version". Figure 9 shows the settings used for our setup.

"Sysroot" is in this example set to:

```
/opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/cortexa9hf-vfp-neon-poky-
linux-gnueabi
```



Name
- Auto-detected
  - *Desktop Qt 5.7.0 GCC 64bit (default)*
- Manual
  - iMX

Name: iMX
File system name:
Device type: Generic Linux Device
Device: iMX (default for Generic Linux)
Sysroot: /opt/fsl-imx-fb/4.1.15-1.2.0/sysroots/cortexa9hf-vfp-neon-poky-linux-gnueabi
Compiler: iMXQt5
Environment: No changes to apply.
Debugger: iMX GDB
Qt version: Qt 5.5.1 (qt5)
Qt mkspec:
CMake Tool:
CMake Generator:
CMake Configuration CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable}; QT_QMAKE_EXECUTABLE:STRIN...

Figure 9 - Kits tab

5.3.1 Connection to target

It is possible to start/stop an application on the device using an SSH connection. To configure this connection you first have to retrieve some information from the target. The instructions below assume that you have a terminal application connected to the target.

Get IP address.

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr CA:71:64:BD:1A:20
          inet addr:192.168.1.222  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80:
```

Allow root login to SSH server:

By default SSH might not allow the user "root" to login. By following these instructions "root" will be permitted to login through an SSH connection. It is not recommended to use on a final application, but during development it can be permitted.

1. Open the configuration file for the SSH server

```
# nano /etc/ssh/sshd_config
```

2. Find the line that starts with #PermitRootLogin and remove the '#' (hash) character. If you cannot find this line just add it to the file (without the hash)

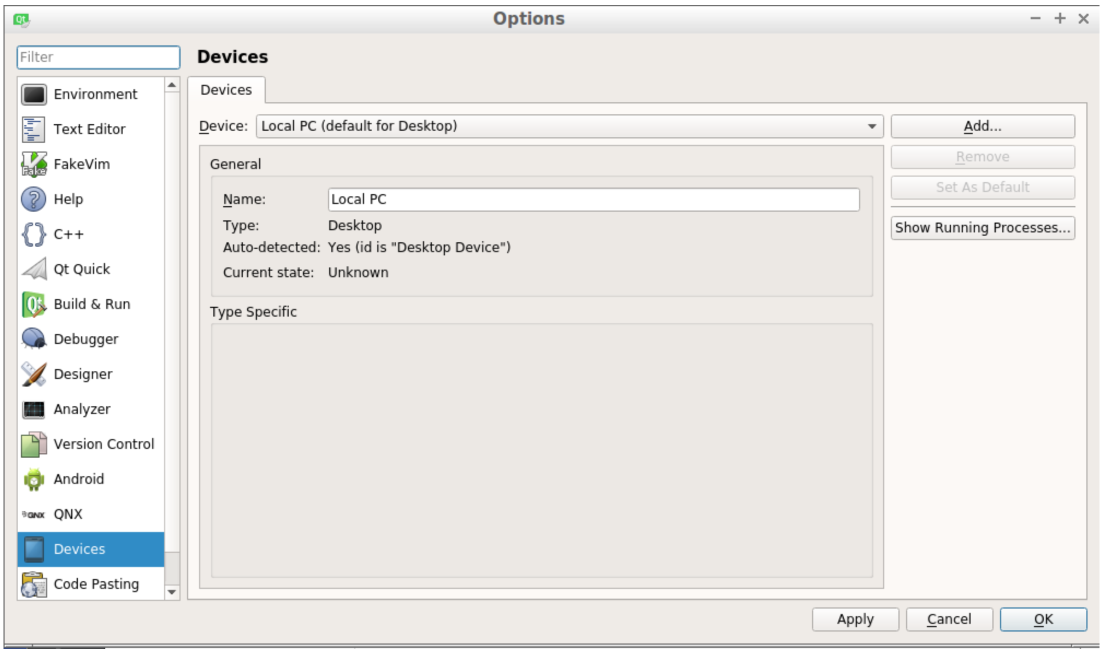
```
PermitRootLogin yes
```

3. Save the file and exit the editor (in `nano` it is Ctrl-X followed by Y and Enter).
4. Restart the SSH server

```
# /etc/init.d/sshd restart
```

Setup a Device in Qt Creator

Go back to Qt Creator, Tools → Options and then select "Devices" in the left menu as shown in Figure 10.

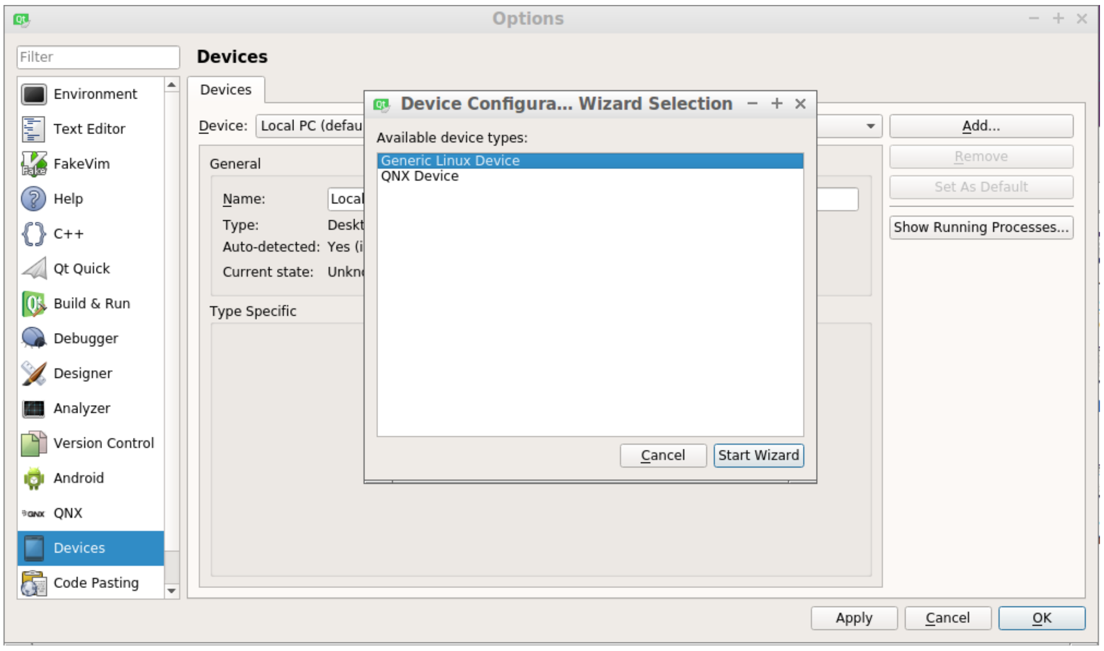


Device: Local PC (default for Desktop)

Name: Local PC
Type: Desktop
Auto-detected: Yes (id is "Desktop Device")
Current state: Unknown

Figure 10 - Devices

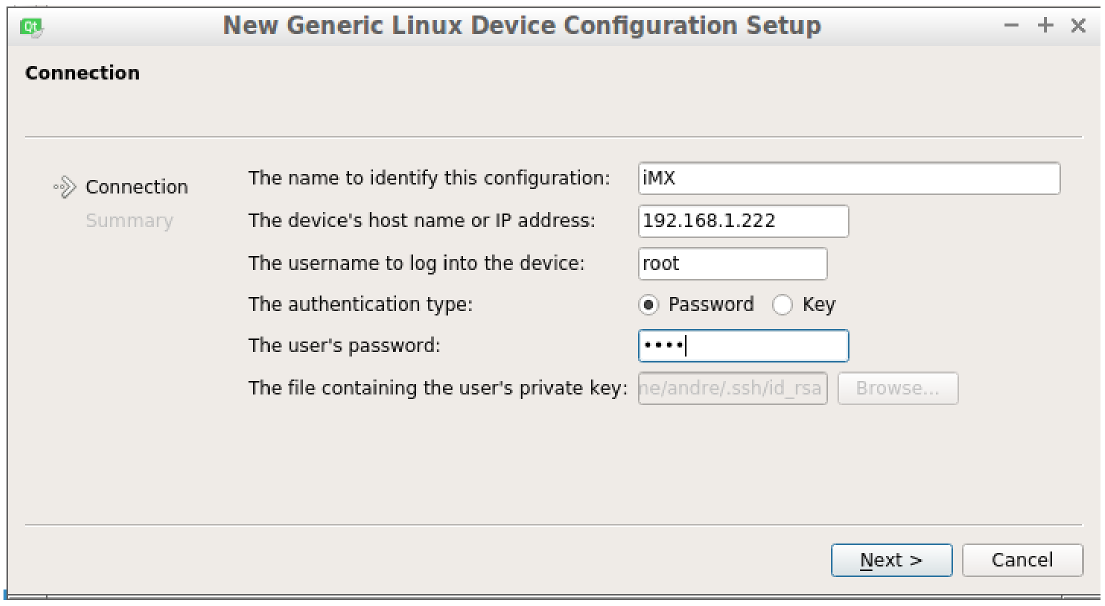
Click the "Add" button and select "Generic Linux Device as shown in Figure 11.



Available device types:
Generic Linux Device
QNX Device

Figure 11 - Select device type

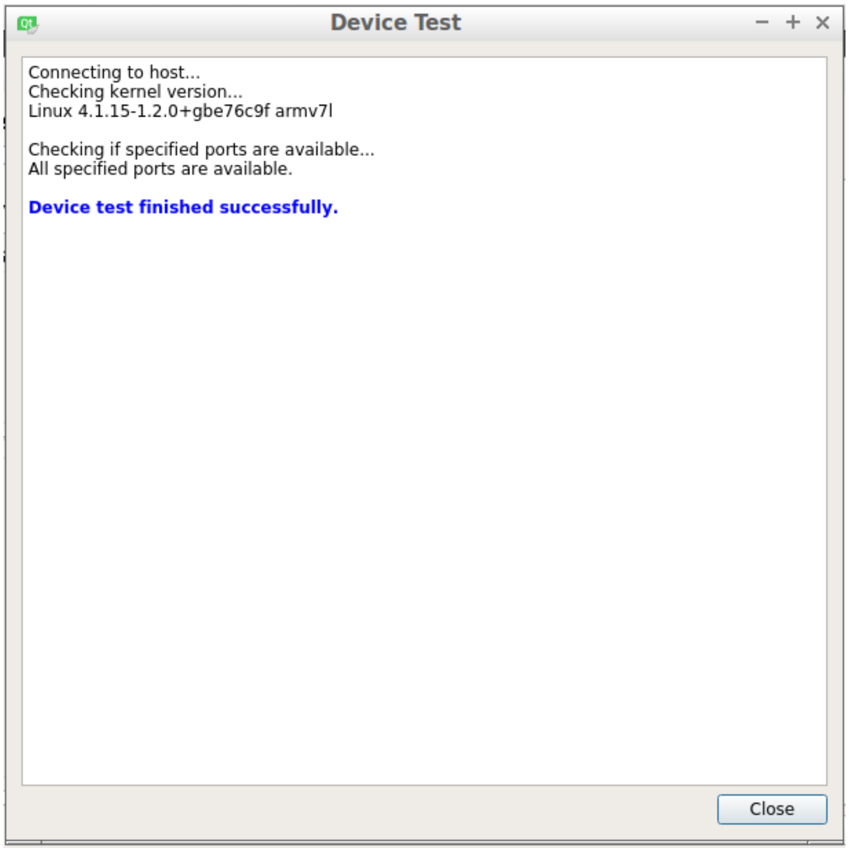
Give the device a name, enter the IP address, username and password as shown in Figure 12. By default the password for the user `root` is **`pass`**.



The name to identify this configuration: iMX
The device's host name or IP address: 192.168.1.222
The username to log into the device: root
The authentication type: [✓ Password] Key
The user's password: ••••
The file containing the user's private key: ne/andre/.ssh/id_rsa

Figure 12 - IP address and user credentials

Click "Next" and then "Finish" and the connection to the target will be created and tested as shown in Figure 13.



```
Connecting to host...
Checking kernel version...
Linux 4.1.15-1.2.0+gbe76c9f armv7l

Checking if specified ports are available...
All specified ports are available.

Device test finished successfully.
```

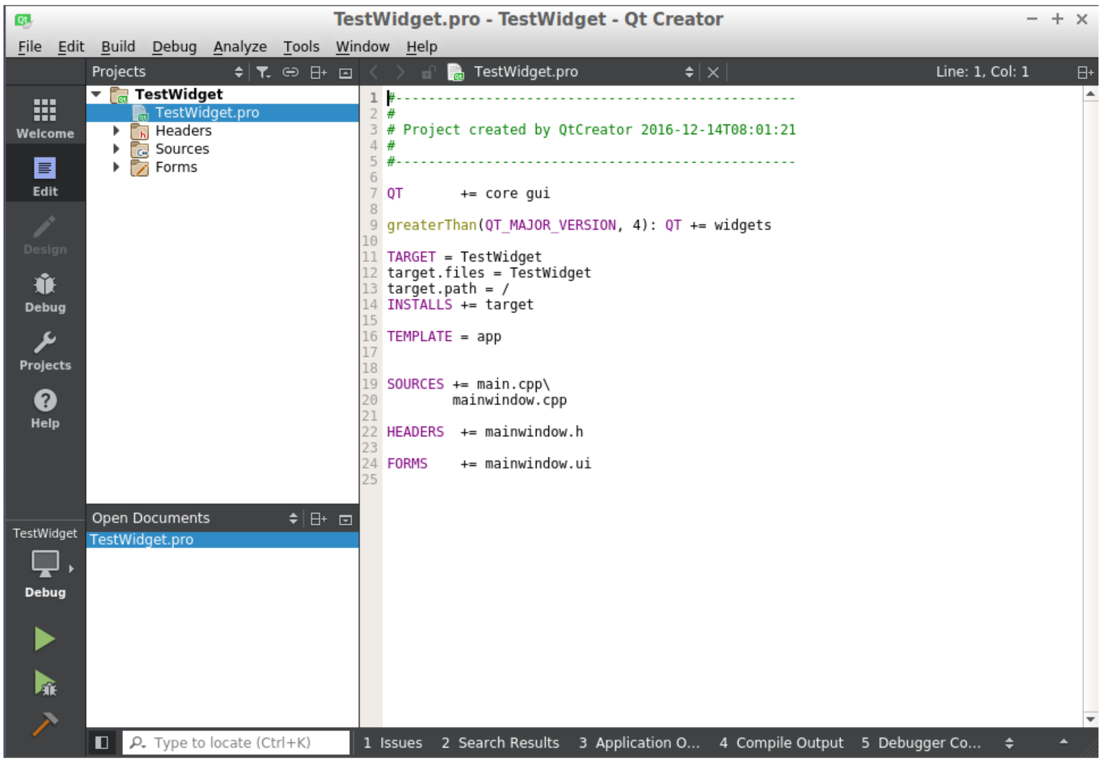
Figure 13 - Testing connection to target

5.4 Run on target

When you have created an application it is possible to configure the project so that the application is downloaded and executed on the target. In the example below an application named TestWidget has been created.

First you need to prepare the ".pro" file to list target files and define installation path for the files.

1. Click on the "Edit" button and then the .pro file as shown in Figure 14.



```
#-------------------------------------------------
#
# Project created by QtCreator 2016-12-14T08:01:21
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = TestWidget
target.files = TestWidget
target.path = /
INSTALLS += target

TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui

```

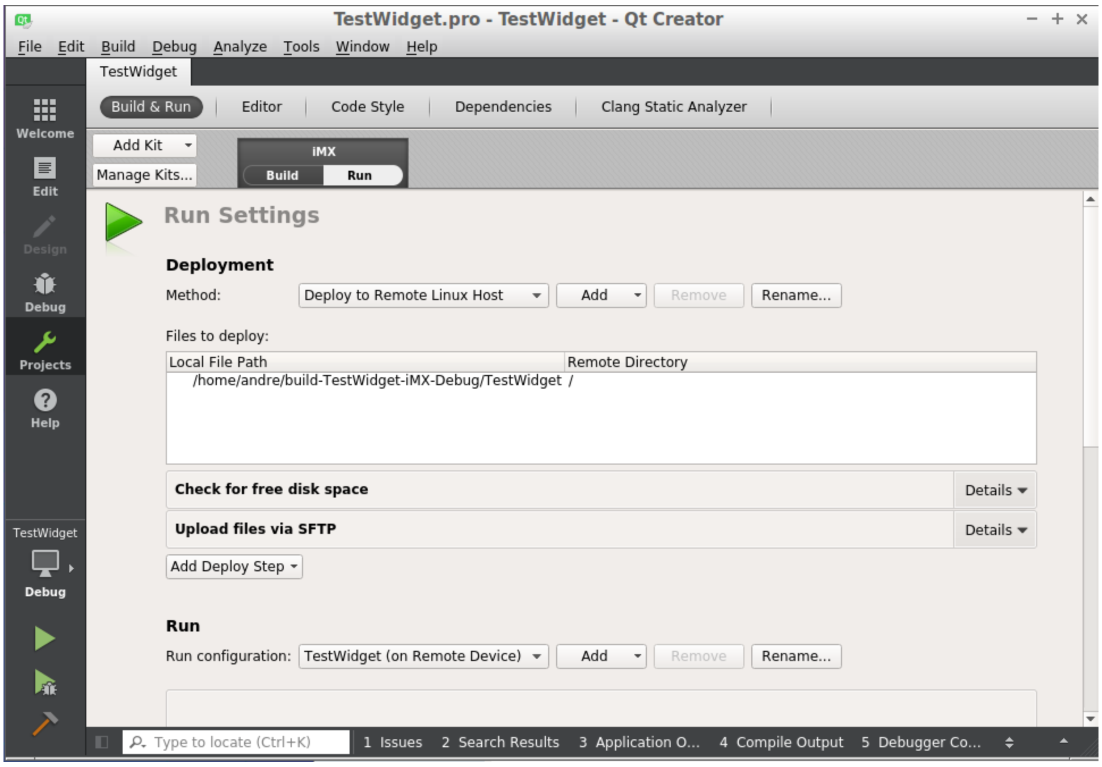
Figure 14 - Editing .pro file

2. Add the information listed below in the .pro file. Change "TestWidget" to the name of your application. You can also change the path if you want the files to be installed somewhere else on the target.

```
target.files = TestWidget
target.path = /
INSTALLS += target
```

3. Save the file

Now click on the "Projects" button and then on the "Run" configuration as shown in Figure 15. In the "Files to deploy" area you should see the path to the application as well as the remote directory. Without the changes in the .pro file the application path and remote directory wouldn't be listed.



Run Settings

Deployment

Method: Deploy to Remote Linux Host

Files to deploy:

Local File Path	Remote Directory
/home/andre/build-TestWidget-iMX-Debug/TestWidget	/

Check for free disk space

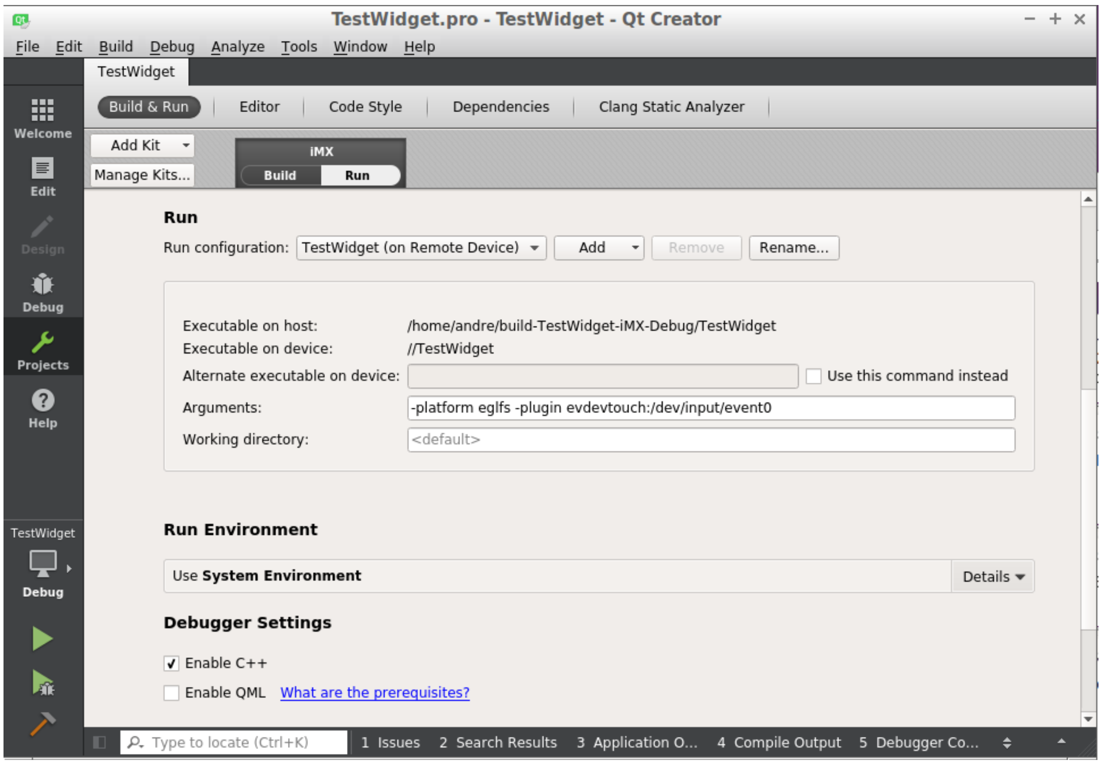
Upload files via SFTP

Run

Run configuration: TestWidget (on Remote Device)

Figure 15 - Run configuration for the project

To be able to start the Qt application on the target some arguments must be given. The "platform" argument specifies which platform plugin to use. For Embedded Artists COM boards with a GPU and with a framebuffer as backend you choose `eglfs`. For the COM boards without a GPU choose `linuxfb` as platform. The "plugin" argument specifies the path to the touch device that is used (if you have a touch screen).



Run

Run configuration: TestWidget (on Remote Device)

Executable on host: /home/andre/build-TestWidget-iMX-Debug/TestWidget
Executable on device: //TestWidget
Alternate executable on device: ☐ Use this command instead
Arguments: -platform eglfs -plugin evdevtouch:/dev/input/event0
Working directory: <default>

Run Environment

Use **System Environment**

Debugger Settings

☑ Enable C++
☐ Enable QML

Figure 16 - Application arguments

It is now be possible to run an application on the target by clicking on the "Play" button. It is also possible to download and debug an application by clicking on the "Debug" button.

For further information about Qt application development please visit. doc.qt.io/.