

F&S Introduction to QT

Debugging an Application

Version 1.11
(17.10.2023)



**Elektronik
Systeme**

© F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Germany

Phone: +49(0)711-123722-0
Fax: +49(0)711-123722-99

About This Document

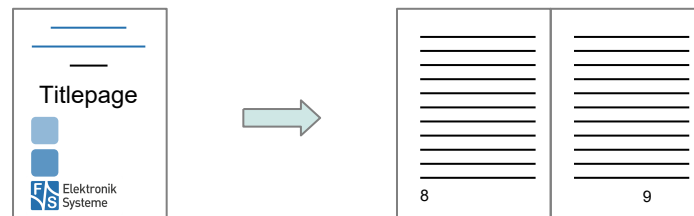
This document describes how to debugging remote device using QT Creator Automatic Remote Debugging Launcher under Linux. The software is configured for architectures fsimx6/fsimx6sx/fsimx6ul/fsimx7ulp/fsimx8mm from F&S under Linux/Buildroot.

Remark

The version number on the title page of this document is the version of the document. It is not related to the version number of any software release. The latest version of this document can always be found at <http://www.fs-net.de>.

How to Print This Document

This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.



Typographical Conventions

We use different fonts and highlighting to emphasize the context of special terms:

File names

Menu entries

```
Board input/output
```

Program code

```
PC input/output
```

Listings

```
Generic input/output
```

Variables

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2018-04-27	1.0	*	A	ALL	Create Documentation	PJ
2018-06-08	1.1	*	M	2.1, 5.2	Improve setup	PJ
2018-06-14	1.2	*	M	3	Update figure 12, it is necessary to use cmake instead of qmake	PJ
2020-05-04	1.3	*	M M M	2 3 All	Improve some sentences Update figures for new QT Creator, improve some sentences Improve layout	PJ
2021-03-11	1.4	*	M	3, 5.2 5.3	Add GDB path for 64 bit CPUs Add target path for Yocto Add backend support for debugging	PJ
2022-07-19	1.5	*	M	2, 3, 5.1,5.2	Add Yocto Paths Updated some sentences	TG
2022-07-26	1.6	*	M	3	Changed Toolchain distro	TG
2022-09-19	1.7	*	A,M	1.2, 3,	Add Paths for i.MX6	TG
2022-11-09	1.8	*	M	1.1	Added installation for QT Creator 4.15.2 for Fedora 30	TG
2023-01-10	1.9	*	M	1, 2.2, 3	Updated figures for new QT Creator Added PicoCORE in Introduction	TG
2023-06-09	1.10	*	M	1.1,2.1,3	Add information about environment-setup Add information about sftp Fix missing "/" in command	PG
2023-09-28	1.11	*	A,M	1, 1.1, 1.2, 3, 4	Add general support for QT6 Add colored separation of displayed steps for QT5 and QT6 + Buildroot and Yocto separation Add paths for QT6	TG

V Version
A,M,R Added, Modified, Removed
Au Author





Table of Contents

1	Introduction	1
1.1	Install Qt Creator on Fedora	2
1.2	Tested Boards and Releases	4
2	Remote Connection	5
2.1	Setup SSH connection on SBC/SOM.....	5
2.2	Setup SSH connection in Qt Creator.....	7
3	Setup Application settings	10
4	Create an Application	17
4.1	Create hello world app.....	26
5	Build and Debug Application	29
5.1	Desktop Run.....	30
5.2	FS Cross Compile Kit for ARM Run	32
5.3	Debug build	33
6	Appendix	36
	List of Figures	36
	Important Notice	37





1 Introduction

F&S offers a whole variety of Systems on Module (SOM) and Single Board Computers (SBC). There are different board families that are named NetDCU, PicoMOD, PicoCOM, PicoCore, armStone, QBliss, and efus.

Linux is available for all of these platforms. Also QT is available for all of these platforms. Linux offers a tool called Qt Creator which is a cross-platform C++, JavaScript and QML integrated development environment which is part of the SDK for the Qt GUI application development framework. It includes a visual debugger and an integrated GUI layout and forms designer.

This document describes how to debug an application on a remote device using the automatic remote debugging launcher plug-in.

This document describes both the steps for Qt5 and Qt6.

The Qt5 steps will be displayed like the following:

```
...example of paths or code.
```

And the Qt6 steps like this:

```
...example of paths or code.
```

If the steps are not displayed separately, it means that the steps work for both Qt5 and Qt6.

1.1 Install Qt Creator on Fedora

Install Qt Creator on your development host with your Linux Distribution specific package manager system. In our development machine, which you can download from our home-page, QT Creator is already installed. On Fedora up to version 27:

```
sudo yum install qt-creator
```

On Fedora 30 we will need to install a new version of QT Creator:

```
wget https://download.qt.io/official_releases/qtcreator/4.15/4.15.2/qt-creator-opensource-linux-x86_64-4.15.2.run
```

Next we will need to give the file execute permissions:

```
chmod +x qt-creator-opensource-linux-x86_64-4.15.2.run
```

Now execute the file and follow the Installers instructions:

```
./qt-creator-opensource-linux-x86_64-4.15.2.run
```

Finally you want to create a link to the new QT Creator. Go into the directory that has been created for the new QT Creator and execute the following command:

```
sudo ln -sf /bin/qtcreator /usr/bin/qtcreator
```

Then start Qt Creator:

```
qtcreator &
```

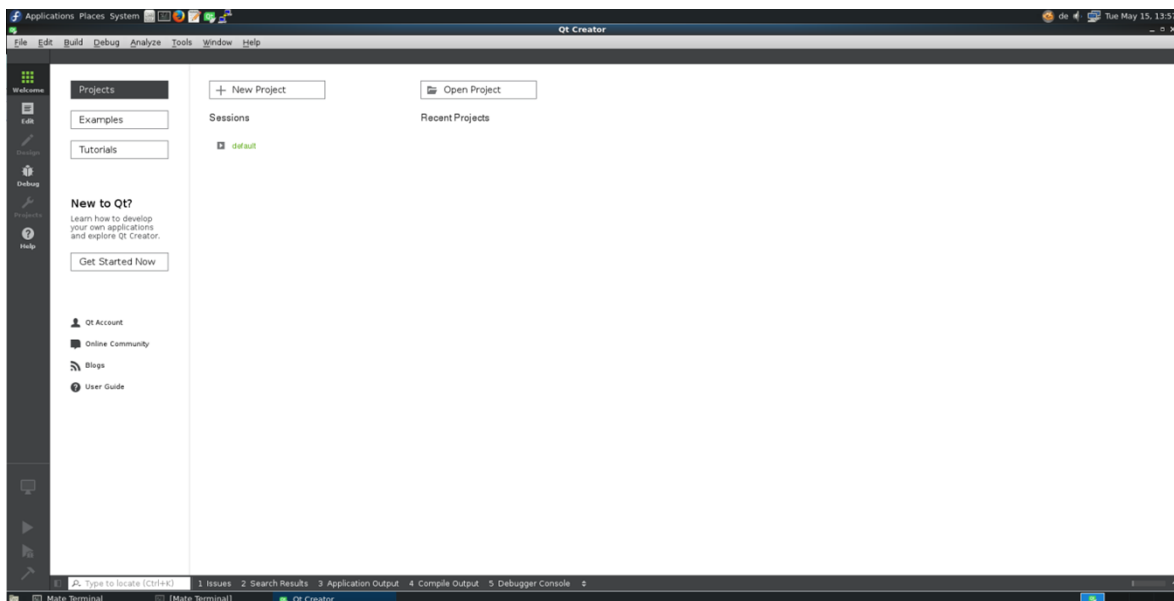


Figure 1: Qt Creator

Note:

If Yocto toolchain is used, qtcreator must be started from the same terminal where SDK environment are set by command:

```
source <toolchain directory>/environment-setup-aarch64-poky-linux
```

See **3 Setup Application settings** on how to build and install the QT5 SDK

Alternatively, create a shell script to start both commands sequentially.

E.g.: qt5-yocto.sh

```
#!/usr/bin/sh
source <toolchain directory>/environment-setup-aarch64-poky-linux
qtcreator &
```

1.2 Tested Boards and Releases

Board:	Revision:	Release:
PicoCoreMX8MM	1.20	Buildroot (fsimx8mm-B2021.06)
PicoCoreMX8MM	1.20	Yocto (fsimx8mm-Y2021.04)
ArmStoneA9	1.10	Yocto (fsimx6-Y2020.03)
ArmStoneA9	1.10	Buildroot (fsimx6-B2021.10.1)
PicoCoreMX8MP2	1.10	Yocto (fsimx8mp-Y2023.09)



2 Remote Connection

First of all the SSH connection on the remote system will be setup.

2.1 Setup SSH connection on SBC/SOM

You have to setup your F&S Board. Therefore you can have a look into F&S *LinuxOnFS-Boards_eng.pdf*. After that boot your F&S Board.

To work with SSH the board should have a valid date. This is necessary to create certificates for SSH. To setup a date you can use the following command:

```
date "2020-05-04 10:13"
```

Afterwards we have to enable the network interface. You can also set the network on command in UBoot to enable network interface at each boot. For further information please take a look in *FSiMX6_FirstSteps_eng.pdf*.

Dynamically:

```
udhcpc -i eth0
```

Static (e.g. ip address is 10.0.0.84):

```
ifconfig eth0 10.0.0.84 up
```

The Root-Filesystem is read-only mounted, but we have to modify something in the filesystem so we need it read-writeable.

```
mount -o remount,rw /
```

Open *sshd_config* file

```
vi /etc/ssh/sshd_config
```

and edit the following lines:

```
...
PermitRootLogin yes
...
```

Optional you can also allow to login without password, but we recommend you to not do this because it's a security risk. If you want to do it anyway add the following line to *sshd_config* file.

```
...
PermitEmptyPasswords yes
...
```

Remote Connection

Make sure to use the internal sftp if no external binary is installed.

```
...  
Subsystem sftp internal-sftp  
...
```

After that you have to start the SSH server (This step is not required when using Yocto).

```
/etc/init.d/S50sshd start
```

Now the SSH server is running on our SBC/SOM. Let's test it, therefore we are going back to our VM. To connect via ssh we open a Terminal and send the following command:

```
ssh root@10.0.0.84
```

```
[jakob@localhost ~]$ ssh root@10.0.0.84  
The authenticity of host '10.0.0.84 (10.0.0.84)' can't be established.  
ECDSA key fingerprint is b1:b1:aa:83:12:d1:f1:21:7b:e3:6a:61:89:6e:31:ea.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '10.0.0.84' (ECDSA) to the list of known hosts.
```

Figure 2: SSH connection terminal

Now the SSH connection is successfully established and tested.

2.2 Setup SSH connection in Qt Creator

Open *Tools*. In the toolbar and select *Options....*

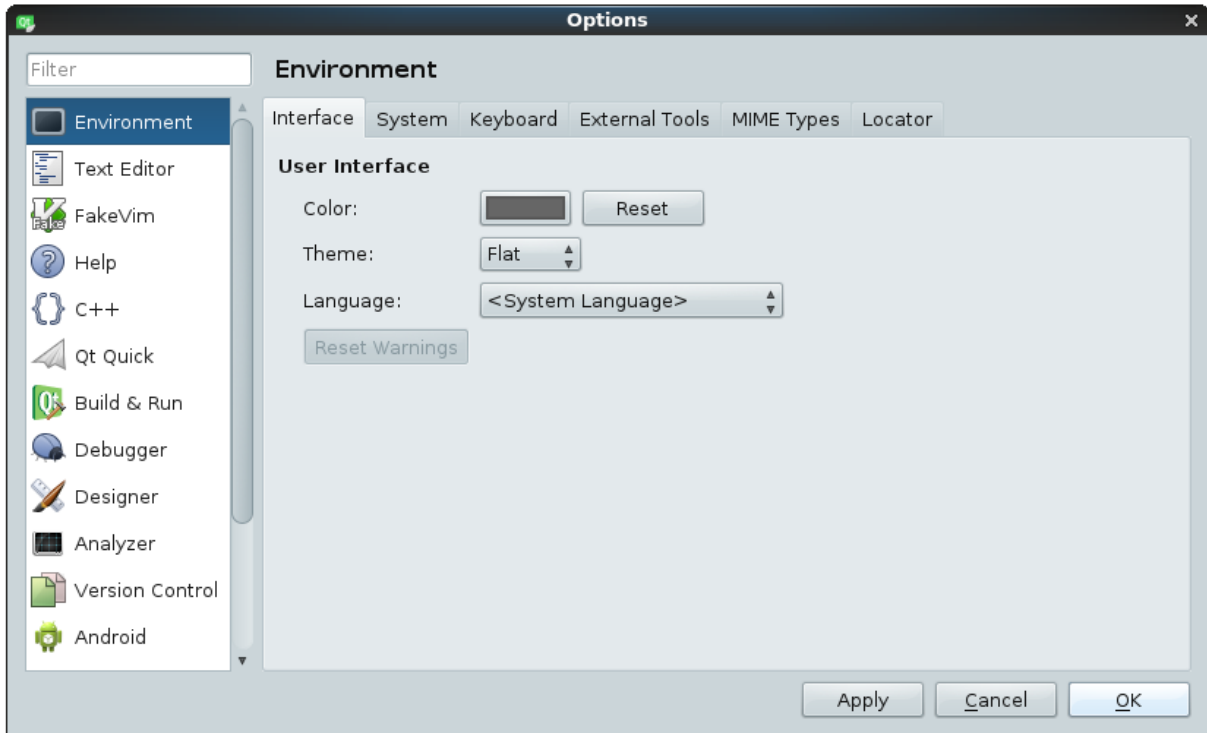
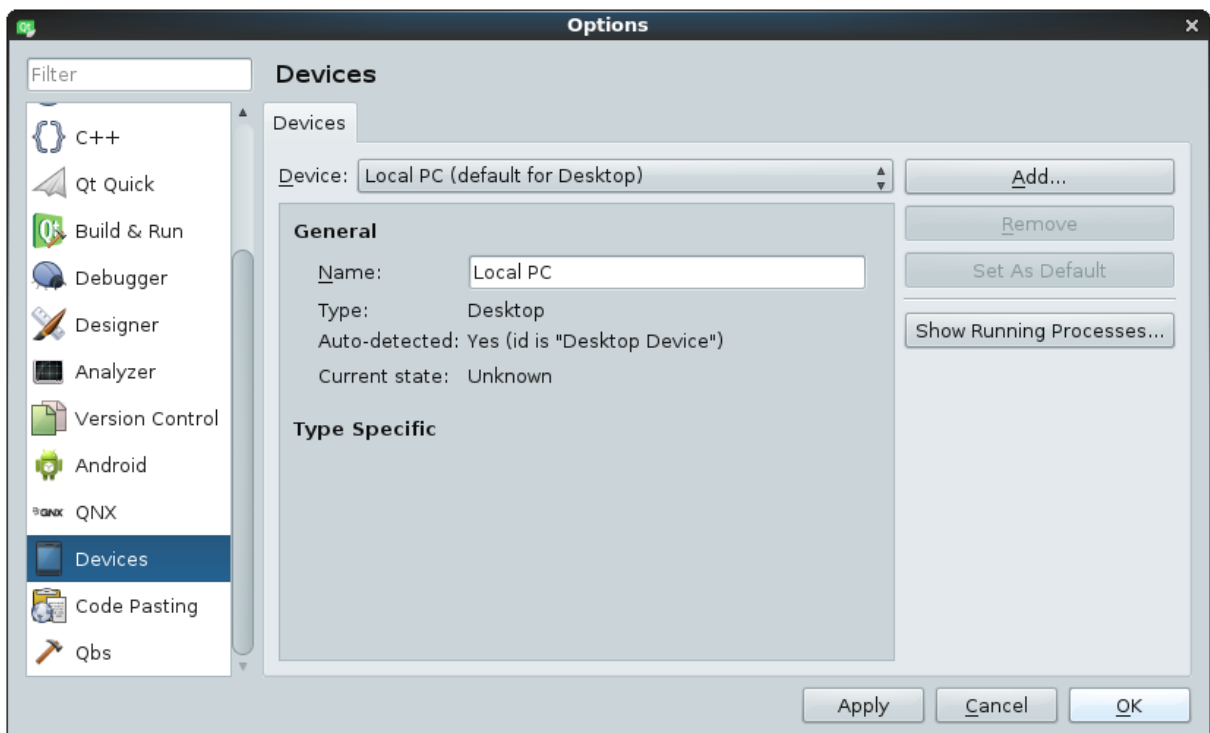


Figure 3: Options

Figure 4: Options - Devices

Choose *Devices* on the left side.



Remote Connection

Now add a new device. Press *Add...*, choose *Generic Linux Device* and click *Start Wizard*.

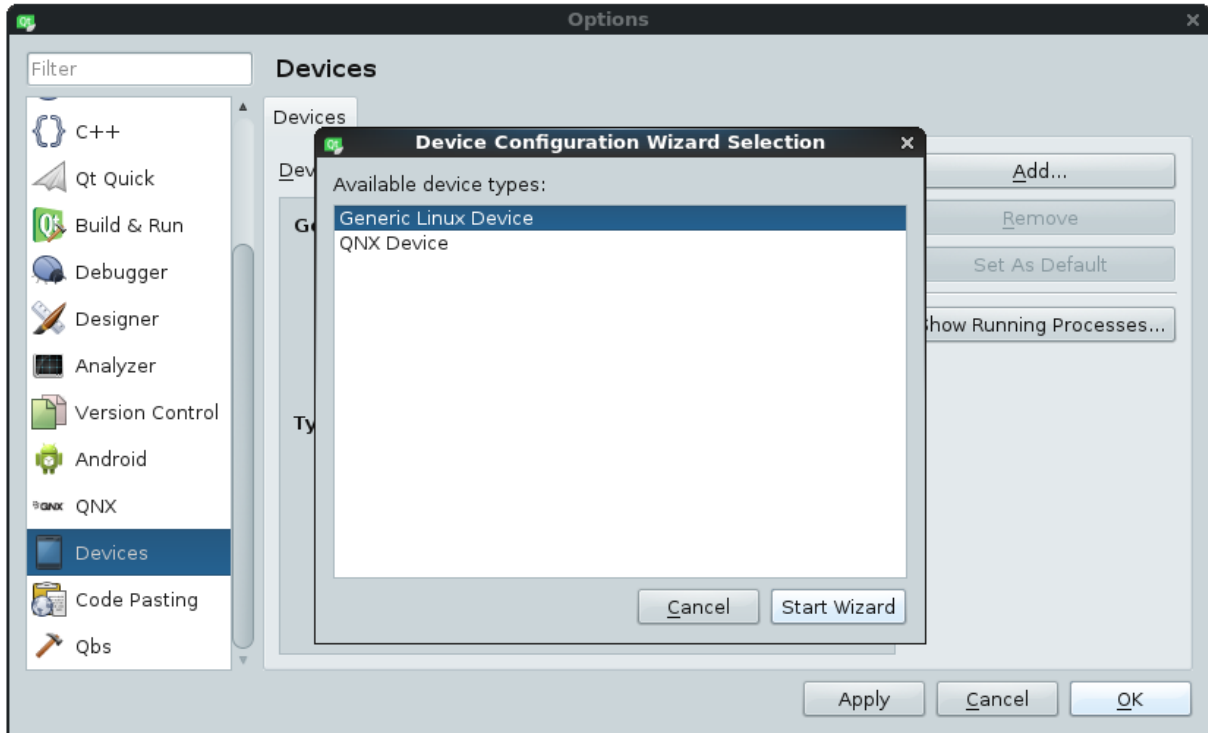


Figure 5: Add device configuration

Now set a name for the configuration, enter the IP address and the login data.

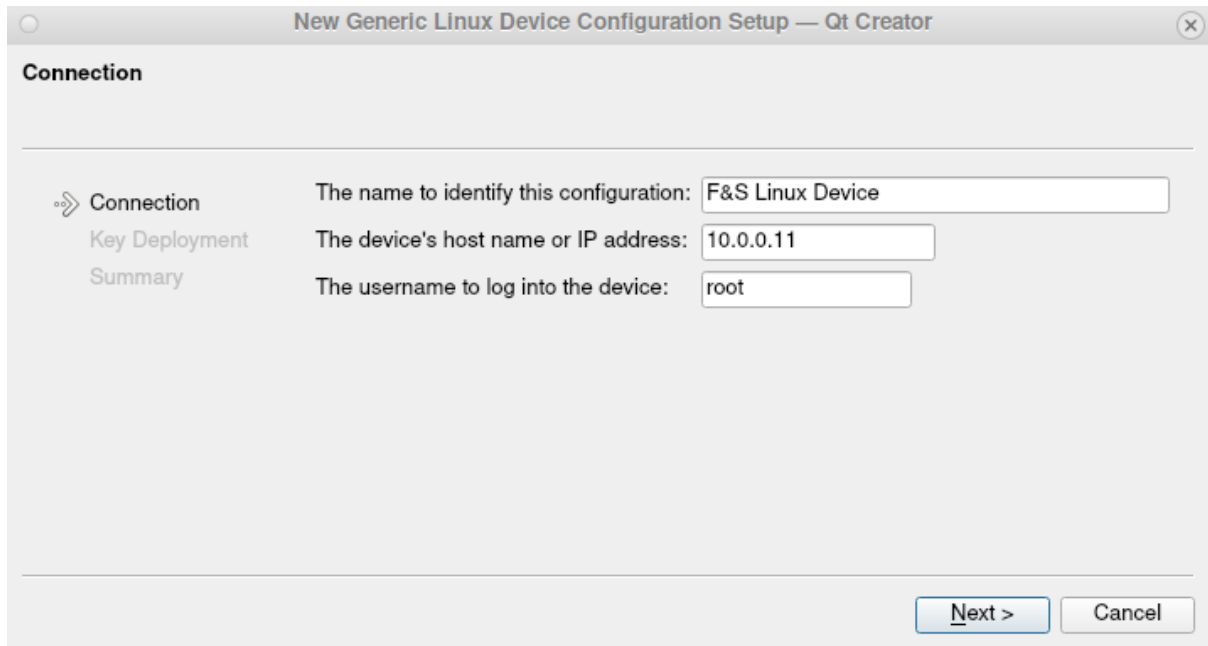


Figure 6: Setup device connection

Afterwards press *Next* > *Deploy Public Key* > *Next* and *Finish*. Then the ssh connection will be automatically tested.

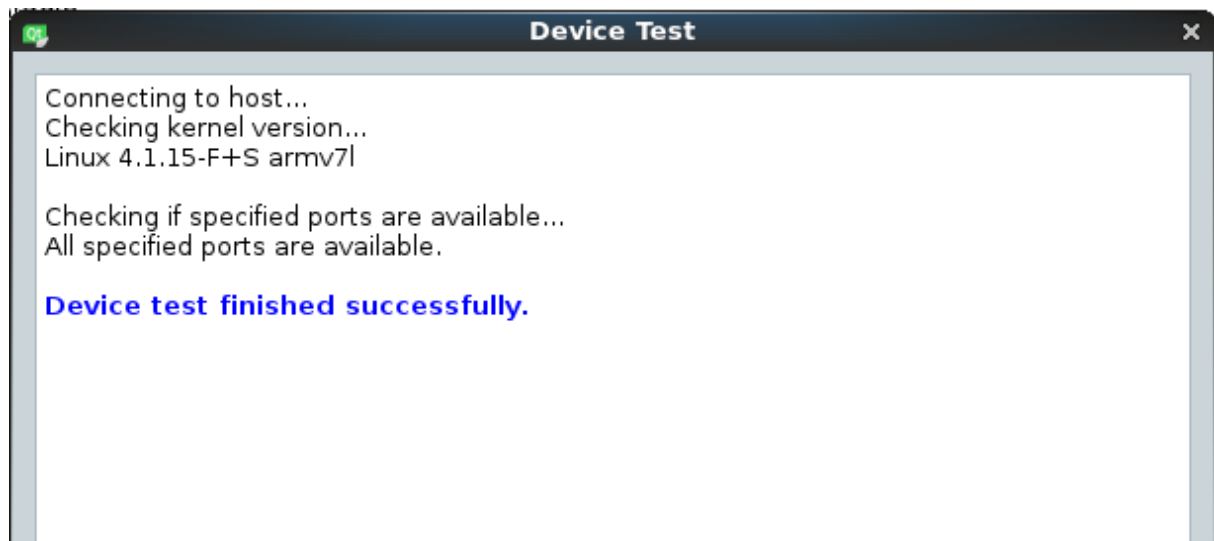


Figure 7: Device test

3 Setup Application settings

Before you can setup the application settings be sure that you have built your Root-Filesystem (**Buildroot/Yocto**) with the correct configuration (`fsimx6/ul/sx_qt5_defconfig`) once. This is necessary because we need the compiled GDB debugger and other packages from this directory. If you are using Yocto you will need to install the meta-toolchain as well. Building the toolchain will take at least 12 GB of RAM.

Note:

If the release version which you are using is less then `fsimx6-V3.1`, `fsimx6sx-V2.1` or `fsimx6ul-V2.1`, then you have to add the following packages to your root filesystem and rebuild it, e.g. through “*make menuconfig*”:

- `ctest`, located in Target packages/Development Tools/ctest
- `Python support`, located in Toolchain/Build Cross gdb for the host/Python support

If you have already built your root filesystem, then you have to run the following command in your terminal:

```
...buildroot-2019.05.3-fsimx6-B2020.04]$ make host-gdb-reconfigure
...buildroot-2019.05.3-fsimx6-B2020.04]$ make
```

Note:

To install the meta-toolchain head over to your build directory and run the following command:

QT5:

```
...<build-directory>]$ source ../../sources/poky/oe-init-build-env
...<build-directory>]$ bitbake meta-toolchain-qt5
```

QT6:

```
...<build-directory>]$ source ../../sources/poky/oe-init-build-env
...<build-directory>]$ bitbake meta-toolchain-qt6
```

Next you need to head to `...<Build-Dir>/tmp/deploy/sdk` and run the `.sh` script that now exist. After that you can find your toolchain in your desired directory.

After that you have to install the new root filesystem on your remote target system. Now select *Tools* and *Options...* in your QT Creator. Select Kits on the left side bar.



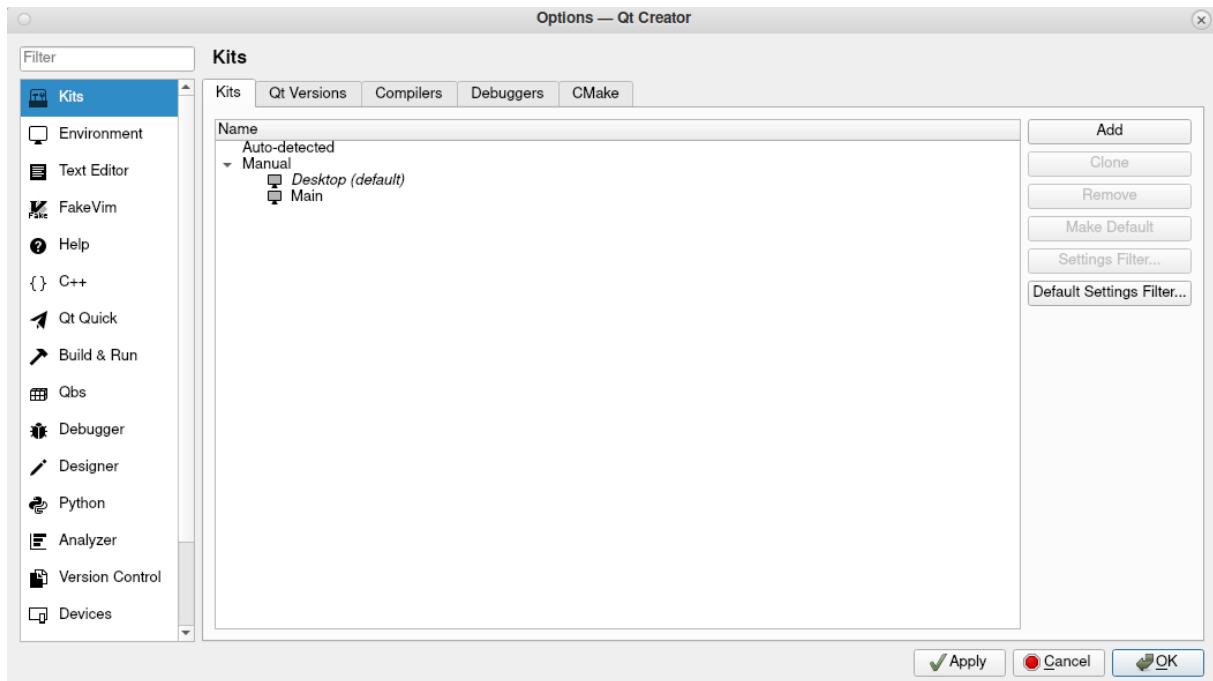


Figure 8: Options - Kits

Setup Application settings

Choose *QT Versions* and select *Add...*, setup the path to the qmake location.

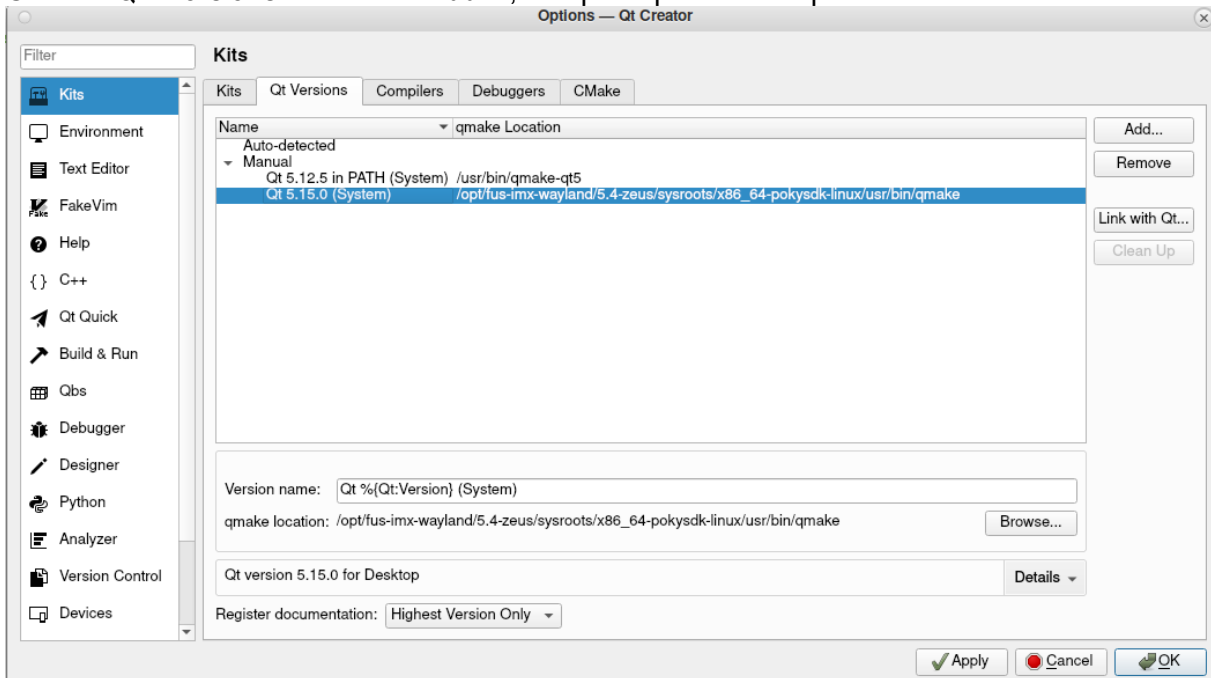


Figure 9: Kits - Qt Versions

Examples:

QT5:

(Buildroot)

```
/home/developer/fsimx6-B2020.04/build/buildroot-2019.05.3-fsimx6-B2020.04/output/host/usr/bin/qmake
```

(Yocto/i.MX8)

```
/opt/fus-imx-wayland/5.4-zeus/sysroots/x86_64-pokysdk-linux/usr/bin/qmake
```

(Yocto/i.MX6)

```
/opt/poky/2.4.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake
```

QT6:

(Yocto/i.MX8)

```
/opt/fus-imx-wayland/5.15-kirkstone/sysroots/x86_64-pokysdk-linux/usr/bin/qmake6
```

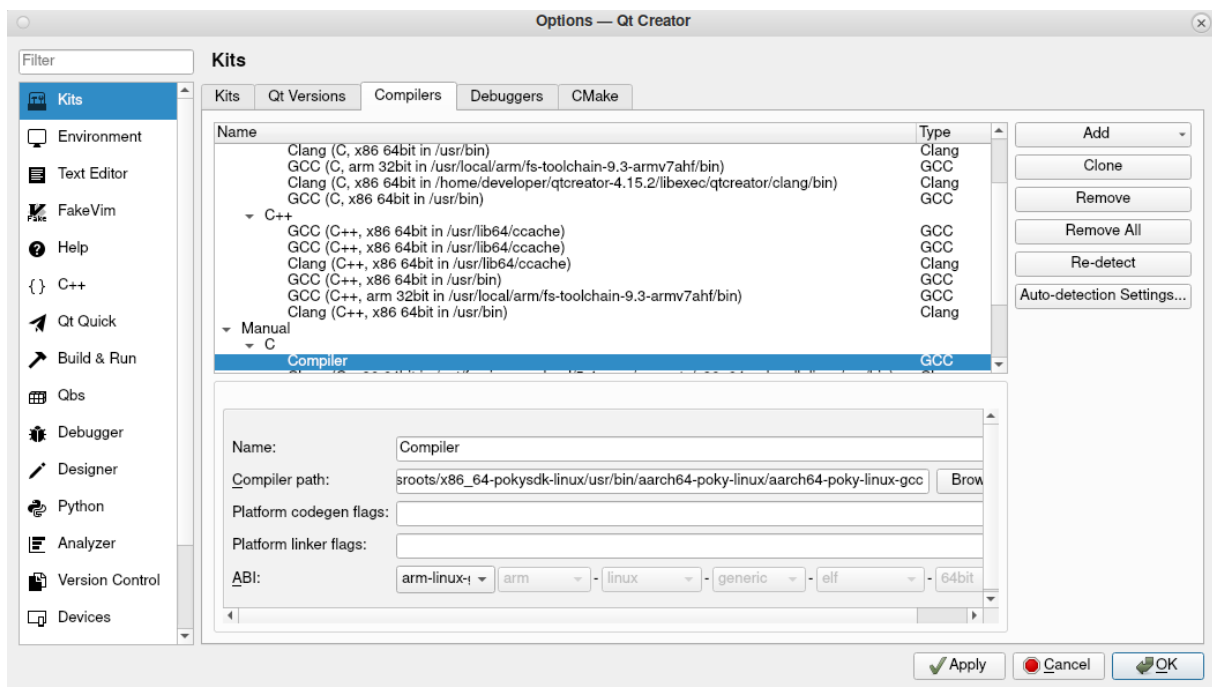


Figure 10:Kits - Compilers

Now select *Compilers* and Add *GCC then C++*. Setup the *Name* and the *Compiler path*. After that *Apply* the settings.

Examples:

QT5:

(Buildroot)

```
/home/developer/fsimx6-B2020.04/build/buildroot-2019.05.3-fsimx6-B2020.04/output/host/usr/bin/arm-linux-g++
```

(Yocto/i.MX8)

```
/opt/fus-imx-wayland/5.4-zeus/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-g++
```

(Yocto/i.MX6)

```
/opt/poky/2.4.2/sysroots/x86_64-pokysdk-linux/usr/bin/i586-poky-linux/i586-poky-linux-g++
```

QT6:

(Yocto/i.MX8)

```
/opt/fus-imx-wayland/5.15-kirkstone/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-g++
```



Setup Application settings

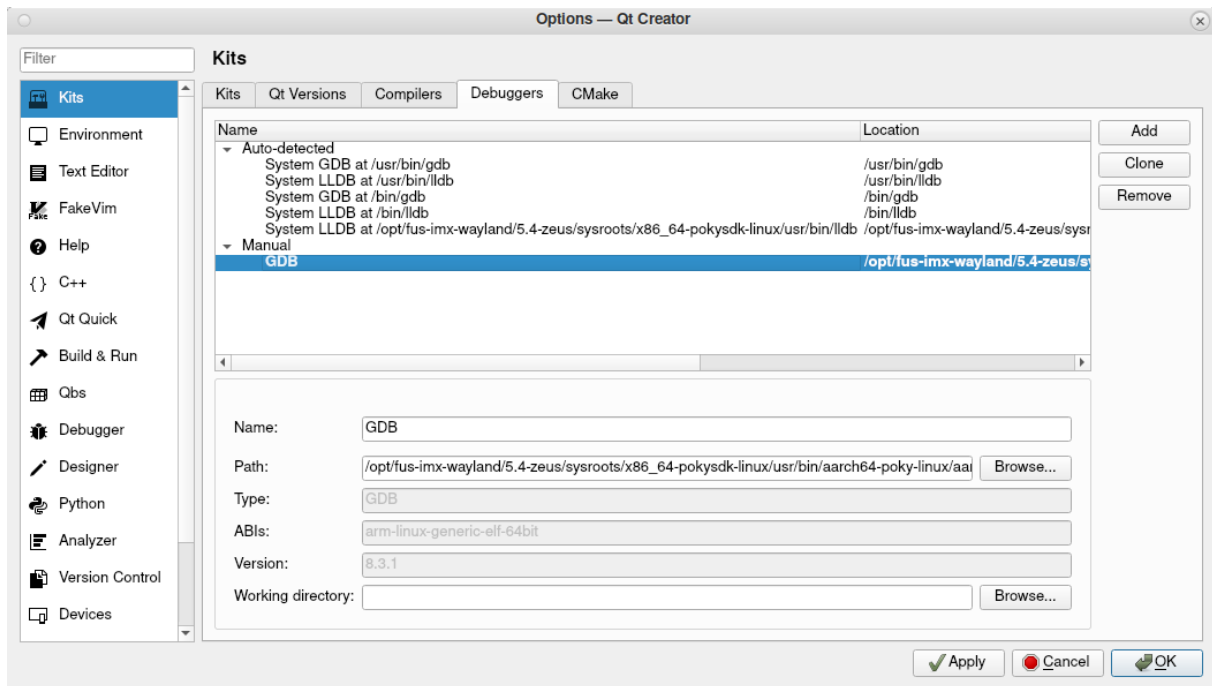


Figure 11: Kits - Debuggers

Afterwards select *Debuggers* and *Add*. Setup *Name* and *Path* and *apply* the settings.

Examples:

QT5:

(Buildroot)

32-bit CPUs:

```
/home/developer/fsimx6-B2020.04/build/buildroot-2019.05.3-fsimx6-B2020.04/output/host/usr/bin/arm-linux-gdb
```

64-bit CPUs:

```
/home/developer/fsimx8mm-B2020.08/build/buildroot-2019.05.3-fsimx8mm-B2020.08/output/host//bin/aarch64-linux-gdb
```

(Yocto)

32-bit CPUs:

```
/opt/poky/2.4.2/sysroots/x86_64-pokysdk-linux/usr/bin/i586-poky-linux/i586-poky-linux-gdb
```

64-bit CPUs:

```
/opt/fus-imx-wayland/5.4-zeus/sysroots/x86_64-pokysdk-linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb
```

QT6:**(Yocto)**

```
/opt/fus-imx-wayland/5.15-kirkstone/sysroots/x86_64-pokysdk-
linux/usr/bin/aarch64-poky-linux/aarch64-poky-linux-gdb
```

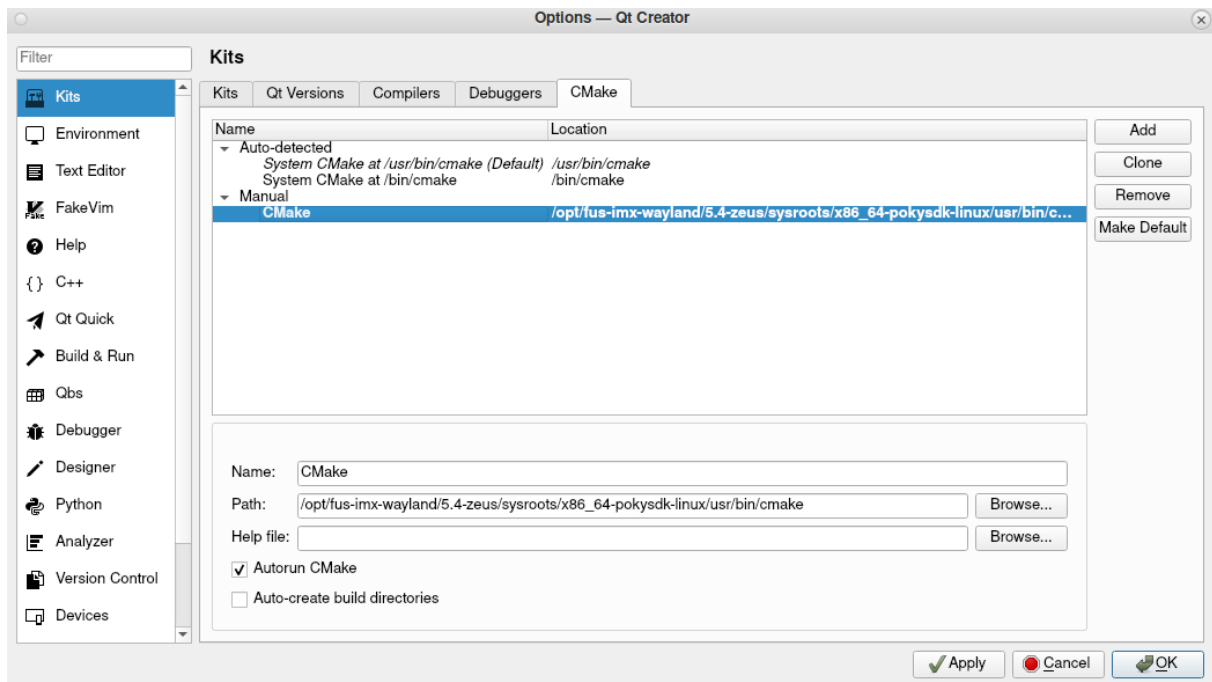


Figure 12: Kits – Cmake

After that select *CMake* and *Add*. Setup *Name* and *Path* and *Apply* the settings.

Examples:

QT5:**(Buildroot)**

```
/home/developer/fsimx6-B2020.04/build/buildroot-2019.05.3-fsimx6-
B2020.04/output/host/usr/bin/cmake
```

(Yocto/.MX8)

```
/opt/fus-imx-wayland/5.4-zeus/sysroots/x86_64-pokysdk-
linux/usr/bin/cmake
```

(Yocto/i.MX6)

Setup Application settings

```
/opt/poky/2.4.2/sysroots/x86_64-pokysdk-linux/usr/bin/cmake
```

QT6:

(Yocto/i.MX8)

```
/opt/fus-imx-wayland/5.15-kirkstone/sysroots/x86_64-pokysdk-linux/usr/bin/cmake
```

Now everything is setup and the Kit can be created. Select *Kits* and *Add*. Setup the Kit (refer to figure 13).

Note:

If you are unable to find the file *cmake* in buildroot directory `output/host/usr/bin`, then you have to add the following package to your buildroot configuration and rebuild it, e.g. through “*make menuconfig*”:

- *host cmake*, located in Host utilities/host cmake

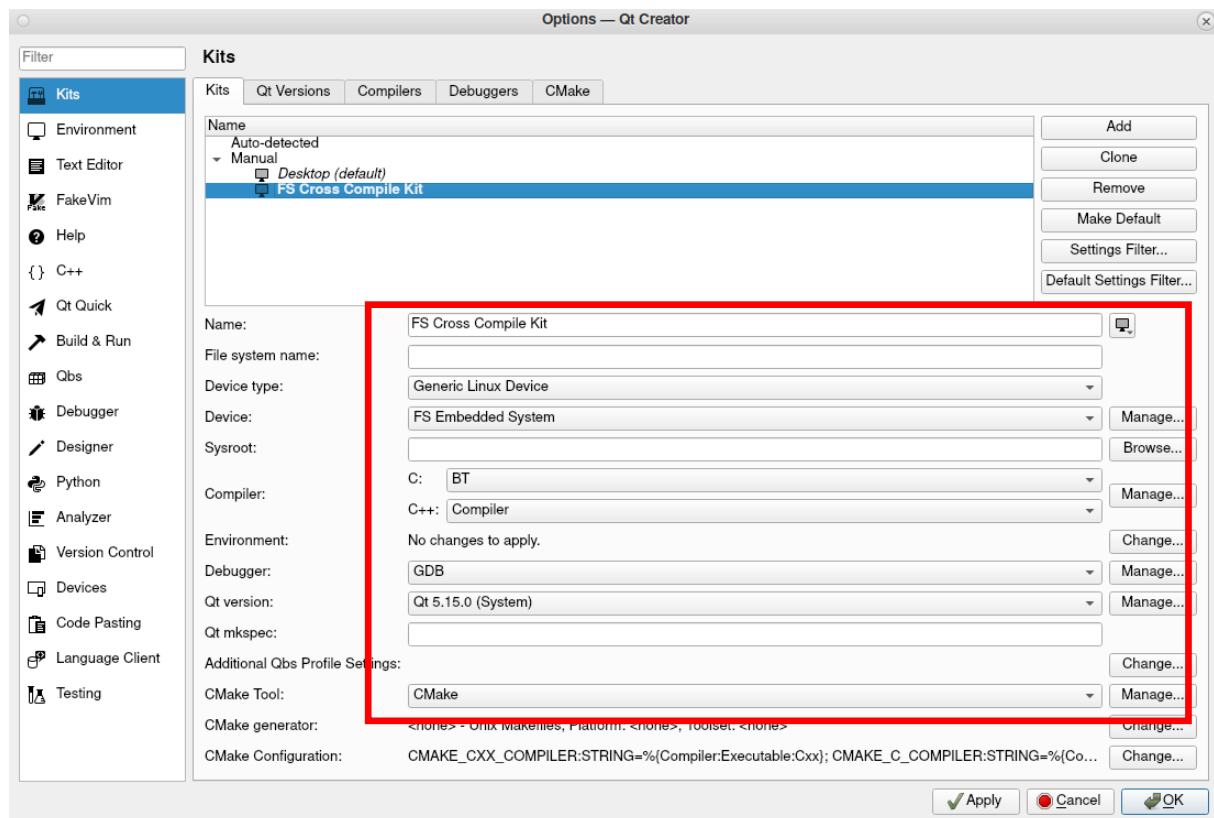


Figure 13: Build & Run - Kits

Select *Apply* and *OK* to finish the setup.

4 Create an Application

QT5

After everything is setup, you can create a new project. From *File* select *New File or Project...* Select *Application* and *Qt Widgets Application* and click *Choose...*

QT6

After everything is setup, you can create a new project. From *File* select *New Project...* Select *Application (Qt)* and *Qt Widgets Application* and click *Choose...*

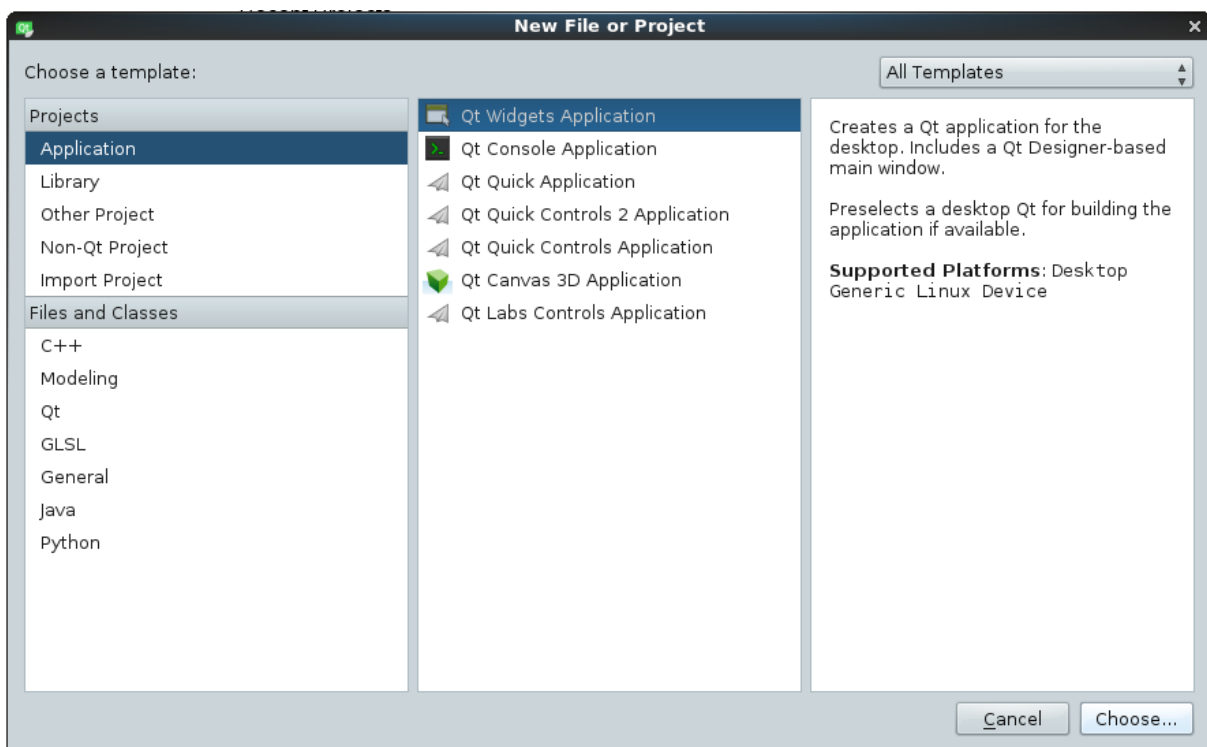


Figure 14: New File or Project

Create an Application

Choose a name of the project and the location of the project. Press *Next >*.

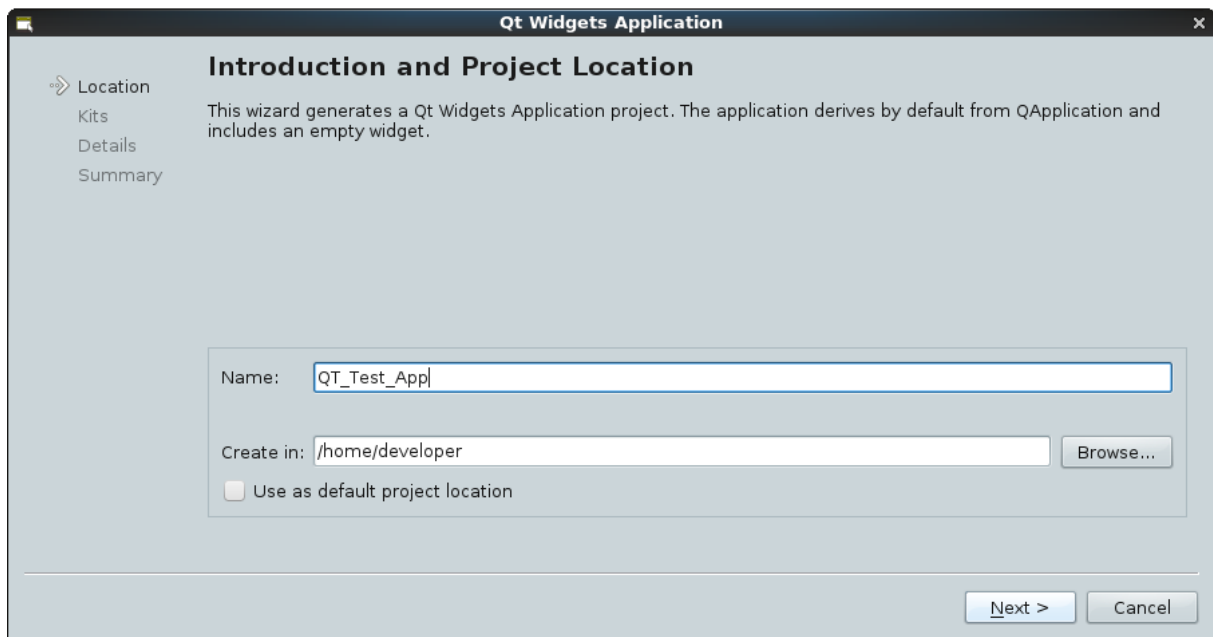


Figure 15: Introduction and Project Location

QT5:

Select the Kits which you want to use and press *Next >*.



Figure 16: Kit Selection

Choose class information's and press *Next >*.

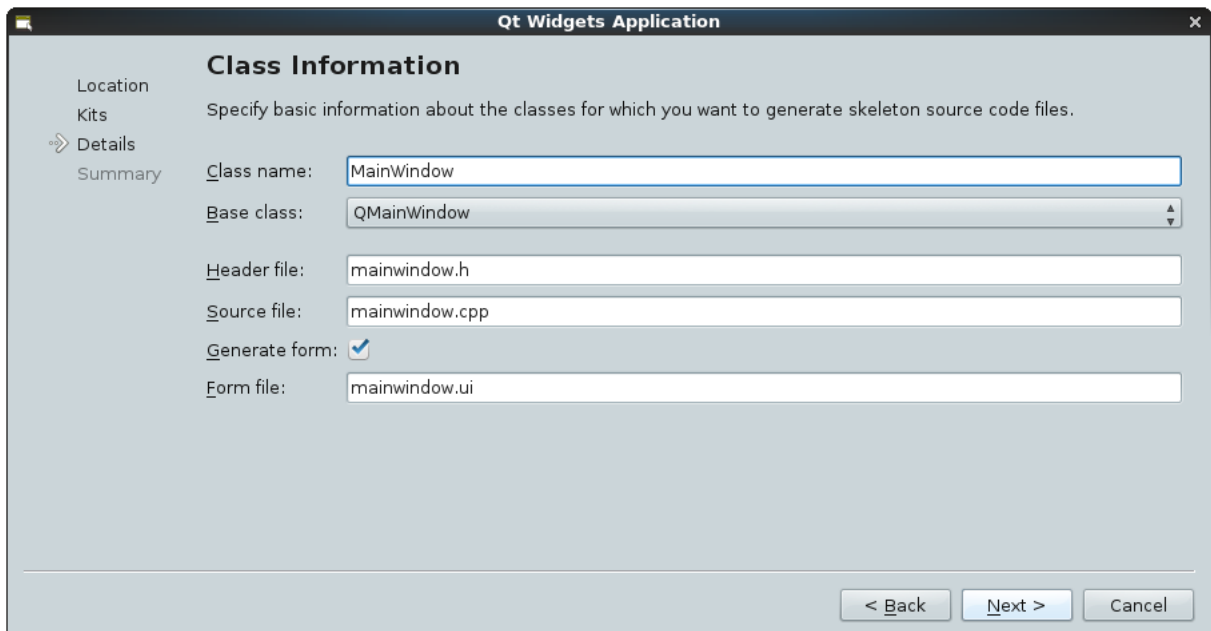


Figure 17: Class Information

Create an Application

Select Project Management and click *Finish*.

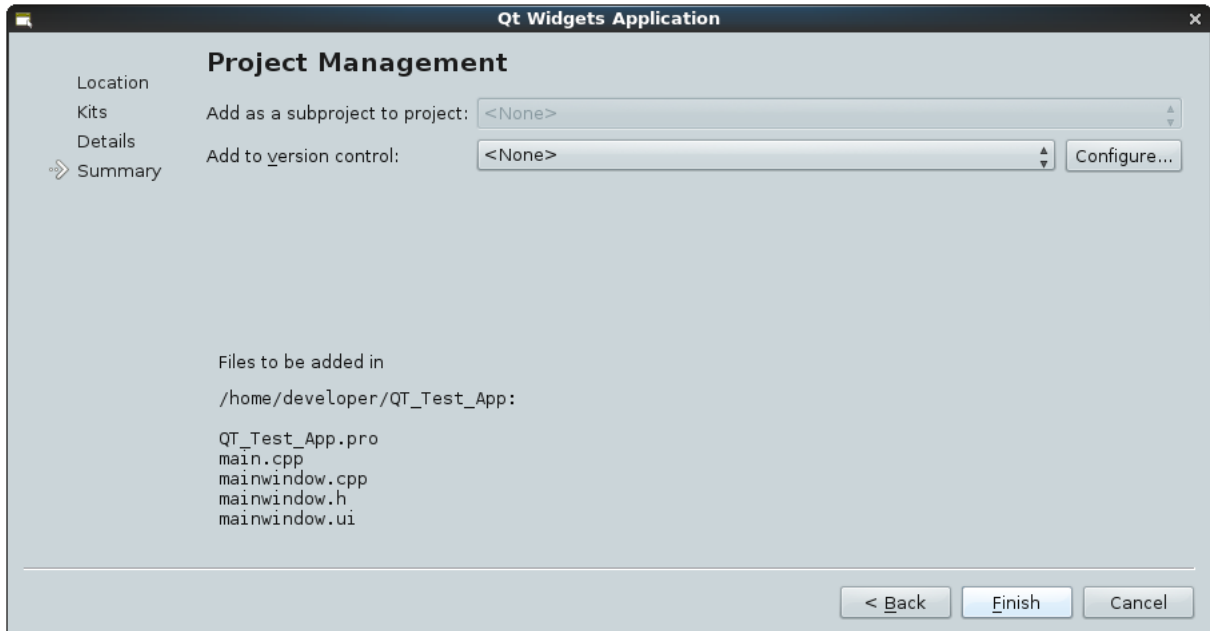


Figure 18: Project Management

QT6:

Choose *qmake* and press *Next >*.

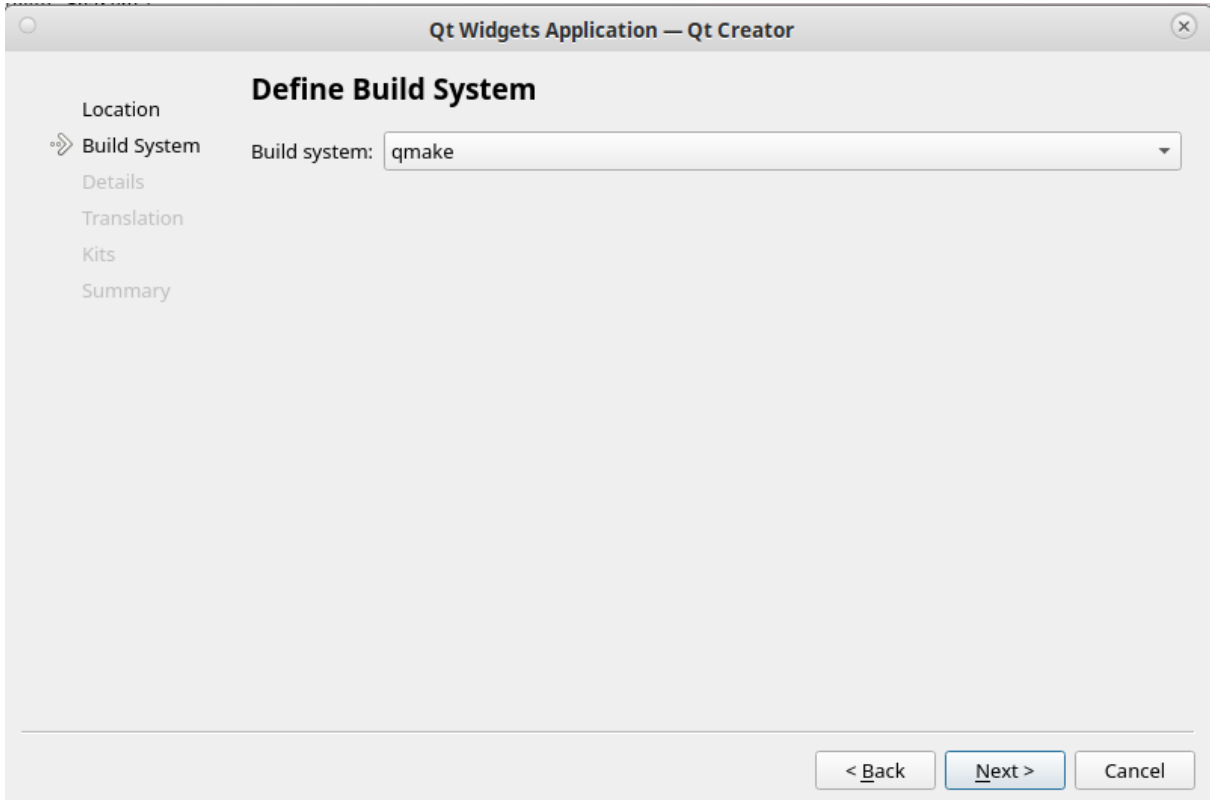


Figure 19: Define Build System

Create an Application

Choose class information's and press *Next >*.

Qt Widgets Application — Qt Creator

Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

Location
Build System
Details
Translation
Kits
Summary

Class name:

Base class:

Header file:

Source file:

Generate form

Form file:

< Back Next > Cancel

Figure 20: Class Information QT6

Choose your Translation File or leave it blank. Press *Next >*.

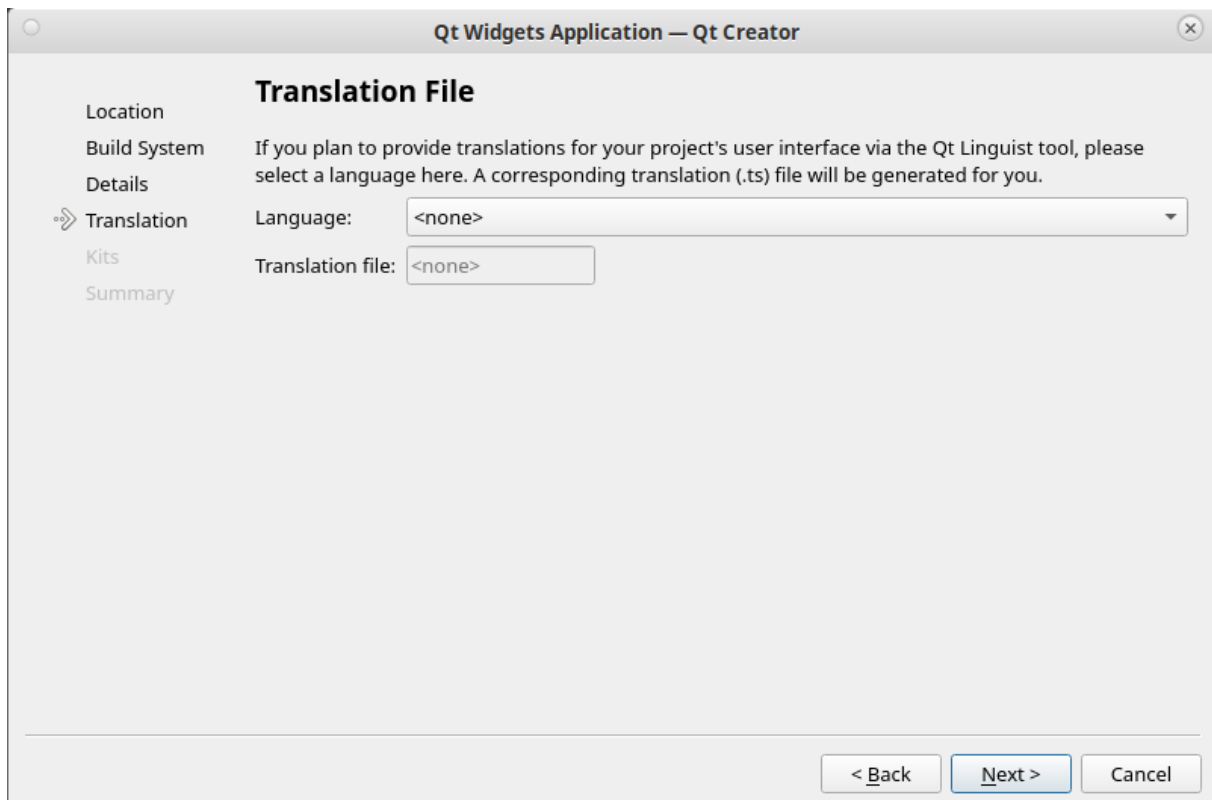


Figure 21: Translation File

Create an Application

Select the Kits which you want to use and press *Next >*.

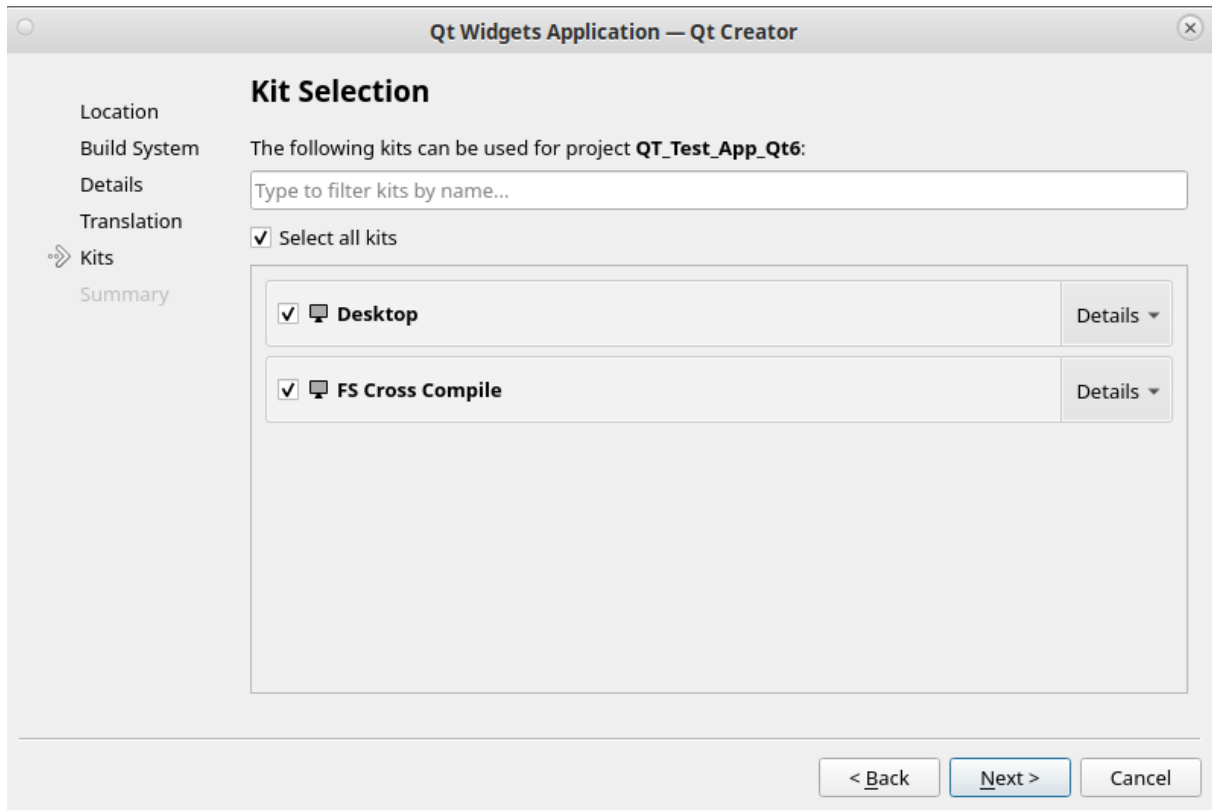


Figure 22: Kit Selection QT6

Click *Finish*.

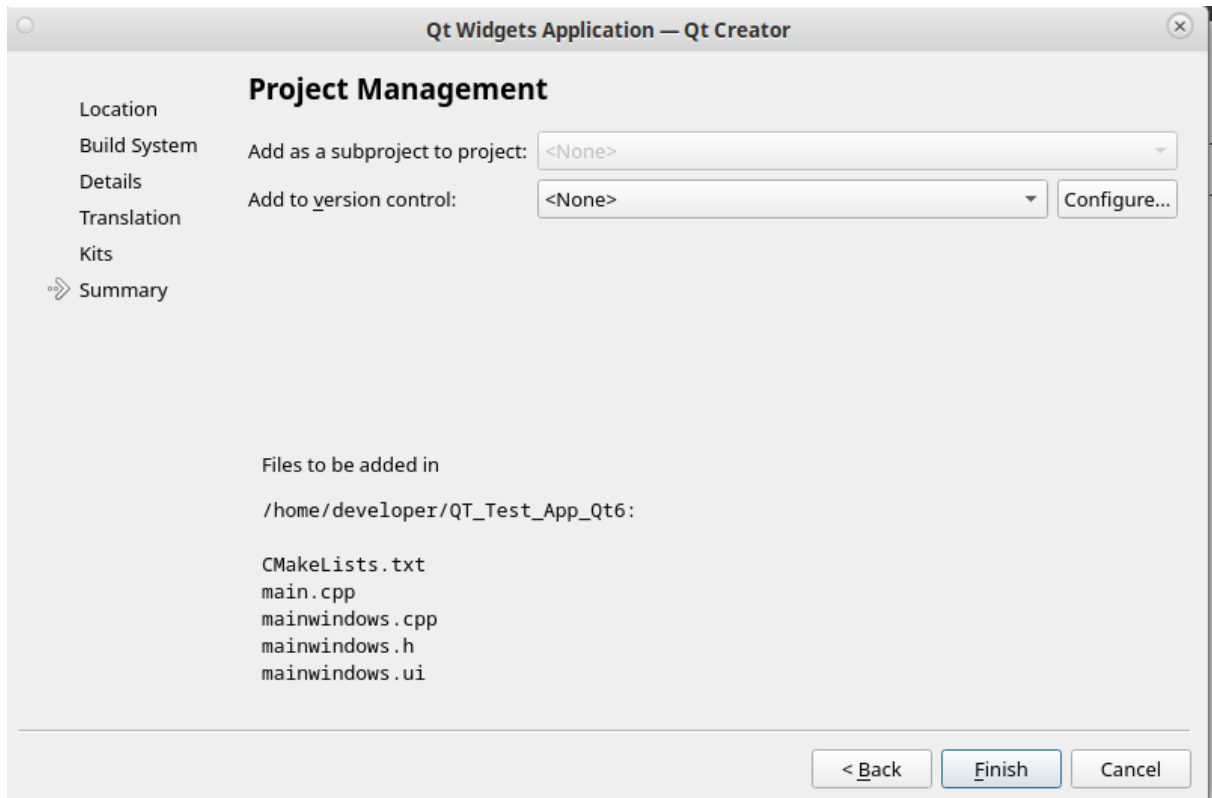


Figure 23: Project Management QT6

Now the project is successfully created.

Create an Application

4.1 Create hello world app

Open the folder Forms on the left side in project view. Double click the mainwindow.ui.

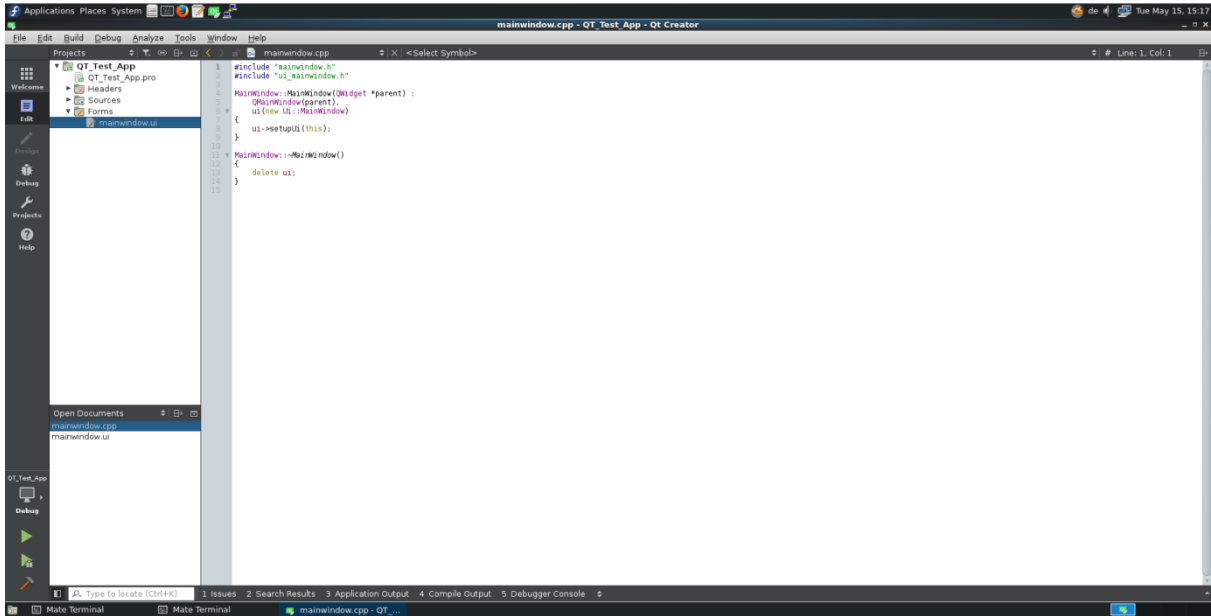


Figure 24: Project file - mainwindow.cpp

Now move 2 Push Buttons to the Main Window and move one Line Edit (refer to figure 20). After that change the name of the buttons with a double click on the button.

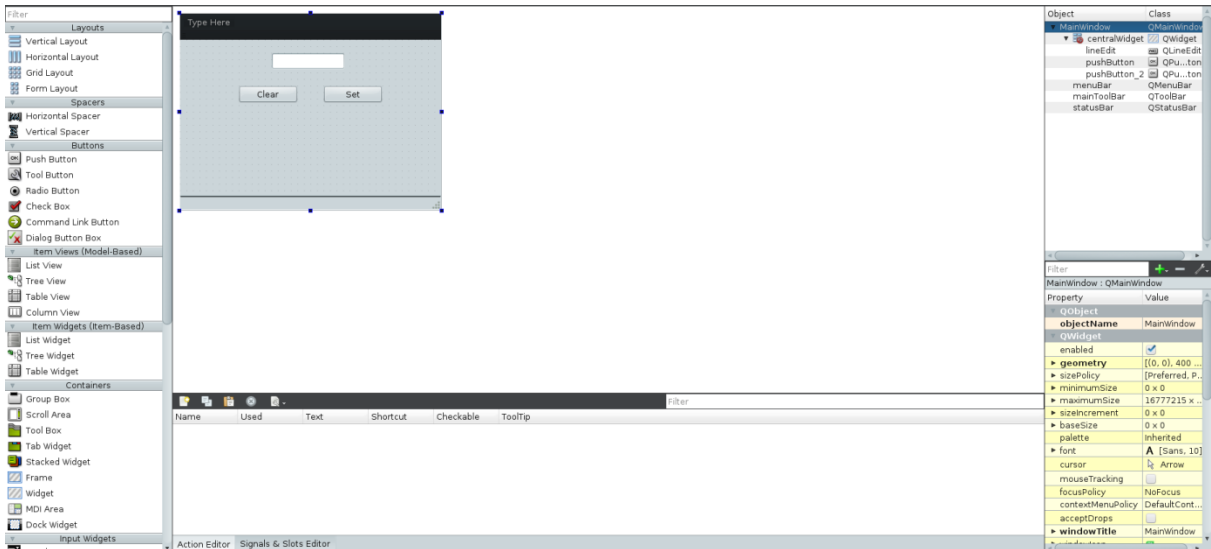


Figure 25: Project file - mainwindow.ui

After that right click the “Set” button and select *Go to slot....* Then select the signal *clicked()* and press *OK*.

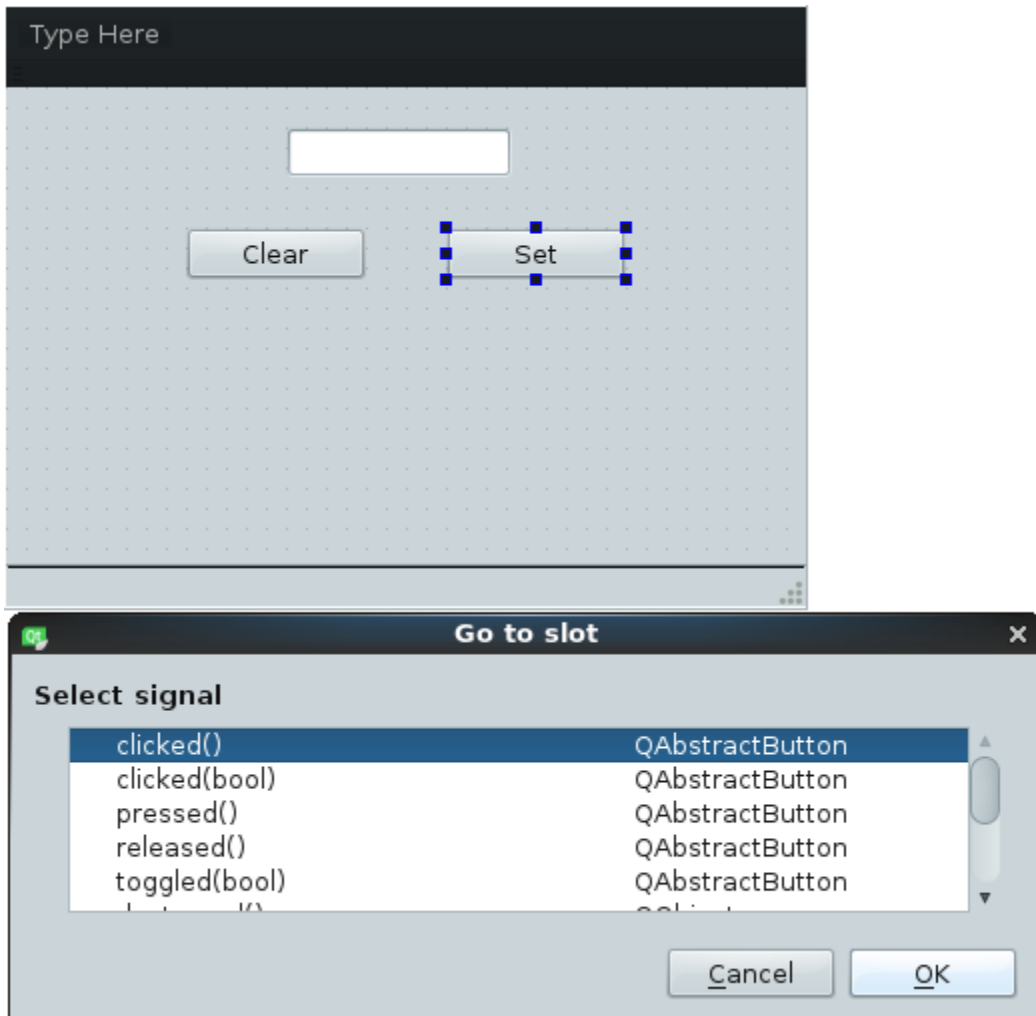


Figure 26: Select signal

The view changes to the `cpp` file. It automatically creates a function called `void MainWindow::on_pushButton_clicked()`. Now add the following line in the created function.

```
...
void MainWindow::on_pushButton_clicked()
{
    ui->lineEdit->setText("Hello World!");
}
```


Create an Application

Now double click `mainwindow.ui` again. After that right click the “Clear” button and select *Go to slot....* Then select the signal `clicked()` and press *OK*. After that the view changes to the `cpp` file again. It automatically creates a function called `void MainWindow::on_pushButton_2_clicked()`. Now add the following line in the created function.

```
...  
void MainWindow::on_pushButton_2_clicked()  
{  
    ui->lineEdit->setText("");  
}
```



5 Build and Debug Application

Select *Build* in the Toolbar and press *Build Project* “QT_Test_App”. Otherwise you can click the build icon in the left toolbar. 

If you have unsaved files then the system ask you what to do with these unsaved changes. We press *Save All*, because we want to save these modifications.

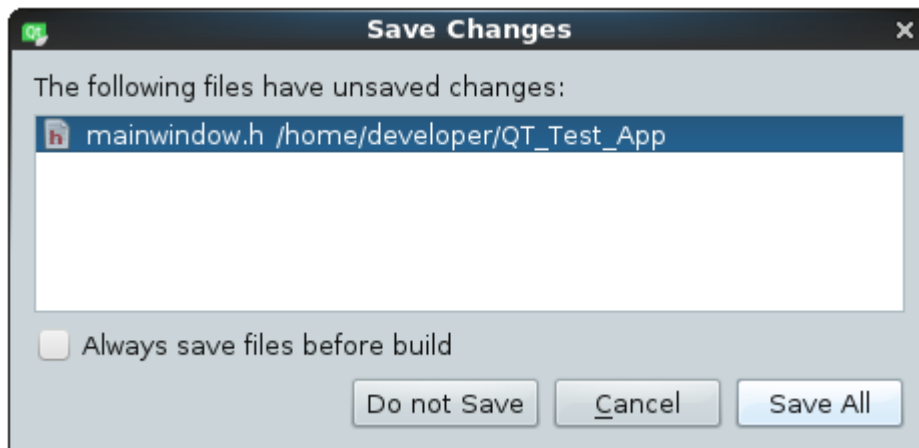


Figure 27: Save changes

To be sure everything is successfully built select at the bottom bar *4 Compile Output* and have a look at the output message.

Build and Debug Application

5.1 Desktop Run

After everything is successfully built, select *Projects* on the left sidebar. After that select *Run* in the Desktop Kit.

If you don't see the kit in the tab, select *Add Kit* and add your corresponding Kit.

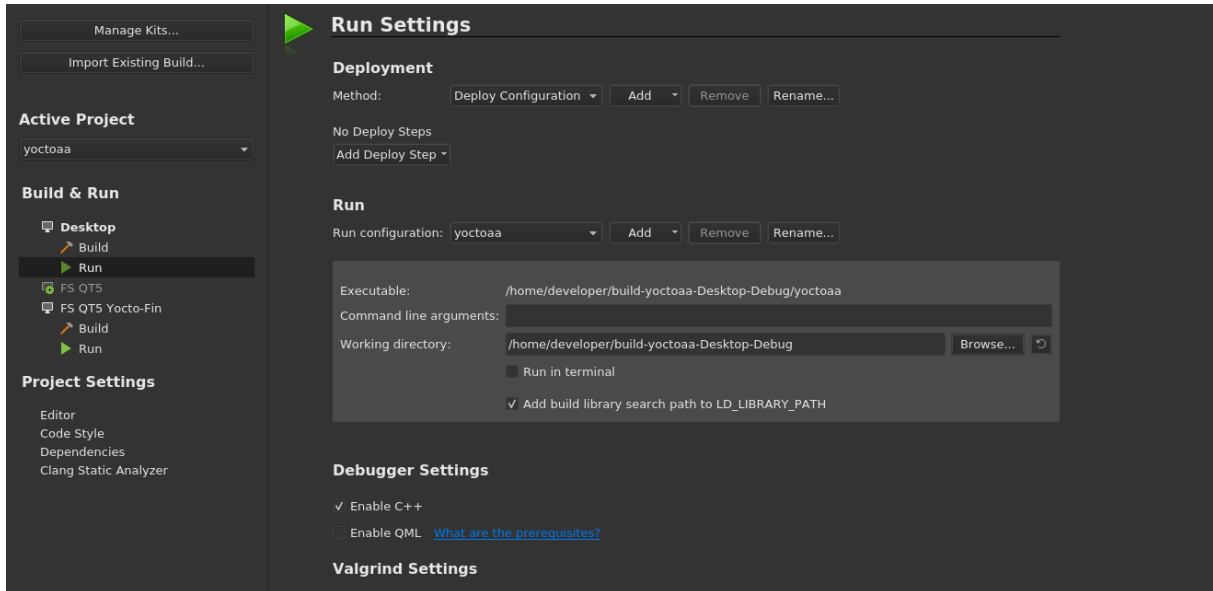



Figure 28: Project - Desktop Run

Then select *Build* on the toolbar and click *Run*. Otherwise you can click the Run icon on the left toolbar. 

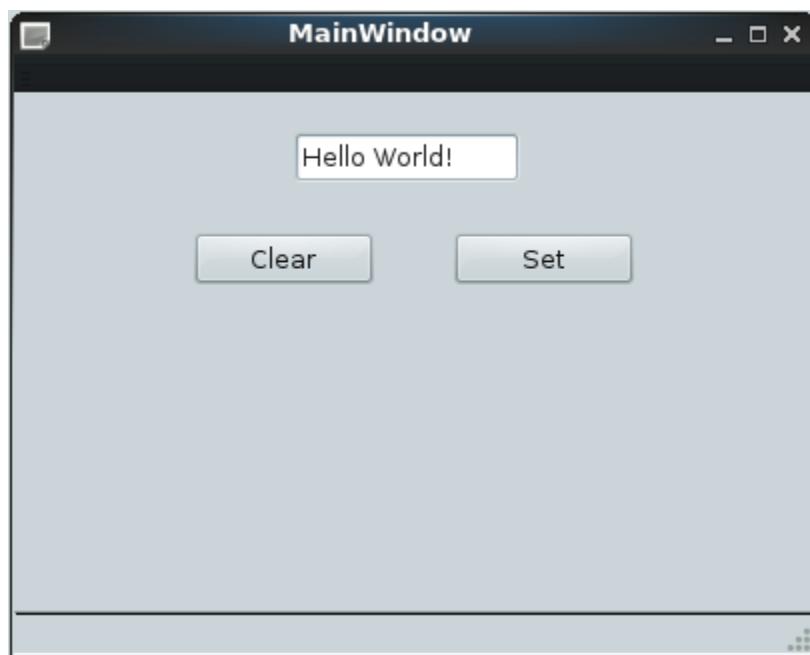


Figure 29: Application

Now the application will be executed on your development machine. Test the app by clicking the *Set* and the *Clear* buttons.

You can click the x in the right corner of the application to stop it.

5.2 FS Cross Compile Kit for ARM Run

First of all we have to setup something in our project. Select .pro file in my case QT_Test_App.pro and add at the end following lines:

```
...
linux-* {
# Buildroot
    target.path = /root
# Yocto
    target.path = /home/root
    INSTALLS += target
}
```



Now save the changes and select *Projects* on the left toolbar.

To run the compiled example for the ARM system, select *Build* in the FS Cross Compile Kit for ARM. Then select *Browse...* and create a folder and set the Build directory.

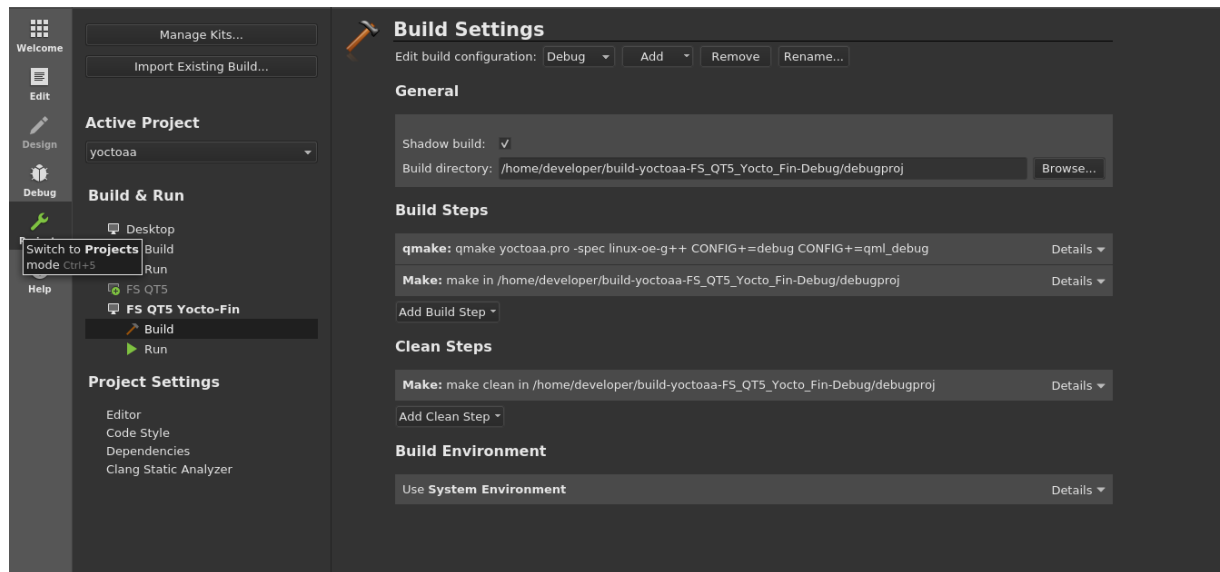


Figure 30: Project - Build FS Cross Compile Kit for ARM

Then build the system by clicking *Build* in the toolbar and select *Build Project* “QT_Test_App”. Otherwise you can click the build icon in the left toolbar.



Then select *Run* in the FS Cross Compile Kit for ARM. Normally it should automatically add the files to deploy with the correct local path and the remote directory.

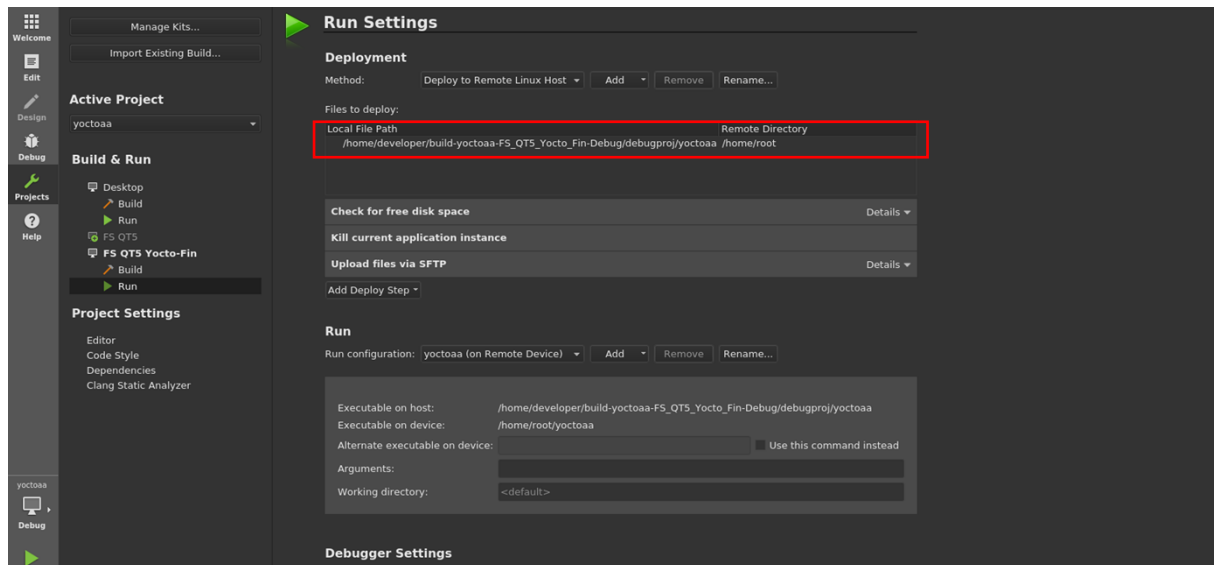


Figure 31: Project - Run FS Cross Compile Kit for ARM

Now everything is setup and you can run your application on your embedded device. Click *Build* in the Toolbar and select *Run*. Otherwise you can click the run icon on the left toolbar.



Now the application is running on your system.

5.3 Debug build

To debug the application you have to choose *Debug* in the toolbar, *Start Debugging* and *Start Debugging* again instead of Build/Run. Otherwise you can click the debug icon on the



left toolbar.

Now you can debug your application on your system.

Note:

If you are using a backend like wayland / X11 you have to fetch the environment variables from the embedded device in Qt-Creator. Select Projects on the left sidebar. After that select Run in the FS Cross Compile Kit for ARM. Now scroll down to Run Environment and press Fetch Device Environment. Now you should see the environment of the embedded device. It is necessary that the SSH connection works.

To setup debug points select your c file which you want to debug and left click on the line in your program, where you want to set a debug point. Click Debug in the toolbar to see the different step options.

Build and Debug Application

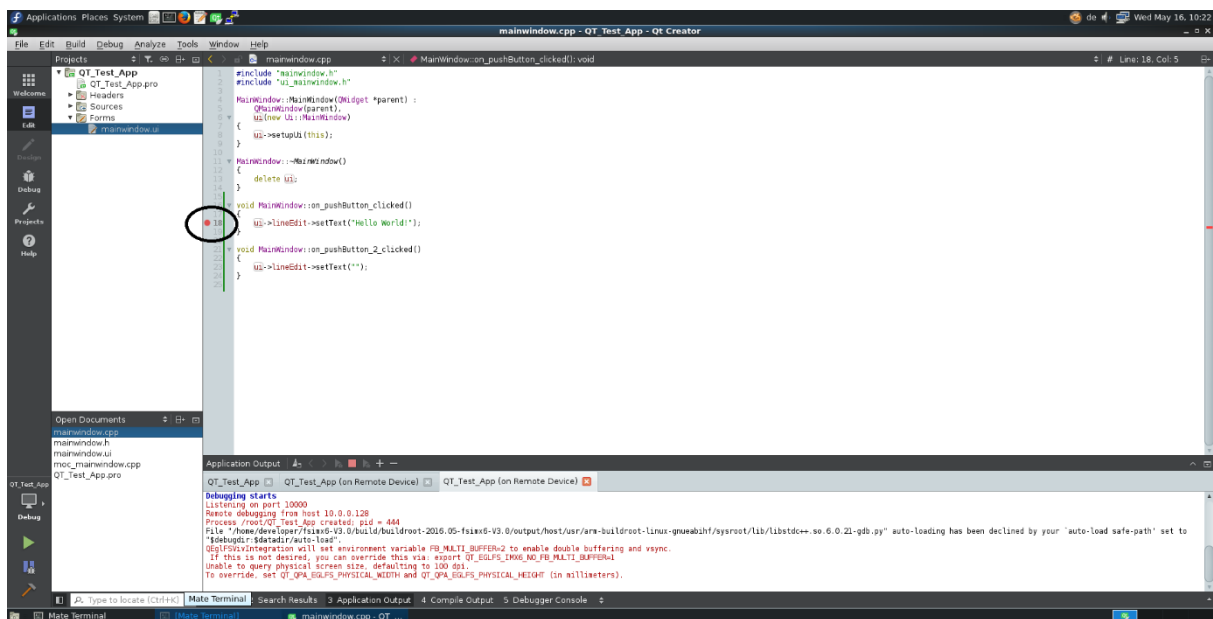


Figure 32: Debug Application





6 Appendix

List of Figures

Figure 1: QT Creator	2
Figure 2: SSH connection terminal	6
Figure 3: Options.....	7
Figure 4: Options - Devices.....	7
Figure 5: Add device configuration.....	8
Figure 6: Setup device connection.....	8
Figure 7: Device test	9
Figure 8: Options - Kits	11
Figure 9: Kits - Qt Versions	12
Figure 10:Kits - Compilers.....	13
Figure 11: Kits - Debuggers	14
Figure 12: Kits – Cmake.....	15
Figure 13: Build & Run - Kits.....	16
Figure 14: New File or Project.....	17
Figure 15: Introduction and Project Location	18
Figure 16: Kit Selection	19
Figure 17: Class Information	19
Figure 18: Project Management.....	20
Figure 19: Define Build System.....	21
Figure 20: Class Information QT6	22
Figure 21: Translation File	23
Figure 22: Kit Selection QT6.....	24
Figure 23: Project Management QT6.....	25
Figure 24: Project file - mainwindow.cpp	26
Figure 25: Project file - mainwindow.ui	26
Figure 26: Select signal.....	27
Figure 27: Save changes	29
Figure 28: Project - Desktop Run.....	30
Figure 29: Application	31
Figure 30: Project - Build FS Cross Compile Kit for ARM	32
Figure 31: Project - Run FS Cross Compile Kit for ARM	33
Figure 32: Debug Application.....	34



Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.

F&S Elektronik Systeme reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S Elektronik Systeme makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S Elektronik Systeme assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Products are not designed, intended, or authorised for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S Elektronik Systeme could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S Elektronik Systeme product for any such unintended or unauthorised application, the Buyer shall indemnify and hold F&S Elektronik Systeme and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorised use, even if such claim alleges that F&S Elektronik Systeme was negligent regarding the design or manufacture of said product.