Tech Note *TN002 - QtCreator Debugging Instructions*

Project	*RPi – DRM*	**Author**	*Sander van der Maar*
Creation Date	*21 July 2015*	**Revision**	*1*

Summary

This document describes how to remotely debug the Qt5Webkit using QtCreator.

Environmental Assumptions

The following directories are important. This document contains screenshots of dialogs showing specific values. These will be different in your installation.

Shortcut	**Description**	**Value in screenshots**
~	Home dir	/home/sander
<*buildroot-root*>	Buildroot main dir	~/Projects/buildroot/rpi2/buildroot
<*project-root*>	Dir containing all local sources	~/Projects/buildroot/rpi2

Installing Qt

The versions of Qt and QtCreator made available by Ubuntu via apt-get are outdated. QtCreator's SSH client implements a limited set of protocols. Instead, the regular Qt distribution should be downloaded and installed. [1]

Editing local versions of projects

Buildroot allows for local editing of source files and then rebuilding the target system. Set this up as follows:

1. Go to <*project-root*> and clone the QtBrowser and Qt5Webkit repositories.
2. Enter the <*buildroot-root*> directory and create a “local.mk” file. This file will override the source location of packages.
3. Edit the “local.mk” file so it points to the local source directories:
   ```
   QT5WEBKIT_OVERRIDE_SRCDIR = <project-root>/qtwebkit
   QTBROWSER_OVERRIDE_SRCDIR = <project-root>/qtbrowser
   ```
4. Test this new set-up by running “make”. There should be an 'rsync' stage for the two projects. Another way of testing is to edit a source file in the local source repo (e.g. adding comments), and rebuilding the package: “make <project>-rebuild”. The source file modifications should then also be in the “output/build/<*projectname*>” dir.

Tech Note	*TN002 - QtCreator Debugging Instructions*		
Project	*RPi – DRM*	**Author**	*Sander van der Maar*
Creation Date	*21 July 2015*	**Revision**	*1*

Opening QtBrowser and Qt5Webkit with QtCreator

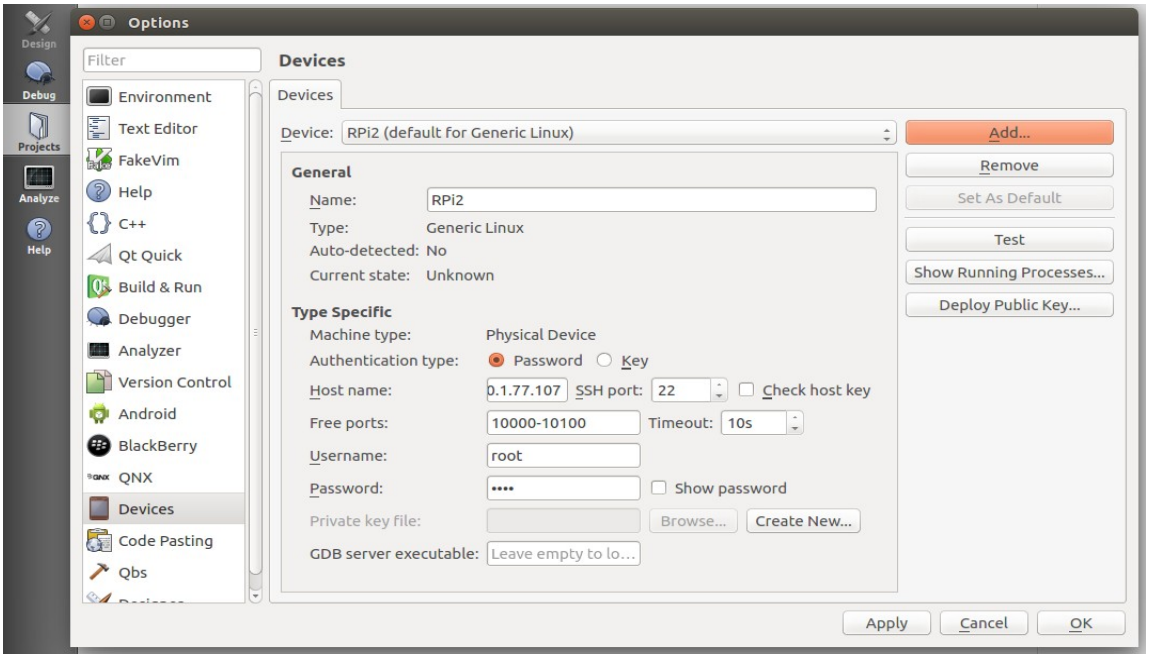
QtCreator reads a project's file list and other settings from the project's .pro-file. We will open both QtBrowser and Qt5Webkit in QtCreator. The respective pro files are located here:

1. For QtBrowser: <*project-root*>/qtbrowser/qtbrowser.pro
2. For Qt5Webkit: <*project-root*>/qtwebkit/WebKit.pro. Note that qtwebkit contains many .pro-files. These are all included by the “WebKit.pro” file. There is no need to open these separately.

Kits

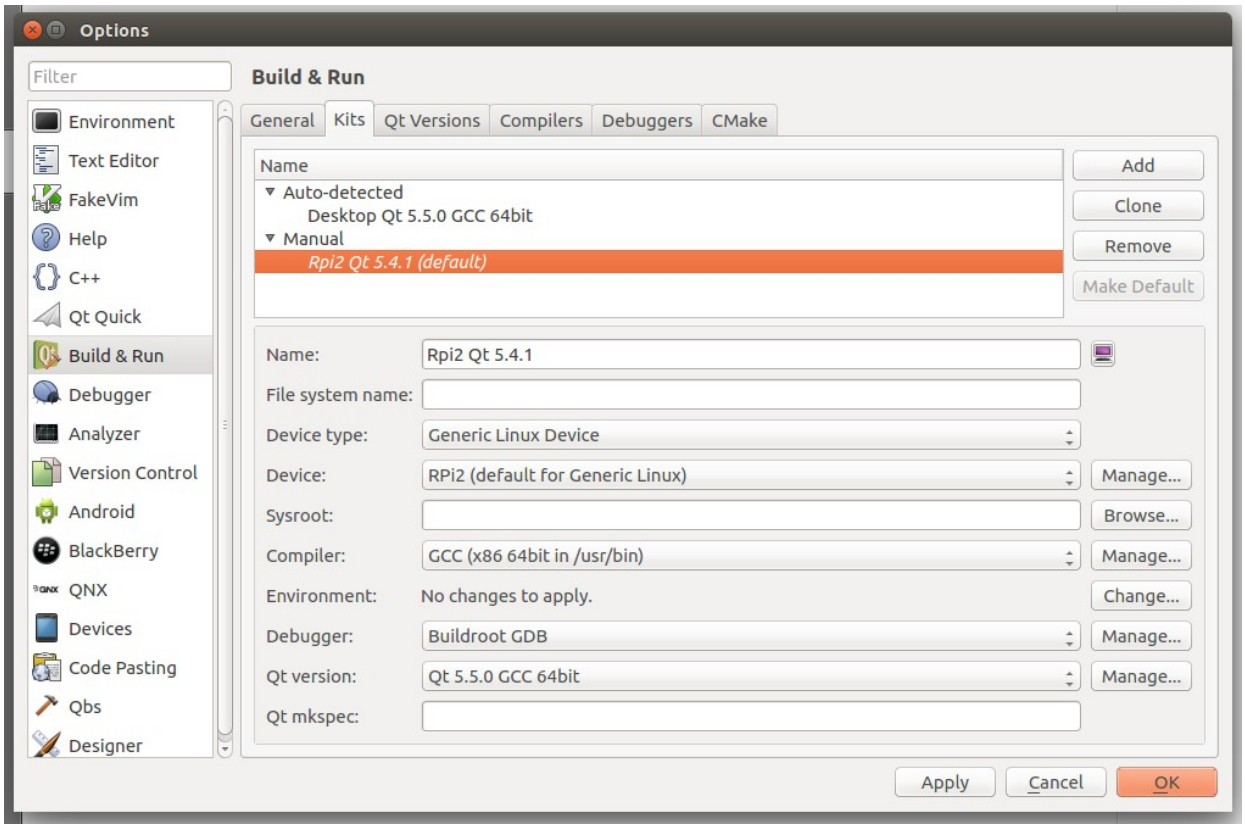
QtCreator manages different installations of Qt as “kits”. These kits contain information about the device on which the application is run, the used compiler, used debugger, etc. When first installed, QtCreator already has one kit: the system default. We will have to register the one used by buildroot when it builds our project. This is done as follows:

1. Open the “Manage kits...” dialog. Change to “Projects” mode by clicking on “Projects”, then click on “Manage kits...”
2. Click on “Devices”



3. Click on “Add...” and select “Generic Linux Device”
4. Enter the correct information about the remote device.
5. Click on “Build & Run”

6. Select “Kits” tab
7. Click on “Add”
8. Enter a descriptive name for the kit
9. Select the device just created
10. Click on “Manage...” next to “Debugger:” to select the gdb executable in “*<buildroot-root>/output/host/usr/bin/*”. It is prefixed by the target archicture, toolchain, OS, and ABI, for example : “arm-buildroot-linux-gnueabihf-gdb”



Overriding default build steps

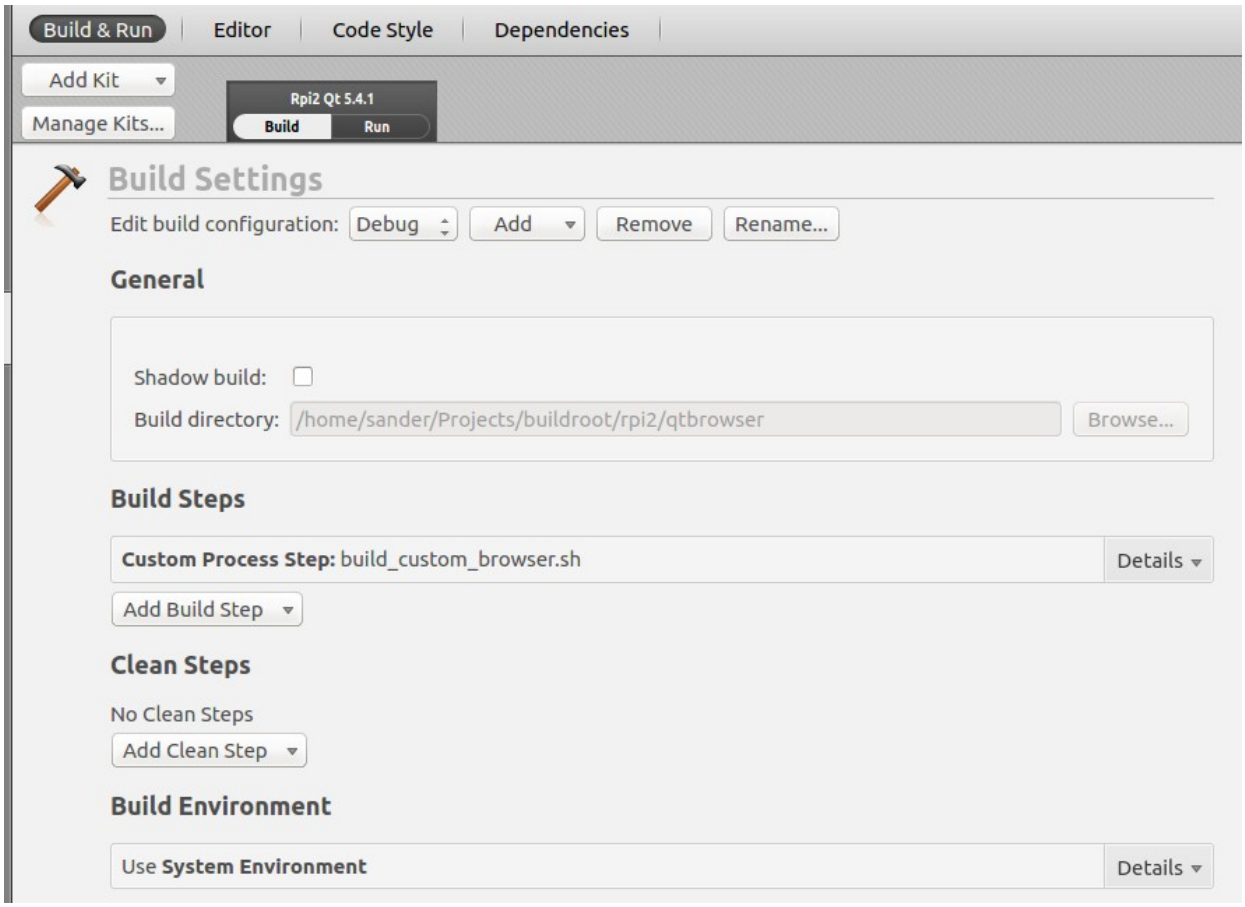
Normaly QtCreator would build the projects using a particular kit. In our case we will have the build to be performed by buildroot. For this we will create an external bash script calling buildroot make and instruct QtCreator to execute this script instead of its regular build steps:

1. Create a bash script for each project that changes into the buildroot dir, forces a rebuild of the particular project, and tells buildroot to create the target file system:

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

```
cd <buildroot-root>
make <project>-rebuild
make
```

2. Open “Build & Run” tab of “Project”
3. Disable “Shadow build”
4. Remove all Build and Clean Steps
5. Add a Build Step
6. Enter the full path to the script in the “Command” text box



7. Clicking on the hammer in the lower left corner of the screen should show building output of buildroot in the “Compile Output” window.

Adding debugging symbols and turning off optimization

To allow for debugging, binaries need to be built with debugging symbols. Turning off optimization allows for a better debugging experience, because all variables can be inspected and the debugger will step through the source linearly.

1. Adding debug symbols to the qtbrowser is done by adding the debug setting to the CONFIG setting in the .pro-file. Add this line to the beginning of the .pro-file:
   `CONFIG += debug`
2. Turning off optimization is done by passing an extra CXXFLAG to the make process. Add this line to the beginning of the .pro-file as well:
   `QMAKE_CXXFLAGS = -O0`
3. Also add these new CXXFLAGS to the .pro-file of Qt5Webkit:
   `QMAKE_CXXFLAGS = -O0`
4. Turn on debugging symbols in buildroot here: Build Options--->build packages with debugging symbols.
5. Disable stripping of packages here: Build Options--->strip command for binaries. Set it to “none”
6. Disable optimization here: Build Options--->gcc optimization level. Set it to 0.

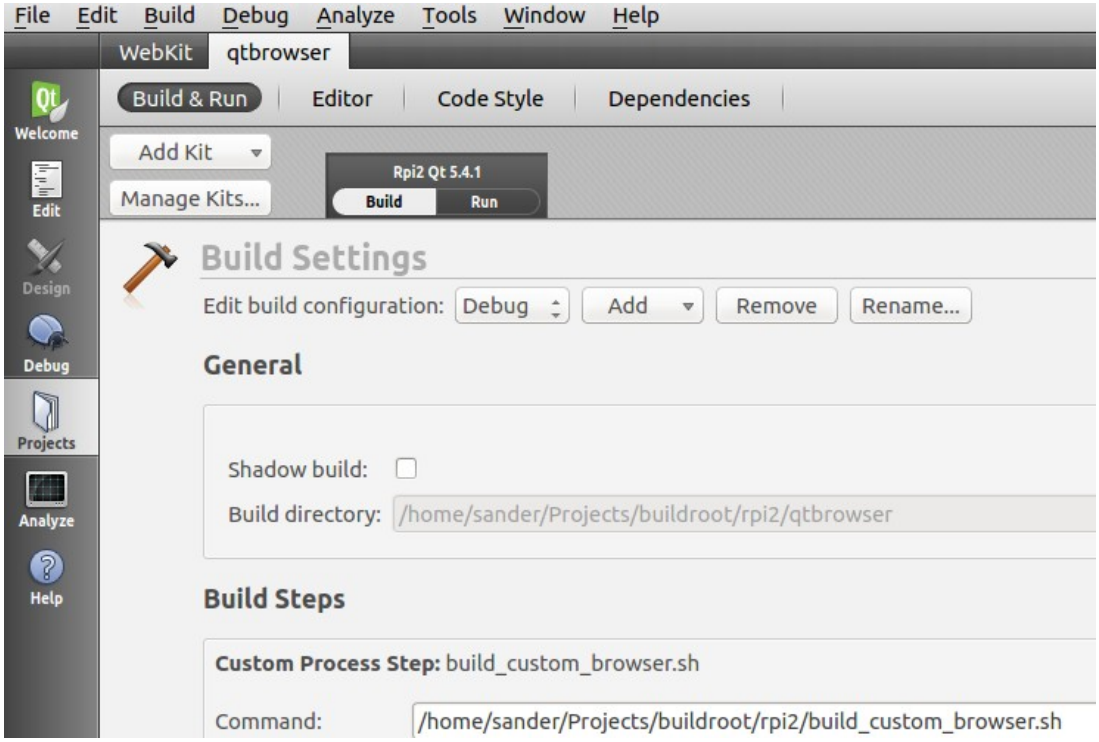
Enable Python in built GDB

QtCreator requires GDB to support Python. Turn it on in buildroot here: Toolchain--->Build cross gdb for host. Select version 7.7.x here: Toolchain--->GDB Debugger Version. Also make sure “gdbserver” for the target is selected: Target Packages--->Debugging, Profiling, and Benchmark--->gdb--->gdbserver.

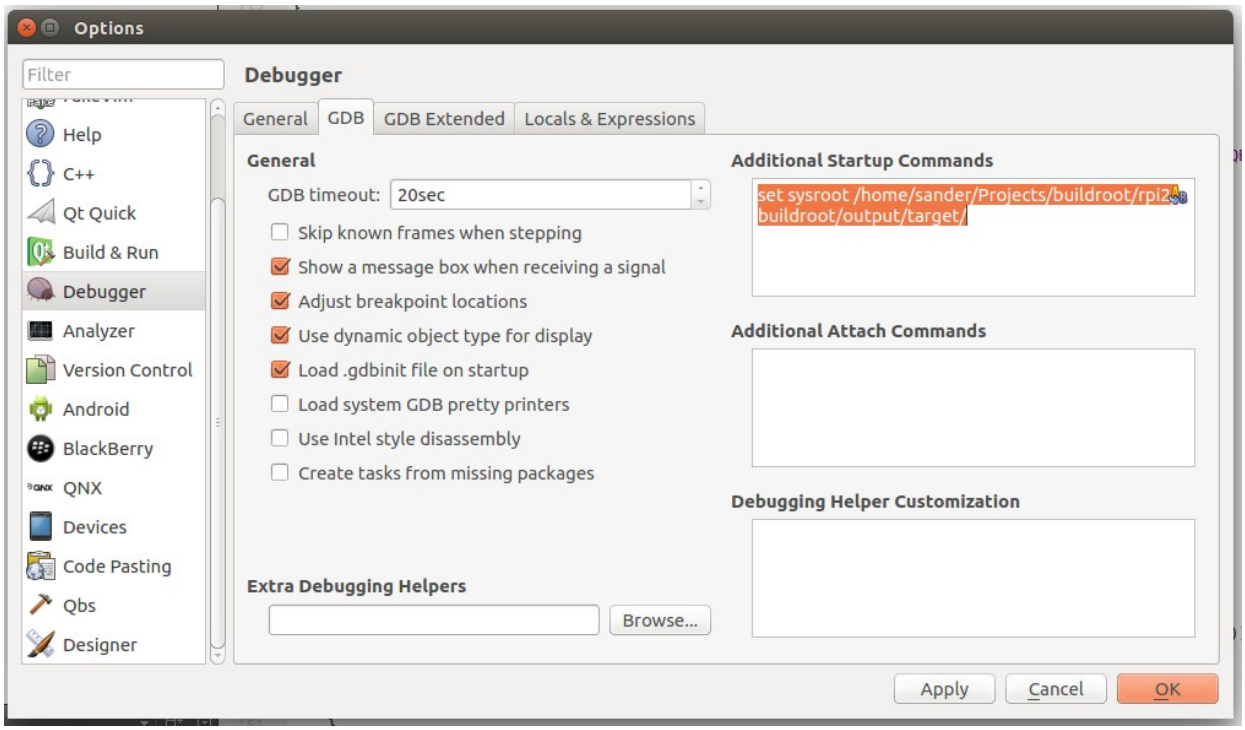
Actual debugging

Once everything is built, we will have to start gdbserver on the target machine and tell QtCreator to connect to it.

1. Make sure the latest version of the image generated by buildroot is running on the target machine.
2. On the target machine, start “gdbserver” to debug “qtbrowser”:
   `gdbserver localhost:2345 qtbrowser --url="<some url>"`
3. Make sure qtbrowser is the active project in QtCreator: right-click on the project and select “Set as active project”
4. Make sure our custom Kit is selected: switch to Project mode and make sure our Kit is selected

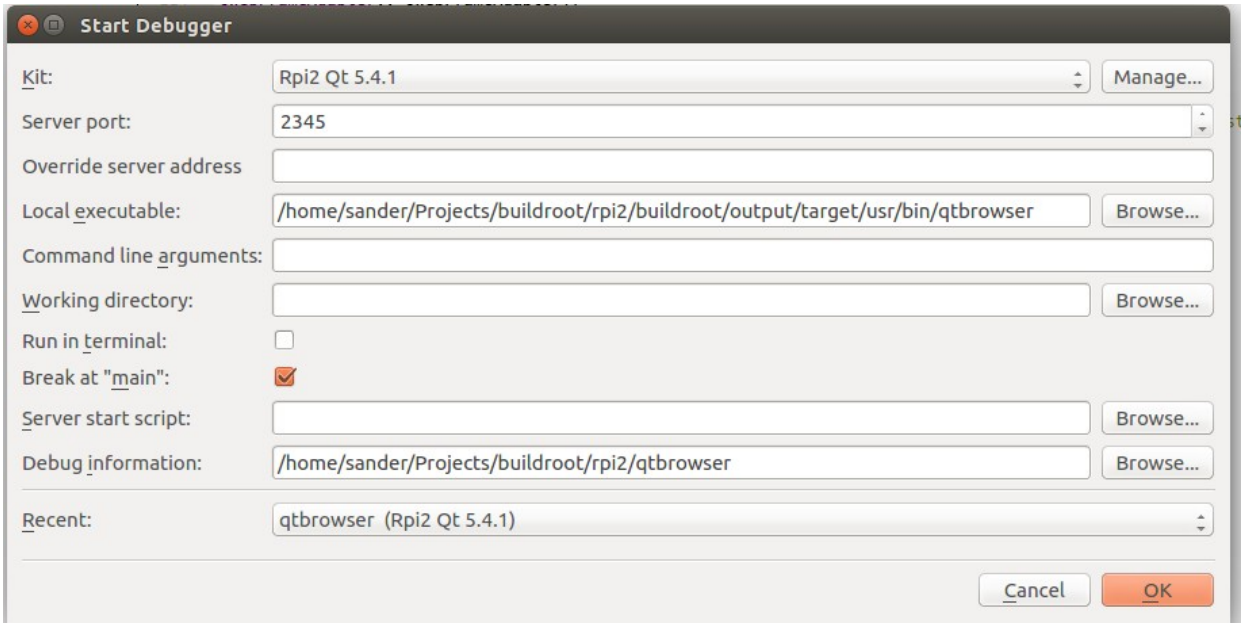


5. Make sure the sysroot of GDB is set correctly. Click on Tools->Options, select “Debugger”, open the “GDB” tab. In “additional startup commands”, enter:
   `set sysroot <buildroot-root>/output/target/`



6. Click on: Debug->Start Debugging->Attach to Running Debug Server. Enter “2345” as the

server port. Point “Local Executable” to the location of qtbrowser in the buildroot target directory. Set Debug Information to the local directory with the qtbrowser project opened in QtCreator.



7. Press ok, and if all goes correctly, execution should be halted at the beginning of main, in qtbrowser.

Todo

- Also make it work with stripped target binaries
- Set up NFS to use local version of “target/” dir
- Speed up debugging with stripping a subset of debugging symbols
- QtBrowser should use debugging and optimization levels set by buildroot

References

[1] https://www.qt.io/download-open-source/