



## **Grayhill 3Dxx Display Products**

Setup and Usage of Qt 5.9.3 Development Software – Windows

Revision C

## Revision History

Revision	Date	Description
A		This release intentionally skipped to provide consistent revisions with releases
B		This release intentionally skipped to provide consistent revisions with releases
pC	02/27/2018	Initial release for Windows Updated glibc support to include gconv_UTF and ZH
C	04/06/2018	Document clean-up and process improvements Updated Grayhill examples to be runnable upon loading Updated gcclibs_4.8.3

## Table of Contents

Revision History .....	2
Table of Contents .....	3
Introduction.....	5
Supported Hardware Products .....	6
Recommended Equipment from Grayhill .....	6
Other Recommended Equipment .....	7
Software Required .....	7
Installation Overview .....	8
Download and Install Qt Creator .....	9
Download and Install Support Files .....	24
PuTTY .....	24
WinSCP .....	25
Grayhill Qt Support Files .....	26
Configuring 3Dxx Display's IP Address .....	28
Transfer Configuration Files to Display.....	34
Execute Configuration Scripts.....	40
Selecting a 3Dxx Qt Widget Demo Project .....	44
Build and Run a 3Dxx Embedded Application (Widget).....	45
Appendix A: Configuring a Manual Qt Kit for Grayhill Displays .....	49
Device.....	51
Compiler.....	57
Debugger .....	60
qmake .....	61
Kit.....	62
Appendix B: Configuring a 3Dxx Project .....	64
Build .....	68
Run .....	72
Quick Reference .....	77
Appendix C: Debugging.....	79
Appendix D: Build and Run 3Dxx Desktop Application .....	85
Appendix E: Build and Run QML Demonstration Program .....	88
Appendix F: Setting up a 3Dxx Qt Program to Run at Boot Up.....	89

<b>Appendix G: Interfacing 3Dxx Hardware from QT Software .....</b>	<b>90</b>
LCD .....	90
LCD Backlight .....	90
Camera Driver Interface .....	91
CAN Driver Interface .....	95
Digital I/O Driver Interface .....	99
Analog Inputs (Model 3D70 only) .....	103
Buzzer (Models 3D70, 3D2104) .....	105
Audio Output (Model 3D70 only) .....	106
<b>Appendix H: Setting 3Dxx Flash File System R/W Mode .....</b>	<b>107</b>
<b>Appendix I: Building Qt Library Source .....</b>	<b>108</b>
<b>Appendix J: Dynamic IP Address .....</b>	<b>113</b>
<b>Appendix K: Static IP Address .....</b>	<b>114</b>

## Introduction

This document describes:

- Setup and usage of the Qt-based development environment for Grayhill 3Dxx display products
- Code development for a 3Dxx Display product in the Qt IDE
- Accessing various 3Dxx hardware features via this code
- Loading developed application code onto a 3Dxx Display product

This Qt cross-platform development environment runs under Windows 7 and Windows 10.

The different features of the Grayhill displays are described below as are differences in their installation.

This document is intended for use by software developers who are familiar with programming in C/C++ using the Qt framework. Experience developing applications for Linux platforms is a definite plus.

Screen shots were designed to be as accurate as possible and should be used for reference.

**Note:** Qt is licensed under the terms of LGPL and GPL; these are open-source licensing agreements. Please reference <https://www1.qt.io/qt-licensing-terms/> for a detailed explanation. Additional information is also located at <https://www.gnu.org/licenses/licenses.html>.

## Supported Hardware Products

The Qt-based development environment is supported on the following Grayhill 3Dxx Color Display Models:

- 3D50
- 3D70
- 3D2104

The table below summarizes the key features of each of these models. Note that the features of a specific product may vary depending on the purchased hardware configuration.

Model Number	3D50	3D70	3D2104
Display Size (inches)	5	7	10.4
Pixel Count (w x h)	800 x 480	800 x 480	1024 x 768
Touch Screen Input	Yes	Yes	Yes
Real Time Clock	Yes	Yes	Yes
CAN Ports	2	2	3
Camera Inputs	2	3	4
USB ports	1 (maintenance only)	1 (maintenance only)	1 (maintenance only)
RS232	1 (maintenance only)	1 (maintenance only)	1 (maintenance only)
Built-in Ethernet	0	1	1
Digital Input (dedicated)	1	4	0
Digital Output (dedicated)	1	4	0
Digital Input / Output	3	0	4
Analog Input	0	2	0
Audio Output	No	1 channel	No
Buzzer	No	Yes	Yes

## Recommended Equipment from Grayhill

If using Model 3D50 5 Inch Display:

3D50DEV-100      3D50 Development Kit

If using Model 3D70 7 Inch Display:

3D70DEV-100      3D70 Development Kit

If using Model 3D2104 10.4 Inch Display:

3D2104DEV-100      3D2104 Development Kit

## Other Recommended Equipment

- An Ethernet port connected to a DHCP server that can be connected to the 3Dxx Display. This port should be on the same network as the development PC.
- PC Running Windows 7/10 with the following minimum configuration:
  - 4 GB RAM
  - 10 GB available hard drive space on C:
  - Ethernet port
  - RS232 Port (or USB to serial adapter)
  - Internet Access

## Software Required

The following files are available for download from Grayhill at: <http://www.grayhill.com/qt43d>

- QtInstaller
  - Qt Windows online installer
- QtGhSupport
  - Files for building Qt applications
  - Support utilities for building Qt applications
  - Example projects from Grayhill
- 3Dxx\_Qt\_Usage\_Guide\_Windows.pdf (this document)

## Installation Overview

This is a brief overview of the installation steps for the Qt-based development environment for a Grayhill 3Dxx Display.

- First connect the 3Dxx Development Kit hardware to the PC being used. This includes connecting the serial port and Ethernet port interfaces. For the 3D50 Display this procedure is described in detail in the document “3D50DEV Quick Start Guide.pdf” and for the model 3D70 Display it is described in the document “3D70DEV Quick Start Guide.pdf”.
- Qt Creator for Windows is downloaded and installed on the development PC
- Additional third party utilities are downloaded, installed, and configured
- Grayhill support files are downloaded and installed
- The serial and Ethernet links to the target 3Dxx Display hardware are established.
- Configuration scripts are run on the target 3Dxx display board and Windows to configure the 3Dxx display to operate with Qt instead of VUI Builder<sup>®</sup>. The display scripts will need to be run on each 3Dxx Display product that will be operated with Qt.
- Finally instructions are provided on how to open and run a Qt demonstration project on the 3Dxx Display target hardware. This demonstration project illustrates:
  - using touch screen “buttons”
  - using touch screen swipes
  - setting the 3Dxx backlight
  - operating the 3Dxx camera input
  - accessing and setting the real time clock

For the 3D70 Display there are also samples of using the audio output, the analog input, and the internal buzzer.

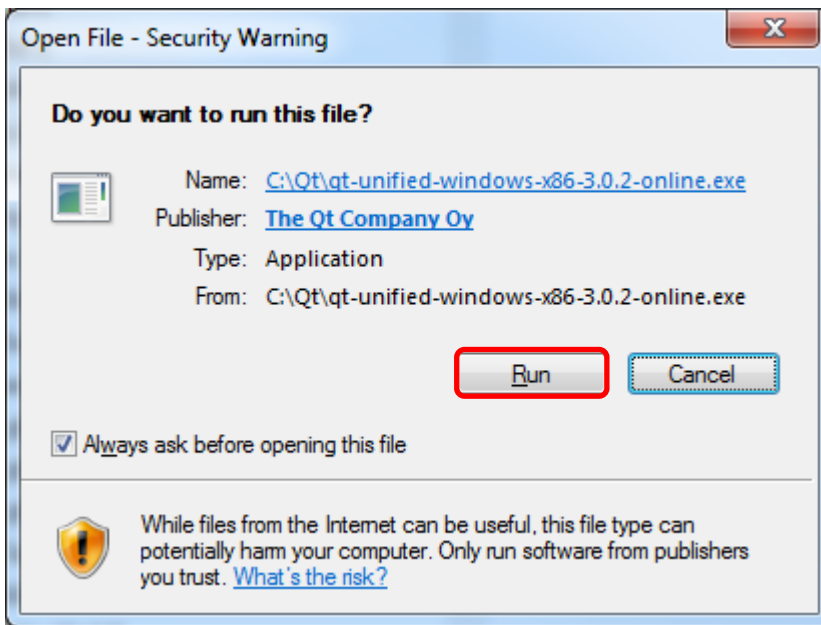
Instructions for using the desktop simulator are in Appendix D: Build and Run 3Dxx Desktop Application.



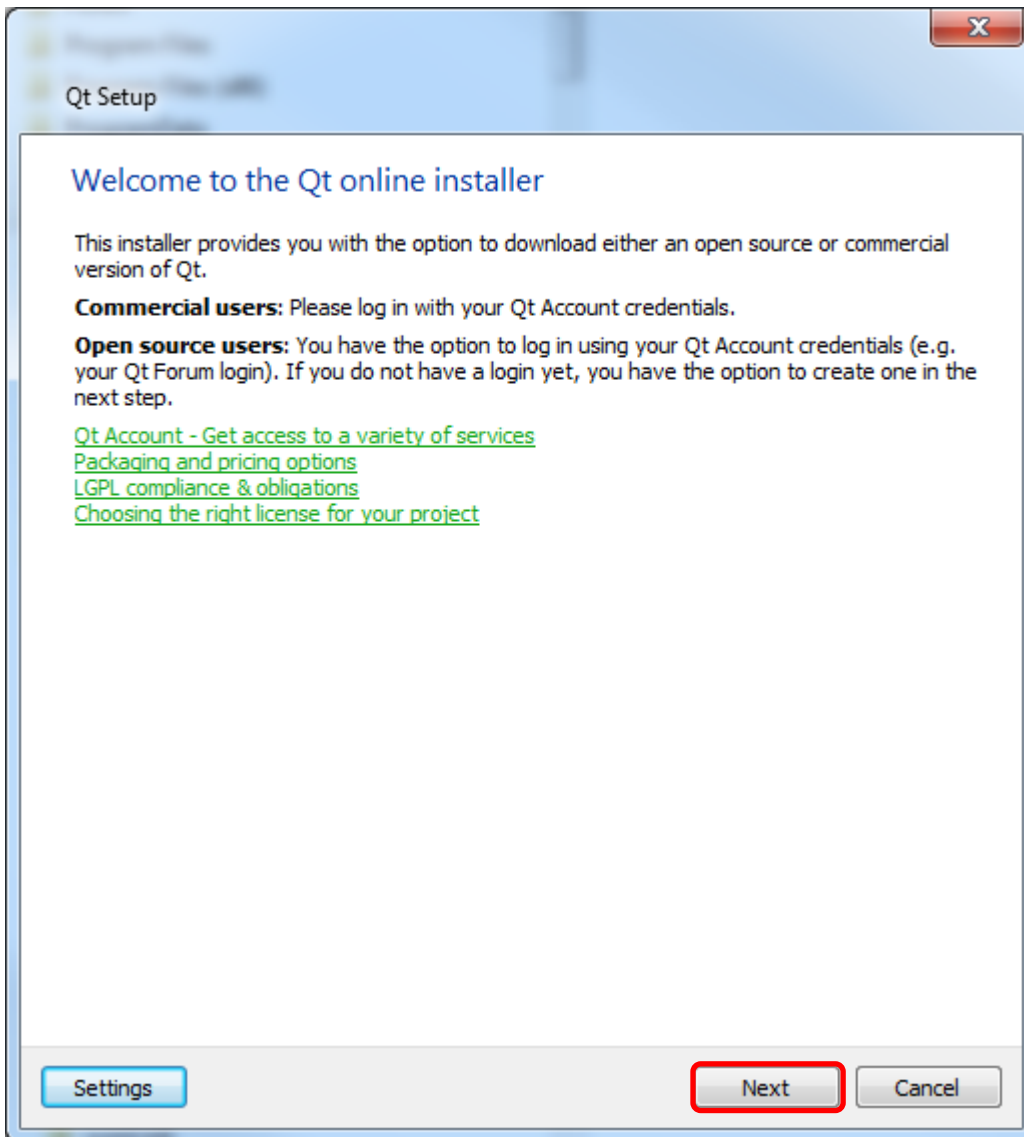
## Download and Install Qt Creator

In this section, the Qt on-line installer will be downloaded from Grayhill and executed to download all the necessary files from Qt. Once all the files are downloaded; Qt will be installed on the development PC.

- Using your favorite web browser, download “Qt Creator Windows Installer” from the Grayhill website
- Open the downloads folder and double click on the file to execute the installer



- Click “Run”



- Click Next

The image shows a 'Qt Setup' dialog box with a title bar containing a back arrow, the text 'Qt Setup', and a close button (X). The main content area is titled 'Qt Account – Your unified login to everything Qt'. Below this, it says 'Please log in to Qt Account'. There are two input fields for 'Login': 'Email' and 'Password'. A green link 'Forgot password?' is positioned below the password field. Underneath, there is a red-bordered box containing the text 'Need a Qt Account?'. Below this box are three input fields for 'Sign-up': 'Valid email address', 'Password', and 'Confirm Password'. A checkbox is located below the 'Confirm Password' field with the text 'I accept the [service terms.](#)'. At the bottom of the dialog, there are three buttons: 'Settings', 'Skip' (which is highlighted with a red border), and 'Cancel'.

- Create an account, if desired – otherwise click “Skip”

The image shows a Windows-style dialog box titled "Qt Setup". The main heading is "Qt Account – Your unified login to everything Qt". Below this, there are two sections: "Please log in to Qt Account" and "Need a Qt Account?".

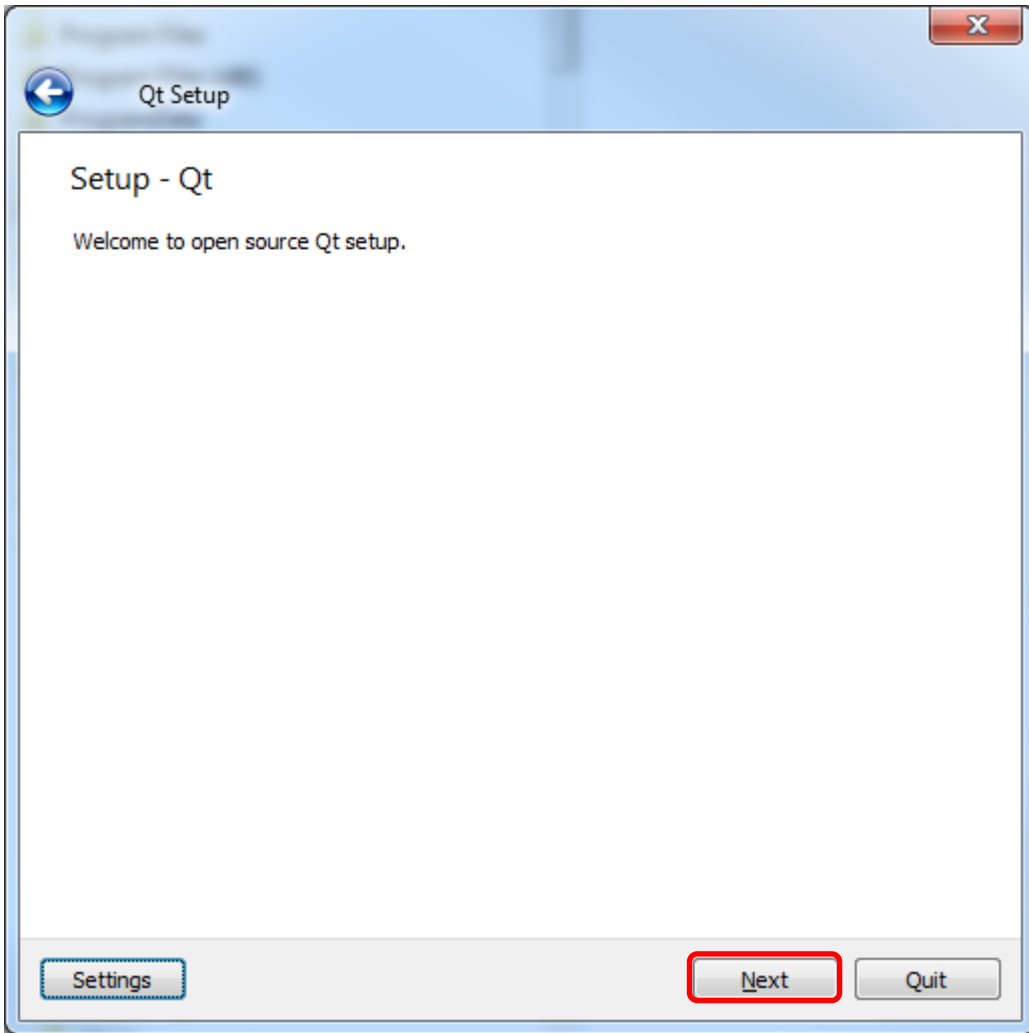
The "Please log in to Qt Account" section includes a "Login" label, an "Email" input field, a "Password" input field, and a green link for "Forgot password?".

The "Need a Qt Account?" section includes a "Sign-up" label, an email input field containing "John.Employee@company.com", two password input fields (both masked with dots), and a checkbox labeled "I accept the [service terms](#)." which is checked.

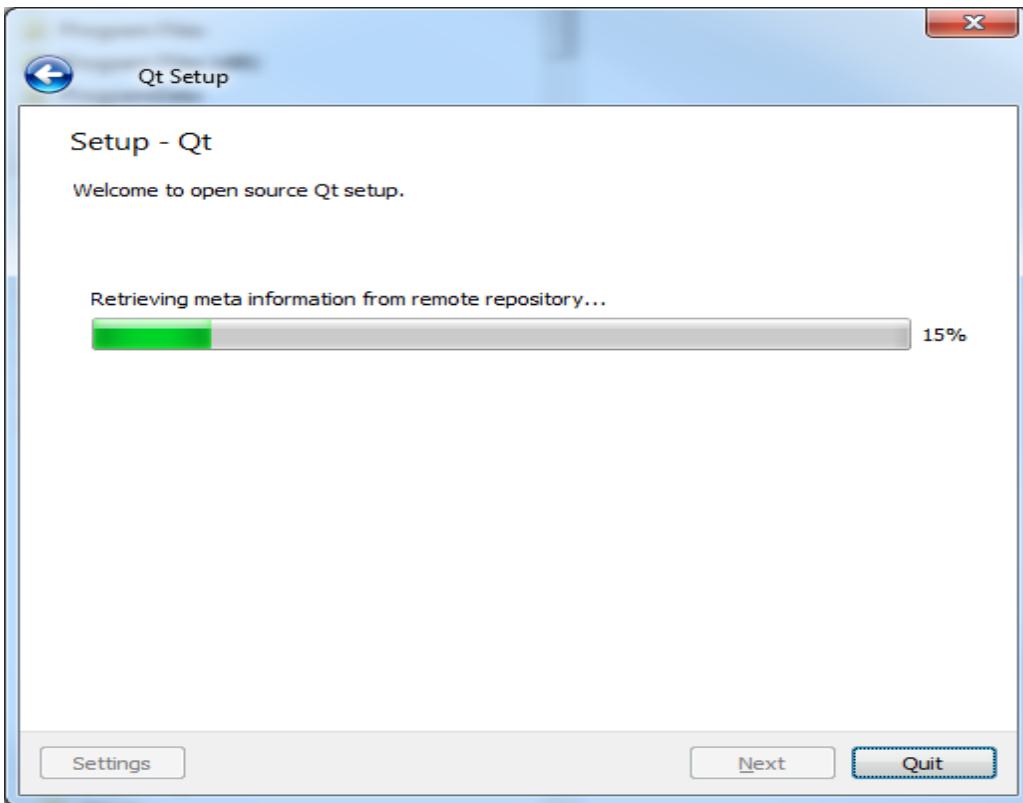
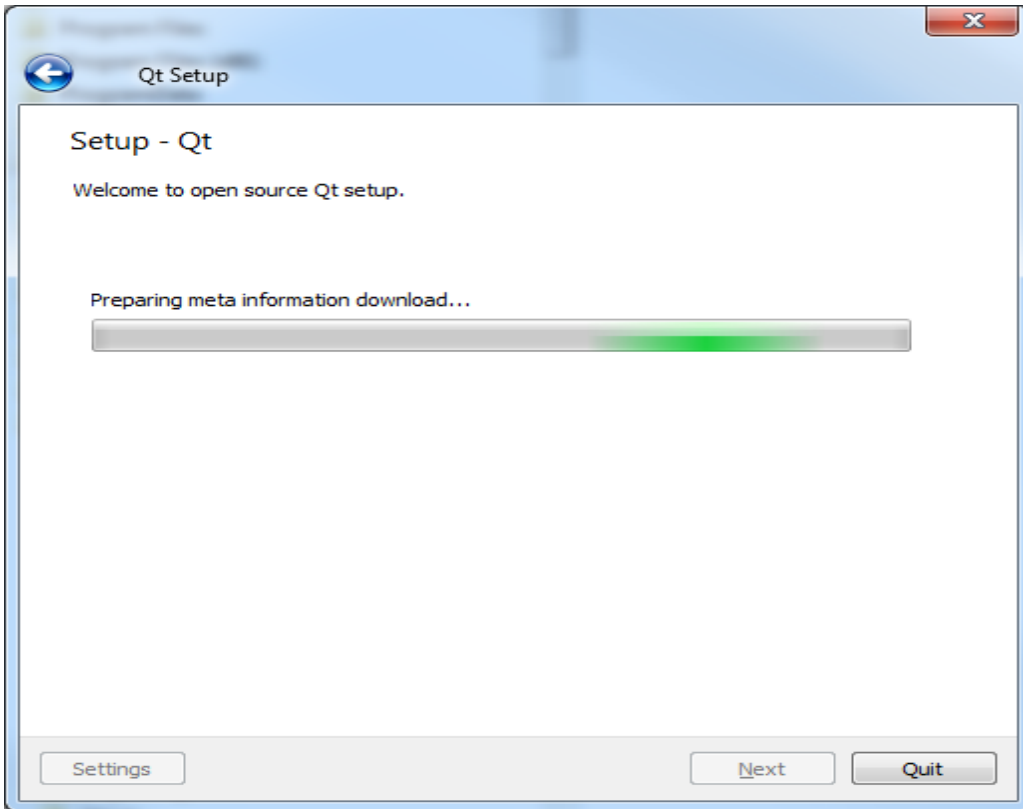
At the bottom of the dialog, there are three buttons: "Settings", "Next", and "Cancel". The "Next" button is highlighted with a red rectangular border.

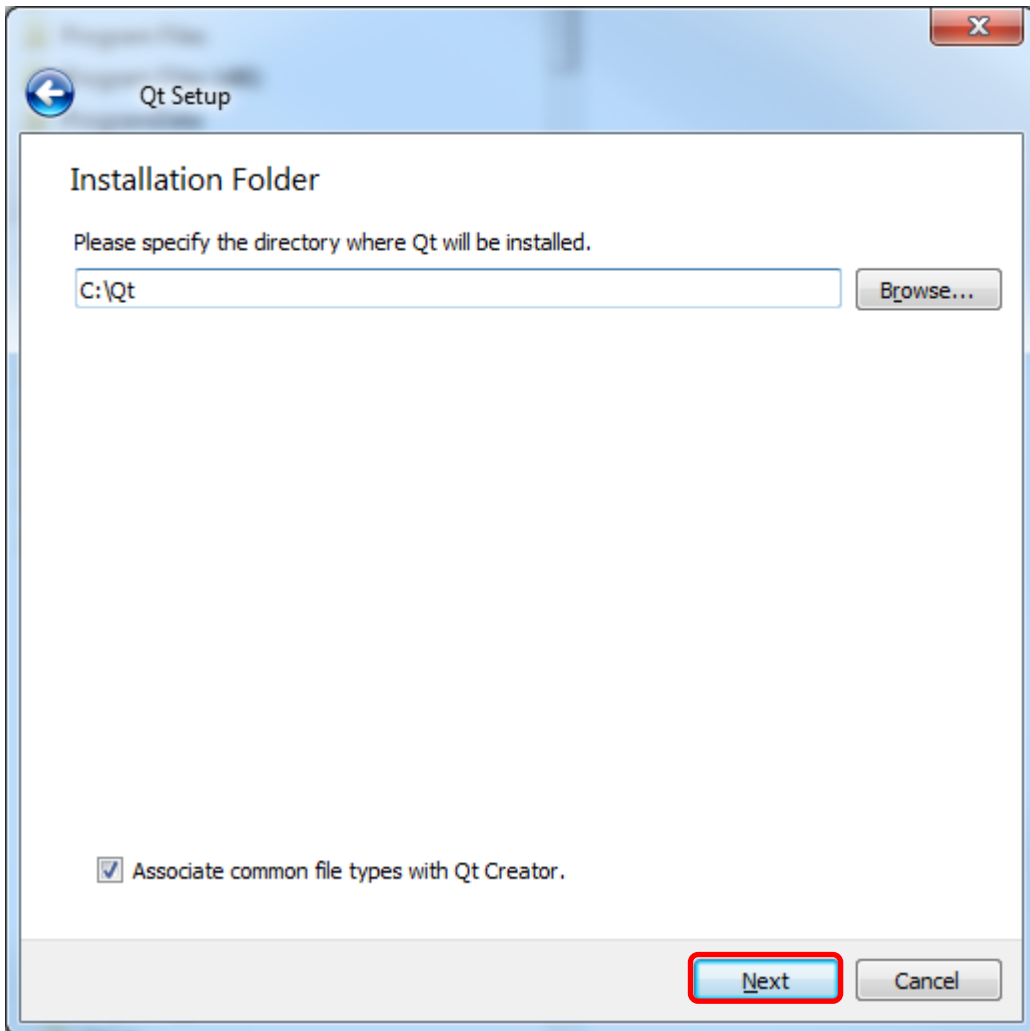
- If an account was created click “Next” – otherwise this screen will not appear

- Whether “Skip” or an account was created; installation continues here



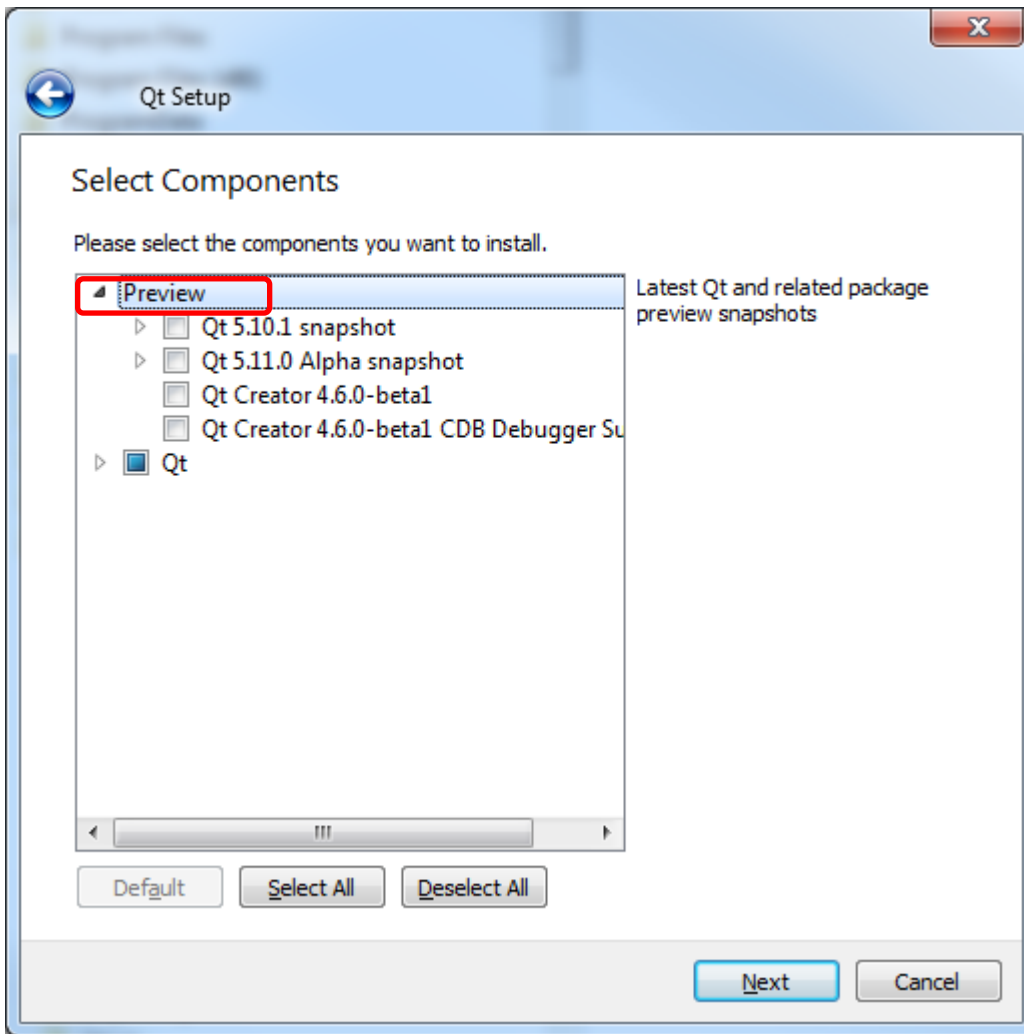
- Click “Next”





- Click “Next”

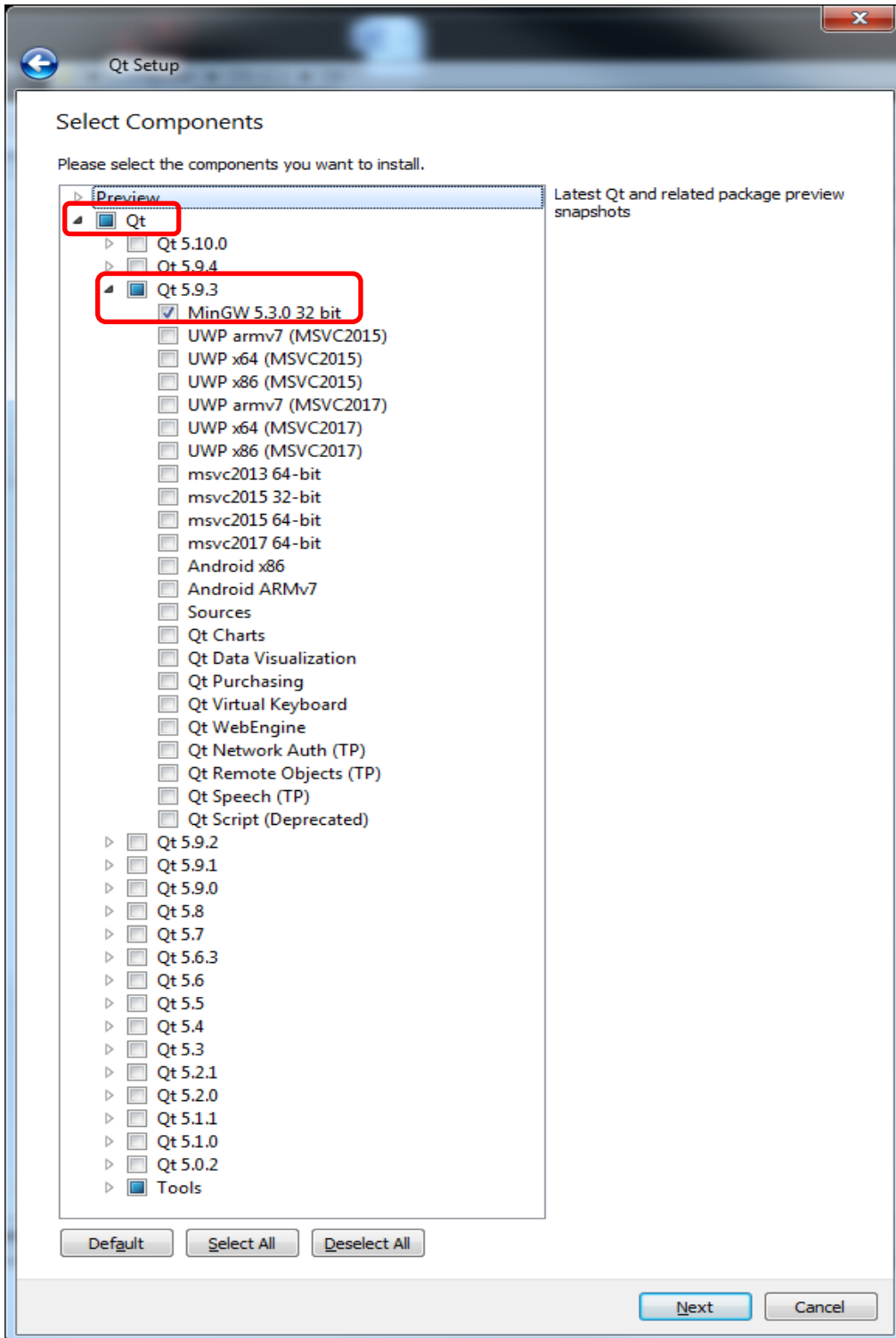
**Note: Due to the nature of Qt and the way it stores configuration information; Qt must be installed in C:\Qt.**



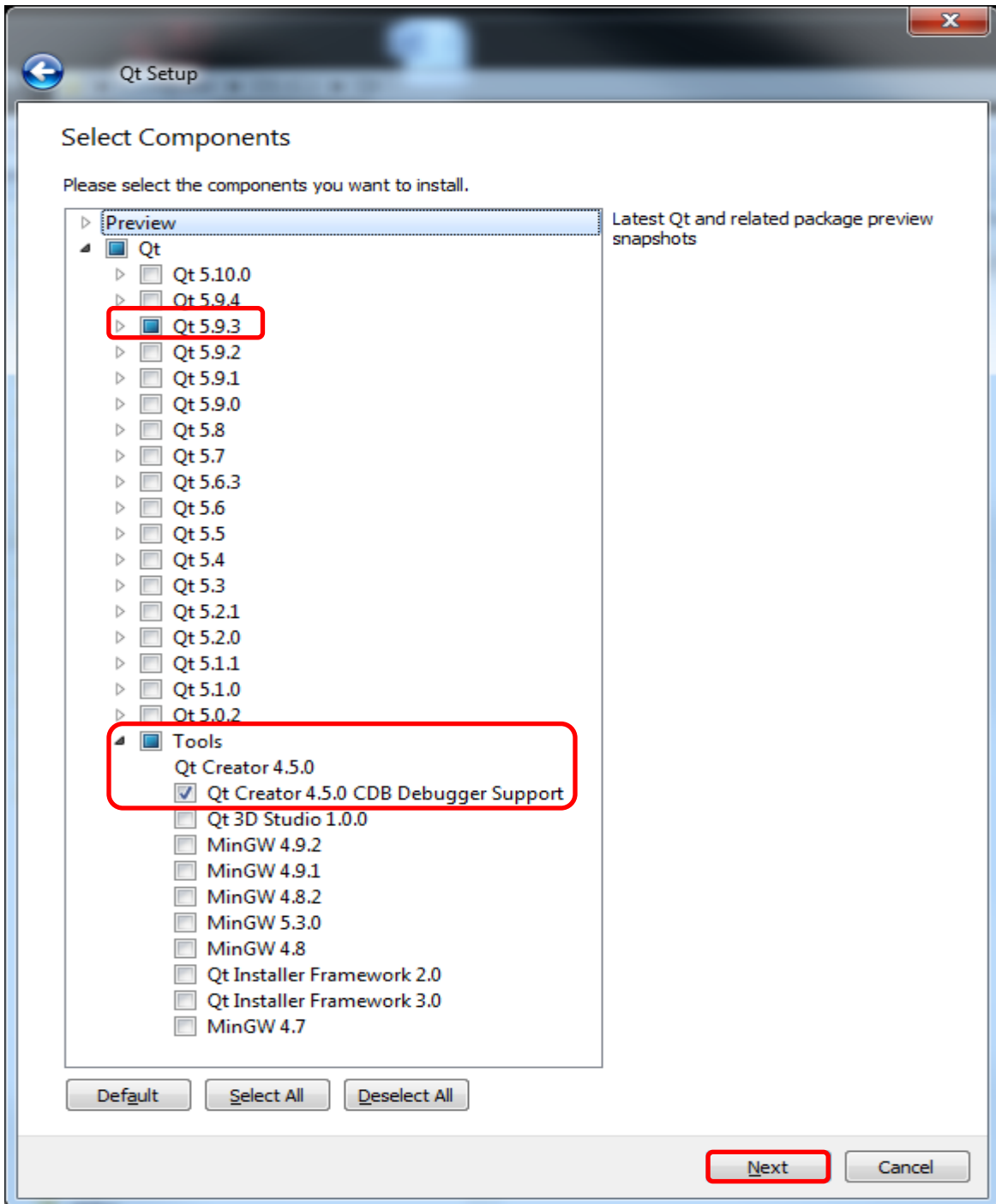
- Minimize Preview (do not select anything)



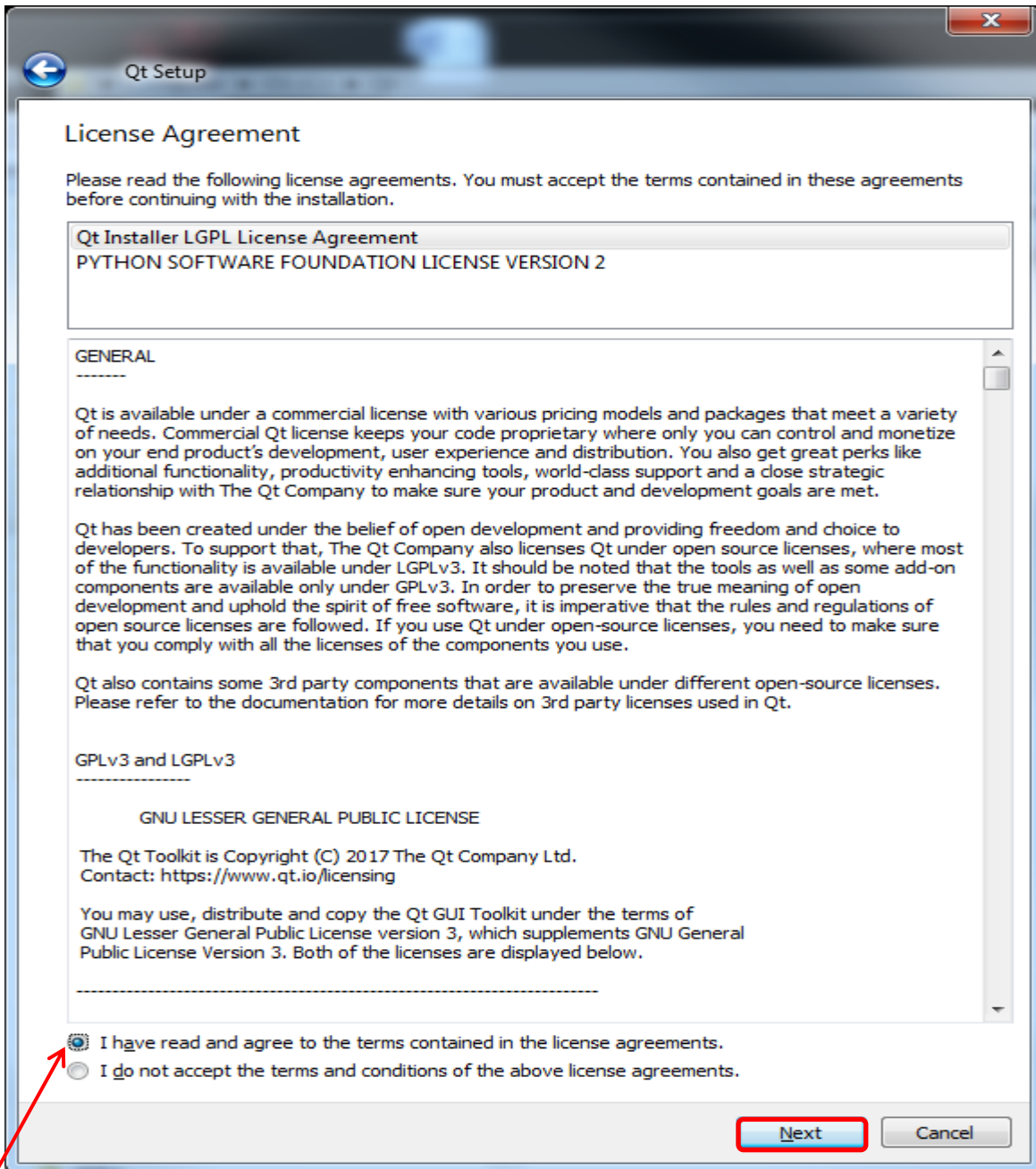
- Expand Qt → Qt 5.9.3
- Select “MinGW 5.3.0 32bit”



- Minimize Qt 5.9.3
- Expand Tools (Use the pre-selected default)

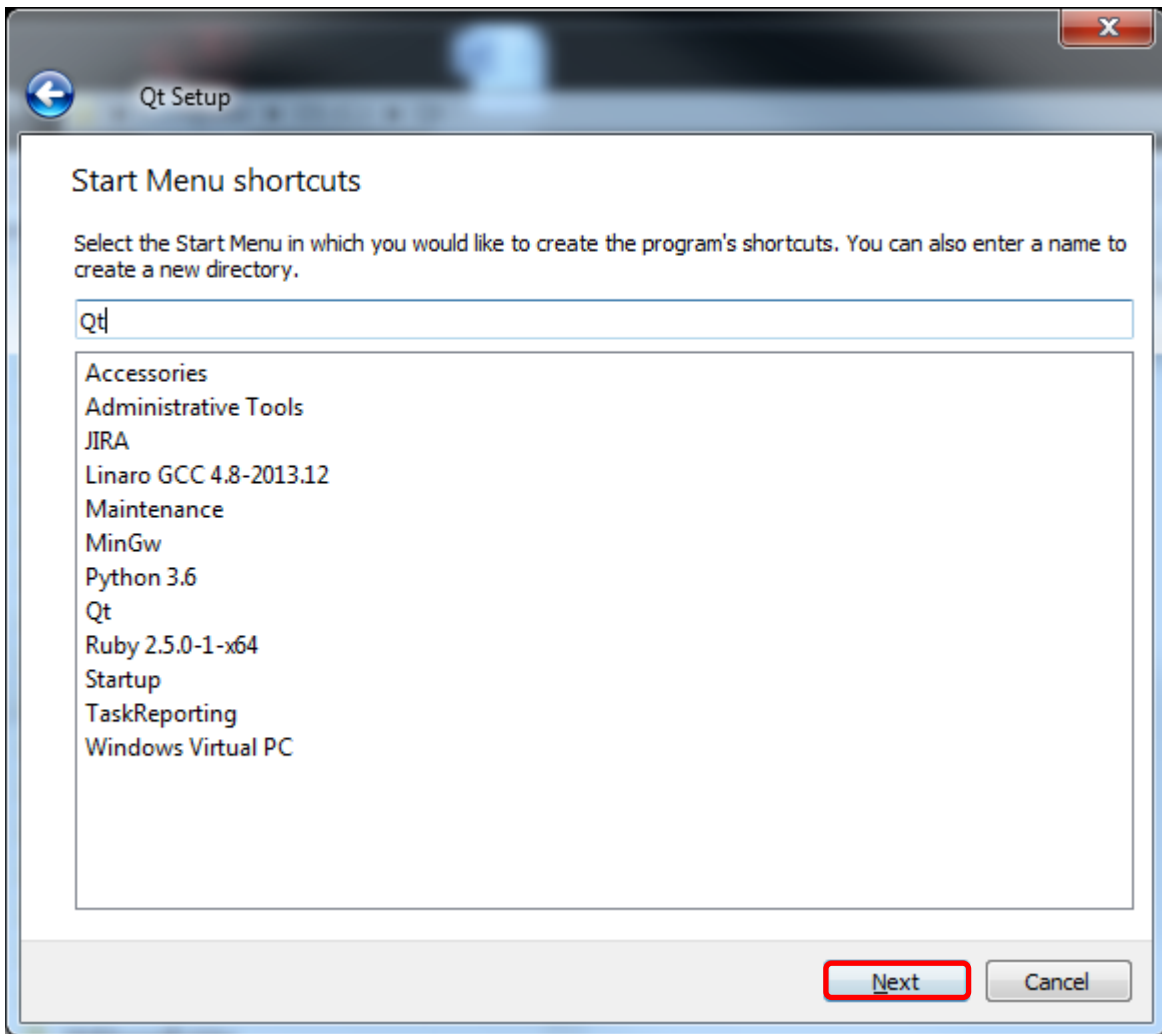


- Click Next

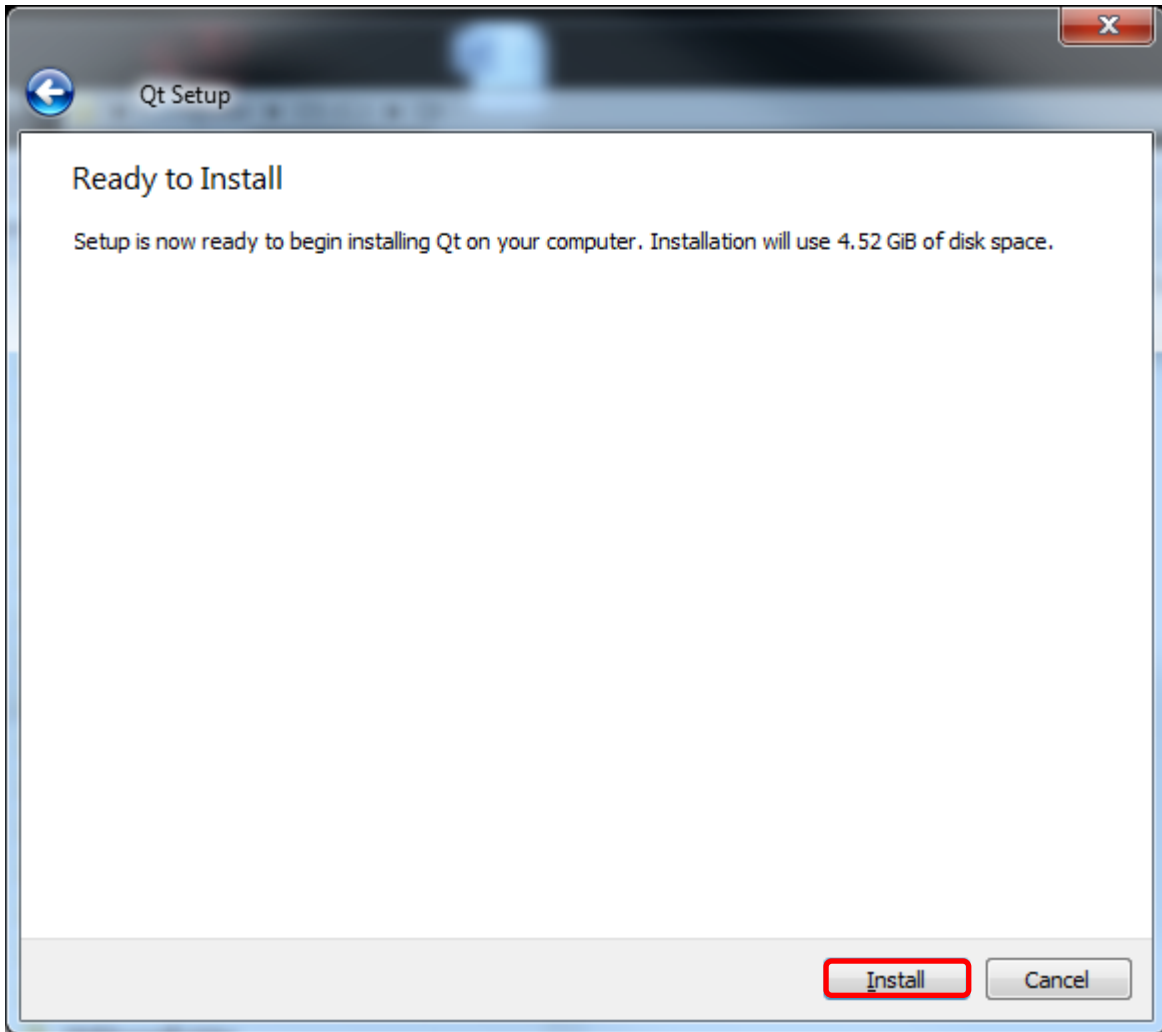


- If accepting of the license agreement select “I have read...”
- Click “Next”

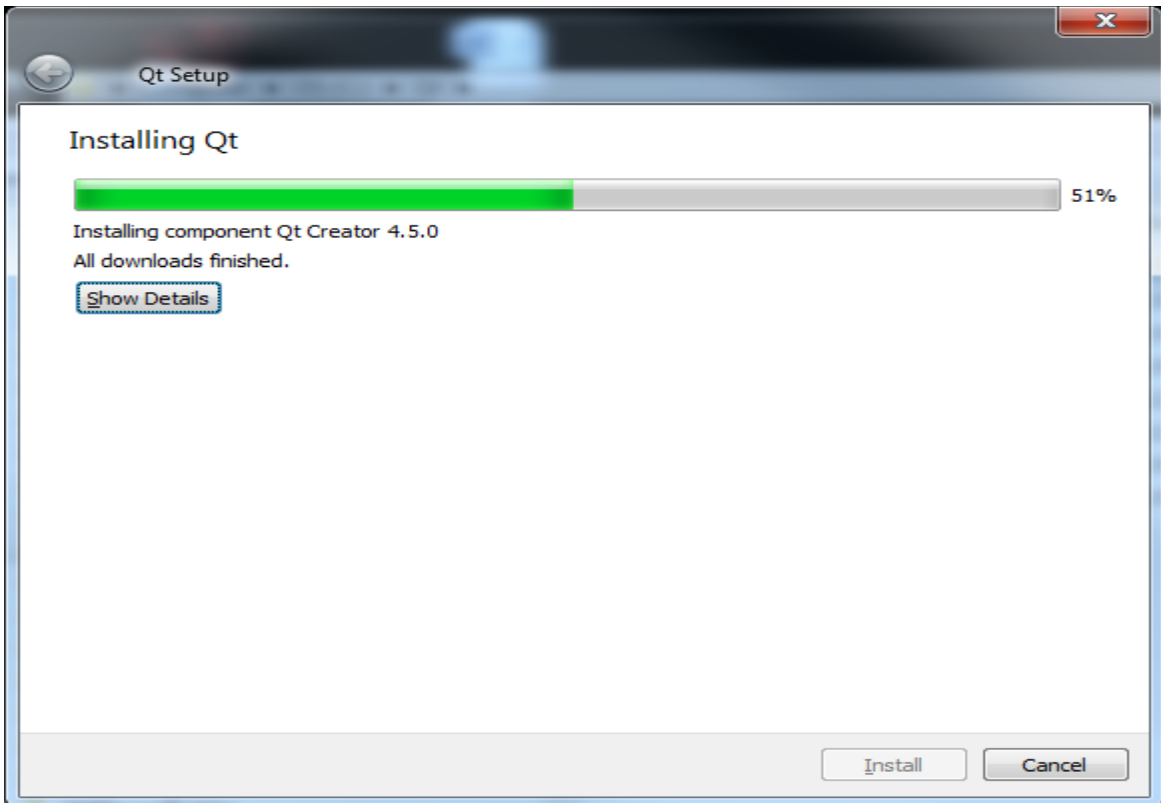
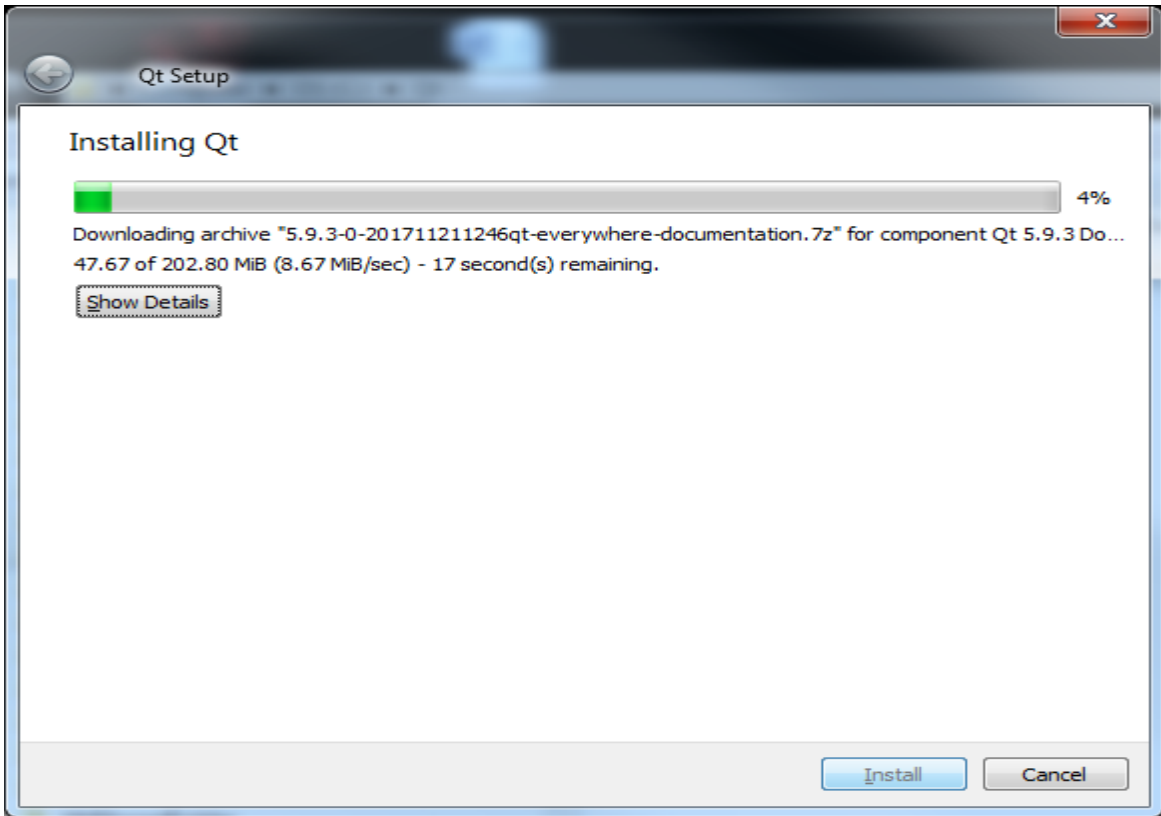
**Note:** Qt is licensed under the terms of LGPL and GPL; these are open-source licensing agreements. Please reference <https://www1.qt.io/qt-licensing-terms/> for a detailed explanation. Additional information is also located at <https://www.gnu.org/licenses/licenses.html>.

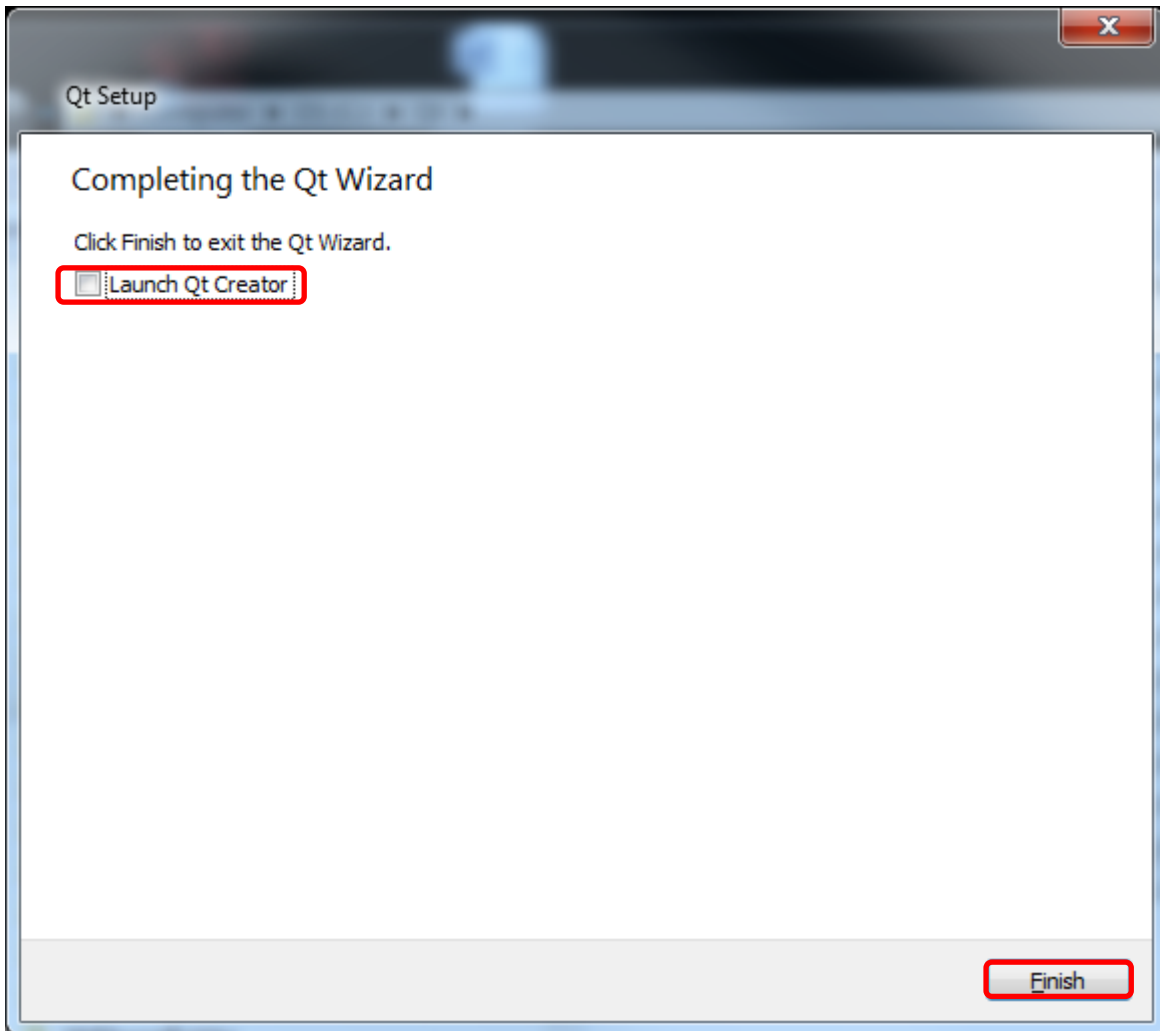


- Click “Next”



- Click “Install”





- Unselect “Launch Qt Creator”
- N.B. Qt Creator does **not** know the IP address of the target board at this time; the target board’s IP address will be discovered and configured later. Any time the IP address of the display changes, Qt Creator must be re-launched if using the /etc/hosts file for IP address resolution.
- Click “Finish”

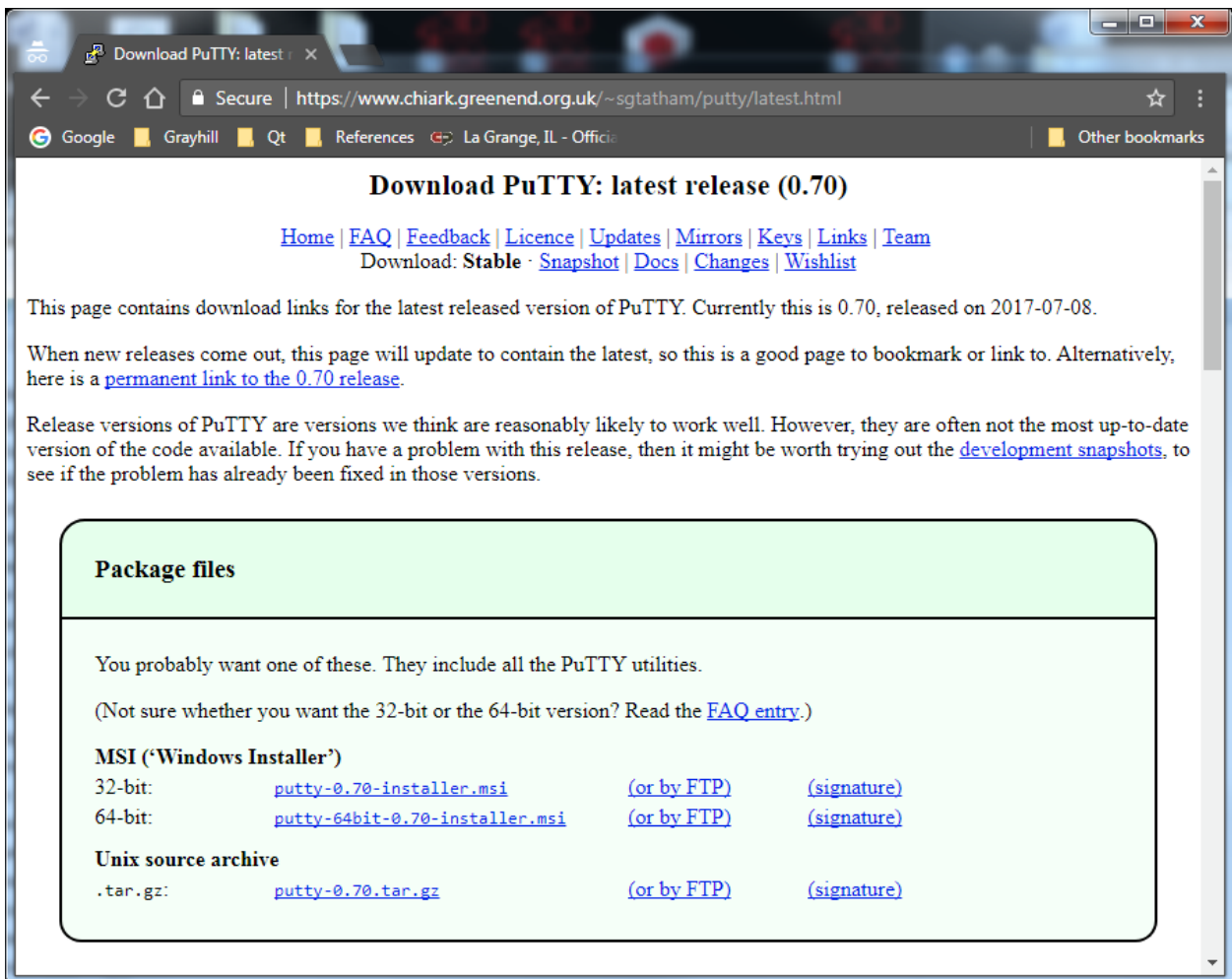
## Download and Install Support Files

This section details the downloading and installation of necessary support files.

### PuTTY

The examples shown in this document reflect the use of PuTTY. Feel free to substitute a different client.

- Navigate to <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html> and download the appropriate version



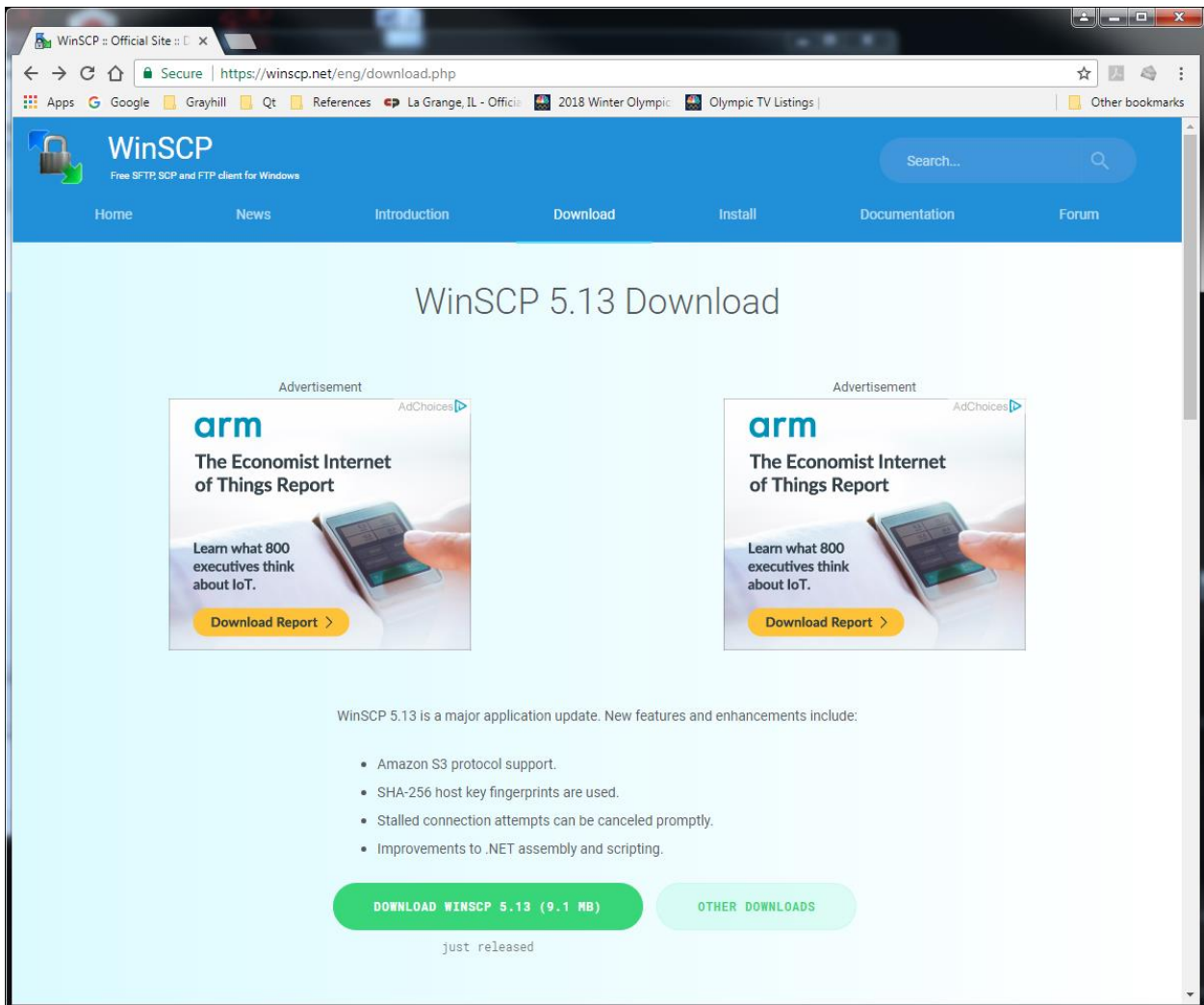
- Open the downloads folder and double click to execute the PuTTY installer
- Follow the installation instructions – connection configuration is described later on in the document



## WinSCP

The examples shown in this document reflect the use of WinSCP. Feel free to substitute a different utility.

- Navigate to <https://winscp.net/eng/download.php> and click “DOWNLOAD WINSCP...”



- Open the downloads folder and double click to execute the WinSCP installer
- Follow the installation instructions
- When installation is complete; select “Launch WinSCP” -- configuration is described later in the document after the IP address is discovered

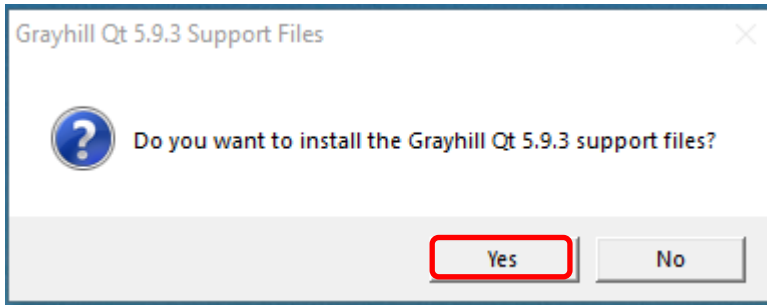
## Grayhill Qt Support Files

This section downloads and installs the necessary Qt support files. It also configures Qt Creator for the 3Dxx Display kit.

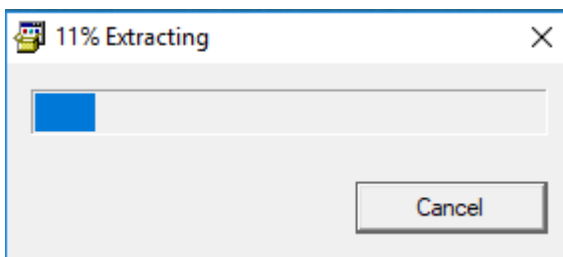
- Download “Qt Creator Windows Support Files” from the Grayhill website
- Open the download folder and double click on “QtGhSupport.exe”

A User Access Control window may pop-up

- Click “Yes” to allow the self-extracting zip file to proceed
- The following window appears



- Click “Yes”



```
C:\Windows\system32\cmd.exe

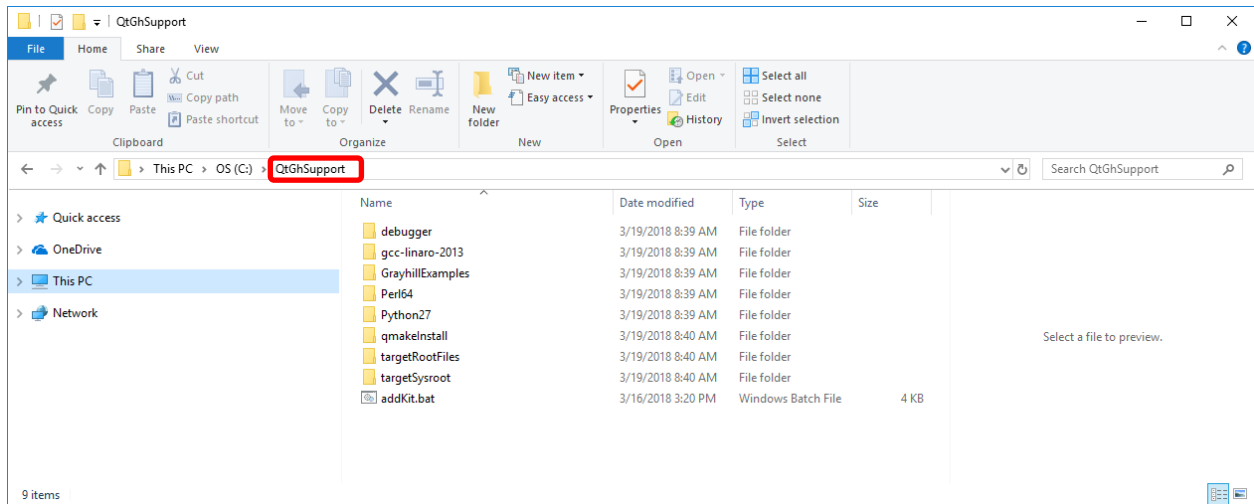
Installing Grayhill Qt 5.9.3 support files in C:\QtGhSupport ...
40578 File(s) copied

Creating Qt kit for Grayhill Display

Adding toolchain
Adding debugger
Adding qmake (Qt Version)
Adding display (target device)
Creating kit

Press any key to close...
```

- Using Windows Explorer; navigate to “C: QtGhSupport” and verify the folder was installed



## Configuring 3Dxx Display's IP Address

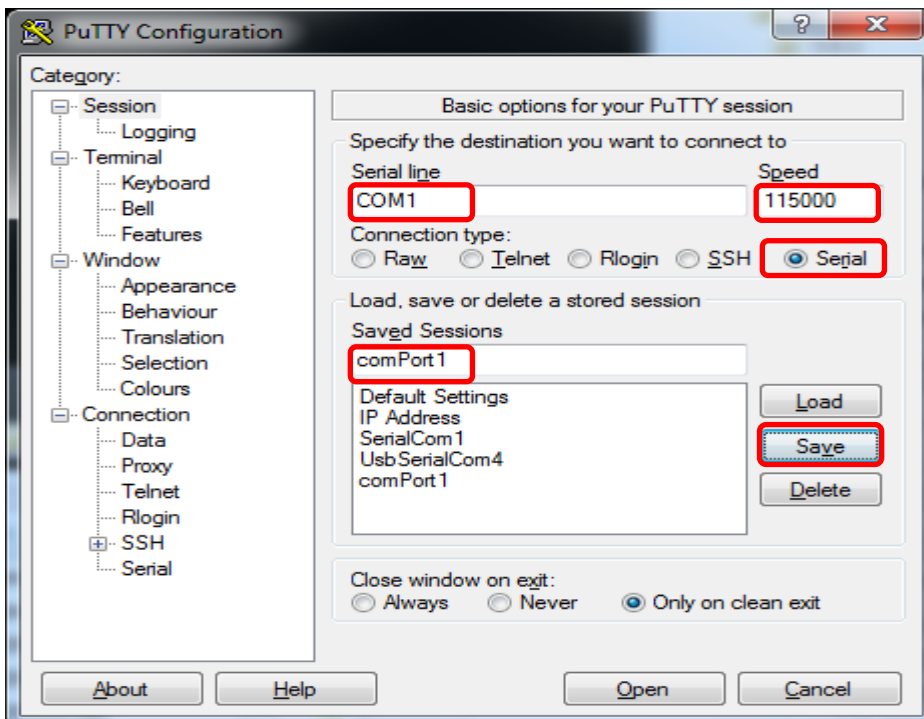
In order to complete the setup of the Qt development environment for the 3Dxx Display hardware; the IP address assigned to the 3Dxx Display must be determined.

In order to perform these tasks, it is necessary to connect the 3Dxx Display to the same network as the development PC.

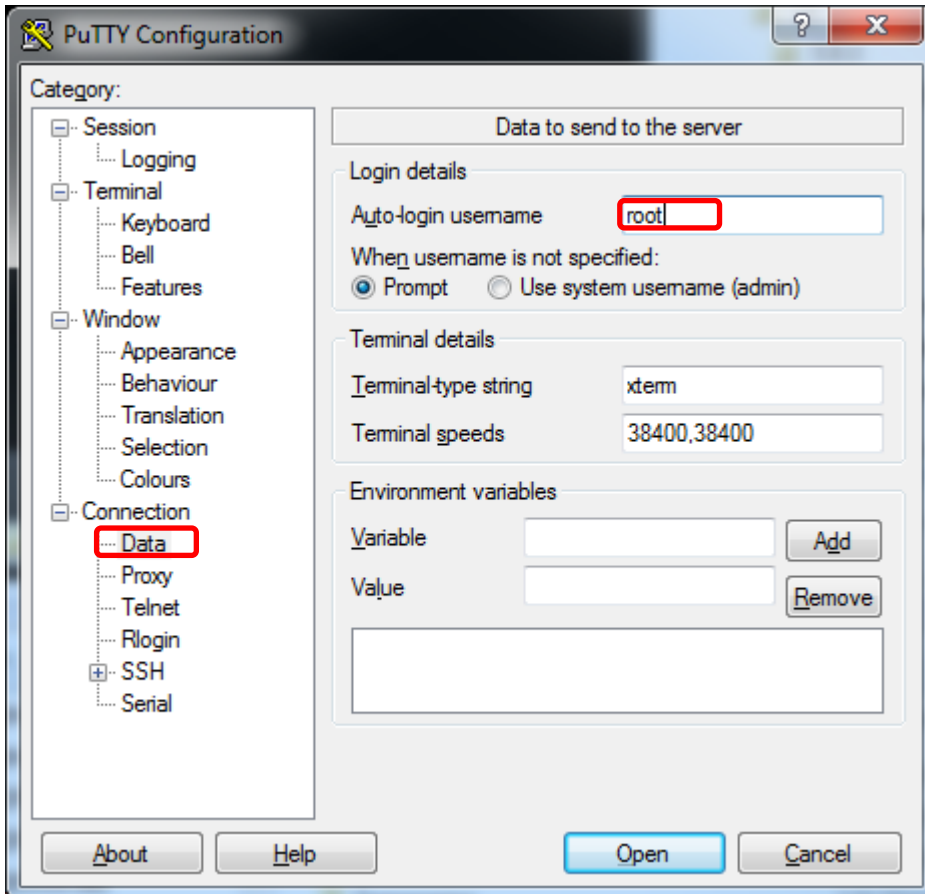
- Connect the 3Dxx Display serial port to a serial port on the development PC
- Determine the serial port device name to use for PuTTY (serial communication between the PC and the target). Usually, COM1 is used. (reference Device Manager → Ports if not certain)
- Launch PuTTY
- The PuTTY Configuration screen appears – configure as follows:

Select the “Serial” button  
Set “Serial line” to appropriate COM Port  
Change the “Speed” to 115000  
Enter a name in “Saved Sessions” (e.g. comPort1)  
Click “Save”

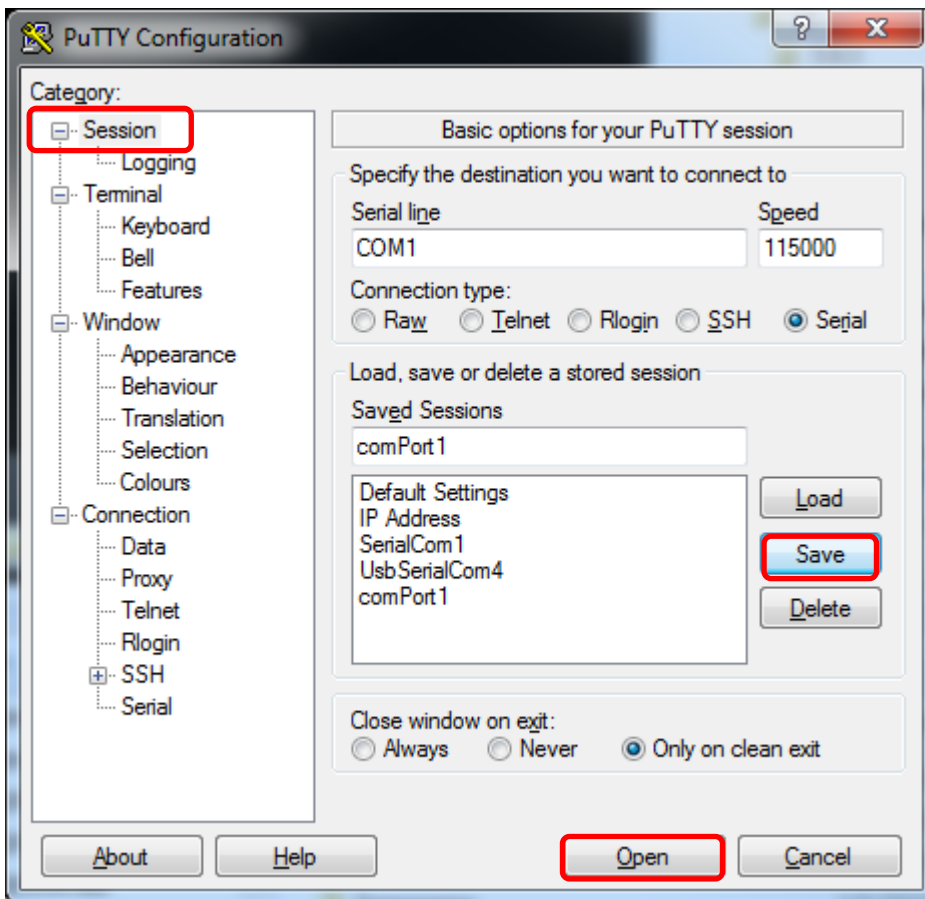
**N.B.** If “Open” is clicked any unsaved configuration modifications are **lost!**



- Click on “Data”
- Set “Auto-login username” to “root”

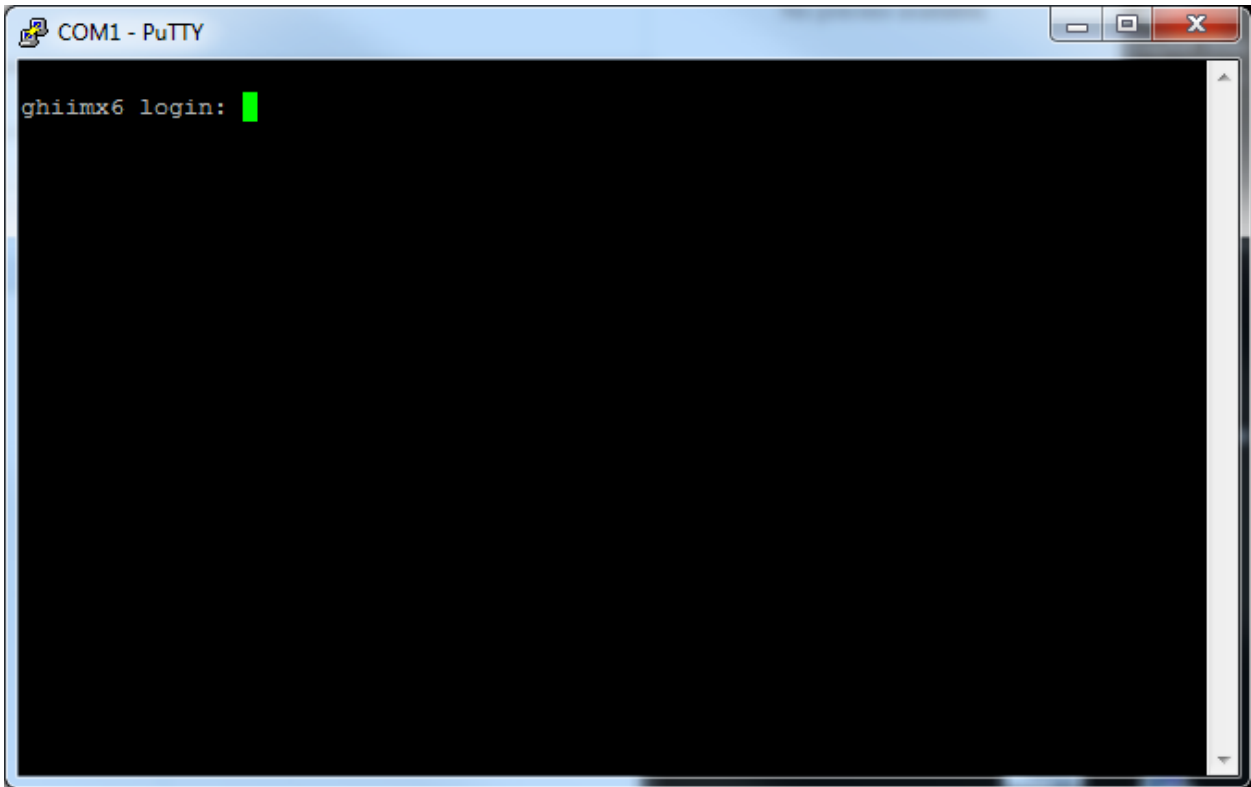


- Click back on “Session”, then click “Save” again



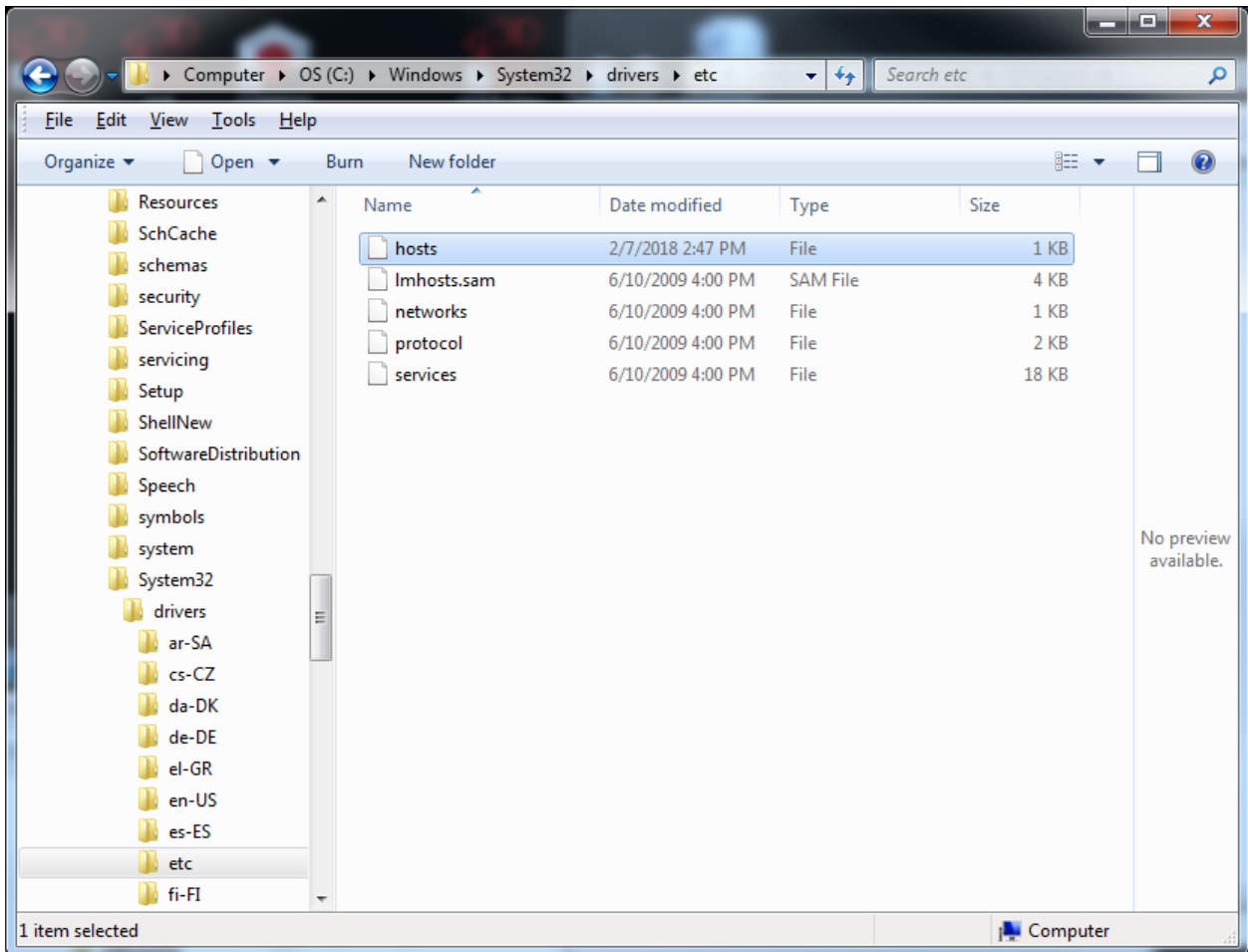
- Lastly, click “Open” to establish a connection

- Make sure that the 3Dxx Display is powered up and press the “Enter” key.



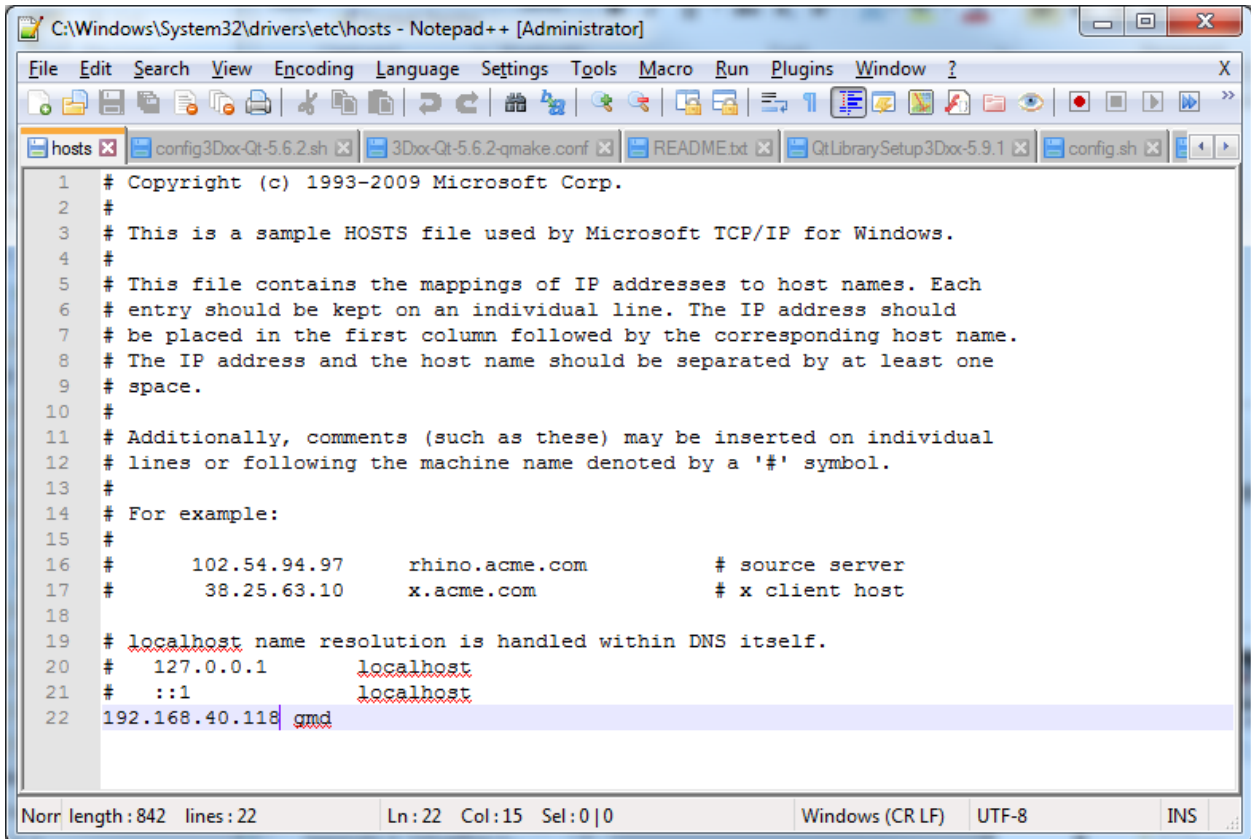
- A “ghiimx6 login:” prompt should appear. If the 3Dxx Display was just powered up; startup messages may appear as well, but when they are done, pressing the “Enter” key should produce a “ghiimx6 login:” prompt as shown.
- At the “ghiimx6 login:” prompt enter “root” (no password is required).
- Depending on the IP address type, refer to the appropriate appendix:
  - Dynamic      Appendix J: Dynamic IP Address
  - Static        Appendix K: Static IP Address

- Open Windows Explorer window (<Window>-e)
- Navigate to C: → Windows → System32 → drivers → etc and select “hosts”





- Right click to edit the file using your favorite flavor of editor (Screenshot illustrates Notepad++)
- After the editor is launched, Windows Explorer can be closed
- Add the IP address and “gmd” as illustrated below:



```
1 # Copyright (c) 1993-2009 Microsoft Corp.
2 #
3 # This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
4 #
5 # This file contains the mappings of IP addresses to host names. Each
6 # entry should be kept on an individual line. The IP address should
7 # be placed in the first column followed by the corresponding host name.
8 # The IP address and the host name should be separated by at least one
9 # space.
10 #
11 # Additionally, comments (such as these) may be inserted on individual
12 # lines or following the machine name denoted by a '#' symbol.
13 #
14 # For example:
15 #
16 #       102.54.94.97       rhino.acme.com           # source server
17 #       38.25.63.10       x.acme.com                # x client host
18
19 # localhost name resolution is handled within DNS itself.
20 #   127.0.0.1           localhost
21 #   ::1                 localhost
22 192.168.40.118| gmd
```

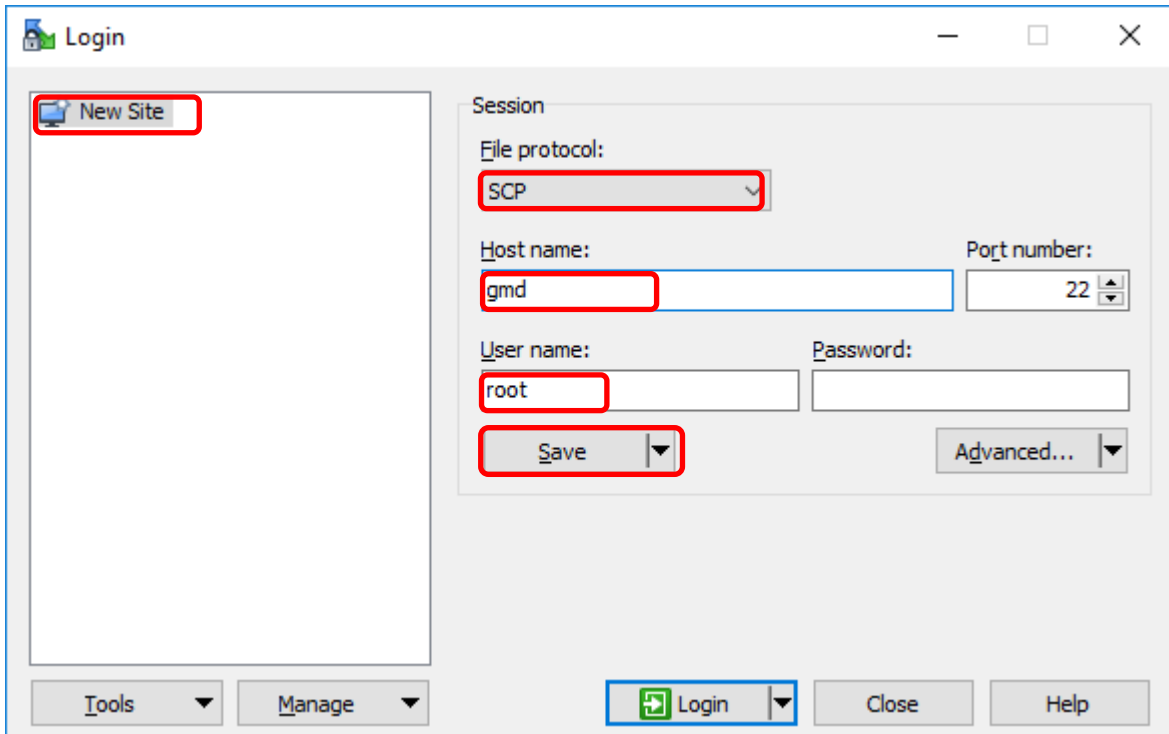
- Save the file

N.B. The editor may ask to restart in admin mode; allow it to continue as *hosts* is a system file

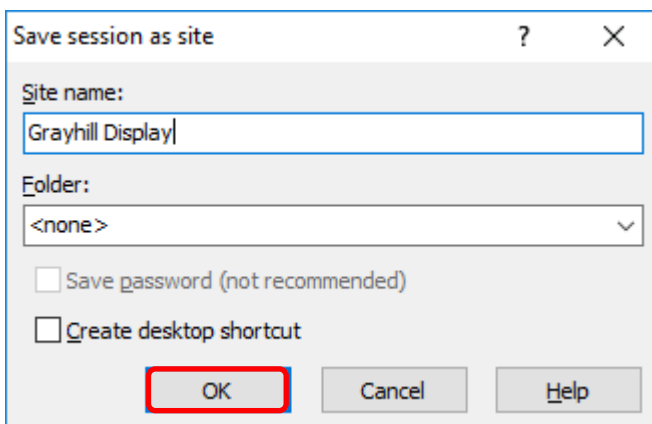
**N.B. If the IP address of the display changes; hosts must be updated and Qt Creator re-launched**

## Transfer Configuration Files to Display

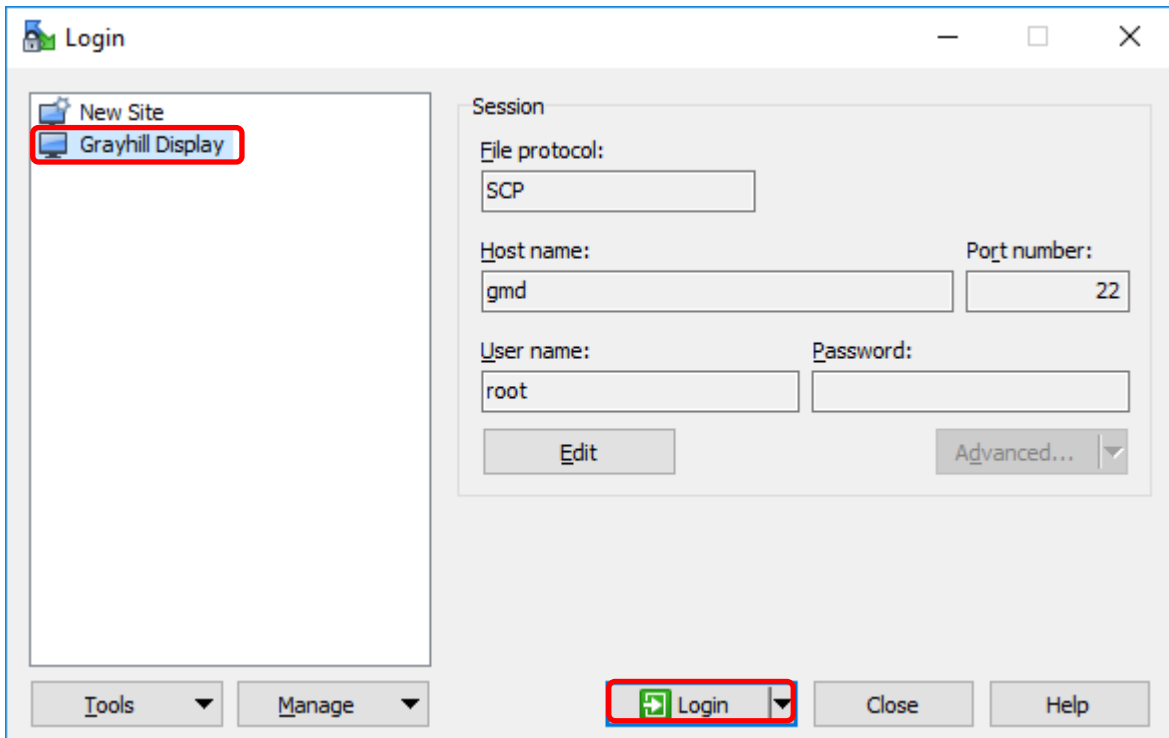
- Return to the WinSCP window and establish a login session and connection
- Select “New Site” and configure as follows:
  - File protocol SCP
  - Host name gmd or <IP address>
  - User name root



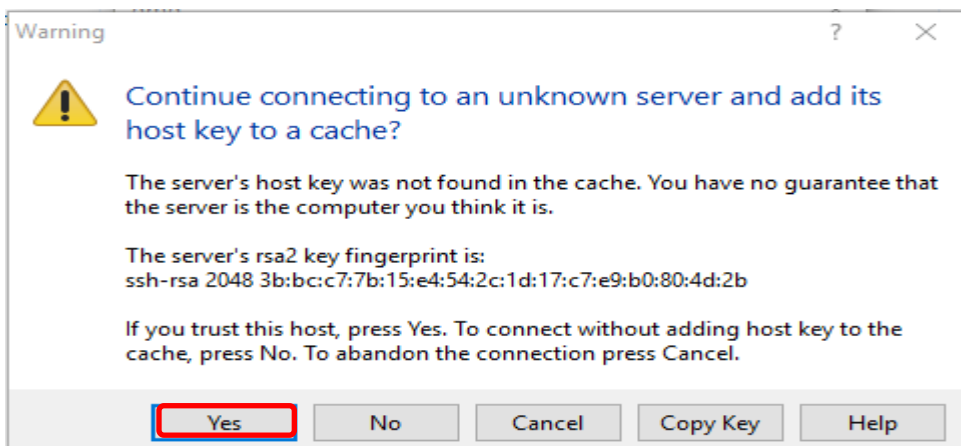
- Click Save and enter a name



- Click “OK”

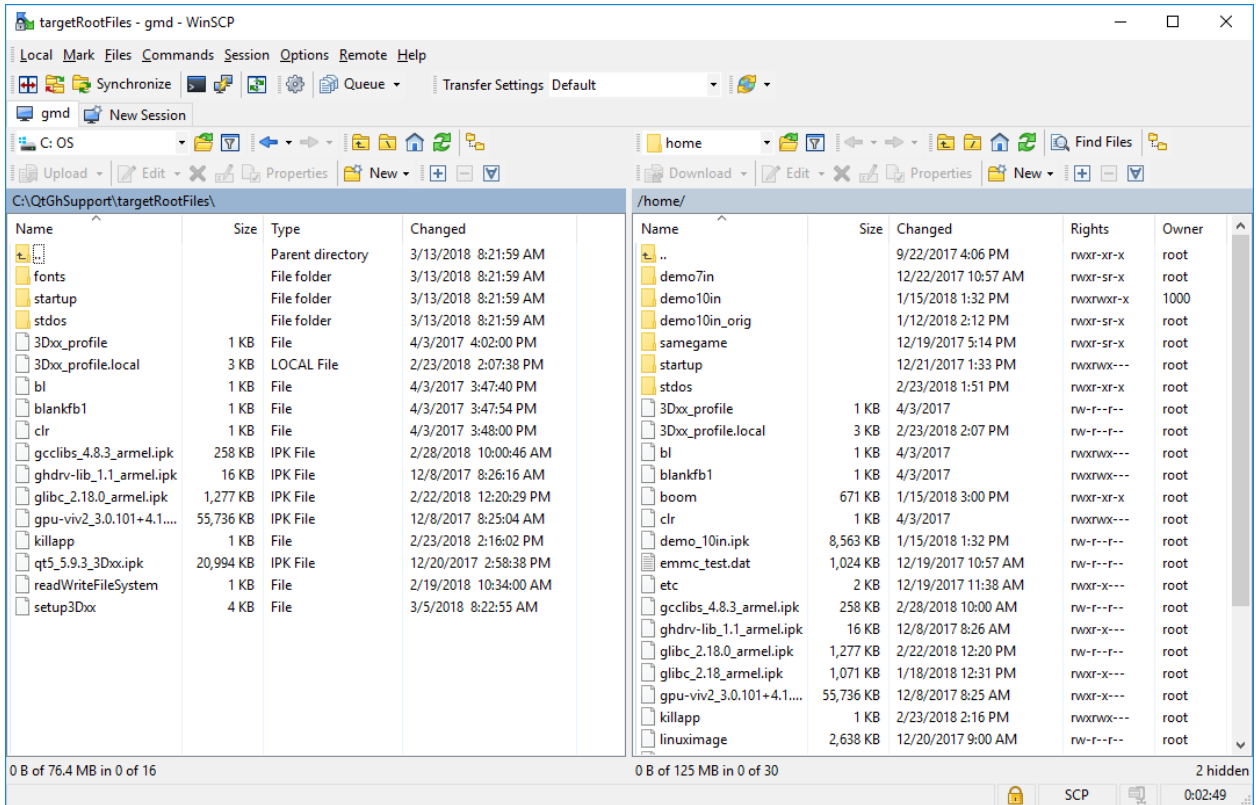


- Select “Grayhill Display”
- Click “Login”
- If this is the first connection to this IP address, the following will pop-up

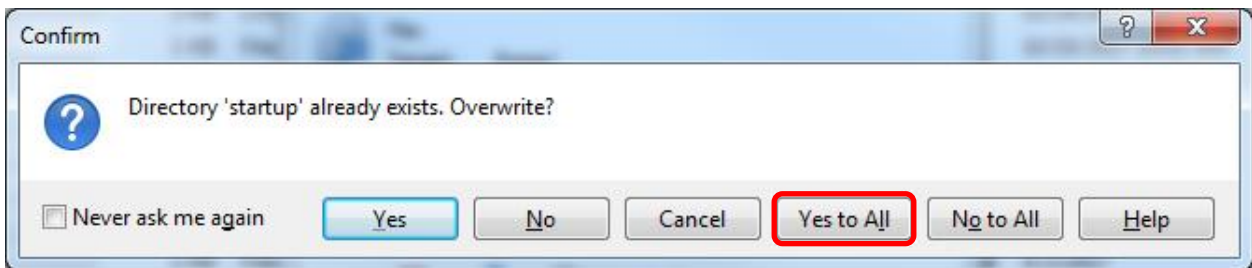


- Click “Yes”
- In the left pane Navigate to C:\QtGhSupport\targetRootFiles  
Hint: Clicking on the “C” goes directly to that directory level

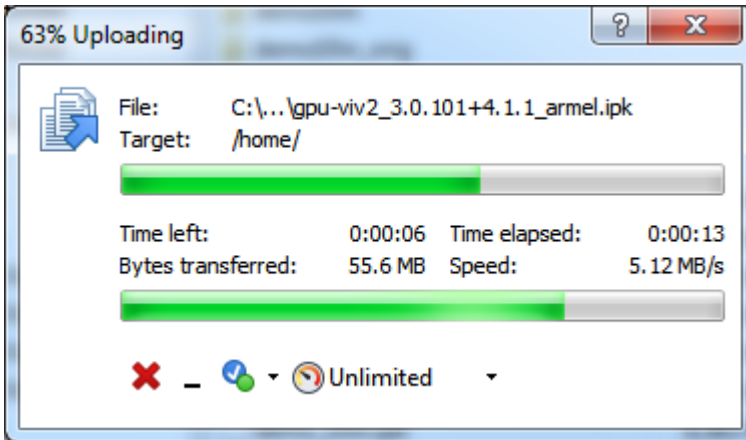
- The right pane defaults to /home on the display; the display may already have files



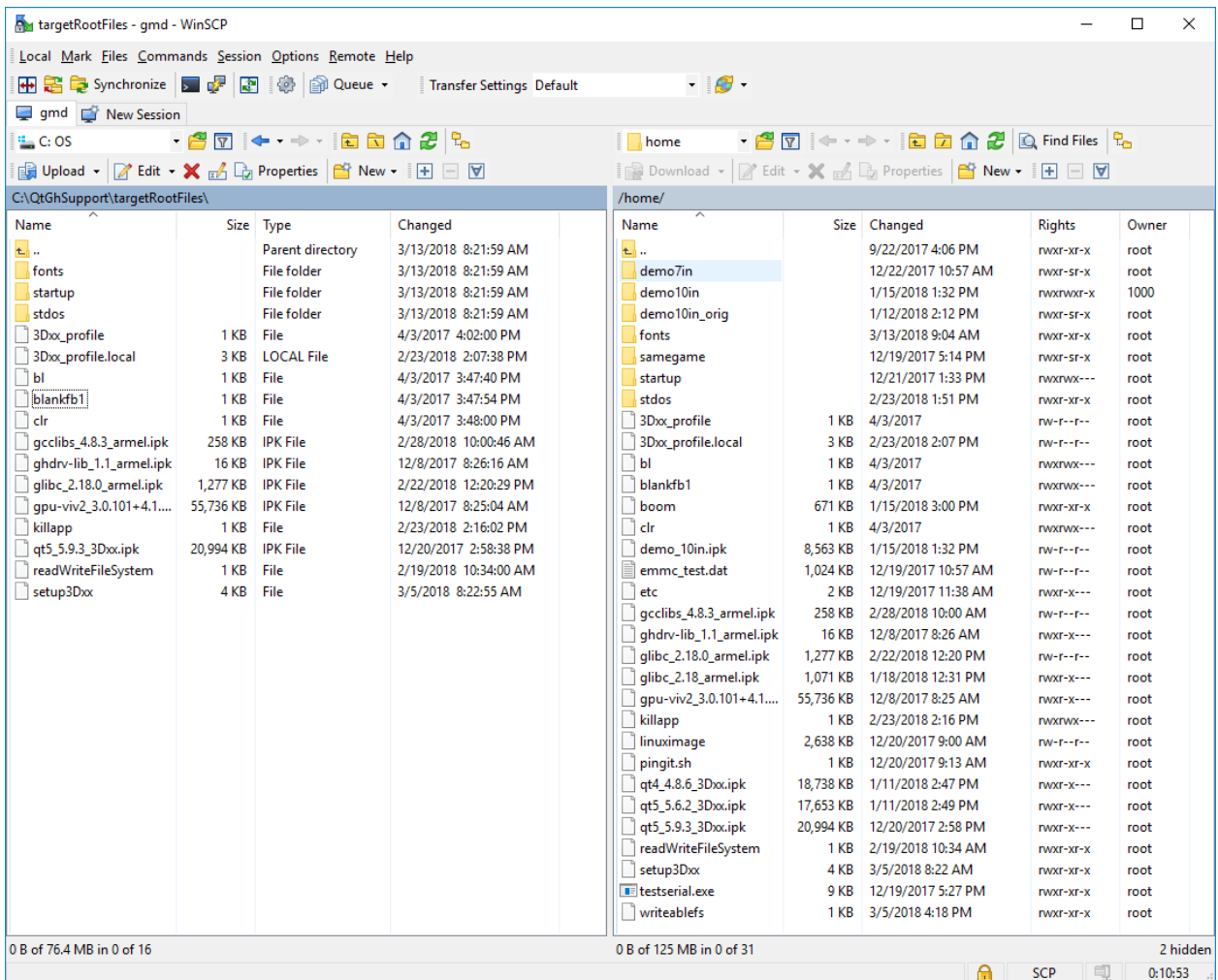
- In the left hand pane, select all the files (<Ctrl>-a) and drag them to the target (right hand pane)
- If some files already exist overwrite them with the new ones



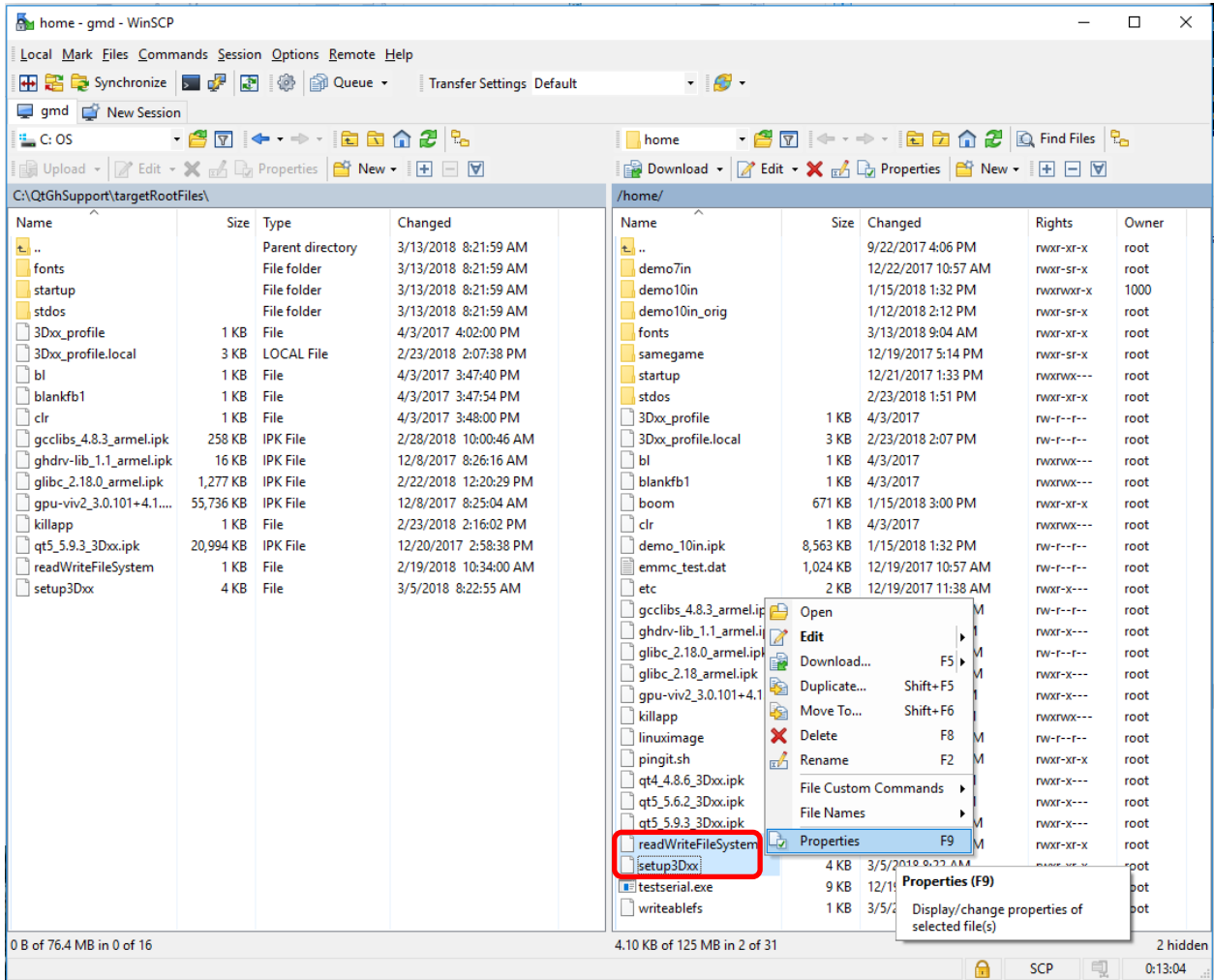
- Click “Yes to All”



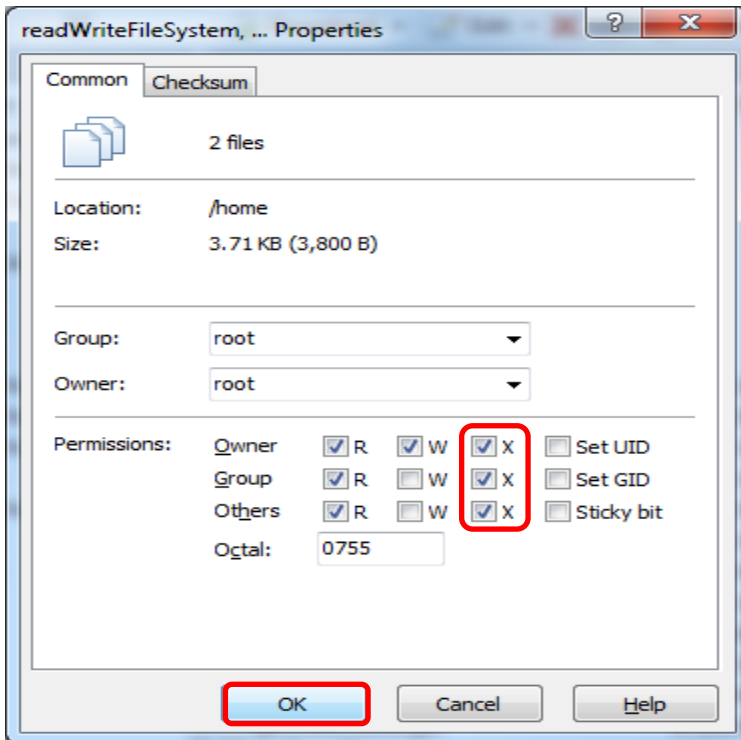
- The right pane should look similar to:



- Select “readWriteFileSystem” and “setup3Dxx” (<Ctrl> click)
- Right click → Properties <F9>



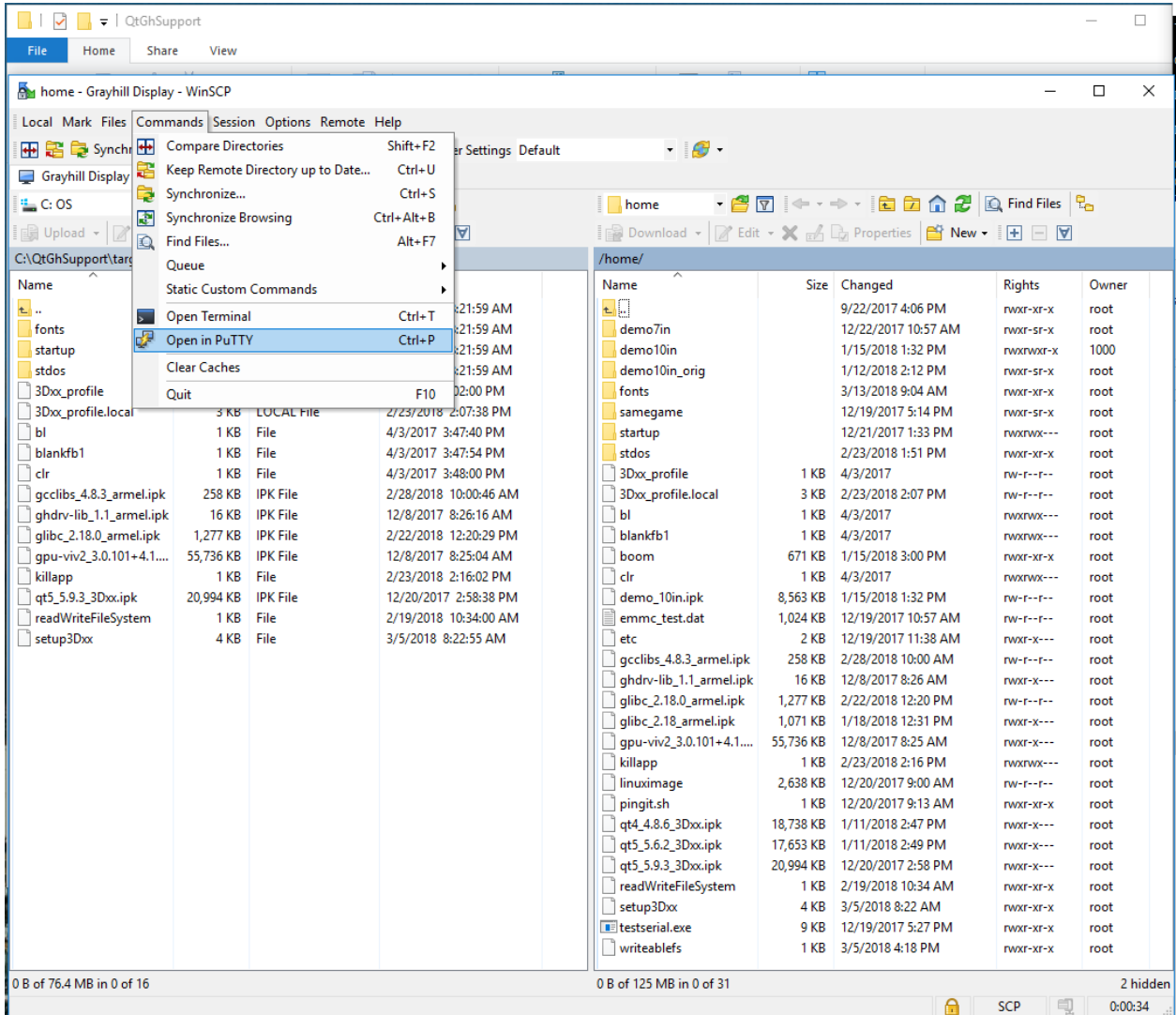
- Make them executable by selecting all the “X” boxes (several clicks may be required to cycle back to the check mark)



- Click “OK”

## Execute Configuration Scripts

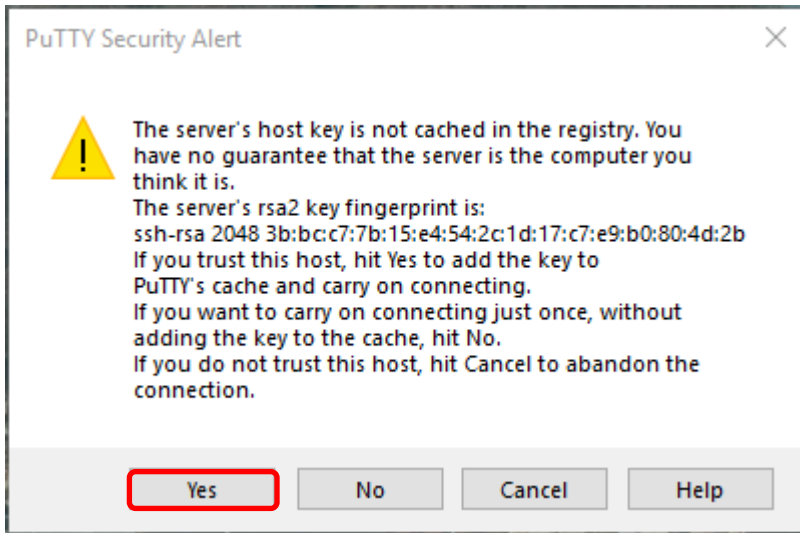
- From Commands select “Open in PuTTY”



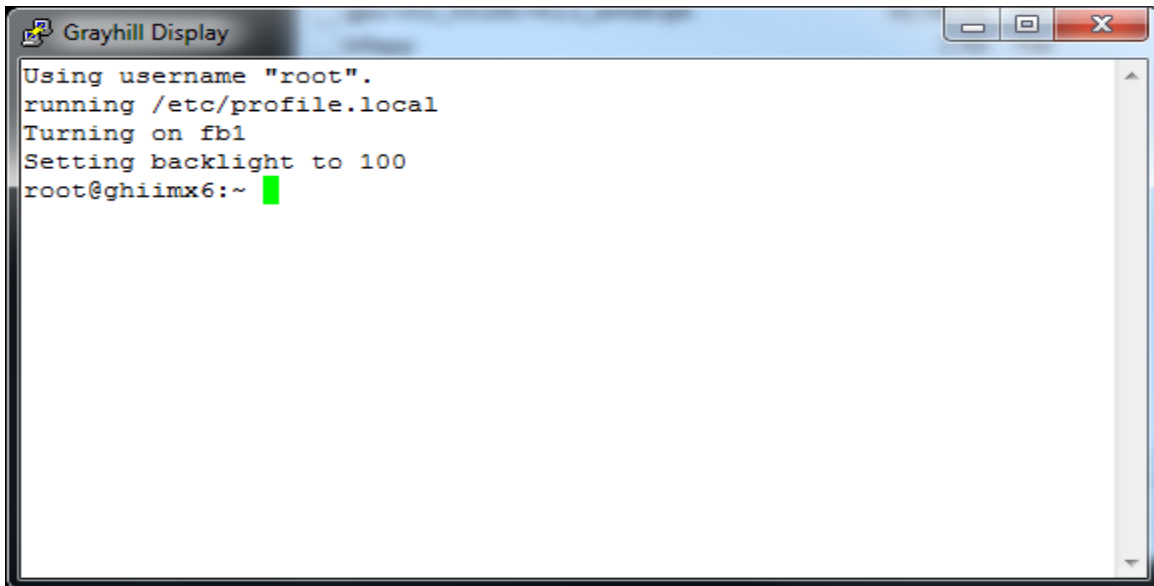
- A PuTTY session is established (via the IP address as opposed to the initial serial based session used to derive the IP address)



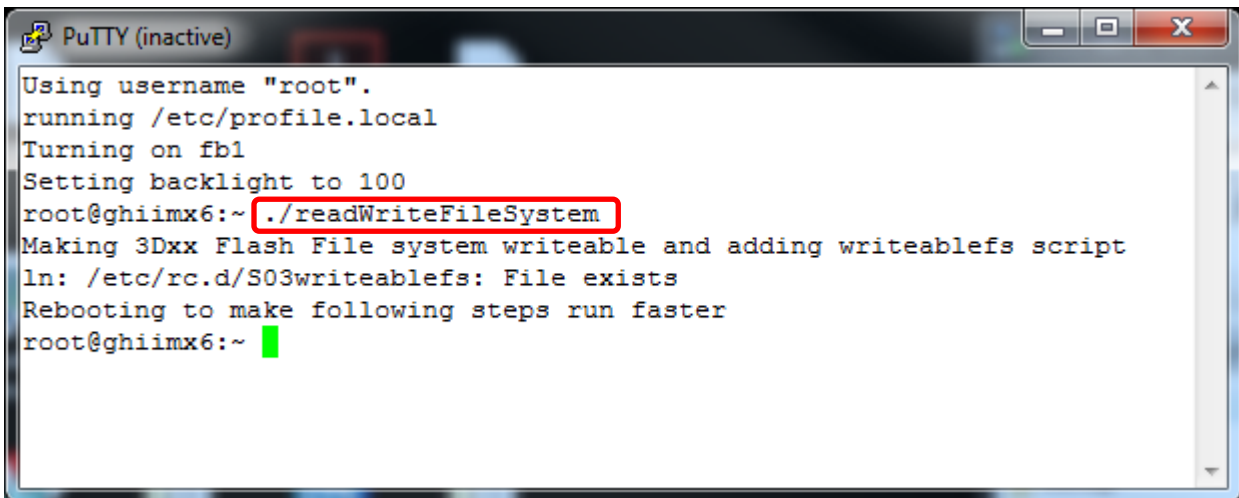
- If this is the first connection to this IP address, the following will pop-up



- Click "Yes"

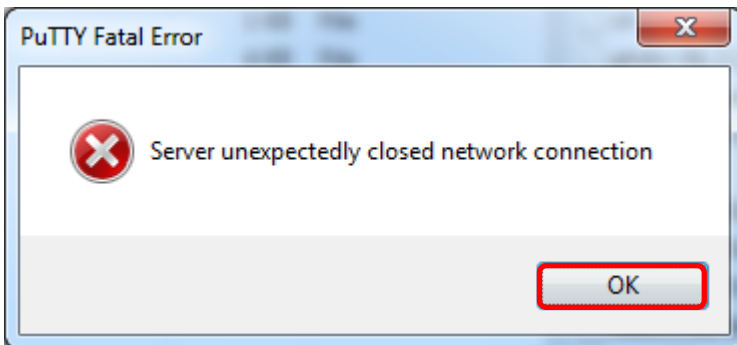


- Execute the script
  - **./readWriteFileSystem**



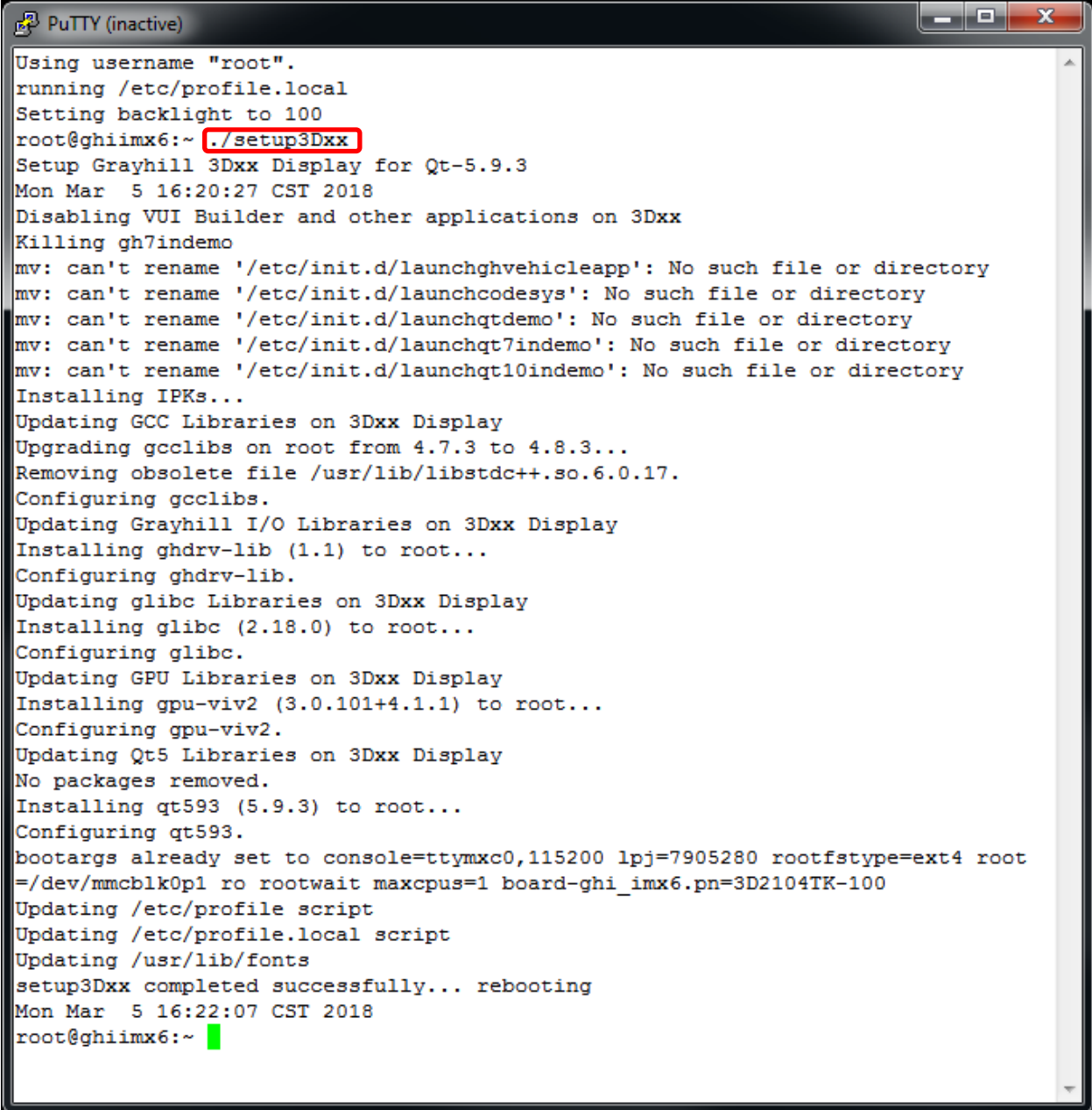
```
PuTTY (inactive)
Using username "root".
running /etc/profile.local
Turning on fb1
Setting backlight to 100
root@ghiimx6:~ ./readWriteFileSystem
Making 3Dxx Flash File system writeable and adding writeablefs script
ln: /etc/rc.d/S03writeablefs: File exists
Rebooting to make following steps run faster
root@ghiimx6:~ █
```

- The display will reboot, which terminates the PuTTY session



- Click “OK”
- Close the “PuTTY (inactive)” window
- Relaunch PuTTY (<Ctrl>-p)

- Execute the setup script
  - `./setup3Dxx`



```
PuTTY (inactive)
Using username "root".
running /etc/profile.local
Setting backlight to 100
root@ghiimx6:~ ./setup3Dxx
Setup Grayhill 3Dxx Display for Qt-5.9.3
Mon Mar  5 16:20:27 CST 2018
Disabling VUI Builder and other applications on 3Dxx
Killing gh7indemo
mv: can't rename '/etc/init.d/launchghvehicleapp': No such file or directory
mv: can't rename '/etc/init.d/launchcodesys': No such file or directory
mv: can't rename '/etc/init.d/launchqtdemo': No such file or directory
mv: can't rename '/etc/init.d/launchqt7indemo': No such file or directory
mv: can't rename '/etc/init.d/launchqt10indemo': No such file or directory
Installing IPKs...
Updating GCC Libraries on 3Dxx Display
Upgrading gcclibs on root from 4.7.3 to 4.8.3...
Removing obsolete file /usr/lib/libstdc++.so.6.0.17.
Configuring gcclibs.
Updating Grayhill I/O Libraries on 3Dxx Display
Installing ghdrv-lib (1.1) to root...
Configuring ghdrv-lib.
Updating glibc Libraries on 3Dxx Display
Installing glibc (2.18.0) to root...
Configuring glibc.
Updating GPU Libraries on 3Dxx Display
Installing gpu-viv2 (3.0.101+4.1.1) to root...
Configuring gpu-viv2.
Updating Qt5 Libraries on 3Dxx Display
No packages removed.
Installing qt593 (5.9.3) to root...
Configuring qt593.
bootargs already set to console=ttymxc0,115200 lpj=7905280 rootfstype=ext4 root
=/dev/mmcblk0p1 ro rootwait maxcpus=1 board-ghi_imx6.pn=3D2104TK-100
Updating /etc/profile script
Updating /etc/profile.local script
Updating /usr/lib/fonts
setup3Dxx completed successfully... rebooting
Mon Mar  5 16:22:07 CST 2018
root@ghiimx6:~ █
```

N.B. The display resets once finished; repeat the above clean-up steps for closing stale windows

- Restore any custom modifications. The setup script preserved original copies as follows:
  - `/etc/profile.old`
  - `/etc/profile.local.old`

## Selecting a 3Dxx Qt Widget Demo Project

Qt Widget demonstration projects are provided for each of the 3Dxx Displays. There is a file in each demonstration program called “ghwrapper.cpp”. This file is a focal point for the demonstration program’s operation and in the very beginning of this file are comments explaining how the demonstration program works.

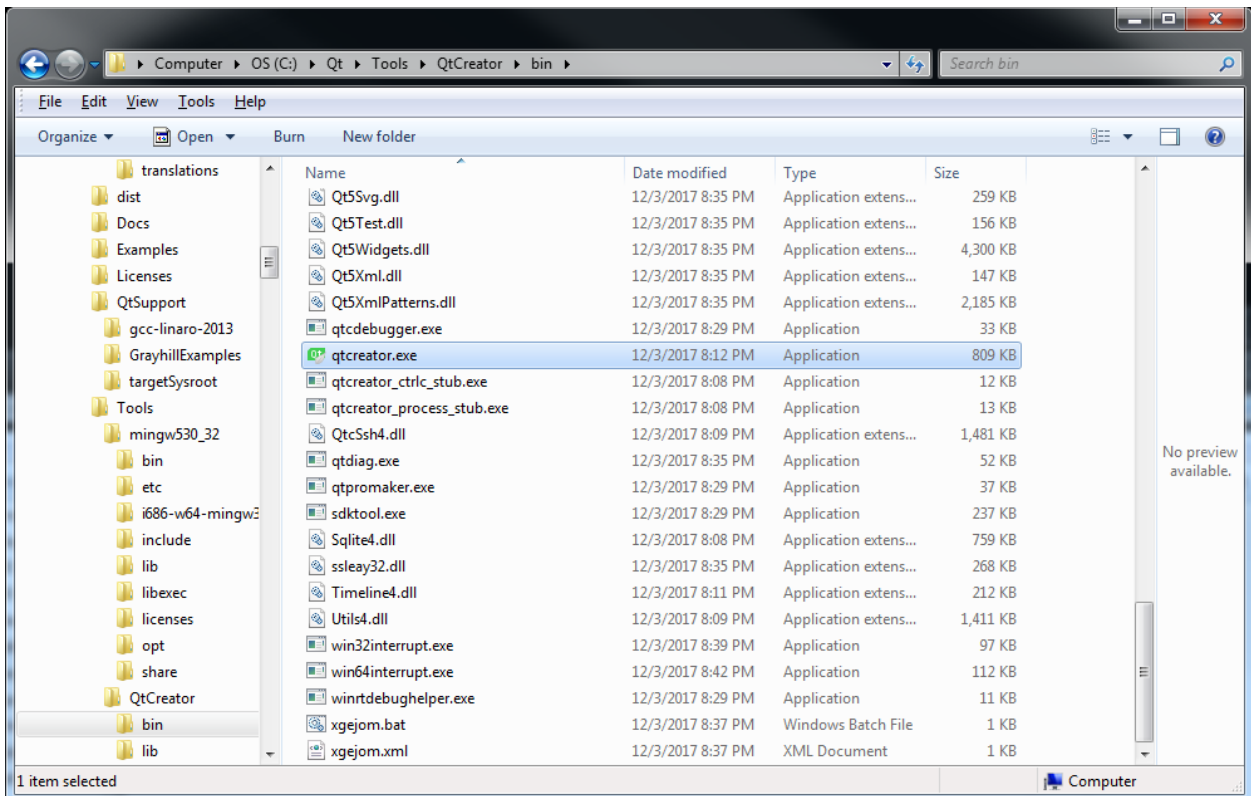
This table compares the features of the demonstration programs:

<b>Program Name</b>	<b>ghqtdemo</b>	<b>gh7indemo</b>	<b>gh10indemo</b>
<b>Target Display</b>	Model 3D50	Model 3D70	Model 3D2104
<b>Orientation</b>	Portrait	Landscape	Landscape
<b>Real Time Clock setting</b>	Yes	Yes	Yes
<b>CAN input</b>	Yes	Yes	Yes
<b>CAN output</b>	No	Yes	Yes
<b>Touch Screen tap input</b>	Yes	Yes	Yes
<b>Touch Screen Swipes</b>	Yes	Yes	Yes
<b>Digital Inputs shown</b>	4	4	4
<b>Digital Outputs shown</b>	4	4	4
<b>Video inputs shown</b>	2	3	3
<b>Buzzer demo</b>	N/A	Yes	Yes
<b>Audio Output demo</b>	N/A	Yes	N/A
<b>Analog Input demo</b>	N/A	Yes	N/A

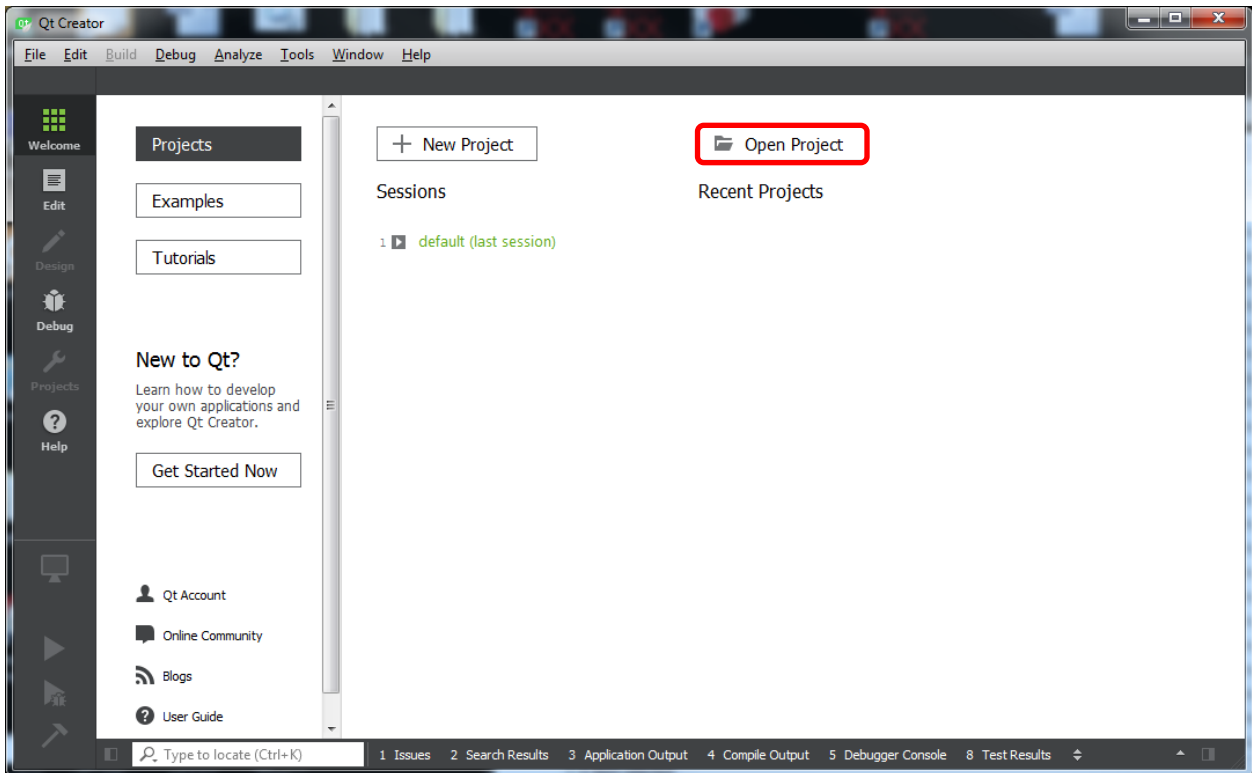
## Build and Run a 3Dxx Embedded Application (Widget)

This section details how to build and run a demo application on the 3Dxx Display.

- Launch Windows Explorer (<Windows>-e)
- Navigate to C: → Qt → Tools → QtCreator → bin → qtcreeator.exe

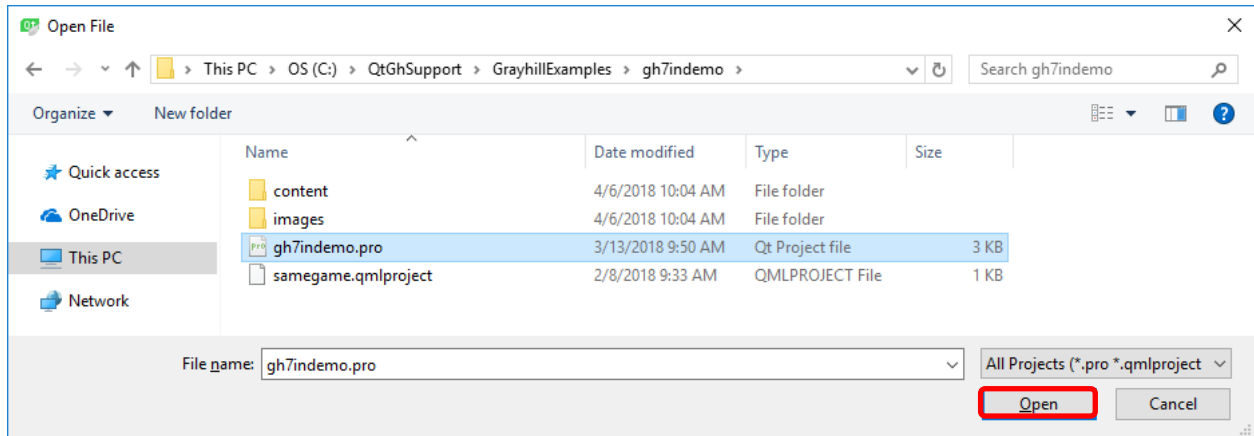


- Right click to select options like  
“Pin to Taskbar”  
“Send to” → Desktop (create shortcut)
- Double click to launch Qt Creator

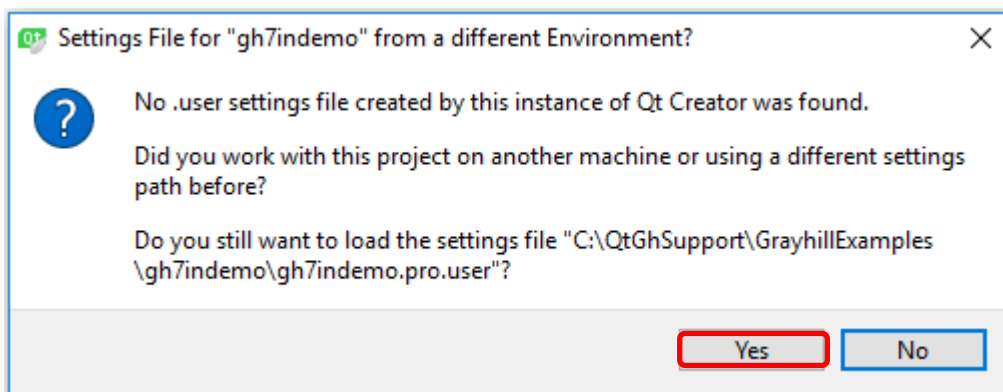


- Click on “Open Project”

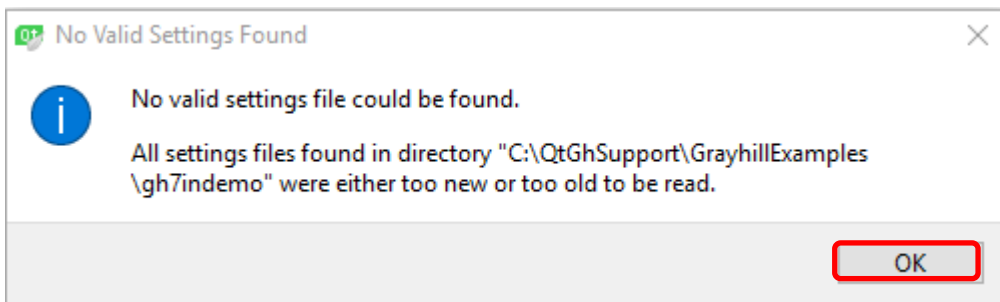
- Navigate to the desired project (C: → QtGhSupport → GrayhillExamples → gh7indemo)
- Select gh7indemo.pro



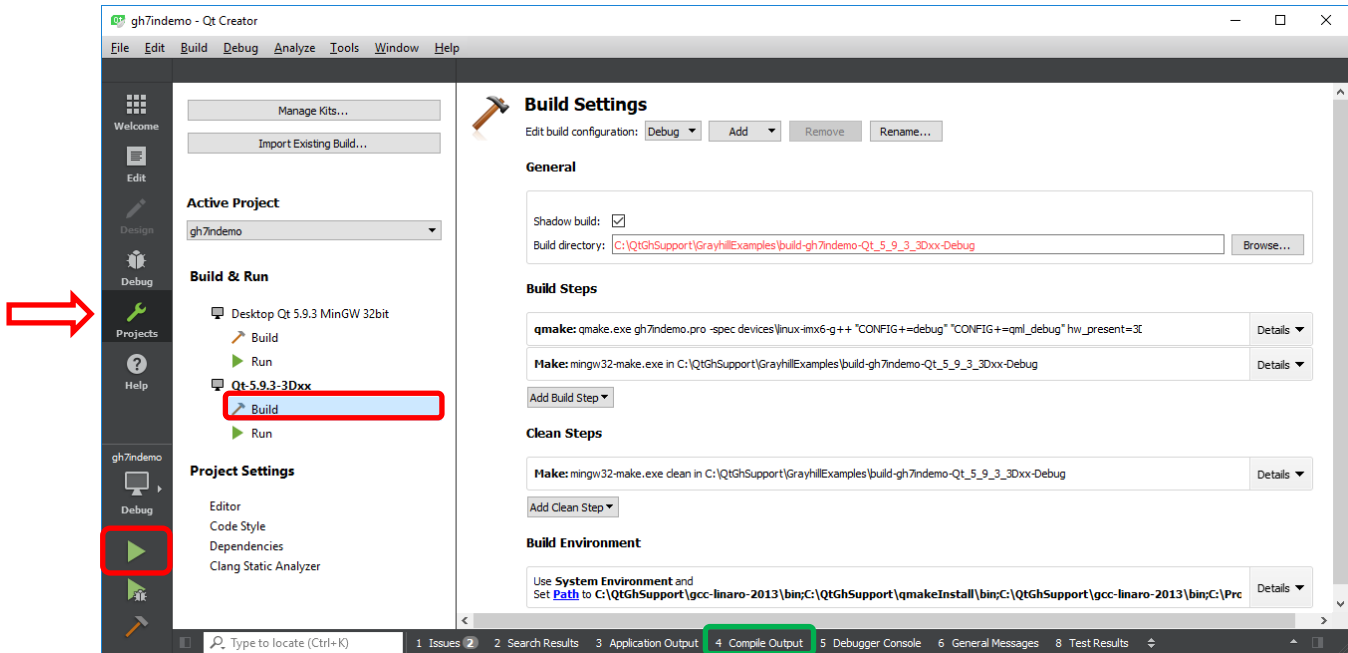
- Click “Open”
- If the following box appears, click “Yes”



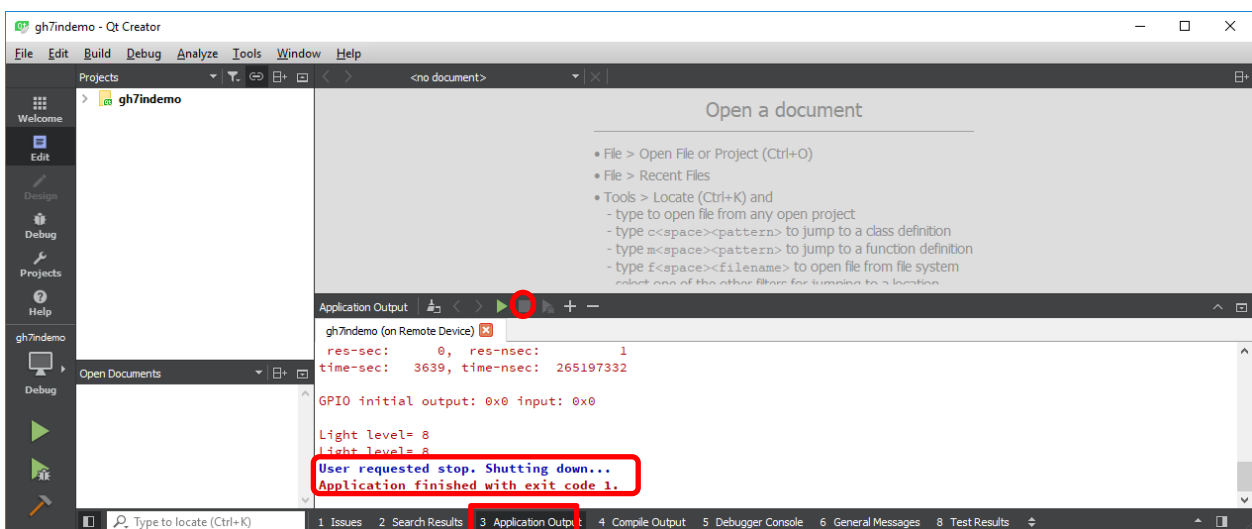
- If the following box appears, click “OK”. **Refer to Appendix B: Configuring a 3Dxx Project before continuing.** The current project configuration file is not compatible with the current version of Qt Creator and the project’s settings need to be re-configured.



- Select “Projects” view
- Select “Build” under “Qt-5.9.3-3Dxx”



- Click on the green arrow to run (a check to see if the executable is up to date is performed; if compilation is necessary the output can be viewed by clicking on the “Compile Output” tab)



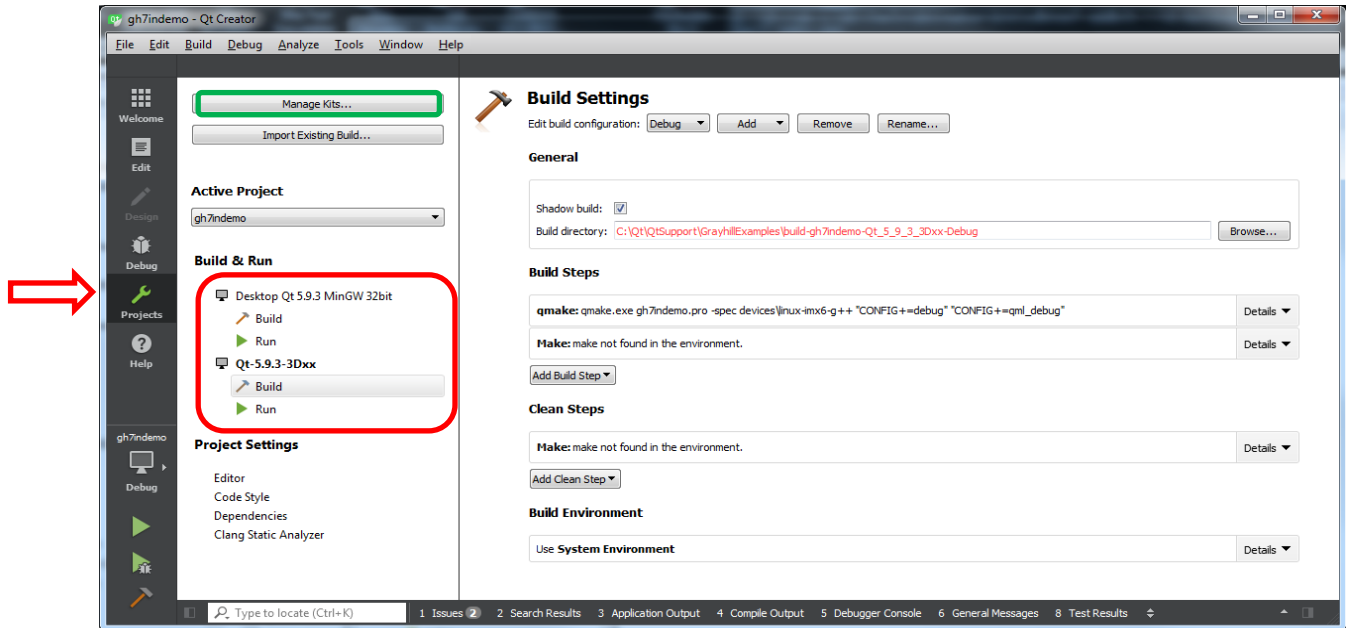
- Select the “Application Output” tab
- Click the red (when application is running on target) square to terminate the target session



## Appendix A: Configuring a Manual Qt Kit for Grayhill Displays

Note: This appendix is included for reference and is not a required installation step; Grayhill automatically installs the kit configuration as part of the support file installation. A kit is a collection of utilities (qmake, compilers, debugger, etc...) used to build a project.

- To see the list of available kits, select the “ Projects” view

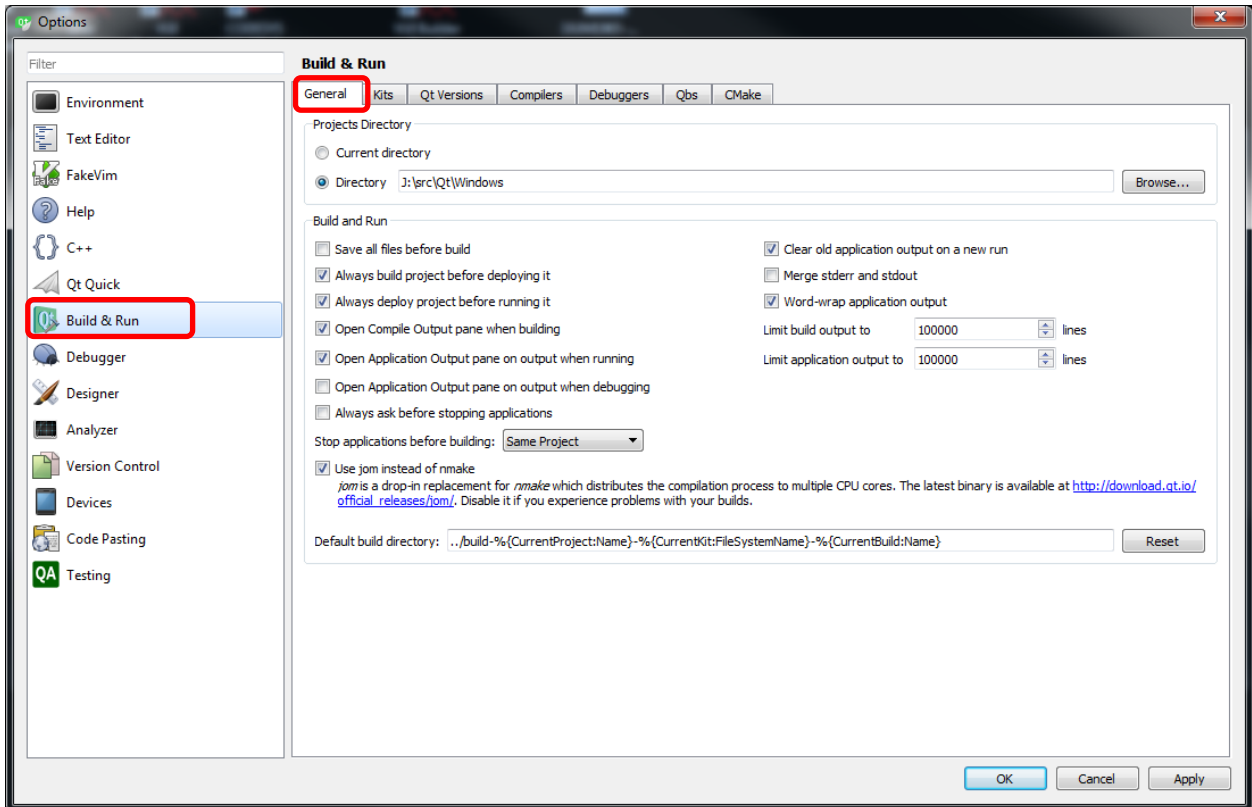


The above image illustrates the presence of two kits.

Should another kit be desired; these instructions describe the procedure for installing a Qt Creator kit.

- Click on “**Manage Kits**” (this is the same as selecting Tools → Options )

- Select “Build & Run”
- Select the “General” tab

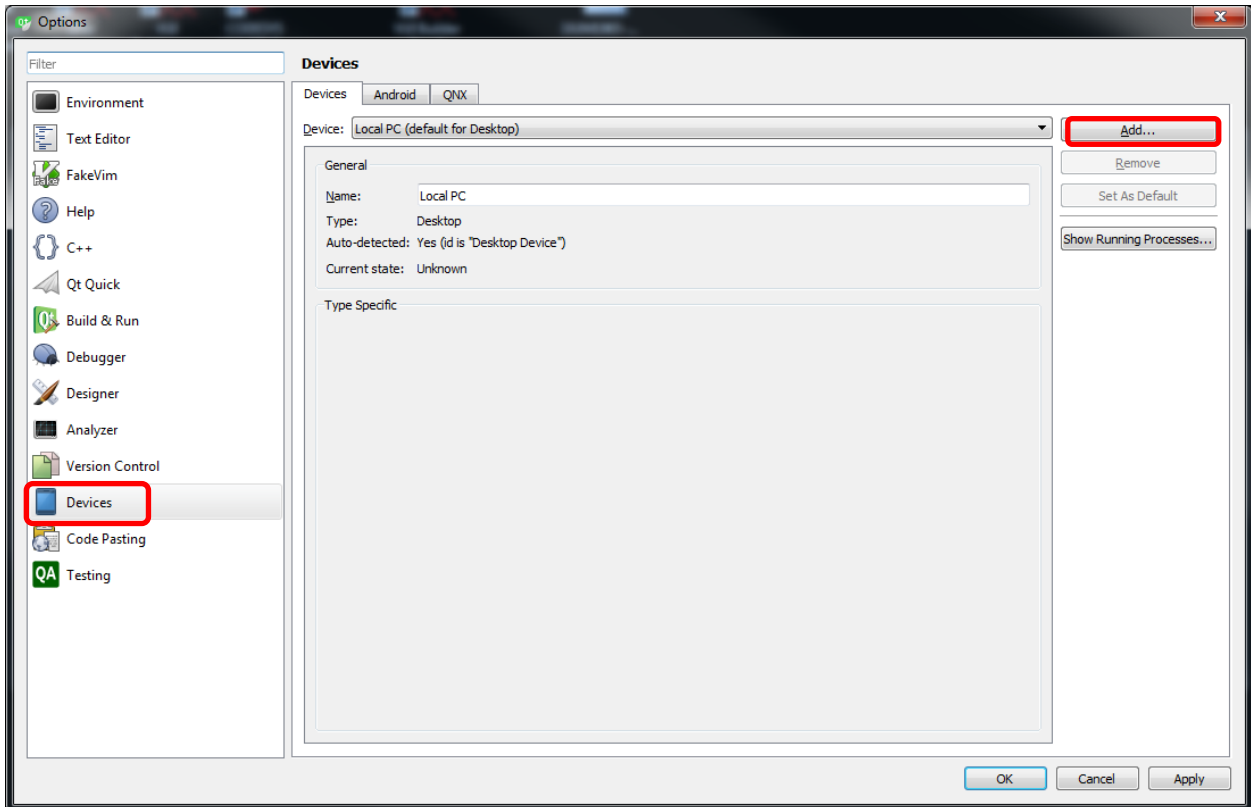


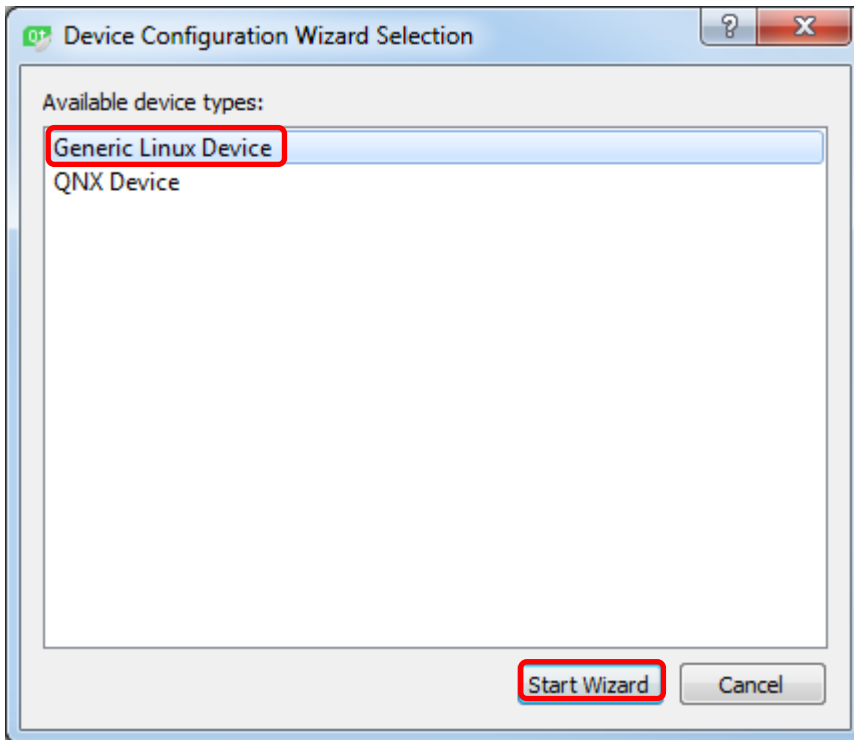
The “General” tab is where project wide customization is done. Review and select the desired configuration.

## Device

The section describes how to establish an Ethernet based connection to the display.

- Select “Devices”
- Click “Add...”





- Select “Generic Linux Device”
- Click “Start Wizard”

New Generic Linux Device Configuration Setup

Connection

Summary

The name to identify this configuration: 3Dxx Target

The device's host name or IP address: gmd

The username to log into the device: root

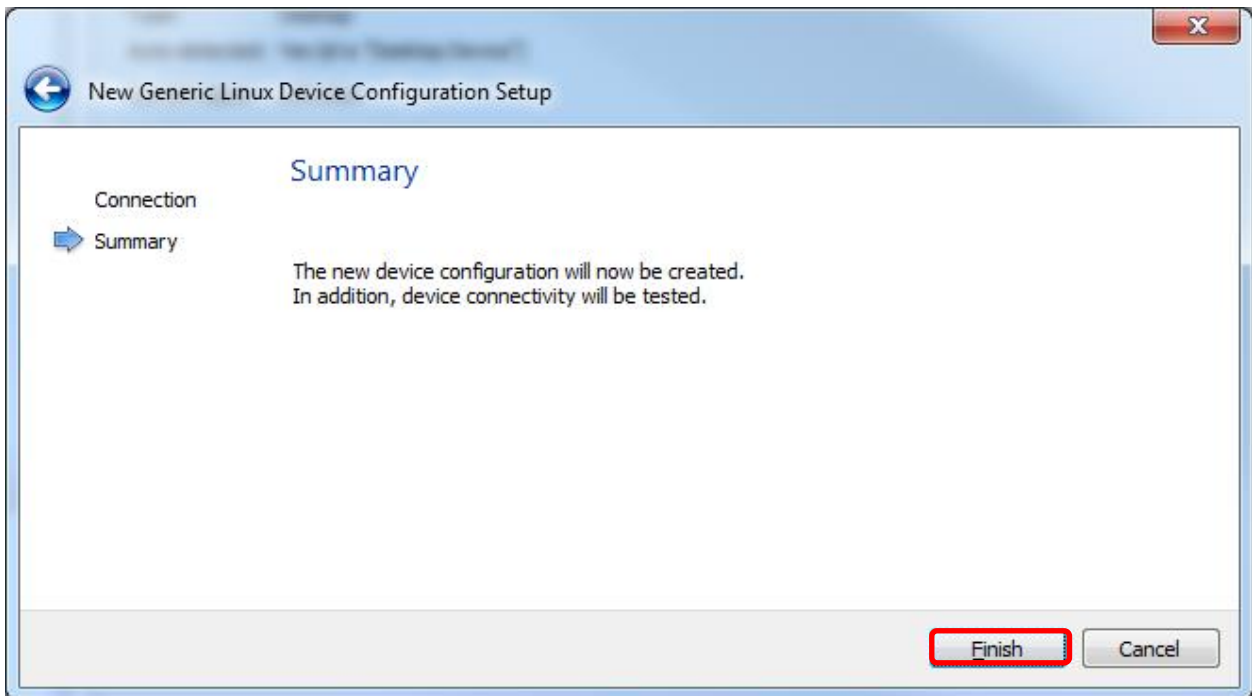
The authentication type:  Password  Key  Agent

The user's password:

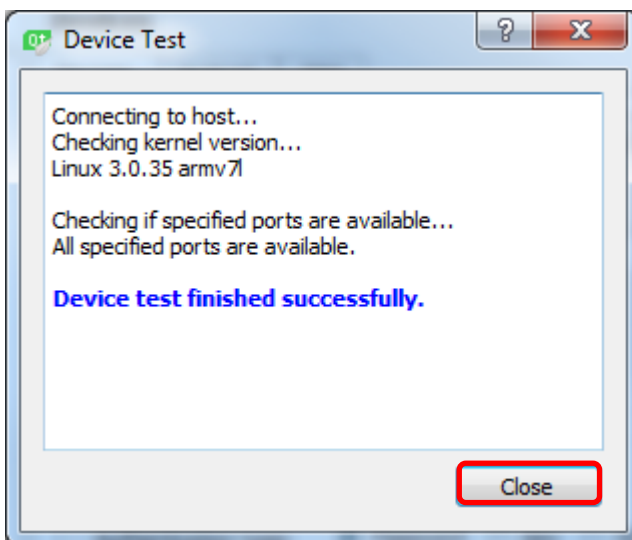
The file containing the user's private key: C:\Users\admin\.ssh\id\_rsa Browse...

Next Cancel

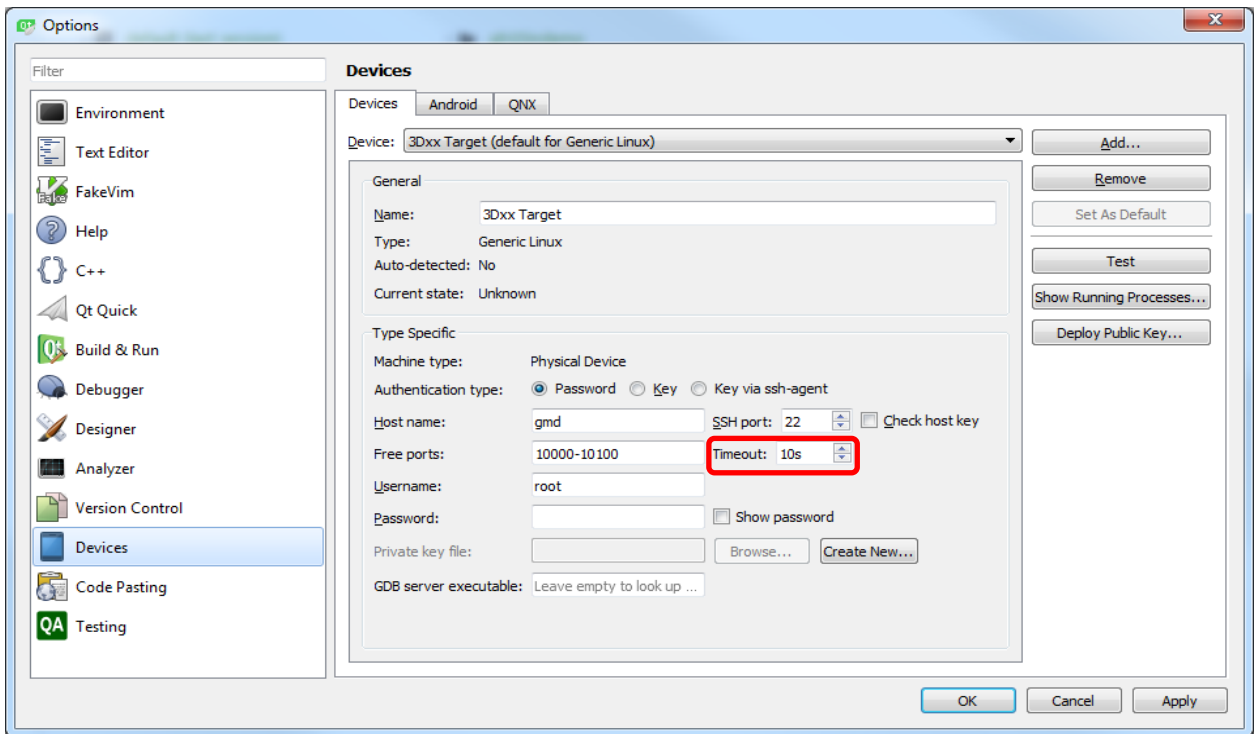
- Populate the fields as illustrated above
- N.B. The IP address associated with gmd was configured in *hosts* (C:\Windows\System32\drivers\etc)
- Click “Next”



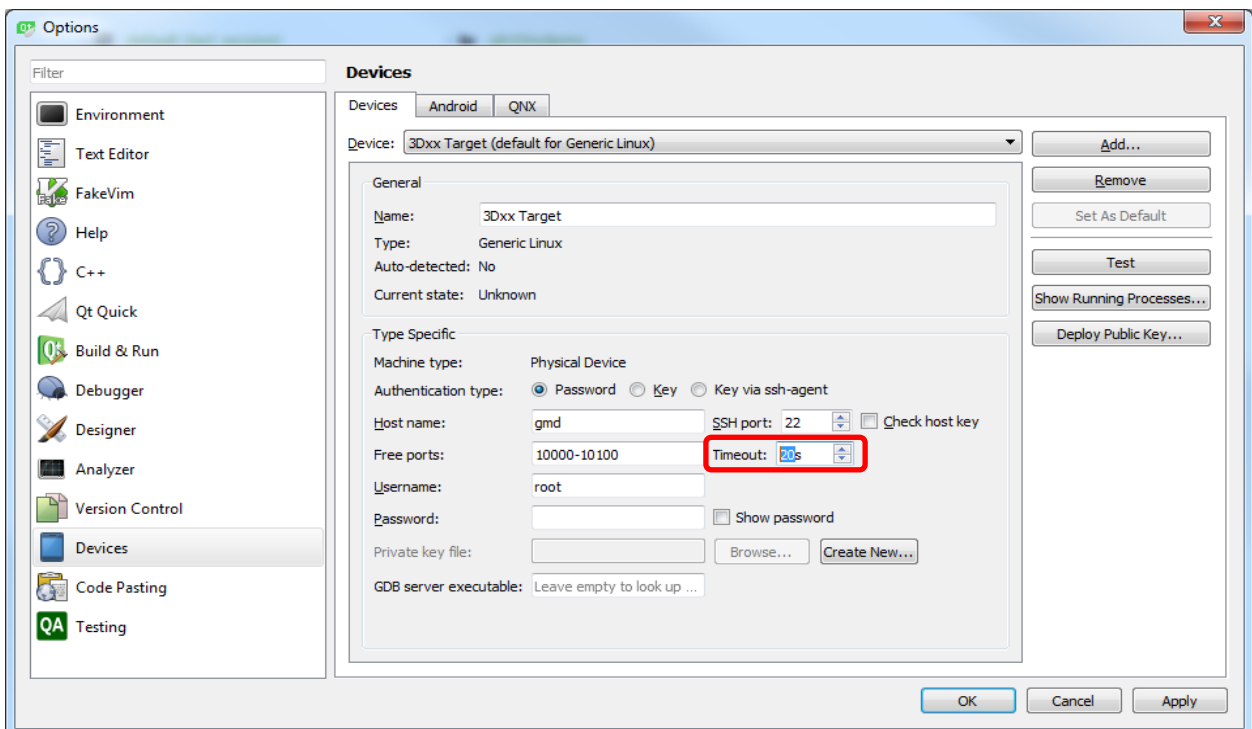
- Verify the 3Dxx Display is still powered up
- Click “Finish” – The Ethernet link to the 3Dxx Display will be tested and if successful the following result screen appears



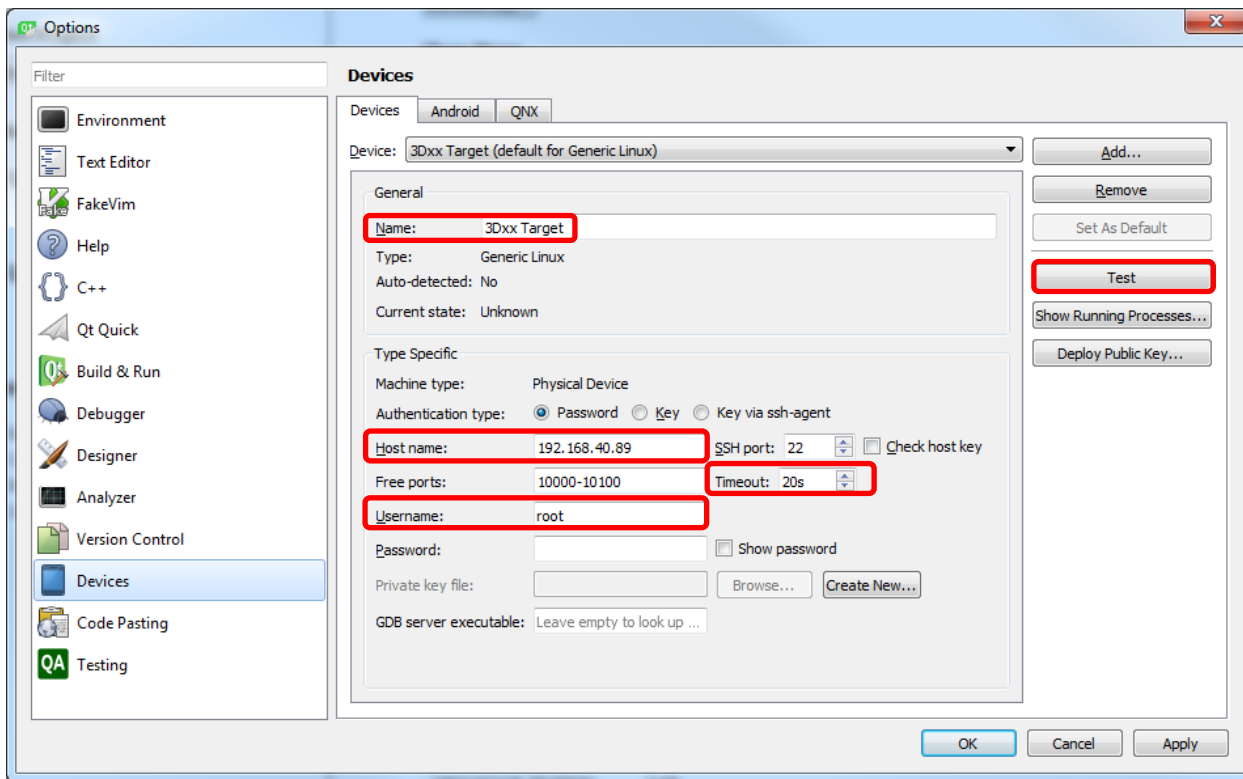
- Click “Close”



- Click the upper arrow on the right side of the “Timeout:” box to increase timeout value to “20s”



- Devices Summary



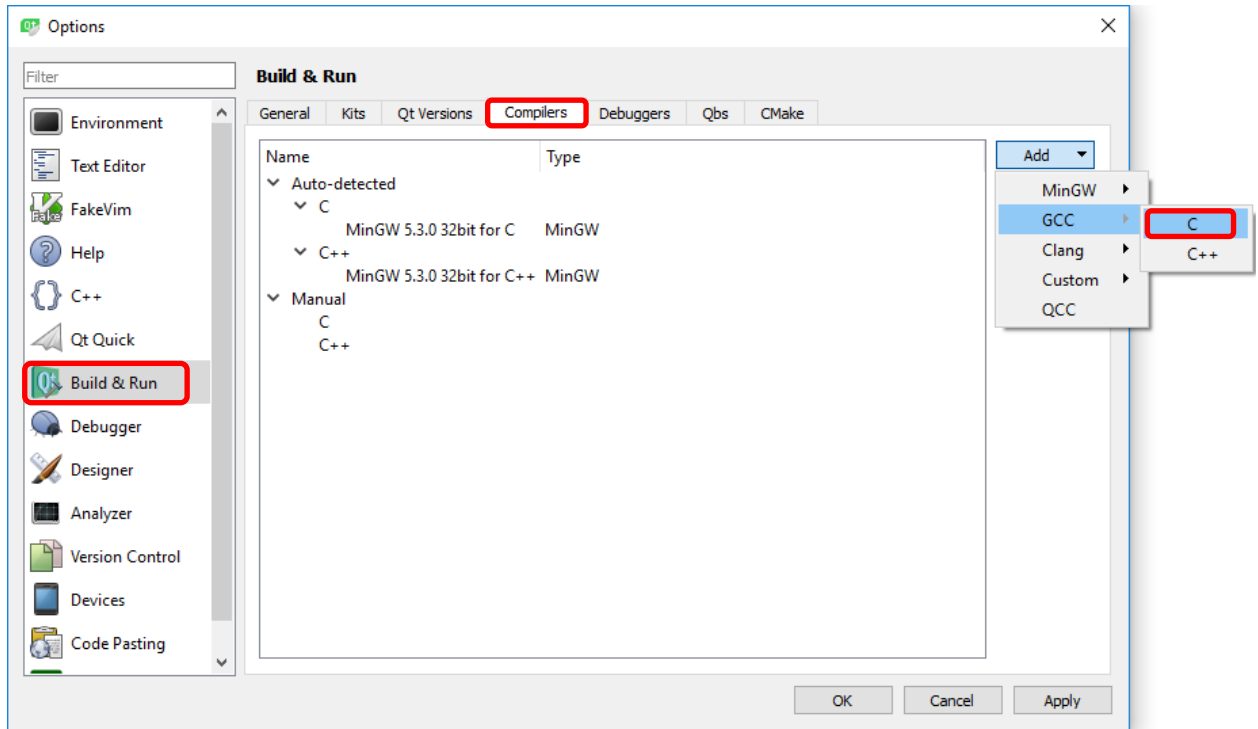
- Name name of the device
- Host name can be “alias” like *gmd* specified in *hosts* or a hard coded IP
- Timeout 20s
- Username root

**N.B. Remember to “Test” to make sure connectivity has been established**

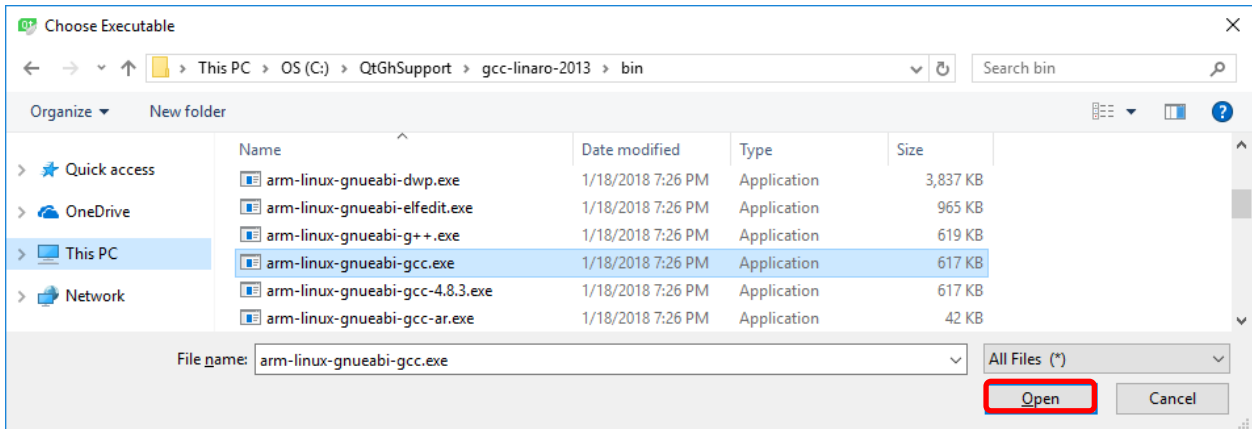


## Compiler

- Select “Build & Run”
- Select “Compilers” tab
- Click “Add”; then select GCC → C

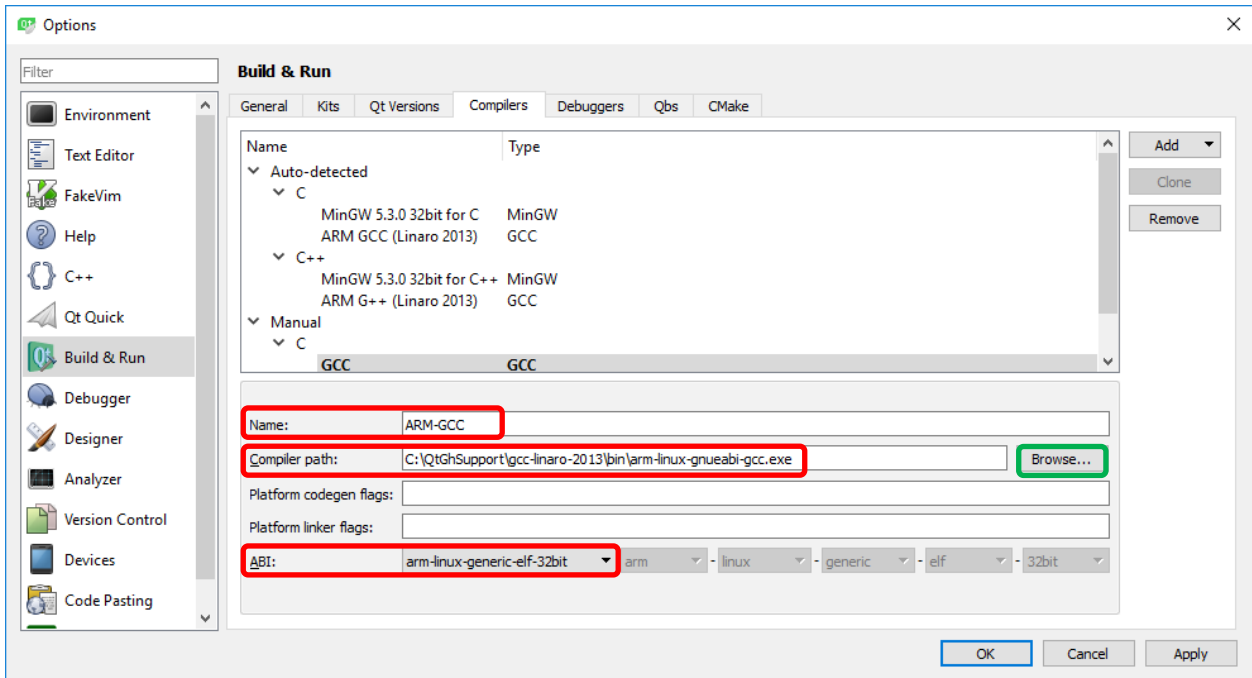


- Populate the fields as illustrated  
 “Name:” ARM-GCC  
 “Compiler path:” Click “Browse...” and navigate to the desired file  
 C:\QtGhSupport\gcc-linaro-2013\bin\ arm-linux-gnueabi-gcc.exe

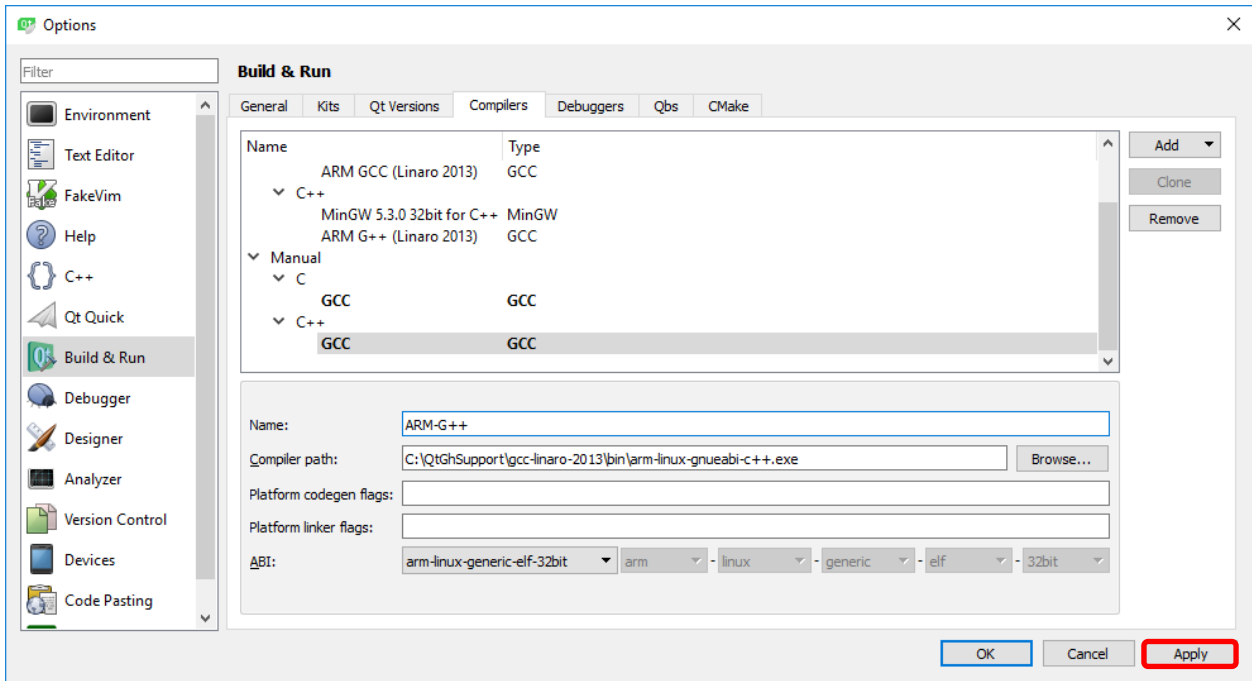


Click “Open”  
 “ABI:” Select “arm-linux-generic-elf-32bit”

- The configuration portion of the screen should look similar to:



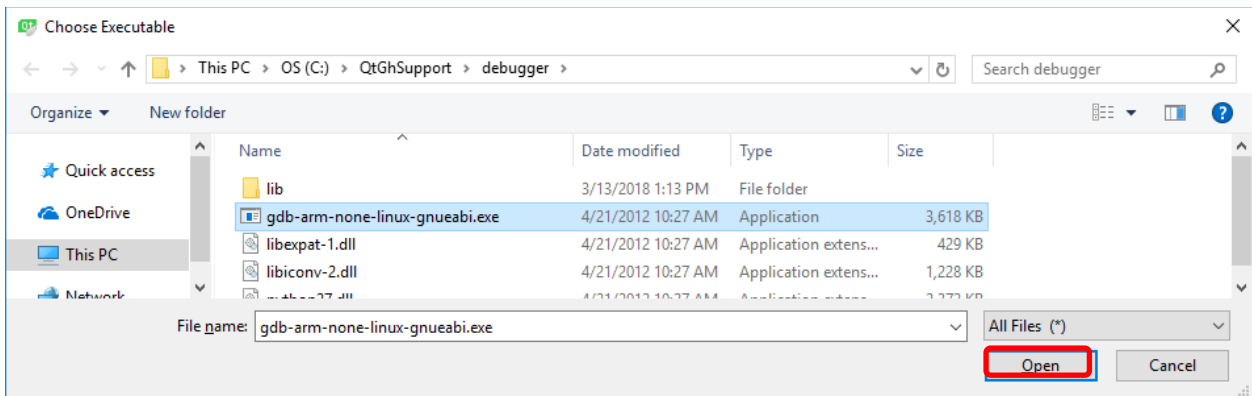
- Repeat the above steps for GCC→C++



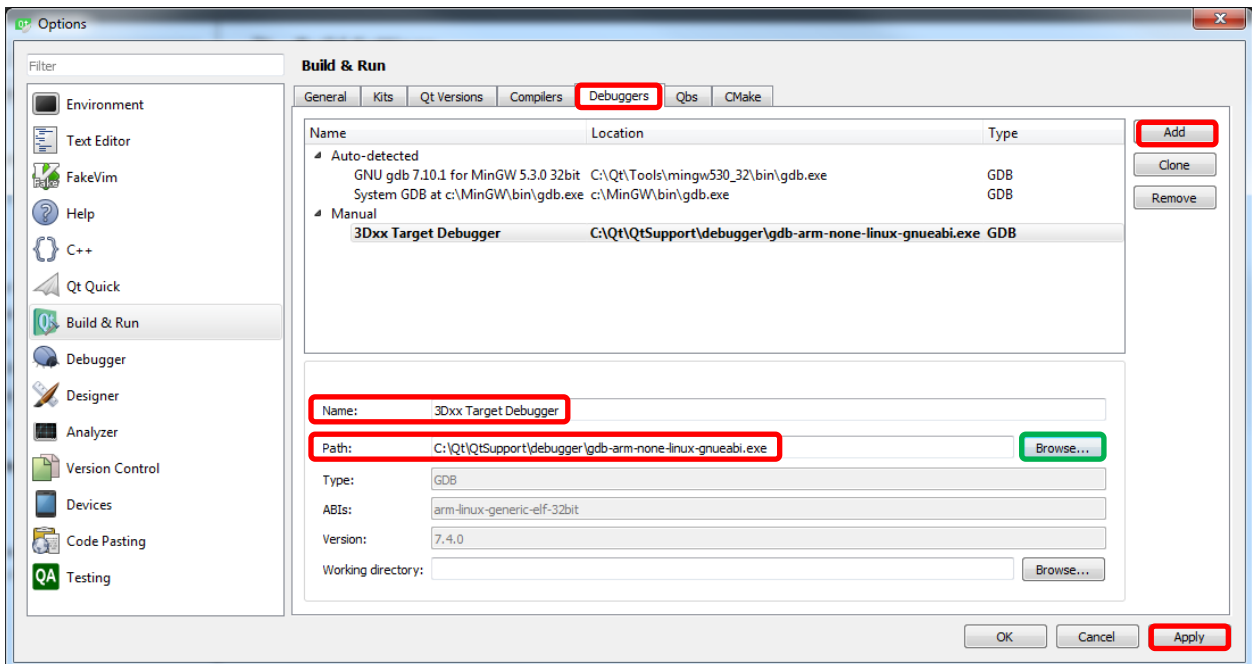
- Click “Apply”

## Debugger

- Select “Debuggers” tab
- Click “Add”
- Populate the fields as illustrated
  - “Name:” 3Dxx Target Debugger
  - “Path:” Click “Browse...” and navigate to the desired file  
C:\QtGhSupport\debugger\arm-linux-gnueabi-gcc.exe



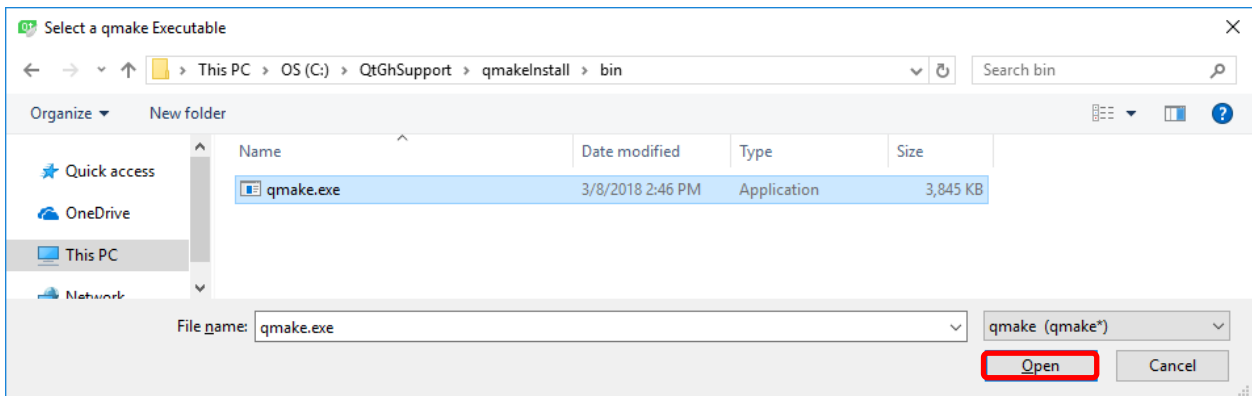
- Click “Open”; the configuration portion of the screen should look similar to



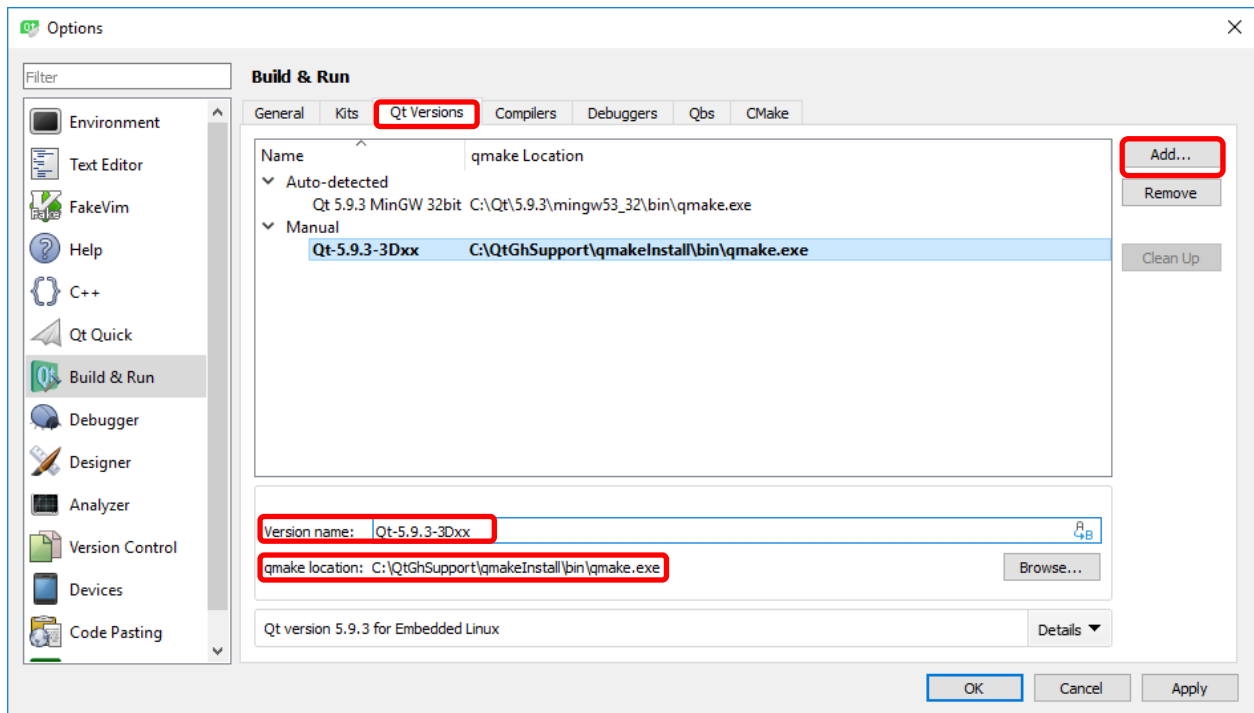
- Click “Apply”

## qmake

- Select the “Qt Versions” tab
- Click “Add” (Select a qmake Executable dialog box appears; still referencing the last path)
- Navigate to the provided qmake version C:\QtGhSupport\qmakeInstall\bin\qmake.exe

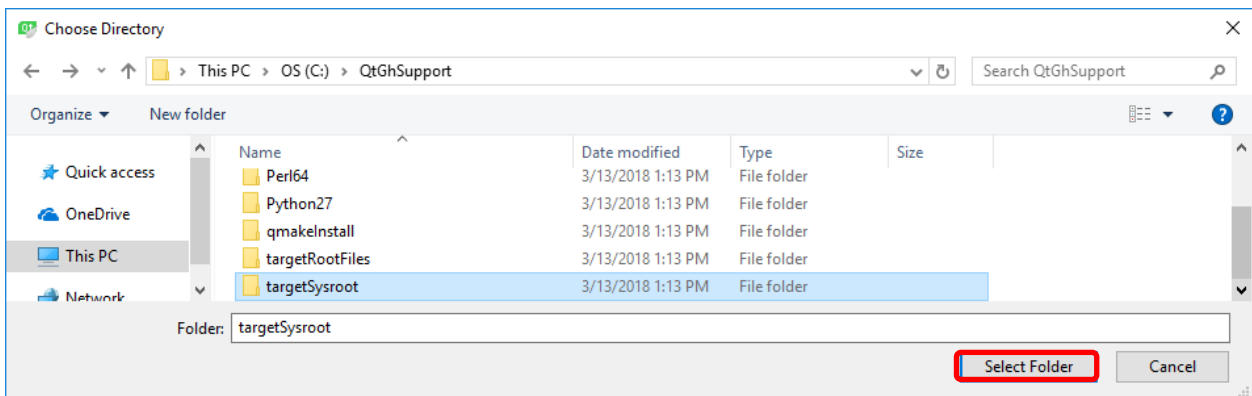


- Click “Open”
- Update “Version name:” to “Qt-5.9.3-3Dxx”



## Kit

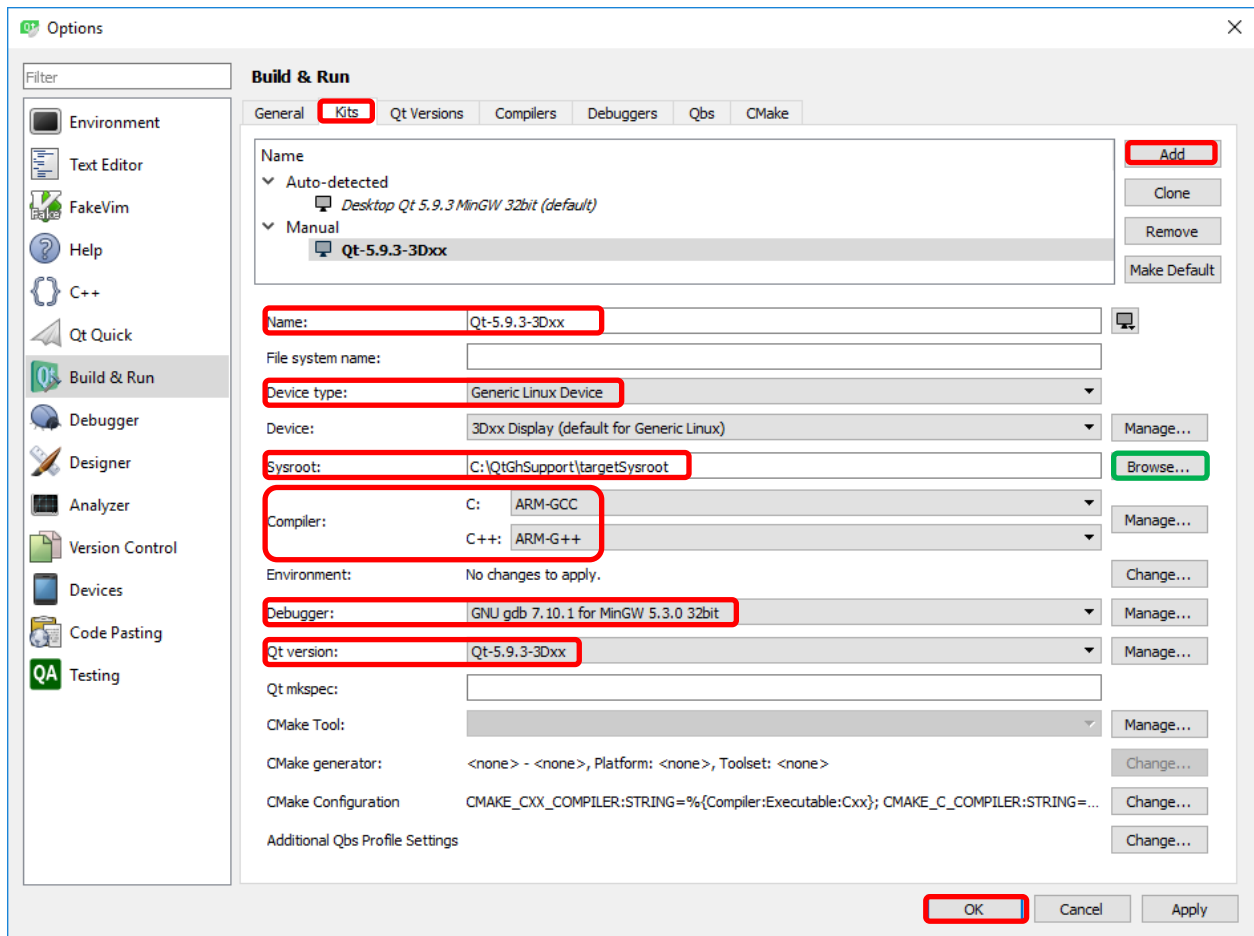
- Select the “Kits” tab
- Click “Add”
- Populate the fields as illustrated
  - “Name:” Qt-5.9.3-3Dxx
  - “Device type:” Select “Generic Linux Device” from the pick list
  - N.B. Automatically updates Device
  - “Sysroot:” Click “Browse...” and navigate to desired path
  - C:\QtGhSupport\targetSysroot



Click “Select Folder”

- “Compiler: C:” Select “ARM-GCC” from the pick list
- “Compiler: C++:” Select “ARM-G++” from the pick list
- “Debugger:” Select “3Dxx Target Debugger” from the pick list
- “Qt version:” Select “Qt-5.9.3-3Dxx” from the pick list

N.B. The selected names match those used when creating the various kit sub-components



- Verify contents are correct
- Click “OK”

Now that a Qt kit is configured; it is possible to develop, build, test, debug, run and enjoy Qt applications.

## Appendix B: Configuring a 3Dxx Project

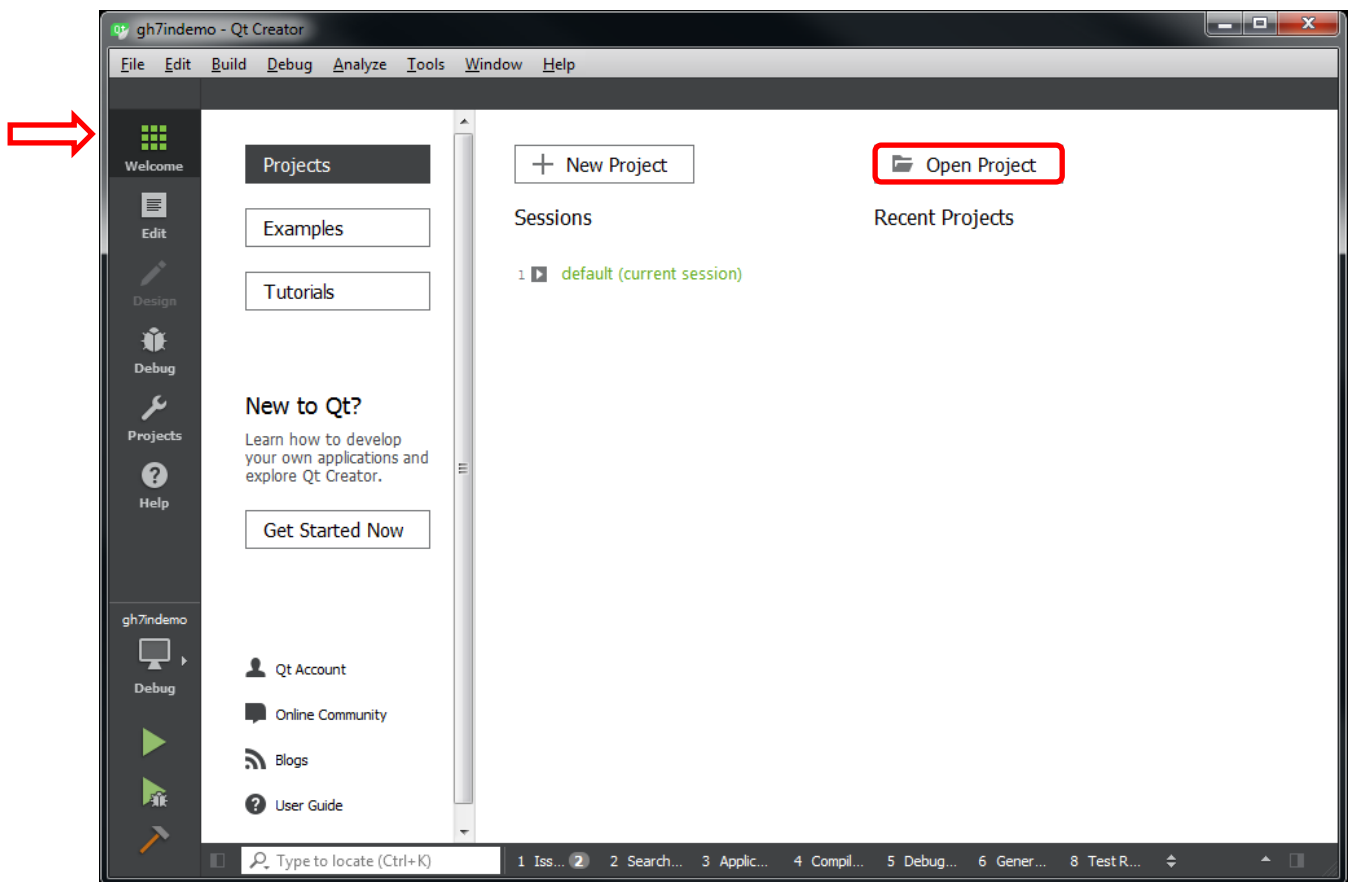
Note: This appendix is included for reference and is not a required installation step; Grayhill automatically configures the project as part of the support file installation.

This section details how to setup and configure a new project for the 3Dxx Display.

If not already running, launch Qt Creator. (See **Build and Run a 3Dxx Embedded Application**)

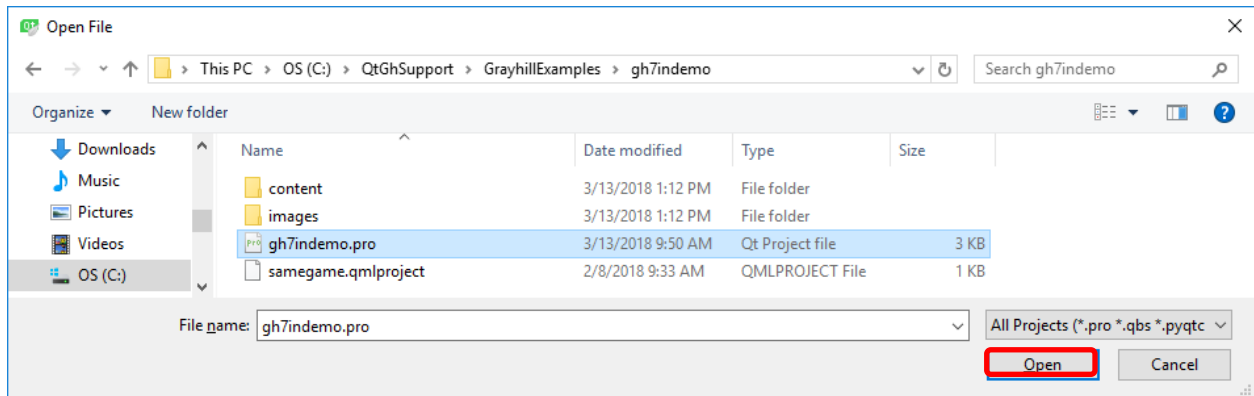
Open the gh7indemo project from “Qt Creator” main window click on “Open Project” button.

N.B. If present, a previous project can be opened by clicking on the project name listed below “Recent Projects”.

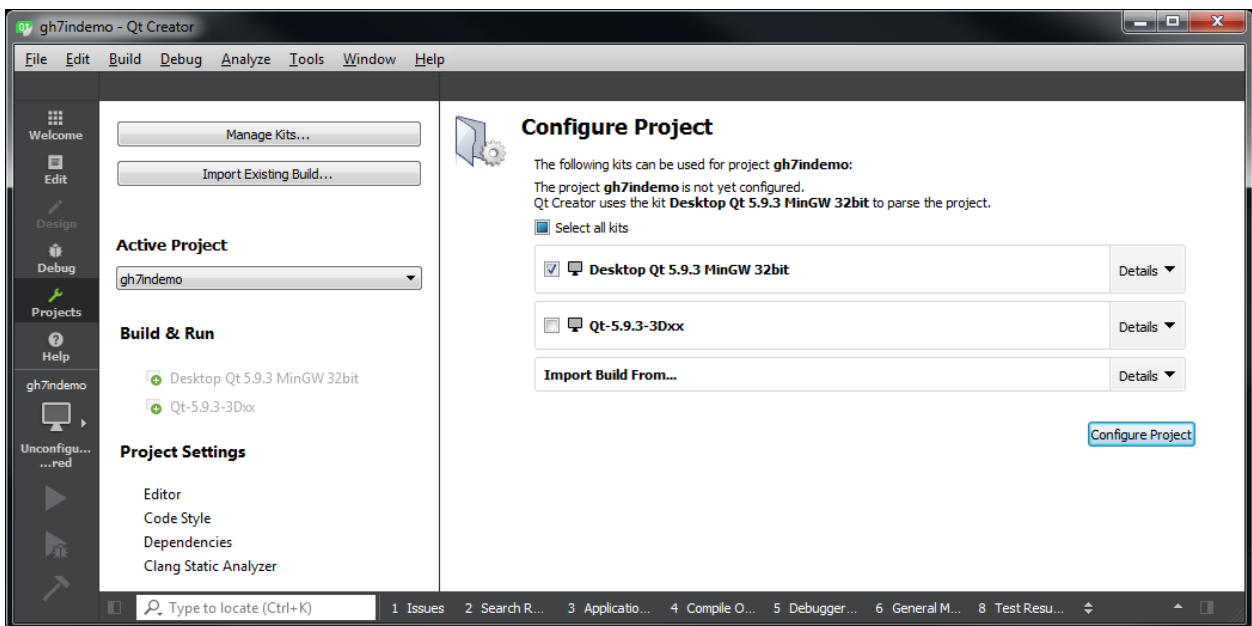




- An “Open File” dialog window will appear
- Navigate to the 3Dxx Demo project’s “.pro” file (and click to select)  
C:\QtGhSupport\GrayhillExamples\gh7indemo\gh7indemo.pro



- Click “Open”
- If the “*project.pro.user*” file is missing, which is normal if the project has never been opened before, a “Configure Project” dialog appears. If this dialog doesn’t appear, proceed to where the “Projects” icon is selected.
- If the “Configure Project” dialog appears (remember screen shot illustrations are for reference purposes and may not reflect current observations)

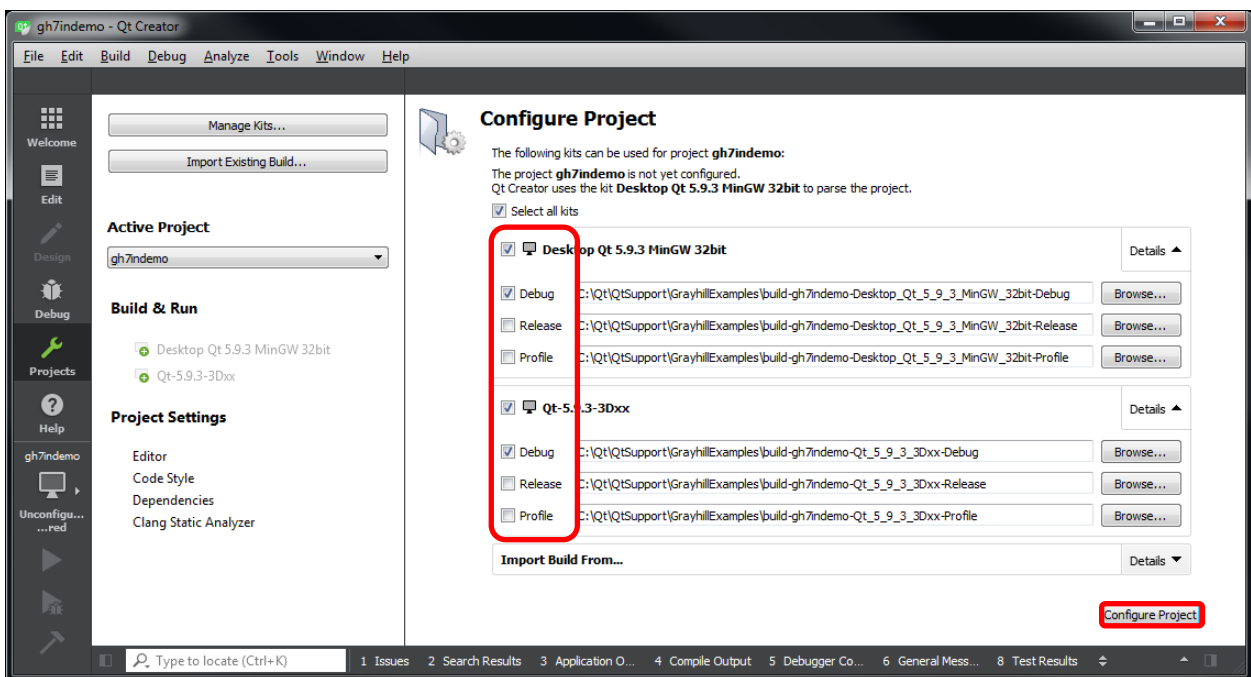


“Desktop Qt 5.9.3 MinGW 32bit”

- Expand by clicking on “Details”  
Unselect “Release”  
Unselect “Profile”

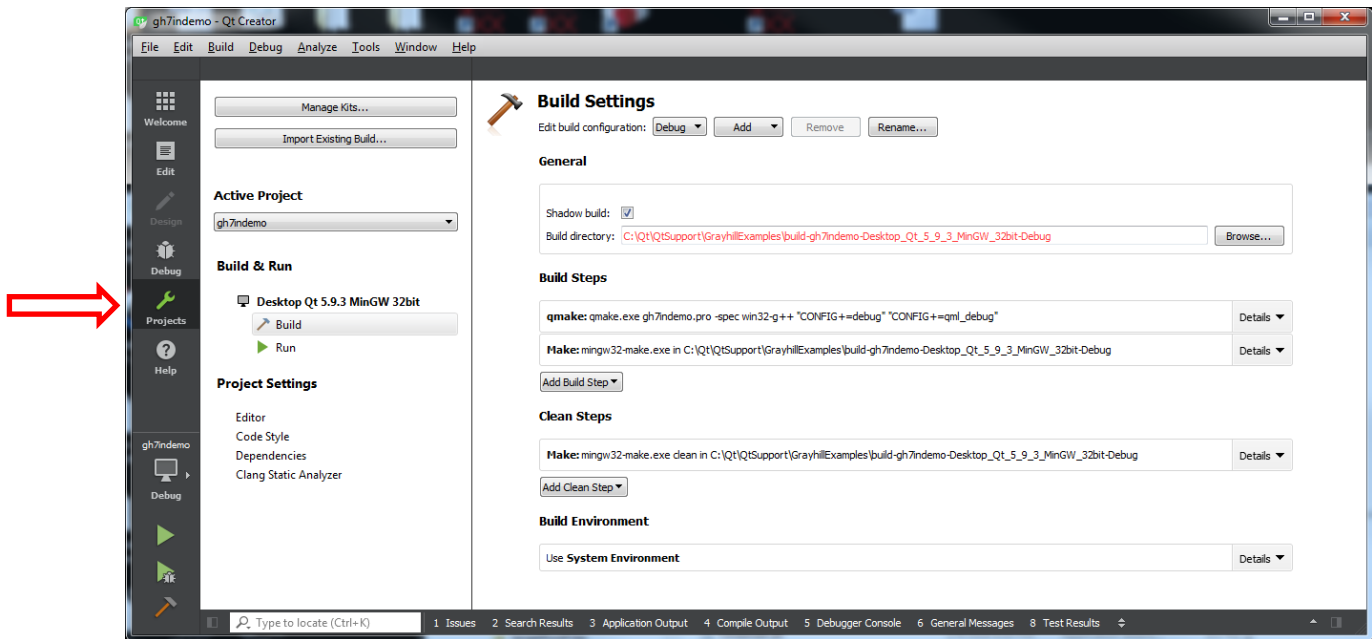
“Qt-5.9.3-3Dxx”

- Expand by clicking on “Details”  
Select “Qt-5.9.3-3Dxx” (this selection will select the three boxes below)  
Unselect “Release”  
Unselect “Profile”



- Click “Configure Project”

- On the main “Qt Creator” window select “Projects”



- If the desired kit is not shown see Appendix A: Configuring a Manual Qt Kit for Grayhill Displays
- N.B. Clicking “Manage Kits” is the same as selecting “Tools → Options”

“Active Project” is a drop down pick list with the active project shown.

“Build & Run” lists the available kits.

N.B. The selected kit is emphasized in **bold**. A kit (set of utilities) is how the project will be built, e.g. the main kit difference is the compiler as the Qt-5.9.3-3Dxx kit uses a cross compiler for the display.

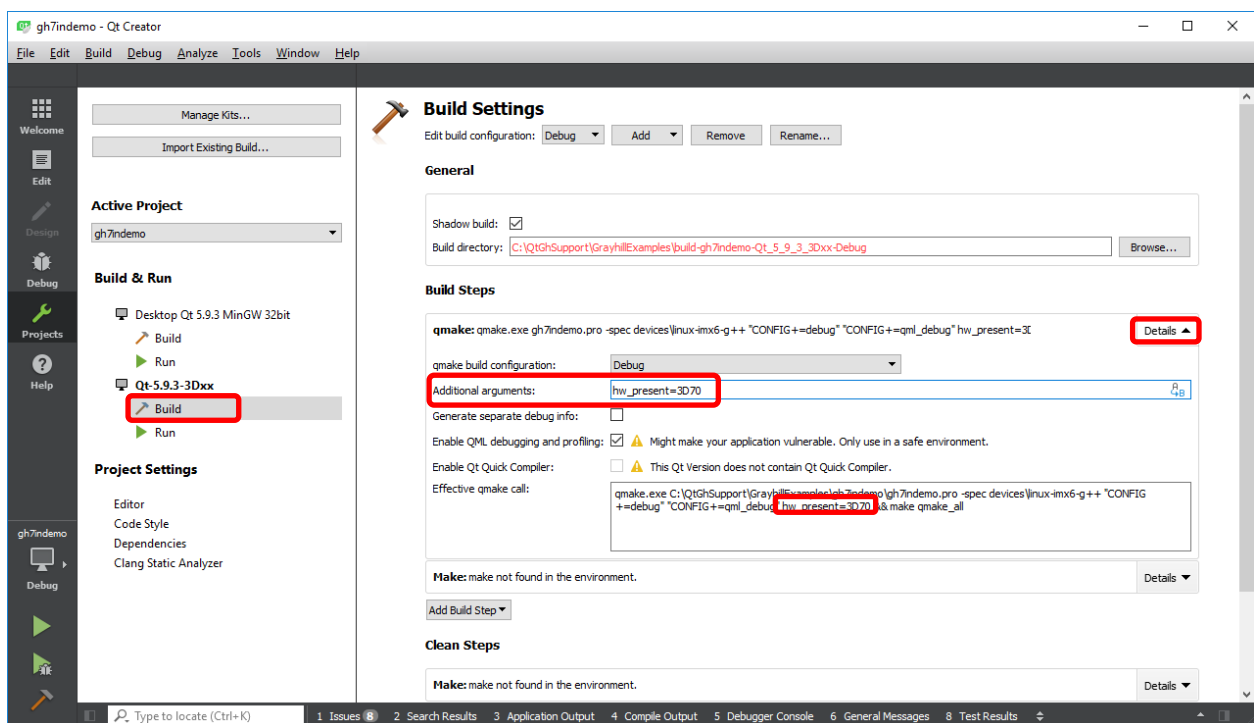
N.B. Clicking on an actual kit name selects either Build or Run (depending on which one was previously selected)

## Build

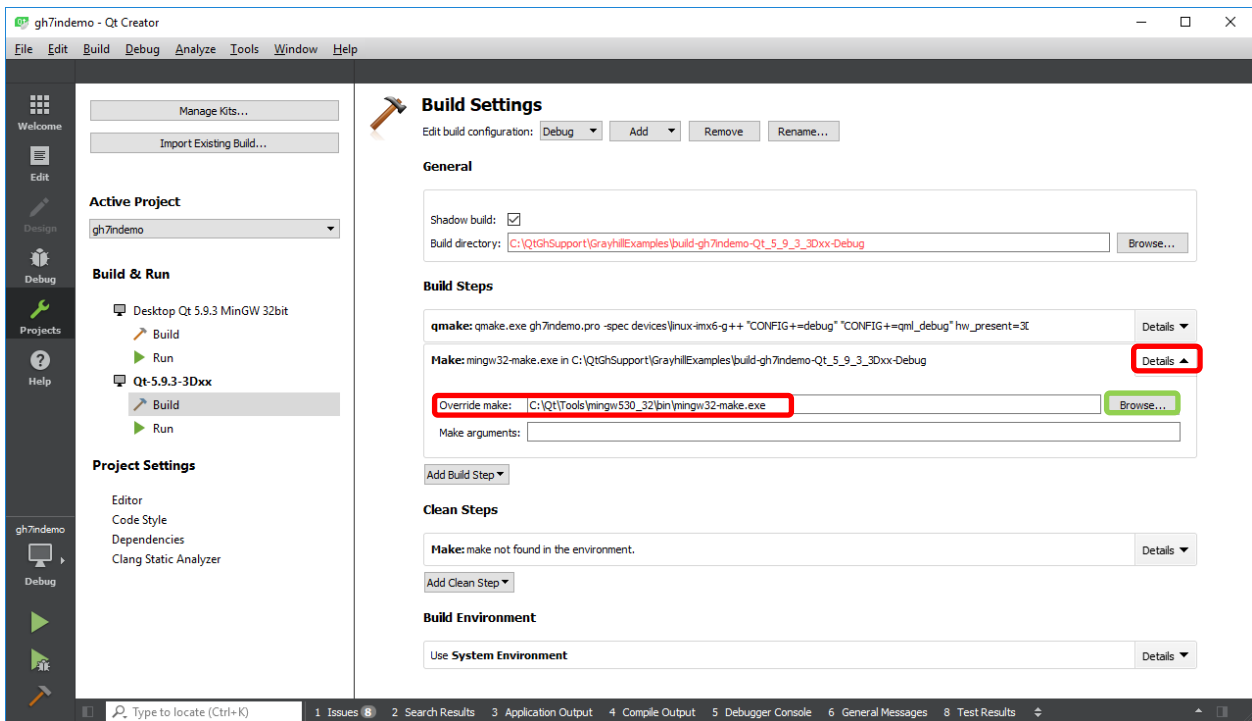
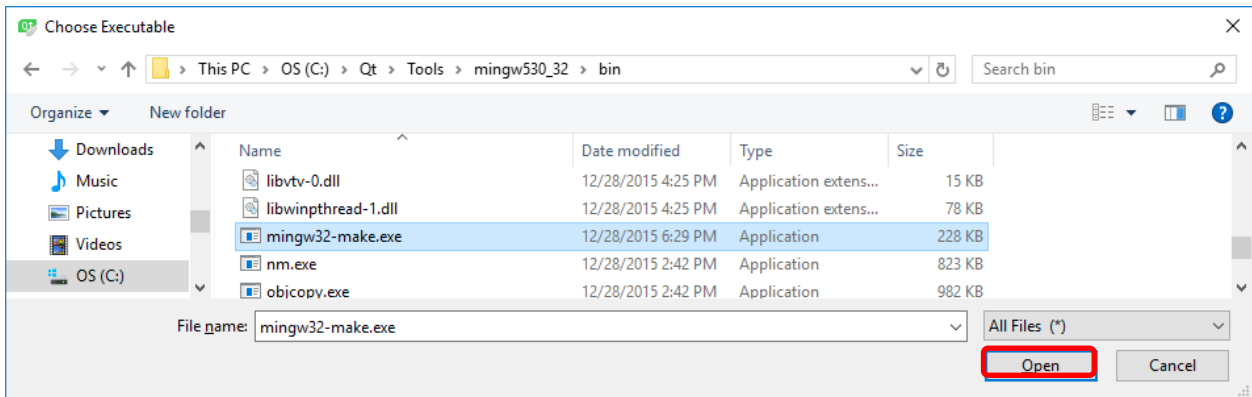
This is a target (3Dxx Display) build example walk-through; select “Build”.

- Expand the Details tab associated with qmake (under Build Steps)
- “Additional arguments” Enter “hw\_present=3D70” – N.B. This is a **case sensitive** field.

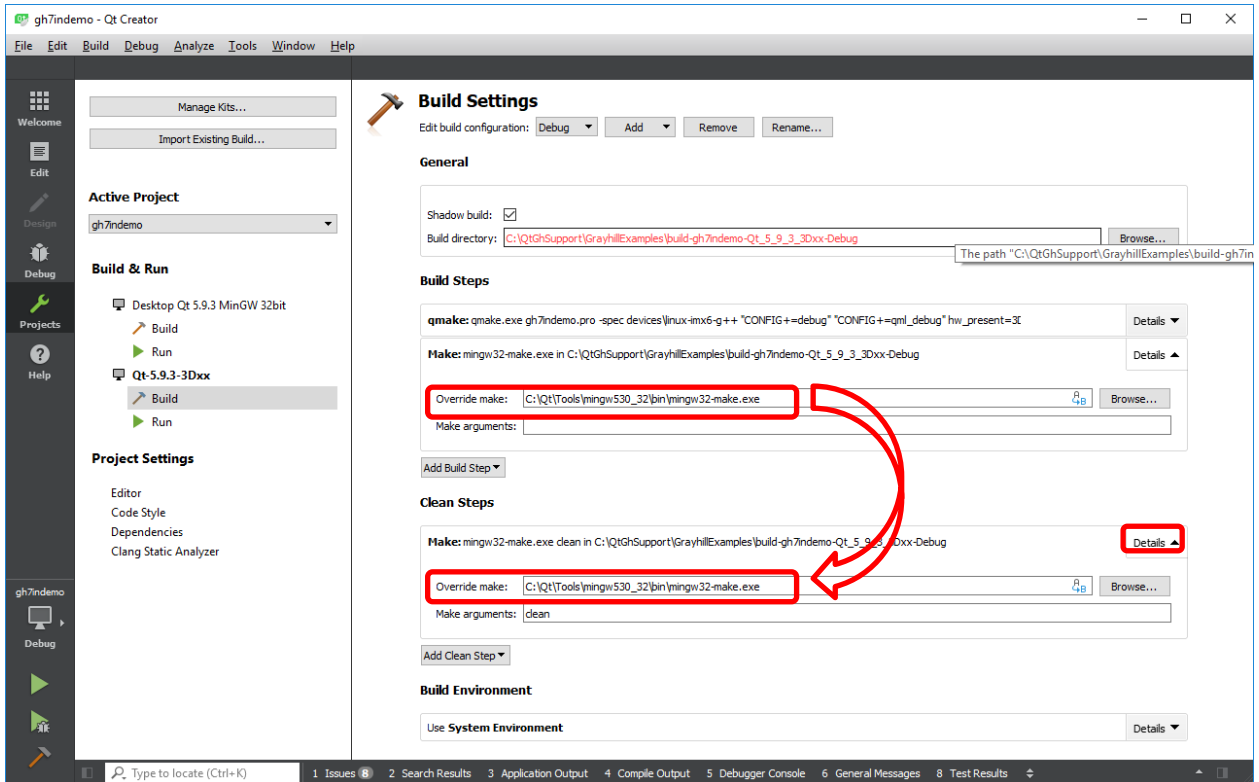
N.B. the parameter is automatically added to the “effective qmake call” command syntax. This field is configured based on the actual target hardware display size.



- Expand the “Details” tab associated with “Make” under “Build Steps”
- Click on “Browse”  
Navigate to C: → Qt → Tools → mingw530\_32 → bin  
Select mingw32-make.exe  
Click “Open”



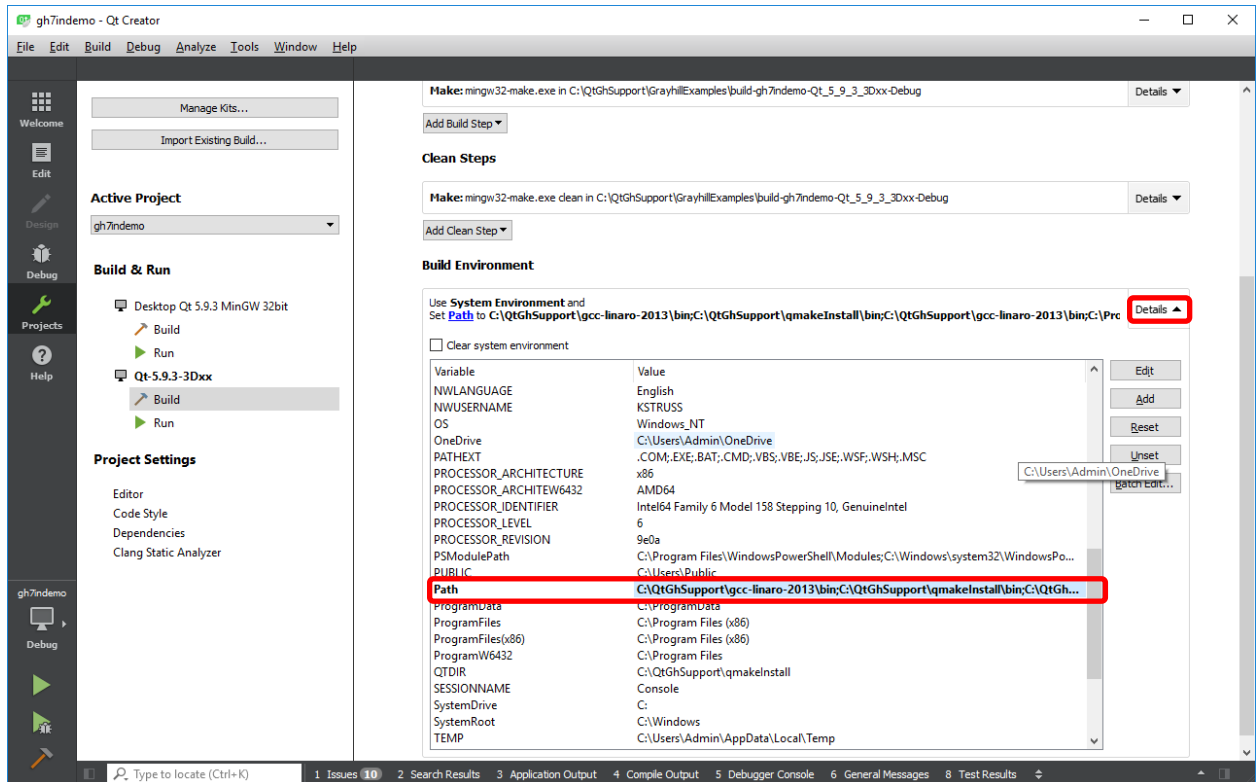
- Expand the “Details” tab associated with “Make” under “Clean Steps”
- Copy and paste the contents of “Override make:” from “Build Steps” to “Clean Steps”



- Expand the “Details” tab associated with “Use System Environment” under “Build Environment”
- Scroll down to “Path” and double click to edit

**N.B. The entire contents are selected; press the right arrow key before typing**

- Append “C:\Qt\5.9.3\mingw53\_32\bin”



## Run

- Select “Run”

- Deployment

Method: Deploy to Remote Linux Host (should be default)

Files to deploy:

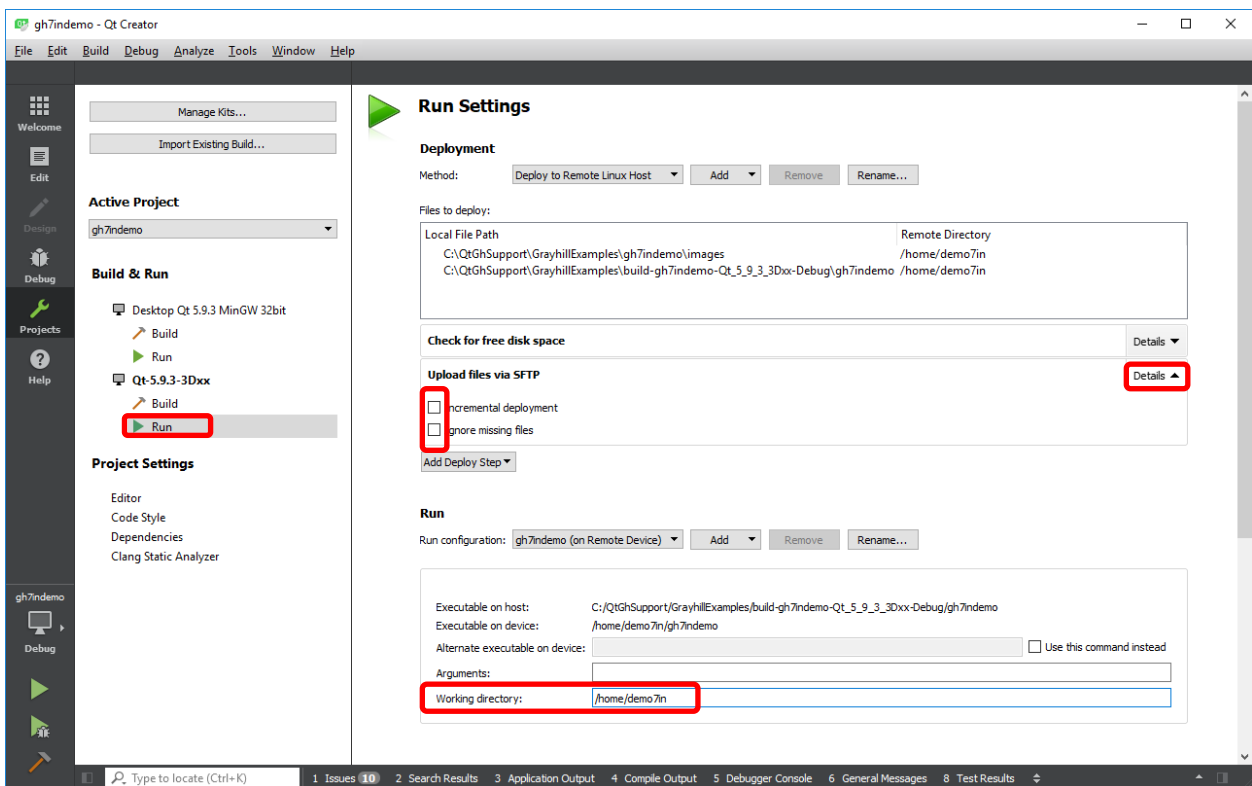
Local File Path	location of the local file(s)	(auto-populated)
Remote Directory	location on the target	(auto-populated)

N.B. The file information may not be populated until after a build is done

- Expand “Details” for “Upload files via SFTP”

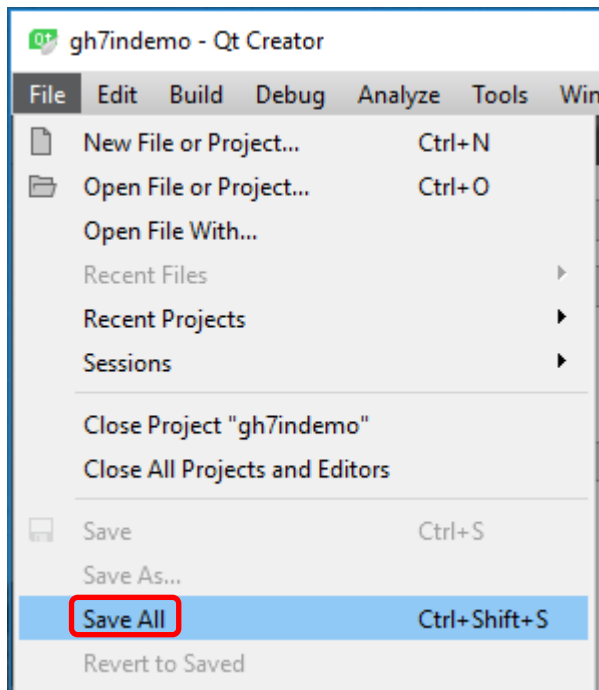
N.B. On rare occasions Qt Creator thinks the files have been deployed and will not re-send the files to the target; disabling this functionality avoids the situation.

- Make sure neither box is selected
- Set “Working directory:” under Run to the directory associated with the “Executable on device:”
- Enter “/home/demo7in” in the box

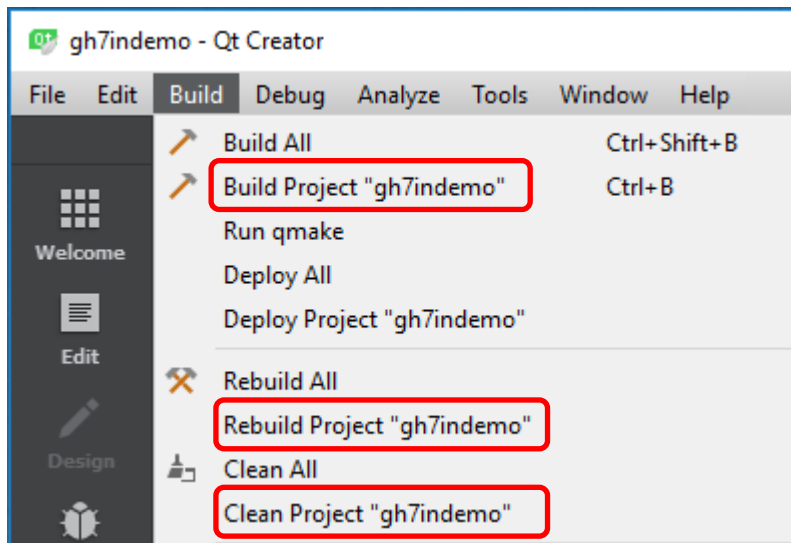




- **Save!** File → Save All



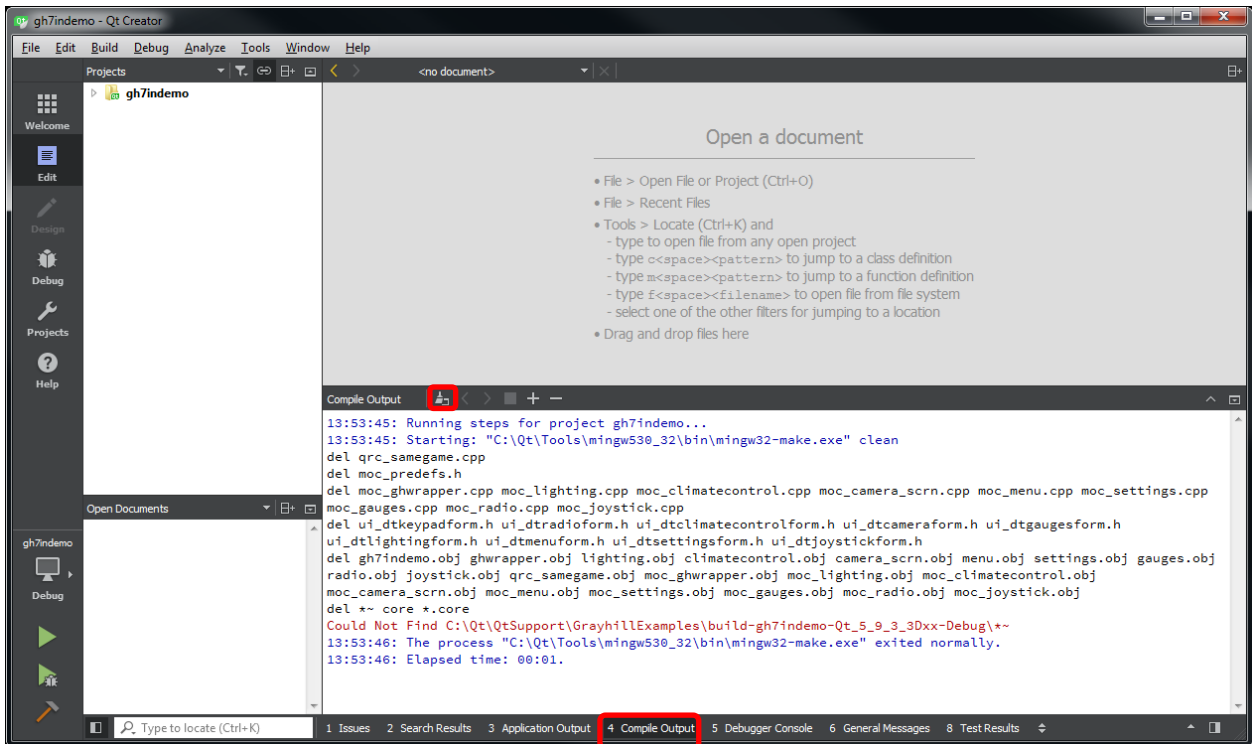
- Build the image for the target



- **Build** Let Qt Creator decide what is out of date
- **Rebuild** Force Qt creator to re-compile everything
- **Clean** Remove all the existing artifacts generated by previous builds

- Select Build → Clean Project “gh7indemo”

The bottom ribbon of Qt Creator has various panes (views) that can be examined. Click on “4 Compile Output). Note: image is shown post click; so the results and actions of the clean are shown.



- Click on the paintbrush icon to clear the contents

- Next, select Build → Build Project “gh7indemo”

The following illustrates the last few lines in “Compile Output”

```

-D_REENTRANT -fPIC -DON_HARDWARE -DQT_QML_DEBUG -DQT_QUICK_LIB -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_QML_LIB -
DQT_NETWORK_LIB -DQT_CORE_LIB -I..\\gh7indemo -I. -I..\\gh7indemo -I..\\targetSysroot\\kernel-headers\\include -I..
\\.\\targetSysroot\\usr\\include -I..\\..\\..\\QtLibrarySrc\\build\\targetInstallDir\\include -I..\\..\\..\\QtLibrarySrc
\\build\\targetInstallDir\\include\\QtQuick -I..\\..\\..\\QtLibrarySrc\\build\\targetInstallDir\\include\\QtWidgets -I..\\
\\.\\QtLibrarySrc\\build\\targetInstallDir\\include\\QtGui -I..\\..\\..\\QtLibrarySrc\\build\\targetInstallDir\\include
\\QtQml -I..\\..\\..\\QtLibrarySrc\\build\\targetInstallDir\\include\\QtNetwork -I..\\..\\..\\QtLibrarySrc\\build
\\targetInstallDir\\include\\QtCore -I. -I. -I/include -I/usr/include -I..\\..\\..\\QtLibrarySrc\\build\\targetInstallDir
\\mkspecs\\devices\\linux-imx6-g++ -o moc_joystick.obj moc_joystick.cpp
C:\\Qt\\QtLibrarySrc\\build4\\..\\..\\QtSupport\\gcc-linaro-2013\\bin\\arm-linux-gnueabi-g++ -Wl, -rpath=/usr/lib -Wl, -
rpath=/lib -mfloat-abi=softfp -Wl, -rpath,/usr/local/Qt-5.9.3/lib -o gh7indemo gh7indemo.obj ghwrapper.obj
lighting.obj climatecontrol.obj camera_scrn.obj menu.obj settings.obj gauges.obj radio.obj joystick.obj
qrc_samegame.obj moc_ghwrapper.obj moc_lighting.obj moc_climatecontrol.obj moc_camera_scrn.obj moc_menu.obj
moc_settings.obj moc_gauges.obj moc_radio.obj moc_joystick.obj -L/lib -L/usr/lib -LC:\\Qt\\QtSupport
\\targetSysroot\\usr\\lib -lghdrv -lrt -lghio -LC:\\Qt\\QtLibrarySrc\\build\\targetInstallDir\\lib -lQt5Quick -LC:\\Qt
\\QtSupport\\targetSysroot\\lib -lQt5Widgets -lQt5Gui -lQt5Qml -lQt5Network -lQt5Core -lGLESw2 -lEGL -lGAL -lpthread
13:59:25: The process "C:\\Qt\\Tools\\mingw530_32\\bin\\mingw32-make.exe" exited normally.
13:59:25: Elapsed time: 00:59.
  
```

Note: When there are errors, they are also highlighted/summarized in the “Issues” tab.

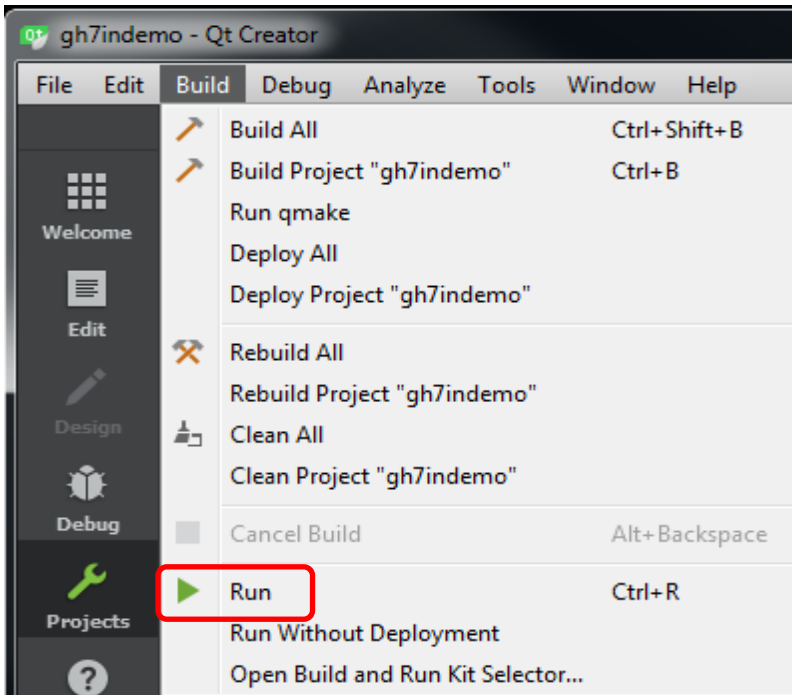
Deployment (running the compiled image on the target) can also be accomplished multiple ways

- Using the green triangle on the left hand side

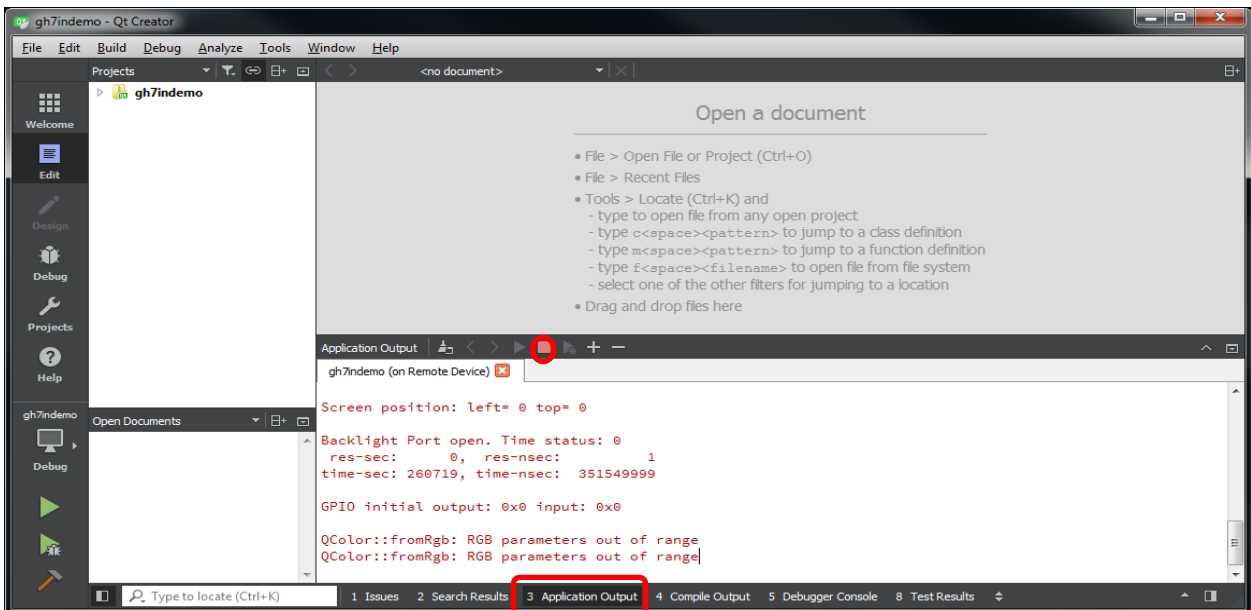
```

14:04:07: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\NHZ_menu.png"...
14:04:07: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\NHZ_radio.png"...
14:04:07: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\NHZ_settings.png"...
14:04:08: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\Open+Pipe+Symphony.mp3"...
14:04:08: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\powerButton.png"...
14:04:08: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\RedRabbit84x78.png"...
14:04:08: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\RedRabbitShot84x78.png"...
14:04:08: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\gh7indemo\\images\\target467x467.png"...
14:04:08: Uploading file "C:\\Qt\\QtSupport\\GrayhillExamples\\build-gh7indemo-Qt_5_9_3_3Dxx-Debug
\\gh7indemo"...
14:04:09: All files successfully deployed.
14:04:09: Deploy step finished.
14:04:09: Elapsed time: 00:05.
  
```

- From the Build menu



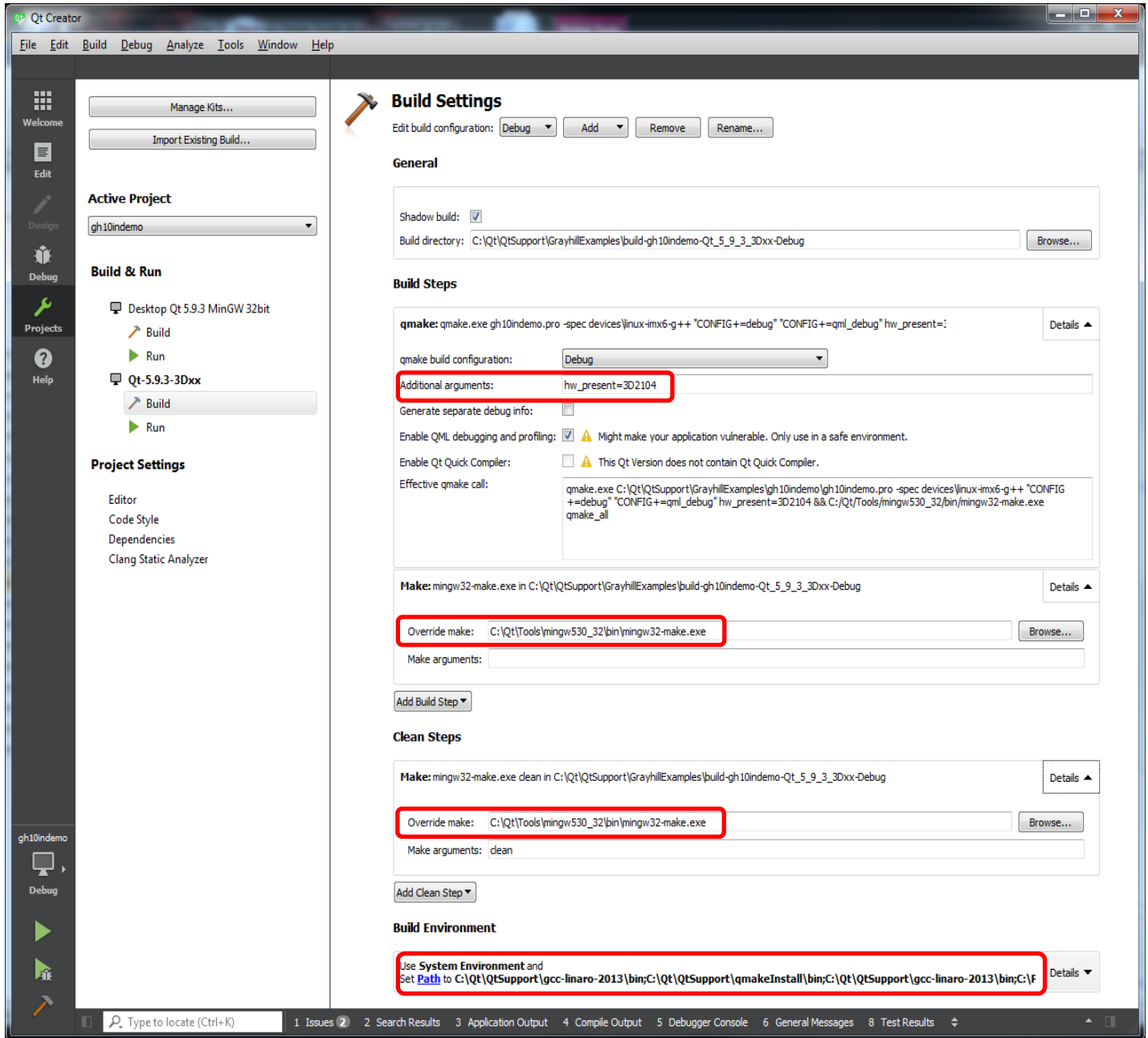
- Keyboard short-cut (see Run above - <Ctrl-r>)



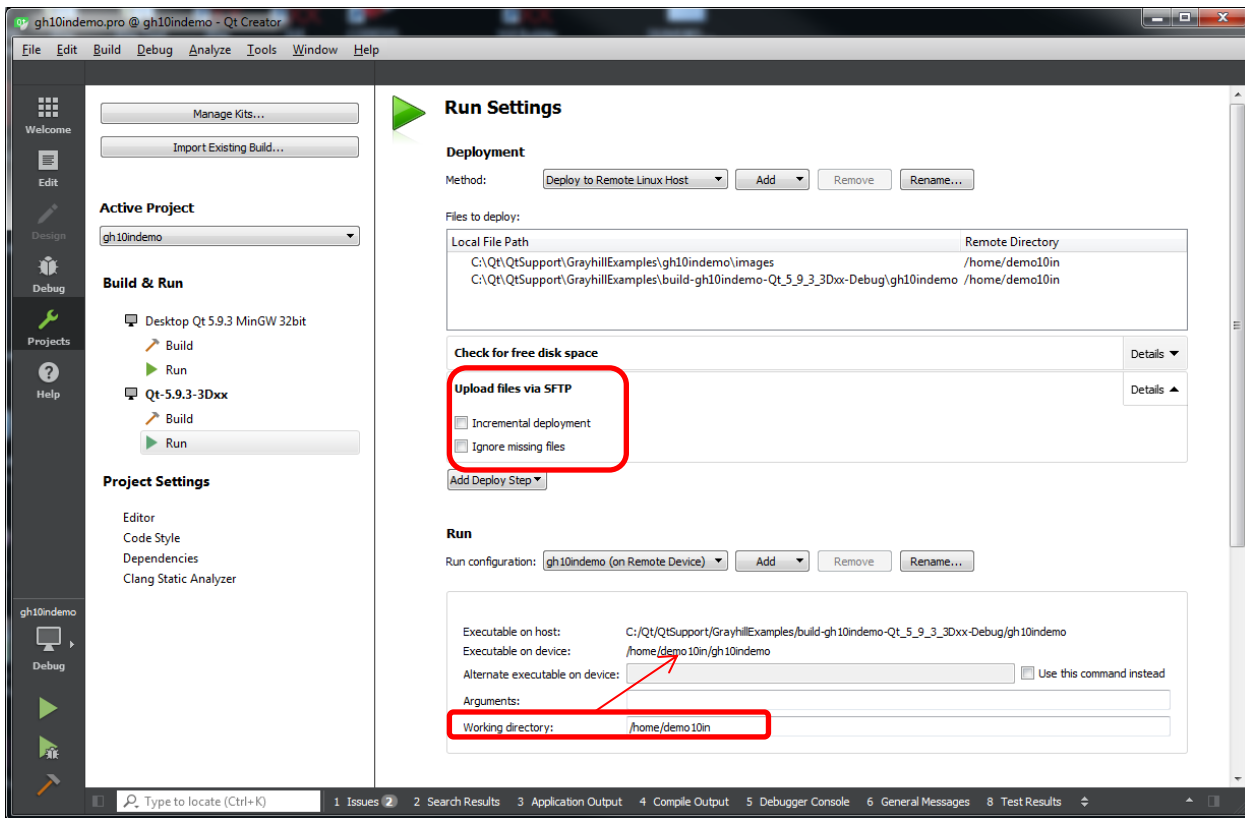
Switch (by selecting) to the “Application Output” tab; this is where qDebug messages are output.

Click the red square to terminate the target session.

## Quick Reference



- Build Steps
    - qmake
    - Make
  - Clean Steps
    - Make
  - Build Environment
    - Path
- Additional arguments → hw\_present=<display model>  
 Override make → C:\Qt\Tools\mingw530\_32\bin\mingw32-make.exe
- Override make → C:\Qt\Tools\mingw530\_32\bin\mingw32-make.exe
- Append ;C:\Qt\5.9.3\mingw53\_32\bin

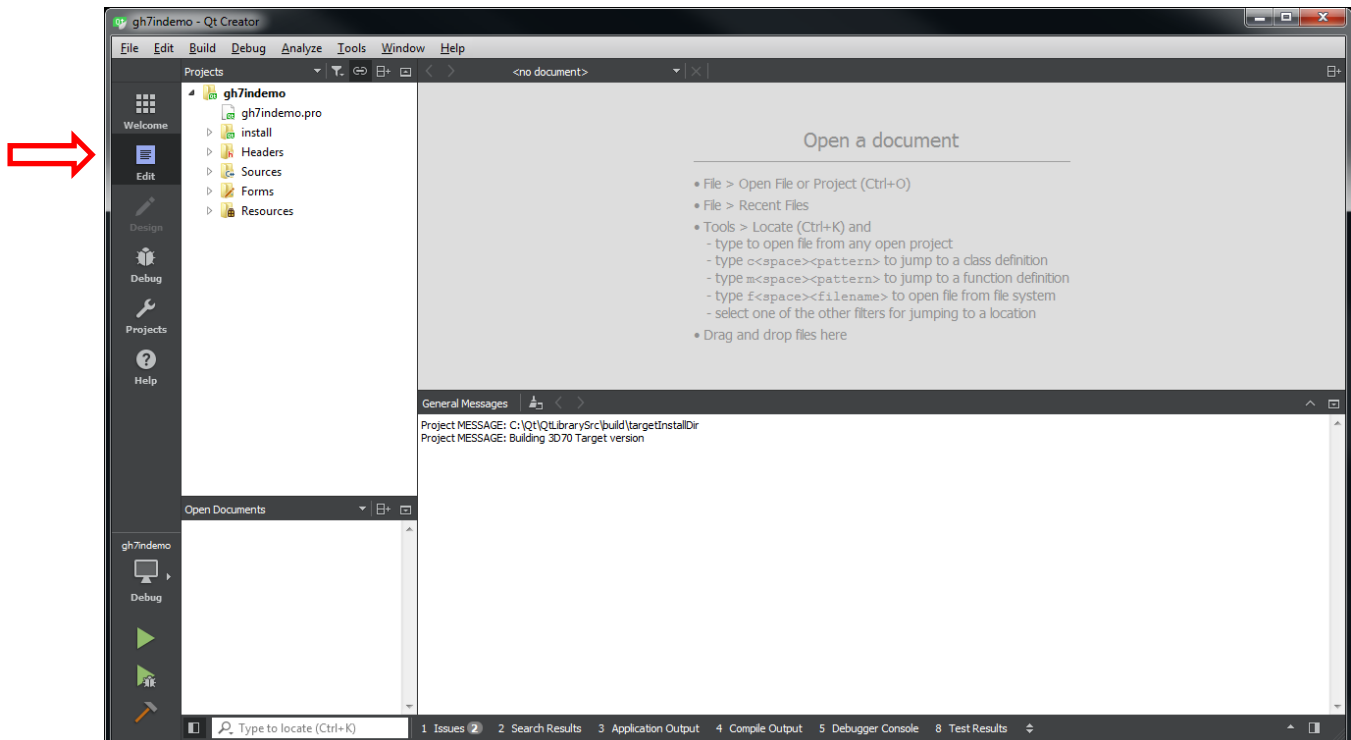


- Deployment
  - Upload files via SFTP                      unselect Incremental deployment
  - unselect Ignore missing files
- Run
  - Working directory                      /home/<path to executable image on display>

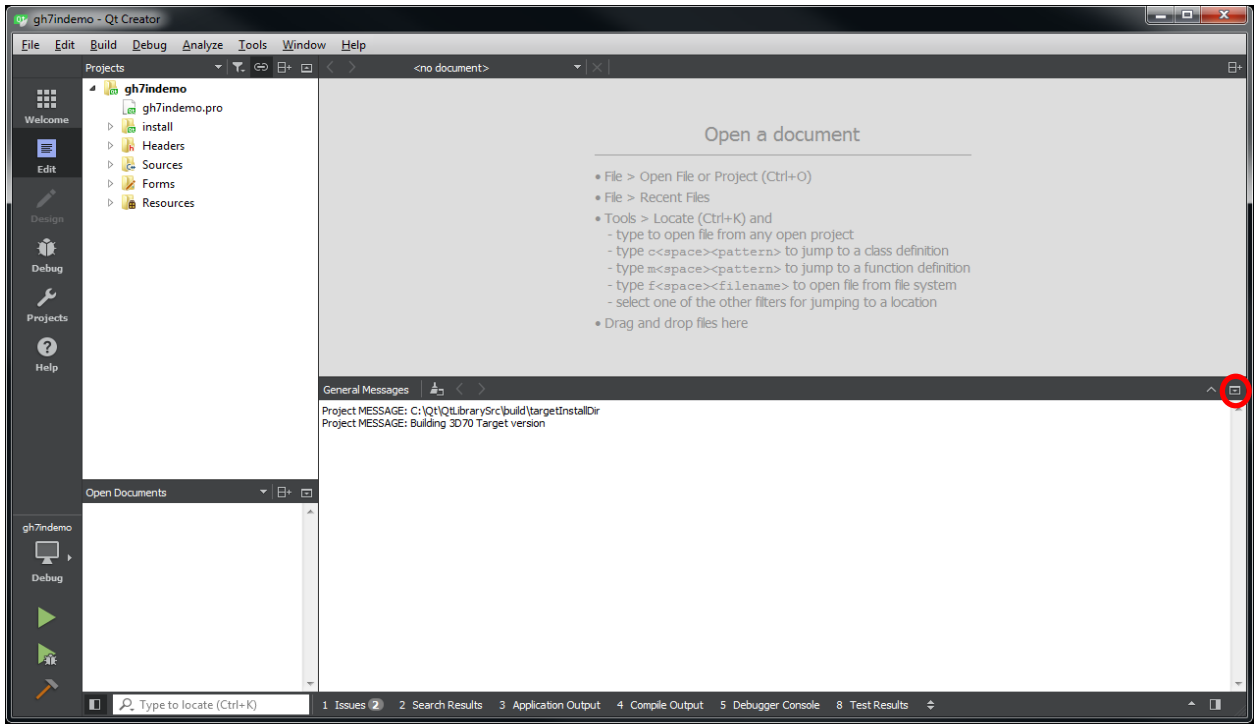
## Appendix C: Debugging

Let's face it; code never initially does what it is *supposed* to do; but rather what it was **told** to do!  
Luckily Qt Creator has a built-in debugger.

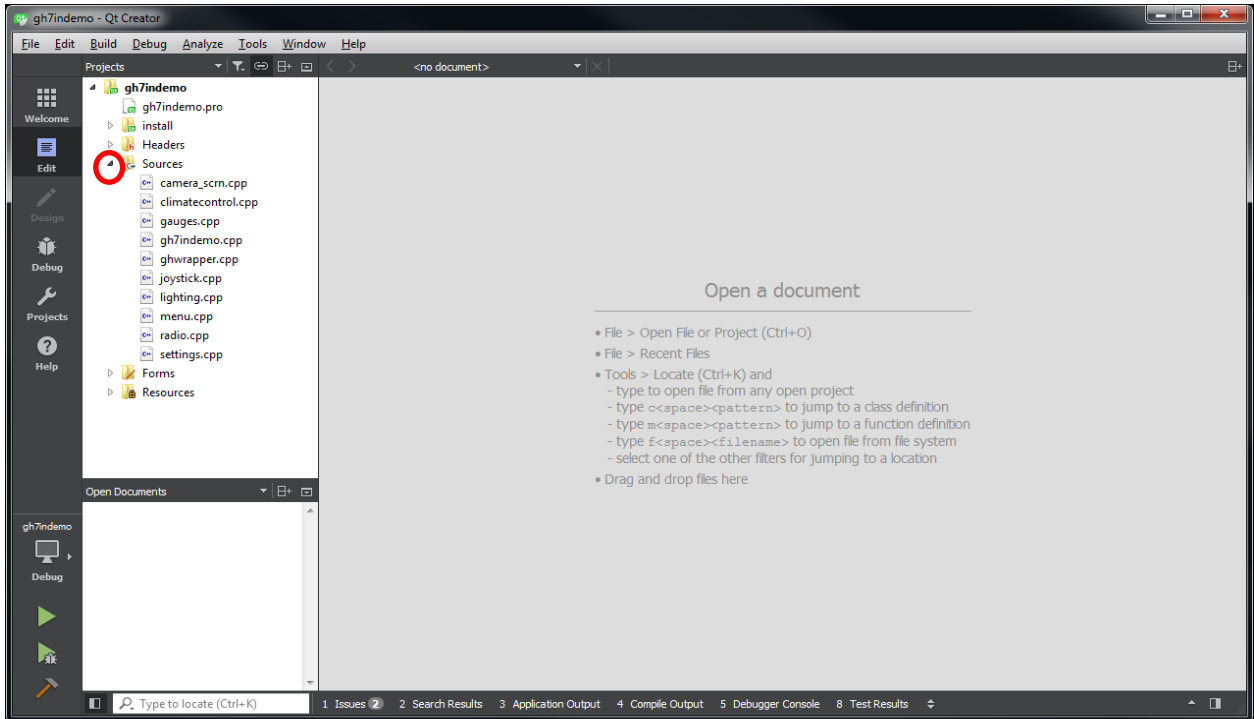
- First set a breakpoint
  - Load gh7indemo
  - Select the “Edit” view
  - Expand contents of gh7indemo



- Close the current open pane (screen shot illustrates “General Messages”)

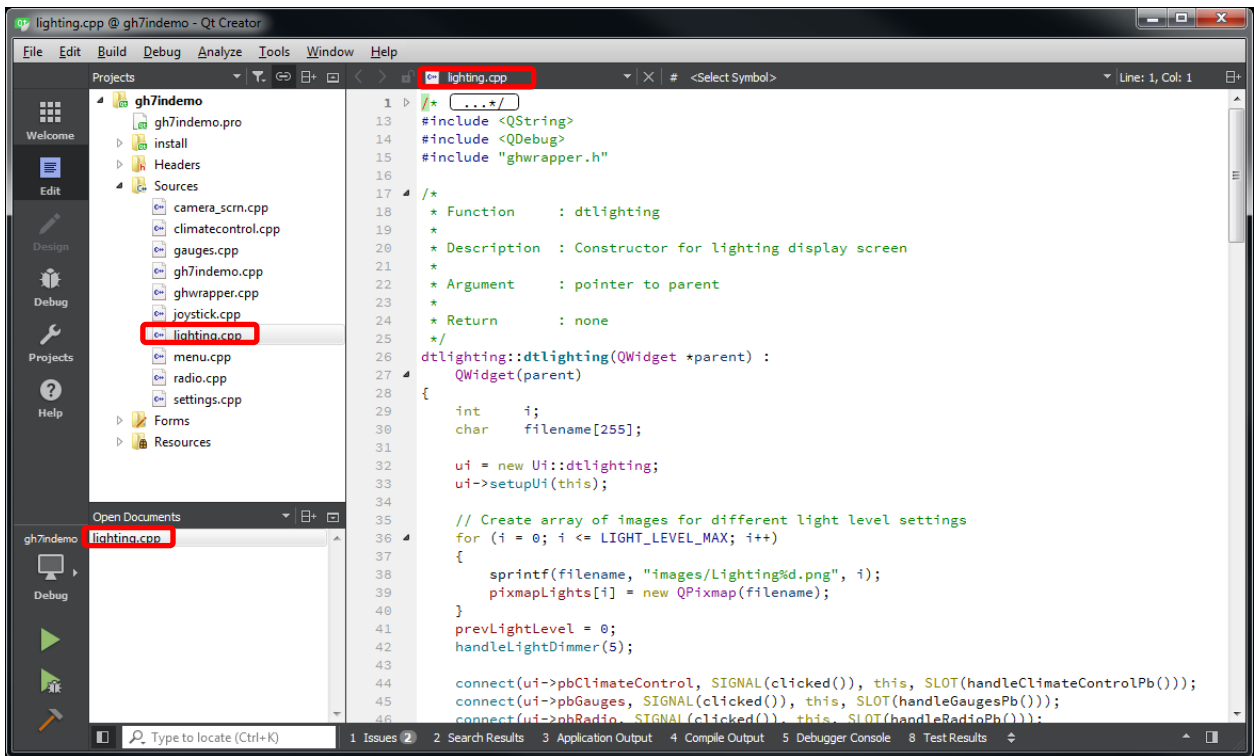


- Expand the “Sources” folder under the project file list





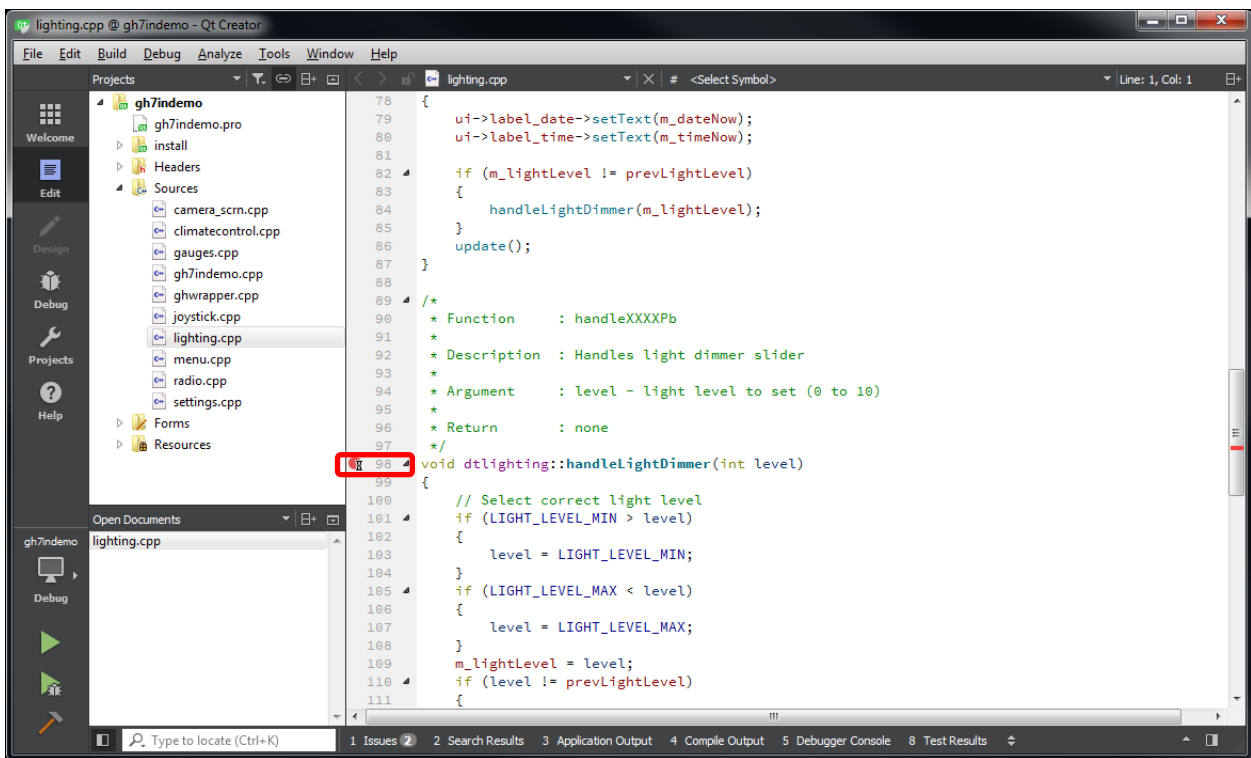
- Select the desired file; under “Sources” select “lighting.cpp” by double clicking



The file being displayed (edited) is shown in the “Open Documents” section as well as on the top of the editor pane. Additional open files can be selected by either selecting them from “Open Documents” or the up/down triangular arrows to the right of the file name. Also, the X to their right will close the file.

Select the line of code to set the breakpoint. N.B. The editor is not context aware; so it is possible to set a breakpoint on a commented out line.

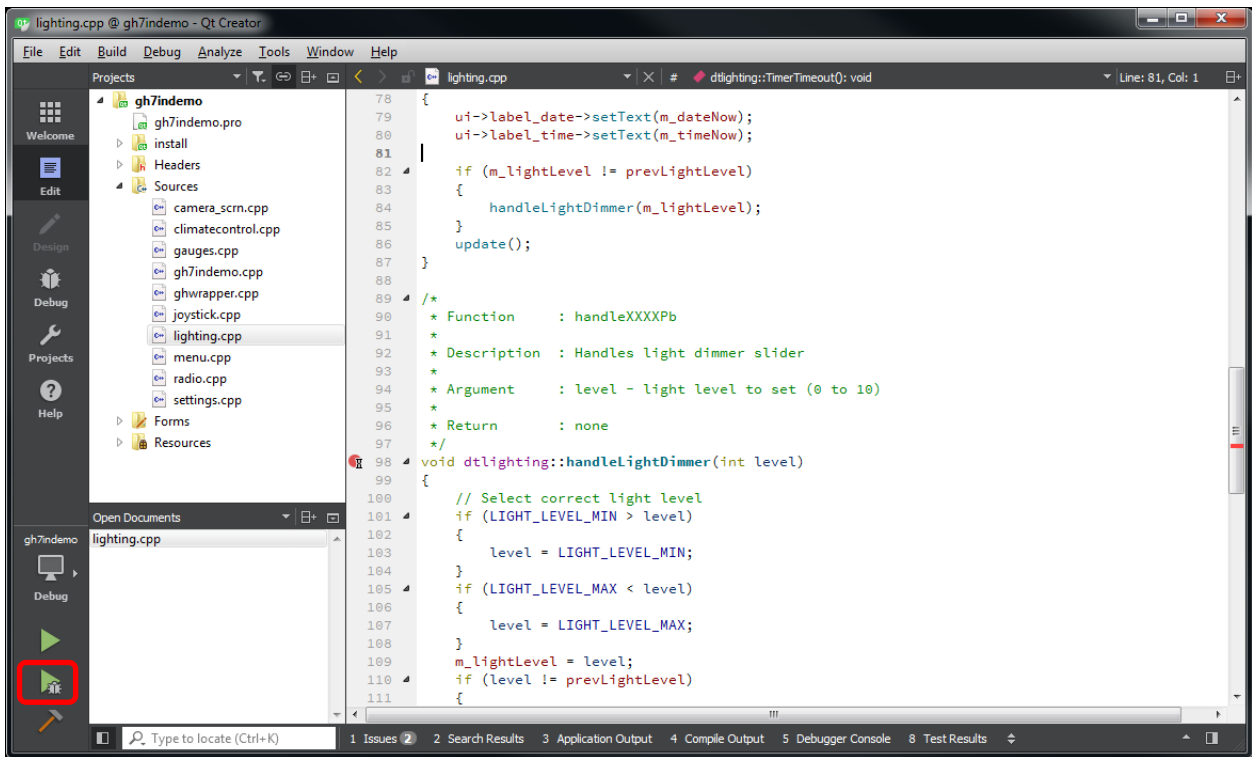
- Scroll down to line 98 (“void dtlighting::handleLightDimmer (int level)”)
- Left click on the mouse to the left of the line number; a red circle will appear



- Notice the scroll bar gutter indicates the relative location of the breakpoint in the file.
- Save

- Click on the green arrow like “Run” from above; but with the homely lady bug.

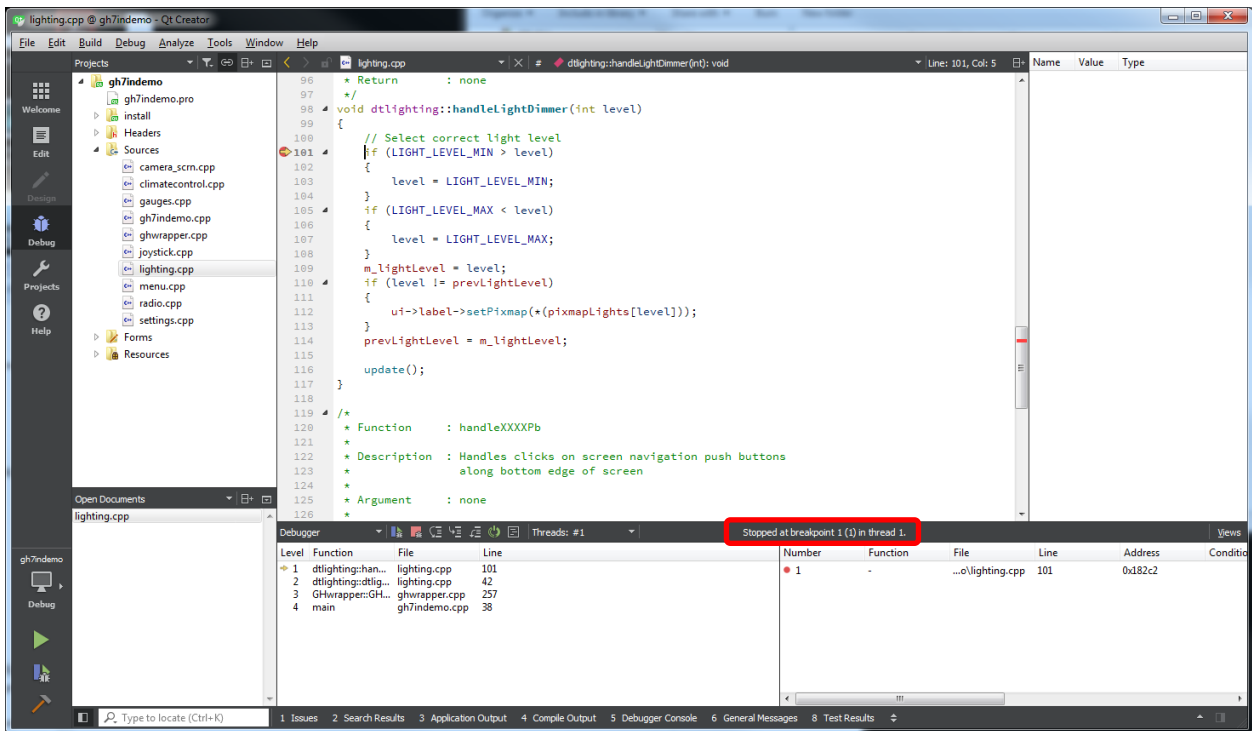
N.B. This may cause the project to be re-compiled if the initial build was not configured for debug.



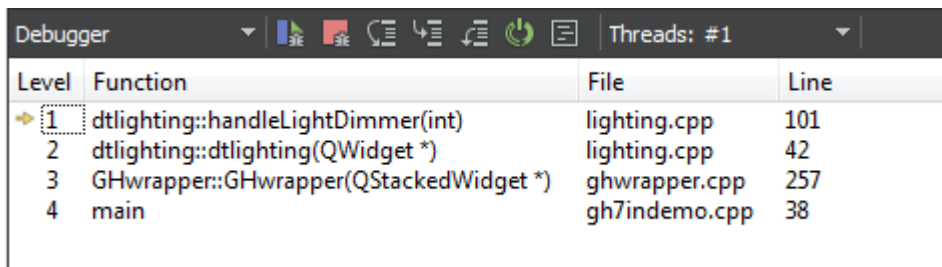
The code begins execution and quickly hits the breakpoint.

Note that the display has not been updated yet. The method (handleLightDimmer) is invoked during the class creation – line 42 handleLightDimmer(5);

Lastly, note the breakpoint is actually at line 101; the first executable statement within the function/method.



- The debugger pane illustrates the calling tree



- Clicking on line 2 jumps to the aforementioned caller
- Debugger stepping option menu



The debugger features the usual (Mouse over the icons for a description)

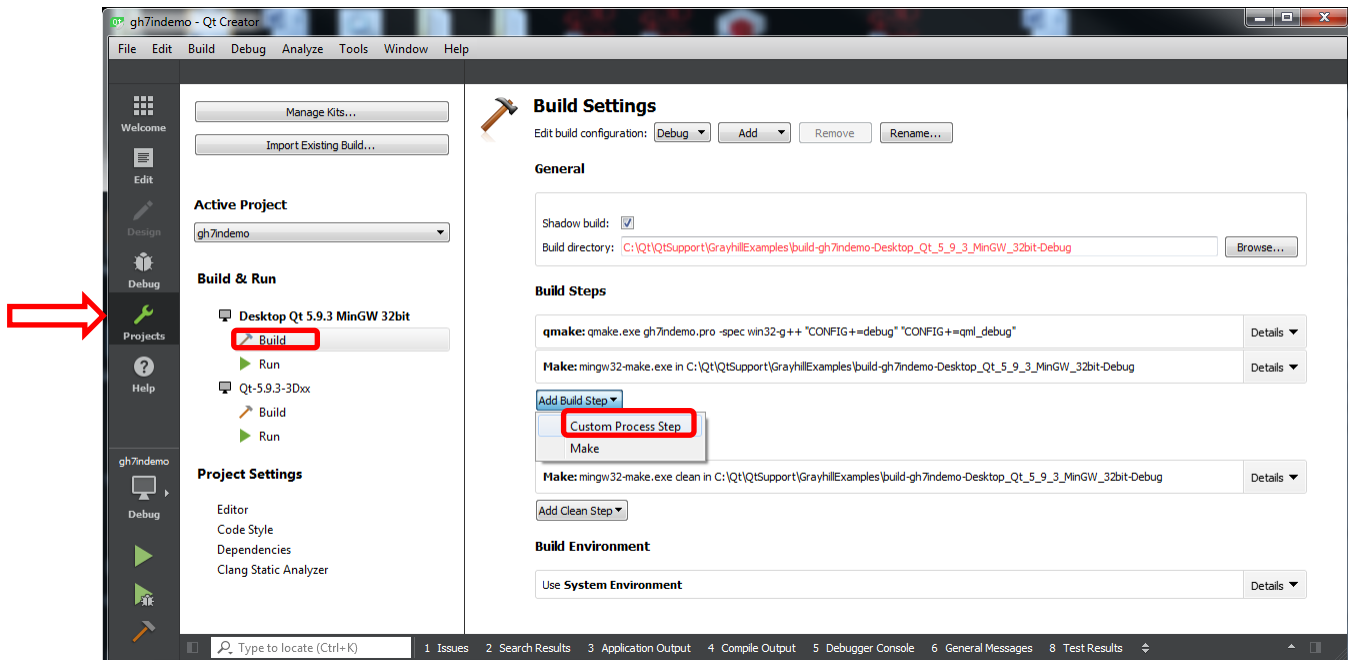
- Step Over <F10>
- Step In <F11>
- Step Out <Shift> + <F11>

## Appendix D: Build and Run 3Dxx Desktop Application

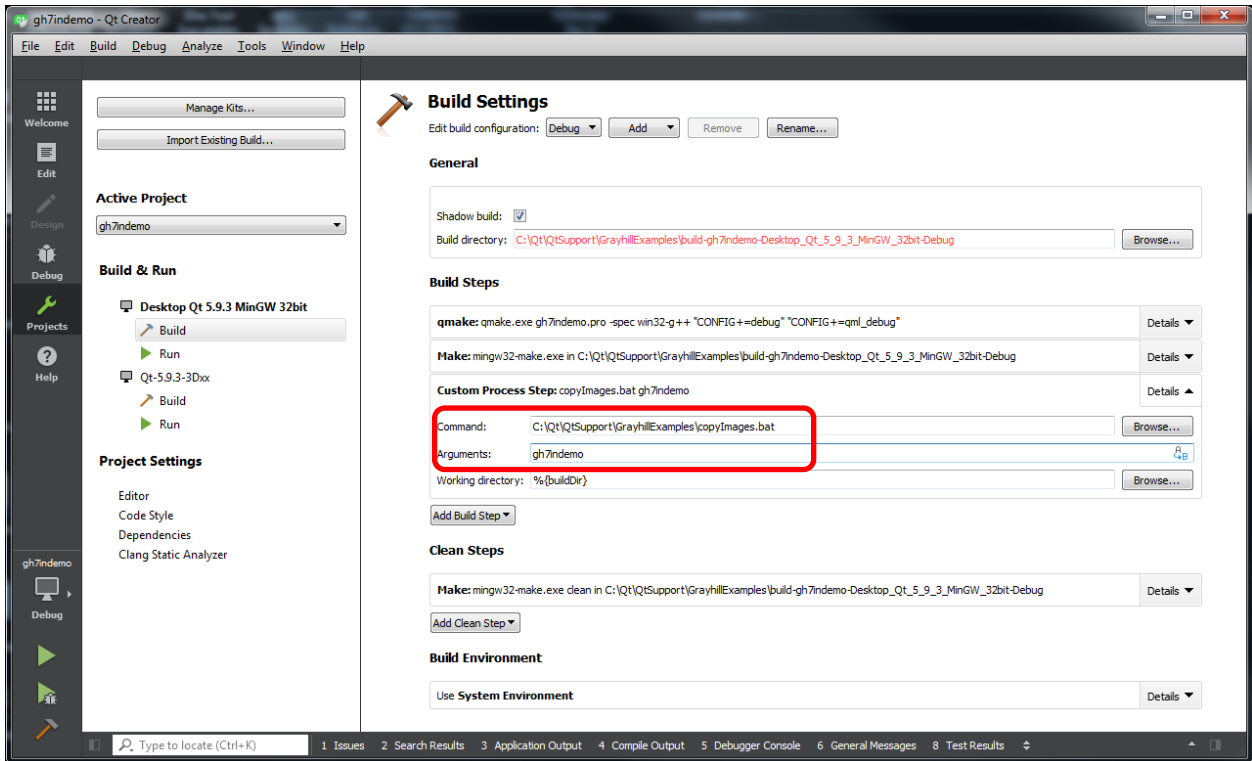
- Select “Projects”
- Select “Build” under “Desktop Qt 5.9.3 MinGw 32bit”

The following steps facilitate the copying of the necessary image files into the desktop simulation folder.

- Click “Add Build Step” → “Custom Process Step”



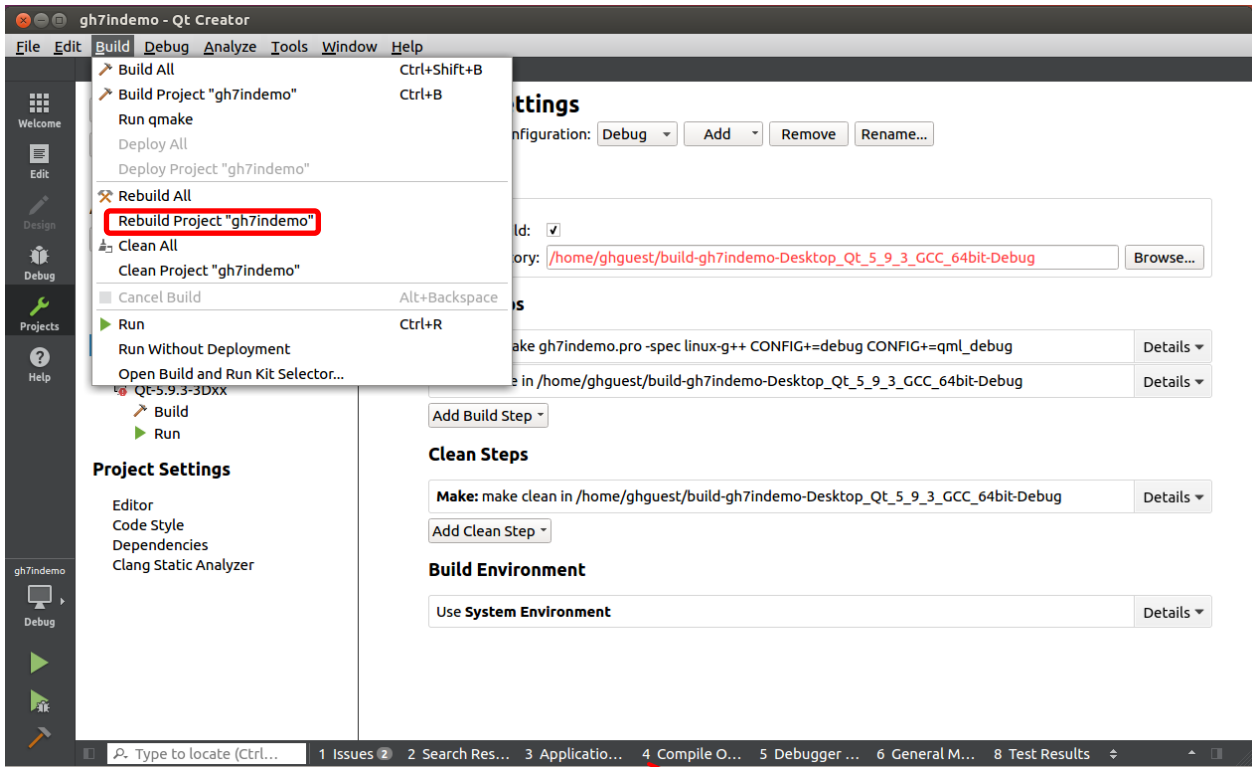
- Command `C:\QtGhSupport\GrayhillExamples\copyImages.bat`
- Arguments `<project>` e.g. `gh7indemo`



- File → Save All

- Select “Rebuild Project “gh7indemo” from the “Build” menu

N.B. It may take a few seconds to refresh the menu options.



Click on the “Compile Output” and “Issues” selectors on the bottom of the Qt Creator window to check for error messages and problems.

The desktop version can now be run by clicking on the big green “Run” arrow on the lower left corner of the Qt Creator window.

Click on the “Application Output” item on the bottom row to view application output.

Click on red square on “Application Output” window to stop application.

## Appendix E: Build and Run QML Demonstration Program

The following steps illustrate how to build and run the QML demonstration program “Samegame”.

- From Qt Creator open the “samegame” project. (Select “Welcome” to go to home screen)
- Select desired kit
- Update “Additional arguments”: under “Build Steps” “Details” to reflect the proper hardware
- Select “Build->Rebuild All” to build program
- Click on the green arrow “Run” button to run program



## Appendix F: Setting up a 3Dxx Qt Program to Run at Boot Up

This section describes how to configure a program to automatically execute at boot up.

- Open a terminal window on the target (Error! Reference source not found. describes how to launch “PuTTY”)
- Create a launch script for the desired application
 

<ul style="list-style-type: none"> <li>○ cd /etc/init.d</li> <li>○ echo “#!/bin/sh -l</li> <li>○ cd /home/demo7in</li> <li>○ /home/demo7in/gh7indemo &amp;” &gt; launchQtApp</li> <li>○ cat launchQtApp</li> <li>○ chmod 755 launchQtApp</li> </ul>	<p style="text-align: center;"><b><u>Explanation</u></b></p> <p><i>set into proper directory</i></p> <p><i>treat as login (runs profile)</i></p> <p><i>set directory for images</i></p> <p><i>spawn application process</i></p> <p><i>verify contents</i></p> <p><i>make script executable</i></p>
---	--

```

COM1 serial port to target
root@ghiiimx6:~# cd /etc/init.d
root@ghiiimx6:/etc/init.d# echo "#!/bin/sh -l
>cd /home/demo7in
>/home/demo7in/gh7indemo &" > launchQtApp
root@ghiiimx6:/etc/init.d# cat launchQtApp
#!/bin/sh -l
cd /home/demo7in
/home/demo7in/gh7indemo &
root@ghiiimx6:/etc/init.d# chmod 755 launchQtApp
root@ghiiimx6:/etc/init.d#
  
```

- Create a link to the launch script created above
 

<ul style="list-style-type: none"> <li>○ cd /etc/rc.d</li> <li>○ ln -s /etc/init.d/launchQtApp S12qtApp</li> <li>○ ls -l S12qtApp</li> </ul>	<p><i>set into proper directory</i></p> <p><i>create soft link to executable file</i></p> <p><i>verify link creation</i></p>
--	--

```

COM1 serial port to target
root@ghiiimx6:~# cd /etc/init.d
root@ghiiimx6:/etc/init.d# echo "#!/bin/sh -l
>cd /home/demo7in
>/home/demo7in/gh7indemo &" > launchQtApp
root@ghiiimx6:/etc/init.d# cat launchQtApp
#!/bin/sh -l
cd /home/demo7in
/home/demo7in/gh7indemo &
root@ghiiimx6:/etc/init.d# chmod 755 launchQtApp
root@ghiiimx6:/etc/init.d# cd /etc/rc.d
root@ghiiimx6:/etc/rc.d# ln -s /etc/init.d/launchQtApp S12qtApp
root@ghiiimx6:/etc/rc.d# ls -l S12qtApp
lrwxrwxrwx 1 root root 23 Jan 18 13:02 S12qtApp -> /etc/init.d/launchQtApp
root@ghiiimx6:/etc/rc.d#
  
```

**Note:** Do not try to launch multiple Qt applications at boot up or try to launch the ghvehicleapp application along with a Qt application as they will conflict with one another.

**Note:** When switching from running one application to another, even between Qt applications, it is a good idea to do a reboot of the 3Dxx Display in between to make sure that the hardware is properly reset. This can be done by entering the “reboot” command on the 3Dxx Display Linux console.

## Appendix G: Interfacing 3Dxx Hardware from QT Software

The 3Dxx Display contains the following custom component interfaces:

- LCD
- LCD Backlight
- Camera driver
- CAN driver
- Digital I/O driver
- Analog Input driver (Model 3D70 only)
- Buzzer (Models 3D70, 3D2104)
- Audio Output (Model 3D70 only)

This section explains how to access the functionality of these components. The programming interfaces and provided API functions are covered, with the syntax and parameters defined. Sample code is also provided where appropriate.

### LCD

The Grayhill 3Dxx Series Display uses a 16 bit per pixel LCD screen. The pixel dimensions of various 3Dxx Display products are shown in the section Supported Hardware Products. The default orientation of the frame buffer is landscape mode (wider pixel dimension is in horizontal direction).

### LCD Backlight

The LCD Backlight setting is a value between 0 (minimum) and 100 (maximum) inclusive. The brightness value can be set in the file `/sys/class/backlight/pwm-backlight.0/brightness`

**Sample Code:**

```
int value = 80;
QFile file("/sys/class/backlight/pwm-backlight.0/brightness");
if (file.open(QIODevice::WriteOnly | QIODevice::Text))
{
    QTextStream out(&file);
    out << value;
    file.close();
}
```

## Camera Driver Interface

The Grayhill 3Dxx Display device can contain multiple camera inputs. NTSC and PAL format video inputs are supported by modifying the camera input sensor parameters. The camera output can be displayed on the LCD. The following camera display parameters can be modified:

- Window parameters – window size and window position
- Color parameters – brightness, contrast, saturation and hue
- Rotation
- Input sensor parameters – provides support for NTSC and PAL formats
- Camera output to LCD foreground or background with color key

Camera output is displayed at 30fps.

**Note:** Only one camera input can be active at a time.

### Interface:

The Qt application can interface with the Camera driver using the Camera class.

### Data Types:

```
typedef struct _SENSORPARAMS // Must be set according to camera input
type
{
    // NTSC    PAL
    unsigned int top;        // 4        5
    unsigned int left;       // 0        4
    unsigned int height;     // 480     567
    unsigned int width;      // 640     640
} SENSORPARAMS, *PSENSORPARAMS;

#define FOREGROUND (1)
#define BACKGROUND (0)

// These are the only allowed values for VIDEO_COLOR_KEY_XXX:
#define VIDEO_COLOR_KEY_BLACK (0x00000000)
#define VIDEO_COLOR_KEY_RED (0x00FF0000)
#define VIDEO_COLOR_KEY_GREEN (0x0000FF00)
#define VIDEO_COLOR_KEY_BLUE (0x000000FF)
#define VIDEO_COLOR_KEY_YELLOW (0x00FFFF00)
#define VIDEO_COLOR_KEY_CYAN (0x0000FFFF)
#define VIDEO_COLOR_KEY_MAGENTA (0x00FF00FF)
#define VIDEO_COLOR_KEY_WHITE (0x00FFFFFF)

typedef struct _DISPLAYPARAMS
{
    unsigned int top;        // top left window y-coordinate
    unsigned int left;       // top left window x-coordinate
                                // (must be divisible by 4)
```

```
    unsigned int height; // window vertical size
    unsigned int width;  // window horizontal size
                        // NOTE: top + height must not exceed height of
display
                        // and left + width must not exceed display width
    unsigned int rotate; // 0-7, see below
    unsigned int fg;     // FOREGROUND or BACKGROUND +
VIDEO_COLOR_KEY_XXX
} DISPLAYPARAMS, *PDISPLAYPARAMS;
```

The camera output always operates in native landscape mode. Use the following rotation values to support other display and camera orientations:

Value	Rotation
0	No rotation
1	Vertical flip
2	Horizontal flip
3	180
4	90 right
5	90 right with vertical flip
6	90 right with horizontal flip
7	90 left

```
#define HUE_CODE_00      (0x00)
#define HUE_CODE_7F     (0x7F)
#define HUE_CODE_80     (0x80)

typedef struct _COLORPARAMS
{
    unsigned int brightness; // 0-255
    unsigned int saturation; // 0-255
    unsigned int hue;        // HUE_CODE_00, HUE_CODE_7F, or
HUE_CODE_80
    unsigned int contrast;   // 0-255
} COLORPARAMS, *PCOLORPARAMS;
```

### Function Prototypes:

#### Camera::Camera

Camera class constructor

#### Syntax

```
Camera::Camera (int camnum, int fbdev = FB_DEV_0);
```

## Parameters

int camnum  
[in]

Camera Number. Valid range 1-2 for Model 3D50, 1-3 for Model 3D70, 1-4 for Model 3D2104

```
#define FB_DEV_0 (0) // GRAPHICS being sent to /dev/fb0  
#define FB_DEV_1 (1) // GRAPHICS being sent to /dev/fb1
```

int fbdev  
[in]

The "fbdev" value must indicate whether the GRAPHICS are being sent to fb0 or fb1. When GRAPHICS are being sent to fb0, then video will be sent to fb1 and only foreground mode is allowed. This is the default assumed if "fbdev" is missing.

If GRAPHICS are being sent to fb1, then video will be sent to fb0 and both foreground and background modes are supported. In order to send GRAPHICS to fb1, add this parameter to the command line that launches Qt: -display LinuxFb:/dev/fb1

## Return Value

none

## Camera::setdisplayparams

Sets the following display window parameters

- origin
- window size
- rotation
- foreground or background with color key (When using background mode the camera video only shows through where the graphics data is set to the color that matches the specified color key. Graphics of any other color will appear on top of the camera video image.)

## Syntax

```
int Camera::setdisplayparams(PDISPLAYPARAMS p);
```

## Parameters

PDISPLAYPARAMS p  
[in]  
refer to DISPLAYPARAMS structure

## Return Value

int  
0 indicates success, -1 indicates failure

## Camera::setcolorparams

Sets the following camera color parameters

- Brightness

- Saturation
- Contrast
- Hue

**Syntax**

```
int Camera::setcolorparams(PCOLORPARAMS p);
```

**Parameters**

PCOLORPARAMS p  
[in]  
refer to COLORPARAMS structure

**Return Value**

int  
0 indicates success, -1 indicates failure

**Camera::setsensorparams**

Sets the camera sensor parameters

**Syntax**

```
int Camera::setsensorparams(PSENSORPARAMS psensor);
```

**Parameters**

PSENSORPARAMS psensor  
[in]  
refer to SENSORPARAMS structure

**Return Value**

int  
always returns 0

**Camera::show**

Enables or disables the camera

**Syntax**

```
int Camera::show(int enable);
```

**Parameters**

int enable  
[in]  
1 = enable, 0 = disable

**Return Value**

int  
0 indicates success, -1 indicates failure

### Required Files:

Header File: camera.h

Link Library : libghdrv.so

### Sample Code:

```
#include "camera.h"

COLORPARAMS color;
DISPLAYPARAMS disp;
int cameranum = 1; // camera input 1

Camera cam(cameranum);

disp.top = 0;
disp.left = 80;
disp.height = 480;
disp.width = 640;
disp.rotate = 4; // rotate 90 degree right
disp.fg = FOREGROUND;
// configure display parameters
cam.setdisplayparams(&disp);

// start camera
cam.show(1);

// change color parameters
color.brightness = 50;
color.saturation = 128;
color.contrast = 128;
color.hue = 0;
// configure color parameters
cam.setcolorparams(&color);

....

// stop 1+camera
cam.show(0);
```

## CAN Driver Interface

The 3D50 and 3D70 Displays includes two CAN controller modules. Available CAN ports are CAN1 and CAN2. The 3D2104 Display includes three CAN controller modules. Available CAN ports are CAN1, CAN2, and CAN3. The CAN controller supports both standard and extended frames.

## Interface:

The Qt demo application can interface with the CAN bus driver using the CAN class.

## Data Types:

```
/* special flag bits for the CAN_ID */
#define CAN_EFF_FLAG 0x80000000U /* EFF flag (add to ID to activate 29-bit ID) */
#define CAN_RTR_FLAG 0x40000000U /* remote transmission request */
#define CAN_ERR_FLAG 0x20000000U /* error frame */

struct _CANMSG
{
    unsigned int ID;
    unsigned int Length; // Data Length Code of the Msg (0..8)
    unsigned char Data[8];
};
typedef struct _CANMSG CANMSG, *PCANMSG;
```

## Function Prototypes:

### CAN::CAN

CAN class constructor

#### Syntax

```
CAN::CAN(int num);
```

#### Parameters

int num  
[in]

CAN Port Number. Valid range 1-2 for Models 3D50, 3D70; 1-3 for Model 3D2104

#### Return Value

none

### CAN::OpenPort

Opens the CAN socket

#### Syntax

```
int CAN::OpenPort(void);
```

#### Parameters

none

#### Return Value



int  
non-zero value indicates success, -1 indicates failure

### **CAN::WritePort**

Writes a single CAN frame to the CAN port.

#### **Syntax**

```
int CAN::WritePort(PCANMSG TxMsg);
```

#### **Parameters**

PCANMSG TxMsg  
[in]  
Contains the CAN frame to be written

#### **Return Value**

int  
0 indicates success, -1 indicates failure

### **CAN::ReadPort**

Attempts to read a single CAN frame from the CAN port. Note that the CAN socket is configured to be non-blocking, so calls to ReadPort will return even if there is no data.

#### **Syntax**

```
int CAN::ReadPort(PCANMSG RxMsg);
```

#### **Parameters**

PCANMSG RxMsg  
[out]  
Contains the CAN frame received

#### **Return Value**

int  
contains the number of bytes read, -1 indicates failure

### **CAN::ClosePort**

Closes the CAN socket

#### **Syntax**

```
void CAN::ClosePort(void);
```

#### **Parameters**

none

#### **Return Value**

none

### Required Files:

Header File: can.h  
Link Library : libghdrv.so

### Sample Code:

```
#include "can.h"

CANMSG TxMsg;
CANMSG RxMsg;
int bytesread = 0;
int cannum = 1;      // CAN1

/* Init TX and RX message */
TxMsg.ID = 0x23;
TxMsg.Length = 8;
for (int i=0; i<8; i++)
    TxMsg.Data[i] = (0x11 * (i+1));    // fill random data
memset((void *)&RxMsg, 0, sizeof(CANMSG));

// CAN1
CAN can(cannum);
can.OpenPort();
can.WritePort(&TxMsg);
do
{
    bytesread = can.ReadPort(&RxMsg);
    // add delay
} while (bytesread != sizeof(CANMSG));
can.ClosePort();
```

## Digital I/O Driver Interface

The Model 3D50 Display, Model 3D70 Display, and Model 3D2104 Display each have four digital inputs and four digital outputs, but they are configured differently and these differences will be explained. Each device uses the same library calls to read the digital inputs and set the digital outputs.

On the 3D50 Five Inch Display Pin 4 on its connector is a dedicated input only pin. Pin 5 is a dedicated output only pin. Pins 6, 7, and 8 are shared I/O pins that can be used to output a signal or input a signal.

On the Model 3D70 Seven Inch Display each of the four inputs are dedicated and so operate independently of any output pins.

On the Model 3D2104 10.4 Inch Display all digital output pins are shared I/O pins that can be used to output a signal or input a signal.

For a shared I/O pin to function as an input, the corresponding output must be set low.

The following table summarizes all of the digital I/O pins for each model:

Model 3D50 Pins	Model 3D70 Pins	Model 3D2104 Pins
Input 1 (Pin 4)	Input 1 (Pin 4 Connector A)	Input 1 or Output 1 (Pin 10)
Input 2 or Output 2 (Pin 6)	Input 2 (Pin 8 Connector B)	Input 2 or Output 2 (Pin 21)
Input 3 or Output 3 (Pin 7)	Input 3 (Pin 9 Connector B)	Input 3 or Output 3 (Pin 32)
Input 4 or Output 4 (Pin 8)	Input 4 (Pin 10 Connector B)	Input 4 or Output 4 (Pin 9)
Output 1 (Pin 5)	Output 1 (Pin11 Connector B)	
	Output 2 (Pin12 Connector B)	
	Output 3 (Pin13 Connector B)	
	Output 4 (Pin14 Connector B)	

### Interface:

A Qt application may set or get the digital I/O pin states by calling the appropriate C library function as described below.

```
#define GHIOLIB_CH1      (0x01)
#define GHIOLIB_CH2      (0x02)
#define GHIOLIB_CH3      (0x03)
#define GHIOLIB_CH4      (0x04)

#define GHIOLIB_MAX_DIGITAL_IO (4)
#define GHIOLIB_DIG_IN_FLOAT (0)
#define GHIOLIB_DIG_IN_PULL_DN (1)
#define GHIOLIB_DIG_IN_PULL_UP (2)

#define GHIOLIB_RET_OK      0
#define GHIOLIB_RET_ERROR    1
#define GHIOLIB_RET_NOTSUPPORTED 2
```

### ghiolib\_setDigIncfg (Model 3D70 only)

Sets input pin pull-up/pull-down configuration.

**Syntax**

```
int ghiolib_setDigIncfg(int ch, uint8_t config);
```

**Parameters**

int ch  
    [in]  
    Input pin to configure (GHIOLIB\_CH1, GHIOLIB\_CH2, GHIOLIB\_CH3, or GHIOLIB\_CH4)  
uint8\_t config  
    [in]  
    GHIOLIB\_DIG\_IN\_FLOAT, GHIOLIB\_DIG\_IN\_PULL\_DN, or GHIOLIB\_DIG\_IN\_PULL\_UP

**Return Value**

int  
GHIOLIB\_RET\_OK, GHIOLIB\_RET\_ERROR, or GHIOLIB\_RET\_NOTSUPPORTED

**ghiolib\_getDigIn**

This function reads the state of an input pin.

**Syntax**

```
int ghiolib_getDigIn(int ch, uint8_t *value);
```

**Parameters**

int ch  
    [in]  
    Input pin to read (GHIOLIB\_CH1, GHIOLIB\_CH2, GHIOLIB\_CH3, or GHIOLIB\_CH4)  
uint8\_t \*value  
    [out]  
    Returns 0 if input is low, else returns 1

**Return Value**

int  
GHIOLIB\_RET\_OK, GHIOLIB\_RET\_ERROR, or GHIOLIB\_RET\_NOTSUPPORTED

**ghiolib\_getDigOut**

Reads the current state of an output pin.

**Syntax**

```
int ghiolib_getDigOut(int ch, uint8_t *value);
```

**Parameters**

int ch  
    [in]  
    Output pin to read (GHIOLIB\_CH1, GHIOLIB\_CH2, GHIOLIB\_CH3, or GHIOLIB\_CH4)  
uint8\_t \*value  
    [out]

Returns 0 if output is set low, else returns 1

#### Return Value

int

GHIOLIB\_RET\_OK, GHIOLIB\_RET\_ERROR, or GHIOLIB\_RET\_NOTSUPPORTED

#### ghiolib\_setDigOut

This function sets the current state of an output pin.

#### Syntax

```
int ghiolib_setDigOut(int ch, uint8_t value);
```

#### Parameters

int ch

[in]

Output pin to set (GHIOLIB\_CH1, GHIOLIB\_CH2, GHIOLIB\_CH3, or GHIOLIB\_CH4)

uint8\_t value

[in]

If 0 sets output pin low, else sets output pin high (Vbatt)

#### Return Value

int

GHIOLIB\_RET\_OK, GHIOLIB\_RET\_ERROR, or GHIOLIB\_RET\_NOTSUPPORTED

#### Required Files:

Header File: ghiolib.h

Link Library: libghiodrv.so

#### Sample Qt Code:

```
#include <QDebug>
```

```
// For access to ghiolib
typedef u_int16_t uint16_t;
typedef u_int8_t uint8_t;
#ifdef __cplusplus
extern "C" {
#endif
```

```
#include "ghiolib.h"
```

```
#ifdef __cplusplus
}
#endif
int channel;
uint8_t digValue;
int gpioOutput;
```

```
int     gpioInput;
int     gpioStatus;

// Set inputs to pull down mode and read current inputs and outputs for each channel
gpioOutput = 0;
gpioInput = 0;
for (channel = 0; channel < GHIOLIB_MAX_DIGITAL_IO; channel++)
{
    // Set input to pull down mode
    gpioStatus = ghiolib_setDigIncfg(channel + 1, GHIOLIB_DIG_IN_PULL_DN);
    if ((GHIOLIB_RET_OK != gpioStatus) && (GHIOLIB_RET_NOTSUPPORTED != gpioStatus))
    {
        qDebug("ERROR (%d) doing ghiolib_setDigIncfg on channel: %d\n",
            gpioStatus, channel + 1);
    }

    // Read current output setting
    digValue = 0;
    gpioStatus = ghiolib_getDigOut(channel + 1, &digValue);
    if (GHIOLIB_RET_OK != gpioStatus)
    {
        qDebug("ERROR (%d) doing ghiolib_getDigOut on channel: %d\n",
            gpioStatus, channel + 1);
    }
    else
    {
        if (1 == digValue)
        {
            gpioOutput |= (1 << channel);
        }
    }

    // Read current input
    digValue = 0;
    gpioStatus = ghiolib_getDigIn(channel + 1, &digValue);
    if (GHIOLIB_RET_OK != gpioStatus)
    {
        qDebug("ERROR (%d) doing ghiolib_getDigIn on channel: %d\n",
            gpioStatus, channel + 1);
    }
    else
    {
        if (1 == digValue)
        {
            gpioInput |= (1 << channel);
        }
    }
}
qDebug("GPIO initial output: 0x%x input: 0x%x\n", gpioOutput, gpioInput);
```

## Analog Inputs (Model 3D70 only)

The Model 3D70 Display has two analog inputs. Analog Input 1 is connected to Pin 4 on Connector B and Analog Input 2 is connected to Pin 5 on Connector B. The Analog Inputs can be used to read resistance, voltage, or current with respect to the analog return pin (pin 7 on Connector B).

### Interface:

A Qt application may configure or read an analog input pin by calling the appropriate C library function as described below.

```
#define GHIOLIB_CH1      (0x01)
#define GHIOLIB_CH2      (0x02)

#define GHIOLIB_MAX_ANALOG_IN  (2)
#define GHIOLIB_ANALOG_5V      (0)
#define GHIOLIB_ANALOG_1500OHM (1)
#define GHIOLIB_ANALOG_10V     (2)
#define GHIOLIB_ANALOG_5000OHM (3)
#define GHIOLIB_ANALOG_20MA    (4)

#define GHIOLIB_RET_OK        0
#define GHIOLIB_RET_ERROR    1
#define GHIOLIB_RET_NOTSUPPORTED 2

typedef struct _ADCVALUES
{
    uint16_t adcch;
    uint16_t adcvref;
    uint16_t adcstatus;
    uint16_t adccfg;
} ADCVALUES, *PADCVALUES;
```

### ghiolib\_setADCCfg (Model 3D70 only)

This function configures an analog input for one of five different reading modes.

#### Syntax

```
int ghiolib_setADCCfg(int ch, uint8_t config);
```

#### Parameters

int ch

[in]

Input to configure (GHIOLIB\_CH1 or GHIOLIB\_CH2)

uint8\_t config

[in]

GHIOLIB\_ANALOG\_5V, GHIOLIB\_ANALOG\_10V, GHIOLIB\_ANALOG\_1500OHM,  
GHIOLIB\_ANALOG\_5000OHM, or GHIOLIB\_ANALOG\_20MA

#### Return Value

int

GHIOLIB\_RET\_OK, GHIOLIB\_RET\_ERROR, or GHIOLIB\_RET\_NOTSUPPORTED

### **ghiolib\_getADCIn (Model 3D70 only)**

This function gets a reading from an analog input pin.

#### **Syntax**

```
int ghiolib_getADCIn(int ch, PADCVALUES p);
```

#### **Parameters**

int ch

[in]

Input to read (GHIOLIB\_CH1 or GHIOLIB\_CH2)

PADCVALUES p

[out]

Reading is returned in member “adcch” of this structure. Other items in this structure can be ignored.

#### **Return Value**

int

GHIOLIB\_RET\_OK, GHIOLIB\_RET\_ERROR, or GHIOLIB\_RET\_NOTSUPPORTED

#### **Required Files:**

Header File: ghiolib.h

Link Library: libghiodrv.so

#### **Sample Qt Code:**

```
#include <QDebug>

// For access to ghiolib
typedef u_int16_t uint16_t;
typedef u_int8_t uint8_t;
#ifdef __cplusplus
extern "C" {
#endif

#include "ghiolib.h"

#ifdef __cplusplus
}
#endif

int channel = 0;
ADCVALUES analogData;
int gpioStatus;

// Set analog input 1 to read 0 to 10 volts
gpioStatus = ghiolib_setADCcfg(channel + 1, GHIOLIB_ANALOG_10V);
```



```
if (GHIOLIB_RET_OK != gpioStatus)
{
    qDebug("ERROR (%d) doing ghiolib_setADCcfg on channel: %d\n",
           gpioStatus, channel + 1);
}

// Get current reading
gpioStatus = ghiolib_getADCin(channel + 1, &analogData);
if (GHIOLIB_RET_OK != gpioStatus)
{
    qDebug("ERROR (%d) doing ghiolib_getDigOut on channel: %d\n",
           gpioStatus, channel + 1);
}
qDebug("Reading from channel %d is %d millivolts\n", channel + 1, analogData.adcch);
```

## Buzzer (Models 3D70, 3D2104)

The Model 3D70 and 3D2104 Displays have an internal buzzer that can be sounded on command.

### Interface:

A Qt application can turn the internal buzzer on or off by sending the proper number to the buzzer control file.

### Required Files:

Header File: none

Link Library: none

### Sample Qt Code:

```
#include <QString>
#include <QDebug>

QFile          buzzerFile;
bool           buzzerFileOpen;

buzzerFile.setFileName("/sys/class/backlight/pwm-
backlight.3/brightness");
buzzerFileOpen = buzzerFile.open(QIODevice::WriteOnly |
QIODevice::Text);

if (false == buzzerFileOpen)
{
    qDebug("Error opening buzzer file\n");
}

// To turn buzzer ON
if (true == buzzerFileOpen)
{
    QTextStream buzzerOut(&buzzerFile);
```

```
    buzzerOut << 10;
}

// . . .

// To turn buzzer OFF
if (true == buzzerFileOpen)
{
    QTextStream buzzerOut (&buzzerFile);
    buzzerOut << 0;
}
```

## Audio Output (Model 3D70 only)

The Model 3D70 Display has the ability to play an mp3 audio file and send the audio output to a monaural line out (pins 1, AUDIO OUT, and 2, AUDIO RET, on the B connector).

### Interface:

A Qt application can start playing an mp3 audio file and can stop the playing of the audio file using a Linux utility called mpg123.

### Required Files:

Header File: none

Link Library: none

Executable: mpg123 (normally installed on Model 3D70 Display)

### Sample Qt Code:

```
// To play mp3 file "sounds.mp3"
// Note that by placing mp3 file in "images" folder, Qt will automatically
// download the mp3 file to the target with the other image files being used.
// Command shown to play mp3 file will first stop playing any mp3 file
// that may already be playing.
system("test `pidof mpg123` && kill `pidof mpg123` ;"
      "mpg123 -q images/sounds.mp3 &");

// To stop playing mp3 file (if any)
system("test `pidof mpg123` && kill `pidof mpg123`");
```

## Appendix H: Setting 3Dxx Flash File System R/W Mode

- To immediately set the 3Dxx Display file system to read-write mode enter this console command:
  - **mount -o remount,rw /**
- The above command only remains in effect until the next reboot and is usually stored in a script file here: /home/writeablefs.
- To have the 3Dxx Display file system set to read-write mode on boot-up, edit the file /etc/init.d/rc-  
once and add the above command to the end of this file just before the final “exit” command like this:

```
...
...
...
case "$1" in
    start)
        do_start >&2
        ;;
    *)
        echo "Usage: $0 {start}" >&2
        exit 1
        ;;
esac

mount -o remount,rw /

exit 0
```

- To leave the 3Dxx Display file system set to read-only mode on boot-up, edit the file /etc/init.d/rc-  
once and remove the “mount -o remount,rw /” line near the end of the file (or comment it out by  
putting a “#” in column one of that line)
- Another way to have the 3Dxx Display file system set to read-write mode on boot-up, is to add a  
link to the “writeablefs” script in the home directory like this:
  - **ln -s /home/writeablefs /etc/rc.d/S03writeablefs**

The 3Dxx configuration script utilizes this technique to configure the 3Dxx Display file system to be in read-write mode to make Qt development more convenient.

## Appendix I: Building Qt Library Source

Note: This appendix is included for reference and is not a required step.

This section describes the procedure to download and build the Qt 5.9.3 library code. The library source code can be downloaded from Grayhill at: <http://www.grayhill.com/qt43d>.

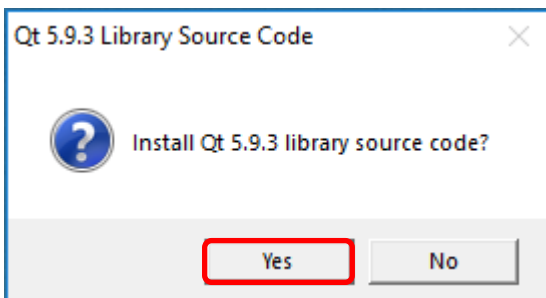
Please reference <http://doc.qt.io/qt-5/windows-requirements.html> for additional information.

This procedure relies on both Qt Creator and the Grayhill support files having been previously downloaded and installed.

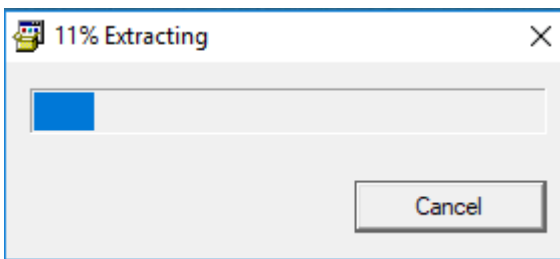
- Download “Qt 5.9.3 Library Source” from the Grayhill website
- Open the download folder and double click on “QtLibrarySrc.exe”

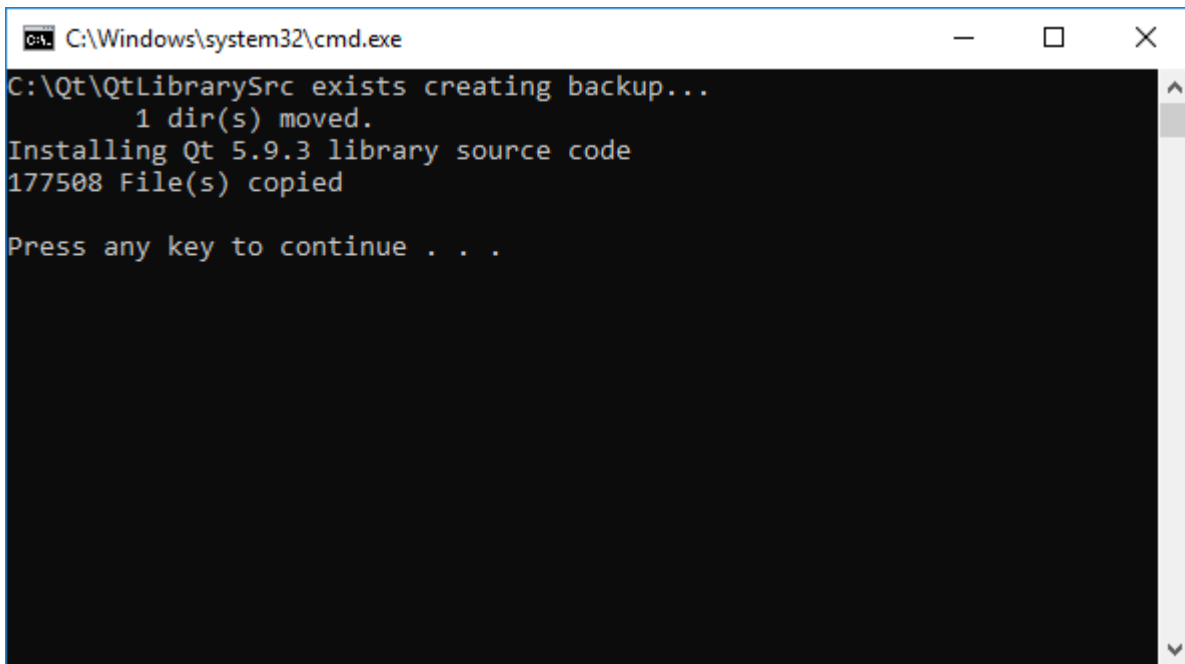
A User Access Control window may pop-up

- Click “Yes” to allow the self-extracting zip file to proceed
- The following window appears



- Click “Yes”

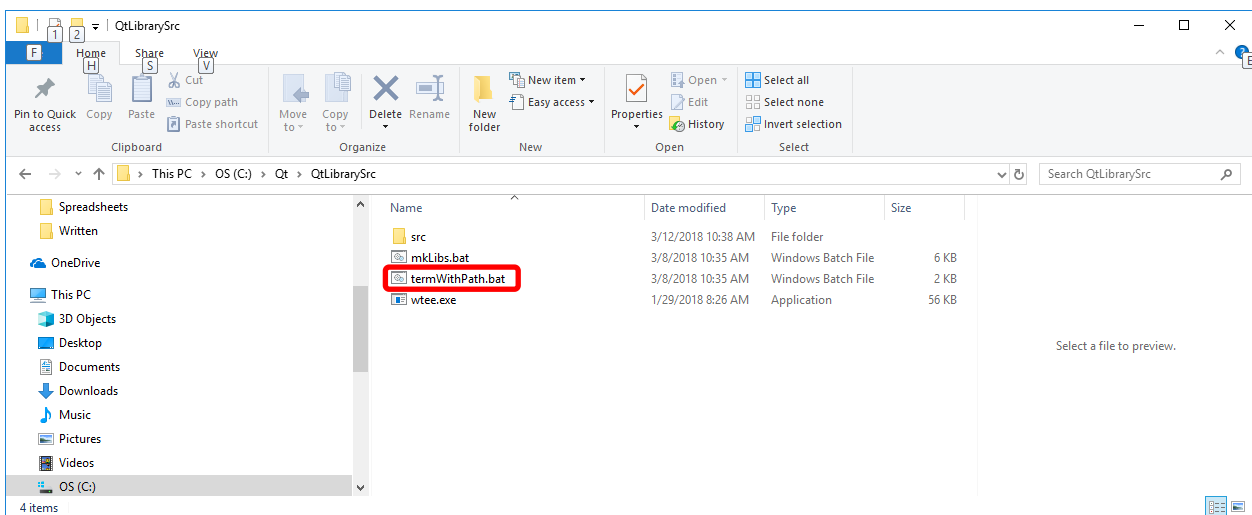




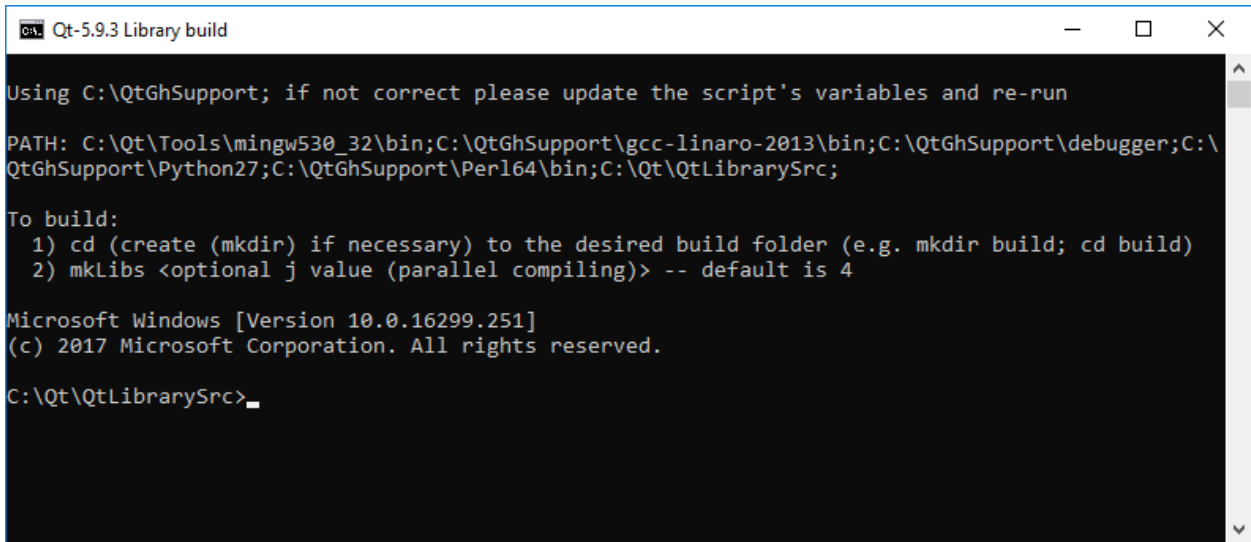
```
C:\Windows\system32\cmd.exe
C:\Qt\QtLibrarySrc exists creating backup...
    1 dir(s) moved.
Installing Qt 5.9.3 library source code
177508 File(s) copied

Press any key to continue . . .
```

- Using Windows Explorer; navigate to “C: Qt\QtLibrary\Src” and verify the folder was installed



- Double click on “termWithPath.bat” – this launches a cmd window with the properly configured path



```
Qt-5.9.3 Library build
Using C:\QtGhSupport; if not correct please update the script's variables and re-run
PATH: C:\Qt\Tools\mingw530_32\bin;C:\QtGhSupport\gcc-linaro-2013\bin;C:\QtGhSupport\debugger;C:\QtGhSupport\Python27;C:\QtGhSupport\Perl64\bin;C:\Qt\QtLibrarySrc;
To build:
  1) cd (create (mkdir) if necessary) to the desired build folder (e.g. mkdir build; cd build)
  2) mkLibs <optional j value (parallel compiling)> -- default is 4
Microsoft Windows [Version 10.0.16299.251]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Qt\QtLibrarySrc>_
```

- mkdir build
- cd build
- mkLibs



```
Qt-5.9.3 Library build
Using C:\QtGhSupport; if not correct please update the script's variables and re-run
PATH: C:\Qt\Tools\mingw530_32\bin;C:\QtGhSupport\gcc-linaro-2013\bin;C:\QtGhSupport\debugger;C:\QtGhSupport\Python27;C:\QtGhSupport\Perl64\bin;C:\Qt\QtLibrarySrc;
To build:
  1) cd (create (mkdir) if necessary) to the desired build folder (e.g. mkdir build; cd build)
  2) mkLibs <optional j value (parallel compiling)> -- default is 4
Microsoft Windows [Version 10.0.16299.251]
(c) 2017 Microsoft Corporation. All rights reserved.
C:\Qt\QtLibrarySrc>mkdir build
C:\Qt\QtLibrarySrc>cd build
C:\Qt\QtLibrarySrc\build>mkLibs_
```

- The window title bar updates as the build progresses

```

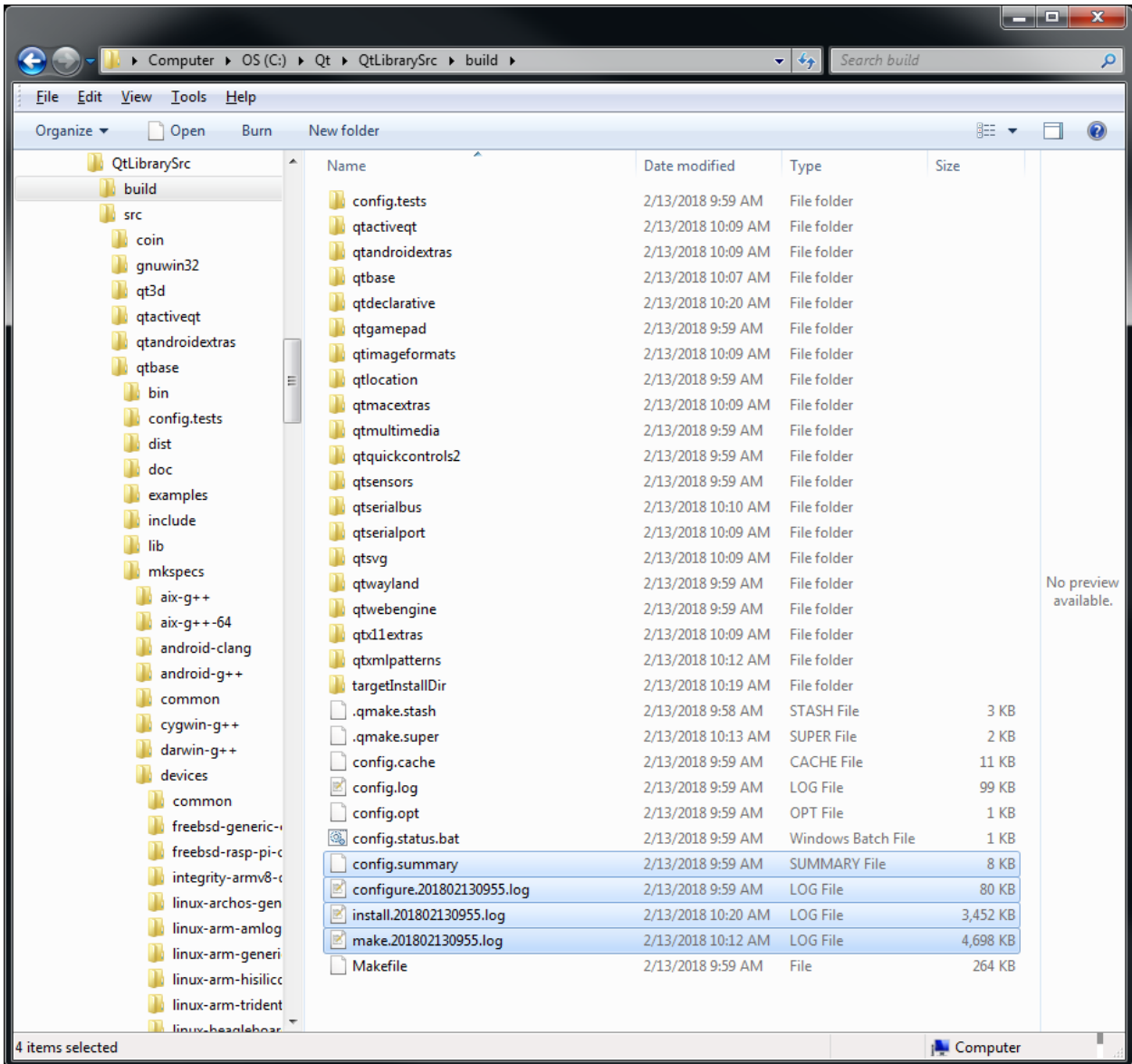
Qt 5.9.3 Library build - Completed!
librarySrc\build\qtbase\include\QtNetwork -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtCore\5.9.3 -IC:\Qt\QtLibrarySrc\src\q
tbase\include\QtCore\5.9.3\QtCore -IC:\Qt\QtLibrarySrc\build\qtbase\include\QtCore\5.9.3 -IC:\Qt\QtLibrarySrc\build\qtb
ase\include\QtCore\5.9.3\QtCore -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtWidgets -IC:\Qt\QtLibrarySrc\build\qtbase\inclu
de\QtWidgets -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtGui -IC:\Qt\QtLibrarySrc\build\qtbase\include\QtGui -IC:\Qt\QtLib
rarySrc\src\qtbase\include\QtCore -IC:\Qt\QtLibrarySrc\build\qtbase\include\QtCore -I.moc -IC:\Qt\QtSupport\targetSysroo
t\include -IC:\Qt\QtSupport\targetSysroot\usr\include -IC:\Qt\QtLibrarySrc\src\qtbase\mkspecs\devices\linux-ix86-g++ -o
.obj\moc_conf.obj .moc\moc_conf.cpp
C:\Qt\QtLibrarySrc\build\..\QtSupport\gcc-linaro-2013\bin\arm-linux-gnueabi-g++ -c -march=armv7-a -mcpu=neon -DLINUX=
1 -DEGL_API_FB=1 -wno-psabi -D QT_OPENGL_FORCE_SHADER_DEFINES -mfloat-abi=softfp -O2 -std=c++11 -fno-exceptions -Wall -W
-wvla -D_REENTRANT -fpic -DQT_QML_DEBUG_NO_WARNING -DQT_NO_NARROWING_CONVERSIONS_IN_CONNECT -DQT_USE_QSTRINGBUILDER -DQ
T_NO_EXCEPTIONS -D_LARGEFILE64_SOURCE -D_LARGEFILE_SOURCE -DQT_NO_DEBUG -DQT_QML_LIB -DQT_NETWORK_LIB -DQT_WIDGETS_LIB -
DQT_GUI_LIB -DQT_CORE_LIB -IC:\Qt\QtLibrarySrc\src\qtdeclarative\tools\qml -I. -IC:\Qt\QtLibrarySrc\src\qtdeclarative\in
clude -IC:\Qt\QtLibrarySrc\src\qtdeclarative\include\QtQml -I.\..\include -I.\..\include\QtQml -IC:\Qt\QtLibrarySrc\sr
c\qtbase\include -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtNetwork -IC:\Qt\QtLibrarySrc\build\qtbase\include -IC:\Qt\QtL
ibrarySrc\build\qtbase\include\QtNetwork -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtCore\5.9.3 -IC:\Qt\QtLibrarySrc\src\q
tbase\include\QtCore\5.9.3\QtCore -IC:\Qt\QtLibrarySrc\build\qtbase\include\QtCore\5.9.3 -IC:\Qt\QtLibrarySrc\build\qtb
ase\include\QtCore\5.9.3\QtCore -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtWidgets -IC:\Qt\QtLibrarySrc\build\qtbase\inclu
de\QtWidgets -IC:\Qt\QtLibrarySrc\src\qtbase\include\QtGui -IC:\Qt\QtLibrarySrc\build\qtbase\include\QtGui -IC:\Qt\QtLib
rarySrc\src\qtbase\include\QtCore -IC:\Qt\QtLibrarySrc\build\qtbase\include\QtCore -I.moc -IC:\Qt\QtSupport\targetSysroo
t\include -IC:\Qt\QtSupport\targetSysroot\usr\include -IC:\Qt\QtLibrarySrc\src\qtbase\mkspecs\devices\linux-ix86-g++ -o
.obj\main.obj C:\Qt\QtLibrarySrc\src\qtdeclarative\tools\qml\main.cpp
C:\Qt\QtLibrarySrc\build\..\QtSupport\gcc-linaro-2013\bin\arm-linux-gnueabi-g++ -wl,-rpath=C:\Qt\QtSupport\targetSysr
oot\usr\lib -wl,-rpath=C:\Qt\QtSupport\targetSysroot\lib -mfloat-abi=softfp -wl,--gc-sections -wl,-O1 -fuse-ld=gold -wl,
--enable-new-dtags -wl,-z,origin -wl,-rpath,$ORIGIN/../lib -o ..\..\bin\qml.obj\main.obj .obj\qrc_qml.obj .obj\moc_con
f.obj -LC:\Qt\QtSupport\targetSysroot\lib -LC:\Qt\QtSupport\targetSysroot\usr\lib -LC:\Qt\QtLibrarySrc\build\qtdecla
rative\lib -lQt5Qml -LC:\Qt\QtLibrarySrc\build\qtbase\lib -lQt5Network -lQt5Widgets -lQt5Gui -lQt5Core -lGLESV2 -lEGL -lGL
-Lpthread
C:\Qt\QtLibrarySrc\build\qtbase\bin\qmake.exe -install qinstall -exe ..\..\bin\qml C:\Qt\QtLibrarySrc\build\targetInstal
ldir\bin\qml
C:\Qt\QtLibrarySrc\build\..\QtSupport\gcc-linaro-2013\bin\arm-linux-gnueabi-strip C:\Qt\QtLibrarySrc\build\targetInst
alldir\bin\qml
mingw32-make[3]: Leaving directory 'C:/Qt/QtLibrarySrc/build/qtdeclarative/tools/qml'
mingw32-make[2]: Leaving directory 'C:/Qt/QtLibrarySrc/build/qtdeclarative/tools'
mingw32-make[1]: Leaving directory 'C:/Qt/QtLibrarySrc/build/qtdeclarative'
Completed installation of the libraries... 10:20:32.48

Qt Library build and installation completed! Start: 9:55:53.22 Finish:10:20:32.53

C:\Qt\QtLibrarySrc\build>

```

- Log files and a configuration summary are also created



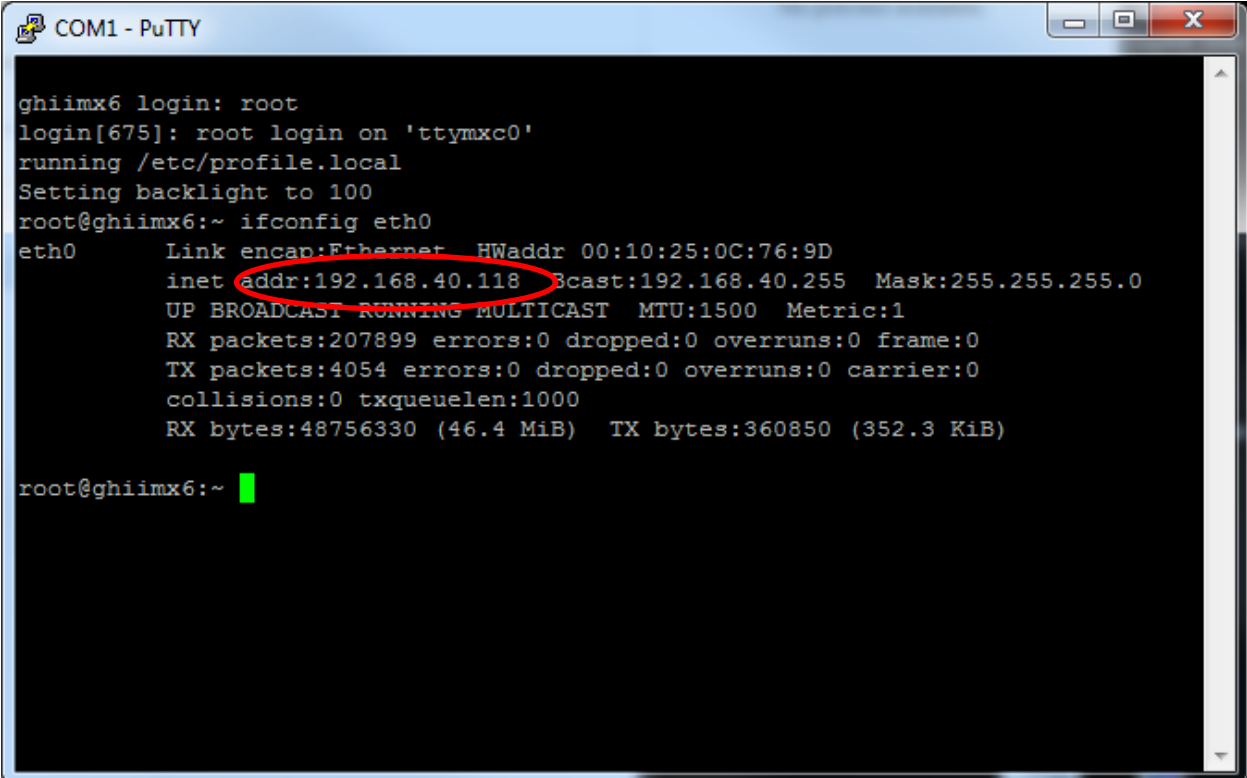


## Appendix J: Dynamic IP Address

- Enter this command to find the 3Dxx Display Ethernet IP address:
  - **ifconfig eth0**

The IP address of the 3Dxx Display is displayed after the tag “inet addr:” and is circled in **red** in the example output shown below.

- If the tag “inet addr:” is not present; enter these commands and try the “ifconfig eth0” command again
  - **ifdown eth0**
  - **ifup eth0**
- In this example the IP address is 192.168.40.118  
Make a note of this IP address



```
COM1 - PuTTY
ghiimx6 login: root
login[675]: root login on 'ttymxc0'
running /etc/profile.local
Setting backlight to 100
root@ghiimx6:~ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:10:25:0C:76:9D
          inet addr:192.168.40.118  Bcast:192.168.40.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:207899 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4054 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:48756330 (46.4 MiB)  TX bytes:360850 (352.3 KiB)

root@ghiimx6:~ █
```

[Return](#)

## Appendix K: Static IP Address

If using a **static** IP address for the display, once the address is determined:

- `cp /etc/network/interfaces /etc/network/interfaces.bak`
- `vi /etc/network/interfaces`
- replace  
    `iface eth0 inet dhcp`  
    `udhcpc_opts -t 5 -T 3 -A 20 -S &`
- with  
    `iface eth0 inet static`  
    `address 192.168.40.118`  
    `netmask 255.255.255.0`

Google “linux interface file” for additional information.

[Return](#)