



Grayhill 3Dxx Display Products

Setup and Usage of Qt 5.9.3 Development Software – Linux

Revision C2

Revision History

Revision	Date	Description
A	01/12/2018	Original Release <ul style="list-style-type: none">• Qt-5.9.3 library support• Ubuntu 16.04 64 bit Virtual Machine with flashback gnome desktop
B	01/18/2018	Minor enhancements to the VM <ul style="list-style-type: none">• Resolved non-system impacting Qt Warning• Updated default configuration Updated screen shots
C	04/06/2018	Minor updates to provide consistency with the release of the Windows documentation
C1	06/13/2018	Updates for installing support and library archives
C2	06/19/2018	Resolve C1 cross reference issues Update library building instructions

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
Installation of VirtualBox	9
Setup VirtualBox Linux Environment	17
Setup VirtualBox Serial Port.....	21
Starting Linux Development Environment from VirtualBox	26
VirtualBox Linux Passwords.....	27
Configuring 3Dxx Display's IP Address	28
Configure VM and Display	33
Selecting a 3Dxx Qt Widget Demo Project	36
Build and Run 3Dxx Embedded Application (Widget).....	37
Appendix A: Configuring a Manual Qt Kit for Grayhill Displays.....	40
Device.....	42
Compiler	48
Debugger	51
qmake	52
Kit.....	53
Appendix B: Configuring a 3Dxx Project.....	55
Build	59
Run	60
Appendix C: Debugging.....	65
Appendix D: Build and Run 3Dxx Desktop Application	71
Appendix E: Build and Run QML Demonstration Program.....	74
Appendix F: Setting up a 3Dxx Qt Program to Run at Boot Up	75

Appendix G: Interfacing 3Dxx Hardware from QT Software	76
<u>LCD</u>	76
<u>LCD Backlight</u>	77
<u>Camera Driver Interface</u>	77
<u>CAN Driver Interface</u>	83
<u>Digital I/O Driver Interface</u>	86
<u>Analog Inputs (Model 3D70 only)</u>	90
<u>Buzzer (Models 3D70, 3D2104)</u>	93
<u>Audio Output (Model 3D70 only)</u>	94
Appendix H: Setting 3Dxx Flash File System R/W Mode	95
Appendix I: Building Qt Library Source	96
Appendix J: Dynamic IP Address	98
Appendix K: Static IP Address	99

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 Linux. The Linux system is supported as a virtual machine using Oracle's VirtualBox (<https://www.virtualbox.org/wiki/VirtualBox>) software.

The virtual machine is Ubuntu 16.04 using gnome flashback for the desktop; additionally PuTTY (telnet client software - <http://www.putty.org>) has been installed.

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

Other Recommended Equipment (for all displays)

- 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 XP* or Windows 7/10 with the following features:
 - 4 GB RAM (minimum)
 - 40 GB available hard drive space (minimum)
 - Ethernet port
 - RS232 Port (or USB to serial adapter)
 - Internet Access

* Note: if using Windows XP make sure that exFAT file system support has been installed in order to read distribution media (<http://www.microsoft.com/en-us/download/details.aspx?id=19364>)

Software Required

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

- Qt Virtual Disk Image
- QtGhSupportLinux
 - Files for building Qt applications
 - Support utilities for building Qt applications
 - Example projects from Grayhill
- 3Dxx_Qt_Usage_Guide_Linux.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 “3D50 Development Quick Start Guide.pdf” and for the model 3D70 Display it is described in the document “3D70 Development Quick Start Guide.pdf”.
- Next the VirtualBox application will be downloaded from the internet and installed on the development PC. This application allows one to run a virtual computer system in a window on the PC. This means that all other Windows PC applications can be running along with this virtual computer application. This virtual computer will be used to run a version of Linux. All Qt-based development will be done under this Linux environment.
- Next the VirtualBox application will be configured. The only thing that the user must configure is the serial port interface, but this procedure will be explained. Grayhill provides a pre-configured image of Linux that has the Qt development environment already installed.
- Then a Linux script will be run to update some items in the Qt development environment. This script will only need to be run once.
- The serial and Ethernet links to the target 3Dxx Display hardware will be established. Another Linux script will be run to configure the actual 3Dxx Display product to operate with Qt instead of VUI Builder©. This second script 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 in either the Linux desktop environment or on the 3Dxx Display target hardware. This demonstration project shows how to use touch screen “buttons”, how to use touch screen swipes, how to set the 3Dxx backlight, how to operate the 3Dxx camera input, and how to access and set the real time clock. For the 3D70 Display there are also samples of using the audio output, the analog input, and the internal buzzer.

Installation of VirtualBox

This section shows how to download and install VirtualBox Version 5.1.8. If a newer version is available, it will probably work just fine, but the screen shots shown below may be different.

- Navigate to this web page: <https://www.virtualbox.org/>
- Click on “Download VirtualBox ...” selection



The screenshot shows the Oracle VM VirtualBox website. The browser address bar displays <https://www.virtualbox.org>. The page features the VirtualBox logo, a search bar, and navigation links for Login and Preferences. The main heading is "Welcome to VirtualBox.org!". Below this, there is a "News Flash" section with two entries: "New October 18th, 2016 VirtualBox 5.1.8 and 5.0.28 released!" and "New July 12th, 2016 VirtualBox 5.1 released!". A large blue button with the text "Download VirtualBox 5.1" is prominently displayed at the bottom, with a red arrow pointing to it from the left.

- Click on download for VirtualBox 5.1.8 for Windows as shown below



The screenshot shows a web browser window with the address bar displaying <https://www.virtualbox.org/wiki/Downloads>. The page features the VirtualBox logo on the left and a search bar on the right. The main content area is titled "Download VirtualBox" and contains the following text:

Here, you will find links to VirtualBox binaries and its source code.

VirtualBox binaries

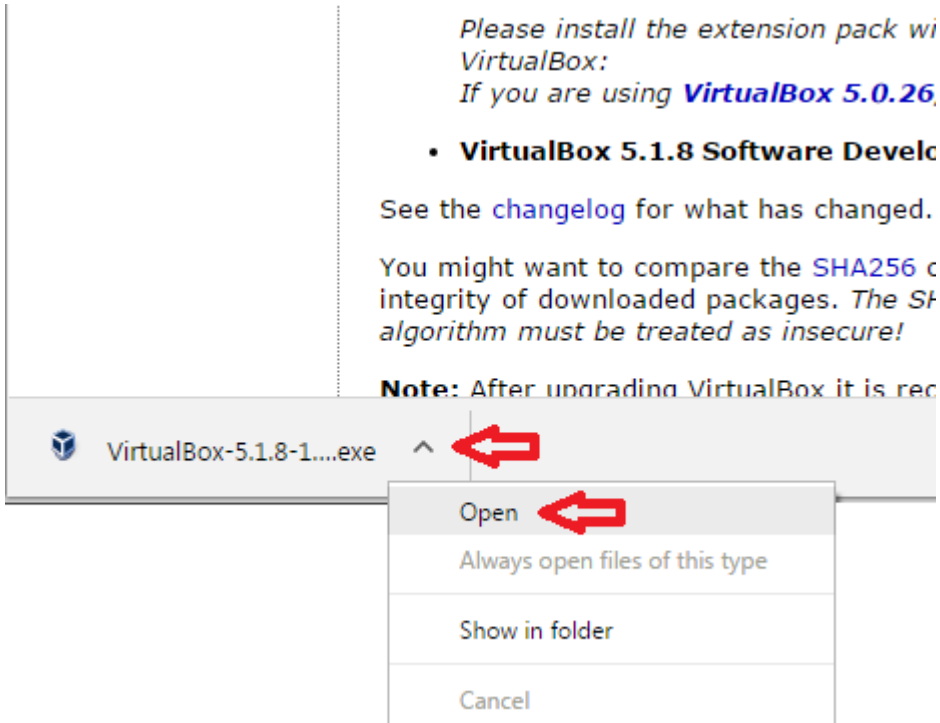
By downloading, you agree to the terms and conditions of the respective license.

- **VirtualBox platform packages.** The binaries are released under the terms of the GPL version 2.
 - **VirtualBox 5.1.8 for Windows hosts** [x86/amd64](#) 
 - **VirtualBox 5.1.8 for OS X hosts** [amd64](#)
 - **VirtualBox 5.1.8 for Linux hosts**
 - **VirtualBox 5.1.8 for Solaris hosts** [amd64](#)
- **VirtualBox 5.1.8 Oracle VM VirtualBox Extension Pack** [All supported platforms](#)
Support for USB 2.0 and USB 3.0 devices, VirtualBox RDP and PXE boot for Intel cards. See [this chapter from the User Manual](#) for an introduction to this Extension Pack. The Extension Pack binaries are released under the [VirtualBox Personal Use and Evaluation License \(PUEL\)](#).
Please install the extension pack with the same version as your installed version of VirtualBox:
*If you are using **VirtualBox 5.0.26**, please download the extension pack [here](#).*
- **VirtualBox 5.1.8 Software Developer Kit (SDK)** [All platforms](#)

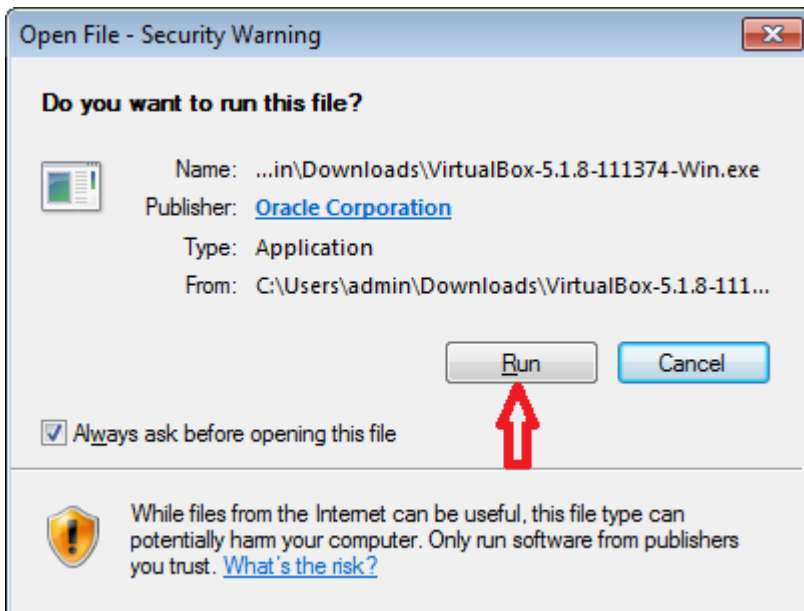
See the [changelog](#) for what has changed.

You might want to compare the [SHA256](#) checksums or the [MD5](#) checksums to verify the

- When the download completes, click on “Open” option. (Note that this download example was done using a Chrome browser. If a different web browser is used then this open operation may look different.)



- When the “Open File – Security Warning” appears, click on “Run”



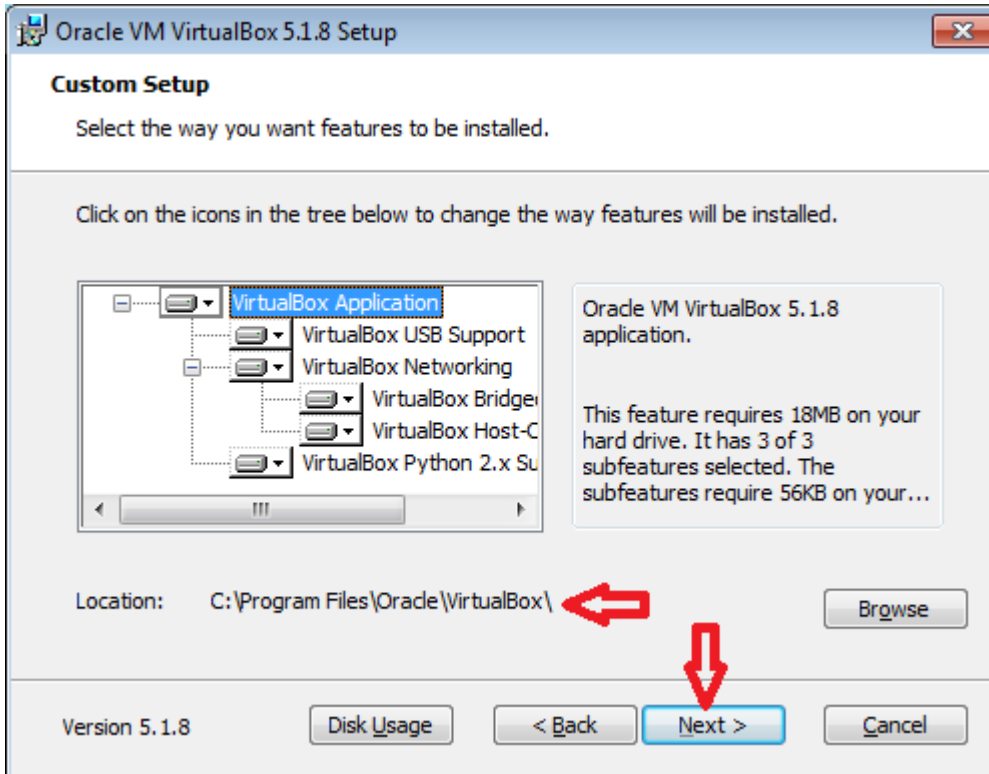
- Next the following “Welcome...” dialog should appear. Click on “Next” button



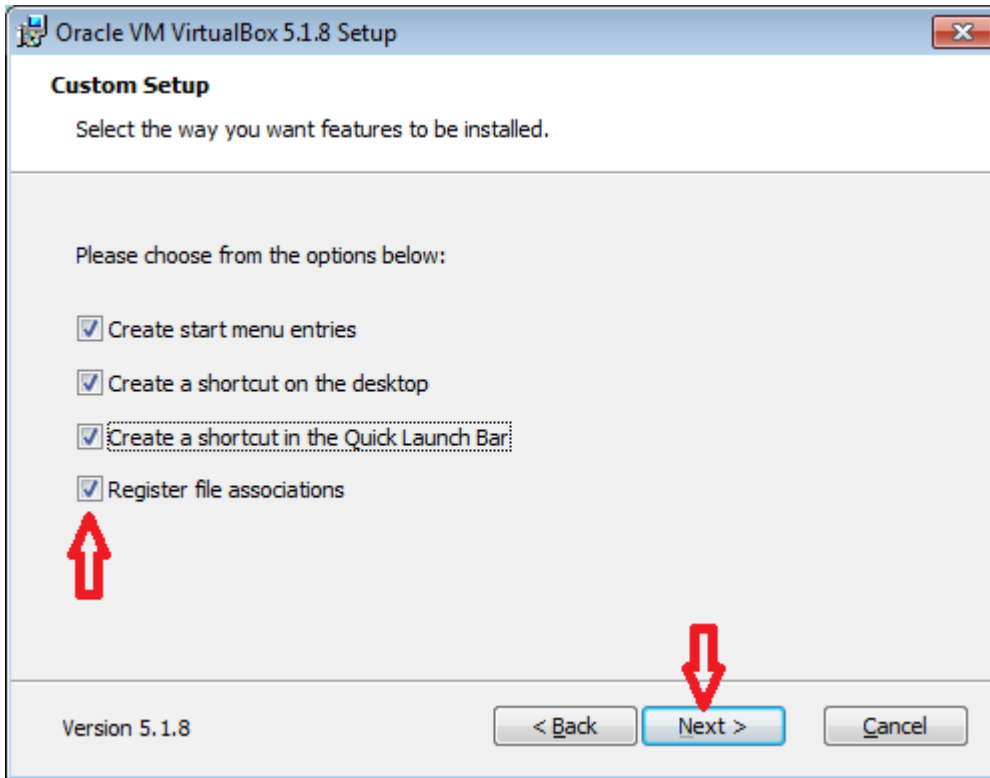
- Next will appear this “Custom Setup” dialog. Do not make any changes, just click “Next” button

NOTE!

Make sure that the installation location is on a local disk drive, not a network storage unit! This is because during installation the network connections will be disconnected so that the VirtualBox can install network adapter software and this will make network storage units inaccessible.



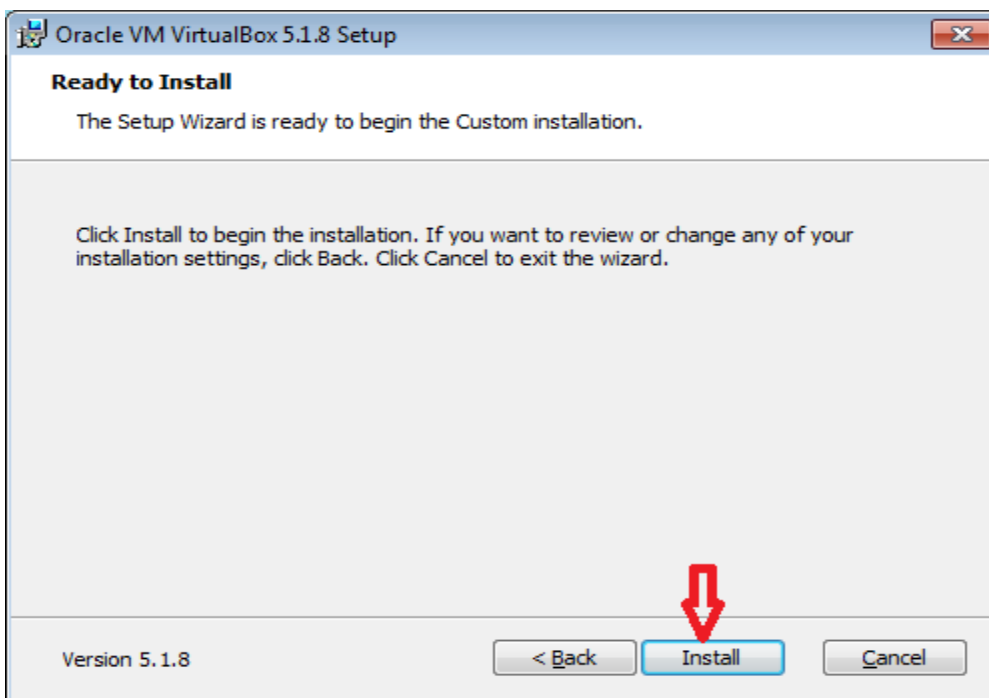
- On the second “Custom Setup” screen the first three options may be adjusted as desired, but leave the “Register file associations” option checked. When done, click “Next” button



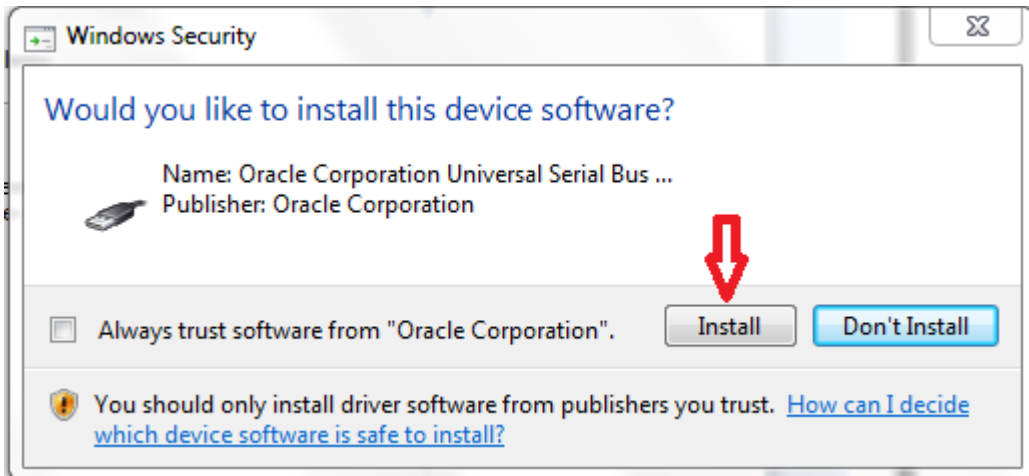
- The following “Warning: Network Interface” dialog should appear next. Make sure that there are no network accessing programs running (i.e. email) on the computer. Exit any such programs and then click “Yes” button.



- The next dialog is “Ready to Install” as shown here. Click on the “Install” button



- If a “User Account Control” window appears, click “Yes” button
- If any “Windows Security” windows appear as shown below, click on “Install” button. Several such windows may appear asking permission to install various driver software modules



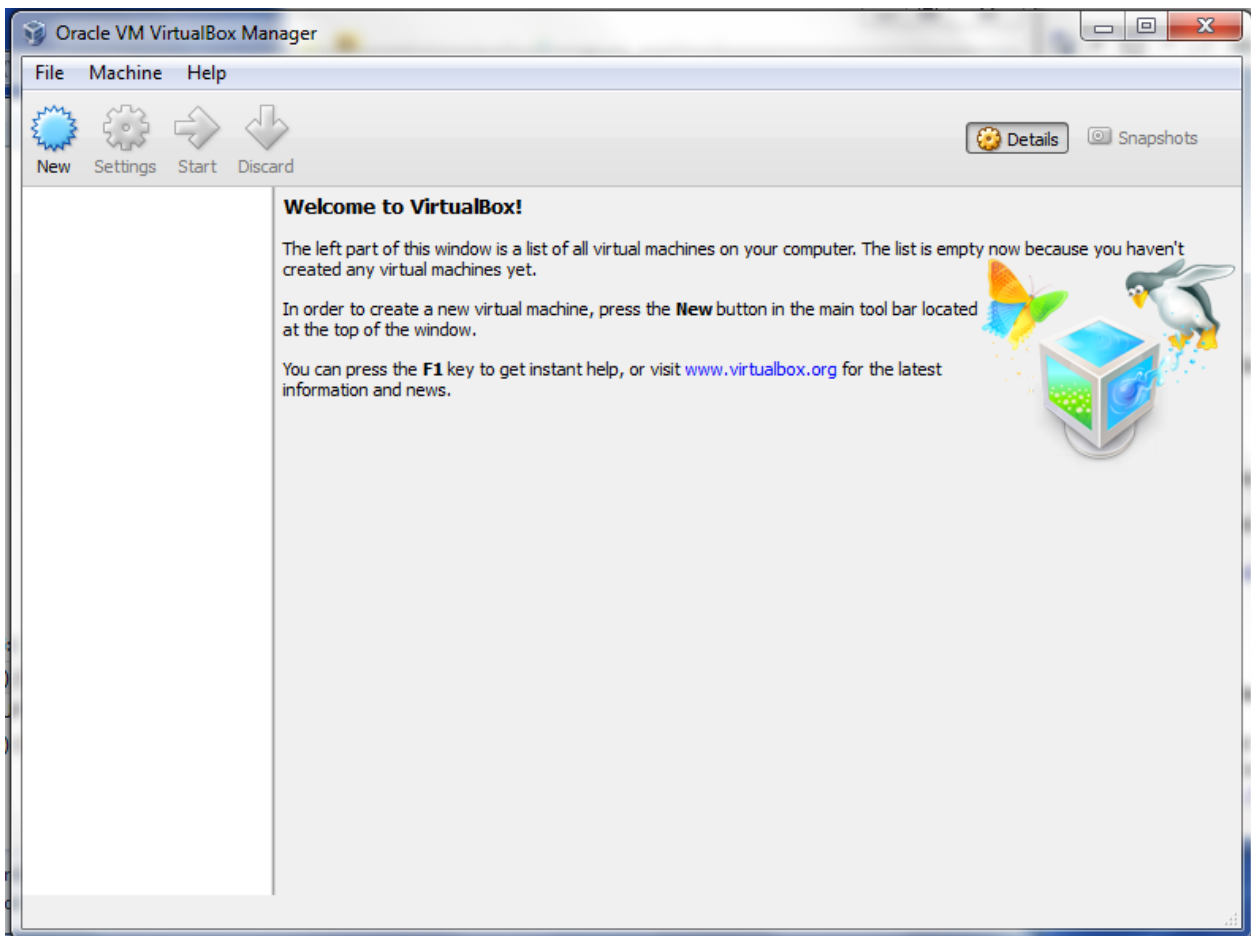
- When installation is finished, this window should appear. Uncheck the “Start Oracle VM VirtualBox...” selection and click the “Finish” button



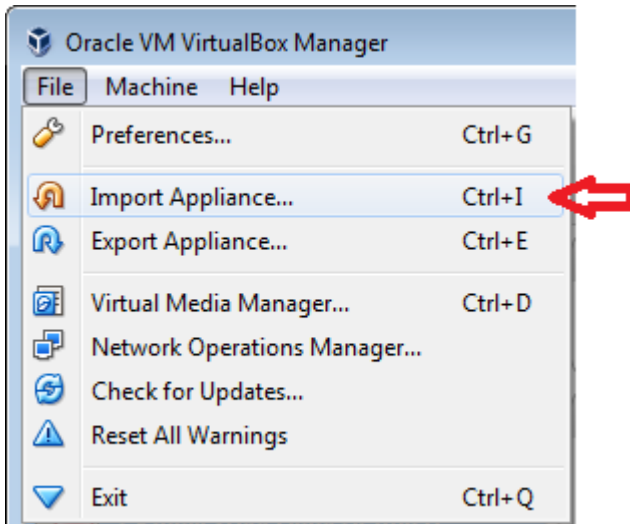
- Reboot computer at this time to re-establish network connections.

Setup VirtualBox Linux Environment

- After the computer reboots; create the “C:\VMSharedFolder” folder. This folder can be used to transfer files between the Windows and Linux environments.
- Copy the file “Qt Virtual Disk Image” to any location on the development PC. Remember the location of this copied .ova file as it will be needed later.
- Start the Oracle VM VirtualBox program. This screen should appear:



- Click on “File -> Import Appliance”

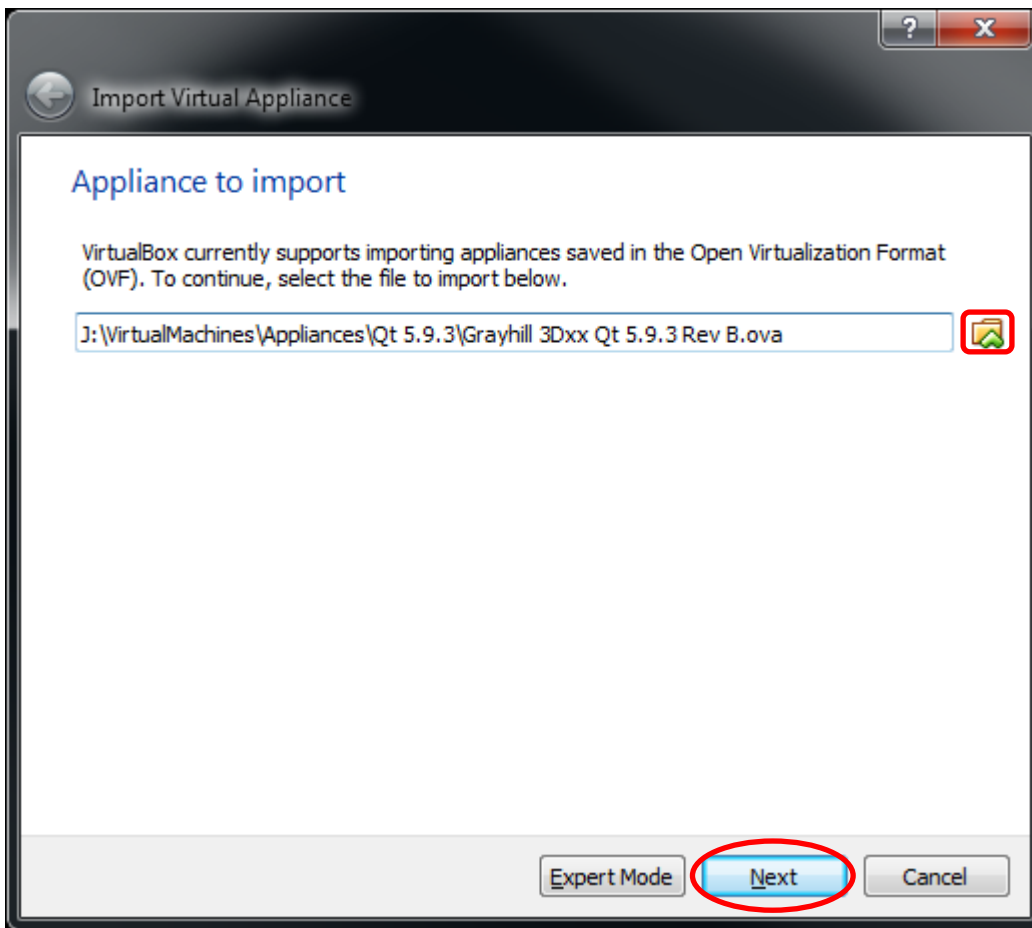


- Click on the open folder icon as shown below and navigate to the previously copied file: “Qt Virtual Disk Image” and open it. N.B. The following example references revision B of the ova.

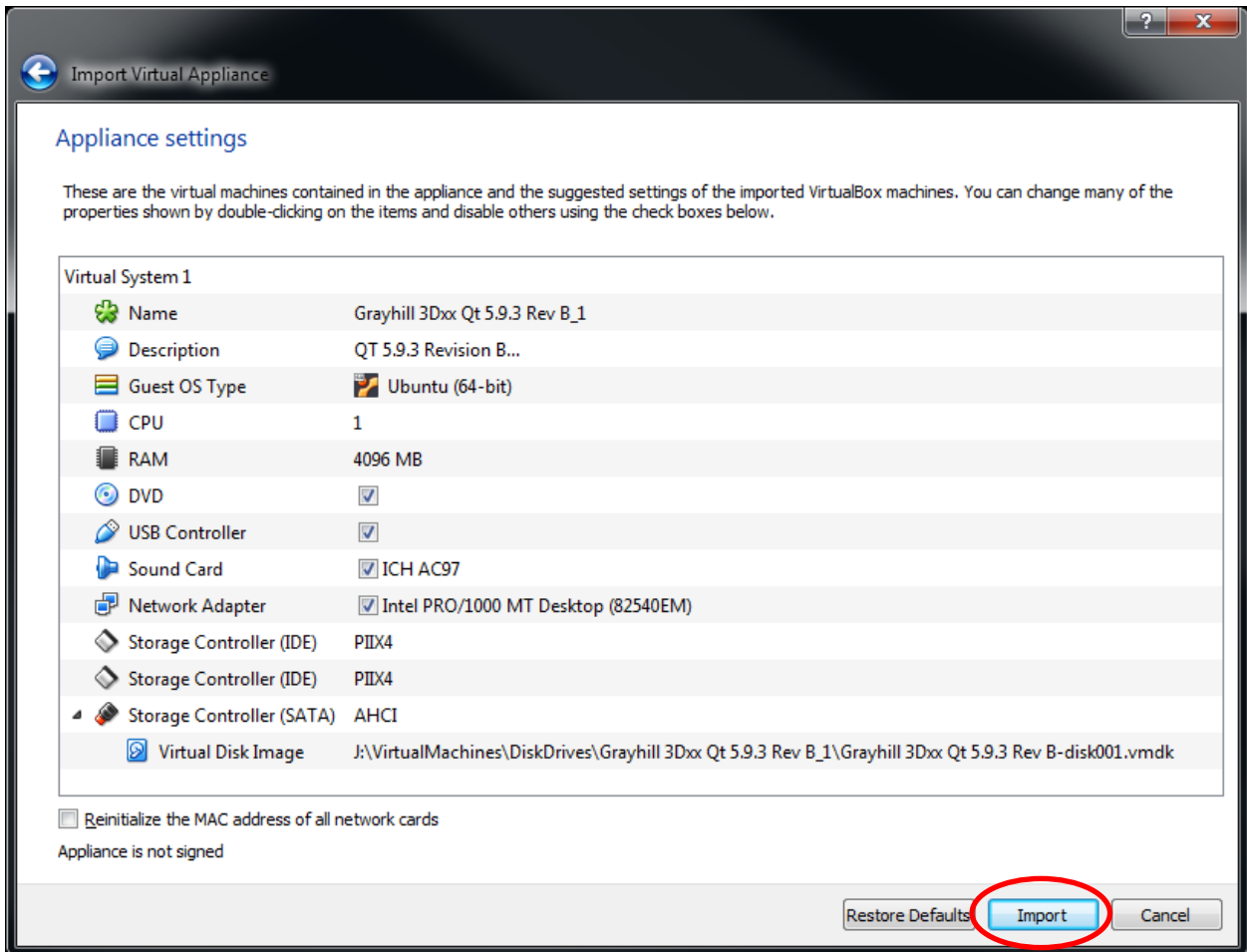
NOTE

The above .ova file is quite large and if distributed on a USB memory stick, that memory stick must use exFAT format. If operating under Windows XP, then the exFAT file system support package must be installed in order to access this media (see note under **Other Recommended Equipment** for more details).

- Click “Next”



- Adjust any settings as appropriate, such as number of processors or RAM size and then click “Import” button.
- Click “Import”



N.B. The example above shows a “_1” suffix; a machine with the name “Grayhill 3Dxx Qt 5.9.3 Rev B” already exists.

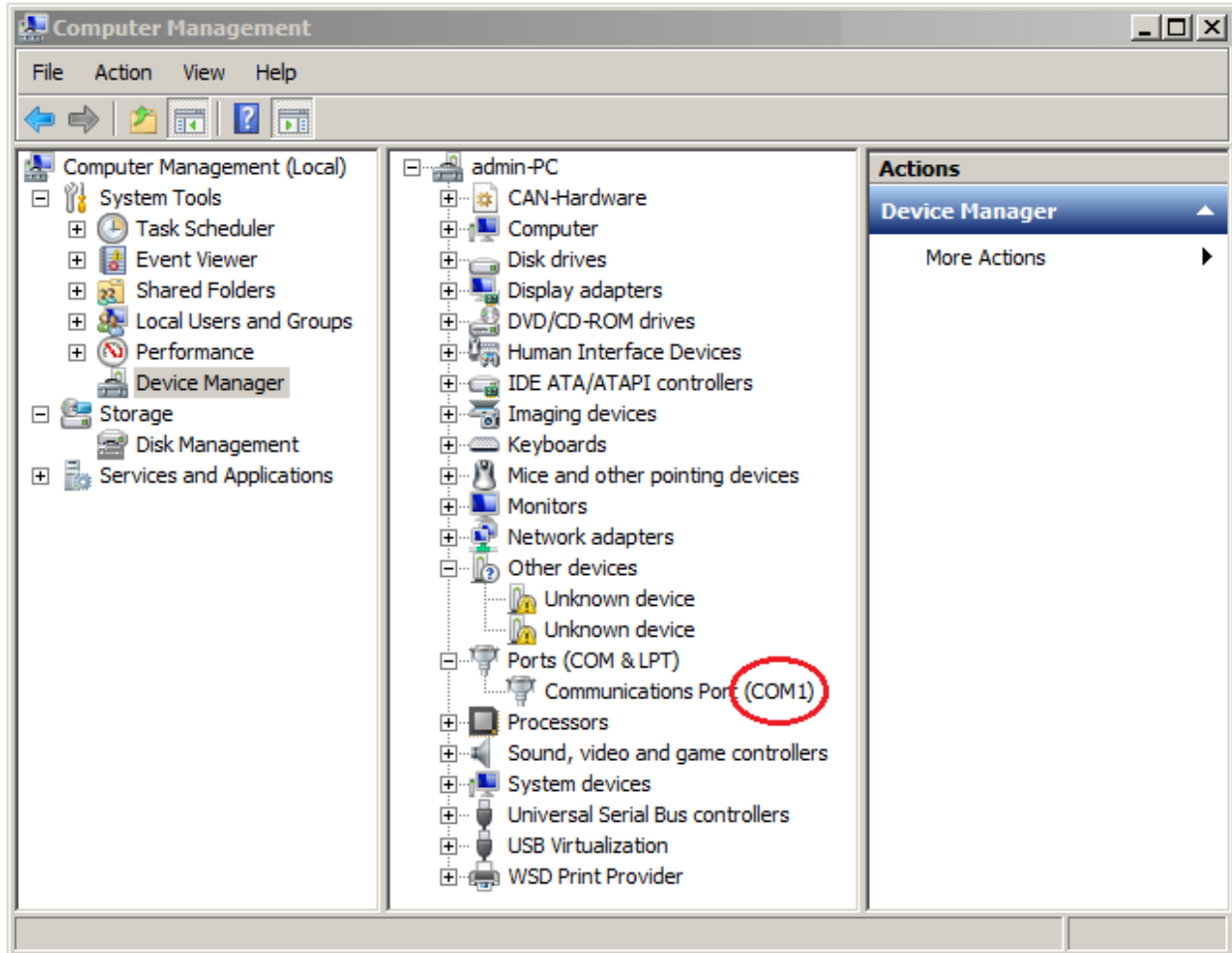
Setup VirtualBox Serial Port

In order to access the 3Dxx Display Linux console, a serial port that operates at 115200 baud is required. If the development PC has a built-in serial port that is going to be used for this purpose, then proceed with the setup instructions below to configure the VirtualBox Serial Port. If a USB to serial port converter is going to be used, then skip the VirtualBox Serial Port setup and continue with the step:

Starting Linux Development Environment from VirtualBox.

Determine what COM port is assigned to the serial port that is going to be used. This can be determined by accessing the Device Manager window and looking under the “Ports (COM & LPT)” entry. In the example shown below the serial port is assigned to “COM1”.

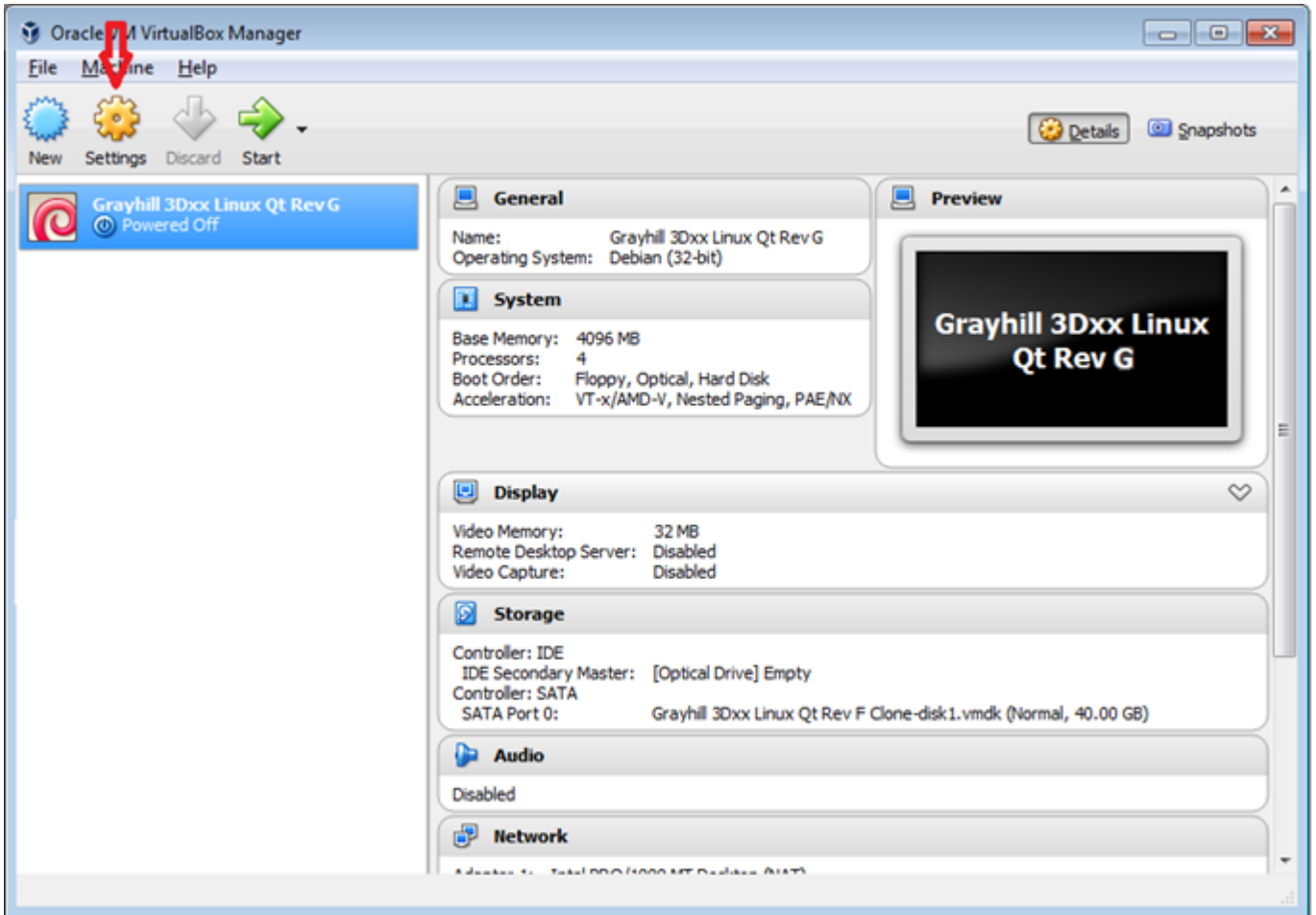
Note: Using a COM port greater than 9 will not work!



Close the “Device Manager” dialog box, but remember the COM port number used.

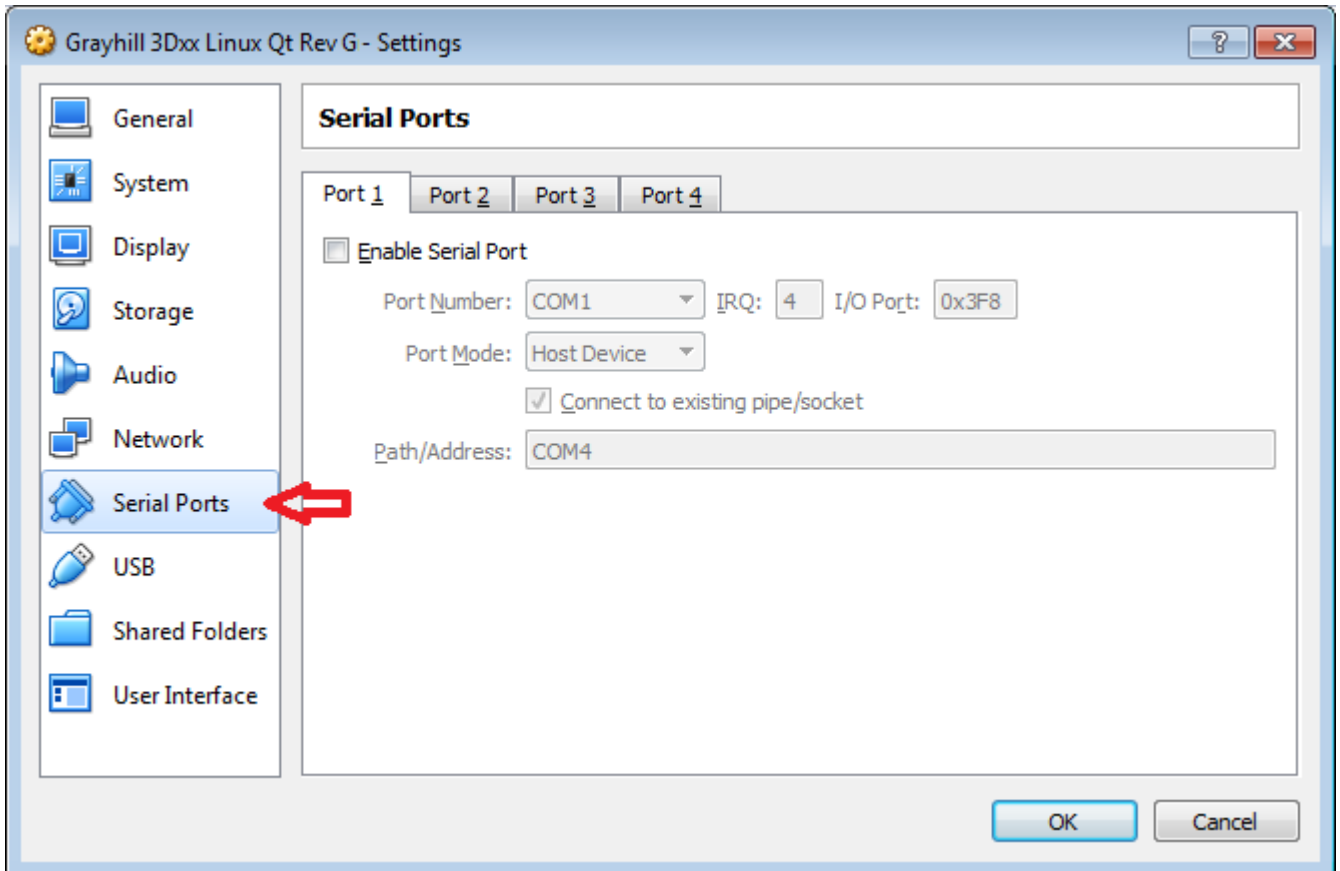
After the previous VirtualBox “Import” operation was performed, this screen should appear.

- Select “Settings”



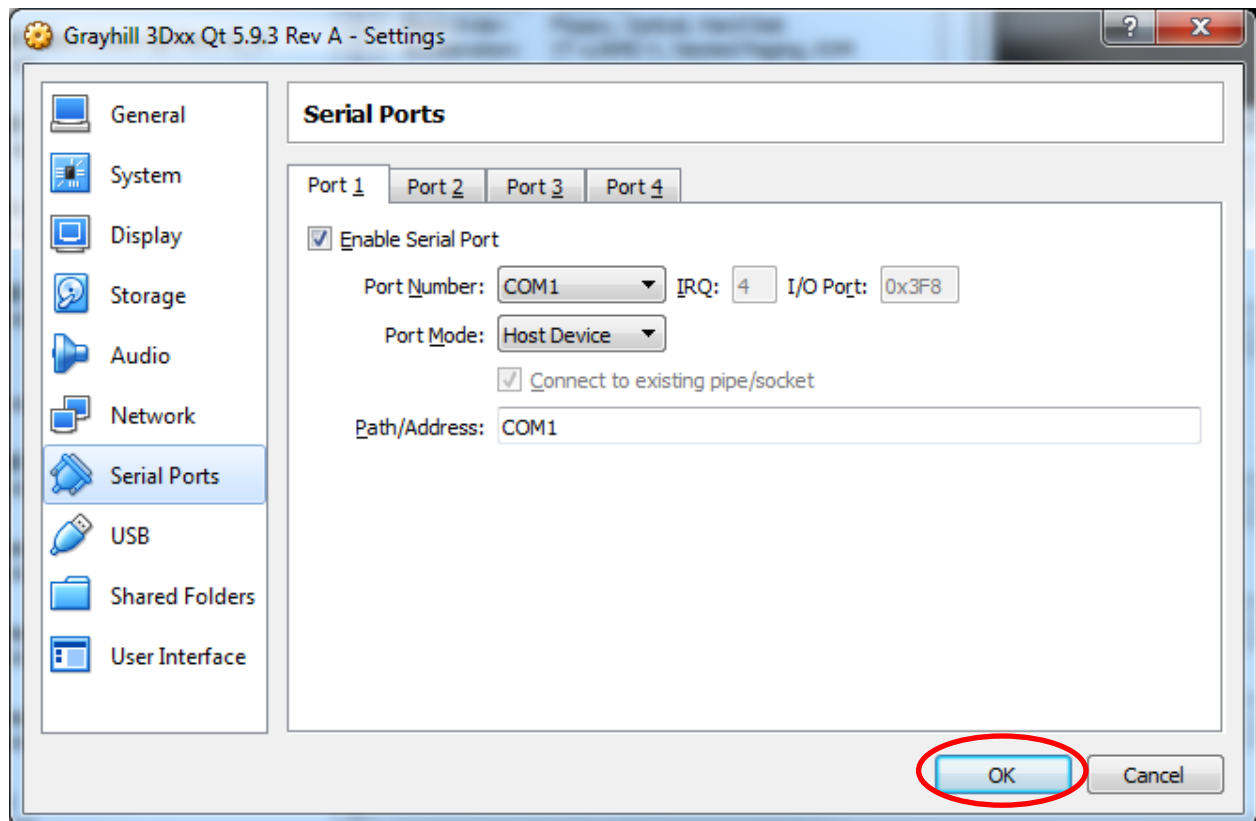
- When the “Grayhill 3Dxx Linux Qt ... -Settings” dialog box appears, click on the “Serial Ports” item on the left column as shown:

N.B. Serial port 1 should already be configured as COM1 as part of the appliance import



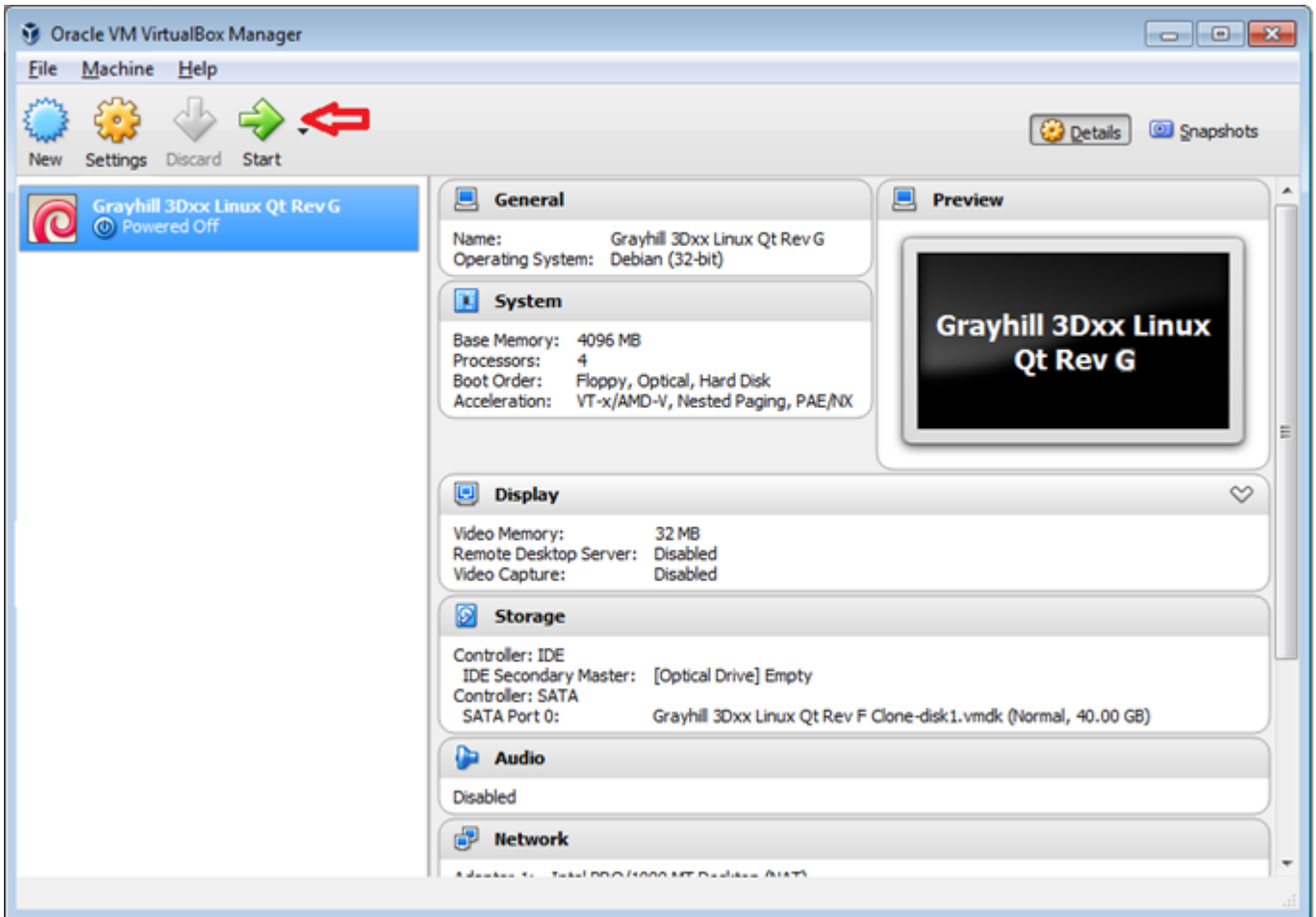
- Select the “Port 1” tab
 - Select “Enable Serial Port”
 - Set “Port Number:” to “COM1” no matter what COM port is being used.
 - Set “Port Mode:” to “Host Device”
 - Set “Path/Address:” to the COM port name and number from the previous “Device Manager” step


An example is shown below. When these settings are correct click “OK”

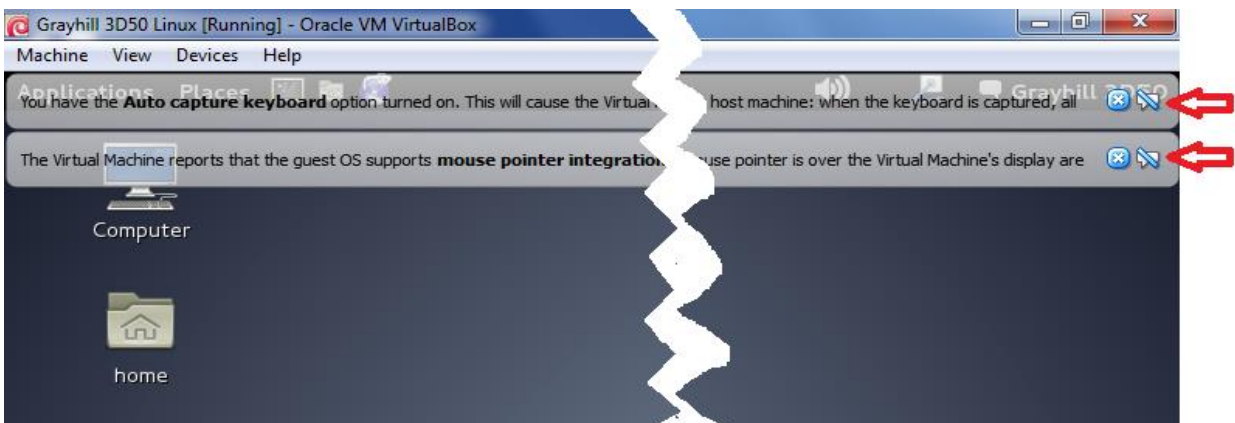


Starting Linux Development Environment from VirtualBox

- When focus returns to this screen, click on the big, green “Start: arrow



- If any messages such as the ones shown below appear when Linux first starts, click on the  icon to prevent these messages from appearing again



VirtualBox Linux Passwords

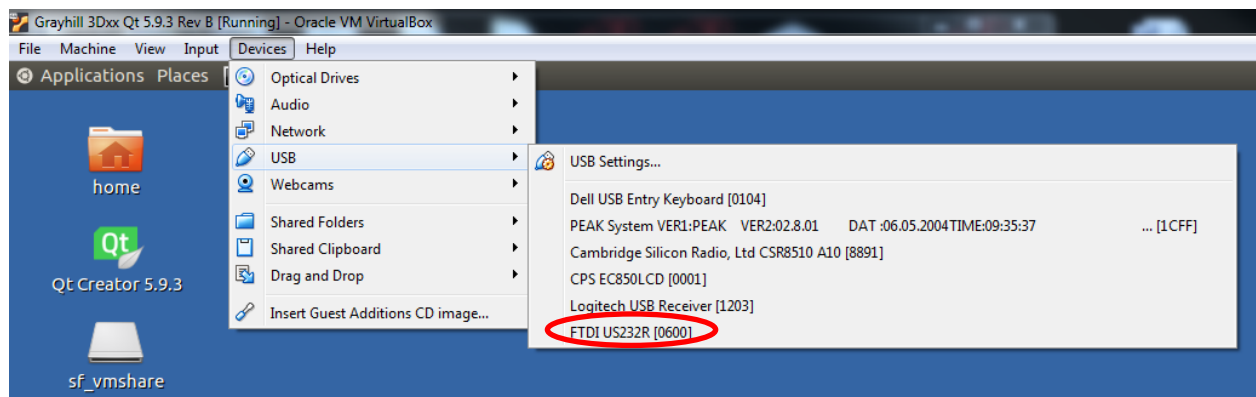
- ghguest !admin!
- root !rty32999!

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 VM and the display). This depends on how the 3Dxx Display serial port is physically connected to the development PC. If using a built-in serial port on the development PC, remember the serial port device name is “/dev/ttyS0”. If using a USB to serial port adapter, remember the serial port device name is “/dev/ttyUSB0”.
- **Only** if using the USB to serial port adapter: it must be activated at this time by clicking on the “Devices” menu option at the top of the VirtualBox screen. Select “USB” and click on the USB to serial port adapter device name in order to select it. A sample is shown here (the USB device name may be different than shown):



Launch PuTTY (multiple options)

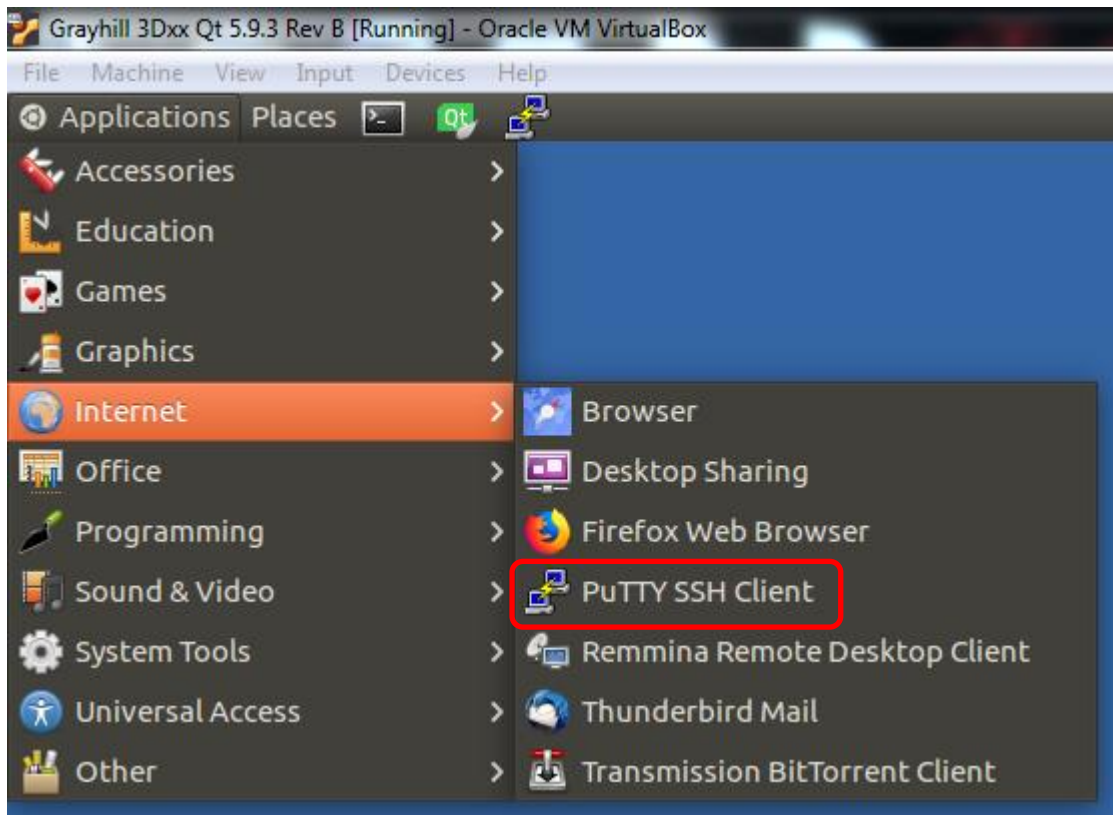
- Click on the top panel PuTTY icon



- Double Click on the desktop PuTTY icon

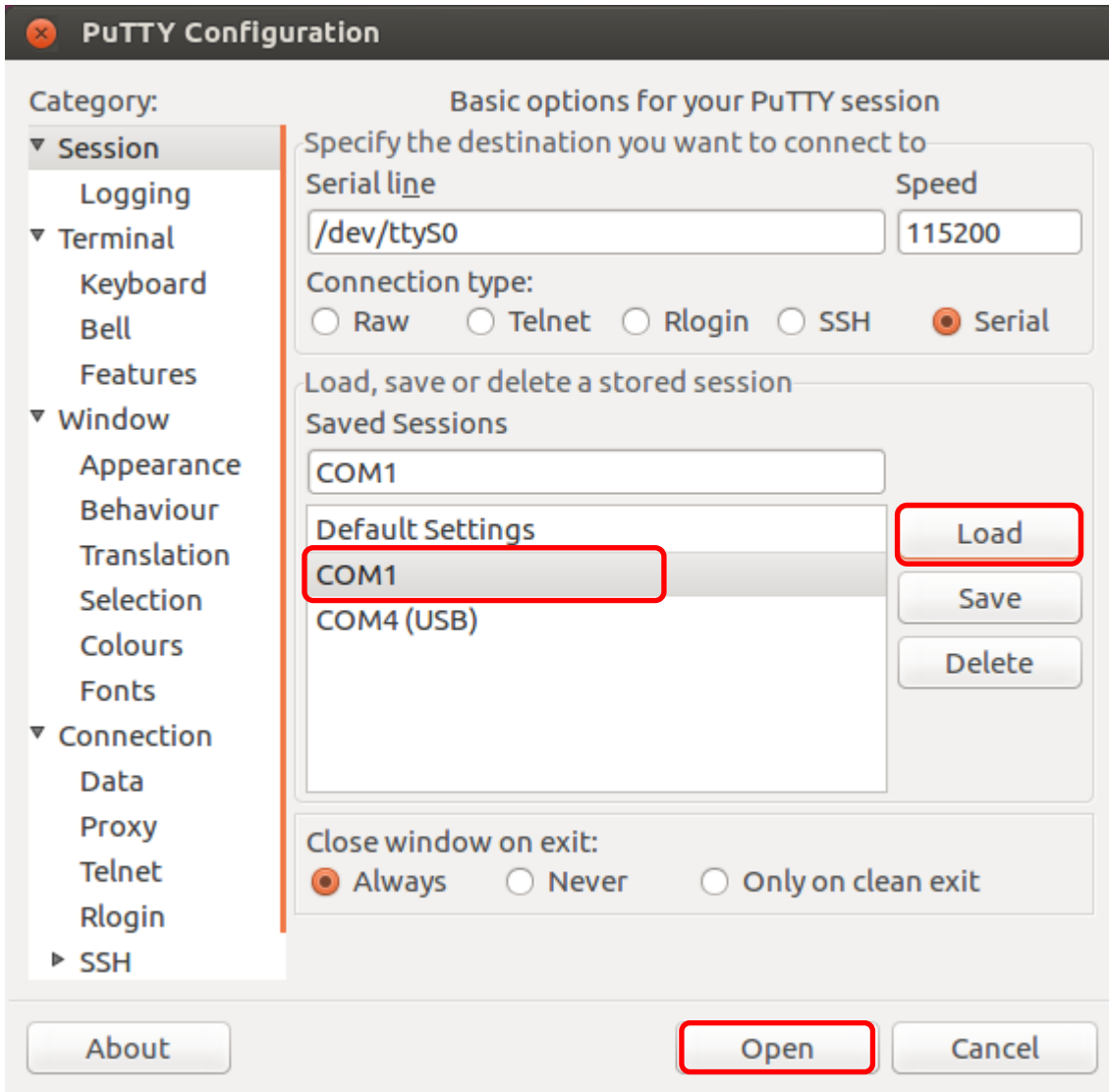


- Select Putty from Applications → Internet → PuTTY SSH Client



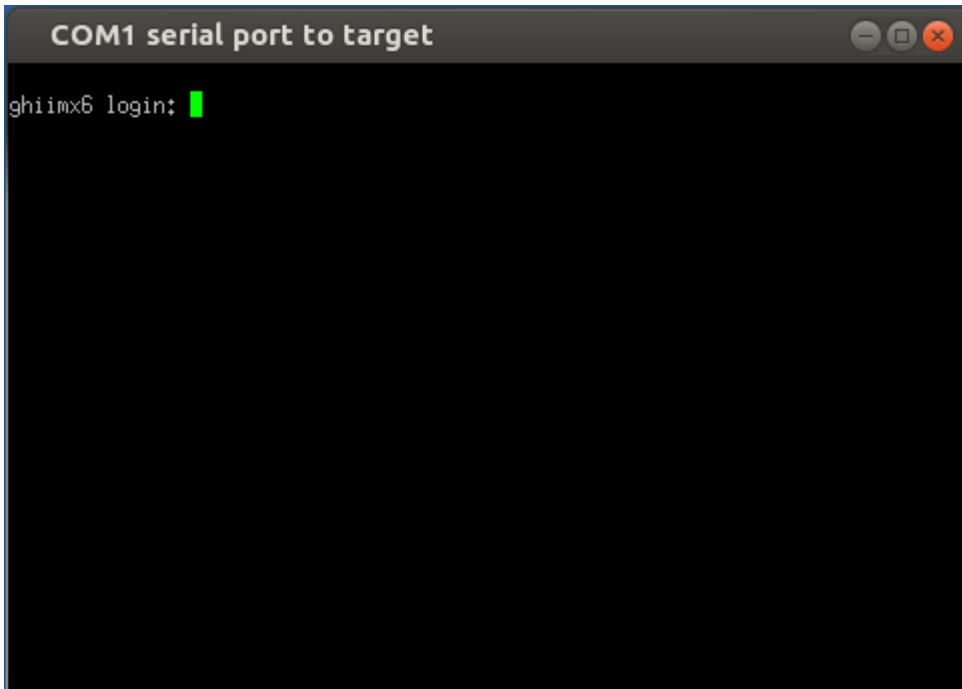
The PuTTY Configuration screen appears.

- Select COM1
- Click “Load”



- Click “Open”

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



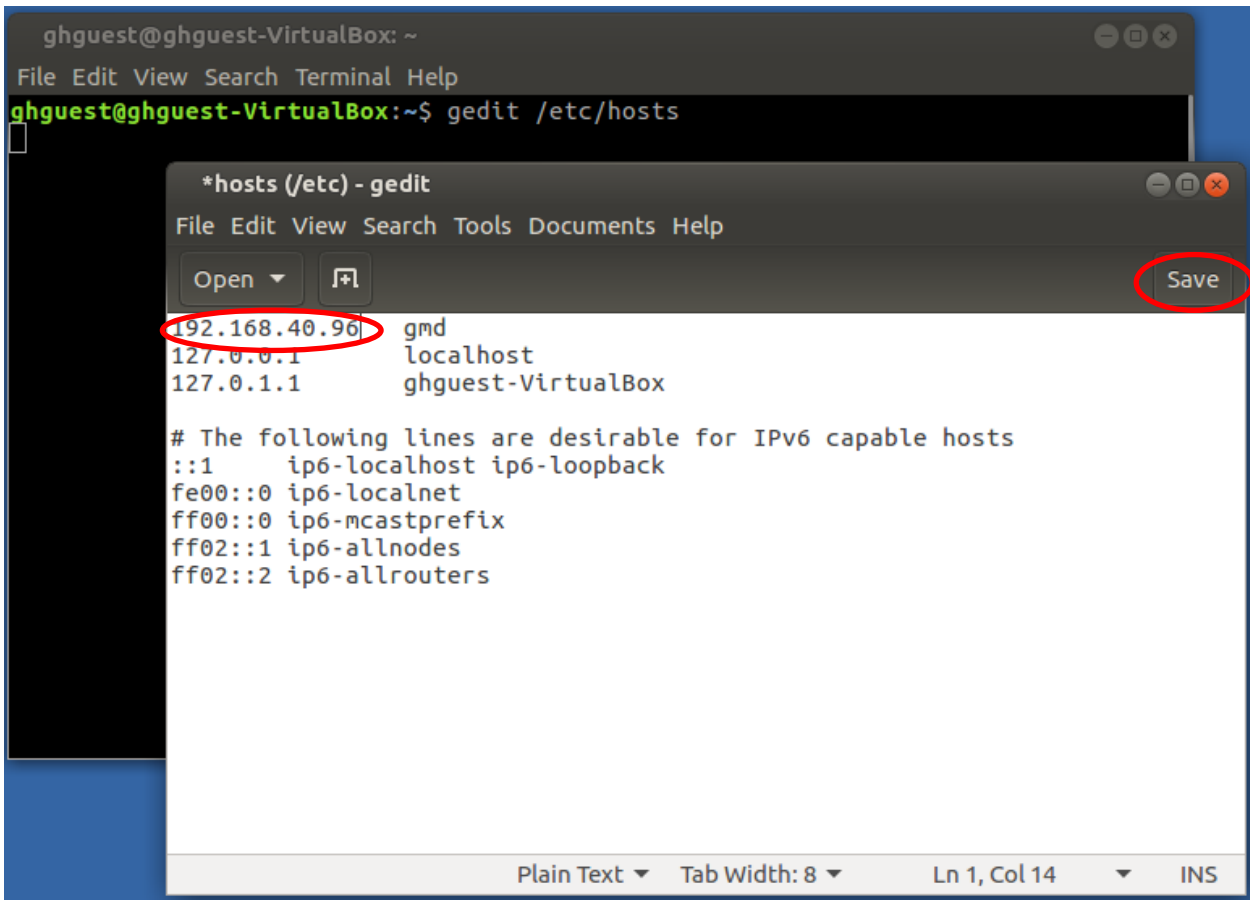
- A “ghiimx6 login:” prompt should appear. If the 3Dxx Display was just powered up; startup messages may appear as well, but when they are done, pressing the “Enter” key should produce a “ghiimx6 login:” prompt as shown.
- At the “ghiimx6 login:” prompt enter “root” (no password is required).
- Depending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



- In the terminal window type the following command:
 - `gedit /etc/hosts`

This opens the IP address configuration file for the Linux system in a text editor program so that the IP address of the 3Dxx Display can be configured.

- Update the IP address associated with “gmd”
- Click “Save”; then close the editing session

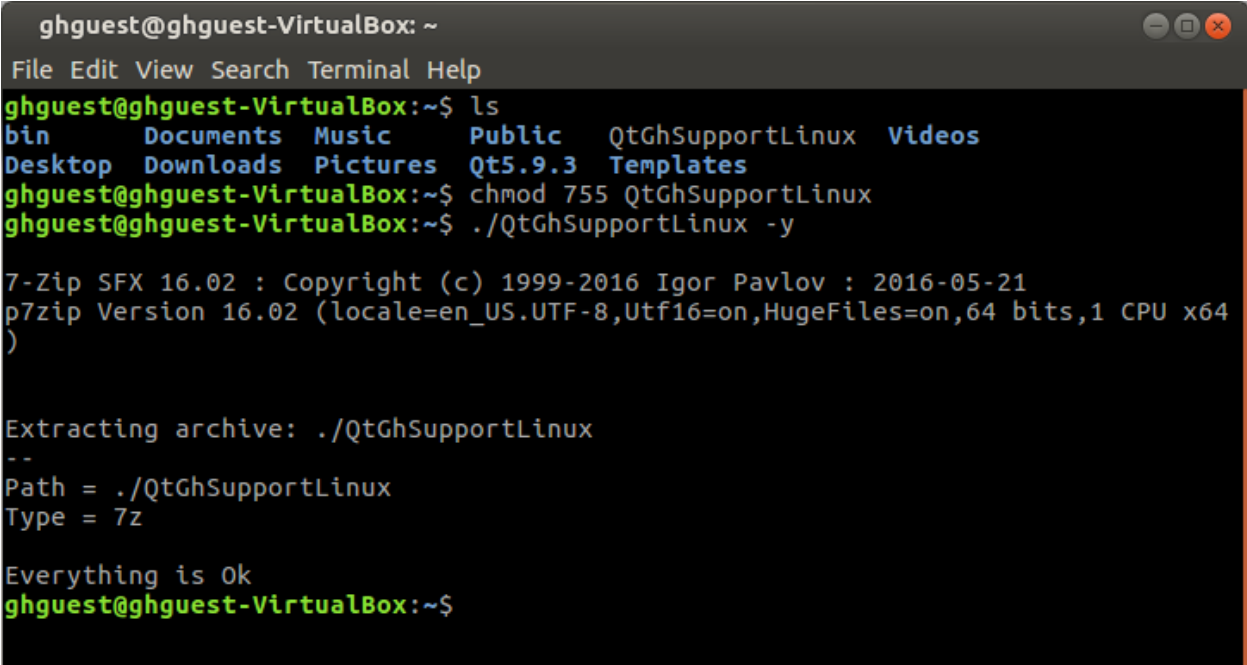


N.B. If the IP address of the display changes; hosts must be updated and Qt Creator re-launched

Configure VM and Display

This section describes the configuration for a 3Dxx Display for operation with the Qt development environment. *QtGhSupportLinux* is a self-extracting archive; when it is executed, the files it contains are extracted. The “-y” flag allows existing files to be replaced. One of the files in the archive, *QtGhSupportLinuxAutoRun*, is the actual installer script and is manually executed after the extraction.

- On the Virtual Machine, launch Firefox (Applications → Internet) and navigate to <http://www.grayhill.com/qt43d>
- Download *QtGhSupportLinux*
- Copy the file to */home/ghguest*, which is just “Home” if after downloading “Open Containing Folder” is selected from Firefox downloads
- Open a terminal window
 - `chmod 755 QtGhSupportLinux`
 - `./QtGhSupportLinux -y`



```
ghguest@ghguest-VirtualBox: ~
File Edit View Search Terminal Help
ghguest@ghguest-VirtualBox:~$ ls
bin      Documents Music      Public  QtGhSupportLinux  Videos
Desktop Downloads Pictures Qt5.9.3 Templates
ghguest@ghguest-VirtualBox:~$ chmod 755 QtGhSupportLinux
ghguest@ghguest-VirtualBox:~$ ./QtGhSupportLinux -y

7-Zip SFX 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU x64
)

Extracting archive: ./QtGhSupportLinux
--
Path = ./QtGhSupportLinux
Type = 7z

Everything is Ok
ghguest@ghguest-VirtualBox:~$
```

- Make the installation script executable
 - `chmod 755 QtGhSupportLinuxInstall`

- *QtGhSupportLinuxInstall* installs VM support files and/or installs the necessary libraries on the display based on the arguments provided.
 - `./QtGhSupportLinuxInstall [vm | 3dxx | -h]`

```
ghguest@ghguest-VirtualBox: ~
File Edit View Search Terminal Help
ghguest@ghguest-VirtualBox:~$ ./QtGhSupportLinuxInstall -h

USAGE: QtGhSupportLinuxInstall [vm | 3dxx | -h]

installs files on the VM or 3Dxx

    -h          displays this help message

Ex: QtGhSupportLinuxInstall          installs files on both
Ex: QtGhSupportLinuxInstall vm       install VM files only
Ex: QtGhSupportLinuxInstall 3dxx    install 3Dxx display files only

ghguest@ghguest-VirtualBox:~$ █
```

- `./QtGhSupportLinuxInstall vm`

```
ghguest@ghguest-VirtualBox: ~
File Edit View Search Terminal Help
ghguest@ghguest-VirtualBox:~$ ./QtGhSupportLinuxInstall vm
Installing/Updating files on the VM...

Virtual Machine configuration complete!

ghguest@ghguest-VirtualBox:~$ █
```

- ./QtGhSupportLinuxInstall 3dxx
- Display installation and configuration takes a few minutes to execute. If everything works correctly the script concludes with the following:
 setup3Dxx completed successfully... rebooting

If this message does not appear, the problem(s) **must** be corrected before continuing.

```
ghguest@ghguest-VirtualBox: ~  
File Edit View Search Terminal Help  
ghguest@ghguest-VirtualBox:~$ ./QtGhSupportLinuxInstall 3dxx  
  
Transferring files to display...  
Configuring the display...  
  
Setup Grayhill 3Dxx Display for Qt-5.9.3  
Fri Jun  8 10:16:41 CDT 2018  
Making 3Dxx Flash File system writeable and creating writeablefs script  
Disabling VUI Builder and other applications on 3Dxx  
Installing IPKs...  
Updating GCC Libraries on 3Dxx Display  
Installing gcclibs (4.8.3) to root...  
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 rw rootwait board=ghi_imx6.pn=3D70VT-100  
Updating /etc/profile script  
Updating /etc/profile.local script  
Updating /usr/lib/fonts  
setup3Dxx completed successfully... rebooting  
  
ghguest@ghguest-VirtualBox:~$ █
```

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:

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

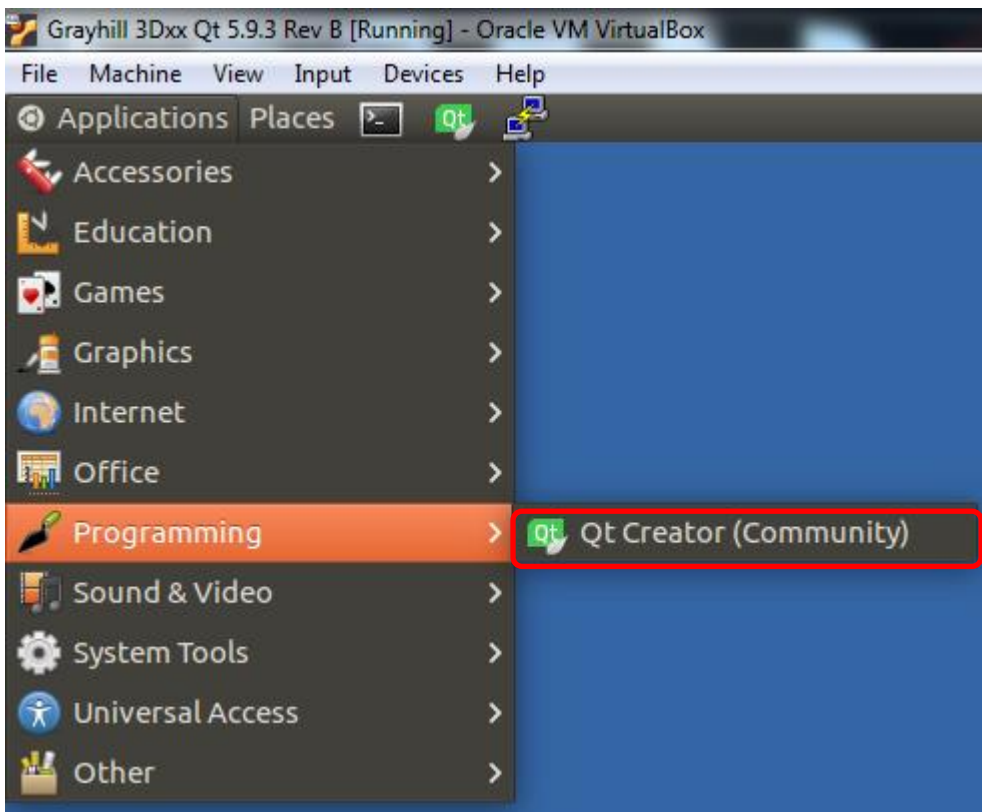
Build and Run 3Dxx Embedded Application (Widget)

The VM comes ready to run (aside from the previous configuration steps). This section details how to build and run a demo application.

Complete instructions for configuration can be found in the appendices.

If not already started, launch Qt Creator by performing one of the following actions:

- Select “Applications” (word in the upper left-hand corner of the Linux window), then navigate through “Programming” and click on “Qt Creator ...”



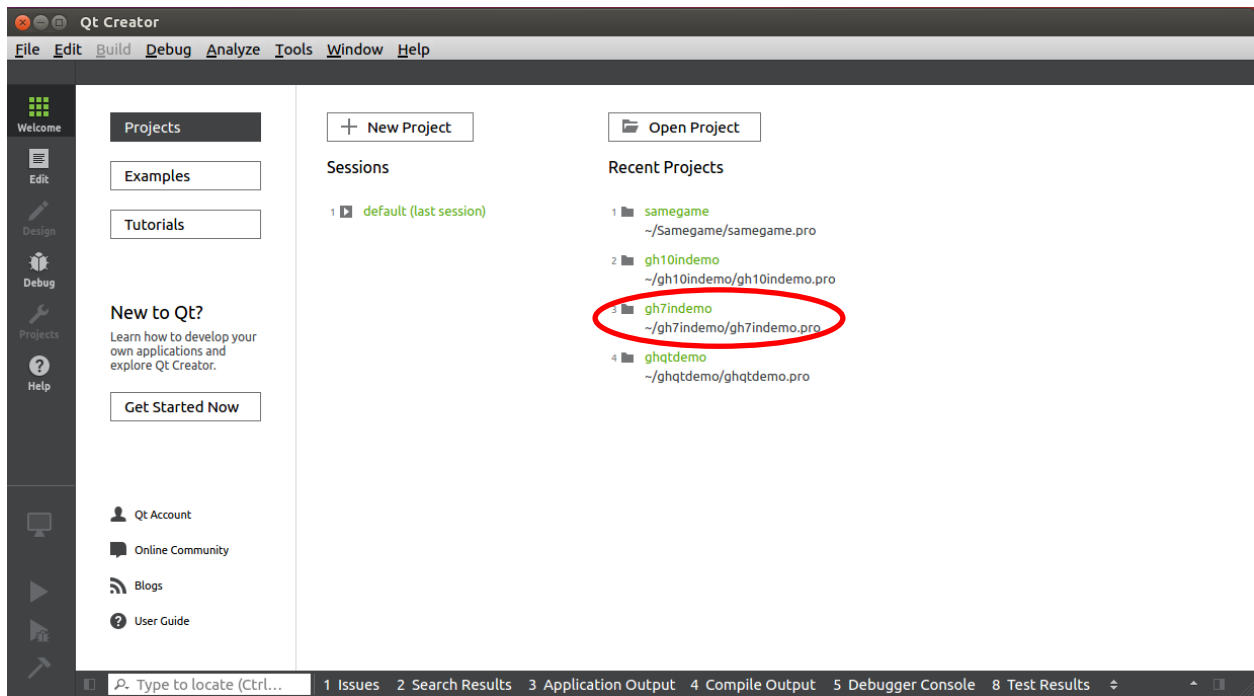
- Click on the Qt icon in the panel



- Double click on the Qt icon on the desktop

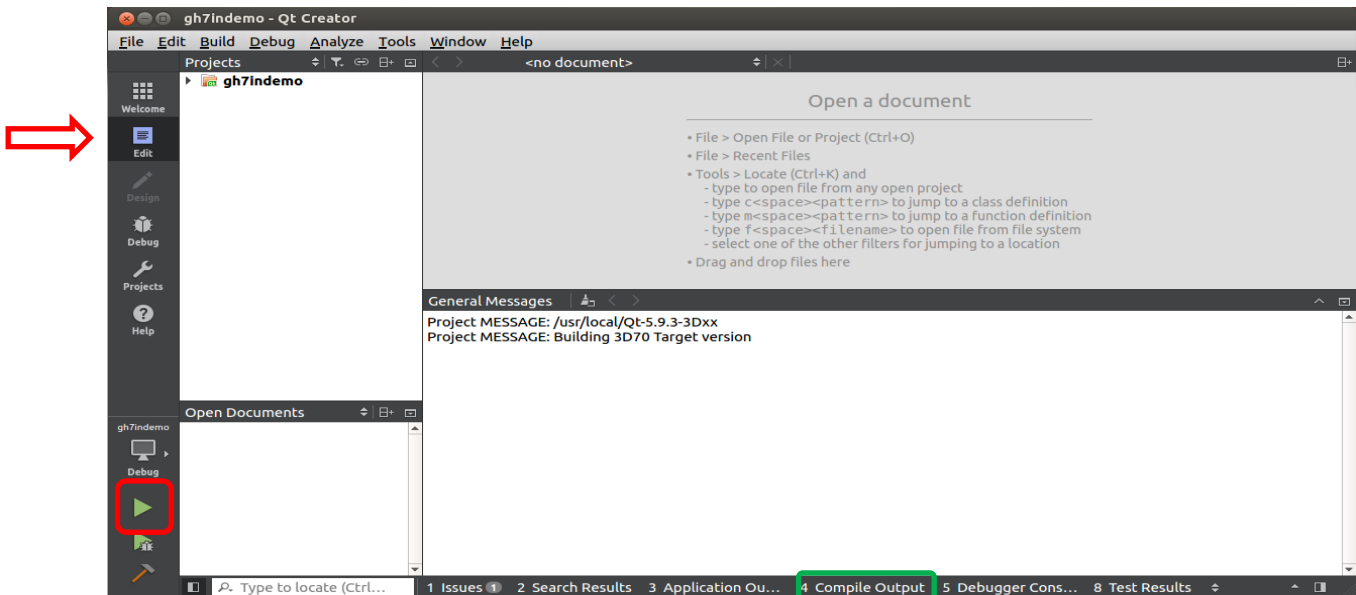


Select the gh7indemo project from “Qt Creator” main window. Click on “gh7indemo”.

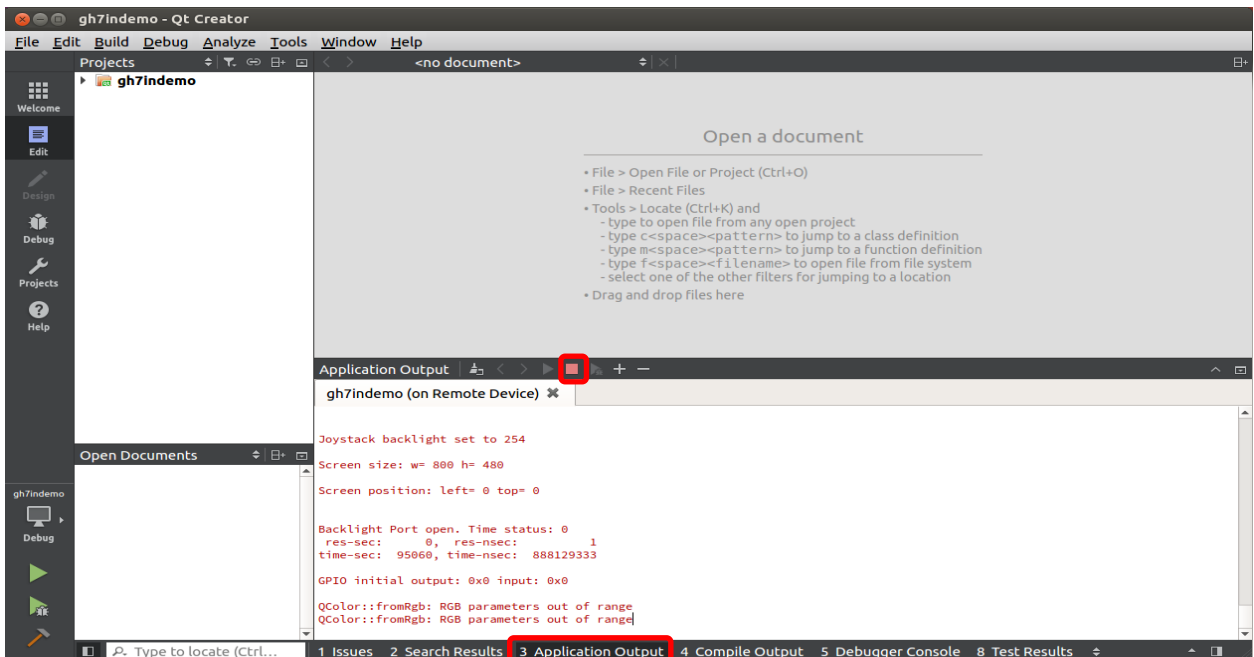


Note: To open a non-recent project; select “Open Project” and navigate to the project’s .pro file

The “Edit” view is displayed. Click on the green arrow to run (a check for building is also done).



- 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)

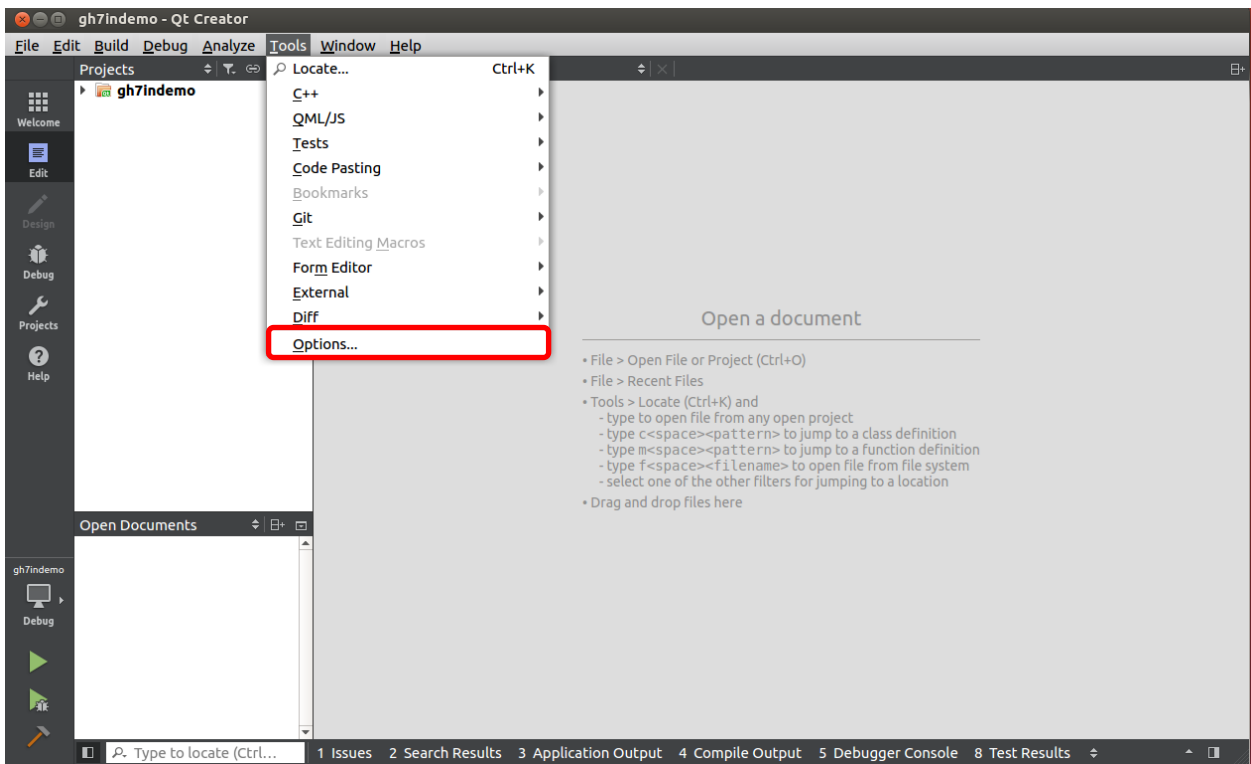


- Select the “Application Output” tab.
- Clicking the red square terminates the target session.

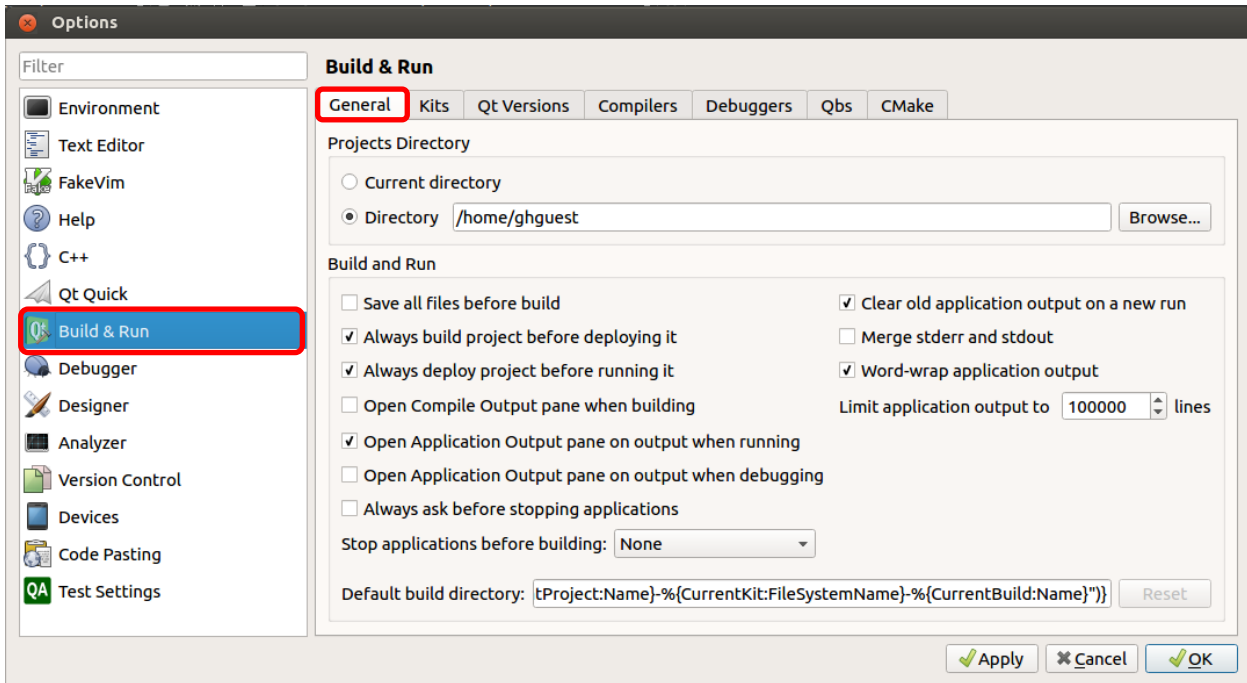
Appendix A: Configuring a Manual Qt Kit for Grayhill Displays

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.

- Launch Qt Creator
- Select Tools → Options



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

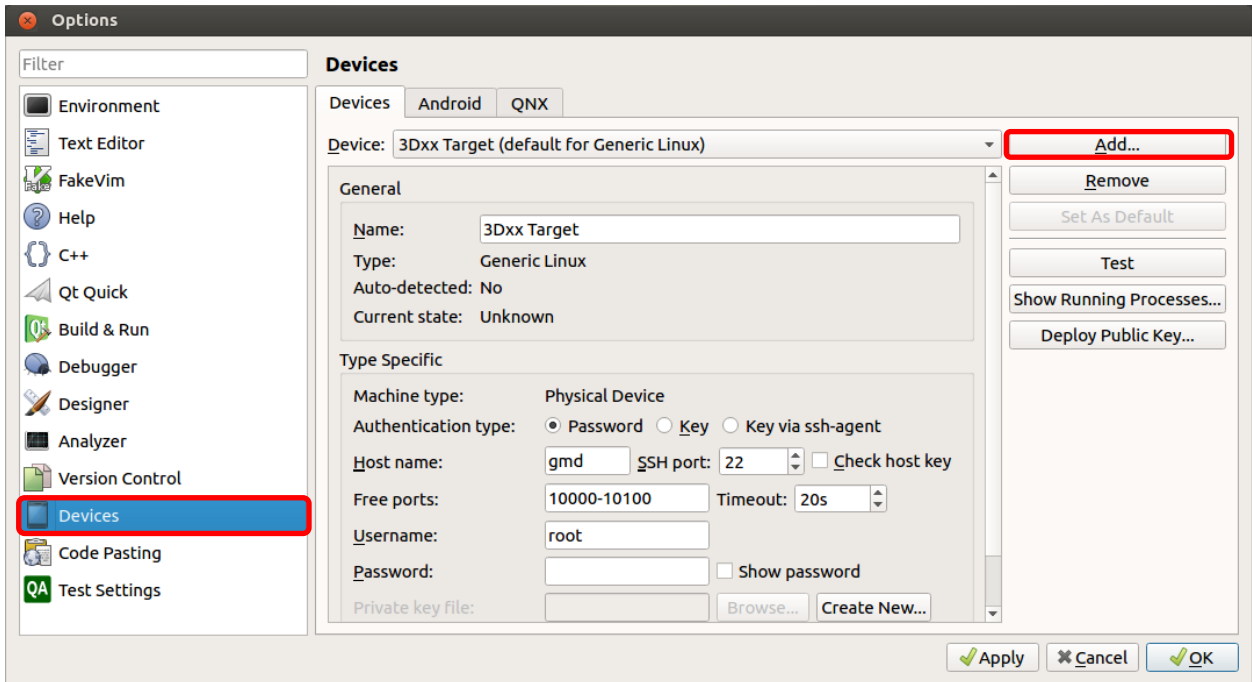


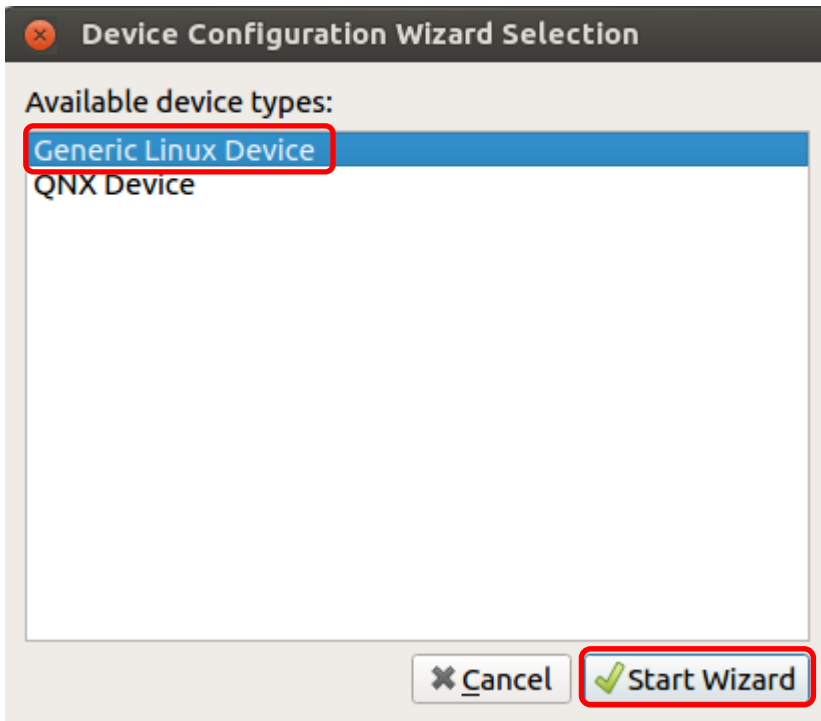
The “General” tab is where project wide customization is done. Review and select the desired configuration.

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

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

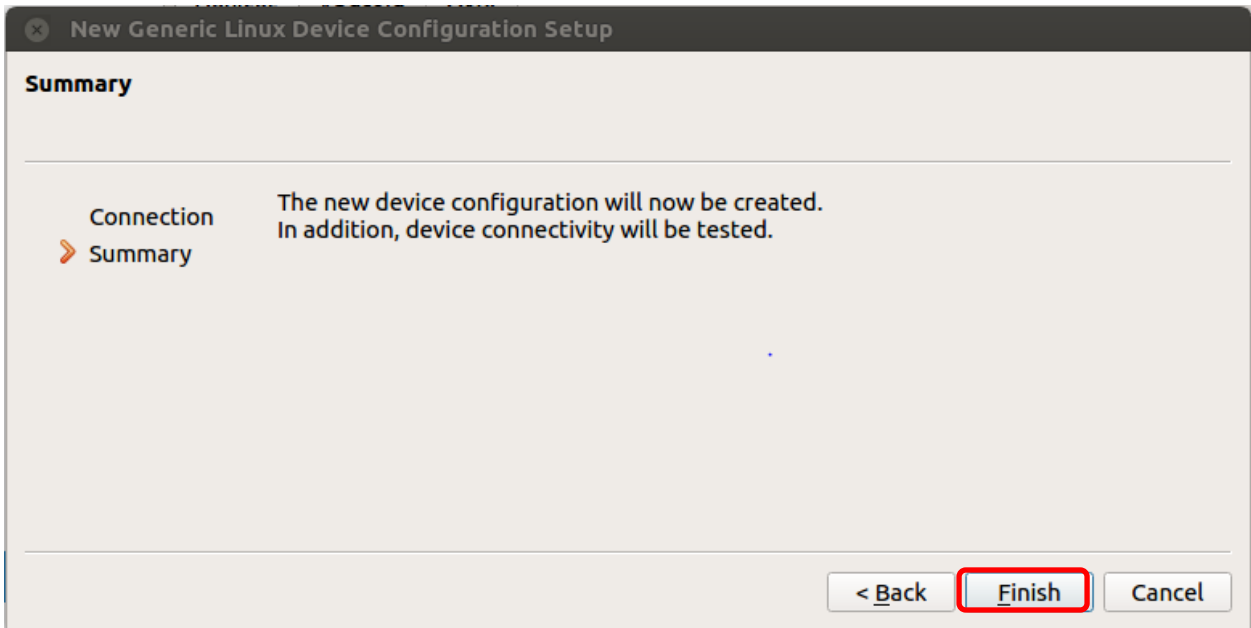
The authentication type: Password Key Agent

The user's password: [Empty]

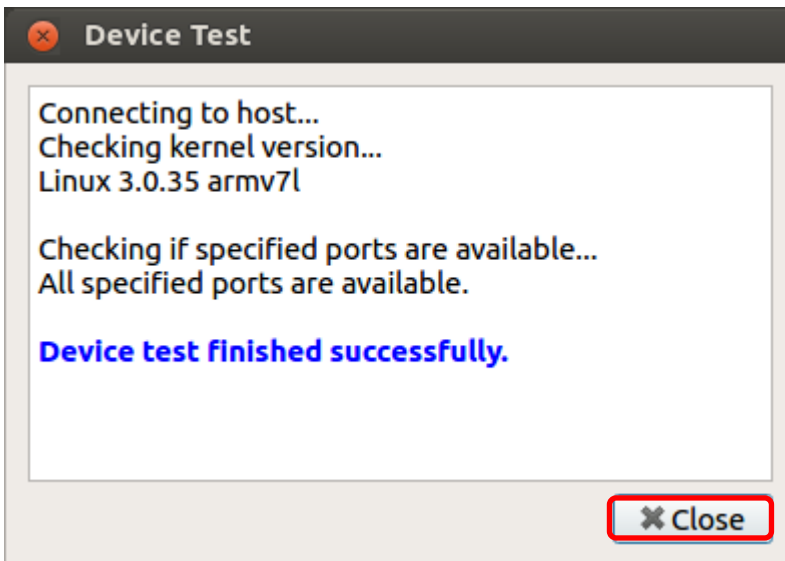
The file containing the user's private key: /ghguest/.ssh/id_rsa [Browse...]

Next > Cancel

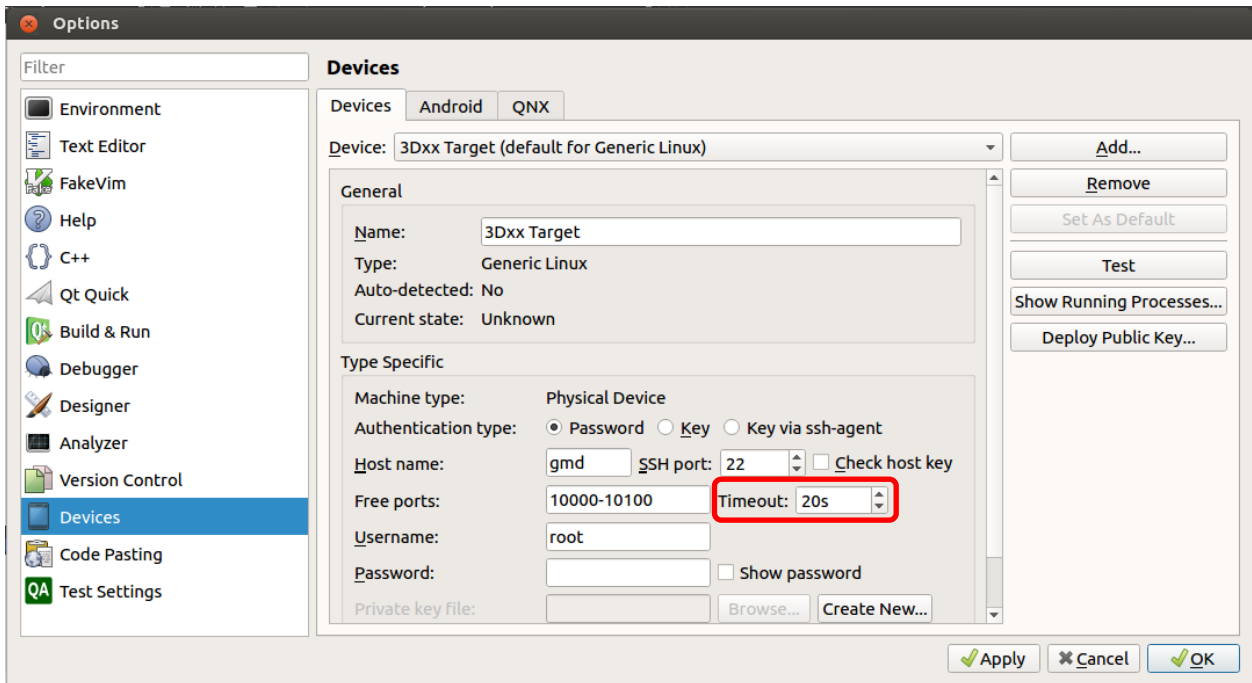
- Populate the fields as illustrated above
- N.B. The IP address associated with gmd was configured in */etc/hosts* (C:\Windows\System32\drivers\etc)
- Click “Next”



- 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

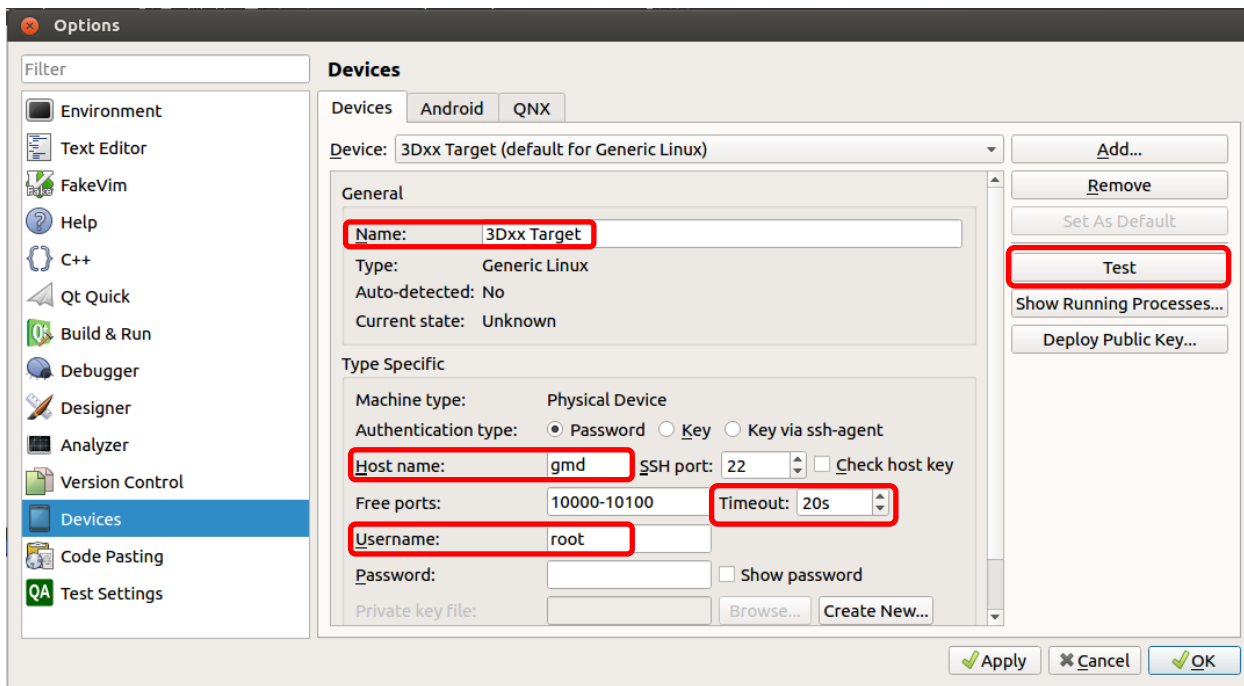


- Click “Close”



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

- Devices Summary

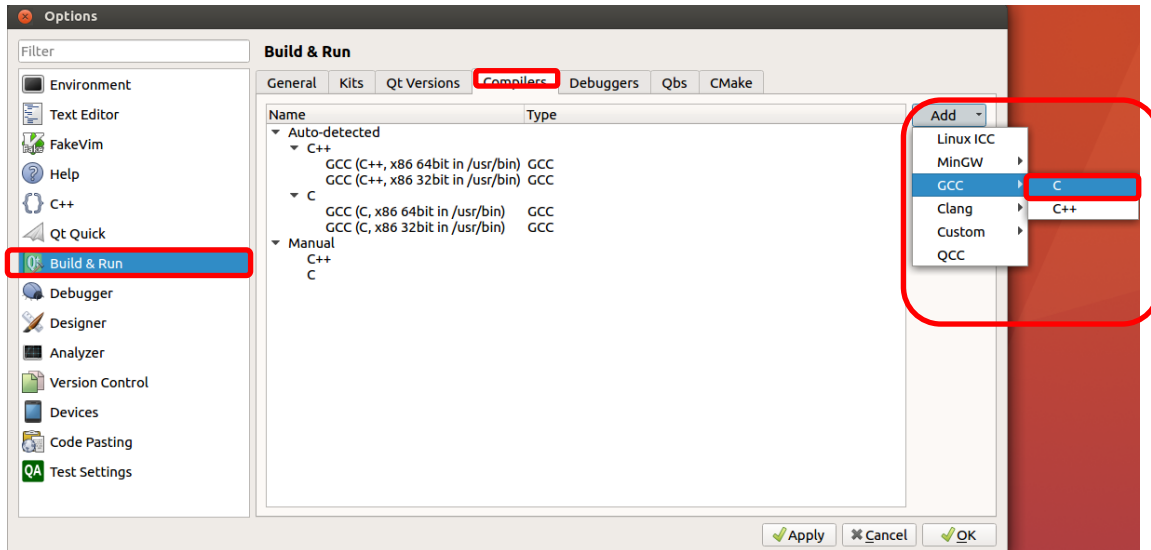


- Name name of the device
- Host name can be “alias” like *gmd* specified in */etc/hosts* or a hard coded IP
- Timeout 20s
- Username root

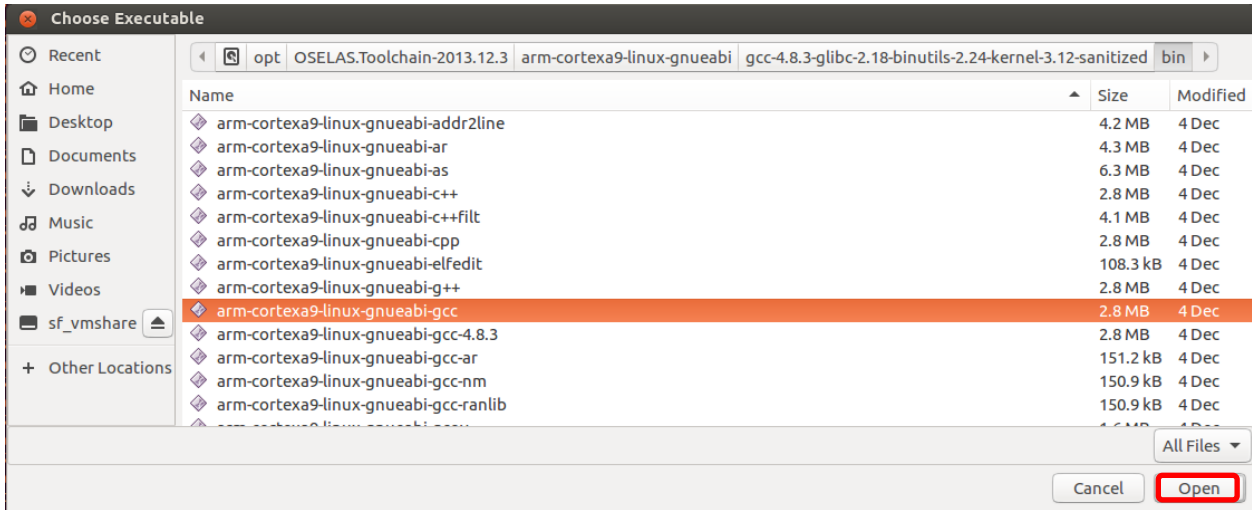
N.B. Remember to “Test” to make sure connectivity has been established

Compiler

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



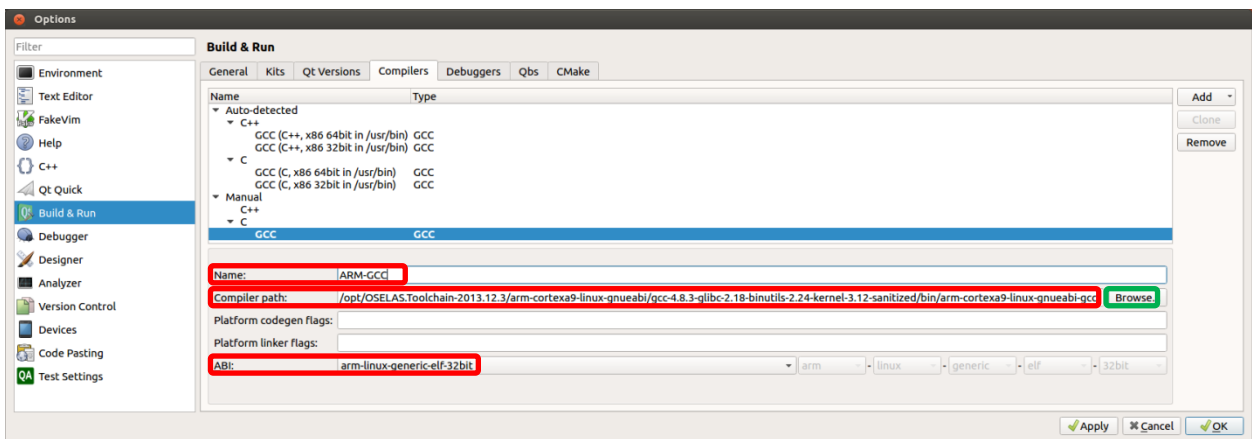
- Populate the fields as illustrated
 “Name:” ARM-GCC
 “Compiler path:” Click “Browse...” and navigate to the desired file
 /opt/OSELAS.Toolchain-2013.12.3/arm-cortexa9-linux-gnueabi/gcc-4.8.3-glibc-2.18-binutils-2.24-kernel-3.12-sanitized/bin/arm-cortexa9-linux-gnueabi-gcc



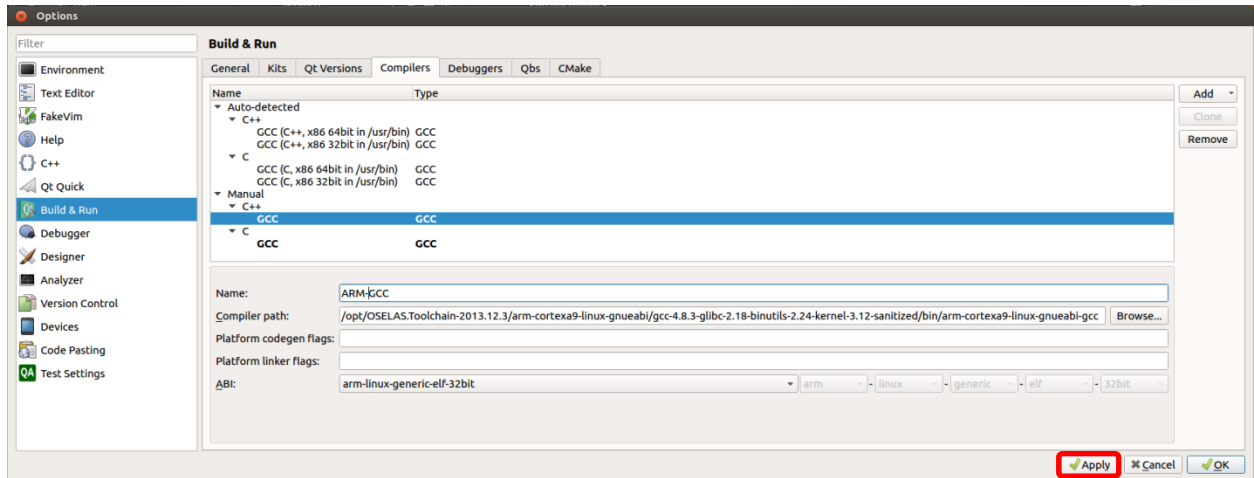
Click “Open”

“ABI:” Select “arm-linux-generic-elf-32bit” (if not automatically pre-selected)

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



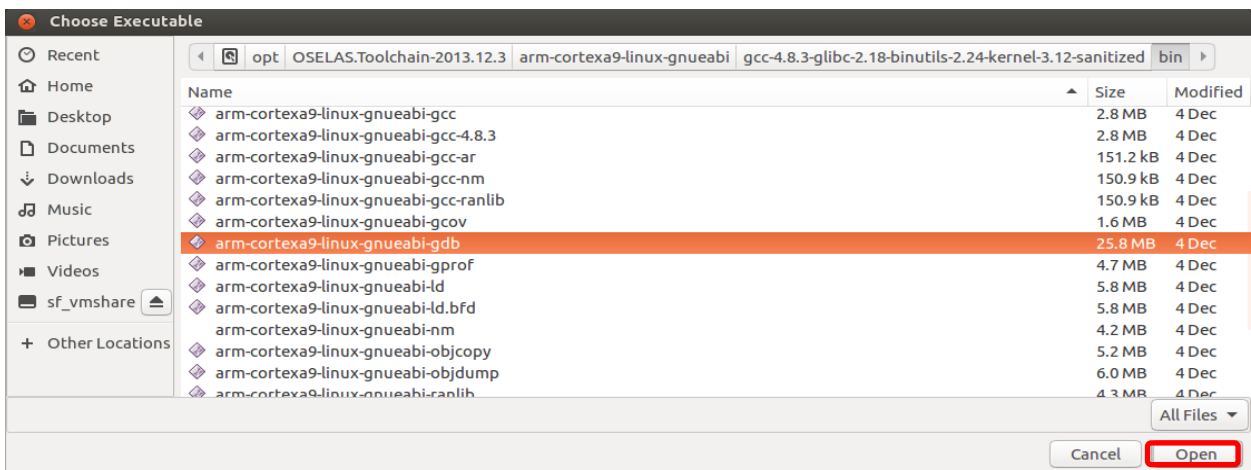
- Repeat the above steps for GCC→C++



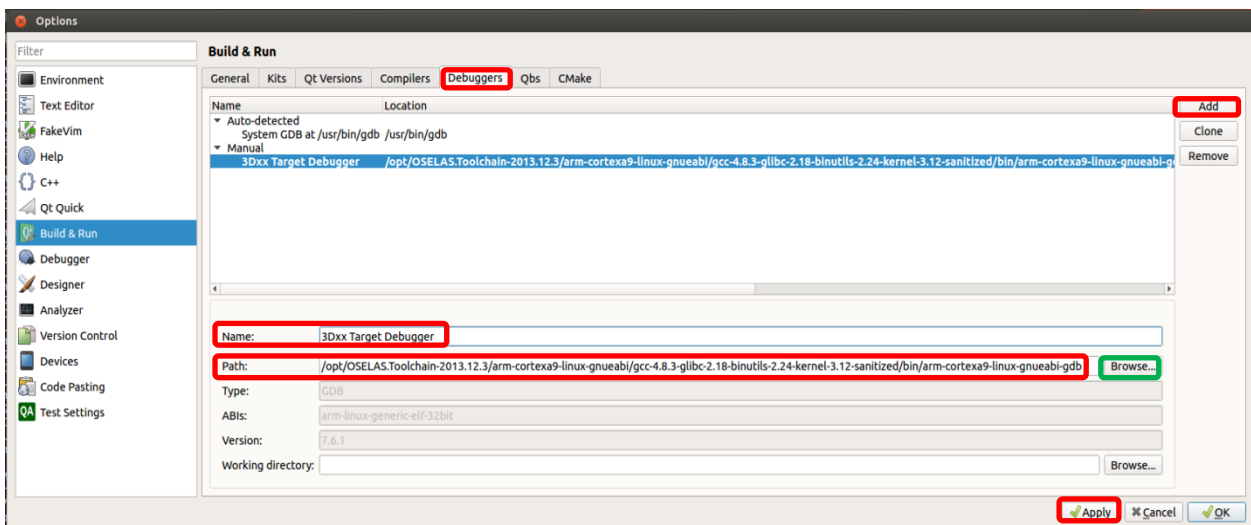
- Click “Apply”

Debugger

- Select the “Debuggers” tab
- Click “Add”
- Populate the fields as illustrated
 - “Name:” 3Dxx Target Debugger
 - “Path:” Click “Browse...” and navigate to the desired file (should be previous directory)
/opt/OSELAS.Toolchain-2013.12.3/arm-cortexa9-linux-gnueabi/gcc-4.8.3-glibc-2.18-binutils-2.24-kernel-3.12-sanitized/bin/arm-cortexa9-linux-gnueabi-gcc



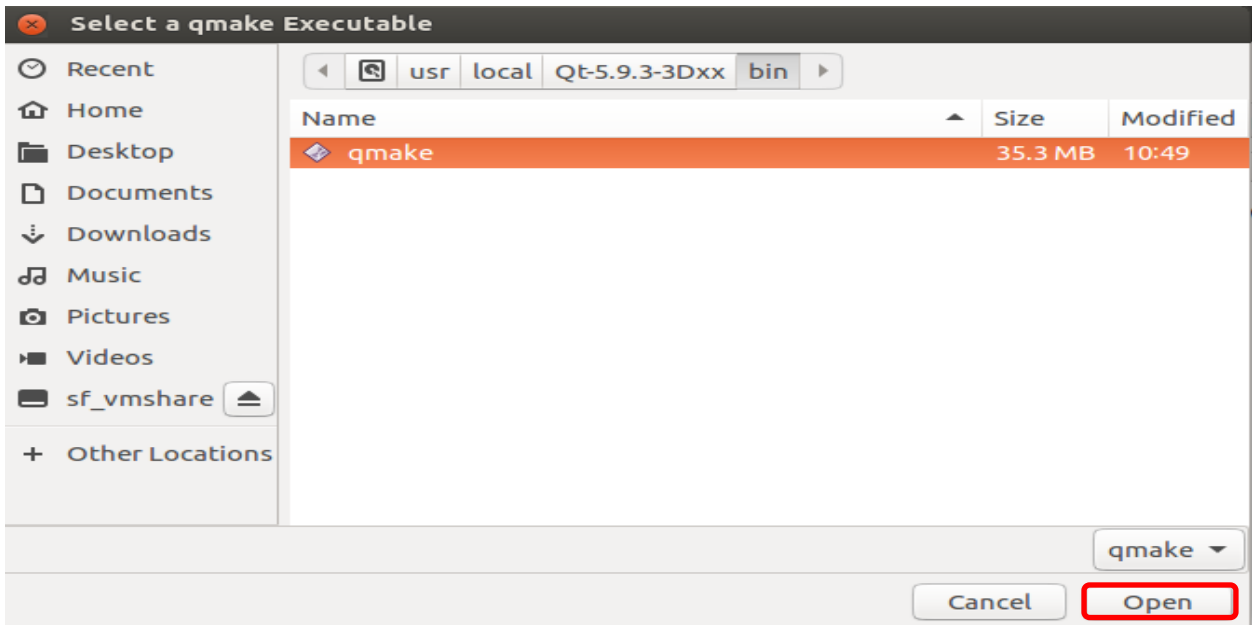
- Click “Open”; the configuration portion of the screen should look similar to



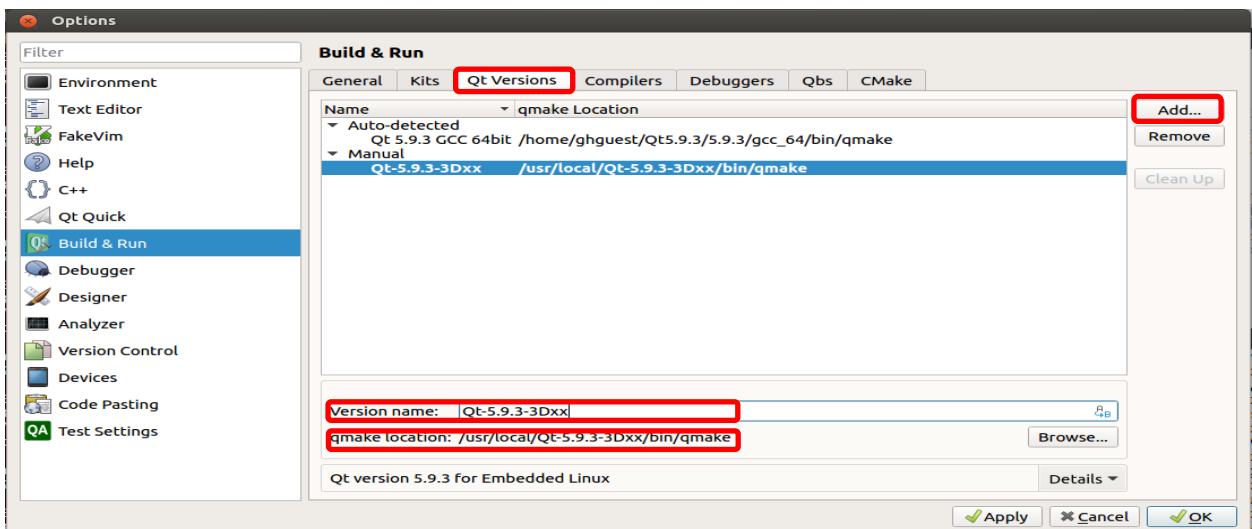
- Click “Apply”

qmake

- Select the “Qt Versions” tab
- Click “Add” (Select a qmake Executable dialog box appears, still referencing the last path)
- Navigate to the qmake version associated with the library (e.g. “/usr/local/Qt-5.9.3-3Dxx/bin/qmake”)

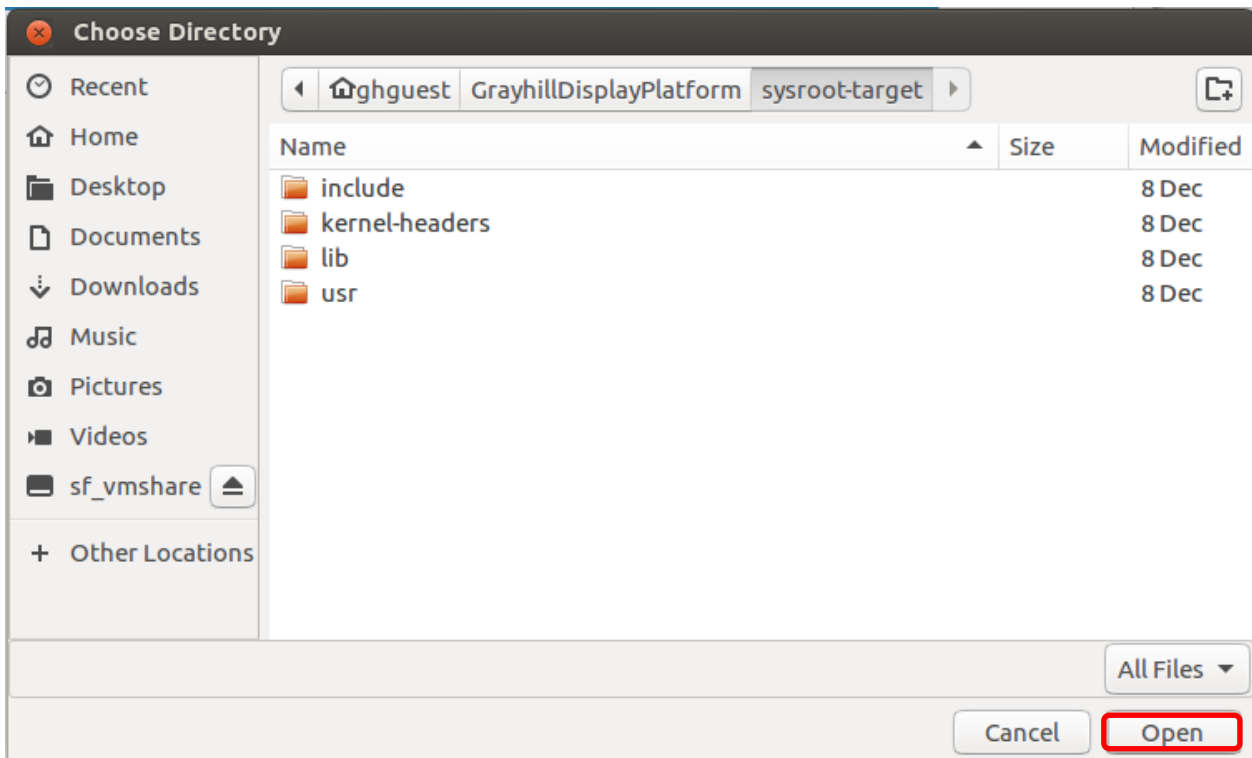


- Click “Open”
- Update “Version name:” to “Qt-5.9.3-3Dxx”



Kit

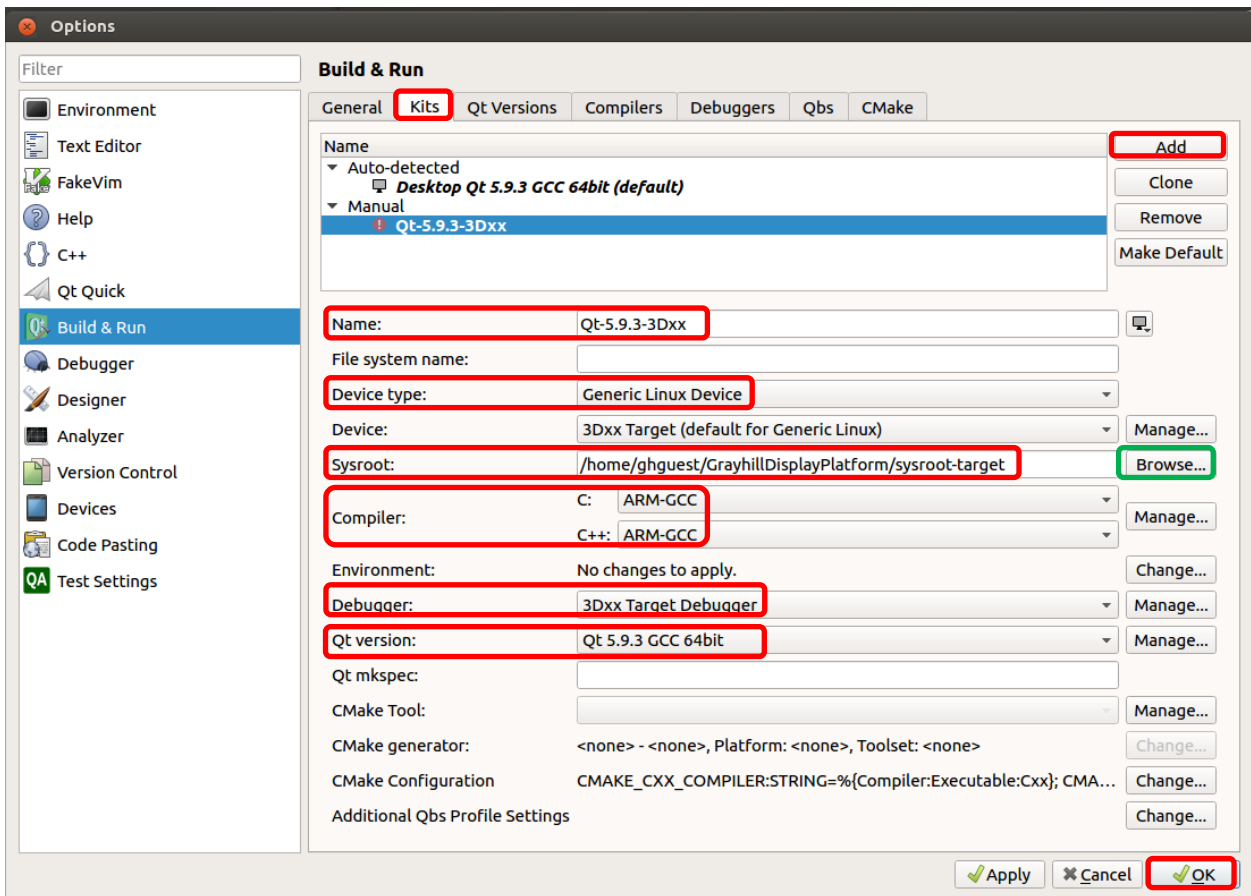
- Select the “Kits” tab
- Click “Add”
- Populate the fields as illustrated
 - “Name:” Qt-5.9.3-3Dxx
 - “Device type:” Select “Generic Linux Device” from the pick list
N.B. Automatically updates Device
 - “Sysroot:” Click “Browse...” and navigate to desired path
/home/ghguest/GrayhillDisplayPlatform/sysroot-target



Click “Open”

- “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.

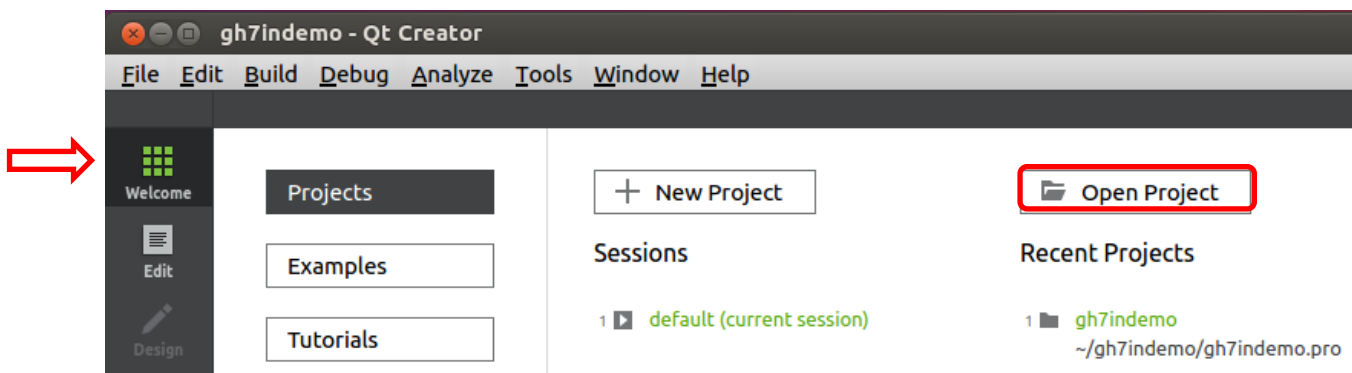
Appendix B: Configuring a 3Dxx Project

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.

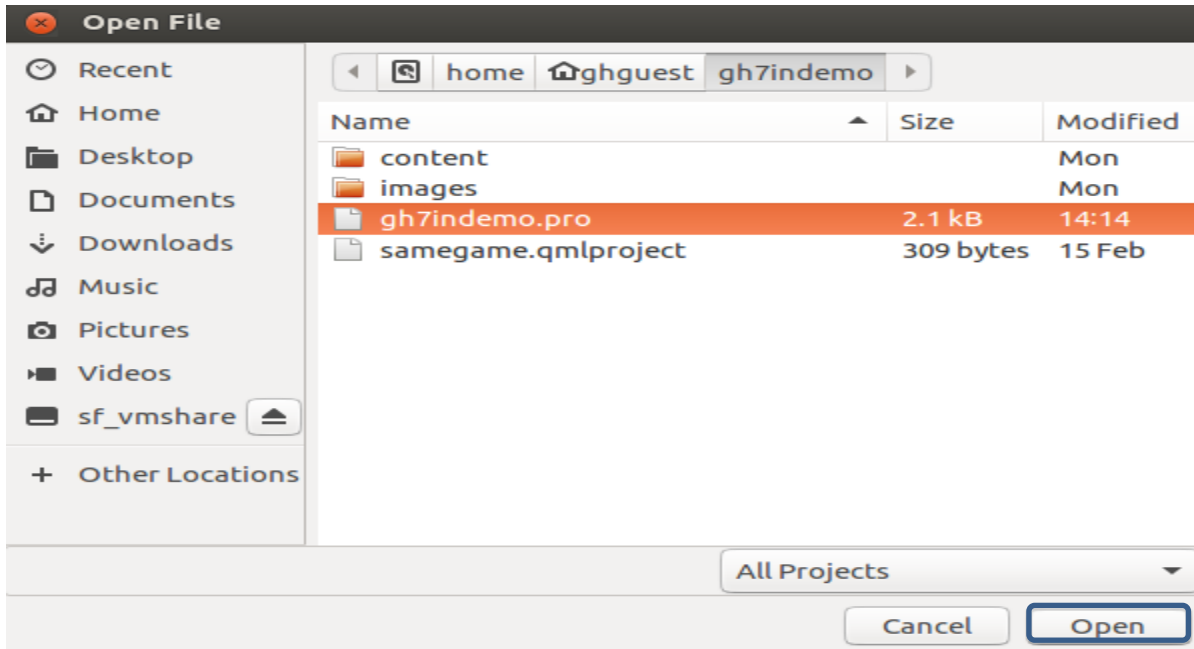
If not already running, launch Qt Creator. (See **Build and Run 3Dxx Embedded Application**)

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

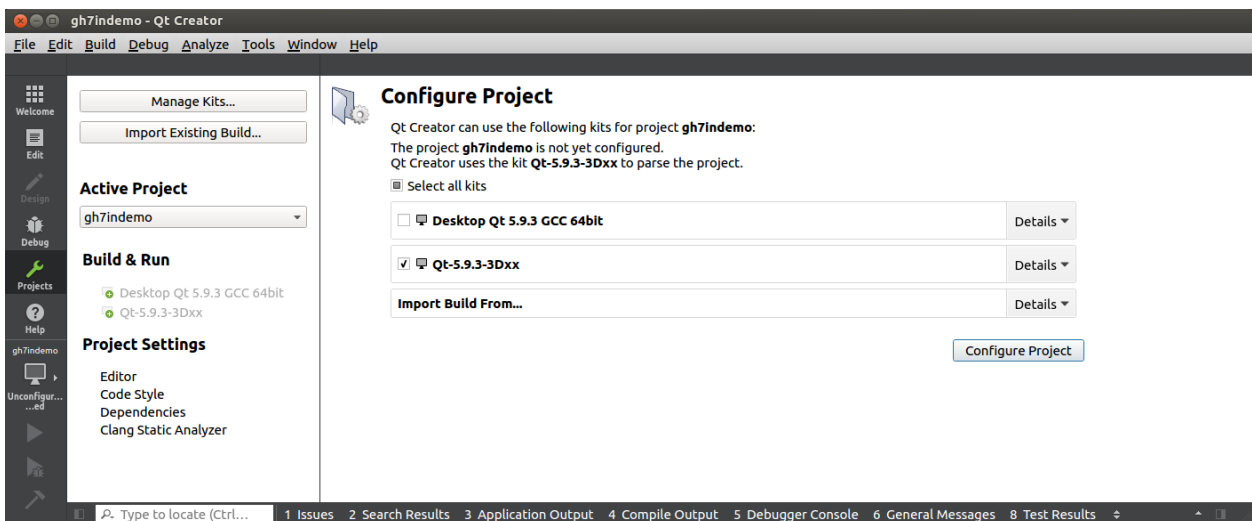
N.B. If present, a previous project can be opened by clicking on the project name listed below “Recent Projects”.



- An “Open File” dialog window will appear
- Navigate to the 3Dxx Demo project’s “.pro” file as shown in the example below;



- 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

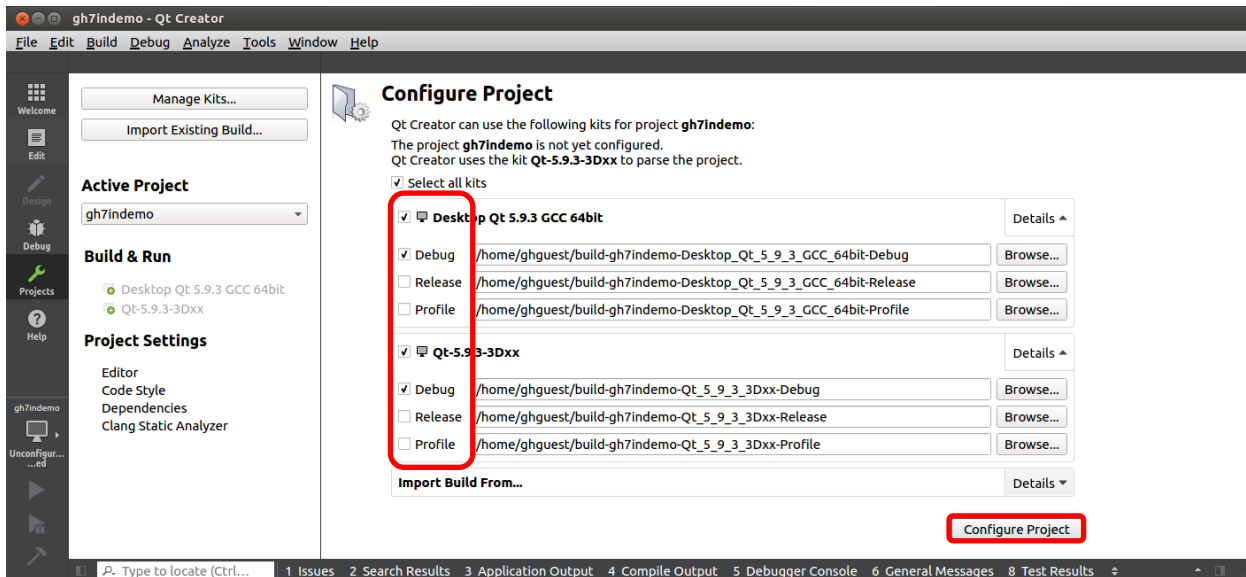


“Desktop Qt 5.9.3 GCC 64bit”

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

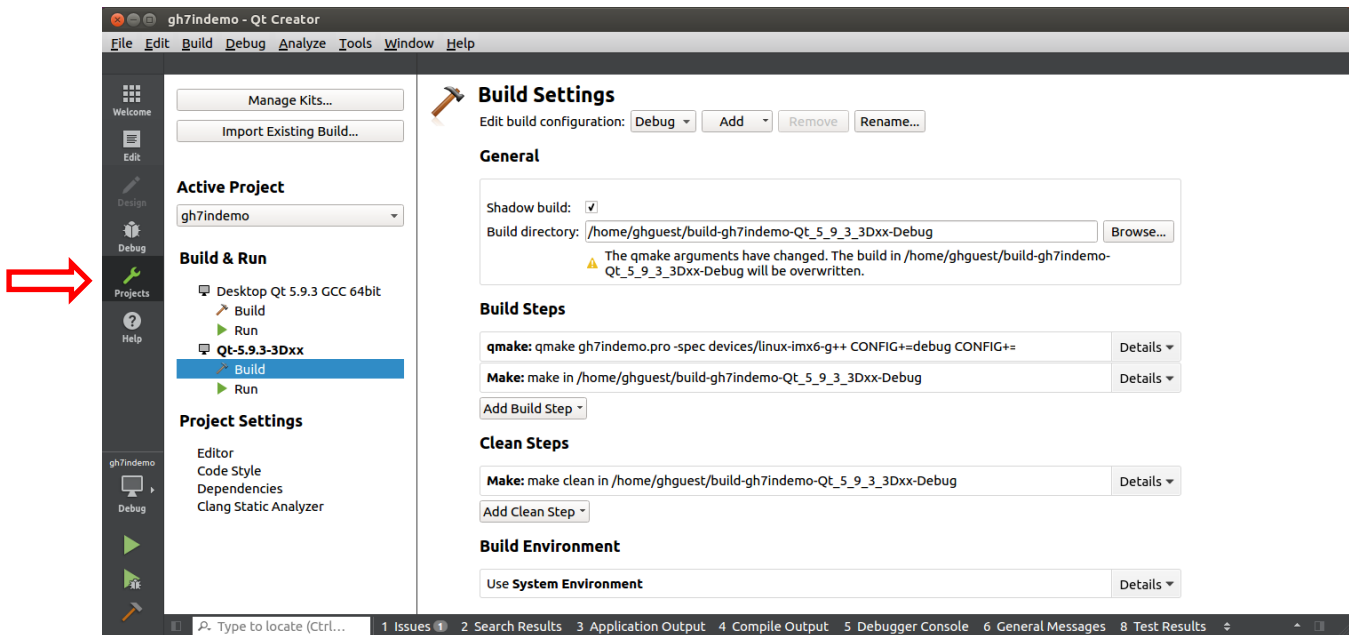
“Qt-5.9.3-3Dxx”

- Expand by clicking on “Details”
Select “Debug”
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.

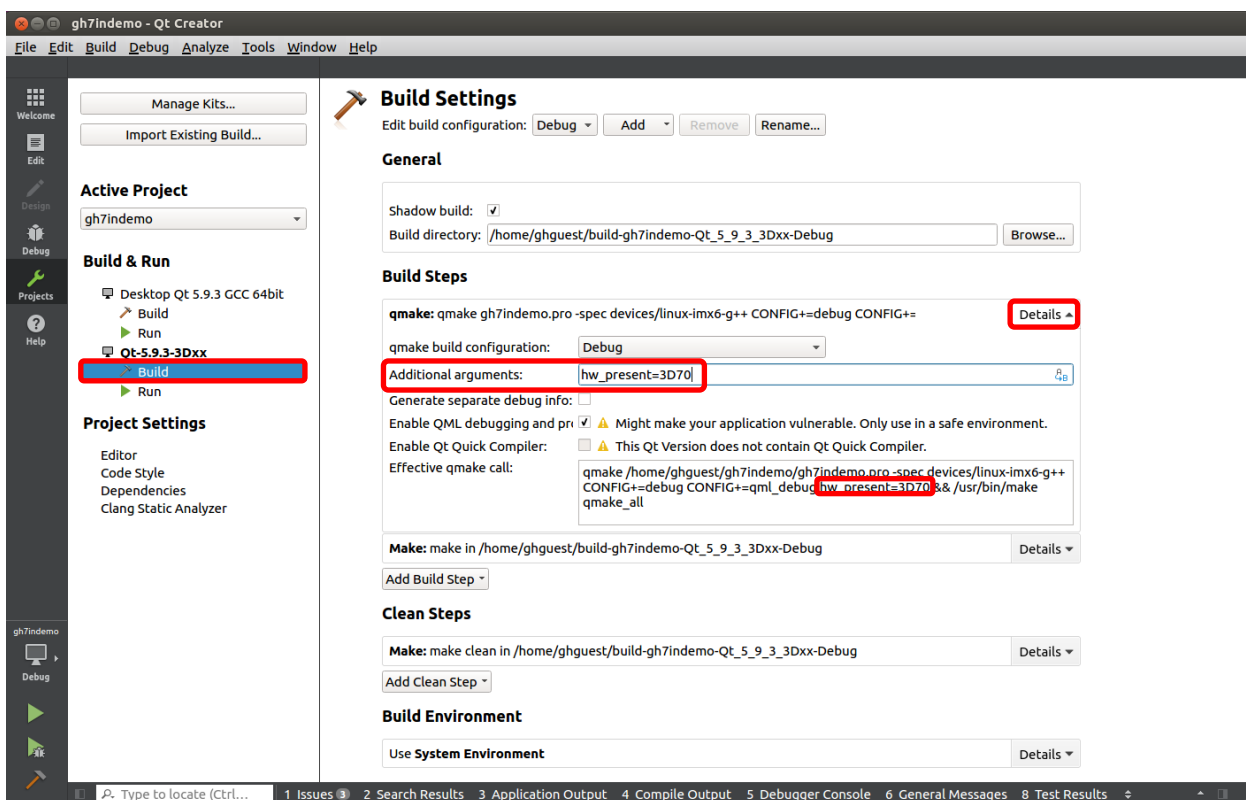
N.B. Clicking on an actual kit name selects either Build or Run (depending on which one was previously selected)

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.



Run

- Select “Run”
- Deployment
 - Method: Deploy to Remote Linux Host (should be default)
 - Files to deploy:

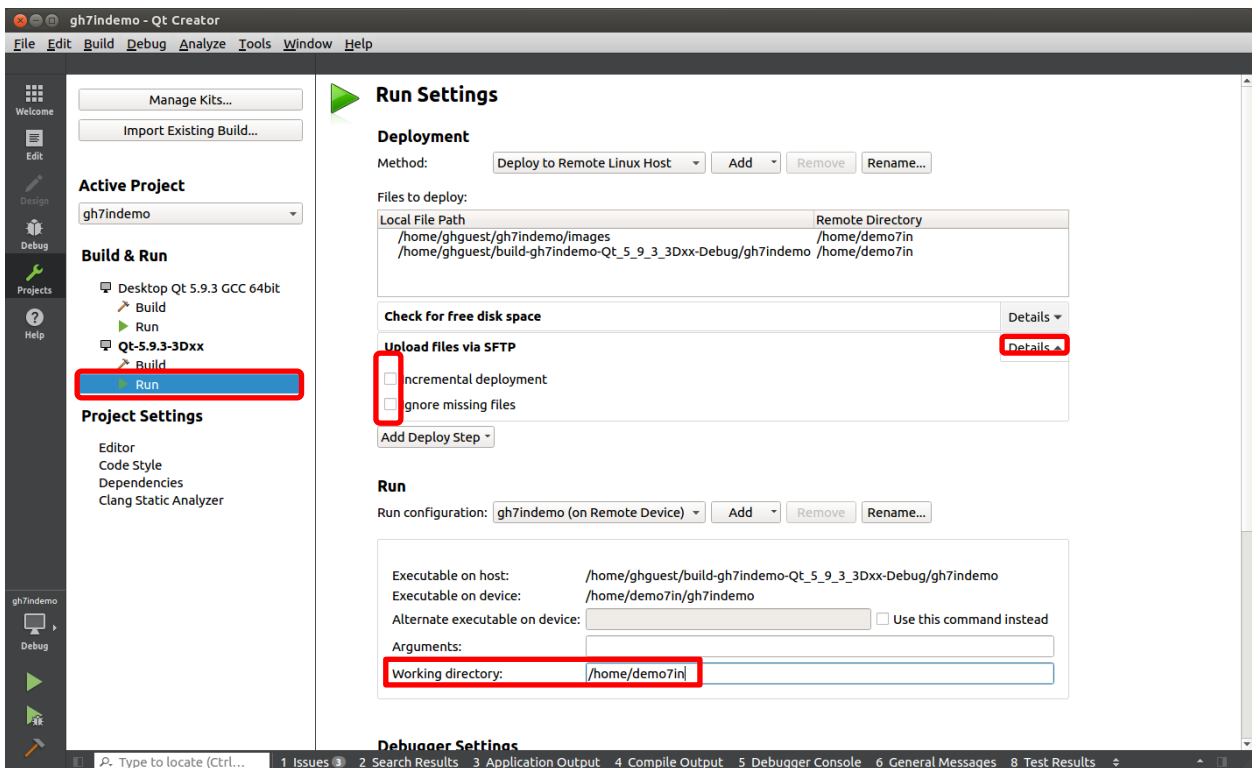
Local File Path	location of the local file(s)	(auto-populated)
Remote Directory	location on the target	(auto-populated)

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

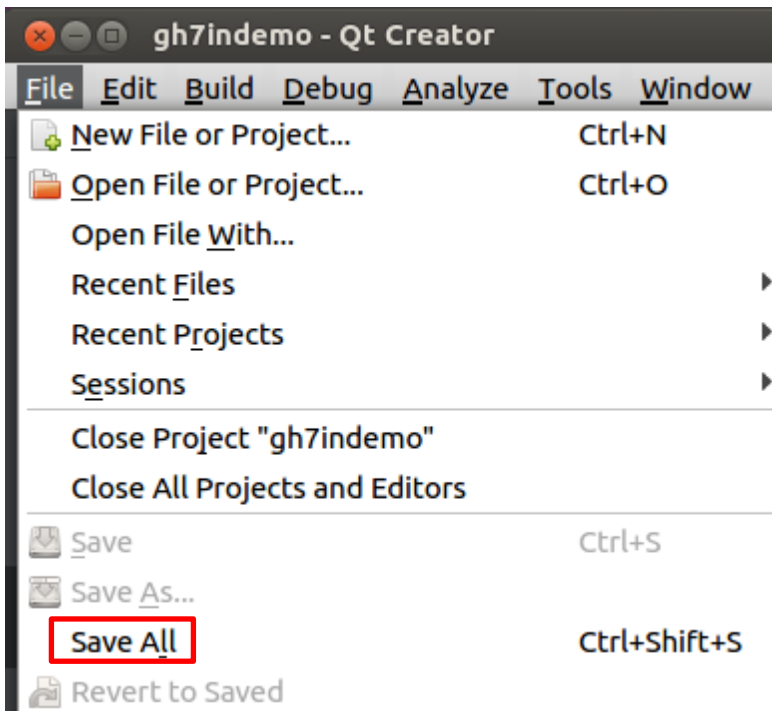
- Expand “ Details” for “Upload files via SFTP”

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.

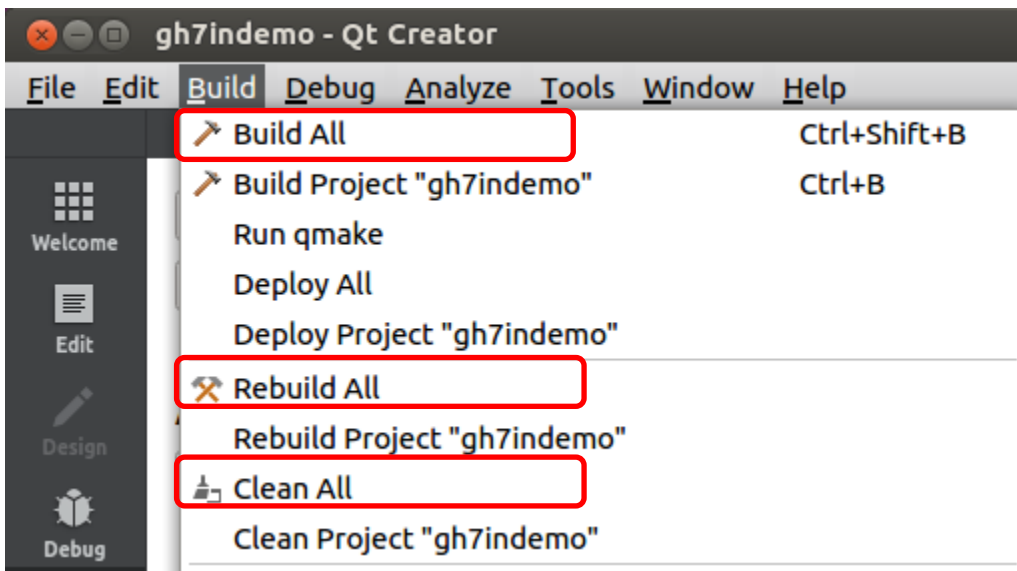
- Make sure neither box is selected
- Set “Working directory:” under Run to the directory associated with the “Executable on device:”
- Enter “/home/demo7in” in the box



- **Save!** File → Save All



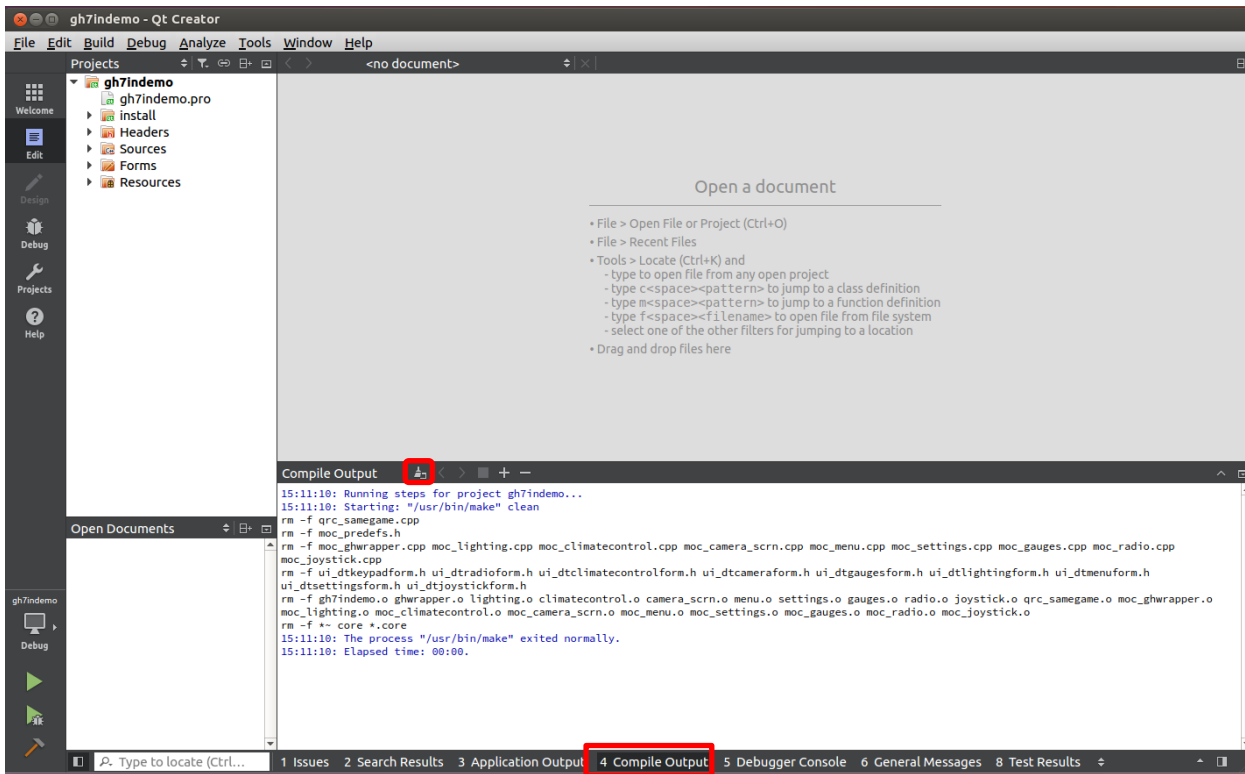
- Build the image for the target



- **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 “Clean Project “gh7indemo””

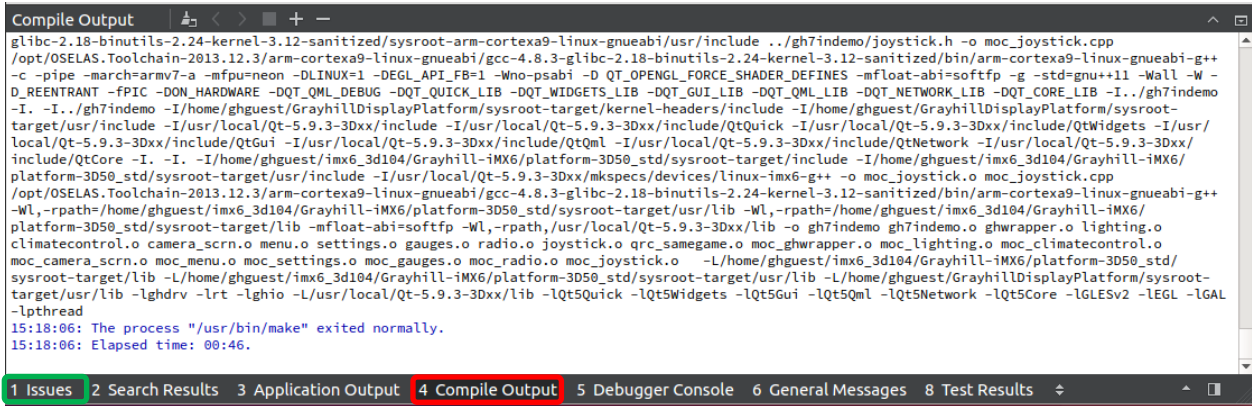
The bottom ribbon of Qt Creator has various panes (views) that can be examined. Click on “4 Compile Output). Note: image is shown post click; so the results and actions of the clean are shown.



Click on the paintbrush icon to clear the contents

- Next, select Build → Build Project “gh7indemo”

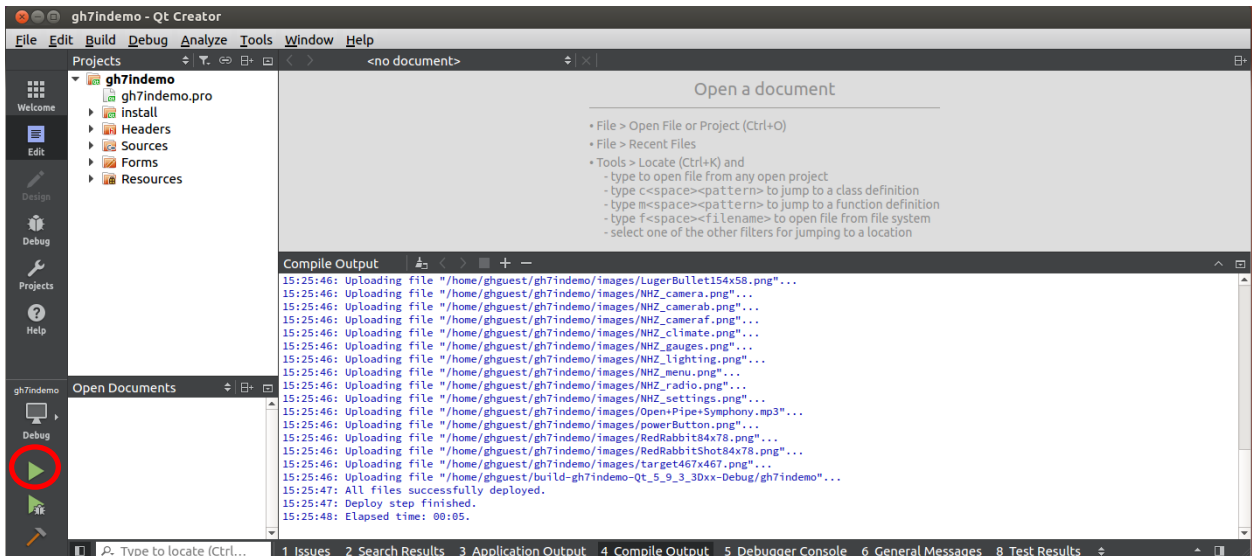
The following illustrates the last few lines in “Compile Output”



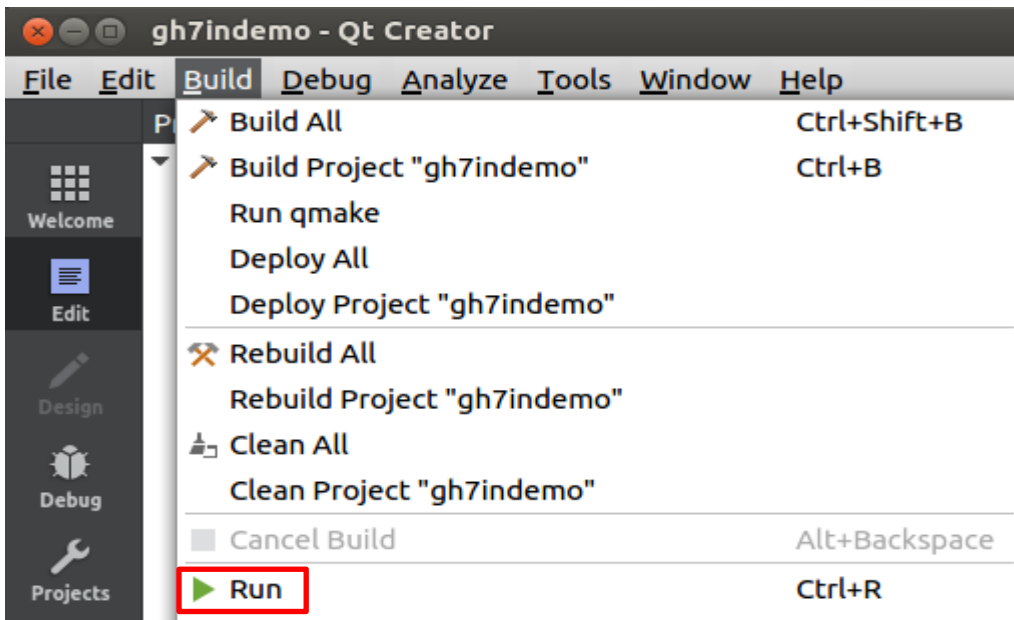
Note: When there are errors, they are also highlighted/summarized in the “Issues” tab.

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

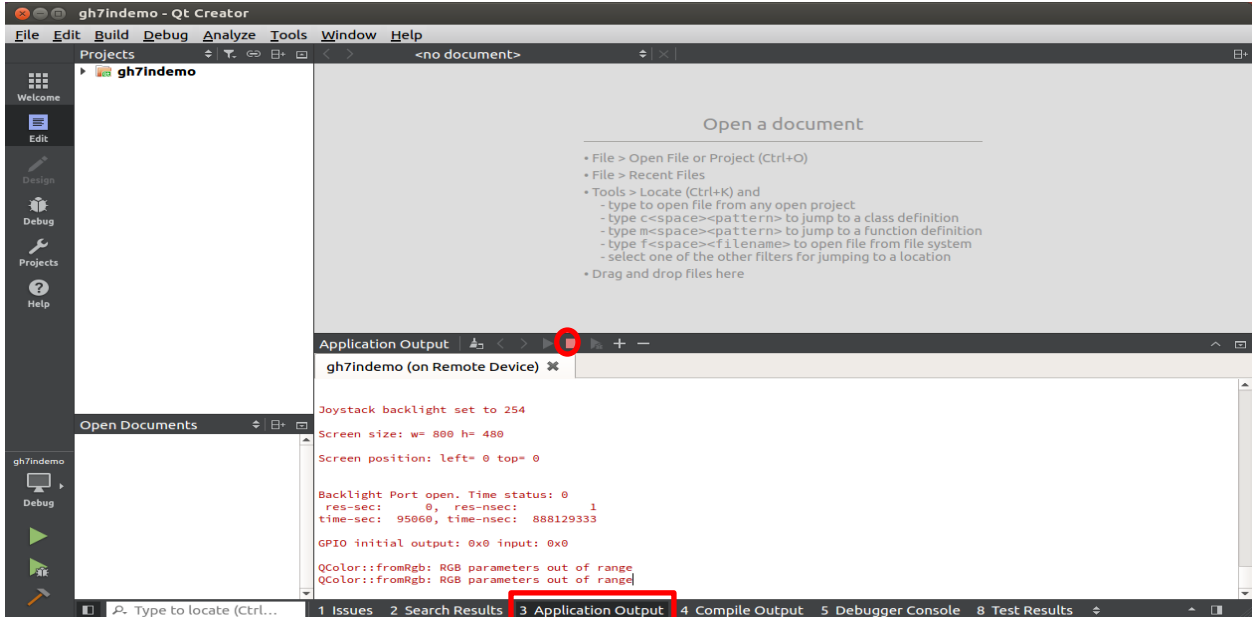
- Using the green triangle on the left hand side



- From the Build menu



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



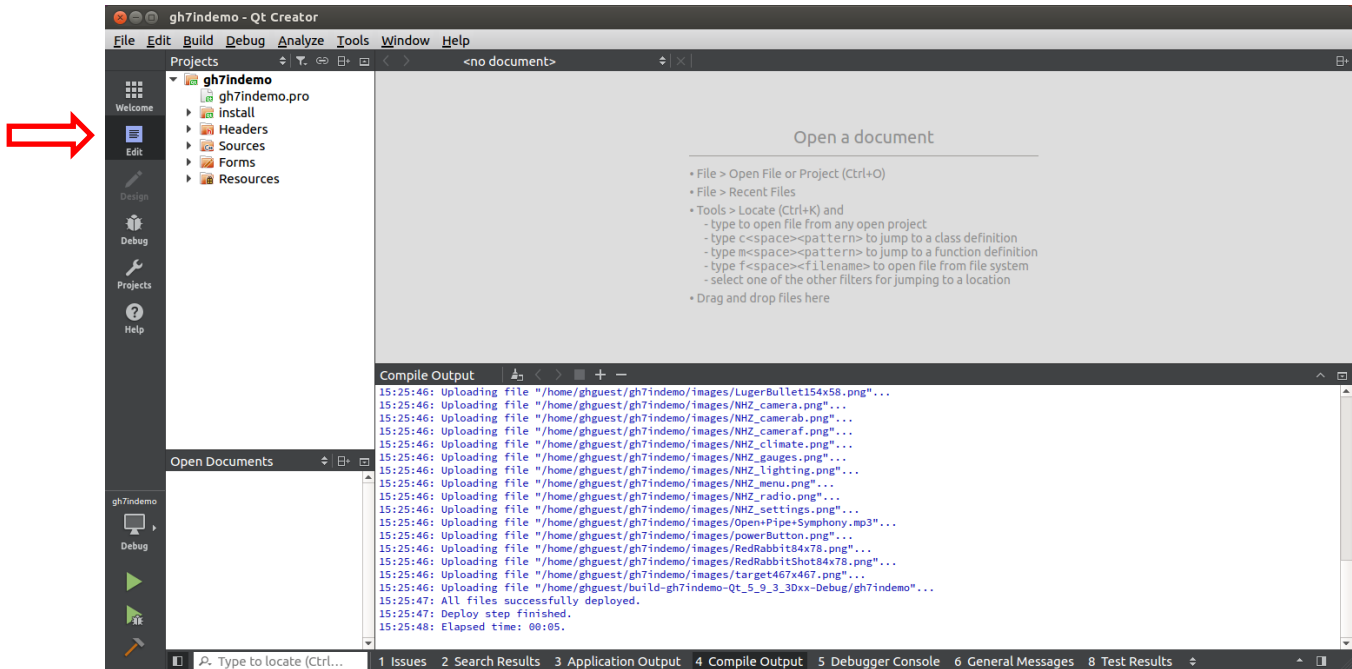
Switch (by selecting) to the “Application Output” tab; this is where qDebug messages are output.

Click the red square to terminate the target session.

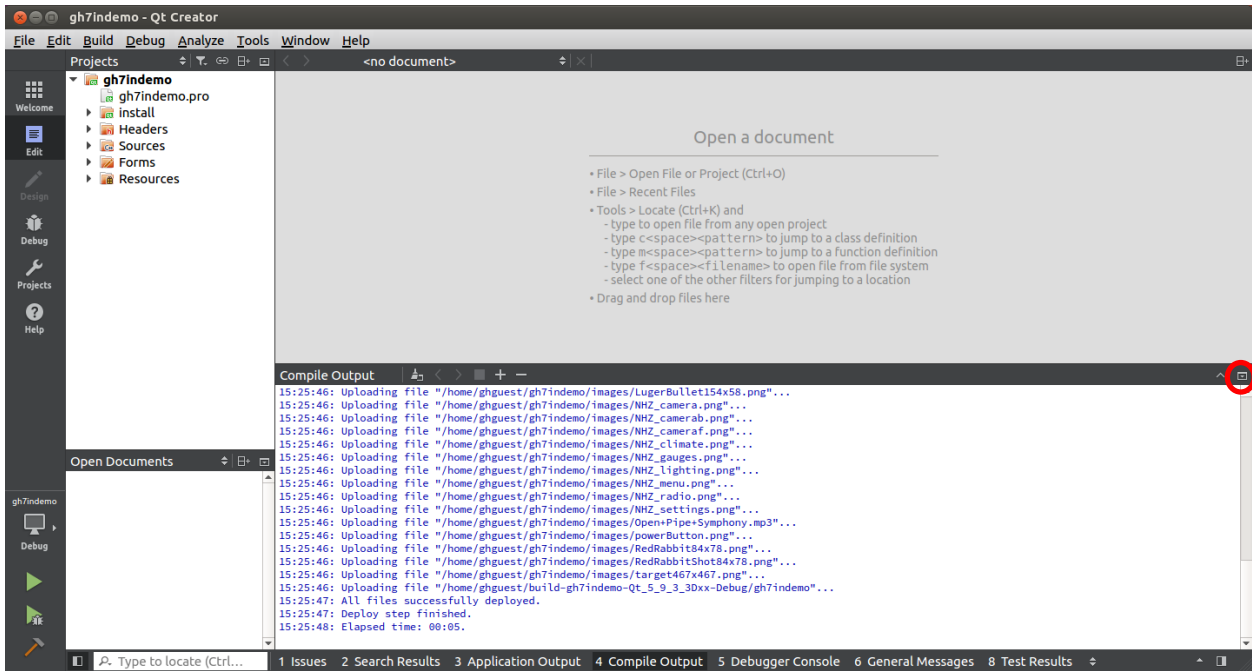
Appendix C: Debugging

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.

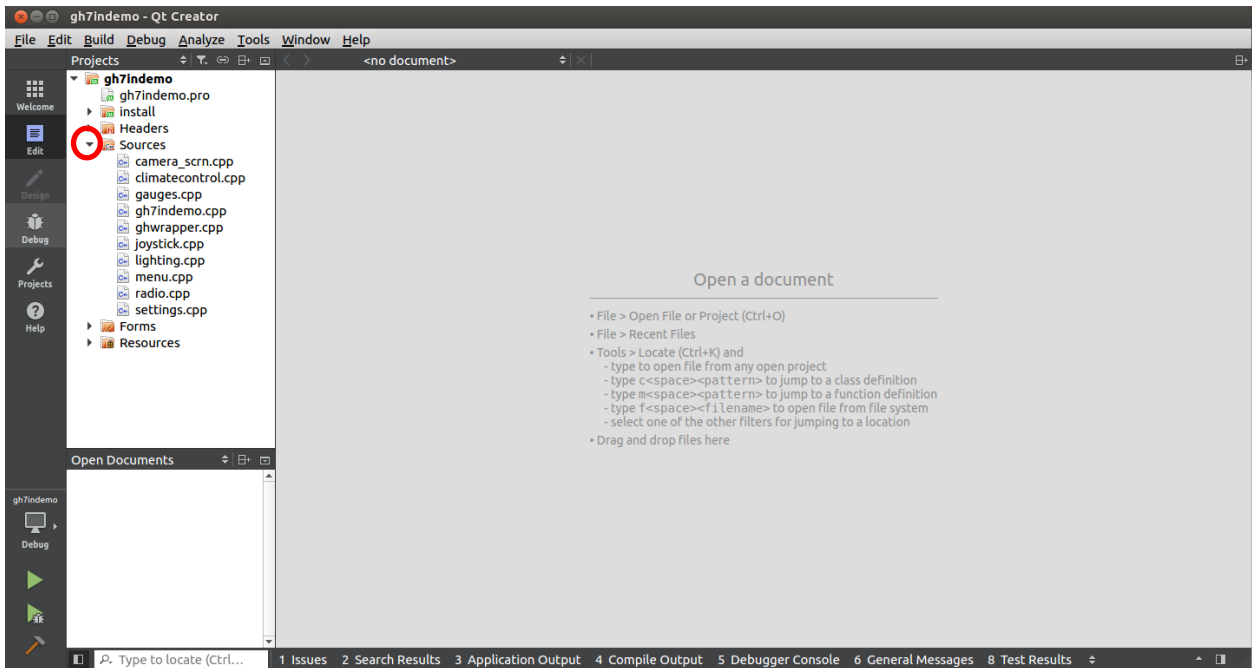
- First set a breakpoint
 - Load gh7indemo
 - Select the “Edit” view
 - Expand contents of gh7indemo



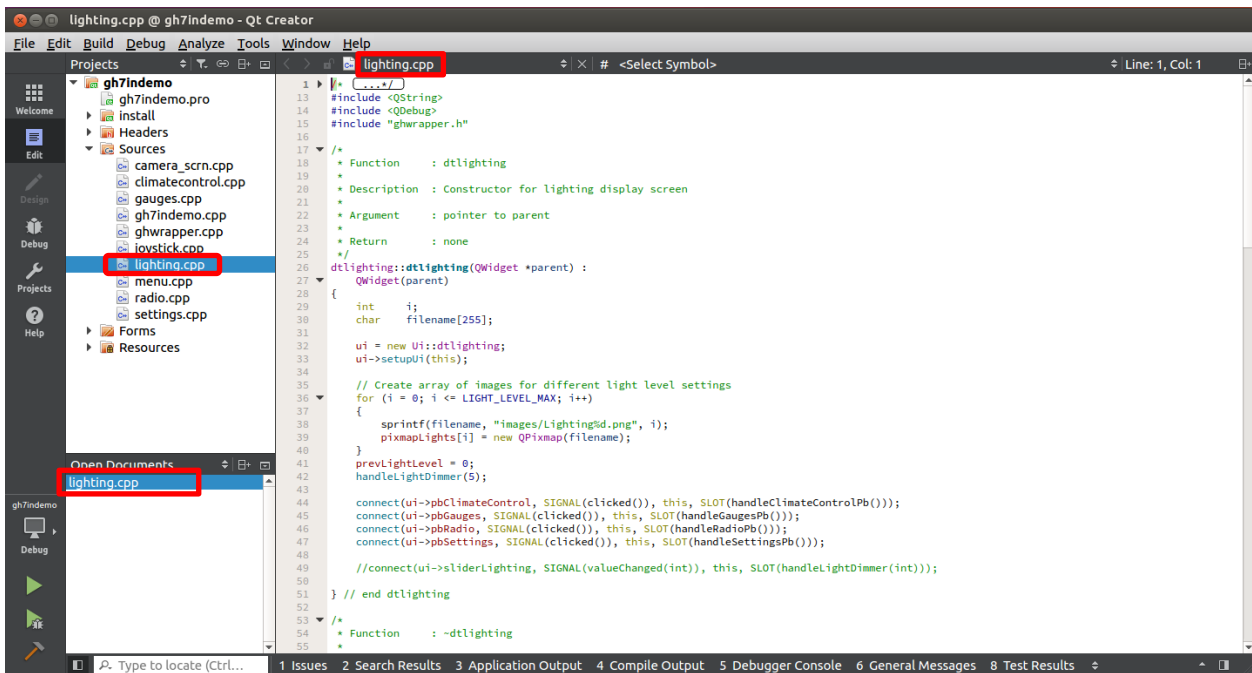
- Close the “Compile Output” pane



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



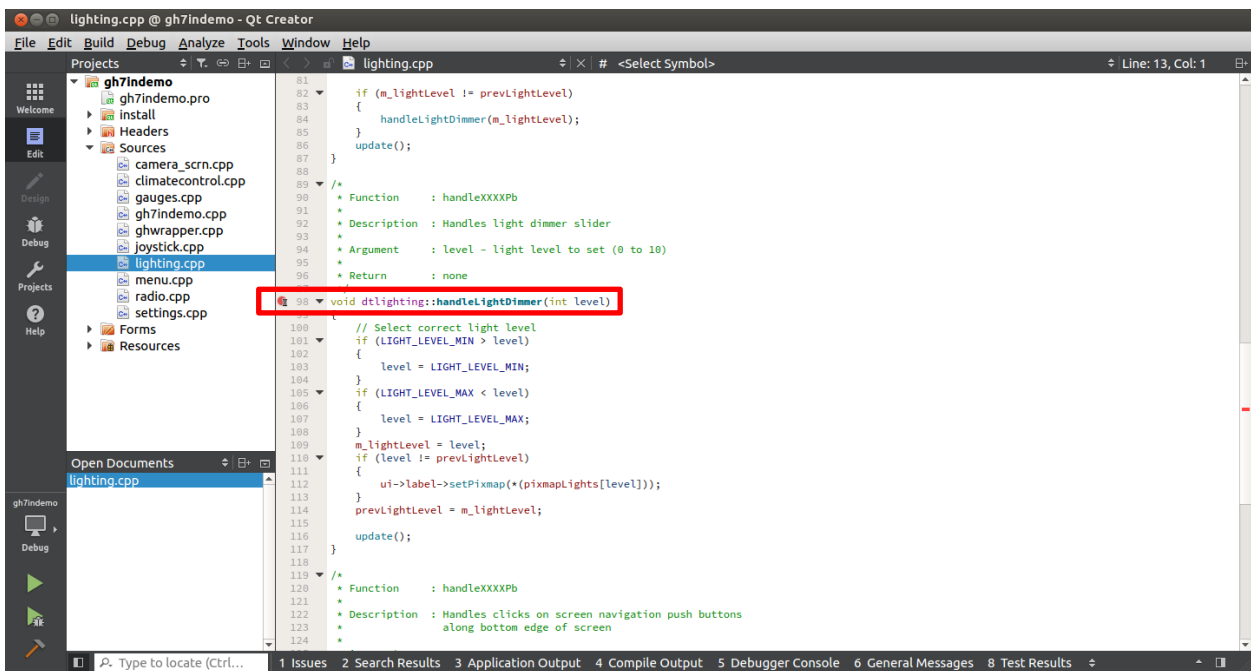
- Select the desired file; under “Sources” select “lighting.cpp” by double clicking



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.

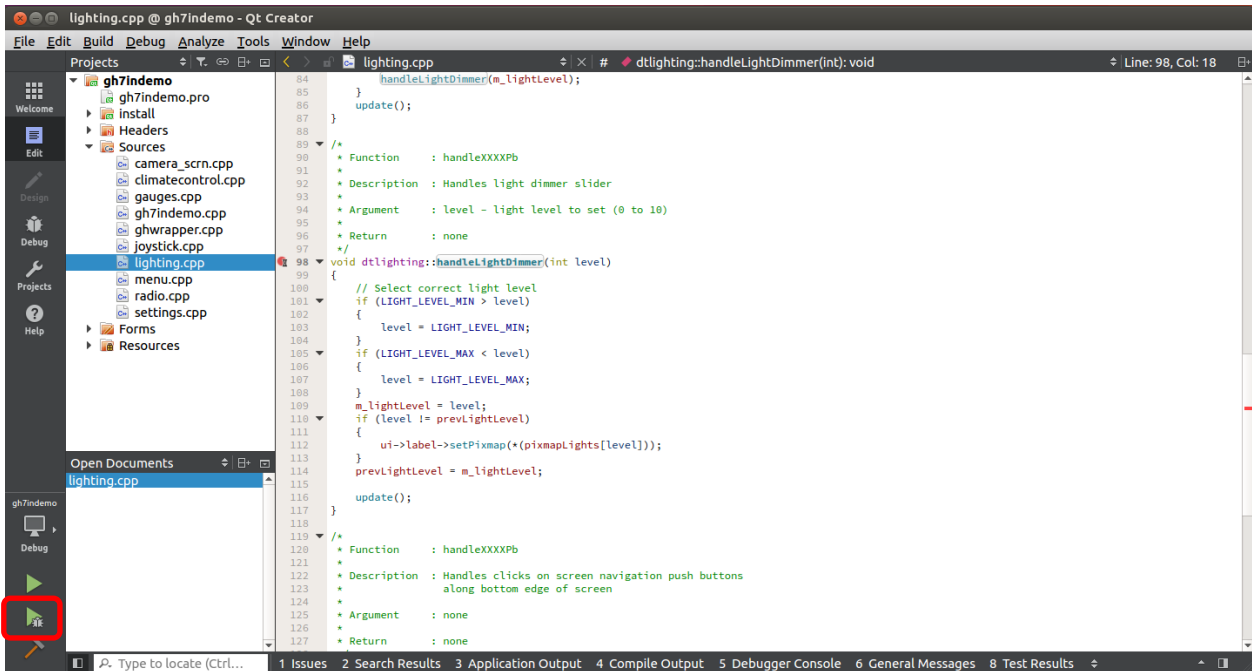
- Scroll down to line 98 (“void dtlighting::handleLightDimmer (int level)”)
- Left click on the mouse to the left of the line number; a red circle will appear



- Also notice in the scroll bar gutter, the indication of where the breakpoint is in relation to the file.
- Save

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

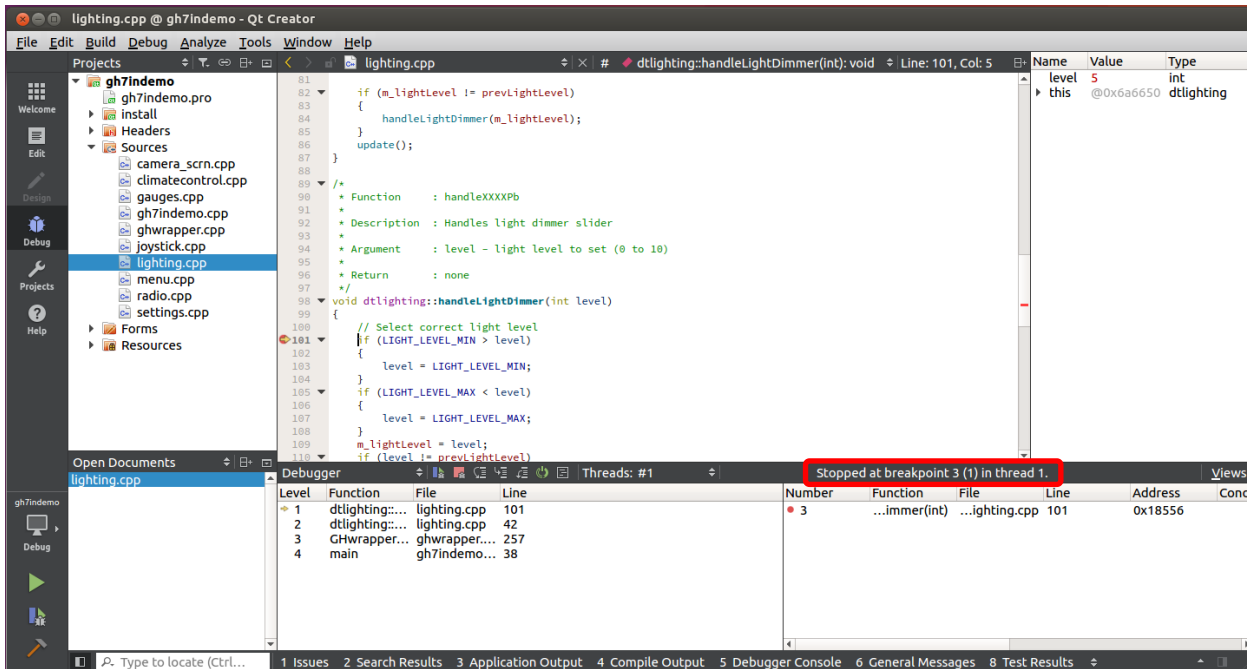
N.B. This may cause the project to be re-compiled if the initial build was not configured for debug.



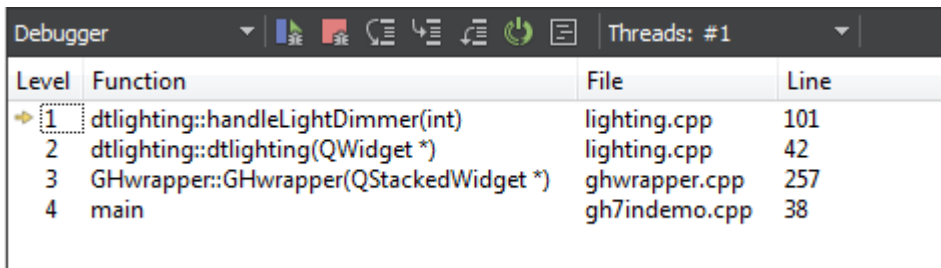
The code begins execution and quickly hits the breakpoint.

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

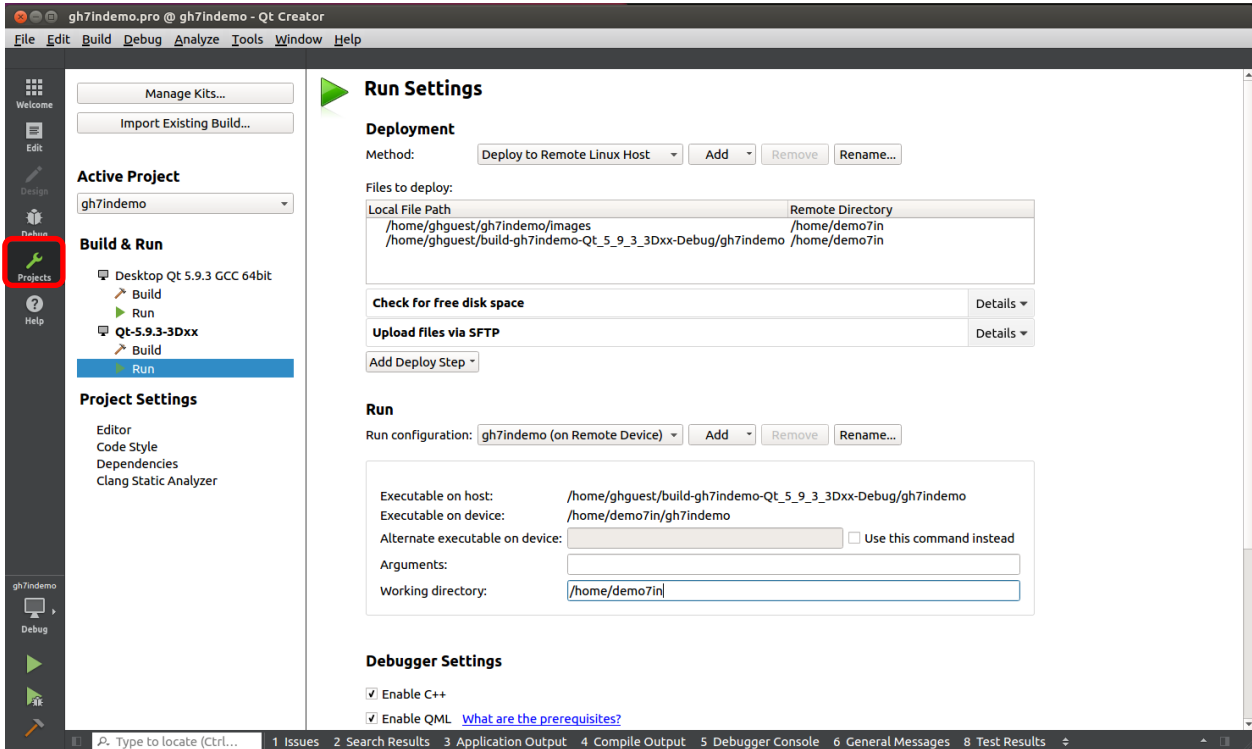


The debugger features the usual (Mouse over the icons for a description)

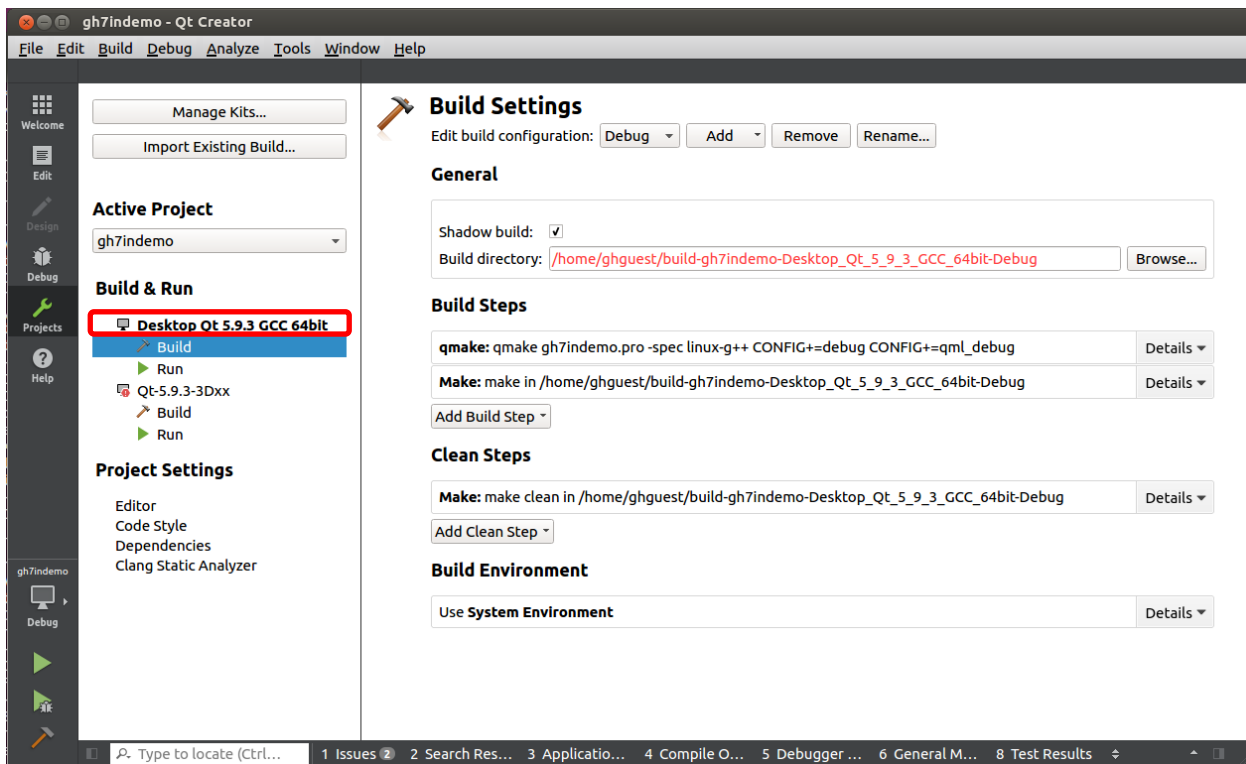
- Step Over <F10>
- Step In <F11>
- Step Out >Shift>+<F11>

Appendix D: Build and Run 3Dxx Desktop Application

Select “Projects”



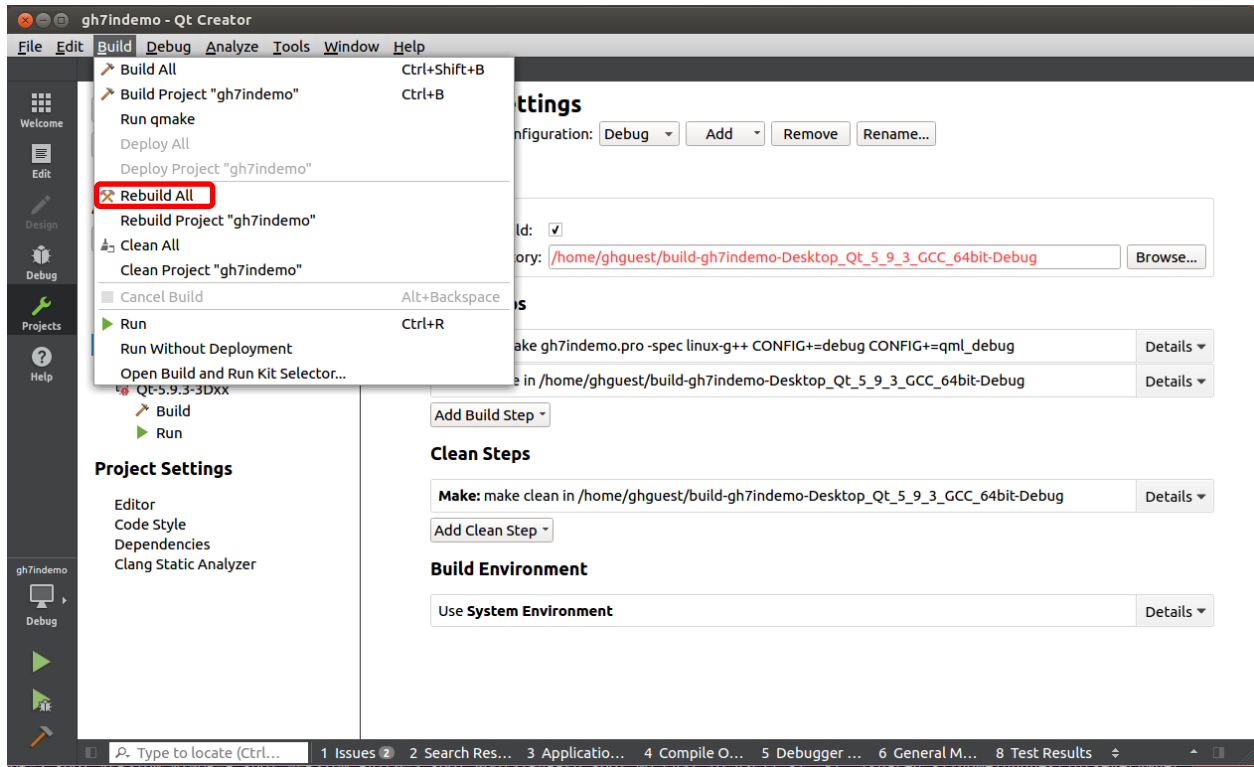
Select Desktop Kit (Desktop Qt 5.9.3 GCC 64bit)



N.B. Selecting “Desktop Qt 5.9.3 GCC 64bit” highlights either “Build” or “Run” depending on which action was previously selected.

Select “Rebuild All” 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
- Create a launch script for the desired application

- cd /etc/init.d
- echo "#! /bin/sh -l
- cd /home/demo7in
- /home/demo7in/gh7indemo &" > launchQtApp
- cat launchQtApp
- chmod 755 launchQtApp

Explanation

set into proper directory
treat as login (runs profile)
set directory for images
spawn application process
verify contents
make script executable

```
COM1 serial port to target
root@ghiiix6:" cd /etc/init.d
root@ghiiix6:/etc/init.d echo "#! /bin/sh -l
>cd /home/demo7in
>/home/demo7in/gh7indemo &" > launchQtApp
root@ghiiix6:/etc/init.d cat launchQtApp
#!/bin/sh -l
cd /home/demo7in
/home/demo7in/gh7indemo &
root@ghiiix6:/etc/init.d chmod 755 launchQtApp
root@ghiiix6:/etc/init.d █
```

- Create a link to the launch script created above
 - cd /etc/rc.d
 - ln -s /etc/init.d/launchQtApp S12qtApp
 - ls -l S12qtApp

set into proper directory
create soft link to executable file
verify link creation

```
COM1 serial port to target
root@ghiiix6:" cd /etc/init.d
root@ghiiix6:/etc/init.d echo "#! /bin/sh -l
>cd /home/demo7in
>/home/demo7in/gh7indemo &" > launchQtApp
root@ghiiix6:/etc/init.d cat launchQtApp
#!/bin/sh -l
cd /home/demo7in
/home/demo7in/gh7indemo &
root@ghiiix6:/etc/init.d chmod 755 launchQtApp
root@ghiiix6:/etc/init.d cd /etc/rc.d
root@ghiiix6:/etc/rc.d ln -s /etc/init.d/launchQtApp S12qtApp
root@ghiiix6:/etc/rc.d ls -l S12qtApp
lrwxrwxrwx 1 root root 23 Jan 18 13:02 S12qtApp -> /etc/init.d/launchQtApp
root@ghiiix6:/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 l+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. *QtLibrarySrcLinux* is a self-extracting archive; when it is executed, the files it contains are extracted. The “-y” flag allows existing files to be replaced. One of the files in the archive, *QtLibrarySrcLinux AutoRun*, is the actual installer script and is manually executed after the extraction. This procedure relies on both Qt Creator and the Grayhill support files having been previously downloaded and installed.

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

- On the Virtual Machine, launch Firefox (Applications → Internet) and navigate to <http://www.grayhill.com/qt43d>
- Download *QtLibrarySrcLinux*
- Copy the file to */home/ghguest*, which is just “Home” if after downloading “Open Containing Folder” is selected from Firefox downloads
- Open a terminal window
 - `chmod 755 QtLibrarySrcLinux`
 - `./QtLibrarySrcLinux -y`

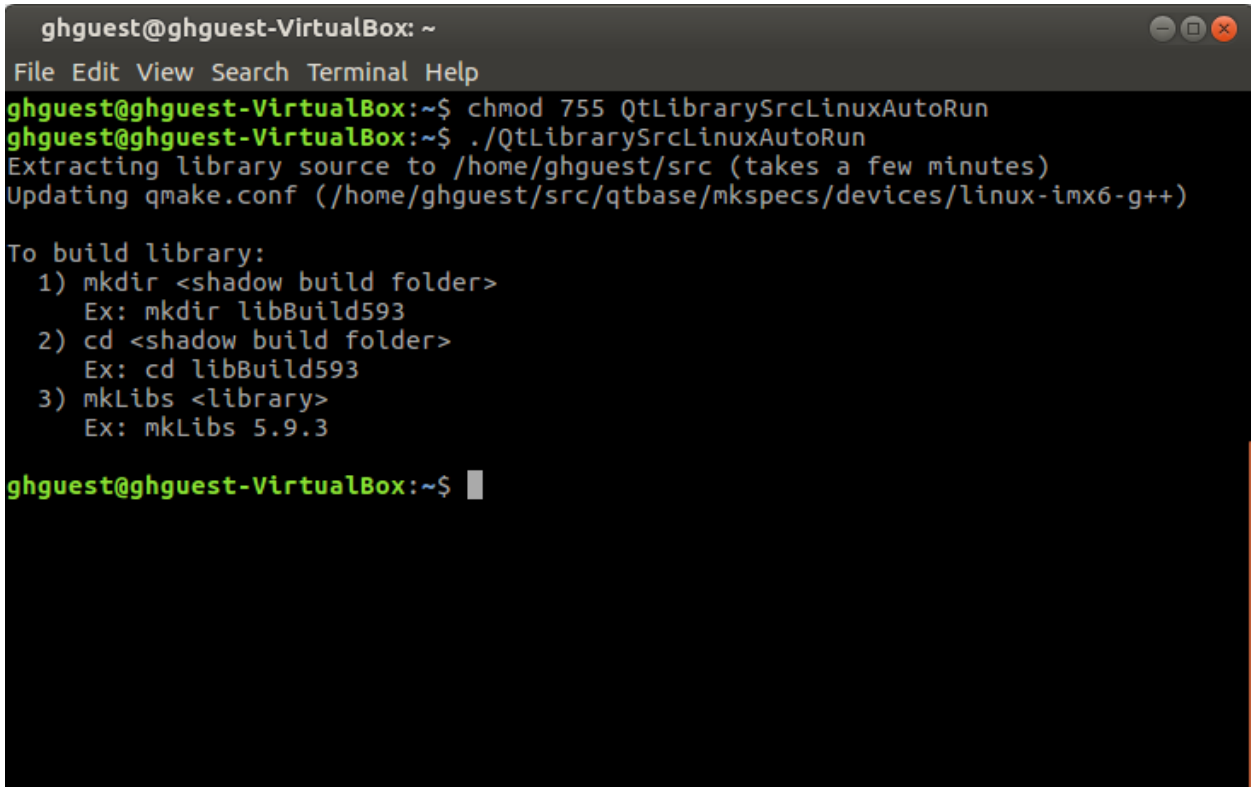
```
ghguest@ghguest-VirtualBox: ~
File Edit View Search Terminal Help
ghguest@ghguest-VirtualBox:~$ ls
bin                               GrayhillExamples  QtGhSupportLinuxAutoRun
build-gh7indemo-Qt_5_9_3_3Dxx-Debug  Music              QtLibrarySrcLinux
build-ghqtdemo-Qt_5_9_3_3Dxx-Debug  Pictures           targetRootFiles
Desktop                           pointercal        Templates
Documents                          Public            Videos
Downloads                          Qt5.9.3
GrayhillDisplayPlatform           QtGhSupportLinux
ghguest@ghguest-VirtualBox:~$ chmod 755 QtLibrarySrcLinux
ghguest@ghguest-VirtualBox:~$ ./QtLibrarySrcLinux -y

7-Zip SFX 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_US.UTF-8,Utf16=on,HugeFiles=on,64 bits,1 CPU x64)

Extracting archive: ./QtLibrarySrcLinux
--
Path = ./QtLibrarySrcLinux
Type = 7z

Everything is Ok
ghguest@ghguest-VirtualBox:~$
```


- `chmod 755 QtLibrarySrcLinuxAutoRun`
- `./QtLibrarySrcLinuxAutoRun`



```
ghguest@ghguest-VirtualBox: ~
File Edit View Search Terminal Help
ghguest@ghguest-VirtualBox:~$ chmod 755 QtLibrarySrcLinuxAutoRun
ghguest@ghguest-VirtualBox:~$ ./QtLibrarySrcLinuxAutoRun
Extracting library source to /home/ghguest/src (takes a few minutes)
Updating qmake.conf (/home/ghguest/src/qtbase/mkspecs/devices/linux-imx6-g++)

To build library:
 1) mkdir <shadow build folder>
    Ex: mkdir libBuild593
 2) cd <shadow build folder>
    Ex: cd libBuild593
 3) mkLibs <library>
    Ex: mkLibs 5.9.3

ghguest@ghguest-VirtualBox:~$ █
```

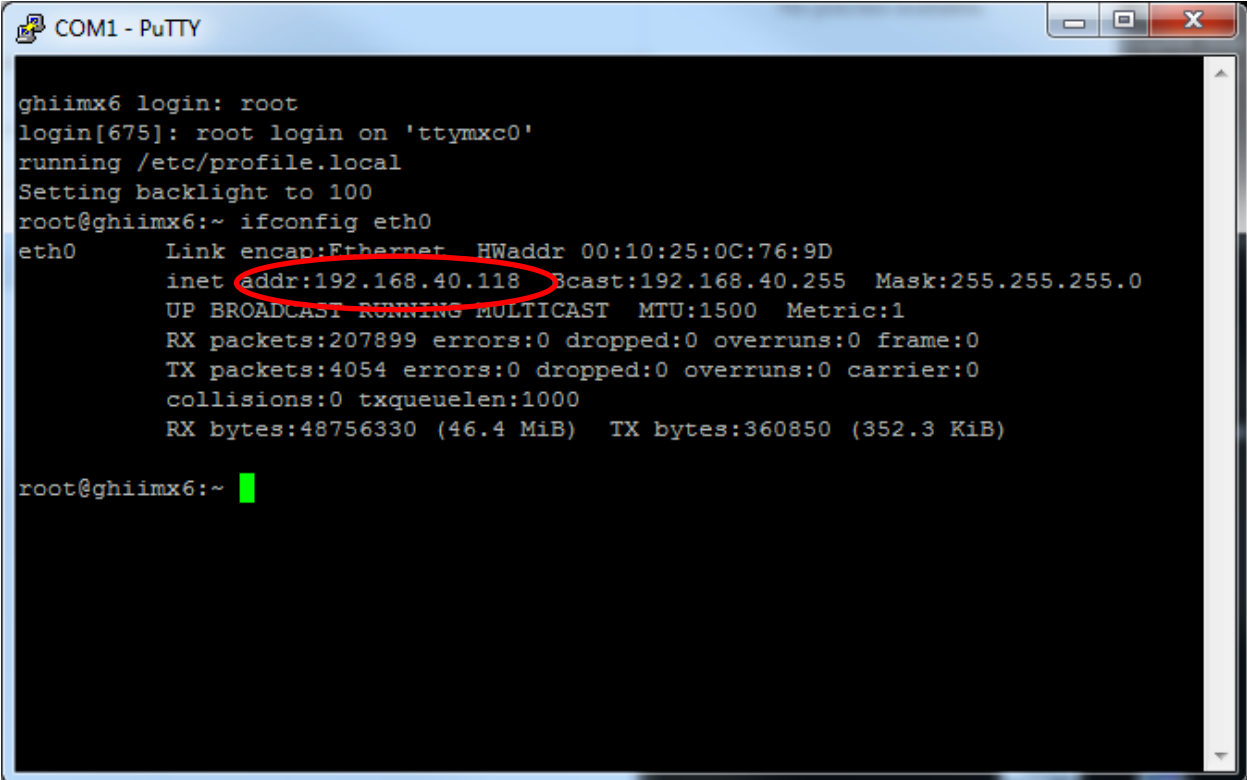
- Verify `SYSROOT_PATH` is set properly before building
 - `echo $SYSROOT_PATH`
 - `export SYSROOT_PATH=${HOME}/GrayhillDisplayPlatform/sysroot-target`

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:~ █
```

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