DEEP SEA ELECTRONICS

DSEM812 Qt Manual

Document Number: 057-319

Author: Anthony Manton

Deep Sea Electronics Ltd
Highfield House
Hunmanby
North Yorkshire
YO14 0PH
ENGLAND

Sales Tel: +44 (0) 1723 890099

E-mail: sales@deepseaelectronics.com
Website: www.deepseaelectronics.com

DSEM812 Qt Manual

Revision History

Issue No.	Comments
1	First Release
1.1	Updated to cover the new Package File that contains both DSEM812 and DSEM870.
1.2	Added information about permissions of the Shared Folder and use of PuTTY.
1.3	Updated Qt version to 5.15
1.4	Minor corrections to table of contents. Added ssh private key note. Added note about sftp/rsync.

TABLE OF CONTENTS

Section **Page**

1 INTRODUCTION

This document details the operation and setup requirements of the DSEM812 Qt Controller and Display, part of the DSEControl® range of products.

DSEM812 CODESYS variants are not covered in this document and are detailed within *DSE Publication 057-318 DSEM812 CODESYS Manual.*

Hardware, Specifications, Settings Pages and Installation Notes for all DSEM812 variants are detailed within *DSE Publication 057-317 DSEM812 Operator Manual.*

Knowledge of Linux and Qt (QML, C++ and JavaScript) is essential and assumed. This manual instructs the Qt programmer how to use Qt in conjunction with the DSE device. This manual does not give instruction for Linux, Qt, QML, C++ or JavaScript.

The manual forms part of the product and should be kept for the entire life of the product. If the product is passed or supplied to another party, ensure that this document is passed to them for reference purposes.
This is not a *controlled document.* DSE do not automatically inform on updates. Any future updates of this document are included on the DSE website at www.deepseaelectronics.com

Observe the operating instructions. Non-observance of the instructions, operation not in accordance with use as prescribed below, wrong installation or incorrect handling seriously affects the safety of the product, operators and machinery.

A robust metal case designed for chassis mounting houses the module. Connections are via locking plug and sockets.

The controller is supplied with no application program. The equipment manufacturer is responsible for creating and managing the application program and installing it in the controller. This is achieved using Qt programming. Contact DSE Technical Support for further details.

Qt is an application development framework for Linux Embedded systems.
Qt is not a programming language on its own. It is a framework written in C++ using signals and slots to pass information to/from Qt Modelling Language files (QML).

1.1 CLARIFICATION OF NOTATION

Clarification of notation used within this publication.

NOTE:	**Highlights an essential element of a procedure to ensure correctness.**
CAUTION!	**Indicates a procedure or practice, which, if not strictly observed, could result in damage or destruction of equipment.**
WARNING!	**Indicates a procedure or practice, which could result in injury to personnel or loss of life if not followed correctly.**

1.2 GLOSSARY OF TERMS

Term	Description
Application	The application is the program that allows the DSEM812 to control the machine it is connected to. The Application within the DSEM812 is designed and provided by the manufacturer of the complete machine.
Bootloader	The Bootloader is the program within the DSEM812 responsible for loading the Operating System.
C++	Programming language used alongside QML to create the complete application program.
CAN	Control Area Network. A high-speed data transmission system used extensively within the Automotive and Off-Highway industries.
Deploy	The compiled application is 'deployed' to the device folder /home/m812
ECU	Electronic Control Unit. For example, the DSEM812 device.
Firmware	The Firmware of the DSEM812 is the Operating System of the DSEM812 that reads and executes the Application program.
FSD	Full Scale Deflection. For example, 0 mA to 20 mA is the Full Scale Deflection of a current sink input.
I/O	Input / Output. For example, "The I/O is taken out to an external terminal strip in the user panel".
IDE	Integrated Development Environment. For example, the Qt application that runs on the host PC is an IDE, containing code editors, compilers and much more.
Ixyyy	An Input, where x is the connector and yyy is the input number. For example, IB003 means Input 3 on Connector B.
Nano	An editor to allow text files to be altered.
PLC	Programmable Logic Controller. Industrial computer used primarily for the automation of electromechanical machinery.
PWM PWMi	A digital signal is used to represent an analogue value by using Pulse Width Modulation. The mark-space ratio of a square wave changes to represent the value. Used for many control applications including proportional valves. PWM= Voltage control. PWMi = Current control.
Off-Highway	An industrial vehicle used primarily "off road". For example construction and farm machinery. A wider interpretation includes on road access platforms, emergency vehicles and other industrial machinery, used either on the road, or off road.
Pin	A male or female pin connection in a housing (plug or socket).

QML	Qt Modelling Language. Used to design/describe the display elements on the DSEM812 Qt variant.
Qt	Application development system supported by DSEM812 Qt Variant Pronounced 'Cute'.
Qxyyy	An Output, where x is the connector and yyy is the output number. For example QB002 means Output 2 on Connector B.
Remote Shell	A *terminal* to allow commands to be run on the target device.
Run.sh	Shell program executed at device boot time. Among other functions, this file instructs the device which application to execute at boot of the device.

1.3 RELATED INFORMATION

This document refers to and is referred by the following DSE publications which are obtained from the DSE website: www.deepseaelectronics.com or by contacting DSE technical support: support@deepseaelectronics.com.

1.3.1 TECHNICAL INFORMATION

DSE Part	Description
055-267	DSEM812 Datasheet
057-317	DSEM812 Operator Manual

1.4 SAFETY INSTRUCTIONS

1.4.1 GENERAL

- These instructions are for authorised persons according to the EMC and low-voltage directives. The device must be installed, connected and put into operation by a qualified electrician.

- It is not permissible to open the controller or to modify or repair the controller. Modification or repairs to the wiring could result in dangerous malfunctions. Repairs to the controller must be performed by DSE. Contact your original equipment supplier in the case of malfunction.

- When the device is unpowered, ensure that no connection pins are connected to a voltage source. Thus, when the supply is switched off, the supply for the electronics, the power outputs and the external sensor supply must be switched off together.

- The controller heatsink at the rear heats up beyond normal ambient temperature during operation. To avoid danger caused by high temperatures, protect against contact.

- The customer is responsible for performing risk analysis of the mobile working machine and determining the possible safety related functions. The user is responsible for the safe function of the application programs created. If necessary, they must additionally carry out an approval test by corresponding supervisory and test organisations according to the national regulations.

- All connectors must be unplugged from the electronics during electrical welding and painting operations.

1.4.2 INSTALLATION NOTES

- Follow the instructions of the connector manufacturer, specifically with respect to preventing water from entering the device. See Section entitled *Cables, Connectors, Harnesses and Spare Parts* for details of DSE Part Numbers.

- To maintain IP67 rating where connectors have unused pins, ensure the use of a suitable Blanking Insert. In the case of a completely unused connector, the plug must be inserted, fully populated with Pin Blanking Inserts. See Section entitled *Cables, Connectors, Harnesses and Spare Parts* for details.

- M12 protection plugs (supplied) must be installed in both the USB and Ethernet interfaces to ensure IP67 rating when the connectors are not in use. Tighten to 0.8 Nm (0.6 lbf ft). Where IP protection is required when the interfaces are in use, suitable O-rings must be fitted.

- The heatsink must be wired to vehicle ground to comply with EMC guidelines. A screw connection point is provided for this purpose. A metallic screw must be used to create an electrical connection to vehicle / machine ground.

2 QT, C++ AND QML

NOTE: Knowledge of Linux and Qt (QML, C++ and JavaScript) is essential and assumed. This manual instructs the Qt programmer how to use Qt in conjunction with the DSE device. This manual does not give instruction for Linux, Qt, QML, C++ or JavaScript.

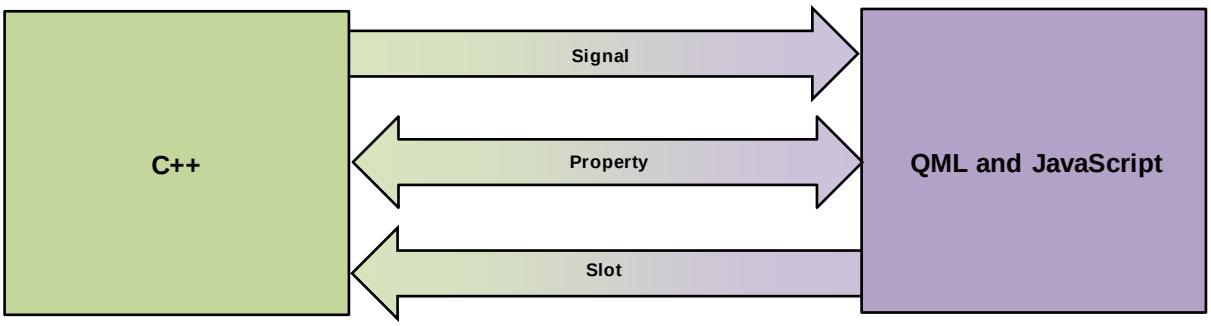
Qt (pronounced 'Cute') is an application development system used by the Linux based DSEM812 devices. C++, QML and JavaScript are used to program the device.

QML is *Qt Modelling Language,* a mark-up language designed to aid development of user interfaces. QML elements support JavaScript both inline and via included .js files.

QML is used to describe only the user interface. Where interaction with device hardware is required, Integration with C++ is used.

Qt allows the use of externally programmed libraries to 'extend' the functionality of the system. This includes the use of DSE supplied libraries to allow for example, the use of the device I/O.

C++ and QML work together to provide the full functionality of Qt.



2.1 PROPERTIES, SIGNALS AND SLOTS

Properties, Signals and Slots are the mechanism that binds together C++ and QML. C++ classes derived from *QObject* or a subclass are registered to allow them to be created as objects within QML.

- Properties are values associated with the object. QML reads and/or writes from/to the properties. For example, a property of a class used to monitor the inputs of the device may be read by the QML to know the voltage applied to the input.
- Signals are emitted by the C++ class upon specific actions occurring within the class. Typically, a signal is emitted when a property value is changed in the C++ class. For example, if a device input monitored by the C++ class changes value, a signal is sent to inform the QML that the input has changed. *Signal Handlers* are used in the QML to act upon signals from the C++.
- A slot is a method of the class called when the property is changed by the QML. For example, if the QML changes a property linked to the state of a device output, the C++ class detects the change and calls the slot method that handles the *setting* of the class variable. In turn this may also trigger a signal as the linked property has also changed.

Examples of properties, signals and slots are given in the following subsections.

2.1.1 SIGNALS

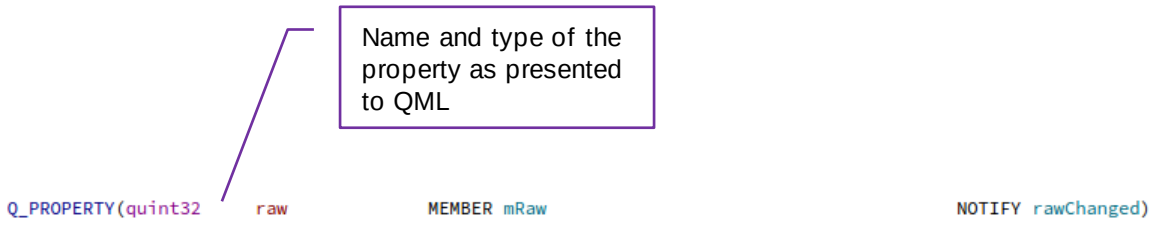
QObject Signals are emitted from C++ and typically used to trigger callback functions in the QML JavaScript.

Signals are emitted by an object when it is required to inform about a change in the object. The *Emit* keyword is used to do this.

Good practice dictates that a signal is used to indicate any properties that have changed.

2.1.1.1 PROPERTY DEFINITION

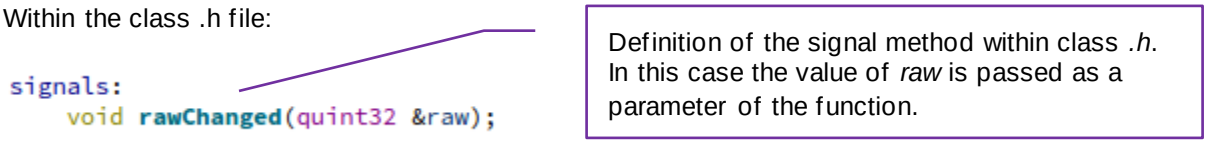
Within the class *.h* file a *Q_PROPERTY* is defined. In this example the property is *read-only* by the QML due to the omission of a WRITE slot definition. For details of this, see section entitled *Slot Definition* elsewhere in this document.



Parameter	Description
MEMBER	Defines the class variable that is used to link to the QML object property. Such class variables are prefixed 'm' to indicate *MEMBER.*
NOTIFY	Defines the signal emitted upon change of the property. The signal must be prototyped in the *.h* file but does not required an associated method creating in the *.cpp* file. The signal is automatically emitted upon change of the property.

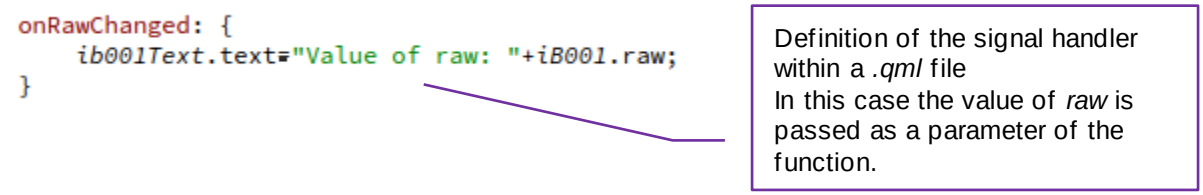
2.1.1.2 SIGNAL DEFINITION

Within the class .h file:



2.1.1.3 SIGNAL USAGE

To receive a notification when a signal is emitted for an object, the object definition includes a signal handler named on <Signal>, where <Signal> is the name of the signal, with the first letter capitalised. The signal handler contains JavaScript code executed when the signal handler is triggered.



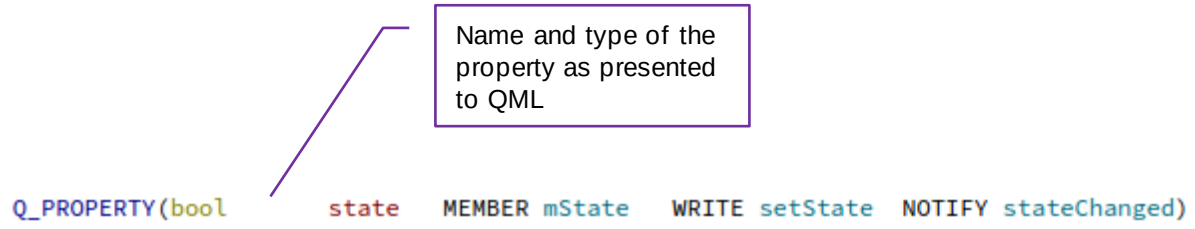
2.1.2 SLOTS

QObject Slots are C++ methods that allow them to be called upon the change of a property.

The class *.h* file defines the C++ class a derived from *QObject* and defines methods as *public slots* to make them accessible to QML.

2.1.2.1 PROPERTY DEFINITION

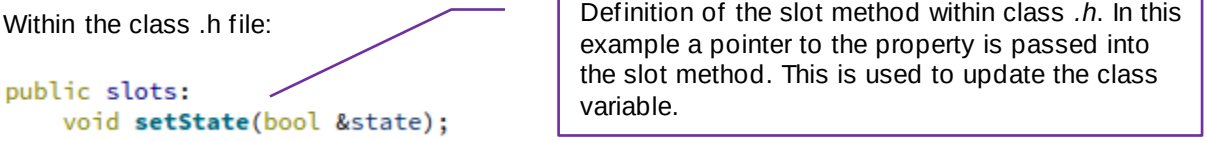
Within the class .h file a *Q_PROPERTY* is defined. In this example the property is read/Write.



Parameter	Description
MEMBER	Defines the class variable that is used to link to the QML object property. Such class variables are prefixed 'm' to indicate *member.*
WRITE	Defines the name of a method called when QML changes the value of the property. This is used to control access to the class variable. Typical application of the slot method is to check if the property value has changed, validate the new property value, update the class member variable with the new property value and finally emit a signal to inform QML of the change.
NOTIFY	Defines the signal emitted upon change of the property. The signal must be prototyped in the *.h* file but does not required an associated method creating in the *.cpp* file. The signal is automatically emitted upon change of the property.

2.1.2.2 SLOT DEFINITION

Within the class .h file:



2.1.2.3 SLOT METHOD

In this example, a device output is controlled, based upon the value of the property pointed to in the method call.

- The output is set to the state requested by the QML property *state*.
- If the property value of *state* has changed (if it is not the same as the class variable *mState*) then the class variable is updated and the signal *stateChanged* is emitted, passing the updated value along with the signal.

```
void Outputs::setState(bool &state)
{
    dse_io_output_set_state(static_cast<dse_io_output_pin_t>(mOutput), state);

    if (state != mState) {
        mState = state;
        emit stateChanged(mState);
    }
}
```

3 INSTALLING THE QT ENVIRONMENT

NOTE: The VM and the Package contain the Virtual Machine and setup for both DSEM812 Qt and DSEM870 Qt.

NOTE: *DSEvm.zip* containing the Virtual Disk Image is available from support@deepseaelectronics.com.

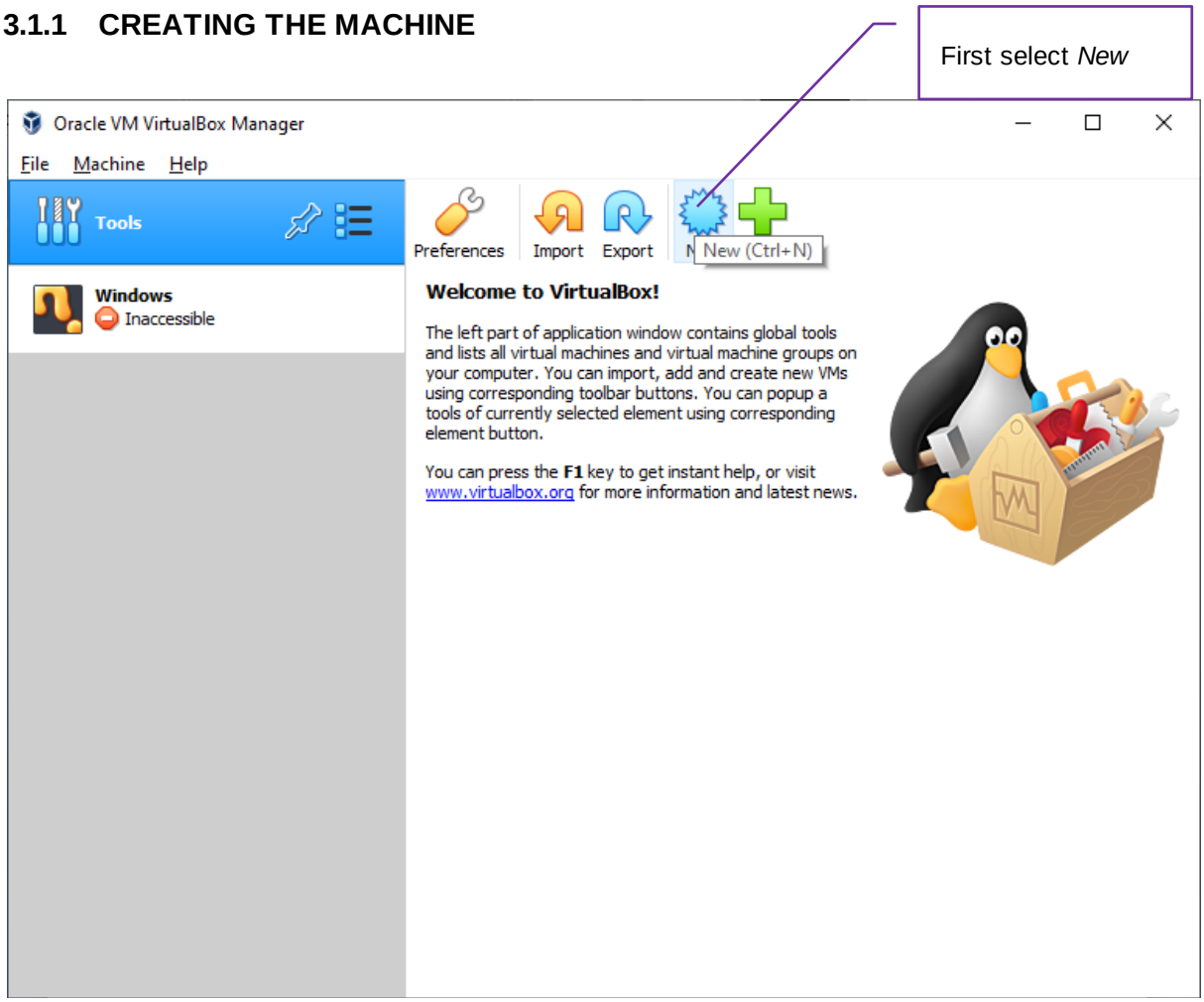
Extract the VDI (Virtual Disk Image) from the containing zip file. The extracted image is over 6 GB. The VDI is designed to be used by Oracle VM VirtualBox PC software and provides a Ubuntu operating system with Qt 5.11 preinstalled and configured for use with DSEM812.

VirtualBox is available from https://www.virtualbox.org/wiki/Downloads

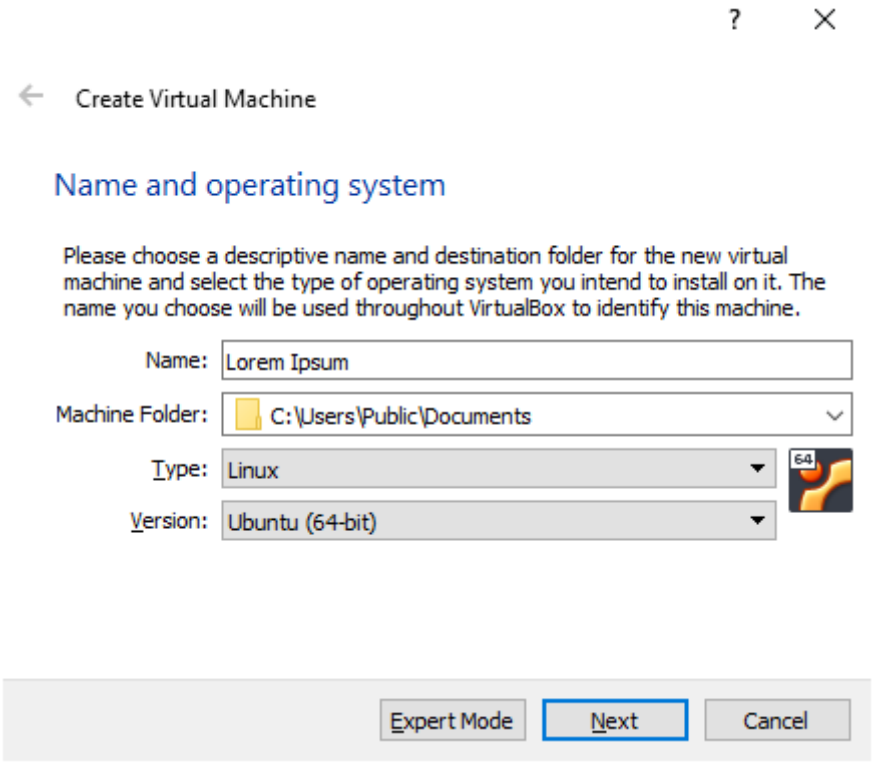
3.1 VM INSTALL

Install VirtualBox. A new *machine* needs to be created with the supplied virtual disk image.

3.1.1 CREATING THE MACHINE

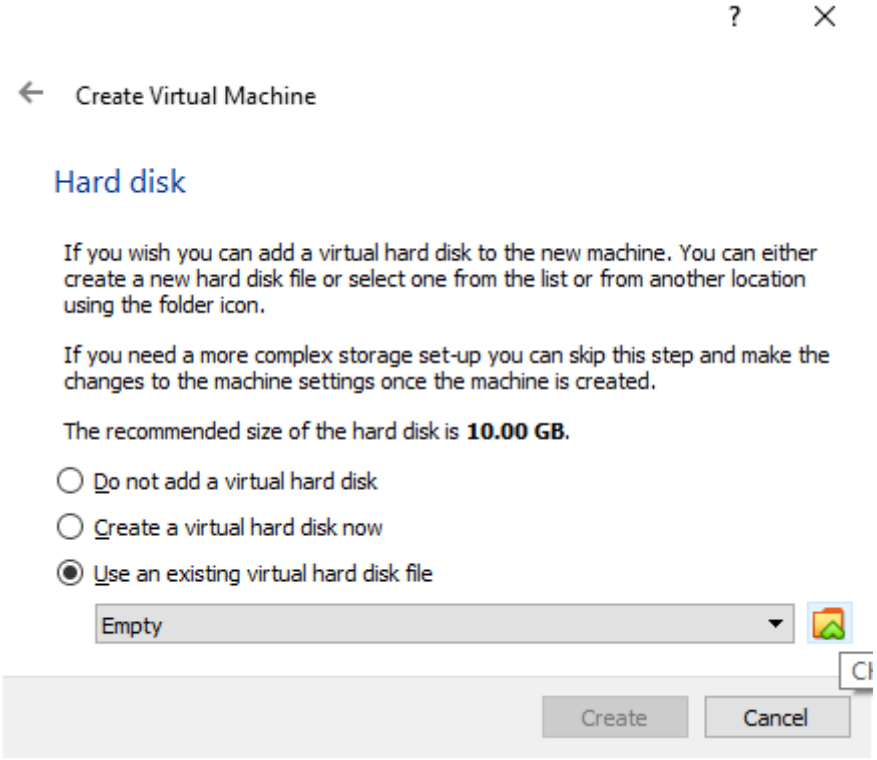


Then enter a name, select a folder and select Linux, Ubuntu (64-bit):

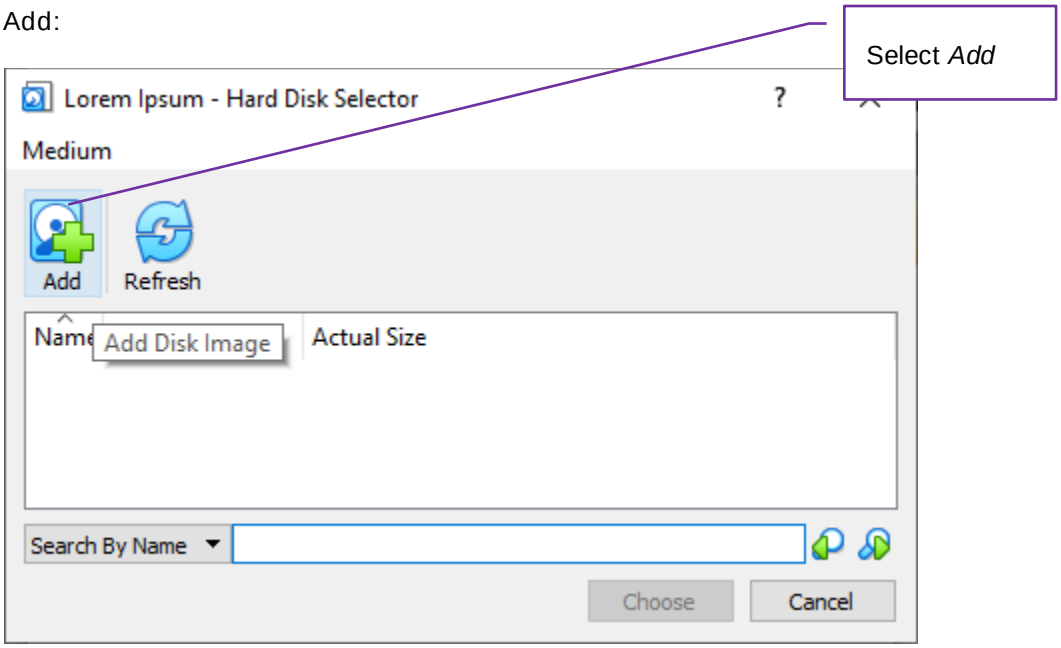


Then select Next, use a **minimum** of 1GB of memory.

Select Next and choose “Use an existing virtual disk file” and select the supplied virtual disk image:



Then select Add:

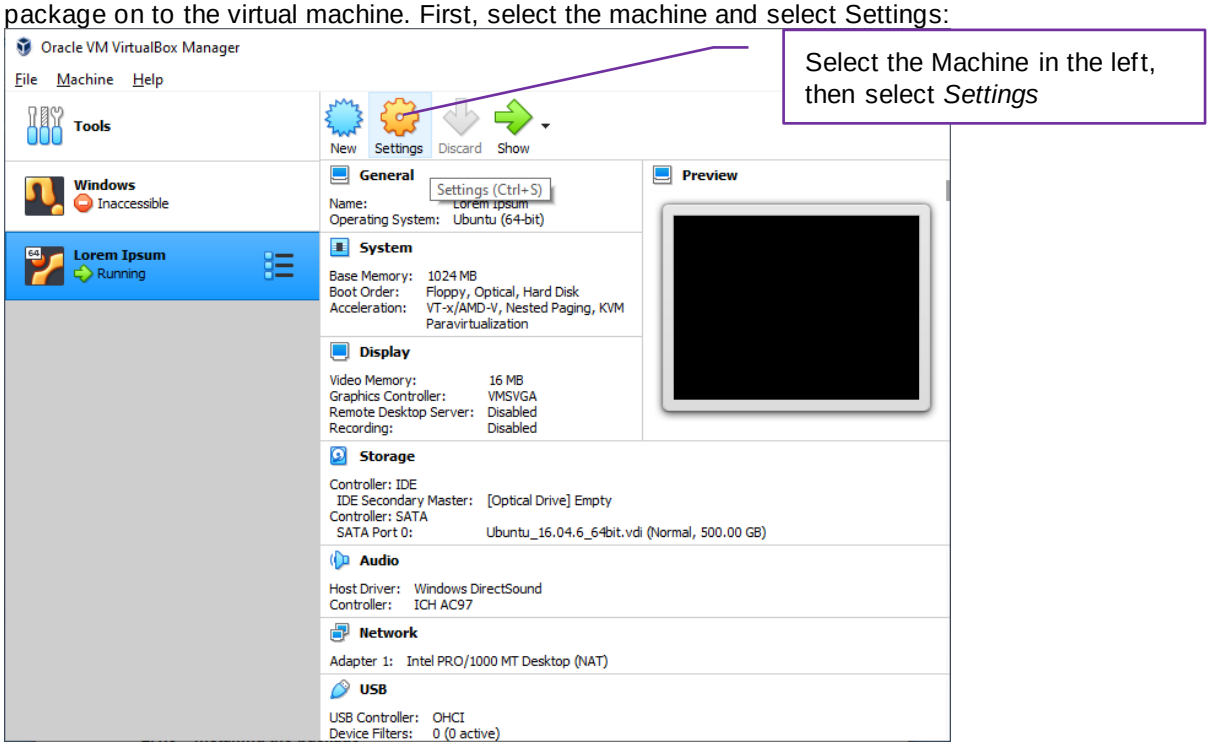


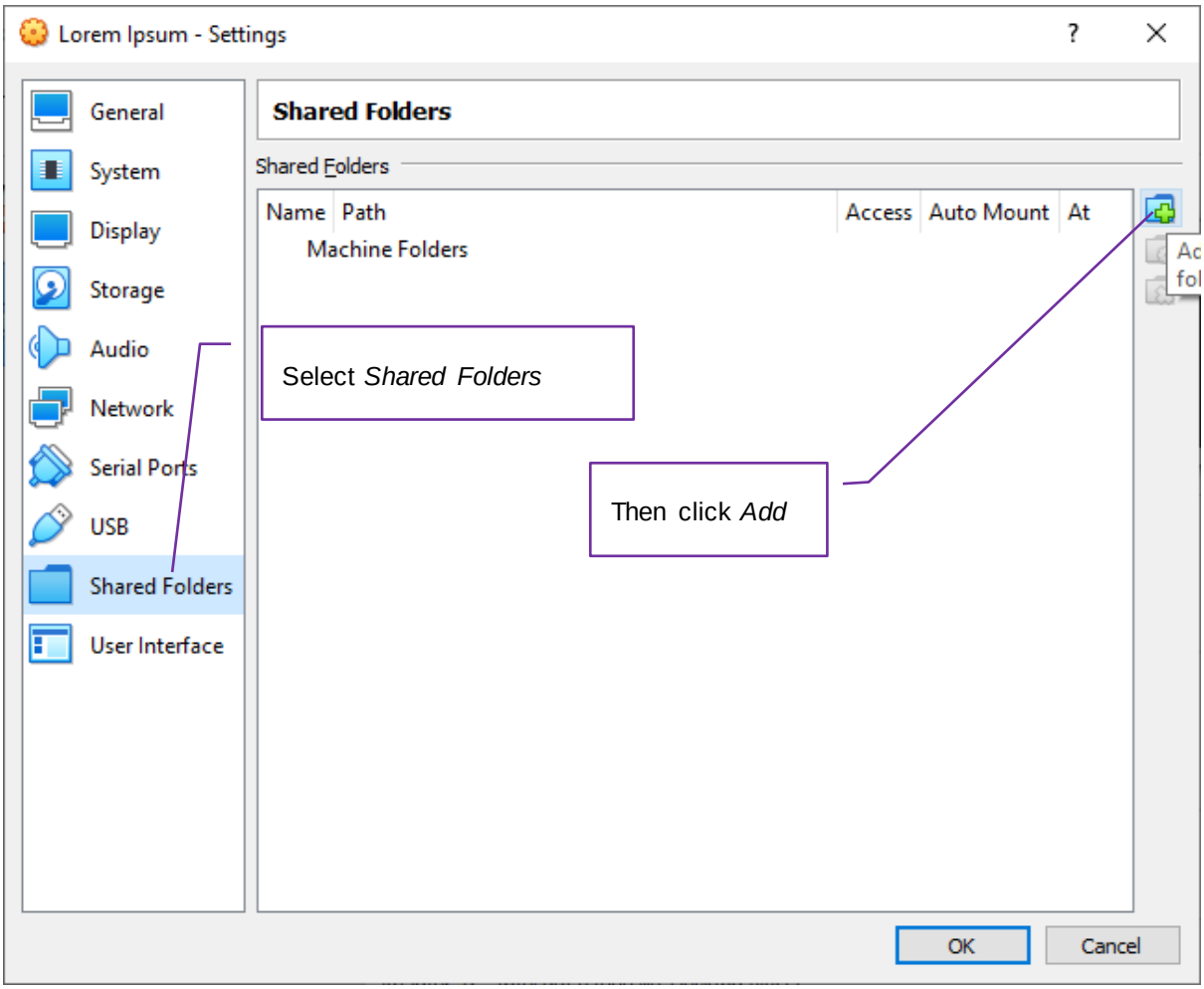
Then select the supplied virtual disk image and select Create.

3.1.2 CREATING A SHARED FOLDER

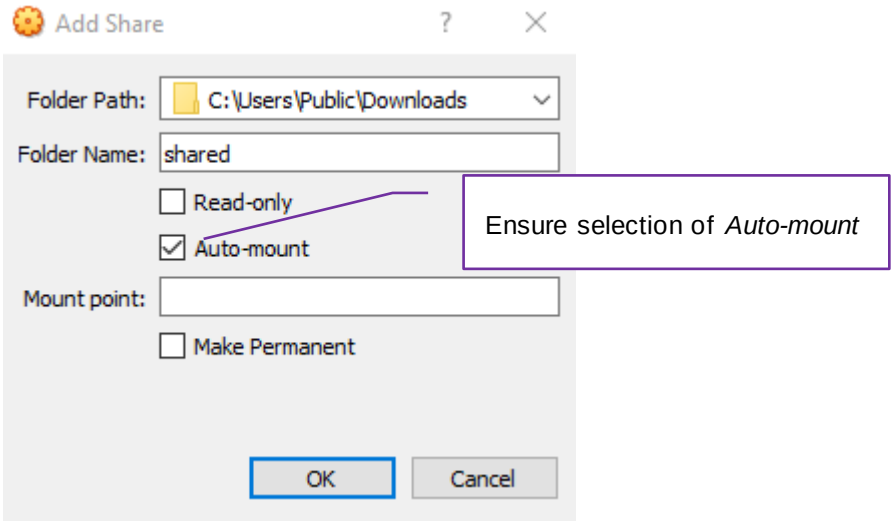
NOTE: Creating a *Shared Folder* is optional, some users prefer to use USB memory to transfer files to the VM. A *Shared Folder* allows files to be transferred from the Host PC to the Virtual Machine without using a USB device.

Once the Virtual Machine has been created, a shared folder needs to be created to transfer the Qt package on to the virtual machine. First, select the machine and select Settings:

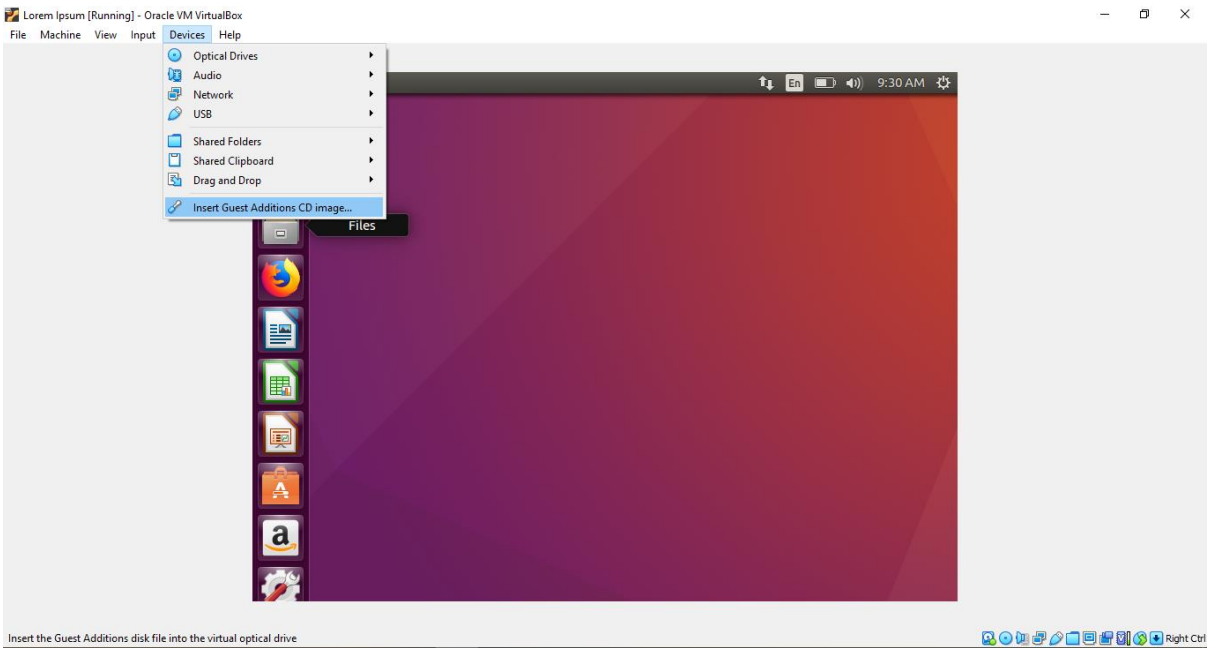




Select the folder path to the supplied Qt package. Use “Shared” for the Folder Name:



Start the Virtual Machine and ensure the shared folder is accessible. Start the machine, allowing it to boot to the desktop, then install Guest Additions by selecting “Install Guest Additions CD Image”:



NOTE: When prompted, enter the password. The factory set password is 'password'

NOTE: The following command assumes the shared folder is mounted as *`/media/sf_shared/`* If the file is not found, use the *mount* command to check how the shared folder is mounted in Linux.

NOTE: File paths in Linux are case sensitive.

NOTE: The version number differs as changes are made to the package so ensure to enter the version of the file you have in place of *x.y.z.*

NOTE: You may need to add the m812 user to the permissions list for vboxsf as follows. Open a terminal and enter `sudo adduser $USER vboxsf`

After installation, reboot the virtual machine. Then, open a terminal (Ctrl+Alt+T) and type the following to copy the package file from the shared folder to the Linux desktop.

```
sudo cp /media/sf_shared/m8xx_qt_package_vx.y.z.tar.gz /home/user/Desktop/
```

Then type the following to take ownership of the file.

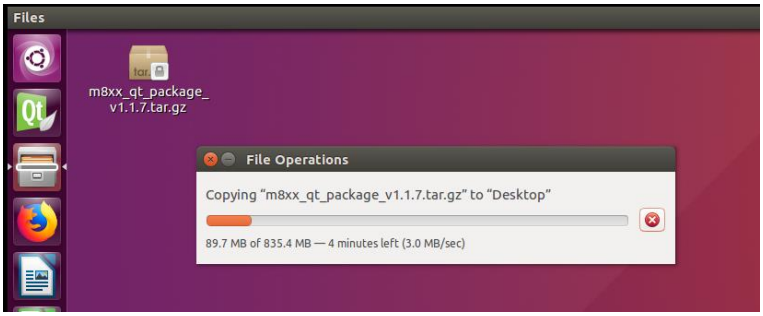
```
sudo chown user:user /home/user/Desktop/m8xx_qt_package_vx.y.z.tar.gz
```

The file is now visible on the desktop, ready to be installed as detailed in the following section.

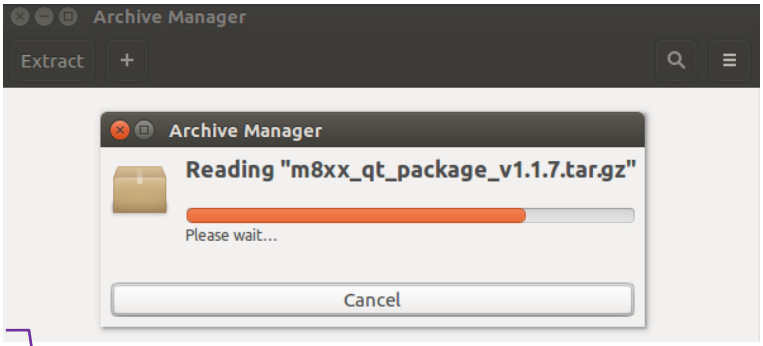
3.2 PACKAGE INSTALL

NOTE: The VM and the Package contain the Virtual Machine and setup for both DSEM812 Qt and DSEM870 Qt.

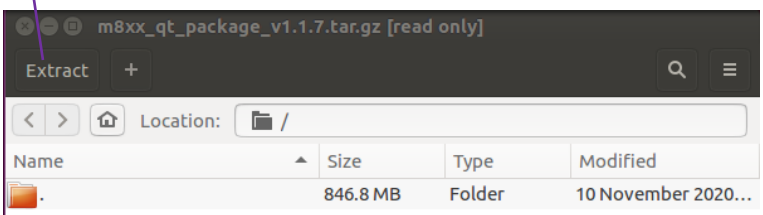
Place the *package* on the Desktop (either by using a *Shared Folder* or a USB memory device),

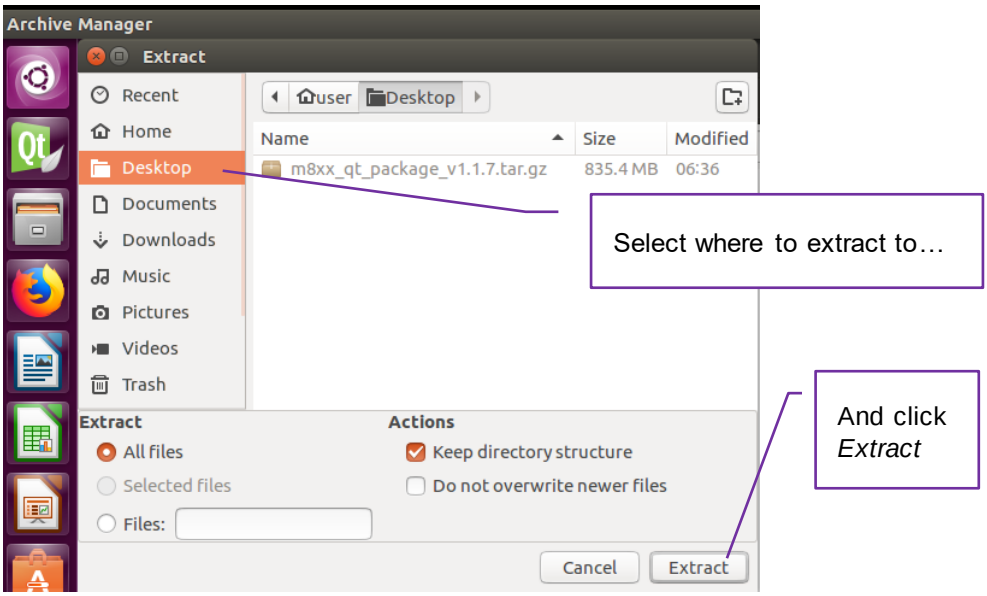


Double click it to read the package.



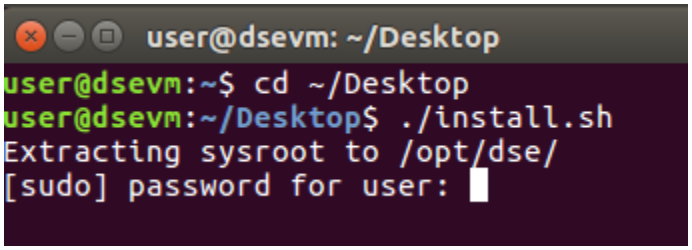
and open the Extract window by clicking *Extract*.



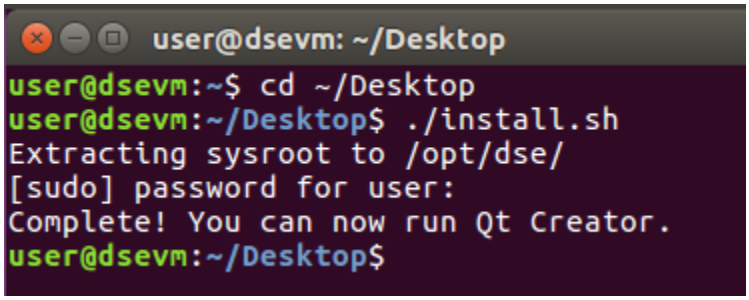


Open a terminal on the Desktop (Ctrl+Alt+T) and run the command

```
cd ~/Desktop
./install.sh
```

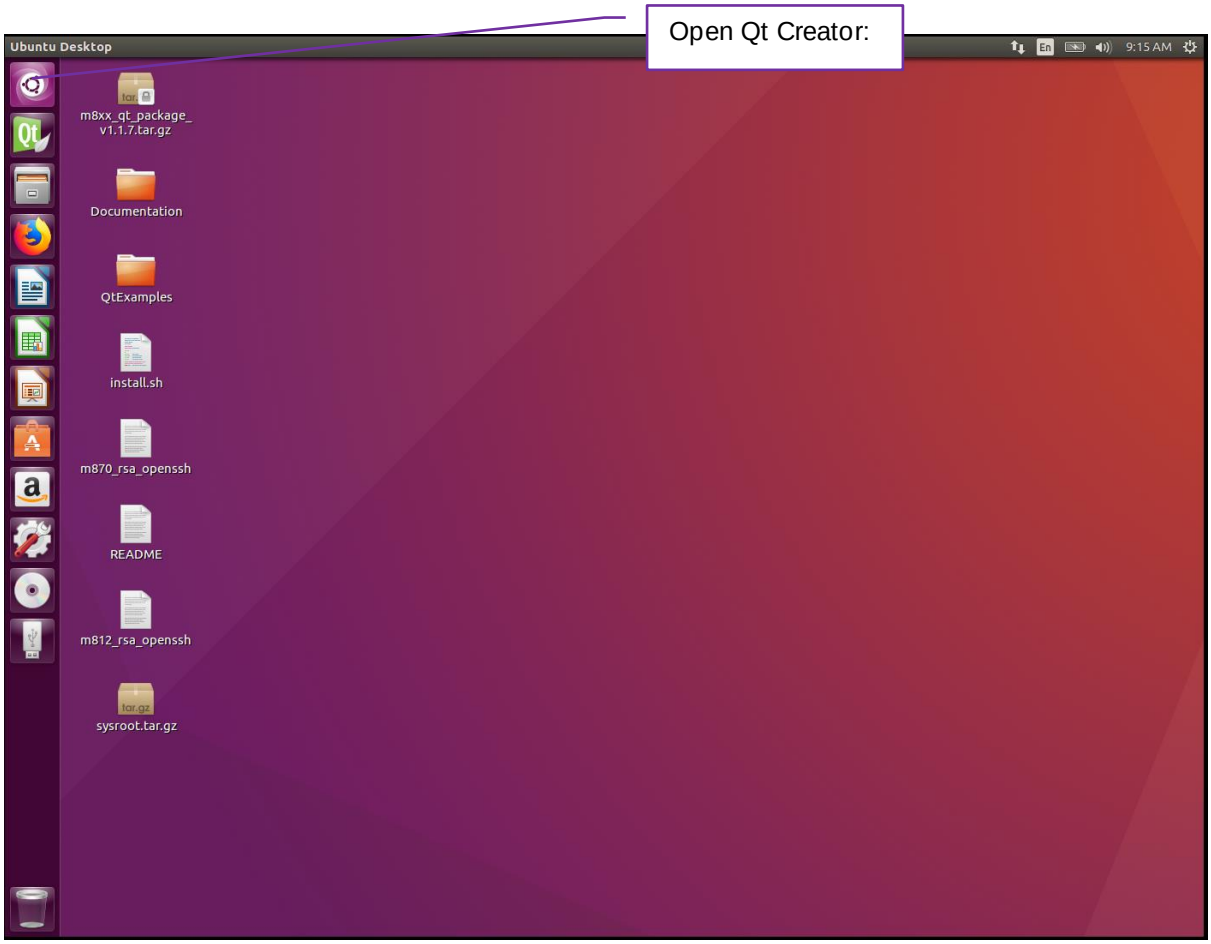


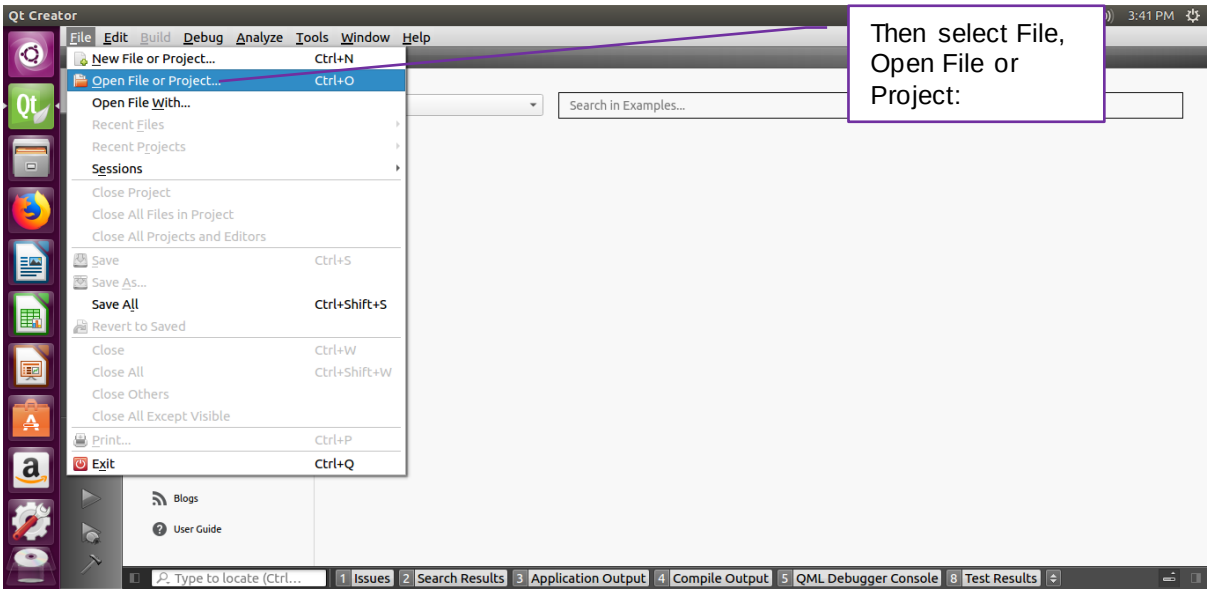
If prompted for a password, enter *password*. Installation is silent and completes in a couple of minutes with confirmation :



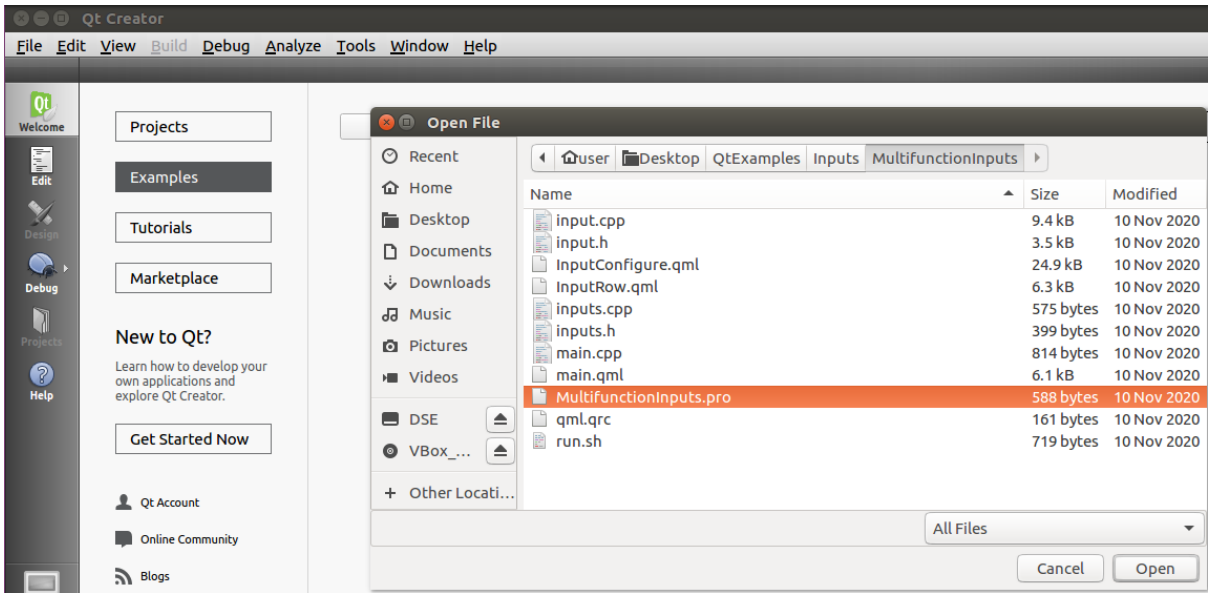
The virtual machine is now ready to compile and program for the chosen device.

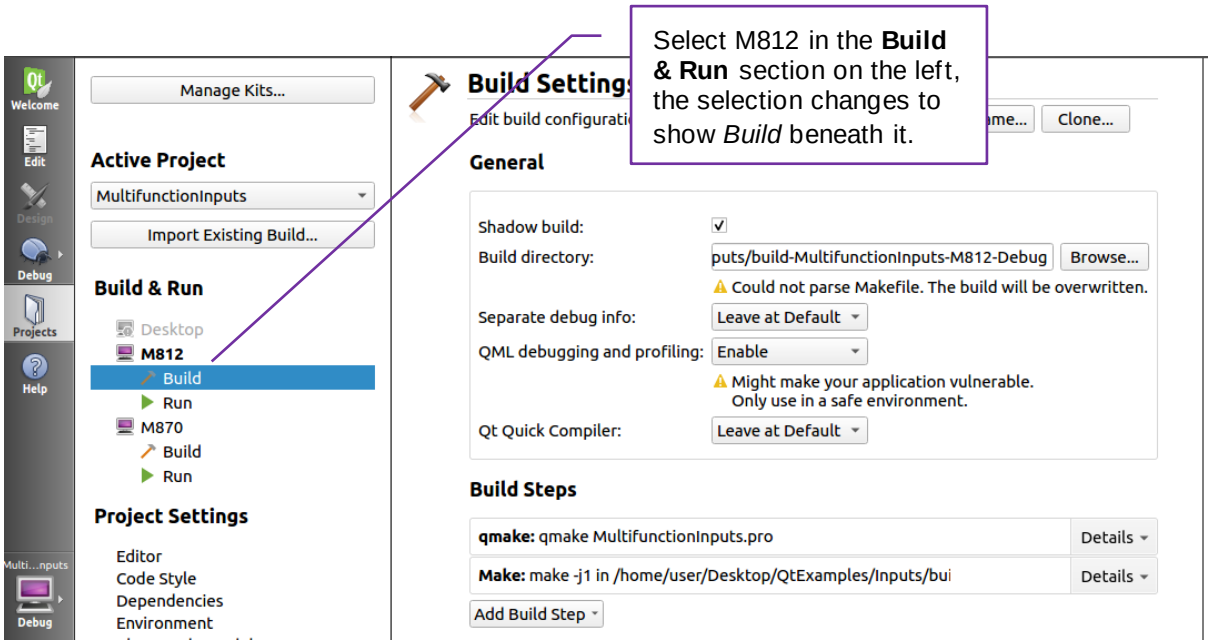
3.2.1 CHECKING THE KIT

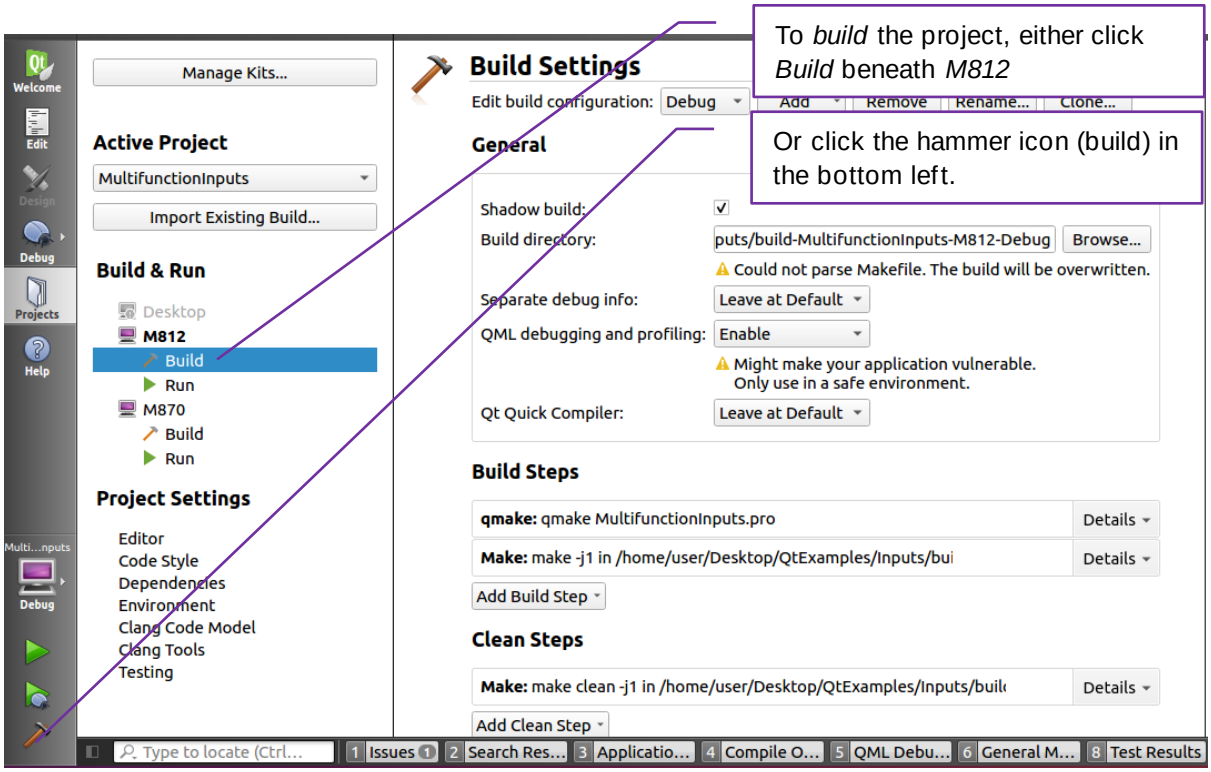




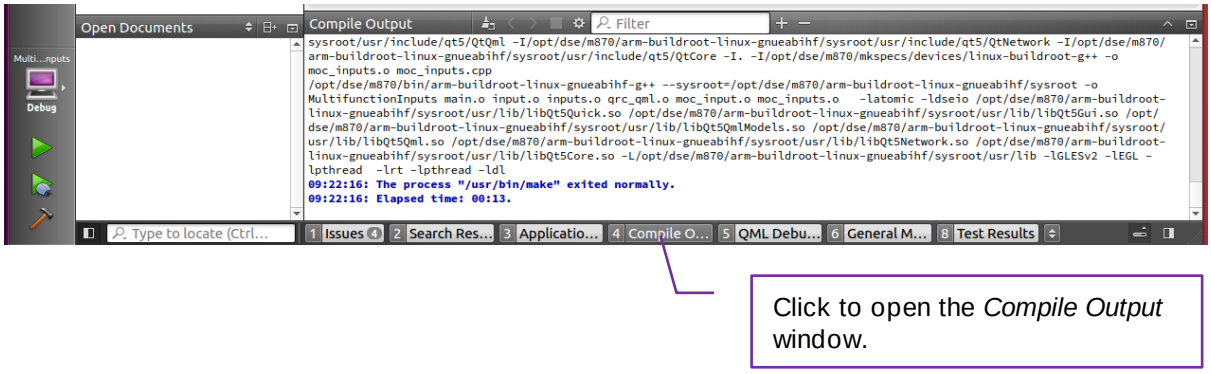
Choose an example from the QtExamples directory on the Desktop. Qt projects have the extension *.pro*. In this case, the QtExamples/Inputs/MultiFunctionInputs/MultiFunctionInputs.pro example was selected:

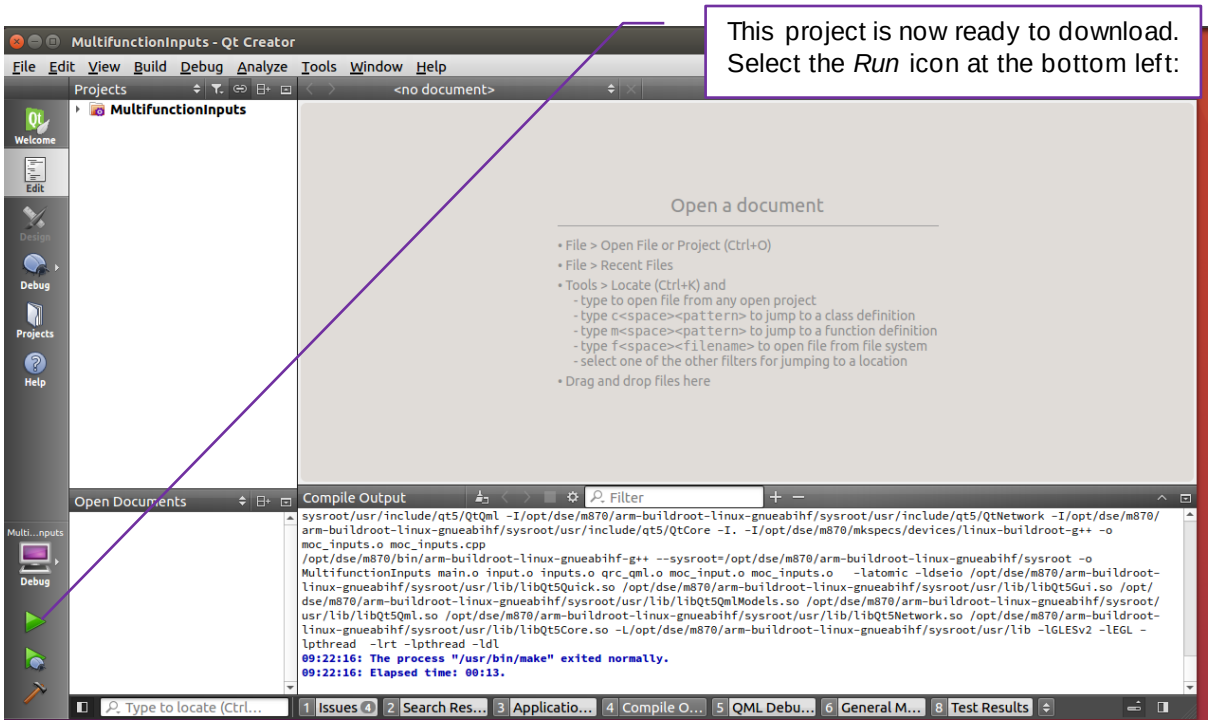






If the build succeeds with no errors, you should have the following in the “Compile Output” window:





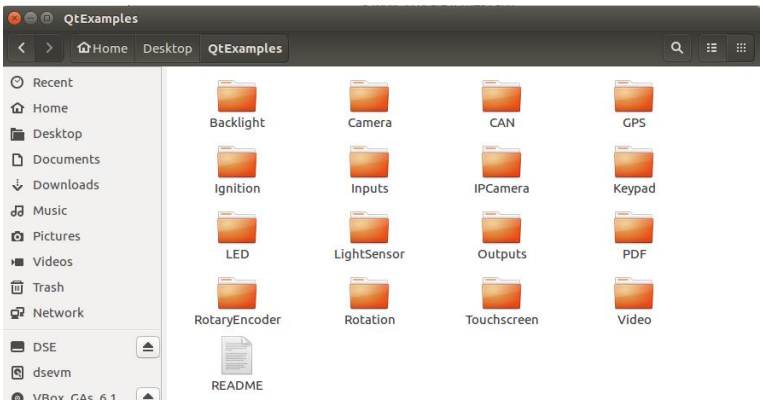
If the program starts on the device, the kit is correct. This is assuming the device is connected to the same network as the virtual machine and the device has the IP address of 192.168.1.100.

If the device has a different IP address, see section entitled *Checking Connection to the Device* elsewhere in this document.

3.2.2 EXAMPLE APPLICATIONS

NOTE: The file README details which examples suit DSEM812.

The package includes example applications relating to DSEM812 and DSEM870 functionality:



4 CONNECTING TO QT

NOTE: Prepare the Qt installation before following this section. See previous section entitled *Installing the Qt Environment* elsewhere in this document.

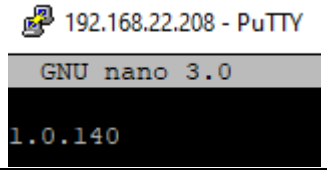
NOTE: Knowledge of Linux and Qt (QML, C++ and JavaScript) is essential and assumed. This manual instructs the Qt programmer how to use Qt in conjunction with the DSE device. This manual does not give instruction for Linux, Qt, QML, C++ or JavaScript. Comprehensive online documentation for Qt is provided at https://doc.qt.io/qt-5/

NOTE: Only one display application may run on the device at a time.

NOTE: ftp is not available. Instead, more secure options are available with *rsync* or *SFTP*.

DSEM812 Qt variant communicates with, and is programmed by, the Qt Integrated Development Environment (IDE). Online documentation for the IDE is available at https://doc.qt.io/qtcreator/

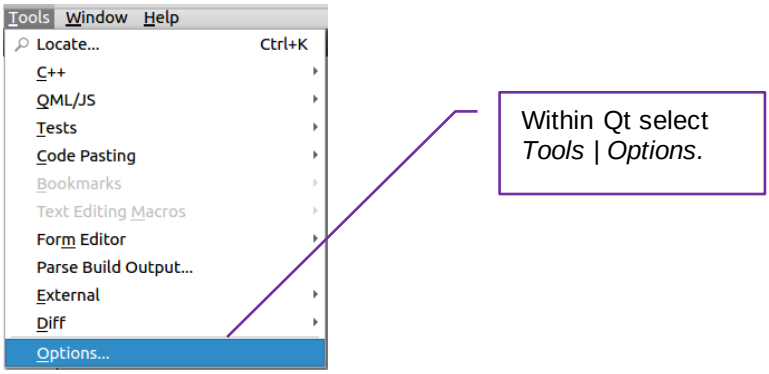
4.1 FILE SYSTEM PATHS

Path	Description
/home/m812	Working folder for all user operations. Contains Run.sh (boot time setup) and all applications that have been deployed to the device.
/etc/Revision.txt	A text file containing the device firmware revision in the format of *major.minor.build.* Example using *nano* to view the file: 

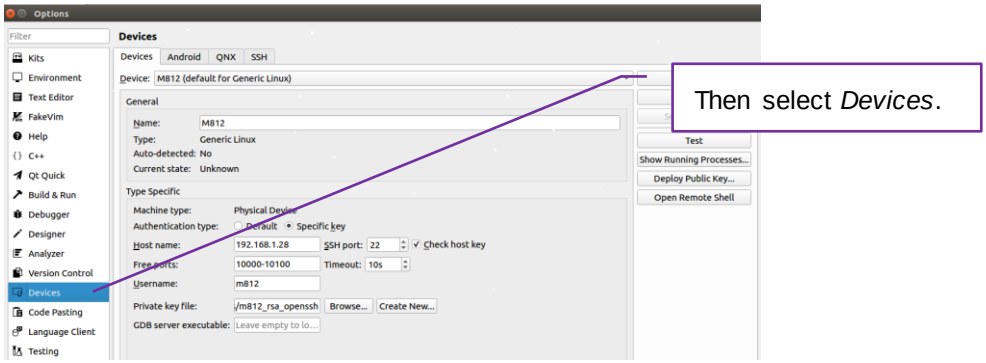
4.2 CHECKING CONNECTION TO THE DEVICE

NOTE: ssh access is secured using a private key file, supplied in the DSE Qt Package (*m812_rsa_openssh).*

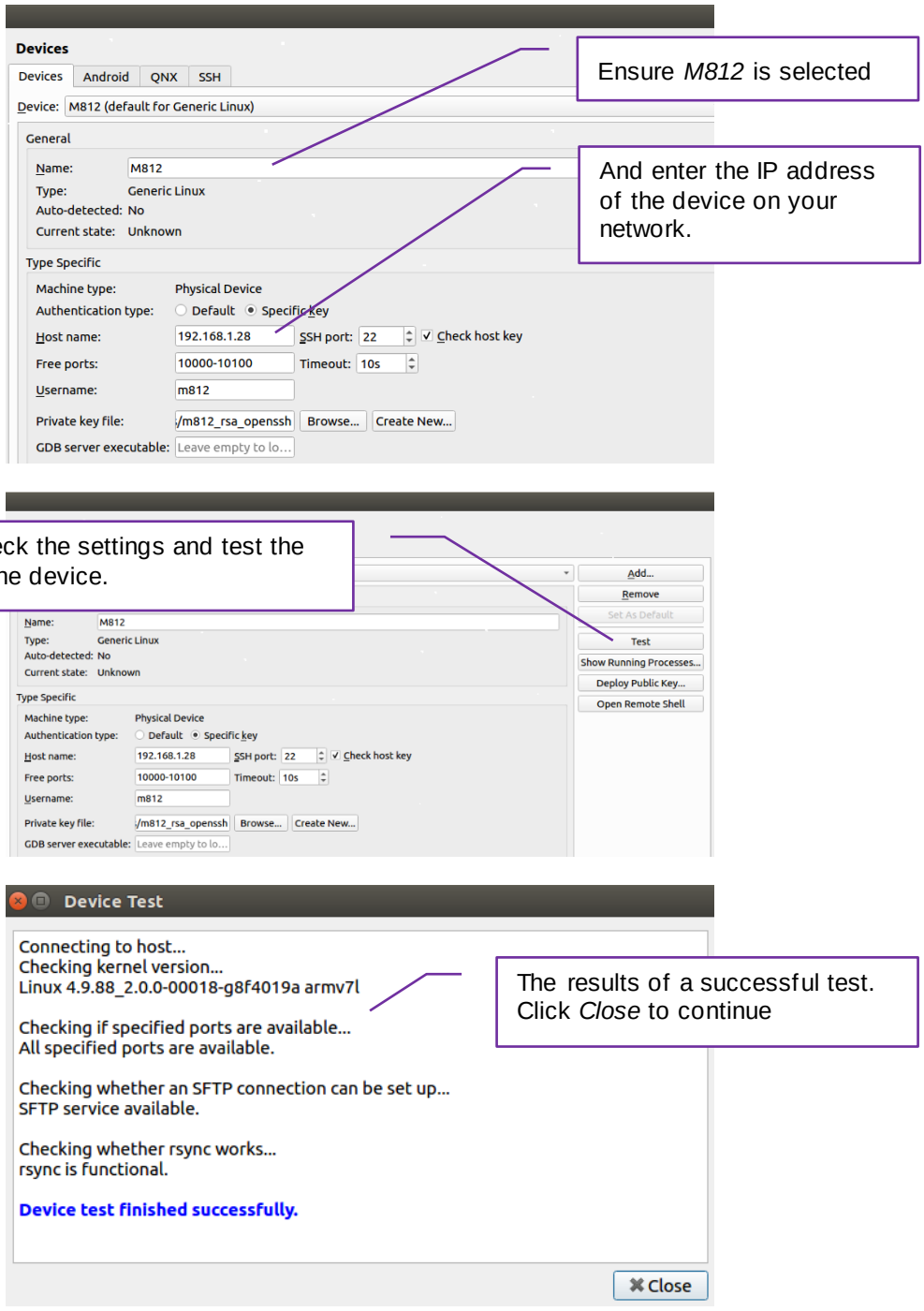
First use the device *Settings Pages* to configure the ethernet port as required. Full instructions how to enter and utilise the *Settings Pages* are contained in *DSE Publication 057-317 DSEM812 Operator Manual.*



NOTE: Changing settings other than those detailed may cause unexpected errors.



NOTE: Changing settings other than those detailed may cause unexpected errors.



Ensure *M812* is selected

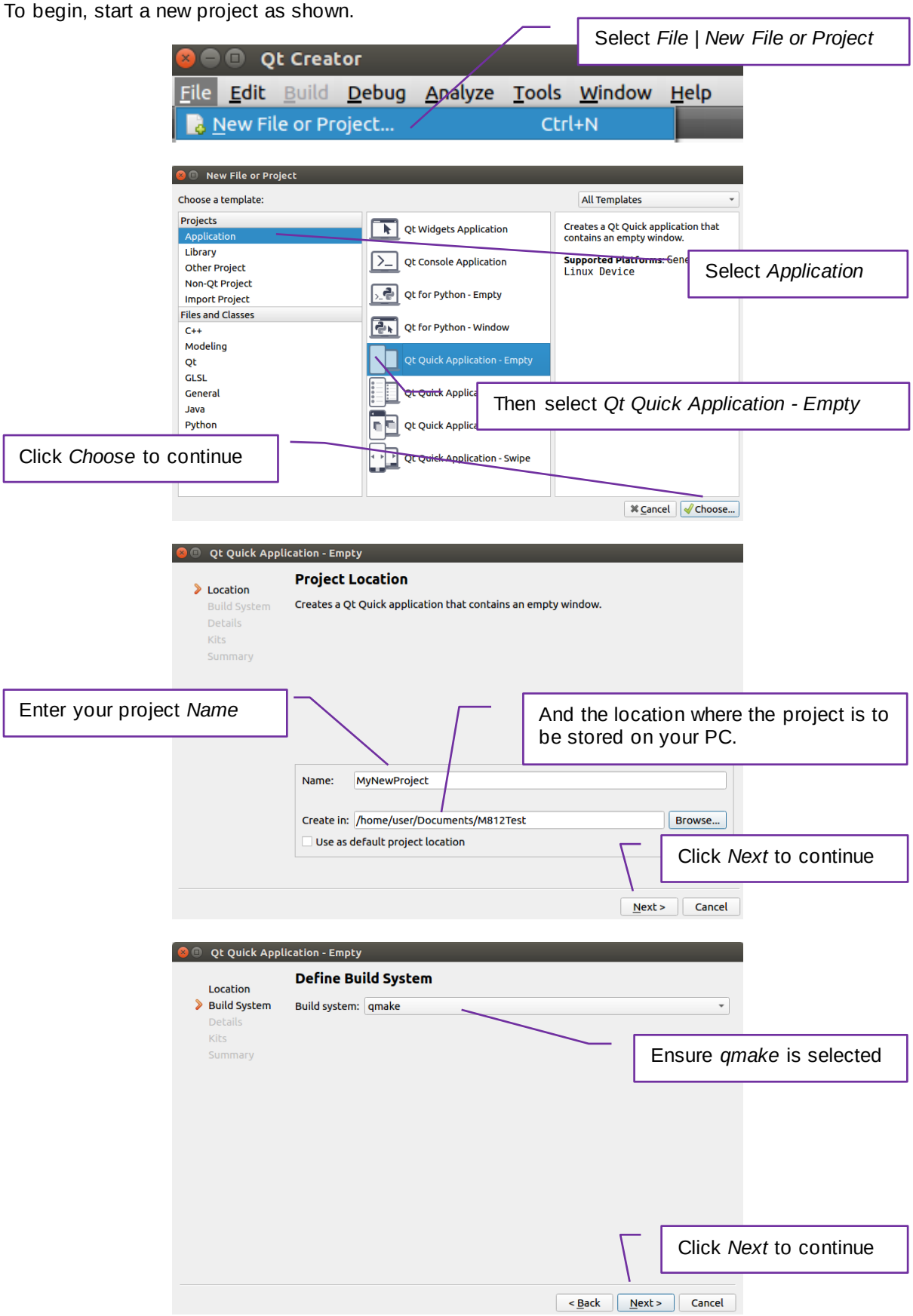
And enter the IP address of the device on your network.

Click *Test* to check the settings and test the connections to the device.

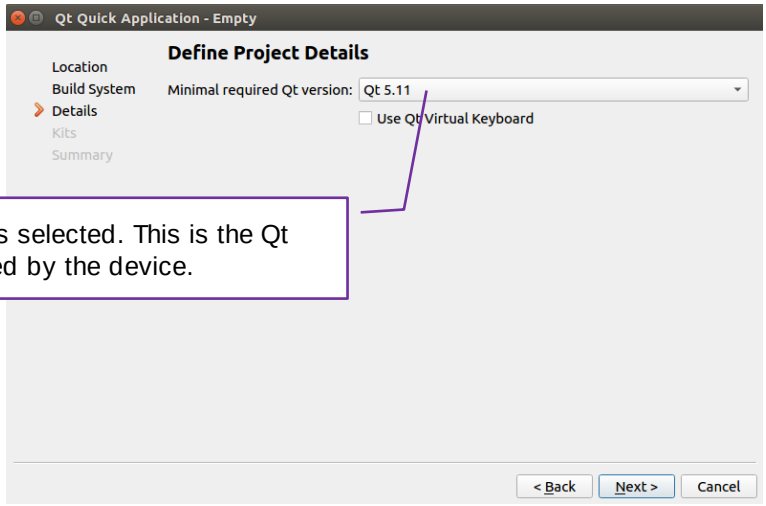
The results of a successful test. Click *Close* to continue

4.3 START NEW PROJECT

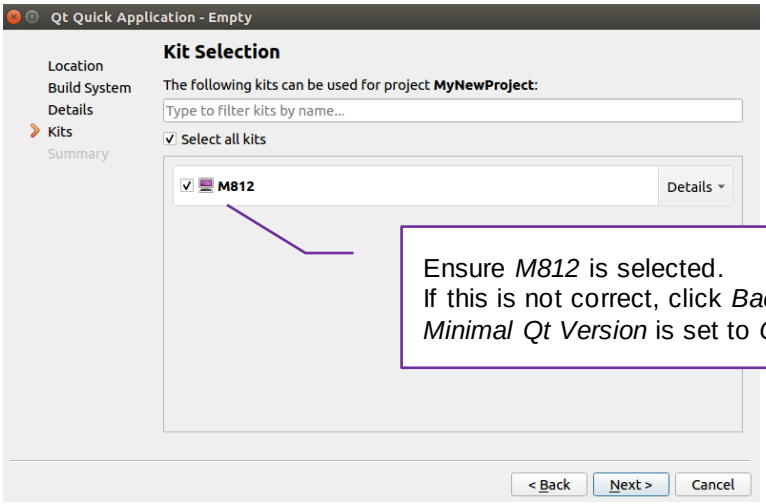
To begin, start a new project as shown.



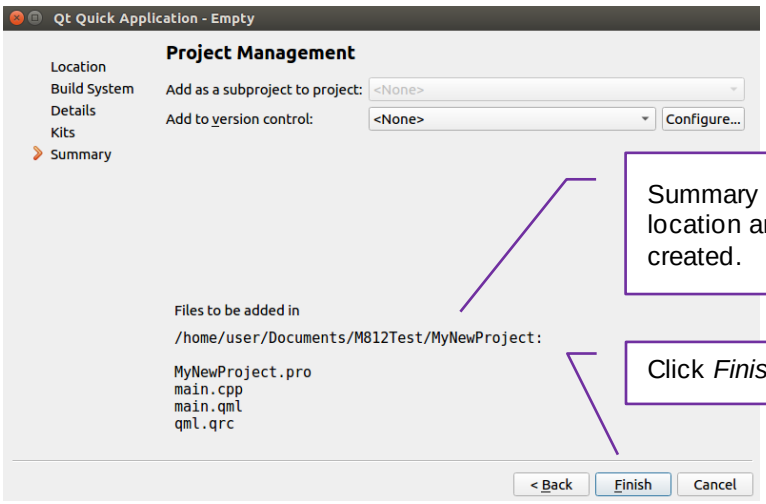
NOTE: DSEM812 is compatible with Qt 5.15. Selection of an incorrect *Minimal required Qt version* may render project creation impossible.



Ensure *Qt 5.15* is selected. This is the Qt version supported by the device.



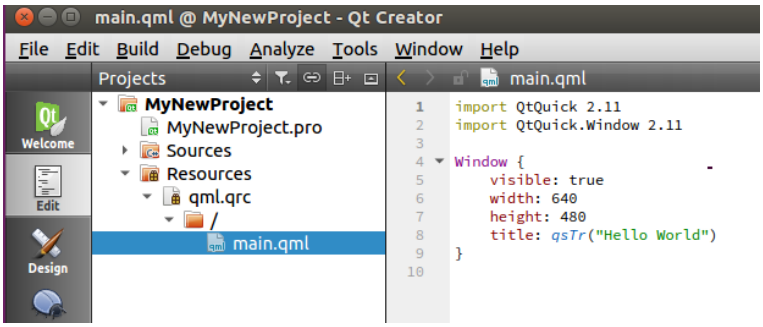
Ensure *M812* is selected.
If this is not correct, click *Back* and recheck *Minimal Qt Version* is set to *Qt 5.15.*



Summary of the project location and files to be created.

Click *Finish* to continue

The new project is created with a simple *Window*.



However, we will modify this to provide a simple application to show something on the screen.

```
import QtQuick 2.11
import QtQuick.Window 2.11

Window {
    visible:    true
    width:      1280
    height:     800

    Text {
        text:       "My New Application"
        visible:    true
        x:          (parent.width - width) / 2
        y:          0
    }
```

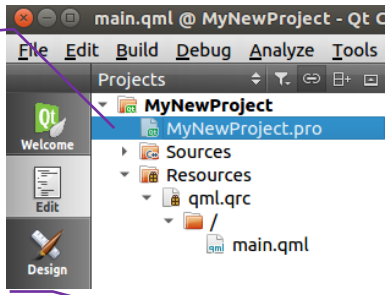
Change *width* and *height* to suite the device's maximum screen size.

Add a text element

Y:0 places the top of *Text* at the top of its *parent (Window).*

parent.width is the width of *Window* (the parent of *Text)*
width is the width of *Text*.
(parent.width – width)/2 places the text centrally within *Window.*

Double-Click on the *.pro* (project) file n the *Projects* window.

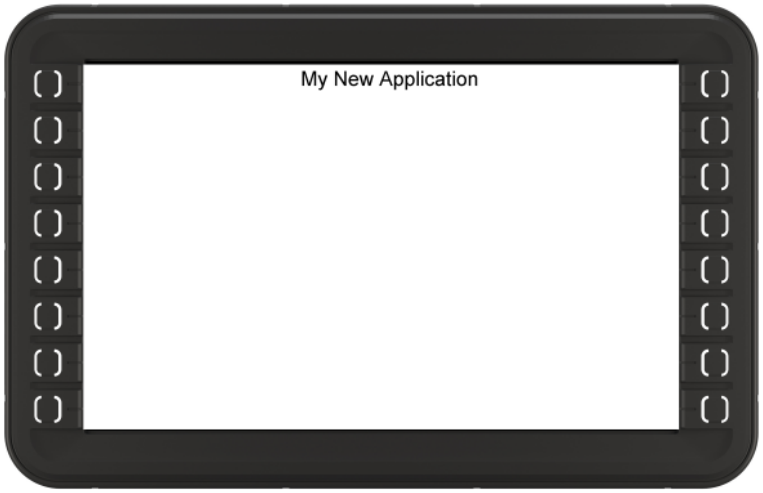


Change the *target.path* to '.'

```
# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = .
!isEmpty(target.path): INSTALLS += target
```

Deploy the application as described in the section entitled *Deploying the Application to the Device* elsewhere in this document.

Results of the application running on the device:



4.4 I/O

Device I/O is controlled and read using *dseio* library. Add the library to *LIBS* within the *.pro* project file.

```
# Add the required DSE libraries
LIBS += -ldseio
```

Include to any *.h* or *.cpp* files that use the library.

```
#include "dseio.h"
```

Then initialised the library before use within *main.cpp.*

```
// Initialise the dseio library
dse_io_lib_init();
```

NOTE: For full details, see *Examples\Inputs* and *Examples\Outputs* provided in the DSEM812 Qt package.

4.5 CAN

DSEM812 is equipped with three CAN ports. Support for configuring them is provided using *ldsenetwork* library.

Add the library to *LIBS* within the *.pro* project file.

```
# Add the required DSE libraries
LIBS += -ldsenetwork
```

Within the cpp file include the required header files

```
#include <QCanBus>
#include "can.h"
#include "dsenetwork.h"
```

Example CAN port configuration
250 = 250 kbit/s
0=Interface 0 (CAN1)

```
dse_network_can_config_t config;

config.size = sizeof(dse_network_can_config_t);
config.bitrate = 250;
config.id = 0;
dse_network_set_can_config(m_interface, &config);
```

4.5.1 PROCEDURE TO CONNECT TO THE CAN

NOTE: A CAN application that does not *receive* CAN messages (used for *transmit* only) must still implement the *FramesReceived* method. This provides a mechanism to retrieve frames from the QCANBus device and prevents the receive memory overflowing.

NOTE: For full details, see *Examples\CAN* provided in the DSEM812 Qt package.

- Use *dse_network_set_can_config* to configure the device hardware.
- Create the *QCANBus SocketCAN*.
- Connect the *SocketCAN* to the device hardware,
- Connect QCANBus device signal *FramesReceived* to the event handler within your own cpp file.
- The event handler you provide for *FramesReceived* is used to filter and parse the incoming CAN frames (messages) as required.

4.6 GPS

NOTE: A suitable external GPS antenna is required. For full specification see DSE Publication *057-317 DSEM812 Operator Manual.*

Within the QML file, the following imports are required:

```
import QtPositioning 5.11
import QtLocation 5.11
```

QML PositionSource item is used to retrieve the location at the given *updateInterval* (ms).

```
// Get the GPS location
PositionSource {
    id: gpsLoc
    updateInterval: 1000
    active: true

    onPositionChanged: {
        // Code to handle the change in position if required
    }
}
```

4.6.1 TROUBLESHOOTING GPS

- Ensure the GPS antenna is correctly connected, has a clear, wide view of the sky, and is preferably mounted outdoors.
- Where the last connection was in a different location, or was some time ago, the GPS receiver needs to connect with the satellites and download the 'almanac'. This takes a number of minutes so be patient while awaiting the first location fix.
- To check the raw data coming from the GPS receiver, see the following subsection.

4.6.2 VIEWING THE RAW GPS DATA

Data is transmitted by the GPS receiver to the DSEM812 in NEMA format. The raw data may be viewed by opening a remote shell to the M812 (from Qt, *Tools* | *Options* | *Open Remote Shell*) and executing the following command:

```
sudo microcom -s 38400 /dev/ttymxc2
```

Messages are received from the GPS every second.

Example with no location fix:
```
$GPRMC,010803.020,V,,,,,0.00,0.00,060180,,,N*4A
```

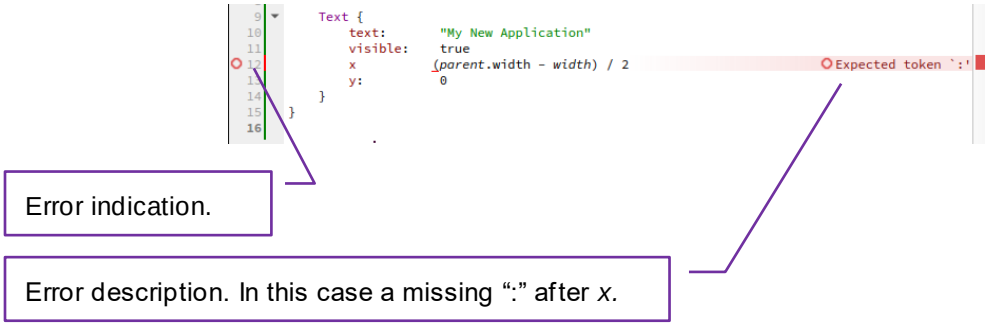
Example with location fix:
```
$GPRMC,085908.000,A,5307.4553,N,00113.9041,W,0.19,154.92,151121,,,A*7D
```

4.7 DEPLOYING THE APPLICATION TO THE DEVICE

NOTE: Successful communications with the device is required before deploying the application. Refer to section entitled *Checking Connection to the Device* elsewhere in this document.

4.7.1 ERROR CHECKING

NOTE: Before clicking *Run* ensure there are no errors in the code. Many errors are detected by Qt before compilation and are shown with red underlines in the code.



Some errors are not detected until the project *Build* process. Press  to build (without deploying).

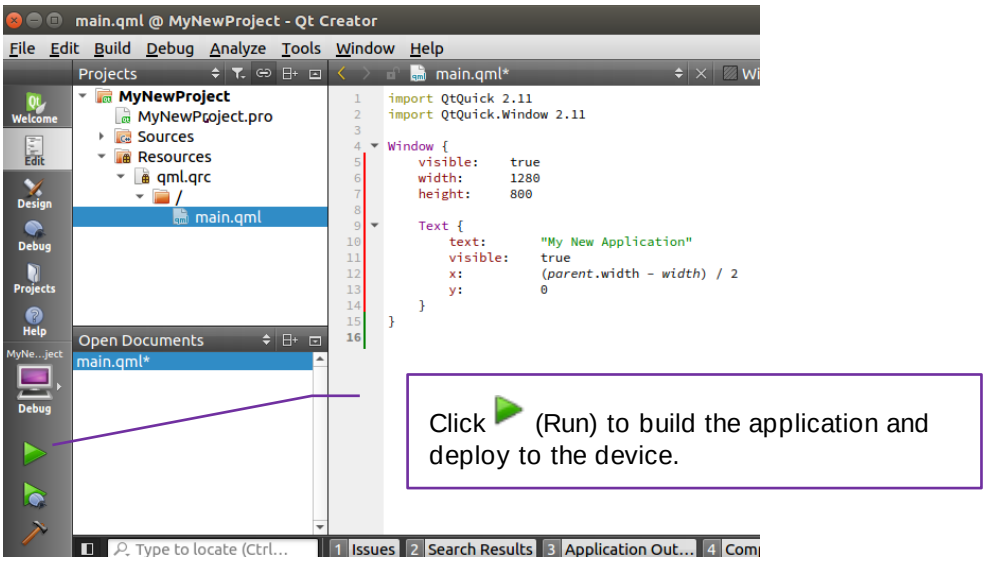
4.7.2 DEPLOY APPLICATION

NOTE: Ensure *Device Settings* application is closed (exited) before sending your application to the device. Failure to do this may result in multiple applications displaying items on the display at the same time.

NOTE: When updating an application on the device, ensure the 'old' version running on the device is closed before writing the updated version to the device. For further details, see section entitled *Ceasing (killing) a Running Application* elsewhere in this document.

NOTE: The device accepts multiple applications to be installed to it at the same time. To select the application to start at device boot up, refer to section entitled *Selecting an Application to Auto Run* elsewhere in this document.

NOTE: Where multiple applications are running on the device simultaneously, ensure there is no conflict in the use of the display. It is recommended that only one application utilises the display.



4.8 SELECTING AN APPLICATION TO AUTO RUN

NOTE: The device detects the presence of *Run.sh* and creates a new 'factory set' file if not present.

To configure the device to automatically start an application upon device boot up we must edit *Run.sh* on the device.

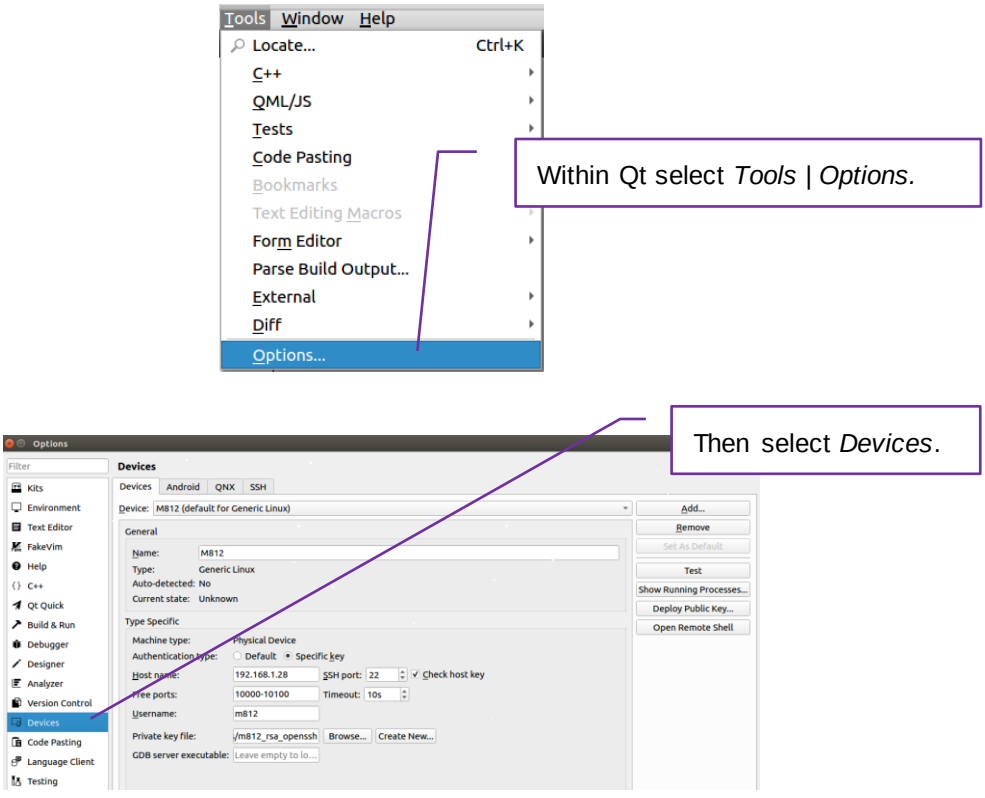
Options are

- Maintain a *Run.sh* file in the project folder. Deploy this file by adding the following to the project file (.pro file)

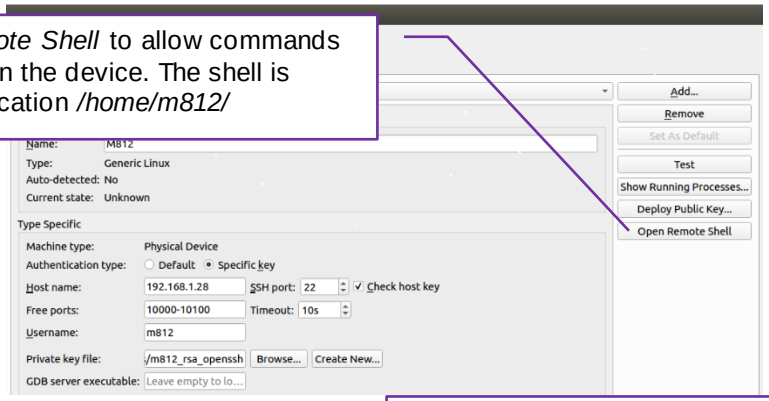
```
scripts.files += $$PWD/run.sh
scripts.path = .
INSTALLS += scripts
```

  This method is detailed within the Examples provided with the DSEM812 Virtual Machine Package.

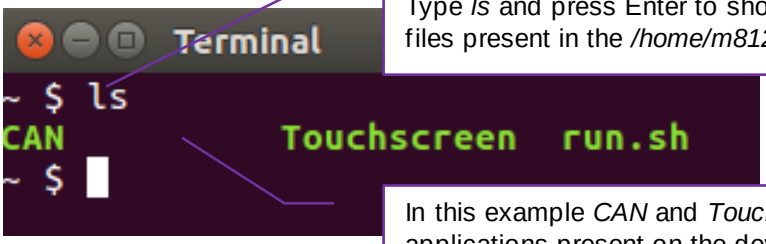
- Modify *Run.sh* in the location */home/m812* as detailed below :



Click *Open Remote Shell* to allow commands to be actioned on the device. The shell is opened at the location */home/m812/*

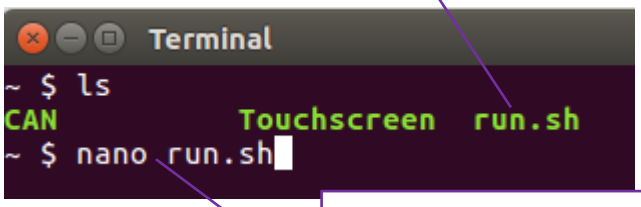


Type *ls* and press Enter to show a list of the files present in the */home/m812* folder.



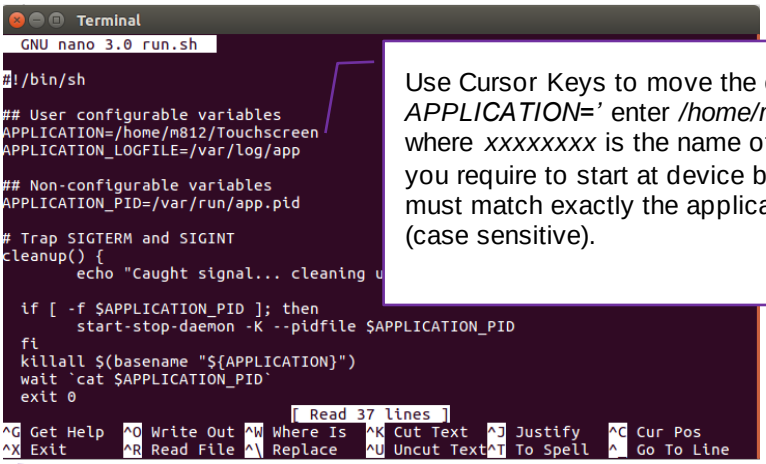
In this example *CAN* and *Touchscreen* are two applications present on the device.

This is the *Shell File* that is run at device bootup.



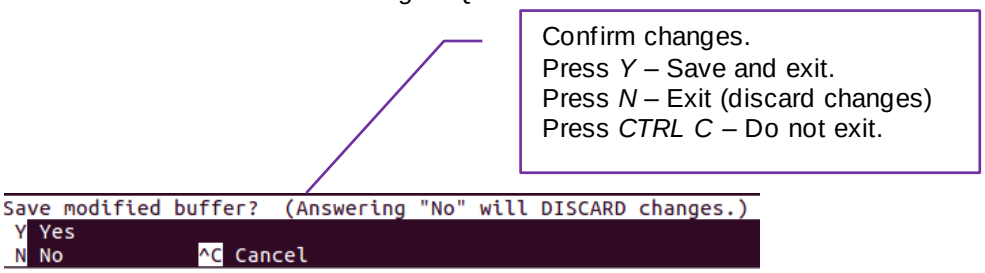
Type *nano run.sh* to open Nano (a text editor) and edit the *Run.sh* file

NOTE: The 'factory set' *Run.sh'* contains *clean-up* code to gracefully close the application should the calling process be terminated. This may be removed if desired, maintaining only the APPLICATION variable naming the application to run at boot time.



Use Cursor Keys to move the cursor and after *APPLICATION='* enter */home/m812/xxxxxxxx* where *xxxxxxxx* is the name of the application you require to start at device boot. This name must match exactly the application name (case sensitive).

Press *CTRL X when done.*

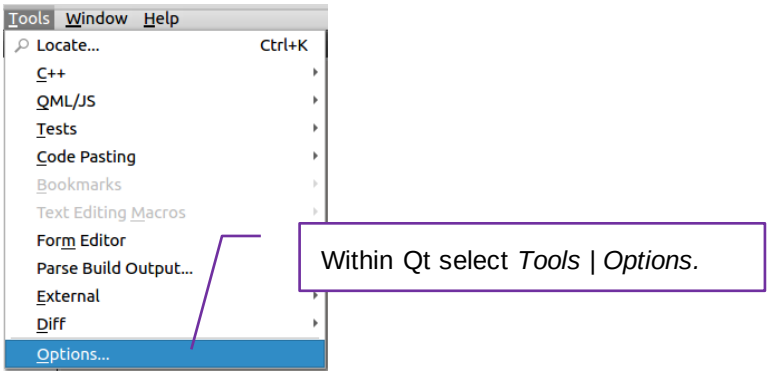


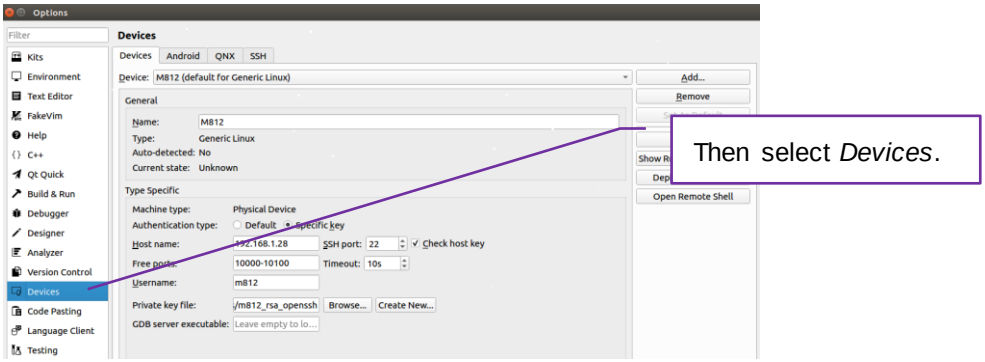
Close the Terminal window and power cycle the device to check the change.
If your application fails to start at boot up:

- Use *ls* to check the presence and exact spelling of the application on the device in folder */home/m812*. (Check case)
- Use *Nano* to check the path is correctly entered (case sensitive).
- Check the application by executing it directly from the *Remote Terminal*. Enter its name at the command prompt.

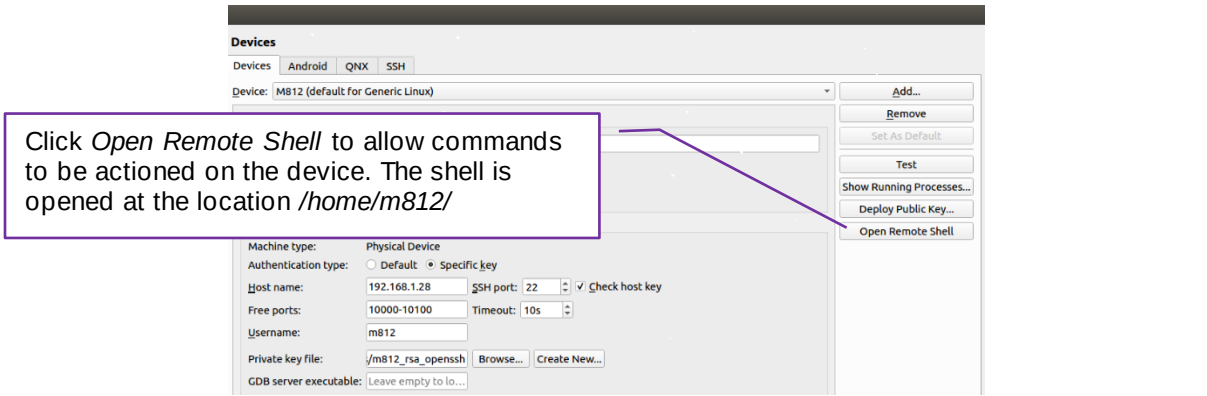
4.9 CEASING (KILLING) A RUNNING APPLICATION

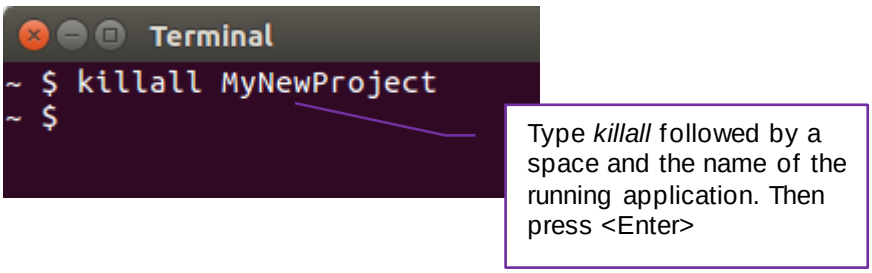
To cease a running application, use a Remote Shell:





NOTE: *Show Running Processes* is an advanced option to list all running applications on the device. The popup may also be used to stop those processes. However accidental use may stop a critical process requiring a device reboot. For this reason, it is recommended to use *Remote Shell* as detailed below.





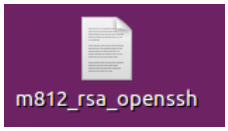
5 ADDITIONAL INFORMATION

This section details additional information not directly related to Qt.

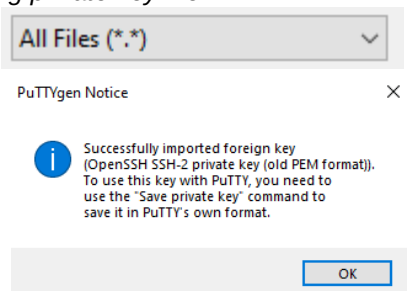
5.1 USE WITH PUTTY

While the remote terminal within Qt is very useful, sometime it's desired to use a standalone terminal program. This section describes the use of the commonPuTTY application.
Security is by Private Key.

5.1.1 CREATE THE PRIVATE KEY

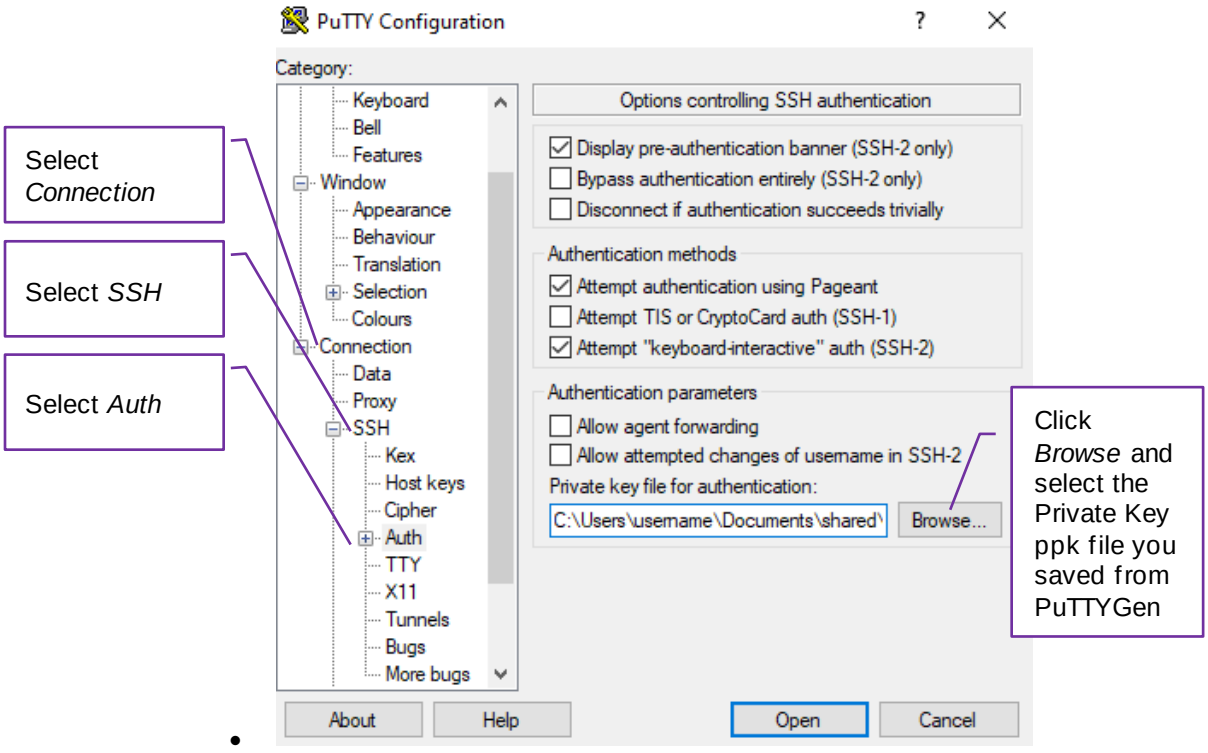


- Locate the file *m812_rsa_openssh* supplied in the DSE Qt Package. This is in the location where you untarred the package, often *Desktop* in the Virtual Machine.
- Using the *shared folder* or a USB memory stick (for example), copy this file to the PC where PuTTY is installed.
- Start PuTTYgen and select *LOAD* alongside *Load an existing private key file.*
- Change the file filter to display *All Files* and select *m812_rsa_opensssh*.
- Confirmation is given:
- Click *OK*
- Click *Save Private Key*. (You can set a passphrase first if you wish).
- The key is saved as *PuTTY ppk* format.

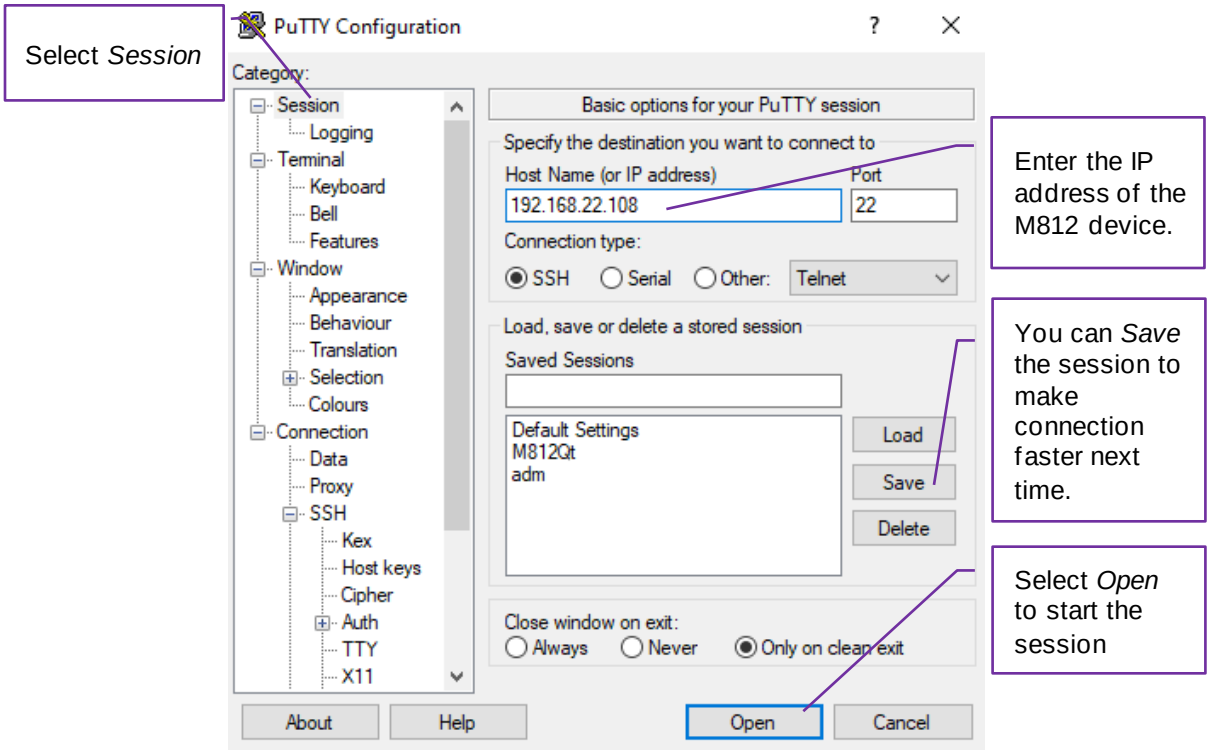


5.1.2 START PUTTY

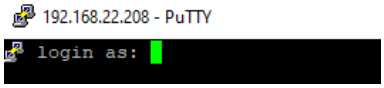
- Start PuTTY.



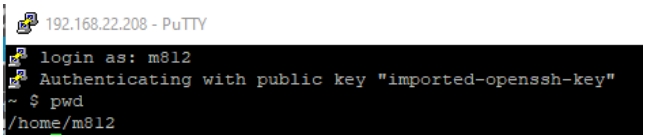
-



- After a short connection delay you are prompted to log in :



- Enter username *m812* and press Enter. You are now logged in.



6 MAINTENANCE AND WARRANTY

The device is *Fit and Forget*. As such, there are no user serviceable parts within the controller.
In the case of malfunction, you should contact your original equipment manufacturer (OEM).

DSE Provides limited warranty to the equipment purchaser at the point of sale. For full details of any applicable warranty, refer to the original equipment supplier (OEM).

7 DISPOSAL

7.1 WEEE (WASTE ELECTRICAL AND ELECTRONIC EQUIPMENT)

If you use electrical and electronic equipment you must store, collect, treat, recycle, and dispose of WEEE separately from your other waste

8 MISCELLANEOUS

This product includes copyrighted third-party software licensed under the terms of the GNU General Public License. A copy of the corresponding source code for all included third-party software is available on request, please contact DSE Technical Support for additional information.

This Page is Intentionally Blank

This Page is Intentionally Blank