Qt Installation Guide

by Brian Fraser
Last update: November 7, 2011

This document guides the user through:

1. Building and installing Qt on for the host (native) and target (cross-compile) environment.
2. Building Qt applications for each target.

Table of Contents

Caution: This guide has not yet been tested in the SFU Surrey Linux Lab (SUR4080). Some changes may be needed.

Do not attempt any of the "sudo" commands in the SFU Linux Lab. You do not have permission to do so, and it notifies our IT staff. If something is not setup, please email me and I'll ask them to set it up for us.

Formatting:

1. Commands starting with > are Linux console commands:
   ```
   > echo "Hello world!"
   ```
2. Almost all commands are case sensitive in Linux and U-Boot.

Revision History:

- Nov 6: Initial version.
- Nov 7: Updated based on feedback:
  - Corrected many references to the public folder and changed to private folder.
  - Changed /sbin/test to /sbin/ts_test
  - Made some copy commands recursive.

1. Download Packages

1. Install the `autoconf` tools (do not try in Linux lab; see caution on front page):
   ```
   > sudo apt-get install autoconf libtool
   ```
2. Download the following archives into `~/cmpt433/private`:
   - From `ftp://ftp.qt.nokia.com/qt/source/` download:
     `qt-everywhere-opensource-src-4.7.3.tar.gz`
   - From `https://github.com/kergoth/tslib` download:
     `tslib-1.0-101-???y.zip`
     - The small "ZIP" button is near the top of the web page, on the left-hand side.
     - The ???'s will be replaced with letters in the file you download.
3. Change to the `~/cmpt433/private` directory and extract each archive:
   ```
   > tar -xvf qt-everywhere-opensource-src-4.7.3.tar.gz
   > unzip tslib-1.0-101-??????.zip
   > mv kergoth-tslib-?????? tslib-src
   ```

2. Touchscreen library:

1. Change to the `tslib` source directory:
   ```
   > cd ~/cmpt433/private/tslib-src
   ```
2. Generate the configuration script file:
   ```
   > ./autogen.sh
   ```
3. Configure `tslib` (one line):
   ```
   > ./configure --prefix ~/cmpt433/private/tslib --host=arm-linux CPPFLAGS="-O0
   -Wall -W -march=armv4t -msoft-float -mtune=arm920t -mcpu=arm920t" CFLAGS="-O0
   -Wall -W -march=armv4t -msoft-float -mtune=arm920t -mcpu=arm920t"
   ```
4. Install `tslib` on Host:
   ```
   > make install
   ```
5. Check `tslib` host installation:
   ```
   > ls ../tslib/
   ```
   - You should see 4 directories:
     ```
     bin  etc  include  lib
     ```
6. Copy `tslib` executables to root-files-system (RFS) on host. This assumes that your RFS is named `fs-qt4.7.3`
   - The executables include:
     - `ts_calibrate`: Utility to calibrate the touch screen.
     - `ts_test`: A simple drawing program to test the touch screen.
   - Copy the contents of `/tslib/bin` into .../`fs-qt4.7.3/sbin/`
     ```
     > cp ~/cmpt433/private/tslib/bin/* ~/cmpt433/private/fs-qt4.7.3/sbin/
     ```
     - Replace any existing files.

- Copy the contents (and sub-folders) of `/tslib/lib` into `.../fs-qt4.7.3/lib`
  ```
  > cp -r ~/cmpt433/private/tslib/lib/* ~/cmpt433/private/fs-qt4.7.3/lib/
  ```
  - Merge with existing content; replacing duplicate files.

7. Configure the Touchscreen (and Qt)
   - Place the following content in a new file named `qt_env.sh` under the `/etc` directory in your **root file system** on the host:
     ```
     > gedit ~/cmpt433/private/fs-qt4.7.3/etc/qt_env.sh
     #!/bin/sh
     # Environment variables for TouchScreen:
     export set TSLIB_TSDEVICE=/dev/event0
     export set TSLIB_CONFFILE=/etc/ts.conf
     export set TSLIB_PLUGINDIR=/lib/ts
     export set TSLIB_CALIBFILE=/etc/pointercal

     # Qt Configuration:
     export set QWS_KEYBOARD="TTY:/dev/tty1"
     export set QWS_MOUSE_PROTO="tslib:/dev/event0 IntelliMouse:/dev/mice"
     export set LD_LIBRARY_PATH=/opt/qtlibs/
     export set QT_QWS_FONTDIR=/opt/qtlibs/fonts
     ```
8. Place the following content in a new file named `launchqt` under the `/sbin` directory of your **RFS** on the host:
   ```
   > gedit ~/cmpt433/private/fs-qt4.7.3/sbin/launchqt
   #!/bin/sh
   # Short script to launch a Qt application and
   # the touch-screen calibration (if necessary).

   if [ ! -f /etc/pointercal ] ; then
           echo "Calibrating TouchScreen..."
           /sbin/ts_calibrate
   fi

   if [ $# -gt 0 ] ; then
           $1 -qws
   fi
   ```
9. Change the permissions on the `launchqt` to executable:
   ```
   > chmod a+xr ~/cmpt433/private/fs-qt4.7.3/sbin/launchqt
   ```
10. Edit the RFS's `/etc/init.d/rcS` file:
    ```
    > gedit ~/cmpt433/private/fs-qt4.7.3/etc/init.d/rcS
    ```
    - Comment out the line which launches the welcome screen:
      ```
      # Not needed at start-up.
      # /opt/launchQtApp /opt/welcomescreen&
      ```
    - Still in `rcS`, add the following lines which will:
      - "source" the above configuration file, setting up the runtime environment to launch Qt applications (setting up the fonts, the touch screen, ...) during startup; and
      - launch the touch-screen to calibrate at start-up if required

```
source /etc/qt_env.sh
/sbin/launchqt &
#/sbin/launchqt yourAppAtStartupHereInsteadOfAbove &
```

- Only have one call to `launchqt` in your start-up process. Call the script without an argument to just run the touch-screen calibration (if required). Or pass it the filename of a Qt program to have it launch it after the touchscreen calibration is complete.

11. Edit the RFS's `/etc/profile` file:
    ```
    > gedit ~/cmpt433/private/fs-qt4.7.3/etc/profile
    ```
    - Add the following which will allow you to launch Qt applications directly from the command line without setting any extra environment parameters:
      ```
      source /etc/qt_env.sh
      ```
12. Build the RFS and download it to the board; boot the board. See the quick-start guide from earlier in the semester for directions.
    - When Linux reboots, it should now show you the touch-screen calibration screen automatically.
    - The calibration will only run once each time you download a new root file system.
13. Test the touchscreen on the target:
    ```
    > /sbin/ts_test
    ```
    - This should display a simple drawing application. When you move your finger around on the screen, the cursor should respond and follow your finger reasonably well.
14. Trouble shooting:
    - Test that touch-screen gives input by running the following on the target:
      ```
      > cat /dev/event0 | hexdump
      ```
      - It should get numbers/text on the screen each time you press the screen. If it does, it means your hardware is working correctly and that Linux is correctly configured.
      - Otherwise, ensure the kernel is correctly configured with the touch-screen drivers. In the kernel sub-directory:
        ```
        > make menuconfig
        ```
        - Under: Device Drivers --> Input device support:
          - Horizontal screen resolution: 320
          - Vertical screen resolution: 240
        - Under: Device Drivers --> Touchscreen
          - <*> EmbedSky TQ2440 TouchScreen input driver
        - Under: Device Drivers --> Graphics support
          - <*> Support for frame buffer devices -->
          - <*> S2C2410 LCD framebuffer support
        - [*] EmbedSky SKY2440/TQ2440 Board Backlight control

        - If you made any changes, rebuild your kernel and re-download it and retry the above hardware test.
    - Failing this, the LCD cable on target may need to re-seated; talk to instructor. I do not recommend you try this yourself.
- The touch-screen calibration should only run once each time you re-download the root-file system. If it does not run, check the following.
    - Prove that the file is there by manually running the calibration:
      ```
      > ts_calibrate
      ```
    - Ensure that your target's RFS has both necessary script files (and are correct):
      ```
      > cat /etc/qt_env.sh
      > cat /sbin/launchqt
      ```
    - Ensure the `launchqt` file is executable:
      ```
      > ls -l /sbin/launchqt
      -rwxr-xr-x    1 1000     1000          231 Oct 26  2011 /sbin/launchqt
      ```
    - Ensure that the start-up file has been setup to correctly source `qt_env.sh` and launch `launchqt`:
      ```
      > cat /etc/init.d/rcS
      ```
- Check that the environment variables defined in `qt_env.sh` are defined on the target:
  ```
  > printenv
  ```
    - If they are not defined, ensure you edited your `/etc/profile` file correctly:
      ```
      > cat /etc/profile
      ```
- Ensure that the `tslib` libraries are on the target:
  ```
  > ls /lib/libts*
  ```
    - You should see at least `libts-1.0.so.0` (and some other files).
- Ensure that the `tslib` extra modules are on the target:
  ```
  > ls /lib/ts
  ```
    - You should see about 30 files (some .la, some .so).
- Ensure that you have a `ts.conf` file:
  ```
  > cat /etc/ts.conf
  ```
    - It may include many commented out lines (with #) but should include at least:
      ```
      module_raw input
      module pthres pmin=1
      module variance delta=30
      module dejitter delta=100
      module linear
      ```
- If the touch-screen needs to be re-calibrated, run:
  ```
  > /sbin/ts_calibrate
  ```

3. Cross-Compiling the Qt Framework

1. Edit GCC's files:
   - Someone (the manufacturer of the board?) edited the GCC files in the package we received. Their changes don't work for Qt, so here we'll reverse a couple changes.
   - Open the following file: `~/cmpt433/private/4.3.3/arm-none-linux-gnueabi/libc/usr/include/unistd.h`
   - Uncomment line 238, "`typedef __intptr_t intptr_t;`"
   - Uncomment line 996, "`extern void *sbrk....`"
2. Change Qt build-configuration file:
   - Edit the file `~/cmpt433/private/qt-everywhere-opensource-src-4.7.3/mkspecs/qws/linux-arm-g++/qmake.conf`
   - Add the following lines before the "`load(qt_config)`" line. Note: Lines ending with \ are to continue on the next line.

   ```
   QMAKE_CFLAGS_RELEASE    += -O0 -msoft-float \
               -D__GCC_FLOAT_NOT_NEEDED \
               -march=armv4t -mtune=arm920t -mcpu=arm920t
   QMAKE_CXXFLAGS_RELEASE  += -O0 -msoft-float \
               -D__GCC_FLOAT_NOT_NEEDED \
               -march=armv4t -mtune=arm920t -mcpu=arm920t

   # Ensure it's Optimization level 0:
   QMAKE_CFLAGS_RELEASE    -= -O2
   QMAKE_CXXFLAGS_RELEASE  -= -O2

   # Getting Touch Screen (tslib) working:
   QMAKE_INCDIR += /home/YOURID/cmpt433/private/tslib/include
   QMAKE_LIBDIR += /home/YOURID/cmpt433/private/tslib/lib
   ```

   - Note that the `QMAKE_INCDIR` and `QMAKE_LIBDIR` must both be full paths; you can't use "~" or $HOME. Change YOURID to be your real user ID.
3. Copy `tslib` files into Qt (one line per command):
   ```
   > cp -r ~/cmpt433/private/tslib/lib/* \
       ~/cmpt433/private/qt-everywhere-opensource-src-4.7.3/lib/
   > cp -r ~/cmpt433/private/tslib/include/* \
       ~/cmpt433/private/qt-everywhere-opensource-src-4.7.3/include/
   ```
4. Configure Qt:
   - Change to Qt directory:
   ```
   > cd ~/cmpt433/private/qt-everywhere-opensource-src-4.7.3/
   ```
   - Configure (command is all on one line):
   ```
   > ./configure -embedded arm -xplatform qws/linux-arm-g++          \
   -prefix ../qt-arm-4.7.3 -little-endian -no-webkit -no-qt3support  \
   -no-cups -no-largefile -optimized-qmake -no-openssl -nomake tools \
   -no-freetype -opensource -confirm-license -qt-mouse-tslib
   ```

- Will take a couple minutes to configure.

5. Build the Qt framework:
   ```
   > make -j2
   ```
   - If you have a single-core host, just use "`make`". Not recommended to use more than 2 cores as may cause build problems.
6. Install Qt (compiled for the ARM target) on the host:
   ```
   > make install
   ```
7. Copy `tslib` to your Qt install directory on the host:
   ```
   > cp -r ~/cmpt433/private/tslib/lib/* ~/cmpt433/private/qt-arm-4.7.3/lib/
   ```
8. Create a symbolic link to the cross-compiled `qmake`:
   ```
   > cd ~/cmpt433/private/4.3.3/bin
   > ln -s ../../qt-arm-4.7.3/bin/qmake arm-linux-qmake
   ```
   - Now you should be able to just type `arm-linux-qmake` anywhere and have it run the correct cross compiled binary because the `..../4.3.3/bin` directory is already in your path.
9. Install Qt libraries onto the target
   - Start by removing any existing Qt files from your RFS:
     ```
     > rm -rf ~/cmpt433/private/fs-qt4.7.3/opt/qtlibs
     ```
   - Copy the contents of the `~/cmpt433/private/qt-arm-4.7.3/lib` directory to your root file system:
     ```
     > cp -r ~/cmpt433/private/qt-arm-4.7.3/lib/ \
             ~/cmpt433/private/fs-qt4.7.3/opt/qtlibs
     ```
   - Delete some of the (so far) unused Qt libraries from `~/cmpt433/private/fs-qt4.7.3/opt/qtlibs`:
     ```
     > rm libQtDeclarative.*
     > rm libQtScript.*
     ```
   - Otherwise, your root-file system will be greater than 64Meg and won't fit in memory to download, or fit into flash on the board.
   - To save more space, delete the folder .../`fs-qt4.7.3/root/Documents`
     ```
     > rm -rf ~/cmpt433/private/fs-qt4.7.3/root/Documents
     ```
10. Trouble shooting:
    - Problem configuring?
      - Make sure `tslib` is compiled and installed correctly.
      - See what is going on by turning on all warnings by adding a -v to the `./configure` command's parameter list. This should help you identify what is going wrong.
    - Building get a "`undefined reference to "ts_read_raw"..."`
      - Ensure you have copied the `tslib` files into `~/cmpt433/private/tslib/lib/`:
        ```
        > cp -r ~/cmpt433/private/tslib/lib/* \
                ~/cmpt433/private/qt-arm-4.7.3/lib/
        ```
    - While downloading the root file system image using UBoot, it may reboot before the

download ("####...") is complete.

- Check the size of the file system image you are trying to download. It should be less than about 60 megs.
- If it's too big, delete unused Qt libraries or fonts from your root-file system on the host.

4. Qt Framework Native Host Installation

This is unnecessary in the SFU Linux lab. Do not execute any `sudo` commands in the lab.

1. Change to Qt's source directory
   ```
   > cd ~/cmpt433/private/qt-everywhere-opensource-src-4.7.3
   ```
2. Clean the build to remove previous build files:
   ```
   > make confclean
   ```
3. Configure Qt for the host:
   ```
   > ./configure
   ```
   - Select open source licence 'o'
   - Select 'Yes'
4. Build Qt for the host:
   ```
   > make -j2
   ```
5. Install Qt on the host:
   ```
   > sudo ./make install
   ```
6. Set paths (add to `~/.profile`):
   ```
   PATH="/usr/local/Trolltech/Qt-4.7.3/bin:$PATH"
   export PATH
   ```
7. Log-out and re-log
8. Test that Qt is installed and in the path by running:
   ```
   > qmake -v
   ```
9. Trouble shooting:
   - If you get the error: "`Basic XLib funcionality test failed`"
     ```
     > sudo apt-get build-dep qt4-qmake
     ```
   - See help for cross-compiling Qt for more trouble-shooting ideas.

5. Cross-Compiling Qt Application

1. Create a directory for your Qt applications:
   ```
   > mkdir ~/cmpt433/private/myApps/qt
   ```
2. Create a directory for a "Hello Qt World" app:
   ```
   > mkdir ~/cmpt433/private/myApps/qt/qthello
   ```
3. Copy the following code into a file `qthello.cpp` in the above directory:

   <REVISIT:- CODE HERE>
4. Run "`qmake -project`" to generate the Qt project file (`qthello.pro`)
   ```
   > cd ~/cmpt433/private/myApps/qt/qthello
   > arm-linux-qmake -project
   ```
   - Full path to ARM's qmake: `~/cmpt433/private/qt-arm-4.7.3/bin/qmake -project`
5. Create the Makefile:
   ```
   > arm-linux-qmake -o Makefile.arm
   ```
6. Make the hello Qt world application:
   ```
   > make -f Makefile.arm
   ```
   - Your executable, `qthello`, should now be built.
7. Copy the executable to the public directory:
   ```
   > cp qthello ~/cmpt433/public/
   ```
8. Run the executable on the **target**:
   ```
   > cd /mnt/remote
   > ./qthello -qws
   ```
   - You should see a (quite small) application. Close it using the touch-screen an the X button.
9. When your application exits, it may say it caused a "Segmentation fault" or "Illegal instruction". It is unknown what causes this, but it *seems* to be OK with these errors.
10. Trouble shooting:
    - While building, if you get the error "`undefined reference to `ts_read_raw'`" then you need to copy the `tslib/lib` contents into the `/qt-arm-4.7.3/lib` folder.
    - If your Qt application runs, but the touch-screen won't work then:
      - Ensure the newly built Qt libraries are copied to the RFS and downloaded to the board.
      - Ensure that the `tslib/lib` files are correctly installed in the root file system.
      - Double check the tslib section and ensure that the Qt runtime environment variables are correctly setup. Check on the target using:
        ```
        > printenv
        ```
    - If your Qt application displays the following:
      ```
      QWSSocket::connectToLocalFile could not connect:: No such file or
      directory
      ```
      - Run your application with the `-qws` option.

6. Native Compiling Qt Application

1. Change to the directory of your Qt application's source code.
2. Generate the makefile, this time creating the X86 (native host) one:
   ```
   > qmake -o Makefile.x86
   ```
3. Clean the current build:
   ```
   > make -f Makefile.x86 clean
   ```
4. Compile:
   ```
   > make -f Makefile.x86
   ```
5. Run application:
   ```
   > ./qthello
   ```
6. To switch between build targets (build native for host, or cross-compile for target), use:
   - HOST:
     ```
     > make -f Makefile.x86 -B
     ```
   - TARGET (cross compile)
     ```
     > make -f Makefile.arm -B
     ```
7. When you add more files to the project, you'll have:
   - Regenerate/edit the .pro file:
     ```
     > qmake -project
     ```
   - Regenerate each makefile:
     ```
     > qmake -o Makefile.x86
     > arm-linux-qmake -o Makefile.arm
     ```
   - Rebuild the application.
8. To check which platform a binary was compiled for, use:
   ```
   > readelf qthello -h
   ```
   - Look for the line "Machine:"
     - Built for host (Intel x86):
       ```
       Machine:                           Intel 80386
       ```
     - Built for target (ARM):
       ```
       Machine:                           ARM
       ```