Using CAEN digitizers under Linux (rev. 4)

Cester Davide, 2012

1. Installing Ubuntu

Download and install Ubuntu. This guide has been tested with versions 11.04 and 11.10.

2. Installing CAEN drivers and libraries

Download from CAEN website:

- CAENDigitizer library
- CAENComm library (should be inside CAENDigitizer package as requirement)
- CAENVMELib library (should be inside CAENDigitizer package as requirement)
- the driver for the interface (PCIe, USB...)

The libraries have their own script to launch as root (`sudo ./install_x64`); the driver must be compiled and loaded in the kernel at startup:

1. Compile the kernel module (`./configure; make`)
2. Copy the .ko file (e.g. `a3818.ko`) somewhere (like `/lib/modules/` or subdirs)
3. Copy the loader script (e.g. `a3818_load`) in /`etc/init.d`; edit it so that the variable `script_path` points to the directory you choose at step 2
4. move inside `/etc/init.d` and launch:
   ```
   sudo update-rc.d a3818_load defaults
   ```
   this will tell the system to load the module at startup

Warning: Ubuntu might automatically try to update the kernel; this will prevent the module to load properly. You have three possibilities:

- never upload kernel
- upload kernel, but select the old one in the boot menu at startup
- repeat steps 1..4 every time you update the kernel

A smart solution would be to copy the *entire* source code folder inside `/lib/modules/caen` and then using that path in the loader script: in this way, only step 1 will be required after

kernel updates (you will probably need to have root privileges to compile in that folder).

3. Installing CAEN Software

3.1. CAENUpgrader

The program is provided in the Linux version. Simply extract the archives, `configure` the source, `make` and `sudo make install`. Then, as root, copy the `CAENUpgraderGUI` executable in `/usr/bin` so it will be available from every directory and for every user.

3.2. CAEN .NET Demo

This program is MS Windows-dependant; we must use Wine. Install Wine from the repositories, then download from CAEN website:

- the Windows version of the driver you are using
- CAENVMELib library
- CAEN VME Demo installer

in addiction, you must use winetricks to install .NET platform inside Wine:

```
$ wget http://winetricks.org/winetricks
$ chmod u+x winetricks
$ winetricks dotnet30
```

now you can install the Windows executables inside Wine:

1. right click, Properties / Permissions / Allow executing file as program
2. right click, Open with Wine Windows Program Loader

You will end up having a Wine folder under Applications menu (top-left corner of the screen): the .NET Demo executable is in:

Applications / Wine / Programs / CAEN / VME / Demos / CAENVMEDemoDotNet

4. Installing Qt libraries

Install the package `libqt4-dev` (or `qt4-designer` if you want more tools to play with).

5. Installing ROOT

Download ROOT from `root.cern.ch` website; choose version 5.32 that fixed some minor bugs in Qt-ROOT integration.

You will also need the following packages:

```
make, gcc, gcc-c++ or g++, binutils, dpkg-dev, libX11-dev,
libXpm-dev, libXft-dev, libXext-dev
```

(list taken from http://root.cern.ch/drupal/content/build-prerequisites).

Now you can extract the ROOT archive and launch `./configure`; add `--enable-qt` if you are going to embed ROOT in a GUI made with Qt. Then launch `make` to compile.

Before you actually can *install* ROOT you have to set up system variables; open hidden file `.bashrc` in your user home directory and add the following lines at the end:

```
# User specific environment and startup programs
export ROOTSYS=/usr/local/root
export LD_LIBRARY_PATH=/usr/lib:$ROOTSYS/lib:$LD_LIBRARY_PATH
export PATH=$PATH:.:$HOME/bin:$ROOTSYS/bin
export CPLUS_INCLUDE_PATH=include:/usr/local/root/include
```

then go as *root* user, set ROOT environment variable, install and switch back to your user:

```
$ sudo su
$ export ROOTSYS=/usr/local/root
$ make install
$ exit
```

Now close and reopen all your terminals.

6. Compiling programs

Compiling a program with Qt and ROOT can be quite tricky if you are not familiar with makefiles and all the libraries. Here there are some notes that can be of help; of course they are not intended to replace official documentation (check Google, `http://root.cern.ch` and `http://developer.qt.nokia.com/`).

6.1. A remainder on compiling commands

Standard c++ compiling under Linux works in this way:

```
c++ other1.cpp other2.cpp main.cpp -o execname -lnameoflib1 ...
```

- `c++` is the name of the compiler
- `.cpp` files are source code files; you must list all files called by `main.cpp` or by another `.cpp` listed file
- `-o execname` tells the compiler how to call the output program file
- `-lnameoflib1` is a library that must be linked to the executable (Qt or ROOT libraries, in our case). Please notice the syntax: if we want to link the libraries contained in files `libRoot1.so` and `libQt2.so` we must write

  ```
  -lRoot1 -lQt2
  ```

ROOT provides an utility (`root-config`) for automatic listing of all available ROOT libraries:

```
$ root-config --glibs
```

```
-L/usr/local/root/lib -lGui -lCore -lCint -lRIO -lNet -lHist -lGraf
-lGraf3d -lGpad -lTree -lRint -lPostscript -lMatrix -lPhysics
-lMathCore -lThread -pthread -lm -ldl -rdynamic
```

so, in the compiling command line, using

```
c++ otherfile.cpp main.cpp -o executablename `root-config --glibs`
```

is equivalent to

```
c++ otherfile.cpp main.cpp -o executablename -L/usr/local/root/lib
-lGui -lCore -lCint -lRIO -lNet -lHist -lGraf -lGraf3d -lGpad -lTree
-lRint -lPostscript -lMatrix -lPhysics -lMathCore -lThread -pthread
-lm -ldl -rdynamic
```

There also can be some “compiler definition”, starting with `-D`; for example, adding `-DLINUX` in the compiling command will cause `#ifdef LINUX` inside the `.cpp` code to be evaluated as TRUE.

6.2. Qt and the MOC files

The use of Qt libraries complicates the situation: it is not only a matter of long listing of files and libraries, but some files need to be somehow “precompiled” in order to correctly handle all interaction functionalities between Qt objects. A Meta-Object Compiler (moc) is needed to produce one more file for each source file that uses Qt objects; these additional files must be included in the final compiling list of sources, and the procedure gets very annoying to be carried on manually. Of course, one could write his own shell script with all the required operations; a standard technique for writing such a script is using makefiles.

6.3. Makefiles

Makefiles are a complex way to manage complex compiling. A typical makefile contains a list of the libraries, of the compiling parameters, of the source files and their mutual dependencies, and something else; the make program will take care of putting them together, no matter how many they are. Here we will see how to create a working makefile for our Qt/ROOT project.

1. Organize the source code as you wish; e.g. with separate folders for headers and code
2. launch the command `qmake -project`: this will produce the file `foldername.pro` containing the listing of all the source files that qmake can find in the current directory and in subdirs. Note that unused or old .cpp files will be included in the list if only they are found in some folder
3. launch the command `qmake foldername.pro`: this will generate a valid Makefile
4. now it's time to manually edit the Makefile, because qmake does not know that we need other libraries, such as CAEN or ROOT ones. Change the lines:

```
DEFINES       = -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED
...
LIBS          = $(SUBLIBS)  -L/usr/lib -lQtGui -lQtCore -lpthread
```

with the following:

```
DEFINES       = -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED
-DQT -DLINUX
...
LIBS          = $(SUBLIBS) -L/usr/lib -L/usr/lib64 -lQtGui -lQtCore
`root-config --glibs` -lGQt -lpthread -lCAENDigitizer
```

now everything should compile well with simply `make`.

Note: `-DQT` and `-DLINUX` are only an example. Check your existing source code, starting from header files, to find out what defines you actualy need.

If your project gets bigger and you create more source files, you have to add them in the Makefile; the easiest way is to repeat steps 1..4. Notice that you have to manually correct the Makefile for additional defines and libraries each time you generate a new one.

By default, qmake-generated Makefiles place all the `moc_` files and the `.o` files in the main folder of your project. You can manually alter many of the Makefiles lines specify different folder for these two families of files, but you will lose all the changes if you recreate the Makefile; it is recommended to spend this time only when your project is stable enough, and then always keep a backup of your custom Makefile.

6.4. make command tips & tricks

- `make -f Makefile`: specify a custom Makefile
- `make clean`: delete all objects and compiled files from a precedent compiling (useful after changes or aborted compiling)
- `make -j2`: use two processor cores for compiling (faster)