LIBQWQNG Version 1.4

ComScire QNG Device Linux Driver

Table of Contents

1. General Information

LIBQWQNG-1.4 – a library which allows access to ComScire QNG devices. LIBQWQNG requires the open source library LIBFTDI1 installed. In addition, LIBUSB 1.0 API is needed by LIBFTDI1 to access FTDI devices.

***** WARNING ***** LIBUSB by design allows other applications to assume control of any USB device including QNG devices. It is recommended that only one QNG device can be attached to any computer. This is a possible security issue of Linux that is out of our control.

The official web site is:
https://comscire.com

Contact at:
<contact@comscire.com>

2. Installation

ComScire QNG devices are accessed through LIBFTDI1 library. This library can be installed using different methods such as: using the software manager of the distribution, using the supplied LIBFTDI1 source package found in the unpacked install root directory, download source packages from LIBFTDI1 web site, or download using git repository.

We provide BASH scripts (located in the installation source package) to compile and install LIBQWQNG and all of its prerequisites on Ubuntu system. For other Linux Distributions, manually install by following the instructions in Steps 3 to 6. Entire Source code for manual installation is found in the installation source package.

***Prior to installation, your environment will need to be setup in order to compile the necessary libraries. GNU C, C++ and CMake compilers are needed (Optional: Doxygen package if you want API documentations generated; git package if you want to download LIBFTDI through git repository).

Installing from Script:

2.1. Make sure the archive packages and the scripts are in the same root directory.

2.2. Copy the installation package to your hard drive and unpack.

Change directory to install directory.

```
$ cd "../path-to-dir"
```

2.3. Give execution permissions to the script for your OS/Arch

```
$ sudo chmod +x ubuntu-x64.sh
```

2.4. Run the script with admin privileges

```
$sudo ./ubuntu-x64.sh
```

Manual installation:

3. Build the LIBUSB-1.0

3.1. Download LIBUSB-1.0 and LIBUSB-1.0-dev packages from distribution's package manager. LIBUSB-1.0-dev package contains the header files and documentation needed to develop applications using LIBQWQNG library. If packages not available through software manager use the provided library found in the installation package root directory or download from the third-party library’s website. See Section 3.2 for manual installation.

a) UBUNTU:
Download from repository using terminal:

```
# sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev
```

b) Other Linux Distributions:
We provide a copy of libusb in the installation package. Go to Section 3.2

3.2. Unpack the supplied libusb source package or download the latest source archive package version from address:

http://sourceforge.net/projects/libusb/files/libusb-1.0/

3.2.1. Unpack the archive package. Open terminal, build source as root.

```
$ cd ~/libusb-1.0.x/
$ ./configure
$ make
# make install
```

3.3. Make symbolic link of *libusb.h* header file as root.

```
# ln -s /usr/local/include/libusb-1.0/libusb.h /usr/local/include/libusb.h
```

4. Build the LIBFTDI1

4.1. If not available on distribution's software management list, use the supplied LIBFTDI1 source package or download the latest source package version from address:

http://www.intra2net.com/en/developer/libftdi/download.php

Unpack archive package. In terminal, go to directory of unpacked archive then proceed to Section 4.3 for installation instruction.

4.2. LIBFTDI1 requires CMake package for compilation. Installing CMake and other optional packages used by LIBFTDI-1.0 are found in Appendix A. Optional packages are not needed for basic functions of LIBQWQNG-1.4 API.

4.3. Go to new directory libftdi-x (substitute x with version).

```
$ cd ~/libftdi1-x/
```

4.3.1. Build and install source as root

```
$ cmake ~/libftdi-x/
$ make
# make install
```

4.4. Make symbolic link of *ftdi.h* header file as root.

```
# ln -s /usr/local/include/libftdi1/ftdi.h /usr/local/include/ftdi.h
```

5. Build the LIBQWQNG-1.4

5.1. Compiling LIBQWQNG-1.4 requires CMake. LIBFTDI and LIBUSB libraries must be installed successfully prior to compiling LIBQWQNG-1.4.

Optional: Doxygen package is needed to generate LIBQWQNG API documentation. LIBQWQNG API Reference is provided in this document. See Section 8 for API reference if you wish not to install Doxygen and generate API documentation.

5.2. Unpack the LIBQWQNG-1.4 source package and go to new directory. Open terminal, build source and install as root.

```
$ cd ~/libqwqng-1.4/
$ cmake ~/libqwqng-1.4/
$ make
# make install
```

6. QNG Device's Permission

6.1. UDEV rules must be modified to allow users to access ComScire's QNG devices. An UDEV .rules file (45-libqwqng.rules) is supplied in *~/libqwqng-1.4/packages/* directory. These rules set read/write permissions and assign QNG devices to the *plugdev* group. Users must be members of *plugdev* group to access the QNG devices.

Installation of LIBQWQNG-1.4 library attempts to install these rules into the */etc/udev/rules.d/* directory. If copying of rules fail, manual installation is necessary (see Section 6.2).

6.1.1. If install successful, restart UDEV as root (unplug QNG device if connected) or reboot computer if *udevadm* command is not available in your distribution.

```
# udevadm control --reload-rules
```

6.2. Manual Installation: As root, copy *45-libqwqng.rules* into */etc/udev/rules.d/* directory.

```
# cp ~/libqwqng-1.4/packages/45-libqwqng.rules /etc/udev/rules.d/
```

6.2.1. UDEV must be restarted (unplug QNG device if connected). Note: Must restart OS if *udevadm* command is not available in your distribution.

```
# udevadm control --reload-rules
```

6.3. Adding *plugdev* group

6.3.1. First check if user is in *plugdev* group.

```
$ groups
```

6.3.2. If your user is not in the group *plugdev*, check if your system already has the group *plugdev*:

```
$ grep plugdev /etc/group
```

If the previous command displays a line beginning with:

```
plugdev:x:
```

then your system has the group *plugdev*.

6.3.3. When the *grep* command does not display any message, then the *plugdev* group doesn't exist on your system. as root create the *plugdev* group:

```
# groupadd plugdev
```

6.3.4. As root add the user *USER* to the group *plugdev* (substitute your own login name for USER:

```
# usermod -G plugdev -a USER
```

Reboot.

6.4. CentOS --

There is a solution for the UDEV permissions in CentOS if supplied rules do not work. Instead of getting your machine to recognize the *45-libqwqng.rules*, edit file */etc/udev/rules.d/50-udev.rules* directly.

Comment out line 343 (may be different in your file):

CODE--

```
ACTION=="add", SUBSYSTEM=="usb_device", \
PROGRAM=... , \
# NAME="%c", MODE="0644"
```

and add line:

```
NAME="%c", MODE="0666"
```

Reboot.

7. Test Installation

7.1. Connect device and run QNGmeter application found in *~/libqwqng-1.4/qngMeter* directory.

```
$ ./qngMeter/qngmeter
```

7.2. Connect device and run test applications found in *~/libqwqng-1.4/examples* directory.

```
$ ./examples/clear
$ ./examples/deviceid
$ ./examples/diagnostics
$ ./examples/errorhandl
$ ./examples/randbytes
$ ./examples/randint32
$ ./examples/randnormal
$ ./examples/randuniform
$ ./examples/reset
$ ./examples/runtimeinfo
```

8. QNGmeter

The ComScire QNGmeter is a continuous statistical tester that uses five powerful and fundamentally different tests on the input data. All tests are designed to provide reliable results for up to 100 terabits of test data. Some tests (OQSO, Entropy, and Serial) have approximate test result distributions and the test results will become unreliable for extremely large test data quantities.

For more information, see QNGmeterDoc.html file in *~/libqwqng-1.4/* directory.

9. API Reference

8.1. RandInt32 --

int RandInt32(long* pVal)

Returns

Random 32 bit integer as LONG.

Remarks

RandInt32 property returns a 32 bit random integer. Each RandInt32 integer contains 32 bits of entropy.

8.2. RandUniform --

int RandUniform(double* pVal)

Returns

Random uniform number [0,1) as DOUBLE.

Remarks

RandUniform property returns a double float that is randomly selected from a uniform distribution. Each RandUniform number contains 48 bits of entropy.

8.3. RandNormal --

int RandNormal(double *pVal)

Returns

Random normal number with mean zero and standard deviation one as DOUBLE.

Remarks

RandNormal property returns a double float that is randomly selected from a normal distribution. RandNormal numbers are produced by transforming uniform numbers, with 48 bits of entropy each, into normal numbers using the Box-Muller method.

8.4. RandBytes --
int RandBytes(char* pVal, long length)

Parameters
Length: Number of bytes to be returned from RandBytes. Must not exceed 8192 for generator output rate of 32 Mbps or less, 65536 for generator output rate of 64 or 128 Mbps.

Returns Byte array as 8 bit character.

Remarks
RandBytes property returns a byte array of random bytes. Each byte contains 8 bits of entropy. If Length exceeds maximum allowed requested bytes, the control will return the QNG_E_IO_ARRAY_OVERSIZED error code.

8.5. Clear --
int Clear()

Remarks
Clear property purges internal data buffers. If random data is not continuously consumed, random data will remain available in internal buffers. A call to Clear will remove "stale" data from the buffers.

8.6. Reset --
int Reset()

Remarks
Reset property closes the active hardware device, clears all buffers, then attempts to restart the hardware device.

8.7. DeviceID --
char* DeviceID()

Returns
Serial number as a sequence of characters.

Remarks
DeviceID property returns the device serial number as a sequence of characters. The serial number is typically 8 ASCII characters long.

8.8. RuntimeInfo --
int RuntimeInfo(float* pVal)

Returns
Float array as 32 bit floating point numbers.

Remarks
RuntimeInfo property returns an array of 17 floating point numbers. The numbers indicate the internal runtime state. Assuming a zero index based array:

runtimeInfo[0]: General statistical status. A zero (0) indicates that all internal statistics are within expected ranges and a minus one (-1) indicates an exception.
runtimeInfo[1]: Entropy H(P) of final output channel.
runtimeInfo[2]: Predictability value (P) of final output channel.
runtimeInfo[3]: Bias of final output channel.
runtimeInfo[4]: 1st order serial correlation of final output channel.
runtimeInfo[5]: Entropy H(P) of 1st raw generator channel.
runtimeInfo[6]: Predictability value (P) of 1st raw generator channel.
runtimeInfo[7]: Bias of 1st raw generator channel.
runtimeInfo[8]: 1st order serial correlation of 1st raw generator channel.
runtimeInfo[9]: Entropy H(P) of 2nd raw generator channel.
runtimeInfo[10]: Predictability value (P) of 2nd raw generator channel.
runtimeInfo[11]: Bias of 2nd raw generator channel.
runtimeInfo[12]: 1st order serial correlation of 2nd raw generator channel.
runtimeInfo[13]: Entropy H(P) of 3rd raw generator channel.
runtimeInfo[14]: Predictability value (P) of 3rd raw generator channel.
runtimeInfo[15]: Bias of 3rd raw generator channel.
runtimeInfo[16]: 1st order serial correlation of 3rd raw generator channel.

8.9. Diagnostics --
int Diagnostics(char dxCode, char* dxInfo)

Parameters
Length: Fixed 128 bytes to be returned from Diagnostics.

Returns
Byte array as 8 bit character.

Remarks
Diagnostics is a property that returns a byte array of 128 bytes for a specified internal pre-output raw data channel on a specific level of processing. Diagnostics provides insight into three entropy combining levels of processing in *Pure Quantum*® (PQ) and the final raw output stream in *CryptoStrong*™

(CS) generators. Each successive level, beginning with level 1, combines more entropy per bit towards the final output. The final output itself exceeds NIST defined *full entropy* without utilizing correction, whitening, or conditioning. Note that low-level pre-output data is not expected to look perfectly random. Therefore, data gathered from Diagnostics should never be used in random data consuming applications. The intention of Diagnostics is to allow measurements into the pre-output generation levels to follow and confirm theoretical generation models. More information on *PureQuantum*® and *CryptoStrong*™ generation internals can be found in whitepapers published on the ComScire website. Diagnostics allows access to three levels of pre-output generation with three channels per level, and the final combined output for *CryptoStrong*™ generators. The following lists corresponding hex Diagnostics codes (dxCode) for each level and channel:

Level 1, Channel 1:	0x10
Level 1, Channel 2:	0x11
Level 1, Channel 3:	0x12
Level 2, Channel 1:	0x13
Level 2, Channel 2:	0x14
Level 2, Channel 3:	0x15
Level 3, Channel 1:	0x16
Level 3, Channel 2:	0x17
Level 3, Channel 3:	0x18
Final Output (CS Model only):	0x19

Note that there may be other undocumented Diagnostics codes (dxCode). However, using these codes may produce unexpected outputs or results. Nonetheless, random data obtained through the random API calls (RandInt32, RandUniform, RandNormal, RandBytes) is entirely unaffected by Diagnostics calls. Diagnostics is not supported on devices prior to the *PureQuantum*® (PQ) and *CryptoStrong*™ (CS) models. Final output stream (dxCode: 0x19) is only available in *CryptoStrong*™ models.

8.10. Error Handling --

The LIBQWQNG library returns a set of error codes. All error conditions will persist and can be cleared by a succesful Reset method call.

Codes

QNG_S_OK 00044400h

QNG device reports success.

QNG_E_GENERAL_FAILURE	80044401h	QNG general error.
QNG_E_IO_ERROR	80044402h	QNG I/O error.
QNG_E_IO_TIMEOUT	80044403h	QNG I/O request has timed out.
QNG_E_IO_ARRAY_OVERSIZED	80044404h	QNG read array size exceeds max size.
QNG_E_STATS_EXCEPTION	80044406h	QNG test statistics exception.
QNG_E_STATS_UNSUPPORTED	80044407h	QNG stats not supported with this device.
QNG_E_DIAGX_UNSUPPORTED	80044408h	QNG diagnostics not supported with this device.
QNG_E_DEVICE_NOT_OPENED	8004440Ah	QNG device not found or already in use.
S_OK	00000000h	No error occurred.

9. Appendix A

9.1. Cmake Install

LIBFTDI and LIBQWQNG require CMake package for compilation.

a) UBUNTU:
Download from repository using terminal:

```
# sudo apt-get install cmake
```

b) CentOS:
Download from repository using terminal:

```
# yum install cmake
```

c) If not available, download the latest release at:

http://www.cmake.org/cmake/resources/software.html

manually build and install.

9.2. Doxygen Install –

Doxygen package is needed to generate LIBFTDI and LIBQWQNG API documentations. LIBQWQNG API Reference is provided in Section 8. If Doxygen is not installed, disregard warning during compilation. If interested in Doxygen, see distribution's software manager for installation.

a) UBUNTU:
Download from repository using terminal:

```
# sudo apt-get install doxygen
```

b) CentOS:
Download from repository using terminal:

```
# yum install doxygen
```

c) If not available on distribution's software management list, download Doxygen at:

http://www.stack.nl/~dimitri/doxygen/download.html#latestsrc

manually build and install.