Quantis User Guide

Version 2.9

Quantis User Guide

Version 2.9

Information in this document is subject to change without notice.

Revision History

Revision 2.9 03.09.2012

- Add QuantisExtension library documentation
- Add Quantis library new Open/Close functions.
- Add EasyQuantis extraction operations.
- Add how to recompile Quantis libraries
- Added an entry in the FAQ.

Revision 2.8 19.12.2011

- Added information concerning the C++11 user interface and installation under FreeBSD and Solaris.
- Corrected bug in the usage sample and provided detailed info about the sample code compilation and execution.

Revision 2.7 16.05.2011

- Added information related to the port of Quantis USB on Mac OS X.

Revision 2.6 10.03.2011

- Corrected minor mistakes and rephrased a few sentences for ease of understanding.

Revision 2.5 12.01.2011

- Added details on the QuantisGetManufacturer method.

Revision 2.4 16.09.2010

- Added instructions to install Quantis on Red Hat Enterprise Linux and CentOS.

Revision 2.3 25.06.2010

- Added details on the scaling algorithms.
- Improved the EasyQuantis installation description on Linux.
- Added Troubleshooting appendix.

Revision 2.2	30.04.2010
• In the Quantis PCI Linux driver installation section: fixed a wrong path and added two sub-sections. • Updated the EasyQuantis installation procedure description under Linux.	
Revision 2.1	26.04.2010
• Added an EasyQuantis command line section. • Added answers in the FAQ.	
Revision 2.0	09.04.2010
• Initial version.	

Table of Contents

1. Introduction 1
    1.1. What You Need 1
        1.1.1. Additional Requirements 2
2. Hardware Installation 3
    2.1. Quantis PCI and PCI Express Installation 3
        2.1.1. Unpacking 3
        2.1.2. Installing the Card 3
    2.2. Quantis USB Installation 4
        2.2.1. Unpacking 4
        2.2.2. Installing the Device 4
3. Driver Installation 5
    3.1. Windows Operating Systems 5
        3.1.1. Windows XP 5
        3.1.2. Windows Vista 9
        3.1.3. Windows 7 13
    3.2. Linux Operating System 18
        3.2.1. Quantis PCI and Quantis PCI Express 18
        3.2.2. Quantis USB 23
    3.3. Mac OS X Operating System 26
        3.3.1. QuantisPCI and QuantisPCI Express 26
        3.3.2. Quantis USB 26
        3.3.3. Installation 26
        3.3.4. Implementation details 26
        3.3.5. Known problems 27
    3.4. Solaris / OpenSolaris 27
    3.5. FreeBSD 28
4. The EasyQuantis application 29
    4.1. Installation 29
        4.1.1. Windows Operating Systems 29
        4.1.2. Linux Operating Systems 29
        4.1.3. Mac OSX 31
        4.1.4. Solaris / OpenSolaris 31
        4.1.5. FreeBSD 31
    4.2. EasyQuantis 31
    4.3. EasyQuantis2 (Windows, Linux) 31
        4.3.1. Acquisition 31
        4.3.2. File extraction 33
        4.3.3. Extraction matrix 34
    4.4. Using EasyQuantis 1.x (other OS) 35
    4.5. The EasyQuantis Command Line 36
        4.5.1. Options 37
        4.5.2. Usage Examples 37
5. The Quantis Library 39
    5.1. Library location 39
    5.2. Device Type 39
    5.3. Basic Functions 40
        5.3.1. QuantisCount 40
        5.3.2. QuantisGetDriverVersion 40
        5.3.3. QuantisGetLibVersion 40
        5.3.4. QuantisGetManufacturer 40
        5.3.5. QuantisGetModulesDataRate 41
        5.3.6. QuantisGetSerialNumber 41
        5.3.7. QuantisRead 41
        5.3.8. QuantisOpen 45
        5.3.9. QuantisClose 45

5.3.10. QuantisReadHandled 45
5.3.11. QuantisStrError 46
5.4. Advanced Functions 46
5.4.1. QuantisBoardReset 46
5.4.2. QuantisGetBoardVersion 46
5.4.3. QuantisGetModulesCount 47
5.4.4. QuantisGetModulesMask 47
5.4.5. QuantisGetModulesPower 47
5.4.6. QuantisGetModulesStatus 48
5.4.7. QuantisModulesDisable 48
5.4.8. QuantisModulesEnable 48
5.4.9. QuantisModulesReset 49
5.5. Recompiling the Quantis Library 49
5.5.1. Windows compilation with Visual Studio 2008 49
5.5.2. Windows Compilation with Visual Studio 2010 51
5.5.3. Linux Debian based 53
5.5.4. Linux RedHat / CentOS 53
5.5.5. Mac OSX 54
5.5.6. Solaris / OpenSolaris 54
5.5.7. FreeBSD 55
6. Quantis Library Wrappers 57
6.1. The C++11 random_device interface 57
6.1.1. About the interface and our implementation 57
6.1.2. Library Compilation for C++11 58
6.1.3. C++11 Sample compilation 59
7. The QuantisExtensions Library 61
7.1. Extractor 61
7.1.1. Basic Functions 61
7.1.2. Advanced Functions 63
8. Sample Code 69
8.1. Windows Compilation / Execution 69
8.1.1. Visual Studio 2008 vs. 2010 69
8.2. Linux / Solaris / OpenSolaris / FreeBSD compilation and execution 69
8.2.1. Mac OSX 70
8.3. C Sample 70
8.4. C++ Sample 71
8.5. Java Sample 72
A. Troubleshooting 75
A.1. EasyQuantis 75
A.2. Quantis Samples 75
B. Frequently Asked Questions (FAQ) 77
B.1. Quantis Library 77
B.2. EasyQuantis 77
C. Migrating to the New API 79
C.1. Compatibility Wrapper 80
D. Notes 81
D.1. Images 81
Bibliography 83

List of Figures

4.1. EasyQuantis Setup Wizard welcome 29
4.2. EasyQuantis2 Acquisition tab 32
4.3. EasyQuantis2 File extraction tab 33
4.4. EasyQuantis2 Extraction matrix tab 34
4.5. EasyQuantis main window 35

List of Tables

1.1. Supported operating systems. 1
C.1. API 1.x and 2.0 functions equivalences. 79

Chapter 1. Introduction

Thank you for purchasing a Quantis Random Number Generator.

A random number generator is a device that produces sequences of numbers whose outcome is unpredictable and which cannot subsequently be reliably reproduced. There exist two main classes of random number generators: software and physical generators. In general, software generators produce so-called pseudo random numbers. Although they may be useful in some applications, they should not be used in most applications where true randomness is required.

Quantis is a physical random number generator exploiting an elementary quantum optics process. Photons - light particles - are sent one by one onto a semi-transparent mirror and detected. The exclusive events (reflection - transmission) are associated to "0" - "1" bit values. The operation of Quantis is continuously monitored. If a failure is detected, the random bit stream is immediately stopped.

Quantum random number generators have the advantage over conventional randomness sources of being invulnerable to environmental perturbations and of allowing live status verification.

1.1. What You Need

To use your Quantis, you need:

- A PC with a supported operating system installed (see Table 1.1, “Supported operating systems.”) and one of the following slots/ports available:
  - A PCI 32-bit slot (for Quantis PCI).
  - A PCI Express x1 slot (for Quantis PCI Express).
  - A USB 2.0 port (for Quantis USB).
- A USB 2.0 port for the USB Flash drive.
- 50MB hard drive space.

Operating System	Quantis PCI/PCIe	Quantis USB
Microsoft Windows XP (32-bit)	✓	✓
Microsoft Windows XP (64-bit)	✗	✗
Microsoft Windows Server 2003	✓	✓
Microsoft Windows Vista (32-bit and 64-bit)	✓	✓
Microsoft Windows Server 2008 (32-bit and 64-bit)	✓	✓
Microsoft Windows 7 (32-bit and 64-bit)	✓	✓
Linux 2.6/3 (32-bit and 64-bit)	✓	✓
Solaris / OpenSolaris	✓	✗
FreeBSD	✓	✓
Mac OS X	✗	✓

Table 1.1. Supported operating systems.

1.1.1. Additional Requirements

1.1.1.1. Linux

On Linux systems, you additionally need:

- Xorg 1.0 or higher (only required to use the EasyQuantis application).

Chapter 2. Hardware Installation

This chapter provides unpacking and installation information for Quantis.

2.1. Quantis PCI and PCI Express Installation

Caution

Under ordinary circumstances, the Quantis PCI and Quantis PCI Express (PCIe) cards will not be affected by static charge as may be received through your body during handling of the unit. However, there are special circumstances where you may carry an extraordinarily high static charge and possibly damage the card and/or your computer. To avoid any damage from static electricity, you should follow some precautions whenever you work on your computer.

1. Turn off your computer and unplug power supply.
2. Use a grounded wrist strap before handling computer components. If you don't have one, touch with both of your hands a safely grounded object or a metal object, such as the power supply case.
3. Place components on a grounded anti-static pad or on the bag that came with the components whenever the components are separated from the system.

The card contains sensitive electric components, which can be easily damaged by static electricity, so the card should be left in its original packing until it is installed.

Unpacking and installation should be done on a grounded anti-static mat. The operator should be wearing an anti-static wristband, grounded at the same point as the anti-static mat.

Inspect the card carton for obvious damage. Shipping and handling may cause damage to your card. Be sure there are no shipping and handling damages on the card before proceeding.

DO NOT POWER YOUR SYSTEM IF THE QUANTIS CARD IS DAMAGED.

2.1.1. Unpacking

Open the shipping carton and carefully remove all items, and ascertain that you have:

- a Quantis PCI or Quantis PCIe card;
- a Quick Install Guide;
- a USB Flash Drive with Manual, Drivers and Samples.

If any item is found to be missing or damaged, please contact your local reseller for replacement.

2.1.2. Installing the Card

1. Shut down the computer, unplug its power cord and remove the chassis cover.
2. Locate the PCI 32-bits slot (for Quantis PCI) or the PCI Express x1 slot (for Quantis PCIe). If necessary, remove the metal cover from this slot, then align your Quantis card with the PCI or PCIe slot respectively and press it in firmly until the card is fully inserted.

3. Install the bracket screw and secure the card to the computer chassis.
4. Cover the computer’s chassis.
5. Switch the computer power on.
6. Install the driver (see next Chapter).

2.2. Quantis USB Installation

2.2.1. Unpacking

Open the shipping carton and carefully remove all items, and ascertain that you have:

- a Quantis USB;
- a USB cable;
- a Quick Install Guide;
- a USB Flash Drive with Manual, Drivers and Samples.

If any item is found to be missing or damaged, please contact your local reseller for replacement.

2.2.2. Installing the Device

1. Connect the Quantis device to a USB 2.0 port on your PC using the cable that came with the Quantis device.
2. Install the driver (see next Chapter).

Chapter 3. Driver Installation

To be able to access your Quantis device, you need to install a driver. This chapter contains instructions on how to install the driver on your operating system.

Important

Quantis PCI Express is software-compatible with Quantis PCI. This means that any software capable of communicating with a Quantis PCI device (e.g driver) is also able to communicate with the Quantis PCI Express. More specifically:

- Quantis PCIe uses the Quantis PCI driver.
- Quantis PCIe is considered by the software (driver, application) as a Quantis PCI device.

3.1. Windows Operating Systems

This section contains instructions on how to install a Quantis device on Windows Operating Systems.

Insert the USB flash drive into an available USB port. This drive contains the Quantis drivers as well as software for your device.

Important

In this section, we assume that the letter of the USB flash drive provided by IDQ is drive `D:`. If this is different on your machine, substitute your corresponding drive name for `D:` in the appropriate places in this instruction.

3.1.1. Windows XP

When a Quantis RNG is inserted into your computer for the first time, the operating system will detect the device automatically and display a *New Hardware Found* message. The following are step-by-step installation instructions.

3.1.1.1. Found New Hardware Wizard: Welcome

Windows will search for a driver on your computer, on removable media (e.g. CD-ROM) and on the Windows Update Web site.

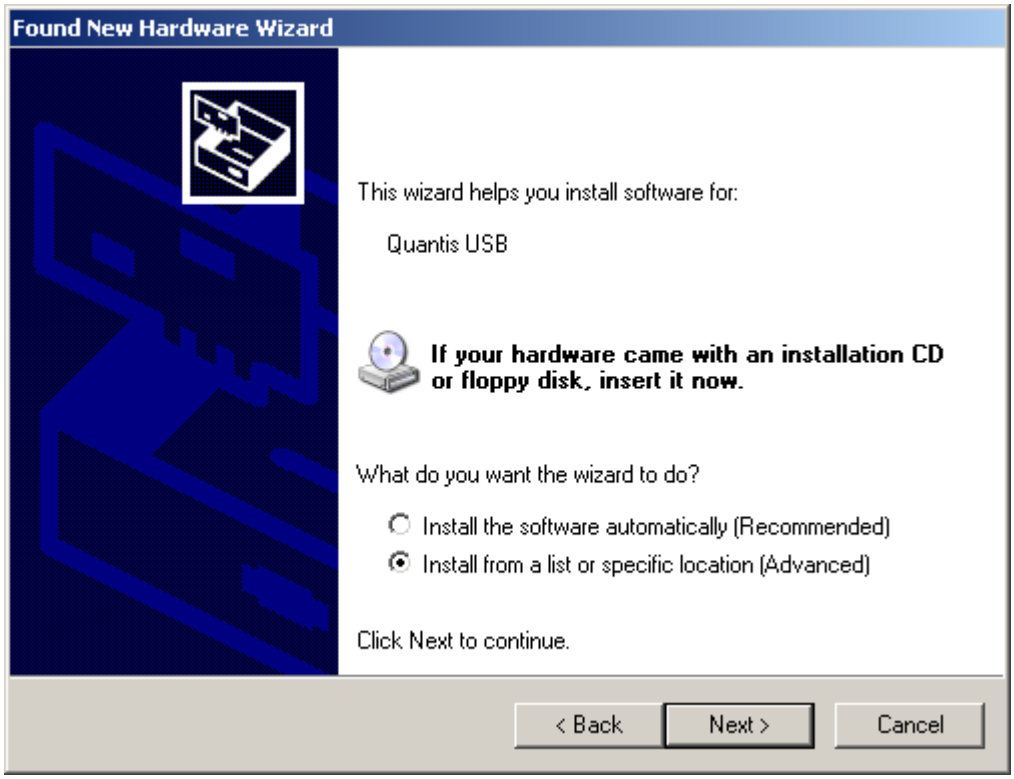
The Quantis driver is not available on the Windows Update Web site. If asked, deny access to the Windows Update Web site and click the **Next** button.

Note

It is harmless to allow the wizard to connect to the Windows Update Web site. The only effect is that the installation process will take a little longer.

3.1.1.2. Found New Hardware Wizard: Quantis

When the wizard asks you what to do to install Quantis software, select *Install from a list or specific location* and click the **Next** button.

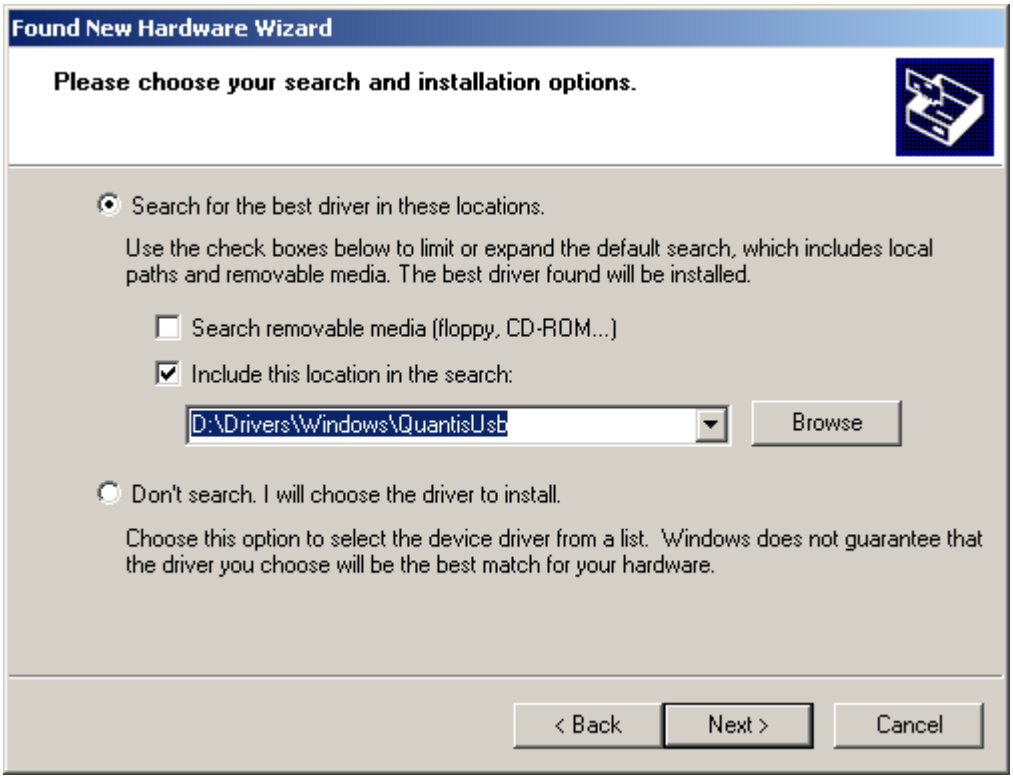


3.1.1.3. Found New Hardware Wizard: Search Location

First select *Search for the best driver in these locations*. Then activate the option *Include this location in the search*. Click the Browse button and select the directory containing the right driver for your device:

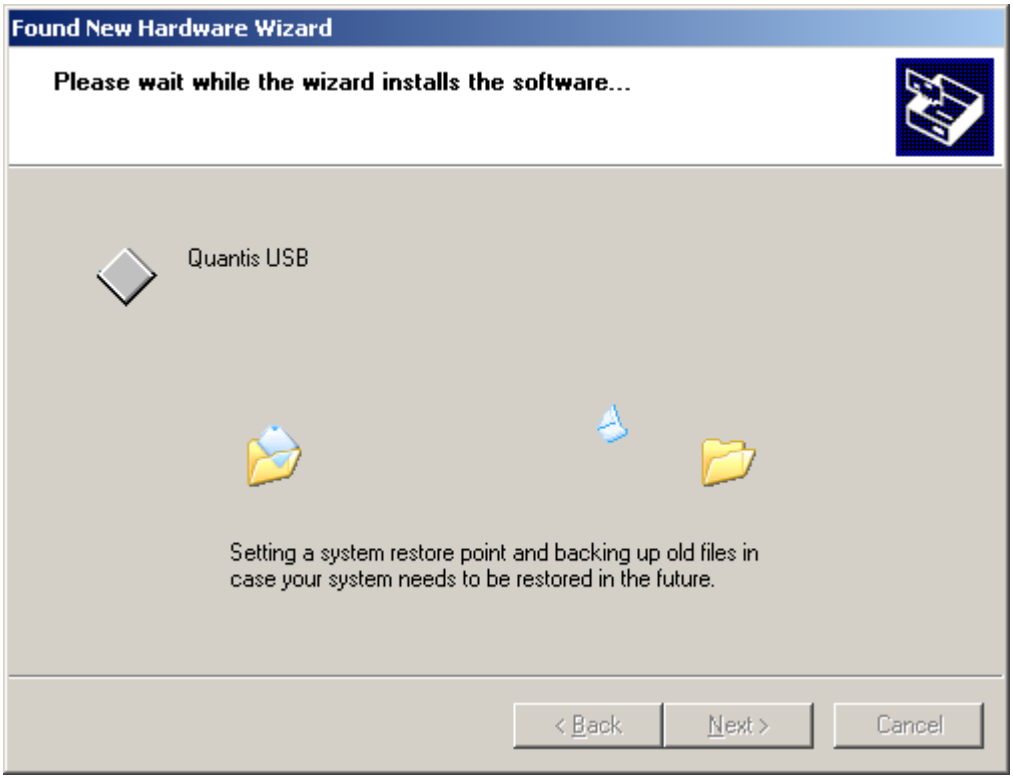
- For the Quantis PCI and Quantis PCIe select `D:\Drivers\Windows\QuantisPci`.
- For the Quantis USB select `D:\Drivers\Windows\QuantisUsb`.

Click the **Next** button to validate.



3.1.1.4. Found New Hardware Wizard: Installation

Wait while the wizard installs the Quantis driver.



3.1.1.5. Found New Hardware Wizard: Completed

When the wizard has finished installing the Quantis driver, click the **Finish** button to exit the installation. Reboot the computer if asked.

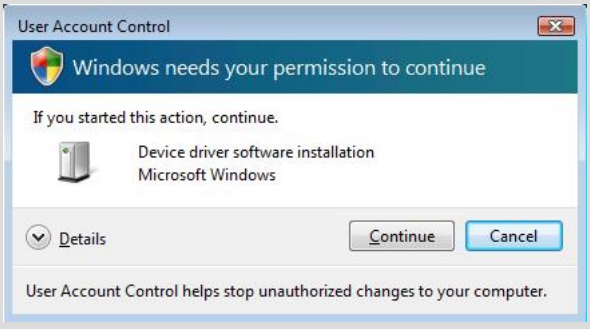
Your Quantis device is now installed. You can go to the next Chapter and install the application software.

3.1.2. Windows Vista

When the Quantis RNG is inserted into your computer for the first time, the operating system will detect the device automatically and display a *New Hardware Found* message. The following are step-by-step installation instructions.

Note

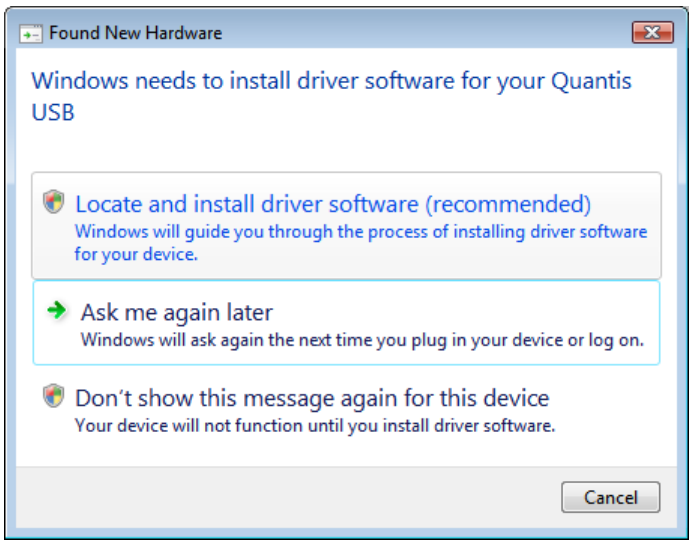
One or more intermediate dialog boxes may appear during the process stating *Windows needs your permission to continue*. Click *Continue* to proceed.



3.1.2.1. Found New Hardware Wizard: Welcome

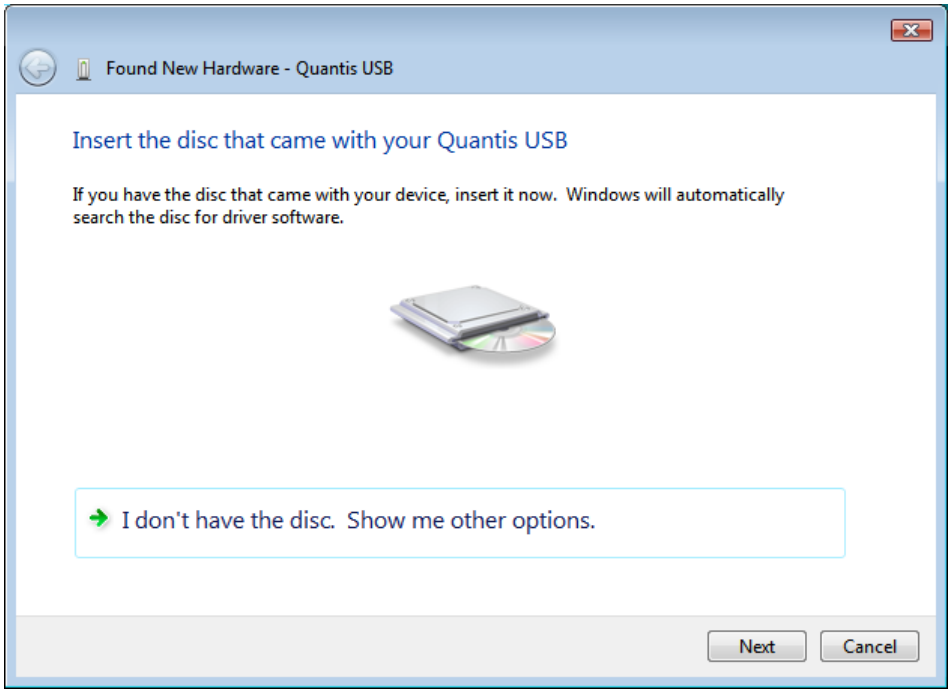
Windows will search for a driver on your computer, on removable media (e.g CD-ROM) and on the Windows Update Web site.

Let Windows try to locate the driver by clicking on *Locate and install driver software*.



3.1.2.2. Found New Hardware Wizard: Insert Disc

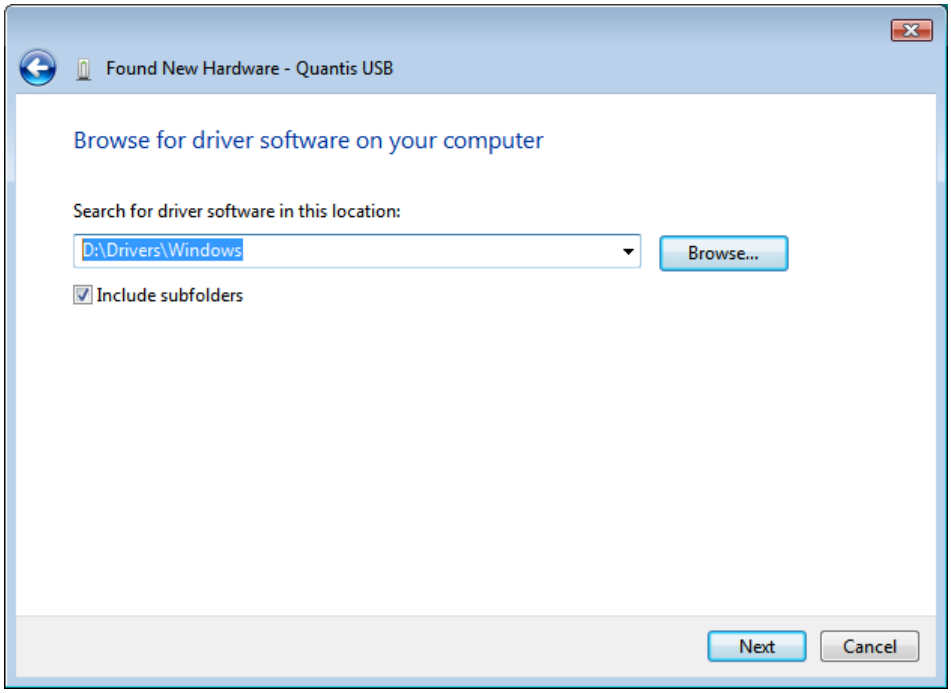
When the wizard asks you to insert the disc that came with your Quantis USB, choose *I don't have the disc. Show me other options*. This allows you to specify the location of the driver available on the USB flash drive.



3.1.2.3. Found New Hardware Wizard: Search Location

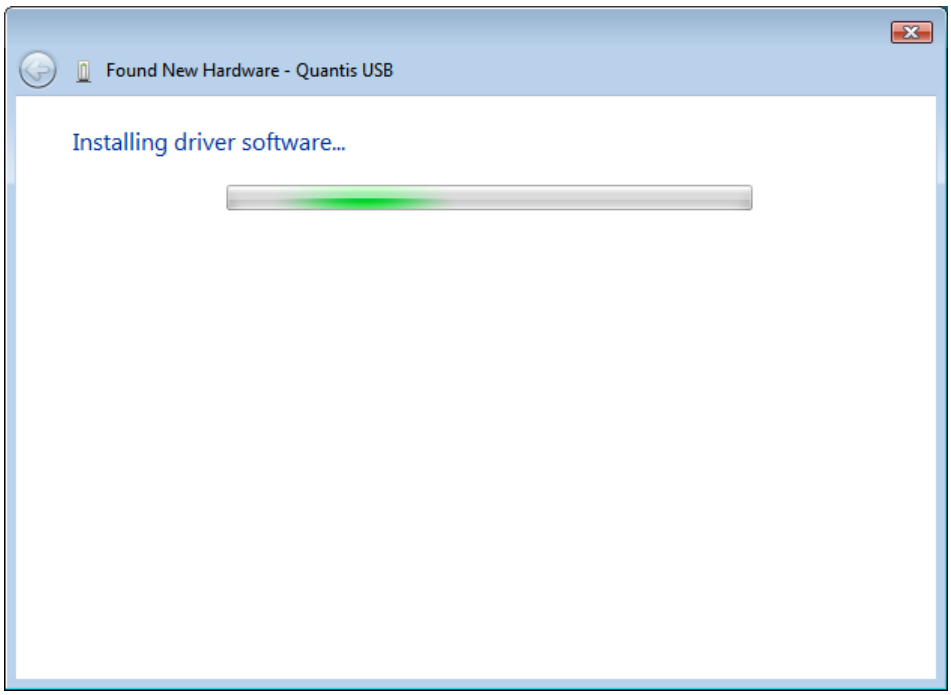
Select *Browse my computer for driver software*.

On the next dialog, click the **Browse** button and select the directory `D:\Drivers\Windows`. This directory contains all drivers for Windows. Activate the option *Include subfolders* and validate your choices by clicking the **Next** button.



3.1.2.4. Found New Hardware Wizard: Installation

Wait while the wizard installs the Quantis driver.



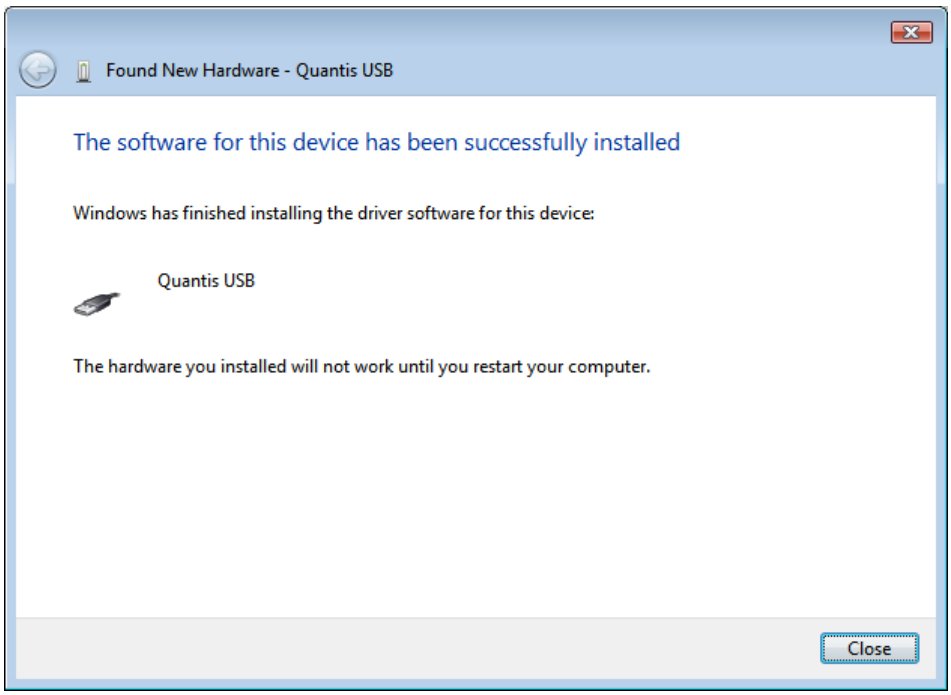
3.1.2.5. Found New Hardware Wizard: Install

If asked, validate the installation by clicking the **Install** button.

You can select *Always trust software from "ID Quantique SA"*, to avoid this question in future. All software with a valid digital signature from ID Quantique will be automatically accepted and will be installed without prompting.

3.1.2.6. Found New Hardware Wizard: Completed

When the wizard has finished installing the Quantis driver, click the **Close** button to exit the installation. Reboot the computer if asked.



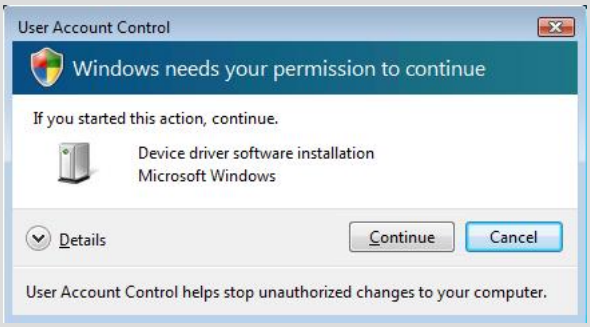
Your Quantis device is now installed. You can go to the next Chapter and install the application software.

3.1.3. Windows 7

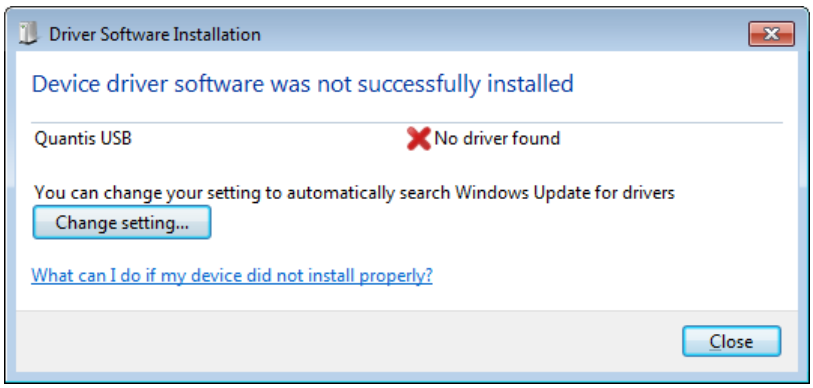
When the Quantis RNG is inserted into your computer for the first time, the operating system will detect the device automatically and search the Windows Update Web site for a driver.

Note

One or more intermediate dialog boxes may appear during the process stating *Windows needs your permission to continue*. Click *Continue* to proceed.



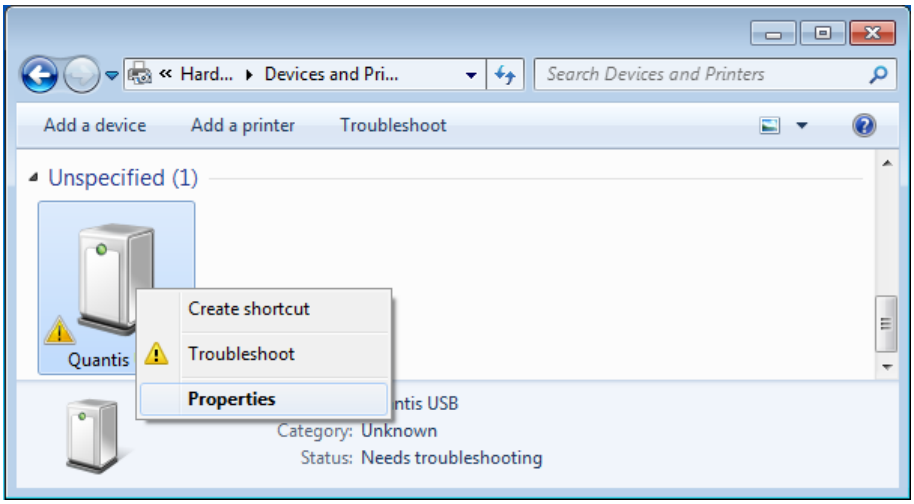
Since the driver for your Quantis device is not available on this site, this search will fail, and you will have to manually point Windows to the driver.



Close the dialog and read the following for the step-by-step installation instructions.

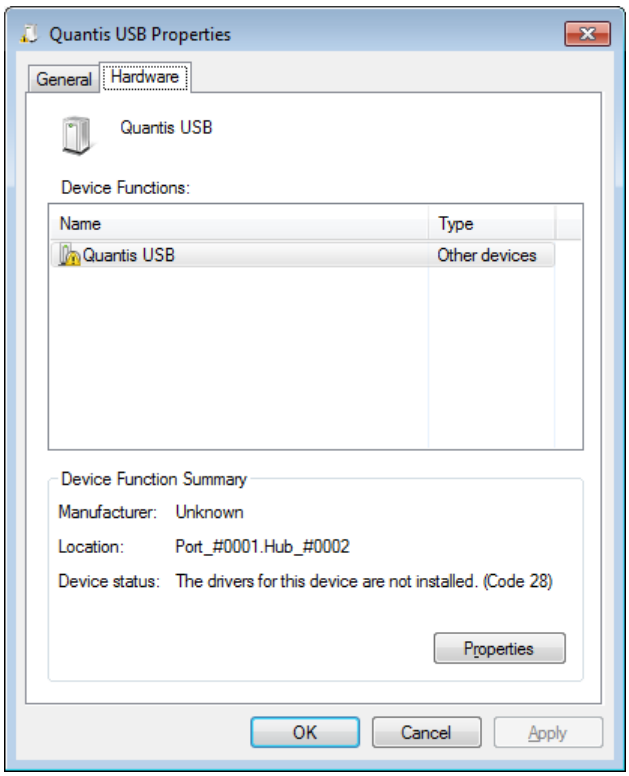
3.1.3.1. Devices and Printers

Open the *Start Menu* and select *Devices and Printers*. Scroll down until the Quantis device appears. Click on the Quantis device with the right mouse button. Select *Properties* on the menu.



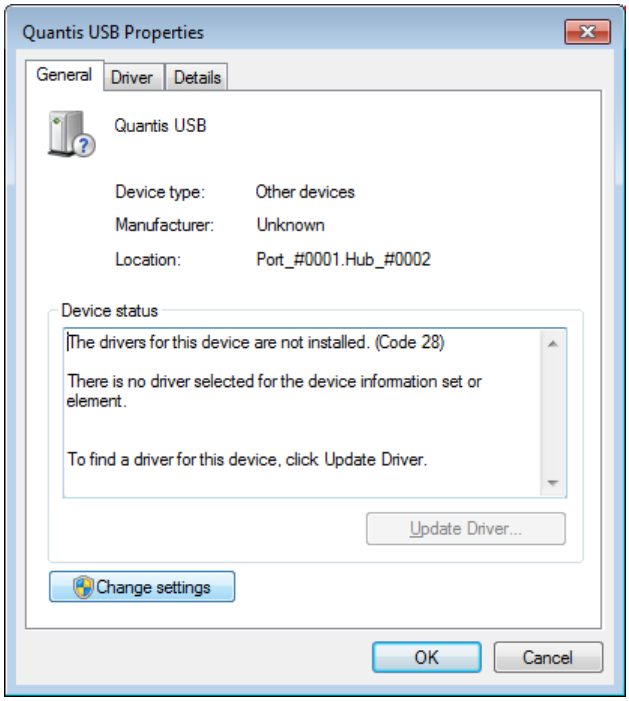
3.1.3.2. Quantis Properties: Hardware

In the Quantis Properties dialog, click on the *Hardware* tab and then on the *Properties* button.

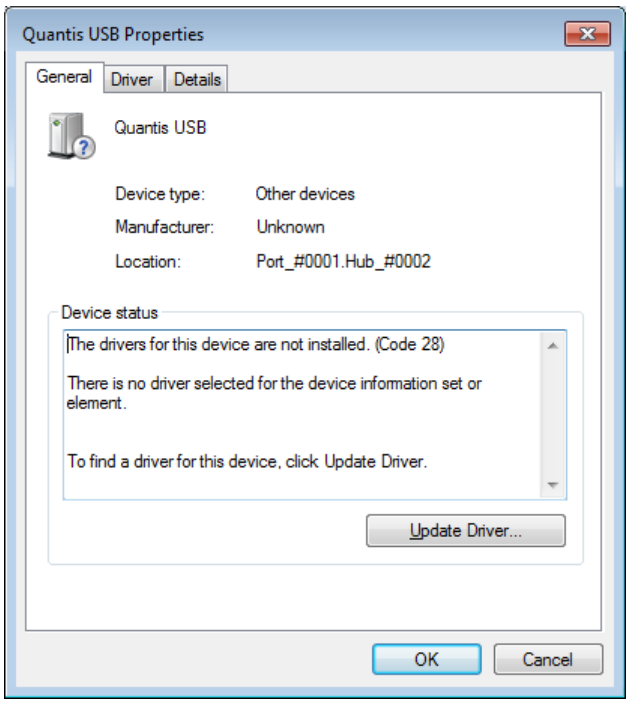


3.1.3.3. Quantis Properties: Update Driver

First click on the button *Change settings*.

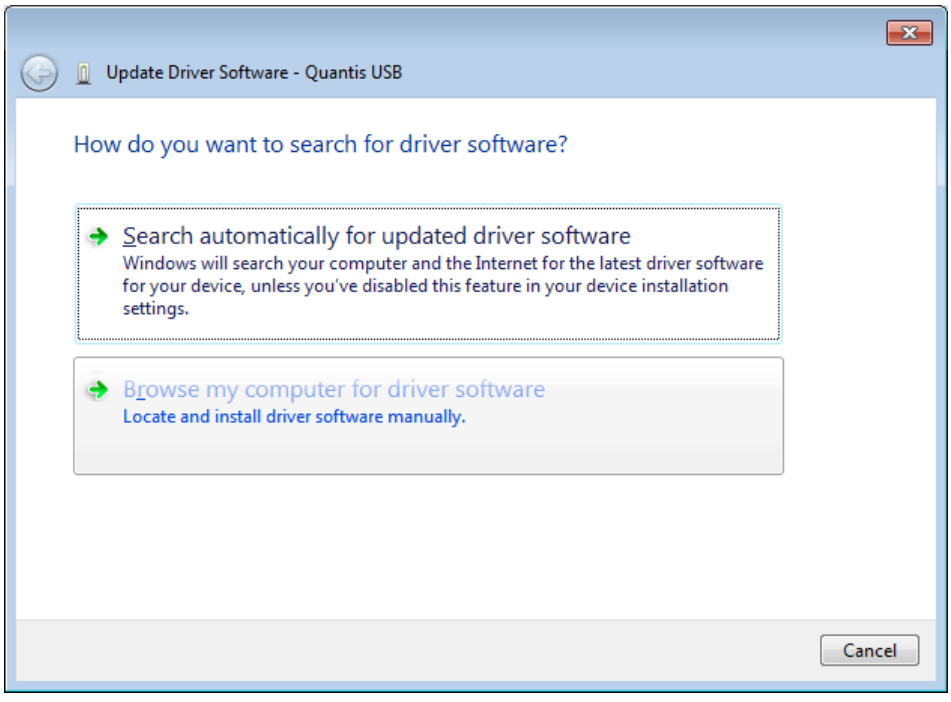


This will enable the *Update Driver* button. Click on it.



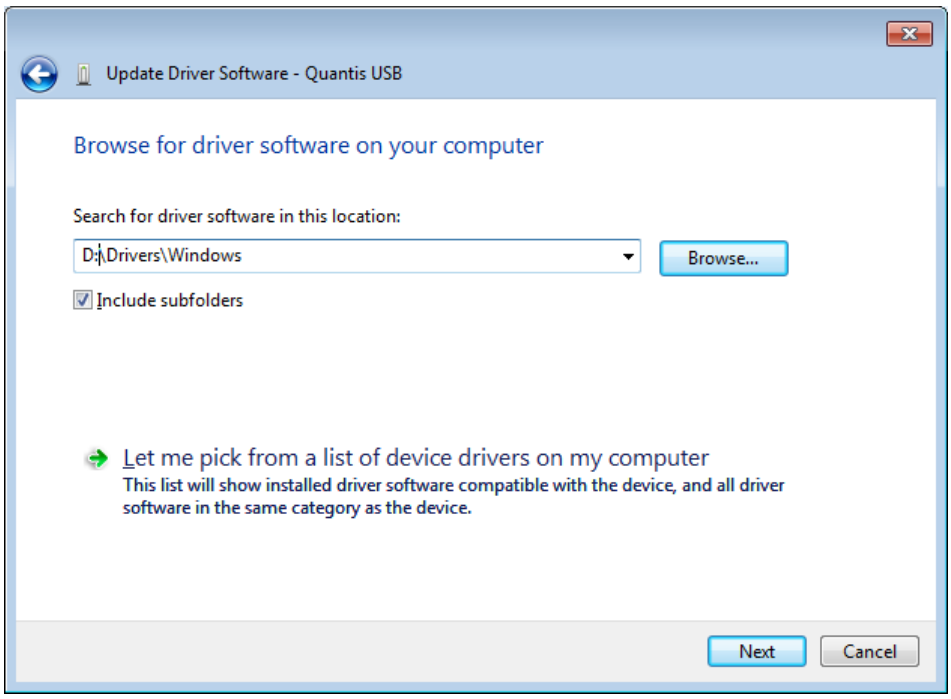
3.1.3.4. Update Driver Software: Search Driver

Driver is available on the USB flash drive provided. Select *Browse my computer for driver software*.



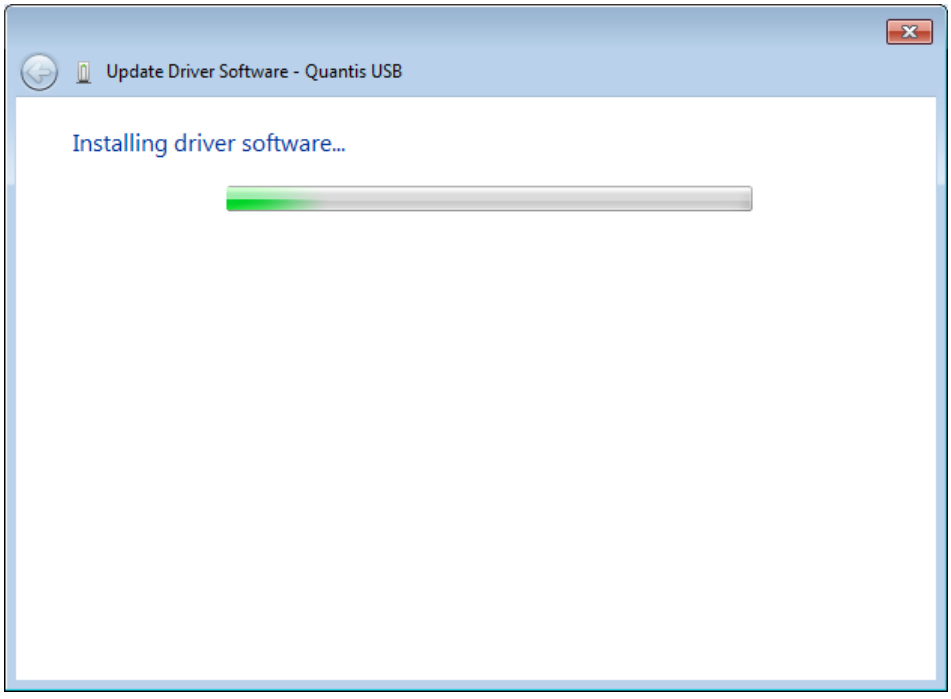
3.1.3.5. Update Driver Software: Search Location

Click the button **Browse** an select the directory `D:\Drivers\Windows`. This directory contains all drivers for Windows. Activate the option *Include subfolders* and validate your choices by clicking the **Next** button.



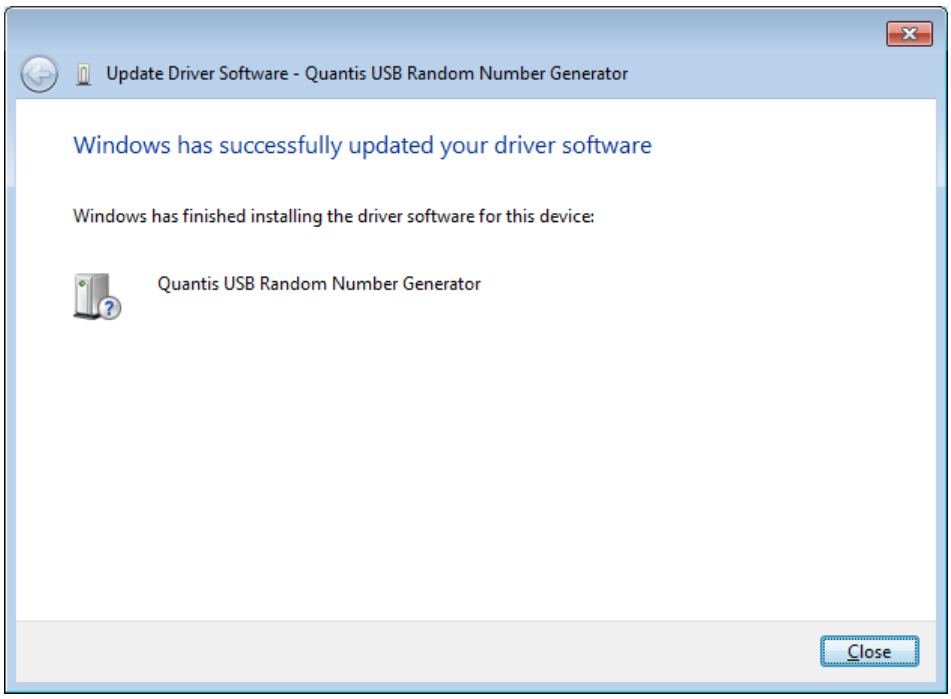
3.1.3.6. Update Driver Software: Installation

Wait while the Windows installs the driver.



3.1.3.7. Update Driver Software: Completed

When the wizard has finished installing the Quantis driver, click the **Close** button to exit the installation. Reboot the computer if asked.



Your Quantis device is now installed. You can go to the next Chapter and install the application software.

3.2. Linux Operating System

This section contains instructions on how to install Quantis devices on Linux Operating Systems.

Note

In this section, we assume that the USB flash drive with the software is mounted on `/media/USB_FLASH`. If this is different on your machine, substitute your corresponding drive name in the appropriate places in this instruction.

Important note for Ubuntu users

Ubuntu does not include the `root` user. Instead, administrative access is given to individual users, who may use the **sudo** application to perform administrative tasks. To use **sudo** on the command line, preface your command with **sudo**:

```
$ sudo my_command_requiring_administrative_access
```

In this document, when a command must be executed as `root` on Ubuntu, preface the command with **sudo**.

Please refer to the Ubuntu guide for more details about the command **sudo**.

3.2.1. Quantis PCI and Quantis PCI Express

The Quantis PCI and Quantis PCIe cards require a kernel module to be compiled and installed to work correctly. The following are step-by-step installation instructions.

3.2.1.1. Install Pre-Requirements

Before being able to compile a Quantis PCI kernel module, you must install a compiler and the Linux kernel sources.

Note

Generally, you do not need the full source tree in order to build a module against the running kernel. Most of the time you just need the kernel headers.

3.2.1.1.1. Debian-based Distributions

Debian-based distributions have a powerful tool for building kernel modules: **module-assistant**. **module-assistant** aims to facilitate the process of building kernel modules from source. Type following command as `root` to install module assistant:

```
# apt-get install module-assistant
```

To download the headers corresponding to the current kernel and other mandatory tools, simply run (as `root`):

```
# m-a prepare
```

This command determines the name of the required kernel-headers package, installs it if needed and creates the `/usr/src/linux` symlink if needed. Also installs the **build-essential** package to ensure that the same compiler environment is established.

All required software has been installed. You can skip to Section 3.2.1.2, “Compile and Install Driver”.

3.2.1.1.2. Red Hat Enterprise Linux and CentOS Distributions

To build kernel modules on Red Hat Enterprise Linux and CentOS distributions it is not necessary to download the entire kernel. To build a module for the currently running kernel, only the matching **kernel-devel** package is required. Run the following command to install the **kernel-devel** package using **yum**:

```
# yum install kernel-devel
```

Important

The previous command installs the kernel headers for the latest kernel available in the repository. If your system is not up-to-date you need first to update the kernel and then boot the new kernel before installing the **kernel-devel** package:

```
# yum update kernel*
# reboot
# yum install kernel-devel
```

To compile the kernel driver you also need to install the developer tools such as GNU GCC C/C++ compilers, make and others. You can install them with the following command (as `root`):

```
# yum groupinstall "Development Tools"
```

All required software has been installed. You can skip to Section 3.2.1.2, “Compile and Install Driver”.

3.2.1.1.3. Other Distributions

Install the GNU GCC compiler and the header corresponding to the current kernel (or the whole source kernel). Please refer to the guide of your distribution for help on installing packages.

3.2.1.2. Compile and Install Driver

Now that all pre-requirements have been installed you can compile and install the driver.

First copy the source code of the driver to `/tmp`:

```
$ cp -R /media/USB_FLASH/Drivers/Unix /tmp/
```

Change to the directory which contains the driver and compile the driver:

```
$ cd /tmp/Unix/QuantisPci/
$ make
```

When compilation finish, install and load the driver with following commands (as `root`):

```
# make install
# modprobe quantis_pci
```

You can verify that the driver has been successfully loaded and all your Quantis PCI and PCIe cards have been detected with the command **dmesg**:

```
$ dmesg | grep quantis_pci
quantis_pci: Initializing Quantis PCI RNG driver version 2.0
quantis_pci:   driver build Feb 12 2010 14:26:14
quantis_pci:   support enabled up to 10 PCI card(s)
quantis_pci: Found card #0
quantis_pci:    core version 0x040a1201
quantis_pci:    device registered at /dev/qrandom0
quantis_pci: Driver loaded. Found 1 card(s)
```

Important

If you update your kernel, you must recompile and reinstall the driver!

3.2.1.3. Auto-load the Driver on Boot-up

Instead of using the **modprobe** command each time you want to load the driver, you can let the system load the driver automatically on boot-up.

Note

Some distributions already load the driver on boot for each detected device (if available).

To check if your system does this for you, reboot your computer and run the command **dmesg** as explained in previous section. If the driver has been loaded and all Quantis devices have been detected, you can skip this section.

3.2.1.3.1. Debian-based Distributions

To automatically load the driver on boot, simply add the driver's name at the end of `/etc/modules`. You can type the following command (as `root`) to add the entry:

```
# echo "quantis_pci" >> /etc/modules
```

3.2.1.3.2. Red Hat Enterprise Linux and CentOS Distributions

Red Hat Enterprise Linux checks for the existence of the `/etc/rc.modules` file at boot time, which contains various commands to load modules. The following commands configure the loading of the `quantis_pci` module at boot time (as `root`):

```
# echo modprobe quantis_pci >> /etc/rc.modules
# chmod +x /etc/rc.modules
```

3.2.1.3.3. Other Distribution

Please consult your distribution's guide to know how to auto-load a driver on boot-up.

3.2.1.4. Modify the Device's Permissions

Depending on the distribution, the Quantis PCI device might be accessible only to user `root`. UDEV (the device manager for the Linux 2.6 kernel series) must be instructed to allow other users to access the Quantis.

3.2.1.4.1. The `plugdev` group

IDQ provides a rule for UDEV that allows all users in group `plugdev` to access the Quantis device. The group `plugdev` is generally created on all modern distributions.

First 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`. When the **grep** command does not display any message, then the `plugdev` group doesn't exists on your system. Type following command (as `root`) to create the `plugdev` group:

```
# groupadd --gid 46 plugdev
```

3.2.1.4.2. Adding users to the `plugdev` group

Every user who is a member of the `plugdev` group can access hot-pluggable devices (digital cameras, USB drives etc.).

You can use the command **groups** to display the groups your user is in:

```
$ groups
users adm dialout cdrom plugdev lpadmin admin sambashare
```

If your user is not in the group `plugdev`, use the **usermod** command (as `root`) to add the user `LOGIN` to the group `plugdev` (substitute your own login name for `LOGIN`):

```
# usermod -G plugdev -a LOGIN
```

3.2.1.4.3. UDEV rules

In the directory `Drivers/Unix/` on the USB flash there are two files with UDEV rules:

- `idq-quantis-rhel.rules` for Red Hat Enterprise Linux and CentOS distributions.
- `idq-quantis.rules` for all other distributions.

Copy (as `root`) the right file into `/etc/udev/rules.d/` directory:

- On Red Hat Enterprise Linux and CentOS distributions:

```
# cp /media/USB_FLASH/Drivers/Unix/idq-quantis-rhel.rules
  /etc/udev/rules.d/
```

- On all other distributions:

```
# cp /media/USB_FLASH/Drivers/Unix/idq-quantis.rules
  /etc/udev/rules.d/
```

Note

The files `idq-quantis-rhel.rules` and `idq-quantis.rules` contain UDEV rules for both Quantis PCI and Quantis USB devices.

The udev daemon must now reload the rules. Type following command (as `root`) to reload the rules:

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

Note

On Red Hat Enterprise Linux and CentOS distributions the **udevadm** command does not exists. User following command to reload the rules instead:

```
# udevcontrol reload_rules
```

Note

The udev daemon only apply rules when creating the device's node (when the drivers loads). If the Quantis PCI driver is already loaded you need thus to unload and reload it to have the right permissions on the device:

```
# rmmod quantis_pci
# modprobe quantis_pci
```

3.2.1.5. Check Your Device

The driver has been installed and the system configured. You can now check if your device works correctly by reading some random bytes from the device. The following command reads 100 bytes from the first Quantis PCI device found on the system (not as `root`):

```
$ head -c 100 /dev/qrandom0
```

Important

It is important not to run the **head** command as `root` but as the standard user who will use the Quantis PCI device. Otherwise you won't be verifying that permission has been granted to you to access the device.

The above command will display some random characters on the console.

Important

If you get one or more *Operation not permitted* messages, you don't have the right permissions to access the Quantis device. In such a case:

- Verify that `/etc/udev/rules.d/idq-quantis.rules` exists and has the same content as the one provided on the USB flash drive.
- Verify that your user is in the `plugdev` group.
- Reboot the system to ensure that the new rules are loaded by the udev daemon.

You Quantis device is now installed. You can go to the next Chapter to install the application software.

3.2.2. Quantis USB

Quantis USB only requires USB support enabled in the kernel[1]. The Quantis USB device is accessed through the open source library libusb-1.0 .

The following are step-by-step installation instructions.

3.2.2.1. libusb-1.0 Installation

Quantis USB device is accessed through the library libusb-1.0[2]. This library is available on all recent distributions and can be installed using the package manager of the distribution.

Warning

Do not confuse libusb-0.1 with libusb-1.0! libusb-0.1 is the legacy release and is not developed any more. As of December 2008, libusb-1.0 is the current stable branch. This new branch, used to access the Quantis USB, adds features missing in the first release.

3.2.2.1.1. Debian-based Distributions

Note for Debian users

libusb-1.0 is only available on Debian Squeeze and newer releases. It is also available on Debian lenny-backports. Please refer to the Debian help on how to enable backports packages. On all other Debian releases, you need to manually install libusb-1.0. Please refer to Section 3.2.2.1.4, “Manually Compile libusb-1.0”.

Note for Ubuntu users

libusb-1.0 is only available on Ubuntu Jaunty (9.04) and newer releases. On previous Ubuntu releases you need to manually install libusb-1.0. Please refer to Section 3.2.2.1.4, “Manually Compile libusb-1.0”.

Type the following command (as `root`) to install libusb-1.0 and the development package (needed if you want to write your own application using the Quantis libraries):

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

3.2.2.1.2. Red Hat Enterprise Linux and CentOS Distributions

libusb-1.0 is currently not available on Red Hat Enterprise Linux nor CentOS distributions. You need to manually install libusb-1.0. Please refer to Section 3.2.2.1.4, “Manually Compile libusb-1.0”.

3.2.2.1.3. Other Distributions

Use the package manager of your distribution to install the library libusb-1.0. If the package is not available, please refer to Section 3.2.2.1.4, “Manually Compile libusb-1.0”.

3.2.2.1.4. Manually Compile libusb-1.0

If library libusb-1.0 can not is not available on the list of packages in the package manager of your distribution, you can easily compile it by hand.

First you need to download the library's sources. Go to the address `http://sourceforge.net/projects/libusb/files/libusb-1.0/` and download the latest version.

[1] USB support in the kernel is generally enabled on all modern Linux distributions.
[2] `http://libusb.org/wiki/Libusb1.0`

Open the **Terminal** application and change the working directory to the one containing the downloaded libusb-1.0 archive. Unpack the archive and compile the library (replace *x* with your version number):

```
$ tar xvjf libusb-1.0.x.tar.bz2
$ cd libusb-1.0.x/
$ ./configure --prefix=/usr
$ make
```

When the library has been compiled, install it with the following command (as `root`):

```
# make install
```

3.2.2.2. Modify the Device's Permissions

By default, the Quantis USB device is accessible only to user `root`. UDEV (the device manager for the Linux 2.6 kernel series) must be instructed to allow other users to access the Quantis. Please follow instructions on Section 3.2.1.4, “Modify the Device's Permissions”.

Important

If the Quantis USB device was already plugged in before the reloading of the udev rules, please unplug and replug the Quantis device, otherwise it will have the wrong permissions.

3.2.2.3. Check Your Device

All requirements have been installed. You can now plug your Quantis USB device into your computer.

You can now check if your device works correctly with the **lsusb** command as following (not as `root`):

```
$ lsusb -d 0aba:0102 -v
```

Important

It is important not to run the **lsusb** command as `root` but as the standard user who will use the Quantis USB device. Otherwise you won't be verifying that permission has been granted to you to access the device.

Note

If above command returns the message:

```
lsusb: command not found
```

then the command **lsusb** is not installed. Install the **usbutils** package to fix the problem.

The output of above command should be similar to the following:

```
Bus 002 Device 035: ID 0aba:0102 Ellisys
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass          255 Vendor Specific Class
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x0aba Ellisys
```

```
  idProduct          0x0102
  bcdDevice            2.00
  iManufacturer           1 id Quantique
  iProduct                2 Quantis USB
  iSerial                 3 070001A410
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           25
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0x80
      (Bus Powered)
    MaxPower              300mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           1
      bInterfaceClass       255 Vendor Specific Class
      bInterfaceSubClass      0
      bInterfaceProtocol      0
      iInterface              0
      Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x86  EP 6 IN
        bmAttributes            2
          Transfer Type            Bulk
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0200  1x 512 bytes
        bInterval               0
Device Qualifier (for other device speed):
  bLength                10
  bDescriptorType         6
  bcdUSB               2.00
  bDeviceClass          255 Vendor Specific Class
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  bNumConfigurations      1
Device Status:     0x0000
  (Bus Powered)
```

Important

Please verify that you have the *Device Qualifier* and *Device Status* information sections and not messages such as:

```
can't get device qualifier: Operation not permitted
can't get debug descriptor: Operation not permitted
cannot read device status, Operation not permitted
```

If you get one or more *Operation not permitted* messages, you don't have the right permissions to access the Quantis device. In such a case:

- Verify that `/etc/udev/rules.d/idq-quantis.rules` or `/etc/udev/rules.d/idq-quantis-rhel.rules` exists and has the same content as the one provided on the USB flash drive.
- Verify that your user is in the `plugdev` group.
- Reboot the system to ensure that the new rules are loaded by the udev daemon.

You Quantis device is now installed. You can go to the next Chapter to install the application software.

3.3. Mac OS X Operating System

This section contains instructions on how to install and operate Quantis devices on Mac OS X Operating Systems. The binaries are only supported from Mac OS X 10.6 (Snow Leopard). In order to use the driver with earlier versions of the OS, the sources should be recompiled.

3.3.1. QuantisPCI and QuantisPCI Express

The Quantis PCI is not yet supported on Mac OS X.

3.3.2. Quantis USB

The binary distribution is contained in the file `EasyQuantis.dmg` and is composed of:

- the **EasyQuantis** application which allow the immediate use of the Quantis;
- the `QuantisRNG-2.7.0-Darwin-i386.pkg` which is a standard package for the Mac OS X installer and which contain all the SDK to use the Quantis from the C/C++ programming language.
- the `Readme` describing briefly how to install the software.

3.3.3. Installation

To access the content of the file `EasyQuantis.dmg` simply double click on its icon (in the **Finder**). Once the disk opened, simply drag the icon of the **EasyQuantis** application in the `Applications` folder (or in any other convenient folder).

To install the SDK, double click on its icon and follow the instructions of the package manager. Once the license is accepted, the different files (headers, library and binaries) will be available in the directory `/opt/IDQQuantis`.

This directory is deliberately chosen in order to avoid interferences with the standard system. The directory name should normally only be used to add search paths for headers files (`/opt/IDQQuantis/include`) or libraries (`/opt/IDQQuantis/lib`) to the compiler command line or to the **XCode** projects.

Note

The directory `/opt/IDQQuantis` is a perfectly standard Unix path and is accessible without problem from the **Terminal**. Since it is not a standard Mac OS X path, it is not available directly in the **Finder**. It is however possible to reach it by choosing the menu Go to Folder (Shift+Command+g).

3.3.4. Implementation details

Like on Linux (see the paragraph Section 3.2.2.1, “libusb-1.0 Installation”), the Quantis USB is available by using the `libusb 1.0`. This software is not included in Mac OS X and should be installed manually. The two projects

1. MacPorts [http://www.macports.org]
2. Fink [http://www.finkproject.org/]

provide an easy solution for installing additional Unix software on Mac OS X.

The MacPorts `libusb 1.0` package has been chosen for the use with the distributed driver. To facilitate the distribution of the software, the library `libusb 1.0` has been statically linked inside the library `Quantis`.

The application **EasyQuantis** needs the libraries `qt`, `boost` and `png`. All these libraries are statically linked inside the application. Unfortunately none of the `qt` provided by the two projects has static libraries. The `qt` library has then been compiled statically directly from the sources provided on the official site [http://qt.nokia.com/products/].

Even if the **EasyQuantis** use graphical Mac OS X widgets, it is not fully recognized by the system as a Cocoa application. To bypass this problem, a small Cocoa program has been written to load the actual **EasyQuantis** program.

Since all the Cocoa applications are Mac OS X bundles,[3]the actual program is located inside the subdirectory `libexec` of the application bundle with the name **EasyQuantis** (the Cocoa wrapper is located in the subdirectory `MacOS` with the name **EasyQuantis**).

By using this trick, the **EasyQuantis** has the standard behaviour of a Cocoa application. It would have perhaps been possible to modify the main source of the **EasyQuantis** application, but this solution would have make the portability across platforms more complicated and the wrapper solution has been preferred.

It is generally not possible to link statically all the libraries on Mac OS X since most of the standard system libraries are provided only as dynamic libraries. To achieve a static linking as complete as possible one has to do:

- add explicitly the name of the library (i.e. `/opt/local/lib/libusb-1.0.a`) to the list of objects (otherwise the Mac linker, when static and dynamic libraries are available, prefers dynamic libraries);
- add the correct parameters to the linker which otherwise do not link with the system dynamic libraries. For instance, the following parameters[4] have been added (when invoking the compiler on command line) in order to link the `Quantis` library: *`-framework IOKit -framework CoreFoundation -lSystem`*

3.3.5. Known problems

The following problems have been noticed:

- the progress bar do not work while the application **EasyQuantis** acquires random data from the Quantis device;
- the actual **EasyQuantis** command (located in `/opt/IDQQuantis/bin`) do not work correctly in graphical mode.

  This problem arise only when the **qt** library is statically linked in the program. It is caused while some additional files for displaying graphic widgets are not found. The problem do not exist when the application is used inside an application bundle or if a framework (containing the necessary resources) for the **qt** library is installed in the system.

3.4. Solaris / OpenSolaris

Install the package containing the drivers by typing the command (after uncompressing the package):

```
pkgadd -d <path-to-quantis>/Packages/Solaris/Sparc/IDQpcidrv-2.1-sol10-sparc"
```

[3]A bundle is a directory with a specific organisation containing all what is needed to run the application (icons, resources, menu, documentation and naturally the program itself).

3.5. FreeBSD

On FreeBSD, first install the Ports package. Then, execute

```
cd /usr/ports/security/quantis-kmod
make
make install
```

to execute the provided makefile.

Chapter 4. The EasyQuantis application

Quantis is delivered with the EasyQuantis application. This application allows you to quickly and easily generate random data.

Important

Quantis PCI Express is software-compatible with Quantis PCI. EasyQuantis considers Quantis PCIe devices as Quantis PCI devices.

4.1. Installation

4.1.1. Windows Operating Systems

EasyQuantis is provided as a Microsoft Installer (MSI) package for easy installation. Just double-click on `EasyQuantis.msi` file and follow on-screen instructions.

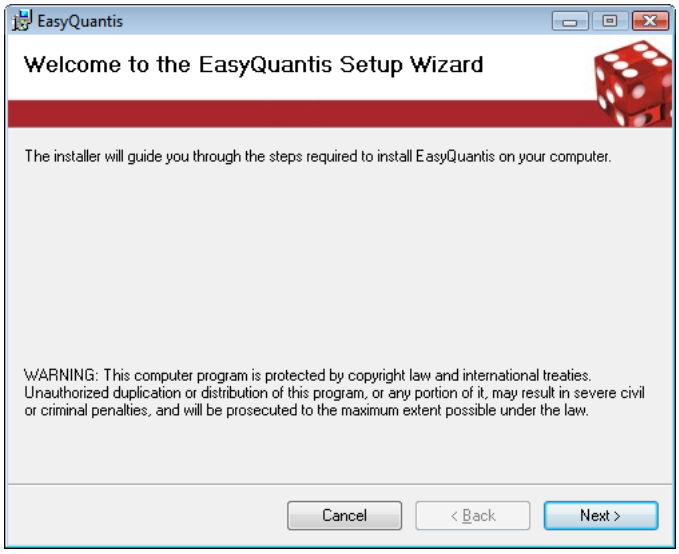


Figure 4.1. EasyQuantis Setup Wizard welcome

To launch the application click on the `EasyQuantis` icon in `Start -> Program`.

4.1.2. Linux Operating Systems

4.1.2.1. Install Requirements

EasyQuantis requires the `libusb-1` and Qt4 libraries (`qt4-core` and `qt4-gui`) to work.

If you have a Quantis PCI card, please follow instructions in Section 3.2.2.1, “libusb-1.0 Installation”. If you have a Quantis USB device, `libusb-1` has already been installed on your system.

Qt libraries are available on all major Linux distributions and they can be installed using the package manager of the distribution.

4.1.2.1.1. Debian-based Distributions

Open a terminal and install `libqt4-core` and `libqt4-gui` using the following command (as `root`):

```
# apt-get install libqt4-core libqt4-gui
```

4.1.2.1.2. Other Distributions

Install `libqt4-core` and `libqt4-gui` using the package manager of your distribution.

4.1.2.2. Install the Application

The EasyQuantis and Quantis libraries are provided in a bz2 archive.

On 32-bit systems, you can install the application using the following commands as `root` (replace *x.y* with your version number):

```
# cd /mnt/USB_FLASH/
# tar xvjf QuantisRNG-2.x.y-Linux-i386.tar.bz2 -C /tmp/
# cd /tmp/QuantisRNG-2.x.y-Linux-i386/
# mv bin/EasyQuantis /bin/
# mv lib/libQuantis* /lib/
```

On 64-bit systems, use the following commands as `root` instead:

```
# cd /mnt/USB_FLASH/
# tar xvjf QuantisRNG-2.x.y-Linux-amd64.tar.bz2 -C /tmp/
# cd /tmp/QuantisRNG-2.x.y-Linux-amd64/
# mv bin/EasyQuantis /bin/
```

On some distributions you need to copy the 64 bits libraries to `/lib64`:

```
# mv lib64/libQuantis* /lib64/
```

On other distributions you need to copy to `/lib/x86_64-linux-gnu`

```
# mv lib64/libQuantis* /lib/x86_64-linux-gnu
```

You can run the EasyQuantis application by typing **EasyQuantis** on a terminal:

```
$ EasyQuantis
```

4.1.2.3. Uninstall the Application

To uninstall, manually remove installed files as follows (as `root`):

For 32 bits system:

```
# rm -Rf /bin/EasyQuantis
# rm -Rf /lib/libQuantis*
```

For 64 bits system:

```
# rm -Rf /bin/EasyQuantis
# rm -Rf /lib64/libQuantis*
```

or

```
# rm -Rf /bin/EasyQuantis
# rm -Rf /lib/x86_64-linux-gnu/libQuantis*
```

4.1.3. Mac OSX

For Mac, there is a dmg package. Simply install it to get the EasyQuantis application.

4.1.4. Solaris / OpenSolaris

Install the package containing the library by typing the command (after uncompressing the package):

```
pkgadd -d <path-to-quantis>/Packages/Solaris/Sparc/IDQLibs-Apps-sparc-2.9"
```

4.1.5. FreeBSD

To install the Quantis code and application, execute

```
cd /usr/ports/security/quantis
make config
make
make install
```

Important

Running "make config" above allows you to choose a number of options. Unless you are sure to need it, disallow the Quantis GUI, since permitting it will require the installation/configuration of a large number of large packages (Qt, Perl, Ruby, boost, ...) and will take very, very long.

4.2. EasyQuantis

In the Quantis software package, two versions are provided:

- **EasyQuantis 1**: Allows to get random numbers from a Quantis device.
- **EasyQuantis 2**: Allows to get random numbers from a Quantis device and provide randomness extraction capabilities (only available on Windows and Linux at the moment).

Note

Randomness extraction principle is explained in a dedicated white paper. Please refer to the "Randomness Extraction for the Quantis True Random Number Generator" document for details. This document may be found on IDQ website.

4.3. EasyQuantis2 (Windows, Linux)

EasyQuantis2 is made of 3 tabs:

- **Acquisition**: Make acquisition of random data from a Quantis device.
- **File extraction**: Process randomness extraction form a file.
- **Extraction matrix**: Create your own matrix file.

4.3.1. Acquisition

The *Acquisition* tab allows to generate random numbers from a Quantis device.

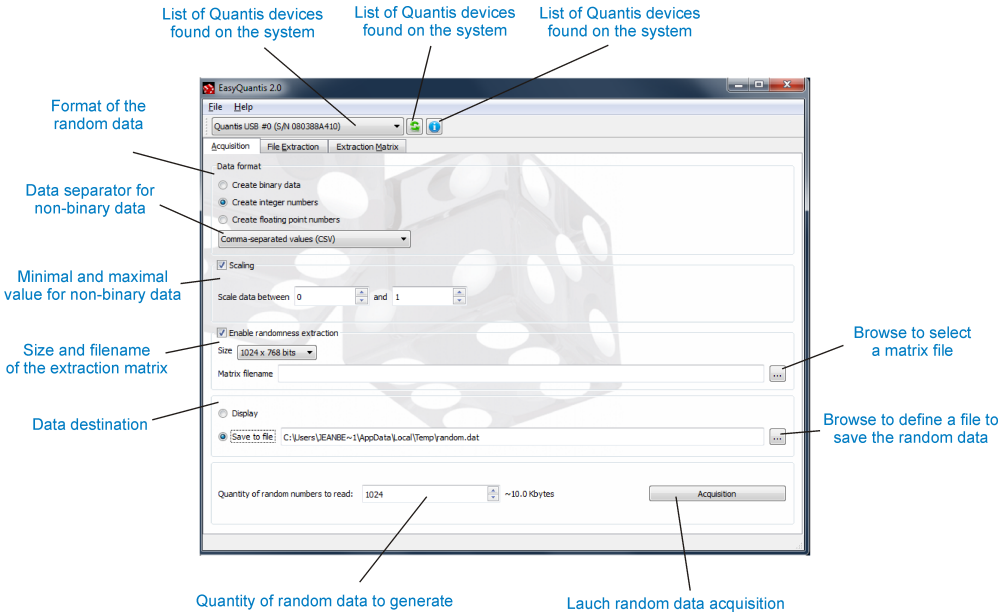


Figure 4.2. EasyQuantis2 Acquisition tab

To generate random data from a Quantis device using EasyQuantis:

1. Select a Quantis device from the list.

2. Select a data format:

   - **Binary data.** Data is read from the Quantis and returned as bytes.

   - **Integer numbers.** Generates 32-bit random numbers ranging between -2'147'483'648 and 2'147'483'647 (inclusive).

   - **Floating point numbers.** Generates numbers between 0.0 (inclusive) and 1.0 (exclusive).

3. Select a data separator:

   - **Comma-separated values (CSV).** CSV is a type of delimited text data, which uses a comma to separate subsequent values. The benefit of CSV is that they allow for the transfer of data across different applications.

     The following is an example of CSV:

     ```
     Value1,Value2,Value3,...,ValueN
     ```

   - **One entry per line.** Each value is on a separate line:

     ```
     Value1
     Value2
     Value3
     ...
     ValueN
     ```

Note

When generating binary data you cannot select a data separator.

4. Optionally, you can scale the random values to be within a smaller range.

Note

For more details about the scaling algorithms, please refer to section 5.3.7.2.1 "Integral Values: The Scaling Algorithm".

5. Optionally you can enable the randomness extraction post processing:
   - **Size**. Select the size of the matrix.
   - **Matrix filename**. Select a matrix filename. The size of the file must be greater or equal to the matrix size.
6. Select the data destination:
   - **Display.** Data is displayed on screen. You can copy-paste the data to your application.

     Use this option for *small* amounts of random data that you want to use it right away.

Note

This option is not available for binary data.

   - **Save to file.** The data is written to a file. Use this option to generate large amounts of random data or to generate data for later use.

Note

On some systems (Mac OS X, Windows) the temporary default directory is not very convenient and should probably changed to a better suited one. This will ease the manipulation of the produced file from outside the **EasyQuantis** program.

7. Select the amount of data to generate. File size is limited to 2 GBytes (2'147'483'647 bytes).
8. Click the Generate button and wait while the application generates the random data.

4.3.2. File extraction

The *File extraction* tab allows to extract the randomness from a file.

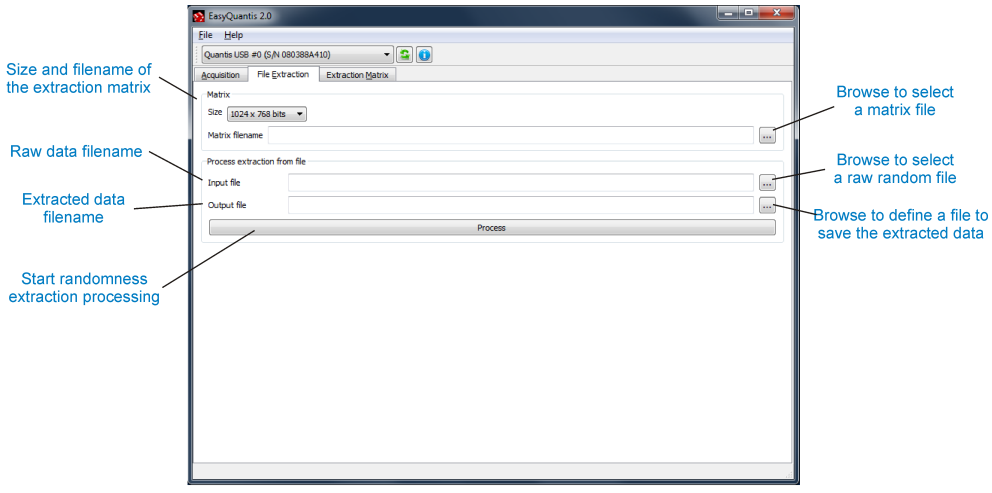


Figure 4.3. EasyQuantis2 File extraction tab

To extract randomness from a raw random file using EasyQuantis:

1. Select a matrix size.
2. Select a matrix file. The file size must be greater or equal to the matrix size.
3. Select the input file to process:
4. Define an output file to save the extracted data.
5. Click the Process button and wait while the application generates the random data.

Note

Randomness extraction algorithm is based on 64 bits integers. Using a 64 bits platform allows significant higher speed processing.

4.3.3. Extraction matrix

The *Extraction matrix* tab allows to create your own extraction matrix file.

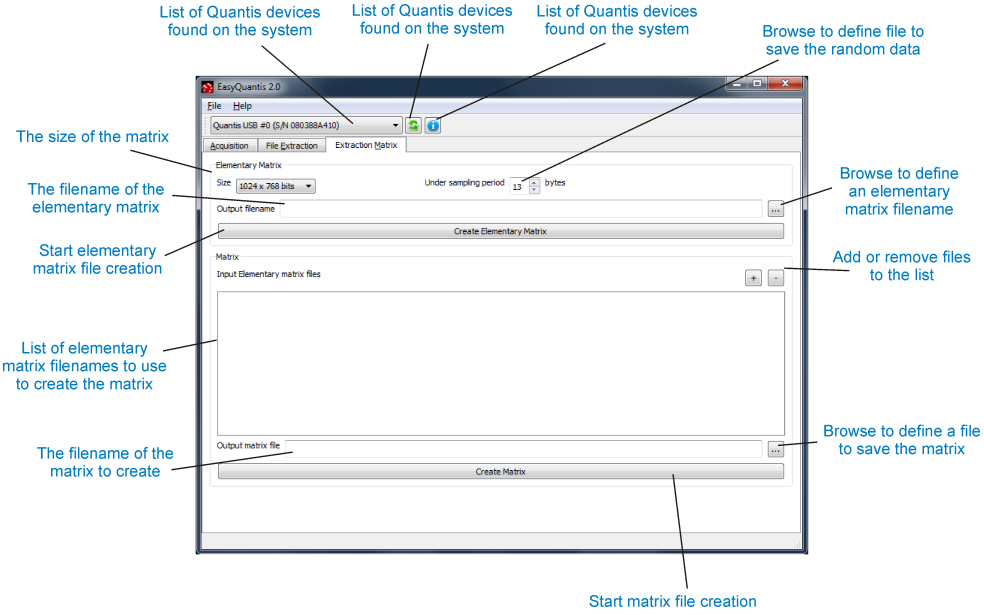


Figure 4.4. EasyQuantis2 Extraction matrix tab

To create an elementary matrix:

1. Select a Quantis device from the list.
2. Select the matrix size
3. Select the the under sampling period. Recommended value is 13 bytes.
4. Define an elementary matrix filename.
5. Click Create elementary matrix.

Note

Repeat this operation in order to have at least 2 elementary matrix file.

To create an extraction matrix:

1. Select a list of the elementary files with the + and - buttons.
2. Define an output matrix filename
3. Click create matrix

4.4. Using EasyQuantis 1.x (other OS)

Figure 4.5, “EasyQuantis main window” shows the main window of the **EasyQuantis** application

Note

On some systems, when the application has been downloaded from Internet, one may have to acknowledge that fact on first use.

.

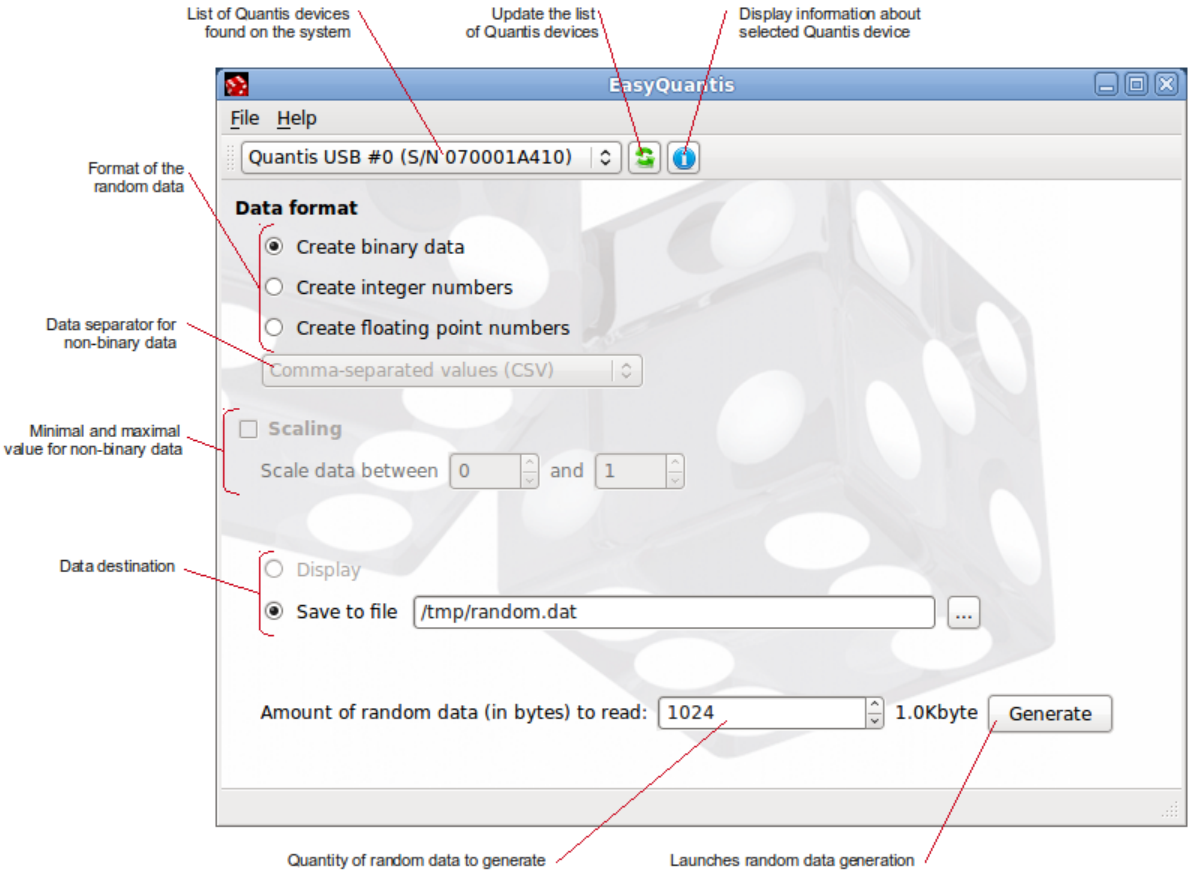


Figure 4.5. EasyQuantis main window

To generate random data using EasyQuantis:

1. Select a Quantis device from the list.
2. Select a data format:
   - **Binary data.** Data is read from the Quantis and returned as bytes.
   - **Integer numbers.** Generates 32-bit random numbers ranging between -2'147'483'648 and 2'147'483'647 (inclusive).
   - **Floating point numbers.** Generates numbers between 0.0 (inclusive) and 1.0 (exclusive).

3. Select a data separator:

   - **Comma-separated values (CSV).** CSV is a type of delimited text data, which uses a comma to separate subsequent values. The benefit of CSV is that they allow for the transfer of data across different applications.

     The following is an example of CSV:

     ```
     Value1,Value2,Value3,...,ValueN
     ```

   - **One entry per line.** Each value is on a separate line:

     ```
     Value1
     Value2
     Value3
     ...
     ValueN
     ```

Note

When generating binary data you cannot select a data separator.

4. If needed, you can scale the random values to be within a smaller range.

Note

For more details about the scaling algorithms, please refer to section 5.3.7.2.1 "Integral Values: The Scaling Algorithm".

5. Select the data destination:

   - **Display.** Data is displayed on screen. You can copy-paste the data to your application.

     Use this option for *small* amounts of random data that you want to use it right away.

Note

This option is not available for binary data.

   - **Save to file.** The data is written to a file. Use this option to generate large amounts of random data or to generate data for later use.

Note

On some systems (Mac OS X, Windows) the temporary default directory is not very convenient and should probably changed to a better suited one. This will ease the manipulation of the produced file from outside the **EasyQuantis** program.

6. Select the amount of data to generate. File size is limited to 2 GBytes (2'147'483'647 bytes).

7. Click the Generate button and wait while the application generates the random data.

4.5. The EasyQuantis Command Line

EasyQuantis v1.1 and newer includes a command line parser, allowing you to use the application from the console or in a script.

Note

Extraction functionalities are not available in command line for the moment.

4.5.1. Options

4.5.1.1. Generic Options

`-h [ --help ]`	Display a summary of available options.

4.5.1.2. Quantis Options

`-l [ --list ]`	List all devices available on the system.
`-p [ --pci ] arg`	Select the given Quantis PCI device as input device. `arg` is the number of the Quantis PCI device to use.
`-u [ --usb ] arg`	Select the given Quantis USB device as input device. `arg` is the number of the Quantis USB device to use.

4.5.1.3. Acquisition Options

`-n [ --size ] arg`	Set the number of bytes or values that should be read. If nothing is specified 1024 is used.
`-b [ --binary ] arg`	Generates a binary file. `arg` designates the filename.
`-i [ --integers ] arg`	Generates a file of integers numbers. `arg` designates the filename.
`-f [ --floats ] arg`	Generates a file of floating point numbers. `arg` designates the filename.
`-s [ --separator ] arg`	Sets the separator string for non-binary files. The default format is one entry per line.
`--min arg`	Specify the minimal value for integers and floats. If specified, requires `--max` to be specified as well.
`--max arg`	Specify the maximal value for integers and floats. If specified, requires `--min` to be specified as well.

4.5.2. Usage Examples

In this section you will find some examples of usage of the EasyQuantis command line.

4.5.2.1. Generate Binary Data

```
EasyQuantis -p 0 -b random.dat -n 1073741824
```

Generates a file named `random.dat` containing 1GByte of binary random numbers using the Quantis PCI device number 0.

4.5.2.2. Generate Numbers

```
EasyQuantis -u 0 -i integers.dat -n 1000
```

Generates a file named `integers.dat` with 1000 integer numbers.

4.5.2.3. Generate Scaled Numbers

```
EasyQuantis -u 0 -i integers.dat -n 1000 --min 1 --max 6
```

Generates a file named `integers.dat` with 1000 integer numbers whose values are between 1 and 6.

Chapter 5. The Quantis Library

To easily access the Quantis device from your application, IDQ provides an abstraction library for all supported operating systems. The library allows you to easily write your (multi-platform) application without knowing how the Quantis devices internally works.

Important

API changed with Quantis library version 2.0. If your application uses a previous Quantis library version, please read the Appendix C, *Migrating to the New API*.

Note for C++ users

Each time you request an operation, the Quantis library:

1. Opens the Quantis device;
2. Performs the requested operation;
3. Closes the Quantis device.

The C++ library wrapper has been optimized to open the Quantis device in the class constructor and close it in the class destructor, thereby leaving the connection to the device open throughout the entire execution. If your application is written in C++, it is suggested to use the C++ wrapper rather than the C wrapper. Please read Chapter 6, *Quantis Library Wrappers* for further information.

5.1. Library location

You can find the library files (`Quantis.so, Quantis.dll, Quantis.lib, ...`) in the following path:

`<path-to-quantis>\Packages\Windows\lib\<your system arch>\` for Windows users, or for Linux users after decompressing

`<path-to-quantis>/Packages/QuantisRNG-<version>-Linux-<your system arch>.tar.gz` under

`QuantisRNG-<version>-Linux-<your system arch>/lib`.

If you recompile your libraries yourself as described in later chapters, you will find the Windows libraries under

`<path-to-filename>\Libs-Apps\Quantis\<your system arch>\`

and the Linux libraries under

`<path-to-quantis>/Libs-Apps/build/Quantis`

5.2. Device Type

Almost all Quantis library functions require the device type to be specified. Currently there are two types:

- `QUANTIS_DEVICE_PCI` to specify a Quantis PCI or a Quantis PCI Express.

- `QUANTIS_DEVICE_USB` to specify a Quantis USB.

Important

Quantis PCI Express is software-compatible with Quantis PCI. There is no distinction between Quantis PCI and Quantis PCIe devices within the library and they are both considered as PCI devices.

5.3. Basic Functions

This section introduces a minimal set functions you need to use to read random data from within your application.

5.3.1. QuantisCount

```
int QuantisCount(QuantisDeviceType deviceType);
```

Returns the number of devices that have been detected. It returns 0 when no card is installed and on error.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

5.3.2. QuantisGetDriverVersion

```
float QuantisGetDriverVersion(QuantisDeviceType deviceType);
```

Returns the version of the driver as a number composed of a major and a minor version number. The value before the point represents the major version number, while the value after the point represents the minor version number.

Returns a `QUANTIS_ERROR` code (cast to float) on failure.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

5.3.3. QuantisGetLibVersion

```
float QuantisGetLibVersion();
```

Returns the version of the library as a number composed of a major and a minor version number. The value before the point represents the major version number, while the value after the point represents the minor version number.

5.3.4. QuantisGetManufacturer

```
char* QuantisGetManufacturer(QuantisDeviceType deviceType,
                             unsigned int deviceNumber);
```

Returns a pointer to the manufacturer name string of the Quantis device. Currently only the USB version supports manufacturer name retrieval. On PCI and PCI Express, the string "*Not available*" is returned.

The string "*Not available*" is returned on failure as well.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.3.5. QuantisGetModulesDataRate

```
int QuantisGetModulesDataRate(QuantisDeviceType deviceType,
                              unsigned int deviceNumber);
```

Returns the data rate (in bytes per second) provided by the Quantis device or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.3.6. QuantisGetSerialNumber

```
char* QuantisGetSerialNumber(QuantisDeviceType deviceType,
                             unsigned int deviceNumber);
```

Returns a pointer to the serial number string of the Quantis device. Currently only the USB version supports serial number retrieval. On PCI and PCI Express, the string "*S/N not available*" is returned.

The string "*S/N not available*" is returned on failure as well.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.3.7. QuantisRead

```
int QuantisRead(QuantisDeviceType deviceType,
                unsigned int deviceNumber,
                void* buffer,
                size_t size);
```

Reads random data from the Quantis device. This function perform an open, a read of the requested data and close the device.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

`buffer` a pointer to the destination buffer. The buffer **MUST** already be allocated. Its size must be **at least** `size` bytes.

`size` the number of bytes to read (cannot be larger than `QUANTIS_MAX_READ_SIZE`).

Warning

If `buffer` is not allocated or the allocated size of memory is insufficient to store the data, the library will deliver unexpected results and may even cause a crash of the enclosing application!

Note

In some situation it could be useful to control the Open and Close of a device. In this case use the QuantisOpen(), QuantisReadHandled() and QuantisClose() functions described later.

5.3.7.1. Reading Large Amounts of Data

QuantisRead is not meant to read large amount of data. The maximal size of a request is defined by `QUANTIS_MAX_READ_SIZE`. If you try reading a larger amount, `QuantisRead` will return an error. To read large amount of data you have to use a loop as in following example:

```
/* Chunk size. Recommended values are 2048 or 4096 */
chunkSize = CHUNK_SIZE;

remaining = SIZE;

while(remaining > 0u)
{
  /* Chunk size */
  if (remaining < chunkSize)
  {
    chunkSize = remaining;
  }

  /* Read data */
  result = QuantisRead(deviceType, 0, buffer, NUM_BYTES);

  /*
   * TODO:
   *  1. Check result (see example at the end of the chapter)
   *  2. Handle buffer (e.g. store data in a file)
   */

  /* Update info */
  remaining -= chunkSize;
}
```

5.3.7.2. Reading Basic Data Types

The function `QuantiRead` is useful to read a high quantity of raw random data. Depending on the application, it can be useful to be able to directly read basic data types. This section introduces functions designed for this purpose.

5.3.7.2.1. Integral Values

```
int QuantisReadShort(QuantisDeviceType deviceType,
                     unsigned int deviceNumber,
                     short* value);

int QuantisReadScaledShort(QuantisDeviceType deviceType,
                           unsigned int deviceNumber,
                           short* value,
                           short min,
                           short max);

int QuantisReadInt(QuantisDeviceType deviceType,
                   unsigned int deviceNumber,
```

```
                    int* value);

int QuantisReadScaledInt(QuantisDeviceType deviceType,
                         unsigned int deviceNumber,
                         int* value,
                         int min,
                         int max);
```

Reads a random 16-bit or 32-bit integral value. Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.
`value`	a pointer to the destination value.
`min`	the minimal value that the returned numbers can take
`max`	the maximal value that the returned numbers can take.

5.3.7.2.1.1. Integral Values: The Scaling Algorithm

Random numbers required by an application are very often in a range (much) smaller than the (fixed) range of the random number produced by Quantis.

To perform the scaling, the largest permitted multiple of the output range is selected. Random values equal or higher this limit are discarded. Below you will find a simplified version of the C code implementing `QuantisReadScaledInt` which produces an *unbiased* number between `minValue` and `maxValue` (inclusive):

```
int rnd;

/* Output range */
unsigned long long range = maxValue - minValue + 1;

/* Range of the rnd value */
unsigned long long maxRange = 2^32;

/* Largest multiple of the output range */
unsigned long long limit = maxRange - (maxRange % range);

/* Read raw random number until it is lower the limit */
do
{
  QuantisReadInt(deviceType, deviceNumber, &rnd);
} while (rnd >= limit);

/* Scale value */
value = (rnd % range) + minValue;
```

Note

This scaling algorithm wastes data when Quantis generates random values equalling or exceeding the limit. In the worst case (when `range = maxRange / 2 + 1`), the probability to drop a generated value is roughly 50%!

Warning

Raw random values are often scaled using the modulus operator, using something like:

```
minValue + (rawRndValue % (maxValue - minValue + 1))
```

where % represents the modulus operator. This formula produces a number between `minValue` and `maxValue` (inclusive), but in certain conditions (when their range is not a multiple of the output range) the distribution of these numbers has a small bias that favours numbers at the lower end of the output range.

5.3.7.2.2. Floating Point Values

```
int QuantisReadFloat_01(QuantisDeviceType deviceType,
                        unsigned int deviceNumber,
                        float* value);

int QuantisReadScaledFloat(QuantisDeviceType deviceType,
                           unsigned int deviceNumber,
                           float* value,
                           float min,
                           float max);

int QuantisReadDouble_01(QuantisDeviceType deviceType,
                         unsigned int deviceNumber,
                         double* value);

int QuantisReadScaledDouble(QuantisDeviceType deviceType,
                            unsigned int deviceNumber,
                            double* value,
                            double min,
                            double max);
```

Reads a random floating point value between 0.0 (inclusive) and 1.0 (exclusive). The scaled versions read a random floating point value between `min` (inclusive) and `max` (exclusive). Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.
`value`	a pointer to the destination value.
`min`	the minimal value that the returned numbers can take.
`max`	the maximal value that the returned number can take.

Note

Floating point values are computed by dividing a random integral value (32-bit for `floats` and 64-bit for `double`) by the integral's value range (2^{32} and 2^{64} respectively).

Warning

In certain conditions, the distribution of numbers produced by the floating point scaling algorithm has a small bias that favours numbers at the lower end of the output range. If you need unbiased random numbers, please consider to use `QuantisReadScaledShort` or `QuantisReadInt` instead:

```
/* Example: how to generate a random number between
 * 1.001 and 75.5 (inclusive)
 */
```

```
float min = 1.001;
float max = 75.5;
float multiplier = 1000.0;

int rndInt;
if (QuantisReadScaledInt(deviceType,
                         deviceNumber,
                         &rndInt,
                         (int)(min * multiplier),
                         (int)(max * multiplier)) < 0)
{
  /* Handle error */
}

float randomValue = (float)rndInt / multiplier;
```

5.3.8. QuantisOpen

```
int QuantisOpen(QuantisDeviceType deviceType,
                unsigned int deviceNumber,
                QuantisDeviceHandle** deviceHandle);
```

Open the Quantis device.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.
`deviceHandle`	a pointer to a pointer to a handle the device.

Note

This function has been implemented in the Quantis library version 2.10.

5.3.9. QuantisClose

```
void QuantisClose(QuantisDeviceHandle* deviceHandle);
```

Close the Quantis device.

Parameter:

`deviceHandle`	a pointer to a handle the device.

Note

This function has been implemented in the Quantis library version 2.10.

5.3.10. QuantisReadHandled

```
int QuantisReadHandled(QuantisDeviceHandle* deviceHandle,
                       void* buffer,
                       size_t size);
```

Read data from Quantis. This function expect *QuantisOpen()* function has been called before.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameter:

`deviceHandle`	a pointer to a handle the device.
`buffer`	a pointer to the destination buffer. The buffer **MUST** already be allocated. Its size must be **at least** `size` bytes.
`size`	the number of bytes to read (cannot be larger than `QUANTIS_MAX_READ_SIZE`).

Note

This function has been implemented in the Quantis library version 2.10.

5.3.11. QuantisStrError

```
char* QuantisStrError(QuantisError errorNumber);
```

Get a pointer to the error message string. This function interprets the value of `errorNumber` and generates a string describing the error. The returned pointer points to a statically allocated string, which may not be modified by the enclosing application. Further calls to this function will overwrite its content.

Parameters:

`errorNumber`	the number assigned to a particular type of error.

5.4. Advanced Functions

This section introduces advanced functions that allow more control over the Quantis device. Most users don't need to use these functions.

5.4.1. QuantisBoardReset

```
int QuantisBoardReset(QuantisDeviceType deviceType,
                      unsigned int deviceNumber);
```

Resets the Quantis board. Returns `QUANTIS_SUCCESS` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.

Note

You generally don't need to reset Quantis devices: the Quantis library already resets the device when needed.

5.4.2. QuantisGetBoardVersion

```
int QuantisGetBoardVersion(QuantisDeviceType deviceType,
```

```
                              unsigned int deviceNumber);
```

Returns the internal version of the board.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.4.3. QuantisGetModulesCount

```
int QuantisGetModulesCount(QuantisDeviceType deviceType,
                           unsigned int deviceNumber);
```

Returns the number of modules that have been detected on a Quantis device or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.4.4. QuantisGetModulesMask

```
int QuantisGetModulesMask(QuantisDeviceType deviceType,
                          unsigned int deviceNumber);
```

Returns a bitmask of the modules that have been detected on a Quantis device or a `QUANTIS_ERROR` code on failure.

Bit `n` is set in the bitmask if module `n` is present. For instance 5 (1101 in binary) is returned when modules 0, 2 and 3 have been detected.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.4.5. QuantisGetModulesPower

```
int QuantisGetModulesPower(QuantisDeviceType deviceType,
                           unsigned int deviceNumber);
```

Returns the power status of the modules on a device. Returns 1 if the modules are powered, 0 if the modules are not powered and a `QUANTIS_ERROR` code on failure.

Note

This function is useful only for Quantis USB devices. Modules of Quantis PCI devices are always powered, thus the function always returns 1 on such devices.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.4.6. QuantisGetModulesStatus

```
int QuantisGetModulesStatus(QuantisDeviceType deviceType,
                            unsigned int deviceNumber);
```

Returns the status of the modules on the given device as a bitmask or a `QUANTIS_ERROR` code on failure.

Bit `n` is set in the bitmask if module `n` is enabled and functional. For instance 5 (1101 in binary) is returned when modules 0, 2 and 3 are enabled and functional.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

5.4.7. QuantisModulesDisable

```
int QuantisModulesDisable(QuantisDeviceType deviceType,
                          unsigned int deviceNumber,
                          int modulesMask);
```

Disable one or more modules. Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

`modulesMask` a bitmask of the modules (as specified in the `QuantisGetModulesMask` function) that are to be disabled.

5.4.8. QuantisModulesEnable

```
int QuantisModulesEnable(QuantisDeviceType deviceType,
                         unsigned int deviceNumber,
                         int modulesMask);
```

Enable one or more modules. Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType` the type (PCI or USB) of the Quantis device.

`deviceNumber` the number of the Quantis device. Note that device numbering starts at 0.

`modulesMask` a bitmask of the modules (as specified in the `QuantisGetModulesMask` function) that are to be enabled.

5.4.9. QuantisModulesReset

```
int QuantisModulesReset(QuantisDeviceType deviceType,
                        unsigned int deviceNumber,
                        int modulesMask);
```

Reset one or more modules. Returns `QUANTIS_SUCCES` on success or a `QUANTIS_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.
`modulesMask`	a bitmask of the modules (as specified in the `QuantisGetModulesMask` function) that are to be reset.

Note

This function executes sequentially `QuantisModuleDisable` and `QuantisModuleEnable` with the given parameters.

5.5. Recompiling the Quantis Library

5.5.1. Windows compilation with Visual Studio 2008

Before compiling the Quantis libraries, you have prepare your machine described below. If you already have some of these components, you may of course skip the relevant section.

Installing the Windows Driver Kit 7.0

Download the Windows Driver Kit version 7.0.0. It comes as an .iso image. Once you've mounted that image, execute the program 'kit.exe', making sure that the kit is installed in the directory `C:\WinDDK`.

Installing the Boost C++ libraries

From the website `http://www.boostpro.com/download/`

download the file `boost_1_43_setup.exe` and run it to install Boost. You should explicitly enable the libraries "program_options" and "filesystem" libraries along with the standard distribution.

Installing Java

Please refer to the Sample Code chapter, section Java.

Installing Nokia QT 4.7

From the website `http://qt.nokia.com/downloads`

Download the QT online installer, you will need the *Qt SDK: Complete Development Environment*. Quantis has been tested with QT 4.7.3 and 4.7.4, ideally you should install one of these versions. If your version is too new or too old, and you find that you have problems with compiling or executing Quantis samples or applications, you might have to update it or to roll back as appropriate.

From the same webpage, install the QT Visual Studio Add-in.

Configuring Visual Studio

In the Menu Tools, Options: Select Projests and Solutions, VC++ Directories:

Platform: *Win32* and Show Directories for *Include files* Add the next entries at the end of the list:

```
C:\WinDDK\7600.16385.0\inc\ddk
C:\WinDDK\7600.16385.0\inc\api
C:\Program Files\Java\jdk1.6.0_25\include
C:\Program Files\Java\jdk1.6.0_25\include\win32
C:\Program Files\boost\Boost_1_43
```

Plateform *Win32* and Show directories for *Librairies files*. Add the next entries at the end of the list:

```
C:\WinDDK\7600.16385.0\lib/wxp/i386
C:\Program Files\boost\boost_1_43\lib
```

Platform: *x64* and Show Directories for *Include files* Add the next entries at the end of the list:

```
C:\WinDDK\7600.16385.0\inc\ddk
C:\WinDDK\7600.16385.0\inc\api
C:\Program Files\Java\jdk1.6.0_25\include
C:\Program Files\Java\jdk1.6.0_25\include\win32
C:\Program Files\boost\Boost_1_43
```

Plateform *x64* and Show directories for *Librairies files*. Add the next entries at the end of the list:

```
C:\WinDDK\7600.16385.0\lib\wlh\amd64
C:\Program Files\boost\boost_1_43\lib
```

Now define the Qt version: Menu Qt, Qt Options: Click Add and browse to the next path: `C:\QtSDK\Desktop\Qt\4.7.3\msvc2008`

Visual Studio requires an Environement variable for QTDIR:

In Windows Control Panel, System, Advanced system settings, click Environement variables and add the variable `QTDIR` with the value `C:QtSDK\Desktop\Qt\4.7.3\msvc2008`.

This configuration has worked for a long time on older compilers since includes were managed more automatically, but recent changes mean that the above instruction might generate errors. In case you get errors with a number of standard headers such as "stddef.h" and certain I/O headers try the following. First, add the include directory

```
C:\WinDDK\7600.16385.0\inc\crt
```

to both include file configurations. Then, again in both the include file configs, arrange the headers in the following order:

```
C:\WinDDK\7600.16385.0\inc\ddk
$(VCInstallDir)include
$(VCInstallDir)atlmfc\include
C:\WinDDK\7600.16385.0\inc\api
C:\WinDDK\7600.16385.0\inc\crt
C:\Program Files\Java\jdk1.6.0_25\include
C:\Program Files\Java\jdk1.6.0_25\include\win32
C:\Program Files\boost\Boost_1_43
$(WindowsSdkDir)include
$(FrameworkSDKDir)include
```

The ordering of header files is important since they are read sequentially. It's important to make sure that the right ones are read first in the cases where different directories contain duplicates.

If the C++ sample crashes and you get an Access Violation error, restart VS2008 in administrator mode since there is some memory that cannot be accessed by a normal user. To do that, right-click on the VS2008 icon and choose "Run in administrator mode".

Compilation

Start Visual Studio and open the Quantis library solution file located in `<path-to-Quantis>\Libs-Apps\Quantis-Libs-Apps.sln`.

In the build configuration, select your architecture, either `Win32` or `x64`.

Win32 architecture

If are on a 32b machine, you can directly perform *build solution*.

If the building of QuantisPci/QuantisUsb-Compat projects fails, make sure that the Quantis project is listed as a prerequisite. To do that, right-click on the solution, then choose "Properties->Common Properties->Project Depenencies". Choose the project from the drop-down list and add Quantis as a prerequisite.

If you want to work on the Quantis wrapper samples, don't forget to add the full path to Quantis.lib to "Properties->Configuration Properties->Linker->Input->Additional Dependencies".

x64 architecture

If you are on a 64b machine, you must make sure that not all projects of the solution are selected, because not all projects can compile in this case. It is easiest if you proceed as follows:

In the solution explorer select the next project: `Quantis, Quantis-NoHw, QuantisPci-compat, QuantisUSB-compat` with your mouse, and then click 'build'.

The EasyQuantis can not be compiled in 64 bits due to Qt is only 32 bits.

If the building of QuantisPci/QuantisUsb-Compat projects fails, make sure that the Quantis project is listed as a Project dependency. To do that in Visual Studio 2008, right-click on the solution, then choose "Properties->Common Properties->Project Depenencies" . Choose the project from the drop-down list and add Quantis as a prerequisite.

If you want to work on the Quantis wrapper samples, don't forget to add the full path to Quantis.lib to "Properties->Configuration Properties->Linker->Input->Additional Dependencies".

5.5.2. Windows Compilation with Visual Studio 2010

Installing the Windows Driver Kit 7.0

Download the Windows Driver Kit version 7.0.0. It comes as an .iso image. Once you've mounted that image, execute the program 'kit.exe', making sure that the kit is installed in the directory `C:\WinDDK`.

Installing the Boost C++ libraries

From the website `http://www.boostpro.com/download/`

download the file `boost_1_43_setup.exe` and run it to install Boost. You should explicitly enable the libraries "program_options" and "filesystem" libraries along with the standard distribution.

Installing Java

Please refer to the Sample Code chapter, section Java.

Installing Nokia QT

As of December 2011, the precompiled Qt libraries cannot be used in VS2010, so they need to be build and configured from scratch. To do this, download the zipped Qt source package from the Nokia Qt download website (as of December 2011 that is `http://qt.nokia.com/downloads`).

Once it is unzipped, go to the root directory of the unzipped folder. In the Visual Studio 2010 command shell (Start->Programs->Visual Studio 2010, then under Tools), execute the command

```
configure.exe -debug-and-release -no-webkit -no-phonon
 -no-phonon-backend -no-script -no-scripttools
 -no-qt3support -no-multimedia -no-ltcg
```

to configure the Qt build. It is important that you execute it in the VS2010 shell since it is configured in a particular way. The options of the configure command make sure that some advanced features don't get built which would make the process take much longer. You may remove them if you need these features, with the exception of the -debug-and-release flag which must be set, otherwise you cannot work in both Debug and Release mode in VS2010 with Qt. With these flags, compilation takes in the rough order of 10-15 minutes.

Once configure.exe has finished, run

```
nmake
setx QTDIR <your Qt root directory>
```

and add `<your Qt root directory>\bin`" to your `PATH` environmental variable. This can be done under Control Panel -> System -> Advanced -> Environment variables. Choose or create the `PATH` variable in 'system variables', and add the above-mentioned path to it, making sure that all entries are separated by a semicolon. The nmake command is likely to take very long, up to 20 minutes.

Now install the Qt Visual Studio Addin as described in "Windows Computer - Visual Studio 2008 -> Nokia QT". Reboot, and you're done.

If you're lucky, you may be able to read all this up on the webpage "http://stackoverflow.com/questions/5601950/how-to-build-qt-for-visual-studio-2010" or from "http://doc.qt.nokia.com/4.7/install-win.html" from where it was copied and which provides snapshots of the entire procedure for easier reading. It was available as of December 2011.

Configuring Visual Studio

The library- and include file paths are the same as those for VS 2008, see the section above. However, since VS2010 does not support solution-specific VC++ Directories, you need to set these for each single project. To do that, right-click on each project, choose "Properties->Configuration Properties->VC++ Directories" and enter the directories as above.

You need to update the path to the Qt libraries for the EasyQuantis-Setup project if they don't point to the Qt libraries you've built for VS2010. To do this, you can either open the file EasyQuantis-Setup.vdproj and edit the line pointing to QtCore4.dll / QtGui4.dll manually, or remove the files from the project in VS2010 by right-clicking on them and choosing "Remove" and adding the correct versions by right-clicking on the project and choosing "Add->File.." and choosing the correct version of the DLL.

Compilation

Start Visual Studio and open the Quantis library solution file located in `<path-to-Quantis>\Libs-Apps\Quantis-Libs-Apps.sln`.

In the build configuration, select your architecture, either `Win32` or `x64`.

Win32 architecture

If are on a 32b machine, you can directly perform *build solution*.

If the building of QuantisPci/QuantisUsb-Compat projects fails, make sure that the Quantis project is listed as a prerequisite. To do that, right-click on the solution, then choose "Properties->Common Properties->Project Depenencies". Choose the project from the drop-down list and add Quantis as a prerequisite.

If you want to work on the Quantis wrapper samples, don't forget to add the full path to Quantis.lib to "Properties->Configuration Properties->Linker->Input->Additional Dependencies".

x64 architecture

If you are on a 64b machine, you must make sure that not all projects of the solution are selected, because not all projects can compile in this case. It is easiest if you proceed as follows:

In the solution explorer select the next project: `Quantis, Quantis-NoHw, QuantisPci-compat, QuantisUSB-compat` with your mouse, and then click 'build'.

The EasyQuantis can not be compiled in 64 bits due to Qt is only 32 bits.

If the building of QuantisPci/QuantisUsb-Compat projects fails, make sure that the Quantis project is listed as a Project dependency. To do that in Visual Studio 2008, right-click on the solution, then choose "Properties->Common Properties->Project Depenencies" . Choose the project from the drop-down list and add Quantis as a prerequisite.

If you want to work on the Quantis wrapper samples, don't forget to add the full path to Quantis.lib to "Properties->Configuration Properties->Linker->Input->Additional Dependencies".

5.5.3. Linux Debian based

Prerequisites

To compile the Quantis library, you will need a certain number of packages. Unless you have already installed them, please execute the following commands:

```
$ sudo apt-get install libusb-1.0-0-dev
$ sudo apt-get install cmake
$ sudo apt-get install libboost-filesystem1.42-dev
$ sudo apt-get install libboost-date-time1.42-dev
$ sudo apt-get install libboost-program-options1.42-dev
$ sudo apt-get install libboost-thread1.42-dev
$ sudo apt-get install default-jdk (openjdk-6-jdk)
```

Now you can compile the library by executing the following commands:

```
cd <path-to-Quantis>/Libs-Apps/
mkdir build
cd build
cmake ..
```

5.5.4. Linux RedHat / CentOS

Prerequisites

To compile the Quantis library, you will need a certain number of packages. Unless you have already installed them, please execute the following commands:

```
yum install libusb-1.0-0-dev
yum install cmake
yum install libboost-filesystem1.42-dev
yum install libboost-date-time1.42-dev
yum install libboost-program-options1.42-dev
yum install libboost-thread1.42-dev
yum install default-jdk (openjdk-6-jdk)
```

Now you can compile the library by executing the following commands:

```
cd <path-to-Quantis>/Libs-Apps/
mkdir build
cd build
```

```
cmake ..
```

5.5.5. Mac OSX

This procedure is extracted from the file

`<path-to-Quantis>/Distribution/MacOS/Readme.`

Prerequisites

- Apple Developer Kit
- Nokia QT libraries
- Boost version 1.42 C++ libraries
- libusb-1.0-0 library

  To install it, execute as superuser **port install libusb**. At the time of writing, this installs the correct version of the library, but you should verify that it is indeed the 1.0 version that has been installed (the version is usually shown on the command line during the installation process).

The aforementioned libraries should ideally be compiled statically, but if this is not possible you can still proceed with the Quantis library compilation.

Compilation

- The compiler to use. E.g., to use clang, set

```
export CC=/Developer/usr/bin/clang
export CXX=/Developer/usr/bin/clang++
```

- The path to the QT libraries should be added to the PATH environment variable:

```
export PATH=${PATH}:<path-to-QT>/qtx.y.z/bin:${PATH}
```

Now you can create the package. Create the build directory as follows:

Make sure the following variables are configured:

```
cd <path-to-Quantis>/Libs-Apps
mkdir build
cd build
```

If the libraries listed above under 'Prerequisites' have been compiled statically, execute the following commands:

```
cmake ..
make
```

Otherwise, execute

```
cmake .. -DUSE_DYNAMIC_LIBS=1
make
```

5.5.6. Solaris / OpenSolaris

Prerequisites

In order to be able to compile the Quantis libraries, you need the following packages:

- libusb-1.0-0

- CMake
- libboost-filesystem1.42-dev
- libboost-date-time1.42-dev
- libboost-program-options1.42-dev
- libboost-program-options1.42-dev
- Java JDK

Compilation

To compile the Quantis libraries, choose your library destination directory (e.g. `/opt/quantis/`), and execute

```
cd /usr/local/Quantis
mkdir build
cd build
cmake .. -DDISABLE_QUANTIS_USB=1-DDISABLE_EASYQUANTIS_GUI=1
mkdir /tmp/quantis
make DESTDIR=<your-dest-directory> install
```

This will install all the Quantis libraries in your destination directory.

5.5.7. FreeBSD

For FreeBSD, the library is created and managed by the FreeBSD developers, not by ID Quantique. We can therefore not provide a recompilation procedure.

Chapter 6. Quantis Library Wrappers

IDQ provides several wrappers to allow you to use the Quantis device with your preferred programming language.

Currently wrappers for the following languages are available:

- C++
- C#
- Java
- VB.NET

The wrappers are for Object-oriented programming languages and they all have the same structure:

- The class is named `Quantis`.
- On class instantiation, you must provide the `deviceType` and the `deviceNumber`.
- The names of public functions in the Wrappers are the same as those in the Quantis C library but without the prefix *Quantis*, for instance `QuantisCount` is named `Count` in the wrapper. Functions other than the constructors do not require providing values for `deviceType` and `deviceNumber` since these are defined globally within the class. The only exception to this rule are static functions which have the same definition in both the C- and the Wrapper code.

Please refer to the sample available with each wrapper for further details.

Furthermore, Quantis can be accessed via the C++11 "random_device" interface which standardises true random number generator access. Usage of this interface is described below.

Note for the C++ Wrapper

Since the Quantis device is kept open until the Quantis class is destroyed, it is highly recommended to reduce the scope of the Quantis variable as much as possible. In particular it is discouraged to make the Quantis variable global.

6.1. The C++11 random_device interface

6.1.1. About the interface and our implementation

The standard C++11 "random_device" interface allows to access true random number generators in a standarised manner. It is derived from the boost "random_device" class, so if your application uses either the C++11 or the boost version of the random_device class, you can switch to Quantis very easily by including the Quantis implementation - i.e. the file "Quantis_random_device.h" - in your code and commenting your previous include, and making a very small number of changes described in the following.

At the time of releasing this interface, C++11 is a very new standard. Many compilers have already implemented parts of it, but which parts are supported varies widely. In order to avoid compile-time issues, the C++11-specific keywords have been commented, and you should uncomment those that your compiler supports in order to be as compatible as possible.

Note that our implementation is a little different from the standard one in that the standard interface accesses a device mounted on the file system, while we must create a Quantis C++ object and access

it to get to Quantis. For this reason, we have included a destructor in our interface that you should call when you no longer need Quantis.

A sample executable in your code shows how to use this interface.

6.1.2. Library Compilation for C++11

It is important to not to compile EasyQuantis when compiling with C++11. If you want to compile both, do two separate compilations.

If you want to test the interface without C++11 support, i.e. with the keywords suppressed, simply compile it as usual, with no options given to cmake. The sample will work either way.

Important

For this feature to work, your compiler must support the C++11 standard. Many compilers support parts of it but not all keywords needed in the "random_device" interface. It may thus happen that even though your compiler has a C++0x/C++11 option, the full feature version will still not be run. In this case, you can still use the "random_device" interface, but the C++11-specific keywords are automatically suppressed to pass compilation in C++98. In most applications using "random_device", it should still be easy to replace your old implementation with Quantis even if you don't have C++11 supported.

The full version of this feature is only supported on Linux and MacOSX systems. On the other supported OSs, the interface of the class will be available along with the Sample code for C++11, but the C++11 specific keywords will be automatically suppressed. Again, this means that you can use the class, but it is not compatible with the C++11 standard in the strict sense. In most applications using "random_device", it should still be easy to replace your old implementation with Quantis even if you don't have C++11 supported.

If you use cmake and the gnu C/C++ compiler version 4.6 or newer, it is possible that the features will work for you even on other systems than Linux and MacOSx.

6.1.2.1. Windows

The Windows Visual Studio 2008/2010 compilers do not support all the necessary C++11 features, so if you want to user C++11 under Windows you must revert to a compilation as under Linux using CMake and GCC 4.6 (see below).

6.1.2.2. Linux

The distribution available by default uses C++98, so you need to recompile the library to use C++11. You will need to have installed CMake and GCC 4.6 or higher. Proceed as follows:

```
cd <your-path-to-Quantis>/Libs-Apps/
mkdir build
cd build
cmake .. -DUSE_CXX11=1 -DDISABLE_EASYQUANTIS=1
make
```

6.1.2.3. Mac OSX

On Solaris, you should use the clang/clang++ compiler. Define the two environment variables CC and CXX in the following way before you call CMake:

```
export CC=clang
export CXX=clang++
```

```
cd <your-path-to-Quantis>/Libs-Apps/
mkdir build
cd build
cmake .. -DUSE_CXX11=1 -DDISABLE_EASYQUANTIS=1
make
```

6.1.2.4. Solaris / OpenSolaris

On Solaris, you may choose whether to compile using GCC or using SunStudio. On a typical system with both options installed, GCC is the default. To use SunStudio, define the two environment variables CC and CXX in the following way before you call CMake:

```
export CC=cc
export CXX=CC
cd <your-path-to-Quantis>/Libs-Apps/
mkdir build
cd build
cmake .. -DUSE_CXX11=1 -DDISABLE_EASYQUANTIS=1
make
```

6.1.2.5. FreeBSD

On FreeBSD, use CMake and make to compile in the following way:

```
cd <your-path-to-Quantis>/Libs-Apps/
mkdir build
cd build
cmake .. -DUSE_CXX11=1 -DDISABLE_EASYQUANTIS=1
make
```

6.1.3. C++11 Sample compilation

For how to compile the C++11 Sample, see the relevant section in the chapter "Sample Code" below.

Chapter 7. The QuantisExtensions Library

The QuantisExtensions library is a set of functions to add extra processing to the random data. For the moment it can provide randomness extraction.

IDQ provides an abstraction library written in C language to ensure support over various operating systems.

Note

For the moment only Windows and Linux operating system are supported. The library also comes with a C++ wrapper. Other languages wrappers are not available for the moment.

7.1. Extractor

The extraction processing allow to improve the randomness of the data.

The Samples directory show how to use the QuantisExtractor functions.

Note

Randomness extraction algorithm use 64 bits integer data type. Using a 64 bits platform improve dramatically the performances.

7.1.1. Basic Functions

This section introduces a minimal set of functions you need to use to process the randomness extraction.

7.1.1.1. QuantisExtractorGetLibVersion

```
float QuantisExtractorGetLibVersion();
```

Returns the version of the library as a number composed of a major and a minor version number. The value before the point represents the major version number, while the value after the point represents the minor version number.

7.1.1.2. QuantisExtractorInitializeMatrix

```
int32_t QuantisExtractorInitializeMatrix(const char* matrixFilename,
                                         uint64_t** extractorMatrix,
                                         uint16_t matrixSizeIn,
                                         uint16_t matrixSizeOut);
```

Reads the extractor matrix from the specified file and store in memory. This function allocate the required memory space. Use *QuantisExtractorUnitializeMatrix* to free memory.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_EXT_ERROR` code on failure.

Parameters:

`matrixFilename` — a pointer to a char array representing the path and the filename of the extractor matrix.

`extractorMatrix`	a pointer to a pointer where the matrix is stored in memory.
`matrixSizeIn`	the number of bits which are input to the extractor
`matrixSizeOut`	the number of bits which are output to the extractor

7.1.1.3. QuantisExtractorUnInitializeMatrix

```
void QuantisExtractorUnInitializeMatrix(uint64_t** extractorMatrix);
```

Free the allocated memory from *QuantisExtractroInitializeMatrix* function.

Parameters:

`extractorMatrix`	a pointer to a pointer where the matrix is stored in memory.

7.1.1.4. QuantisExtractorGetDataFromQuantis

```
int32_t QuantisExtractorGetDataFromQuantis(QuantisDeviceType deviceType,
                                           unsigned int deviceNumber,
                                           uint8_t* outputBuffer,
                                           uint32_t numberOfBytesRequested,
                                           const uint64_t* extractorMatrix);
```

Reads the random data from the specified Quantis device and apply randomness extraction processing.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_EXT_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.
`outputBuffer`	a pointer to the destination buffer. The buffer **MUST** already be allocated. Its size must be **at least** `numberOfBytesRequested` bytes.
`numberOfBytesRequested`	the number of bytes to read
`extractorMatrix`	a pointer where the matrix is stored in memory.

7.1.1.5. QuantisExtractorGetDataFromFile

```
int32_t QuantisExtractorGetDataFromFile(char* inputFilePath,
                                        char* outputFilePath,
                                        const uint64_t* extractorMatrix);
```

Apply randomness extraction to data coming from an input file and save the result to an output file.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_EXT_ERROR` code on failure.

Parameters:

`inputFilePath`	a pointer to a char array representing the path and the filename of the input file to process.
`outputFilePath`	a pointer to a char array representing the path and the filename of the output file to save the data after extraction.
`extractorMatrix`	a pointer where the matrix is stored in memory.

7.1.1.6. QuantisExtractorGetDataFromBuffer

```
void QuantisExtractorGetDataFromBuffer(const uint8_t* inputBuffer,
                                       uint8_t* outputBuffer,
                                       const uint64_t* extractorMatrix,
                                       uint32_t nbrBytesAfterExtraction);
```

Apply randomness extraction to data stored in an input buffer in the memory and put the result in an output buffer in memory.

Parameters:

`inputBuffer`	a pointer to an allocated input buffer.
`outputBuffer`	a pointer to an allocated output buffer.
`extractorMatrix`	a pointer where the matrix is stored in memory.
`nbrBytesAfterExtraction`	the number of processed bytes which should be produced.

7.1.1.7. QuantisExtractorStrError

```
char* QuantisExtractorStrError(QuantisExtractorError errorNumber);
```

Reads the extractor matrix from the specified file and store in memory. This function allocate the required memory space. Use *QuantisExtractorUnitializeMatrix* to free memory.

Return a pointer to an array representing an human readable error message.

Parameters:

`errorNumber`	The error number.

7.1.2. Advanced Functions

This section introduces advanced functions that allow more control more finely.

If you require to:

- Get a progression status during the processing, you may require to run the randomness extraction in a thread.
- Create you own matrix file.
- Get multiple time small amount of random data and you want to increase the speed of processing using a temporary buffer.

7.1.2.1. QuantisExtractorGetMatrixSizeIn

```
uint16_t QuantisExtractorGetMatrixSizeIn();
```

Returns the input size in bits of the matrix. See *QuantisExtractorInitializeMatrix*.

7.1.2.2. QuantisExtractorGetMatrixSizeOut

```
uint16_t QuantisExtractorGetMatrixSizeOut();
```

Returns the output size in bits of the matrix. See *QuantisExtractorInitializeMatrix*.

7.1.2.3. QuantisExtractorComputeBufferSize

```
int32_t QuantisExtractorComputeBufferSize(uint32_t numberOfBytesRequested,
```

```
                                        uint32_t* numberOfBytesAfterExtraction,
                                        uint32_t* numberOfBytesBeforeExtraction);
```

Get the parameters for reading the bytes to perform randomness extraction

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_EXT_ERROR` code on failure.

Parameters:

`numberOfBytesRequested` the number of bytes to read (not larger than QUANTIS_MAX_READ_SIZE).

`numberOfBytesAfterExtraction` the number of bytes that will be output after the extraction.

`numberOfBytesBeforeExtraction` the number of bytes required before the extraction.

7.1.2.4. QuantisExtractorInitializeOutputBuffer

```
int32_t QuantisExtractorInitializeOutputBuffer(uint32_t inputBufferSize,
                                               uint8_t** outputBuffer);
```

The function initializes an output buffer in order to contain the result of the extraction of an input buffer of inputBufferLength bytes. Use *QuantisExtractorUninitializeOutputBuffer* to free memory.

Return the number of bytes that should be obtained after the extraction, a QUANTIS_EXT_ERROR otherwise

Parameters:

`inputBufferSize` the number of bytes contained in the input buffer

`outputBuffer` a pointer to a pointer where the output buffer is stored in memory.

7.1.2.5. QuantisExtractorUninitializeOutputBuffer

```
void QuantisExtractorUninitializeOutputBuffer(uint8_t** outputBuffer);
```

This function free the memory allocated by *QuantisExtractorInitializeOutputBuffer*.

Parameters:

`outputBuffer` a pointer to a pointer where the output buffer is stored in memory.

7.1.2.6. QuantisExtractorProcessBlock

```
void QuantisExtractorProcessBlock(const uint64_t* inputBuffer,
                                  uint64_t* outputBuffer,
                                  const uint64_t* extractorMatrix);
```

Apply extraction processing function to a block of matrixSizeIn bits and produce matrixSizeOut processed bits.

Parameters:

`inputBuffer` a pointer to the input of the extractor.

`outputBuffer` a pointer to the output of the extractor.

`extractorMatrix` a pointer where the matrix is stored in memory.

7.1.2.7. QuantisExtractorMatrixCreate

```
int32_t QuantisExtractorMatrixCreate(uint32_t nbrElementaryMatrices,
                                     uint32_t nbrBytesToXor,
                                     char* elementaryMatrixFilenames[],
                                     char* extractorMatrixFilename);
```

XOR the bitstreams contained in the nbrElementaryMatrices files specified in elementaryMatricesFilenames.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_EXT_ERROR` code on failure.

Parameters:

`nbrElementaryMatrices`	number of elementary matrices that should be XORed.
`nbrBytesToXor`	number of bytes that should be XORed
`elementaryMatrixFilenames`	a pointer to a char array containing the name of the files where the different elementary matrices are stored.
`extractorMatrixFilename`	a pointer to a char array representing the path and the filename of the output file to save the matrix.

7.1.2.8. QuantisExtractorMatrixCreateElementary

```
int32_t QuantisExtractorMatrixCreateElementary(QuantisDeviceType deviceType,
                                               unsigned int deviceNumber,
                                               uint16_t matrixSizeIn,
                                               uint16_t matrixSizeOut,
                                               uint16_t underSamplingPeriod,
                                               char* elementaryMatrixFilename);
```

Write to file an elementary matrix created by applying *QuantisExtractorMatrixUnderSamplingRead* to the buffer produced by *QuantisExtractorMatrixUnderSamplingRead*.

Returns `QUANTIS_SUCCES` on success or a `QUANTIS_EXT_ERROR` code on failure.

Parameters:

`deviceType`	the type (PCI or USB) of the Quantis device.
`deviceNumber`	the number of the Quantis device. Note that device numbering starts at 0.
`matrixSizeIn`	the number of bits which are input to the extractor
`matrixSizeOut`	the number of bits which are output to the extractor
`underSamplingPeriod`	number of bytes to down sampled. should >= 13
`elementaryMatrixFilename`	a pointer to a char array representing the path and the filename of the output file to save the data after extraction.

7.1.2.9. QuantisExtractorMatrixUnderSamplingRead

```
int32_t QuantisExtractorMatrixUnderSamplingRead(QuantisDeviceType deviceType,
                                                unsigned int deviceNumber,
                                                uint32_t nbrOfBytesRequested,
                                                uint16_t underSamplingPeriod,
                                                uint8_t* sampledBuffer);
```

Sample the *QuantisRead* output according to `underSamplingPeriod` and save the output stream in `sampledBuffer`

Return the number of read sampled bytes if success, QUANTIS_EXT_ERROR otherwise.

Parameters:

deviceType	the type (PCI or USB) of the Quantis device.
deviceNumber	the number of the Quantis device. Note that device numbering starts at 0.
nbrOfBytesRequested	number of bytes which should be output to the user after the sampling of the data read from the Quantis
underSamplingPeriod	number of bytes to down sampled. should >= 13
sampledBuffer	a pointer to a buffer where to store the sampled sequence.

7.1.2.10. QuantisExtractorMatrixProcessBufferVonNeumann

```
uint32_t QuantisExtractorMatrixProcessBufferVonNeumann(uint8_t* inputBuffer,
                                                       uint8_t* outputBuffer,
                                                       uint32_t inputBufferSize);
```

Apply extraction processing function to a block of matrixSizeIn bits and produce matrixSizeOut processed bits.

Return the number of bytes at the output of the Von Neumann processing.

Parameters:

inputBuffer	a pointer to the input buffer to process.
outputBuffer	a pointer to the output buffer. The buffer should be already initialized by the function which invokes this method; please note that the number of bytes that will be output by the Von Neumann processing is not deterministic, as it depends on the input sequence, but it can never exceed 1/2 of the input buffer size.
inputBufferSize	the size of the input buffer.

7.1.2.11. QuantisExtractorStorageBufferEnable

Enable the storage buffer and allocate MAX_STORAGE_BUFFER_SIZE bytes for it.

```
int32_t QuantisExtractorStorageBufferEnable();
```

Returns QUANTIS_SUCCESS if enabling is successful, QUANTIS_EXT_ERROR otherwise.

7.1.2.12. QuantisExtractorStorageBufferDisable

Disable the storage buffer (if it was previously activated) and free the allocated memory.

```
int32_t QuantisExtractorStorageBufferDisable();
```

Returns QUANTIS_SUCCESS if enabling is successful, QUANTIS_EXT_ERROR otherwise.

7.1.2.13. QuantisExtractorStorageBufferClear

Reset the storage buffer (free and reallocate memory).

```
int32_t QuantisExtractorStorageBufferClear();
```

Returns QUANTIS_SUCCESS if enabling is successful, QUANTIS_EXT_ERROR otherwise.

7.1.2.14. QuantisExtractorStorageBufferSet

Set the first bytesToCopy bytes of the storage buffer to the bytesToCopy bytes pointed by bufferToCopy. Any data previously written in the storage buffer will be overwritten If the buffer to copy is bigger than MAX_STORAGE_BUFFER_SIZE, extra data will be dropped

```
int32_t QuantisExtractorStorageBufferSet(uint8_t* bufferToCopy,
                                         uint32_t bytesToCopy);
```

Returns always QUANTIS_SUCCESS.

Parameters:

bufferToCopy	pointer to the buffer which should be copied into the storage buffer.
bytesToCopy	number of bytes in bufferToCopy to be copied into the storage buffer.

7.1.2.15. QuantisExtractorStorageBufferAppend

Append bytesToAppend bytes of the bufferToCopy to the storage buffer Existing data in the storage buffer will be preserved Please note that appendable bytes are limited by

MAX_STORAGE_BUFFER_SIZE, i.e. bytes are appended as long as the storage buffer is not full.

```
int32_t QuantisExtractorStorageBufferAppend(uint8_t* bufferToCopy,
                                            uint32_t bytesToCopy);
```

Return the number of actually appended bytes.

Parameters:

bufferToCopy	pointer to the buffer which should be copied into the storage buffer.
bytesToCopy	number of bytes in bufferToCopy to be copied into the storage buffer.

7.1.2.16. QuantisExtractorStorageBufferRead

Read numberOfBytesRequested bytes from the storage buffer.

```
int32_t QuantisExtractorStorageBufferRead(uint8_t* outputBuffer,
                                          uint32_t numberOfBytesRequested);
```

Return QUANTIS_SUCCESS if reading is successful, QUANTIS_EXT_ERROR otherwise.

Parameters:

outputBuffer	a pointer to the buffer where the read bytes should be written.
numberOfBytesRequested	number of bytes to read.

7.1.2.17. QuantisExtractorStorageBufferGetSize

Get the number of bytes which has been written into the buffer.

```
int32_t QuantisExtractorStorageBufferGetSize();
```

Return the number of bytes in the storage buffer.

7.1.2.18. QuantisExtractorStorageBufferIsEnabled

Return the state (enabled/disabled) of the storage buffer.

```
int8_t QuantisExtractorStorageBufferIsEnabled();
```

Return 0 = disabled, 1 = enabled.

Chapter 8. Sample Code

You will find sample code on how to use each library wrapper in the subdirectory `<path-to-Quantis>/Samples/`. In this chapter, we give a few further examples and explanations concerning compilation and usage.

8.1. Windows Compilation / Execution

On Windows, use Visual Studio 2008 or Visual Studio 2010. Open the solution file (extension .sln) for the sample you wish to compile, e.g. if you want to use the C# example, open the file `QuantisDemo.sln`.

- For **VB.Net** and **C#**, you don't need to do any further configuration, just build the solutions. You will get an executable called `QuantisDemo.exe`, which is located in the same directory as the solution file. Double-click on it to execute it.
- For the **C/C++** samples, proceed as follows: Make sure to add the path `<path-to-Quantis>\Libs-Apps\Quantis`

  to your VC++ include directories and

  `<path-to-Quantis>\Libs-Apps\Quantis\<your system arch>`

  to your VC++ library directories. Also, add `<path-to-Quantis>\Libs-Apps\Quantis\<your system arch>\Quantis.lib`

  to your "Linker input additional dependencies" in Visual Studio. Now build the solution. This will create an executable called `QRNG.exe`. Execute it from your command prompt as follows:

  ```
  QRNG.exe -<device type> <device number>
  ```

  where you substitute the type by "u" or "p" for USB-device or "PCI device", and the number by your device number.
- The **Java** Sample compilation is described further below since it is system independent.
- The **C++11** sample can in principle be executed similarly to the C++ sample, but since the necessary features of C++11 are not yet enabled in VS2008 or in VS2010, it will be executed as normal C++ code. To use the actual C++11 features, you would need to recompile the Quantis library. Further details on how to do this can be found under the section "The C++ random_device interface".

8.1.1. Visual Studio 2008 vs. 2010

The Samples available for Visual Studio have been created under VS2008, but they are also compatible with 2010.

8.2. Linux / Solaris / OpenSolaris / FreeBSD compilation and execution

- The **VB.Net** and **C#** samples can't be compiled on Unix-based systems.
- For the **C** and **C++** samples, type

  ```
  cd <path-to-quantis>/Samples/<chosen language>
  make
  ./qrng -<device type> <device number>
  ```

  <device type> is "u" if you're using a Quantis USB and "p" if you're using a PCI/PCIe device.

- For the **Java** sample, see the section "Java Sample" below.
- The **C++11** sample can in principle be executed similarly to the C++ sample, but if you use it with the library provided it will be executed as normal C++ code. To use the actual C++11 features, you would need to recompile the Quantis library. Further details on how to do this can be found under the section "The C++ random_device interface".

8.2.1. Mac OSX

- The **VB.Net** and **C#** samples can of course not be compiled on Mac OSX systems.
- For the **C** and **C++** samples, type

```
cd <path-to-quantis>/Samples/<chosen language>
make OS=Darwin
./qrng -<device type> <device number>
```

  <device type> is "u" if you're using a Quantis USB and "p" if you're using a PCI/PCIe device.
- For the **Java** sample, see the relevant section below.
- The **C++11** sample can in principle be executed similarly to the C++ sample, but if you use it with the library provided it will be executed as normal C++ code. To use the actual C++11 features, you would need to recompile the Quantis library. Further details on how to do this can be found under the section "The C++ random_device interface".

8.3. C Sample

The following is a simple example of usage of the Quantis library:

```
/* Global includes */
#include <stdio.h>
#include <stdlib.h>

/* Includes Quantis library's header */
#include "Quantis.h"

/* Define the number of bytes that should be read */
#define NUM_BYTES 100

int main()
{
  QuantisDeviceType deviceType;
  unsigned char* buffer;
  int result;
  int i;

  /* Select device type */
  if (QuantisCount(QUANTIS_DEVICE_PCI) > 0)
  {
    /* There is one ore more Quantis PCI device... */
    deviceType = QUANTIS_DEVICE_PCI;
  }
  else if (QuantisCount(QUANTIS_DEVICE_USB) > 0)
  {
    /* There is one ore more Quantis USB device... */
    deviceType = QUANTIS_DEVICE_USB;
  }
  else
  {
    /* No Quantis device has been found on the system */
    printf("No Quantis device found\n");
    return -1;
  }

  /* Allocate buffer's memory */
  buffer = (unsigned char*)malloc(NUM_BYTES);
```

```
  if (!buffer)
  {
    fprintf(stderr, "Unable to allocate memory\n");
    return -1;
  }

  /* Read random data from the Quantis*/
  result = QuantisRead(deviceType, 0, buffer, NUM_BYTES);
  /* Check if there are some errors */
  if (result < 0)
  {
    /* An error occured. Print the error message */
    fprintf(stderr,
            "An error occured when reading random bytes: %s\n",
            QuantisStrError(result));
    goto cleanup;
  }
  else if (result != NUM_BYTES)
  {
    /* Quantis did not return the number of bytes asked */
    fprintf(stderr,
            "Asked to read %d byts but received %d bytes\n",
            NUM_BYTES,
            result);
    goto cleanup;
  }

  /* Display buffer in HEX format */
  printf("Displaying %d random bytes in HEX format:\n",
         NUM_BYTES);
  for(i = 0; i < NUM_BYTES; i++)
  {
    printf("%02x ", buffer[i]);
  }
  printf("\n");

  /* Cleanup */
cleanup:

  if(buffer)
  {
    free(buffer);
  }

  return 0;
}
```

A more detailed example is available on the USB flash drive in the `Samples` directory.

8.4. C++ Sample

Here is the example presented in the previous chapter modified to use the C++ Wrapper:

```
/*

Compile like this (if Quantis software is installed under
 "/opt/IDQQuantis").

g++ -L/usr/lib -L/opt/IDQQuantis/lib
-I/opt/IDQQuantis/include -l Quantis
-o quantis_osx quantis_osx_wrapper.cpp

*/

/* Global includes */
#include <iomanip>
#include <iostream>
#include <cstdlib>
#include <string>
/* Includes Quantis library's header */
```

```
/* Note the hpp extension! */
#include "Quantis.hpp"

/* Define the number of bytes that should be read */
#define NUM_BYTES 100
using namespace std;
using namespace idQ;


int main()
{
    QuantisDeviceType deviceType;
    int result;

    /* Select device type */
    if (Quantis::Count(QUANTIS_DEVICE_PCI) > 0)
    {
        /* There is one ore more Quantis PCI device... */
        deviceType = QUANTIS_DEVICE_PCI;
    }
    else if (Quantis::Count(QUANTIS_DEVICE_USB) > 0)
    {
        /* There is one ore more Quantis USB device... */
        deviceType = QUANTIS_DEVICE_USB;
    }
    else
    {
        /* No Quantis device has been found on the system */
        cout << "No Quantis device found" << endl;
        return -1;
    }
    try
    {
        /* Creates a quantis object */
        Quantis quantis(deviceType, 0);
        /* Read random data from the Quantis*/
        string buffer = quantis.Read(NUM_BYTES);
        if (buffer.length() != NUM_BYTES)
        {
            /* Quantis did not return the number of bytes asked */
            cerr << "Asked to read " << NUM_BYTES
                 << " byts but received " << buffer.length()
                 << " bytes" << endl;
            return -1;
        }
        // Display buffer in HEX format
        cout << "Displaying " << NUM_BYTES
             << " random bytes in HEX format:" << endl;
        string::iterator it = buffer.begin();
        while (it != buffer.end())
        {
            cout << setw(2) << setfill('0') << hex
                 << static_cast<int>(static_cast<unsigned char>(*it++))
                 << " ";
        }
        cout << endl;
        return 0;
    }
    catch (runtime_error &ex)
    {
         cerr << "Error while accessing Quantis device: "
              << ex.what() << endl;
         return -1;
    }
}
```

8.5. Java Sample

Install Java and Apache Ant - All OS

To use the Java wrapper, install the Java Standard Edition (JSE) JDK matching your OS. For instance, the Java SE 64 bits JDK should be installed with a Windows 64 bits. You will find the JSE JDK in all available versions and a detailed installation instruction on the website of OpenJDK or on the webpage of Oracle (at the time of writing this guide, `http://openjdk.java.net/` and `http://www.oracle.com/technetwork/java/javase/overview/index.html` respectively).

In addition to Java, Apache Ant should be installed. The Apache Ant website contains a user manual that explains how to install it on your system (at the time of writing, `http://ant.apache.org`).

From here, go to the section describing your OS.

Linux

In order to be able to work with Java and Ant, you need to add their binaries to the PATH system variable, if they aren't there yet. Additionally, Java needs you to set the JAVA_HOME variable. You can set these variables as follows:

Open the file `~/.bashrc` (on some systems this may be called `~/.bash_profile`). This is a script that will run automatically and set the variables in it for you, so you won't need to set them by hand every time you open a new terminal.

In the file, add the following entry on a separate line:

```
export PATH=$PATH:<path-to-java-bin-directory>:
<path-to-ant-bin-directory>
```

To find out where your Java bin directory is, type in your terminal

```
which java
```

which will result in something like `/usr/bin/java`. Your <path-to-java-bin-directory> is therefore `/usr/bin`. The same procedure applies to finding your ant directory. To apply the changes, execute the command

```
source ~/.bashrc
```

To compile the Java samples, the easiest solution is to

- Change to the directory where the Java samples are located:

  ```
  cd <path-to-quantis>/Samples/Java
  ```

- In order for Java to be able to use the Quantis library, execute the following command which will add the Quantis library to the Java library path:

  ```
  export
  LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<directory-containing-libQuantis.so>
  ```

  The location of the Quantis library is explained in section "Library location" in the chapter "The Quantis Library".

  You may also add this line to the `~/.bashrc` file as you did before when adding Java and Ant to the PATH variable, if you want it to be loaded automatically when you open a new shell.

- Type the command **ant** . The Java archive created is `dist/Quantis.jar`.

- To execute the program, pass the command **java -jar dist/Quantis.jar** . Alternatively, type **ant run**.

Windows

In order to be able to work with Java and Ant, you need to add their binaries to the PATH system variable, if they aren't there yet. Additionally, you need to add the directory containing the Quantis libraries to PATH as well.

You can set it in Control Panel -> System Properties -> Tab 'Advanced' -> Environment Variables, note that usually you have to be Administrator user to do this. Choose or create the variable `PATH` (upper case important) under 'System Variables', and add `<path-to-ant>\bin;<path-to-java>\bin;<path-to-Quantis-libs`, making sure that all entries are separated by a semicolon. To find your Quantis libraries, refer to chapter "The Quantis Library", section "Library Location". E.g., if you have decompressed the Ant folder in the location `C:\Program Files\Ant\apache-ant-x.y.z`, your Java JDK is under `C:\Program Files\Java\jdk-xxx` and your Quantis libraries are under `<path-to-Quantis\Packages\Windows\lib\Win32`, you should add the following line:

```
C:\Program Files\Ant\apache-ant-x.y.z;C:\Program Files\Java\jdk-xxx; <path-to-Quantis\Packages\Windows\lib\Win32
```

Important

On some 64b Windows systems, you may have to recompile the Quantis libraries before executing the Java sample, in order to avoid JNI-related errors. For how to do this, refer to chapter "The Quantis Library", section "Recompiling the Quantis Library". When you have done that, you will need to update the entry in the `PATH` variable pointing to the Quantis libraries. The section "Library Location" tells you where you can find your recompiled Quantis libraries.

Now you can execute the Java Sample:

```
cd <path-to-Quantis>\Samples\Java
ant
ant run
```

Instead of **ant run** you can also execute **java -jar dist\Quantis.jar**.

Appendix A. Troubleshooting

A.1. EasyQuantis

A.1.1. EasyQuantis crashes on Linux, with one of the following errors:

- *Segmentation fault.*
- *symbol lookup error: EasyQuantis: undefined symbol: _ZN14QPlainTextEditC1EP7QWidget.*

Such errors are generally caused by an incompatible Qt library binary. The binary of EasyQuantis provided by ID Quantique has been linked against Qt version 4.3.4. To solve this issue you need to install Qt4 version 4.3.4 or newer.

If Qt version 4.3.4 (or newer) is not available on your system, you can still use EasyQuantis in command line mode, which is not affected by this issue. Please refer to Section 4.5, “The EasyQuantis Command Line”.

This issue can also be solved by recompiling the Quantis library and EasyQuantis on your system.

A.2. Quantis Samples

A.2.1. When I try to run the C++ sample code after a successful build, it crashes and gives me an Access Violation error. What should I do?

If the C++ sample crashes upon build and you get an Access Violation error, restart VS2010 in administrator mode since there is some memory that cannot be accessed by a normal user. To do that, right-click on the VS2010 icon and choose "Run in administrator mode". Then reopen your solution/project and proceed as normal.

A.2.2. I am on a Windows 64 bits machine. When I execute the Java sample, I get JNI-related errors. What should I do?

This error occurs on some Windows 64 bits systems, and can be solved by recompiling the Quantis libraries on your own machine. Instructions for recompiling the library can be found in chapter "The Quantis Library" in section "Recompiling the Quantis Library".

Appendix B. Frequently Asked Questions (FAQ)

B.1. Quantis Library

B.1.1. Can I use the 32-bit Quantis library on a 64-bit system?

Yes, you can use the 32-bit Quantis library within a 32-bit application on 64-bit systems. Note however that you can neither use the 32-bit Quantis library within a 64-bit application nor the 64-bit Quantis library within a 32-bit application.

B.1.2. On Microsoft Windows, is it necessary to copy the `Quantis.dll` library to the system directory (`C:\Windows\System32`)?

No, this is not mandatory. IDQ recommends to install the `Quantis.dll` library in the directory in which your application resides.

B.1.3. On Microsoft Windows, when I use `Quantis.dll` within my application I get the error "*The application has failed to start because WINUSB.DLL was not found. Re-installing the application may fix this problem*". What should I do?

This problem occurs with `Quantis.dll` v2.1 (and older) when the Quantis USB driver is not installed. This issue has been fixed in `Quantis.dll` v2.2. Please update your `Quantis.dll` to the latest available version.

B.1.4. I have changed the name of the Java package Java for Quantis from *com.idquantique.quantis* to something else/I have moved the Java code to another directory. Now I can load the dynamic library `Quantis.dll` but I get the error message

> Exception in thread "main" java.lang.UnsatisfiedLinkError: random.utils.Quantis.QuantisCount(I)I.

The Java classes use mainly native functions (by using the JNI interface). A native function has to obey to strict rules, and if you don't abide by them your application may not execute or compile properly.

For instance, the name of the package your code resides in is used to define the name of functions in that package. So if you change the package name, say to *random.quantis*, the function names will change in the Java code, but not match the native code function names anymore and thus produce errors. The best solution is to keep the original name *com.idquantique.quantis* for the package.

If you absolutely need the Java code in another package, change the two files `Libs-Apps/Quantis/Quantis_Java.h` and `Libs-Apps/Quantis/Quantis_Java.cpp` to reflect the new package name in function names. After the modification, the dynamic `Quantis` library has to be recompiled and reinstalled.

B.2. EasyQuantis

B.2.1. On Microsoft Windows, when I launch EasyQuantis I have the error *"The application has failed to start because WINUSB.DLL was not found. Re-installing the application may fix this problem*". What should I do?

This problem occurs with EasyQuantis 1.0 when the Quantis USB driver is not installed. This issue has been fixed in EasyQuantis 1.1. Please update EasyQuantis to the latest available version.

B.2.2. When I launch EasyQuantis on Microsoft Windows, a console appears for a few seconds and disappears when the GUI window comes up. Why does this happen?

EasyQuantis integrates a command line interface and a graphical interface. However, on Microsoft Windows it is not possible to build an hybrid Windows/Console application. EasyQuantis has been built as a Console application. When launched, the system automatically creates a console window. If no argument has been provided to the application (giving arguments would invoke the console version), the console window is hidden and the graphical interface is displayed. Avoiding this issue is very difficult.

Appendix C. Migrating to the New API

The Quantis library version 2.0 has a slightly different API than its predecessors. This is mainly due to the merge of the old Quantis library (used to access Quantis PCI devices) and Quantis-USB library (used to access Quantis USB devices) into a single library.

The main difference between versions 1.x and 2.0 is the addition of the parameter `deviceType`, which allows you to specify the type of device to use (PCI/PCIe or USB). Additionally, functions names have been modified when ambiguous. See Table C.1, "API 1.x and 2.0 functions equivalences." for equivalences between API 1.x and 2.0.

API 1.x functions	API 2.0 functions

`int quantisBoardReset(`
`  int cardNumber);`
	`int QuantisBoardReset(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`int quantisBoardVersion(`
`  int cardNumber);`
	`int QuantisGetBoardVersion(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`int quantisCount();`	`int QuantisCount(` `  QuantisDeviceType deviceType);`
`int quantisDriverVersion();`	`float QuantisGetDriverVersion(` `  QuantisDeviceType deviceType);`

`int quantisGetModules(`
`  int cardNumber);`
	`int QuantisGetModulesMask(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`char* quantisGetSerialNumber(`
`  int cardNumber);`
	`char* QuantisGetSerialNumber(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`int quantisLibVersion();`	`float QuantisGetLibVersion();`

`int quantisModuleDataRate(`
`  int cardNumber);`
	`int QuantisGetModulesDataRate(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`int quantisModulesDisable(`
`  int cardNumber,`
`  int moduleMask);`
	`int QuantisModulesDisable(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber,`
	`  int modulesMask);`

`int quantisModulesEnable(`
`  int cardNumber,`
`  int moduleMask);`
	`int QuantisModulesEnable(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber,`
	`  int modulesMask);`

`int quantisModulesPower(`
`  int cardNumber);`
	`int QuantisGetModulesPower(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`int quantisModulesReset(`
`  int cardNumber,`
`  int moduleMask);`
	`int QuantisModulesReset(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber,`
	`  int modulesMask);`

`int quantisModulesStatus(`
`  int cardNumber);`
	`int QuantisGetModulesStatus(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber);`

`int quantisRead(`
`  int cardNumber,`
`  void* buffer,`
`  unsigned int size);`
	`int QuantisRead(`
	`  QuantisDeviceType deviceType,`
	`  unsigned int deviceNumber,`
	`  void* buffer,`

API 1.x functions	API 2.0 functions
	size_t size);

Table C.1. API 1.x and 2.0 functions equivalences.

C.1. Compatibility Wrapper

IDQ provides a compatibility wrapper that allows you to use the old API with the new library. This is meant to facilitate the migration of your application to the new API.

Important

It is highly recommended to update your application to the new API as soon as possible.

To use the compatibility wrapper, define `QUANTIS_DEVICE_TYPE` and then include `Quantis-Compat.h` instead of `quantis.h` and recompile your application:

```
/*
 * Define Quantis type:
 *   - set QUANTIS_DEVICE_TYPE to 1 for Quantis PCI/PCIe
 *   - set QUANTIS_DEVICE_TYPE to 2 for Quantis USB
 */
#define QUANTIS_DEVICE_TYPE 1

/* Includes compatibility wrapper */
#include "Quantis-Compat.h"
```

Note

On Microsoft Windows systems, you can try to rename `QuantisPci-Compat.dll` to `Quantis.dll` or `QuantisUsb-Compat.dll` to `Quantis-Usb.dll` and replace your old library with the renamed one. This way you normally do not need to recompile your application.

Appendix D. Notes

D.1. Images

Some images used in this manual and in the Quantis software are from VistaICO.com.

Bibliography

Websites

[USB] *Official USB website*. http://www.usb.org/ .