Cross-Platform Custom HID Demo

Contents

QT

Qt is a cross-platform framework developed by Nokia that includes an API and IDE for developing an application that can be used on all operating systems (http://qt.nokia.com/). It is distributed under three licensing options: Lesser General Public License (LGPL) v. 2.1, General Public License (GPL) v. 3.0, and Nokia's Commercial license. For more information, visit http://qt.nokia.com/products/licensing. This demo application was created using Qt under the LGPL license. It is highly recommended to thoroughly go over the licensing options, because code developed with the LGPL license cannot be added, modified, or used with the commercial license. So, if an application is developed under the commercial license, this code can only be used as a guideline, and cannot be used directly in the application. This demo was created with QT 4.7.0 (32 bit) and QT Creator 2.0.1, and was only tested for Windows 7 Professional (64 bit and 32 bit), Mac OS X 10.5.8, and Ubuntu 10.04.

HIDAPI Library

The HIDAPI library is a cross-platform library developed by Signal11 that allows for communication with HID-class USB devices (http://www.signal11.us/oss/hidapi/). The doxygen html documentation for the library and the instructions on downloading a version of hidapi can be found on the HIDAPI webpage. There are two options, either use git to download latest trunk revision clone from the GitHub website, or download the zip files directly from the website GitHub website. However, the zip files are not updated as often. There are three source files, one for each OS, and are in the corresponding subdirectory according to which OS it is used for, Linux (hid-libusb.c or hid.c), Mac OS X (hid.c), or Windows (hid.cpp). There is also a common header file that is in the parent directory. These four files are used in this demo application to create the library.

HIDAPI is available under three license agreements, the GNU Public License, v. 3.0, a BSD-style license, or the original HIDAPI license. These three licenses are documented on the hidapi webpage and its GITHub website. This demo uses the original HIDAPI license. For commercial use, the BSD-style or original license is most applicable.

Application Overview

This demo application uses two Qt projects that both need to be compiled separately for each OS. The HIDAPI C++ statically linked library project is the project that creates the HIDAPI static library file ('.a' file). It conditionally compiles the appropriate source file according to what OS is being used. This can be seen in the HIDAPI.pro file:

```
macx:  SOURCES += mac/hid.c
unix: !macx:  SOURCES += linux/hid-libusb.c
win32: SOURCES += windows/hid.cpp
```

The macx, win32, and unix keywords are used for conditionally adding the appropriate source file. The source files and header file were downloaded from the Signal11 GitHub website using git.

After compiling the HIDAPI project, the libHIDAPI.a file is created in the subdirectory according to the OS on which it was built. This Qt static library file is used by the HID_PnP_Demo Project. The HID_PnP_Demo project is a GUI Application project with two C++ classes called demoapp and hid_pnp. The demoapp class controls the application's GUI, and the hid_pnp class controls the hid communication and uses the previously created HIDAPI static library file. The HID_PnP_Demo.pro file shows how to add a library:

```
macx: LIBS += -L./HIDAPI/mac -lHIDAPI
win32: LIBS += -L./HIDAPI/windows -lHIDAPI
unix: !macx: LIBS += -L./HIDAPI/linux -lHIDAPI

macx: LIBS += -framework CoreFoundation -framework IOkit
win32: LIBS += -lSetupAPI
unix: !macx: LIBS += -lusb-1.0
```

The LIBS keyword followed by –L defines what directory to look in, and the –l keyword gives the name of the library (remove the lib prefix from the library filename). Because the HIDAPI project output the library files in different subdirectories for each OS, the correct library has to be added between the three. Also, since the library created uses OS frameworks, these are also required in the project. For Mac, the IOkit is required in addition to the CoreFoundation, for windows, the SetupAPI is required, and for linux, LibUSB v. 1.0 is required. After compilation, the output files again are placed in subdirectories corresponding to the correct OS.

Qt has a special feature added to C++ applications called signals and slots. These signals and slots allow for smoother communication between classes. Signals are connected to slots, so that when a signal is called from one class, the slot from another class immediately receives that signal. There are two signal-to-slot connections made in this project. The first is used to call the PollUSB function in the hid_api class to poll USB communication either at 15ms intervals if the device is attached, or at 250ms intervals if it is not, by using a QTimer object to send a signal at the end of the set interval. The second is used to update the GUI after the USB communication has been polled. This uses a signal from the PollUSB function in the hid_pnp class to call the UpdateGUI function in the demoapp class. Because of the signals and slots feature, the communication between the two classes allows for the entire application to be done on a single thread. This eliminates any potential race conditions that can occur when using multiple threads.

Graphical User Interface Overview

Qt comes with a built in GUI designer. For QT GUI Application Projects, a form is created when a new project is created. Similar to other GUI designers, buttons, labels, progress bars, etc. can be added to the form. This demo application is similar to the other custom HID examples available in the Microchip Application Library, where the application is plug-and-play, meaning it will automatically detect when a Microchip custom HID device is attached or detached. When detached, the labels, progress bar, and button all become disabled. When attached, they are enabled, they are updated every 15ms by the device's potentiometer reading and pushbutton status, and when the GUI button is pressed, the LEDs on the device will toggle.

Common Pitfalls

There are many pitfalls that can arise when developing an application, especially one that is supposed to run on multiple architectures. Below are a few that might commonly occur during development, specifically with a Linux machine.

Getting LibUSB v. 1.0 installed on Linux

LibUSB is an open source library created for use on Linux and other Unix-based OS's to simplify communications to USB devices. There are two APIs that are widely used, v. 0.1 and v. 1.0. Version 1.0 is used in HIDAPI and is the more stable and recommended version. To download and install this library from the Ubuntu repositories (if it isn't already), the following command can be typed in the terminal:

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

If this fails, then you probably will have to build from the source which is available on SourceForge.

Setting permissions to the USB Device on Linux

For non-root users in Linux, a USB device by default has read-only access. This prevents any packets sent from the application intended for the device to pass. To override the permission rules, a text file must be added in the /etc/udev/rules.d directory with a name with format similar to the following:

```
##-arbitraryName.rules
```

The number prefix gives the priority to the rules inside of the text file, the lower the number, the higher the priority. Since the default udev rules in /lib/udev/rules.d/50-udev-default.rules have a priority of 50, any number below 50 will work.

To write in the /etc/udev/rules.d directory, root access is needed, so use the sudo command. After creating the rules file with a text editor, the following information needs to be added to it:

```
#Add some comments for what this device is
```

```
SUBSYSTEM==”usb”, ENV{DEVTYPE}==”usb_device”, MODE=”0664”, GROUP=”Name of the
group that contains your user name”, SYSFS{idVendor}==”Your Vendor ID”,
SYSFS{idProduct}==”Your Product ID”
```

And close and save the file. Restart your system so that these changes will take effect.