Some Thoughts
Celebrating the spark when two rocks collide

Arduino Leonardo Remote Control (Multimedia Keys) Support for Arduino 1.0.5

Stefan / September 13, 2013

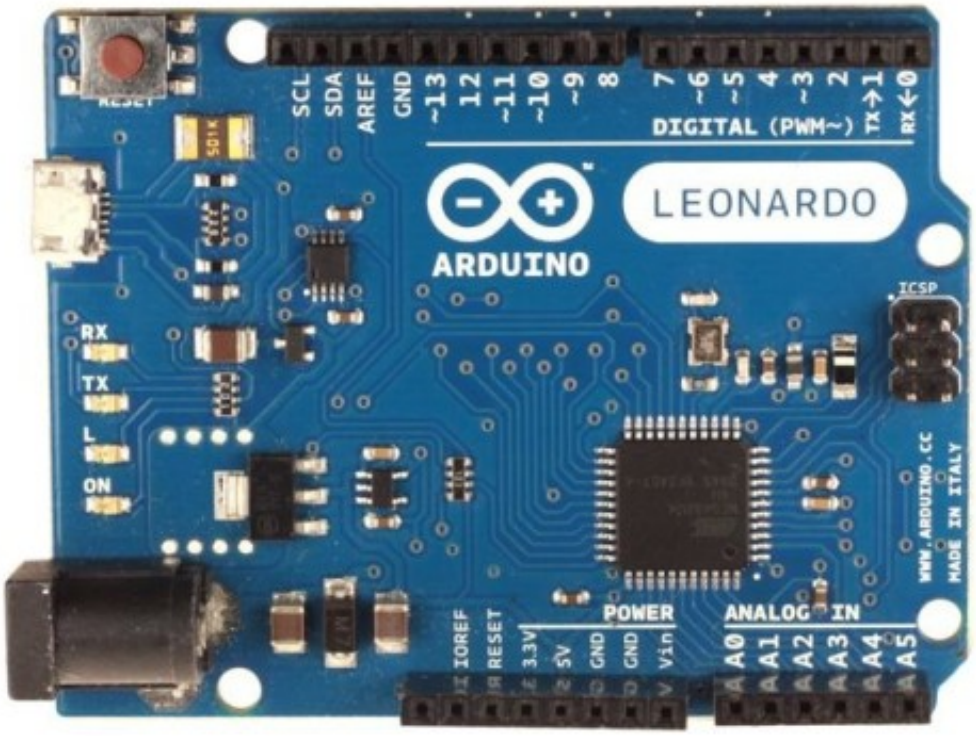
One of the perks of the **Arduino Leonardo** platform compared to the more traditional **Arduino Uno** is the ability to appear as a **USB Human Interface Device** to a host computer. Instead of communicating over serial, the Leonardo can appear as any of a variety of Human Interface Devices.

The Arduino software (as of 1.0.5) includes support for Keyboard and Mouse profiles, but in this post, I'll go over how to add **Remote Control** and **Multimedia Keys** support to your **Arduino Leonardo, Micro** or **ATMega32u4**-based Arduino. This guide also reportedly works with the **Arduino Due**. If you find another platform that the guide works with, feel free to let me know and I'll update this section.

In short, this will enable your Arduino sketches to use the **Volume Up, Volume Down, Mute, Play, Pause, Stop, Next Track, Previous Track, Rewind** and **Fast Forward** hotkeys (and more!).

This includes support for **Windows, Mac, Linux**, and even **Android with a USB-OTG cable**.

Source files for everything on this page are located in their own GitHub repository.



Locating the Arduino Source

Windows

The default location for the source files on a Windows machine running a 64-bit OS is located at:

C:/Program Files (x86)/Arduino/hardware/arduino/cores/arduino/

Mac OS X

From **Finder**, navigate to your **Applications**

Right click on **Arduino.app** and select **Show Package Contents**

Inside the package, you'll find the source files in
/Resources/Java/hardware/arduino/cores/arduino/

Back up the original files

Make sure you take a backup of these two files before continuing:

USBAPI.h
HID.cpp

Optional – Grab the source from GitHub

You can grab the updated source files from this GitHub repository, or modify your own in the following sections.

Modifications to USBAPI.h

We want to add the headers for a new **Remote** class which can be called by Arduino sketches, and the methods that will send individual commands to the host computer.

After the headers for the Keyboard and Mouse classes are defined, add the following definition for **Remote.** (Line 137)

Here, each command is encoded as the integer value of a single bit, that when set will send a matching command to the host machine.

```
//================================================================================
//================================================================================
//      Remote

#define REMOTE_CLEAR 0
#define VOLUME_UP 1
#define VOLUME_DOWN 2
#define VOLUME_MUTE 4
#define REMOTE_PLAY 8
#define REMOTE_PAUSE 16
#define REMOTE_STOP 32
#define REMOTE_NEXT 64
#define REMOTE_PREVIOUS 128
#define REMOTE_FAST_FORWARD 256
#define REMOTE_REWIND 512

class Remote_
{
private:
public:
        Remote_(void);
        void begin(void);
        void end(void);

        // Volume
        void increase(void);
        void decrease(void);
        void mute(void);

        // Playback
        void play(void);
```

```
        void pause(void);
        void stop(void);

        // Track Controls
        void next(void);
        void previous(void);
        void forward(void);
        void rewind(void);

        // Send an empty report to prevent repeated actions
        void clear(void);
};
extern Remote_ Remote;
```

Save the file, and you're finished with the headers.

Adding Class to HID.cpp

You'll want to declare a new singleton for the **Remote** class, located at the top of the source file.

```
//      Singletons for mouse and keyboard (and remote)

Mouse_ Mouse;
Keyboard_ Keyboard;
Remote_ Remote;
```

Skipping ahead to the very end of the file, we need to provide implementation details for the Remote object after the Mouse and Keyboard are defined. This can be added before the two trailing **#endif** statements in the file:

```
//================================================================================
//================================================================================
//      Remote

Remote_::Remote_(void)
{
}

void Remote_::begin(void)
{
}

void Remote_::end(void)
{
}

void Remote_::increase(void)
{
        u8 m[2];
        m[0] = VOLUME_UP;
        m[1] = 0;
```

```
        HID_SendReport(4,m,2);
}

void Remote_::decrease(void)
{
        u8 m[2];
        m[0] = VOLUME_DOWN;
        m[1] = 0;
        HID_SendReport(4,m,2);
}

void Remote_::mute(void)
{
        u8 m[2];
        m[0] = VOLUME_MUTE;
        m[1] = 0;
        HID_SendReport(4,m,2);
}

void Remote_::play(void)
{
        u8 m[2];
        m[0] = REMOTE_PLAY;
        m[1] = 0;
        HID_SendReport(4,m,2);
}

void Remote_::pause(void)
{
        u8 m[2];
        m[0] = REMOTE_PAUSE;
        m[1] = 0;
        HID_SendReport(4,m,2);
}

void Remote_::stop(void)
{
        u8 m[2];
        m[0] = REMOTE_STOP;
        m[1] = 0;
        HID_SendReport(4,m,2);
}

void Remote_::next(void)
{
        u8 m[2];
        m[0] = REMOTE_NEXT;
        m[1] = 0;
        HID_SendReport(4,m,2);
}

void Remote_::previous(void)
{
        u8 m[2];
        m[0] = REMOTE_PREVIOUS;
        m[1] = 0;
        HID_SendReport(4,m,2);
```

```
}

void Remote_::forward(void)
{
        u8 m[2];
        m[0] = 0;
        m[1] = REMOTE_FAST_FORWARD >> 8;
        HID_SendReport(4,m,2);
}

void Remote_::rewind(void)
{
        u8 m[2];
        m[0] = 0;
        m[1] = REMOTE_REWIND >> 8;
        HID_SendReport(4,m,2);
}

void Remote_::clear(void)
{
        u8 m[2];
        m[0] = 0;
        m[1] = 0;
        HID_SendReport(4,m,2);
}
```

We will add one more change to the code, adding a **USB HID Descriptor** for the **Consumer Control** Page, which hosts the typical **Remote Control** and **Multimedia Keys** available to modern operating systems.

From the Wikipedia Entry on USB Human Interface Devices:

> “Devices define their data packets and then present a “HID descriptor” to the host. The HID descriptor is a hard coded array of bytes that describe the device’s data packets. This includes: how many packets the device supports, the size of the packets, and the purpose of each byte and bit in the packet. For example, a keyboard with a calculator program button can tell the host that the button’s pressed/released state is stored as the 2nd bit in the 6th byte in data packet number 4.

The USB Descriptor will be added after this section in the Arduino source:

```
#if RAWHID_ENABLED
        //      RAW HID
    ...
#endif
```

Note that in the following snippet, each of the “Usage” lines corresponds to a bit of data defined in the **USBAPI.h** file. The final block leaves six bits remaining for passing your own

additional functionality.

```
//------------------------------------------------------------------------

    /* Cross-platform support for controls found on IR Remotes */

    0x05, 0x0c,                 //   Usage Page (Consumer Devices)
    0x09, 0x01,                 //   Usage (Consumer Control)
    0xa1, 0x01,                 //   Collection (Application)
    0x85, 0x04,                 //   REPORT_ID (4)
    0x15, 0x00,                 //   Logical Minimum (0)
    0x25, 0x01,                 //   Logical Maximum (1)
    0x09, 0xe9,                 //   Usage (Volume Up)
    0x09, 0xea,                 //   Usage (Volume Down)
    0x75, 0x01,                 //   Report Size (1)
    0x95, 0x02,                 //   Report Count (2)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xe2,                 //   Usage (Mute)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xb0,                 //   Usage (Play)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xb1,                 //   Usage (Pause)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xb7,                 //   Usage (Stop)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xb5,                 //   Usage (Next)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xb6,                 //   Usage (Previous)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)

    0x09, 0xb3,                 //   Usage (Fast Forward)
    0x95, 0x01,                 //   Report Count (1)
    0x81, 0x06,                 //   Input (Data, Variable, Relative)
```

```
0x09, 0xb4,                    //   Usage (Rewind)
0x95, 0x01,                    //   Report Count (1)
0x81, 0x06,                    //   Input (Data, Variable, Relative)

0x95, 0x06,                    //   Report Count (6) Number of bits rem
0x81, 0x07,                    //   Input (Constant, Variable, Relative
0xc0                           //   End Collection
```

That's everything you need to do with the Arduino source. Save your work and try compiling a sketch with the profile for the 'Arduino Leonardo' board. If any problems persist, you can always compare with the source on GitHub.

Sample Sketch for Arduino Multimedia Keys

Uploading this to an Arduino should alternately mute and un-mute the system volume every five seconds. Calling Remote.clear() after each action is recommended.

```
/*
   Sample sketch for Multimedia Keys
   Alternately mute and un-mute the system volume every five seconds.

   http://stefanjones.ca/blog/arduino-leonardo-remote-multimedia-keys/
*/
void setup() {
}

void loop() {
  delay(5000);

  // Prevent duplicate activation
  Remote.mute();
  Remote.clear();
}
```

Advanced Usage

You can add additional features from the Consumer Page by referring to Page 75 of the ~~USB HID Usage Tables~~ and adding them to the descriptor:

```
0x09, 0xbc,                    //   Usage (Repeat, with usage code 'BC'
0x95, 0x01,                    //   Report Count (1)
0x81, 0x06,                    //   Input (Data, Variable, Relative)
```

```
      0x95, 0x05,                    //   'Blank' Report Count decreased fron
```

USBAPI.h

```
#define REMOTE_REPEAT 1024
...
void repeat(void);
```

HID.cpp

```
void Remote_::repeat(void)
{
        u8 m[2];
        m[0] = 0;
        m[1] = REMOTE_REPEAT >> 8;
        HID_SendReport(4,m,2);
}
```

In a follow-up post, I'll be providing a method I found useful for using an IR Sensor with the Leonardo, and a version of the AdaFruit IR-Commander and Raw-IR-Decoder sketches with reduced memory requirements.

September 13, 2013 in Programming. Tags: arduino, leonardo

Configuring Side Buttons on Logitech Mice under Ubuntu Linux →

Leave a Message

Thanks for making it this far. If you'd like to drop me a comment or suggestion about this entry, feel free to submit one here:
These comments will not be published.

Name *

Email *

Comment

Leave Comment

Search …

About

A dropbox of ideas and writing from Stefan Jones, posting on matters related to technology, programming, and various miscellany.

Recent Posts

Launching a Random Steam Game

Installing the RadeonSI Driver and Steam on Ubuntu 14.04

Fixing the Endless Spinning Circle of Dots During Boot on Windows 8 (Mac Version, BootCamp)

Configuring Side Buttons on Logitech Mice under Ubuntu Linux

Arduino Leonardo Remote Control (Multimedia Keys) Support for Arduino 1.0.5

Categories

Linux

Mac

Programming

Uncategorized

Windows

Contact

stefan@stefanjones.ca