USB/Bluetooth Media Controller
Final Report

40056761
SET09118
Edinburgh Napier University

22-04-2016

Abstract

This document is the final report and a summary to the ”USB/Bluetooth Media Controller” project started in February [Appendix F] [Appendix G]. The goal of the project was to research, develop and evaluate a microcontroller based system and mobile application. The purpose of this report is to catalogue the projects various stages, suggest future work and assert the overall success of the assignment.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Background and Rationale

The goal for this project was to produce a system that would allow users to interact with their PC media in a more intuitive manner. From tape cassettes to compact discs and finally to digital MP3 files, the way in which media is stored and experienced has changed dramatically over the last twenty years but the way that users interact with their media has remained stagnant.

While playing or pausing media, changing track or altering volume are typically not difficult tasks for most users, however it can often get complicated when the media is not in focus or if other applications are in use. Many companies have attempted to solve this issue by incorporating media control keys into keyboards. However these buttons are often in hard to reach locations, or they only work with specific applications.

For the reasons mentioned above, it is clear that an independent Media Controller, that supports a multitude of media applications would be a popular device for many users that utilise a PC as the main source of their media. One of the key deliverables of this project is the aforementioned Media Controller, but the other is a companion mobile application that will allow users to access media functions while away from the PC or controller.

1.2 Aims and Deliverables

As stated previously, the two major deliverables discussed in this report are a microcontroller based Media Controller and an Android mobile application to control said device. The aim of both deliverables is to simplify the way in which a user interacts with media. This was a major consideration for design and implementation of the systems.

Media Controller The Media Controller is a device that can be connected to a PC via USB, and provide users with a simple, physical interface with which to interact with their media.

In terms of functionality, the controller will provide users with the ability to play or pause their music, skip to the next or previous track and raise or lower the volume of the music. These functions are accessible through a set of push buttons, providing tactile feedback to the user.

Basing the system around an Arduino microcontroller [1] allows for simple expansion of the project. For instance, additional buttons, switches or potentiometers can be easily added to the system and allow a user to customise the controller and augment its functions.

Mobile Application The Media Controller was designed to extrapolate any media functions into the simplest interface possible and the Android application attempts to do the same.

The benefit of a mobile application is that it allows a user to interact with their media, even if they are not at their desk.

The goal of the application's appearance was to keep it as minimal as possible. The reason for this was to ensure the application's design was appealing to the user and to keep it as seamlessly simple as possible to use.

In terms of functionality, the Android application will offer the user the same PC media functions as the physical controller, but via a remote, digital interface.

Limitations In an ideal world, a Media Controller would be able interact with any device and any media application, allowing for complete control without the need for installation and configuration. Unfortunately, it would be impossible to ensure total functionality on every combination of operating systems, hardware and media application, so limitations were imposed to keep the implementation reasonable. MoSCoW method was utilized to determine what was within scope of the project.

MoSCoW
Must:

- Media Controller functionality on a PC running Windows operating system
- Media key support for the Google Play Music application

Should:

- Not require proprietary code running on PC to execute commands

Could:

- No configuration required / plug-n-play support
- Support for additional host devices
- Support for additional media applications

Won't:

- IOS or Windows Phone support

2 Project Management

To be sure that the project reached fruition, project management techniques were utilized. The previously mentioned MoSCoW method was used to determine the features to be included in the project and their priority in terms of the finished product. To keep track of the projected time needed to implement each feature a Gantt chart (Figure 1) was used.

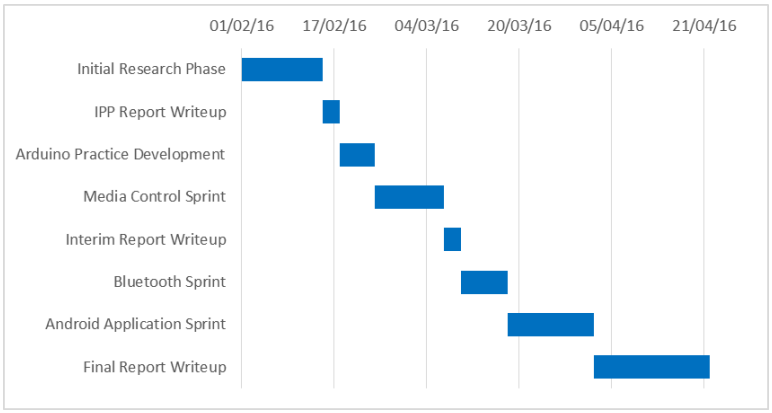


Figure 1: A Gantt chart of the project time line.

By utilizing development sprints it was possible to implement the systems and document the process with very little overlap. The sprint cycles also had the added benefit of forcing speedy development, preventing feature creep and over engineering.

The version control software git was heavily utilized during the development process. Git allowed new and potentially volatile features to be tested, without any risk to the code base's integrity. Git also allows developers to view any previous changes they have made to their code, an indispensable feature when attempting to identify bugs or dysfunctional features.

3 Design

3.1 Media Controller Design

While it was a simple task to specify what functionality the Media Controller would have, it proved much harder to determine the way in which it would be delivered. A circuit digram (Figure 2) was

produced to plan the layout of the system's physical components, after which research of potential software implementation begun.

Key press scripts Preliminary research indicated that a common method of allowing a microcontroller to execute commands on a PC was through "man-in-the-middle" scripts, such AutoHotkey [2] or AutoIt [3]. These programs would be run on a computer, intercept signals from a connected microcontroller and execute the requested command as a simulated keystroke. While key scripts would be usable, they did come with major flaws. Firstly, key scripts would only work with media applications that already support keyboard short-cuts and secondly, the MoSCoW method produced for this project (Section 1.2, Page 4) stated that propriety programs should be avoided if possible. When it was determined that Key press scripts would only offer a partial solution to the project, further research was taken to find a superior method.

Human Interface Devices Peripherals such as USB keyboards, game pads and joysticks are all classified as USB Human Interface Devices or HID's [4] [5]. When a HID is plugged into a PC, it transmits a list of hexadecimal values called a usage table to the host PC. Each number in the usage table corresponds to a specific command or function for the host device to carry out. The commands that can be included in the table consist of everything from printing characters (keyboard) to moving a mouse cursor (track ball). In addition, a number of media controls are available through the use of an HID usage table.

Microcontroller With the discovery of usage tables in mind, a suitable microcontroller had to be found to act as an HID. Research of various Arduino product specifications [6] [7] [8] and data sheets [Appendix E] indicated that the Arduino Leonardo would be the most suited platform to prototype the system on. The reason for this was that unlike the Uno, the Leonardo can interact with a PC as an HID, this is because the Leonardo only has a single processor to both run Sketches and deal with USB communications.

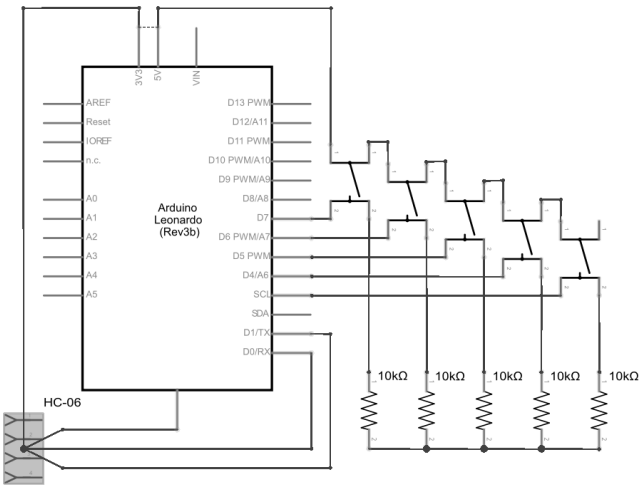


Figure 2: A circuit diagram of the proposed Media Controller.

3.2 Mobile Application Design

The premise for the mobile application was simple - to allow a user to access the functions of the Media Controller remotely, and thus the overall aesthetic of the application was designed to be simplistic as well. The most challenging part of the application's design stage was determining how the application would communicate with the microcontroller.

Remote Serial Communication Preliminary research indicated that two major technologies existed to communicate serial signals over-the-air, namely Bluetooth and Wi-Fi. A simplified breakdown of each standard is presented in Table 1.

Table 1: A comparison of two wireless communication technologies

Technology	Range	Frequency	Data Rate	Power Use	Cost
Bluetooth 2.0	∼100m	2.4 GHz	1-3 Mbit/s	Medium	Low
Wi-Fi 802.11g	∼100m	2.4 GHz	54 Mbit/s	High	Medium

As one can see from the table, both technologies offer similar benefits and either one could have been used in the final system. In the end it was decided that Bluetooth would be the most suitable medium, due to its lower cost and previous personal experience using this technology. To finalise this decision, a HC-06 Bluetooth module [9] was purchased to allow the Media Controller to transmit a Bluetooth signal.

Bluetooth Communication While Bluetooth is a relatively simple technology in terms of use, it still must be correctly configured to ensure expected functionality. An activity diagram of the Bluetooth connection process [Appendix D] was created to ensure that the application's communication procedure was well documented and implemented correctly.

User Interface As previously stated, the goal of the application's interface was to keep it as simple as possible. Wire frames were produced to prototype potential layouts. In the end, a simple panel of vertical buttons was chosen for its ease of use and minimalistic style.

4 Implementation

4.1 Media Controller Implementation

With the knowledge that HID usage commands would be the most feasible way to complete the project, the creation of a bespoke media usage table commenced. However, to complete the implementation further, examples of usage tables were required.

By examining Arduino's default mouse and keyboard HID library's [10] and utilizing information from the USB Implementers Forum [11], it became clear how to implement a usage table as an Arduino library. With the usage table implemented, it still had to be filled with the relevant hexadecimal codes.

Usage Codes A three year-old personal blog post that had previously been dismissed for being too outdated was able to provided a sample Usage Table with a host of different media commands [12]. The post in question had been analysed as potential solution to the project during the initial research phase. However, due to the age of the post and the fact the code was designed to interact with an IDE fourteen iterations out of date (at time of writing), it was deemed unusable. Even a brief conversation on the official Arduino IRC channel with a lead developer determined that the libraries needed to get the blog code running again were no longer available in the Arduino IDE.

While the majority of the hex codes presented in the blog's table were usable, a few had depreciated and no longer worked. The USB Implementers Forum once again provided the answer in the form of a PDF with all the current in use hex codes [13].

The usage codes used in the finished device are visible in Table 2, an example of how they appear in the source code is seen in [Appendix B].

Push Buttons Once the media library was implemented they had to be made accessible to the user. This was done through the use of an open source Arduino button library called Bounce2 [14]. The reason for using a external library was to ensure that any electrical or interference would not cause the microcontroller to register a single button-press as multiple, calling a media function several times when a user only wanted to use it once.

Once the buttons had been made foolproof, it was a simple matter of having each button call a command from the media library, see Listing 1, and the physical system implementation was complete.

Table 2: A usage table of the HID commands utilized by the Media Controller

Usage ID	Usage Name	Usage Type
0xCD	Play/Pause	OSC
0xB0	Play	OOC
0xB1	Pause	OOC
0xB3	Fast Forward	OOC
0xB4	Rewind	OOC
0xB5	Scan Next Track	OSC
0xB6	Scan Previous Track	OSC
0xB7	Stop	OSC
0xE2	Mute	OOC
0xE9	Volume Increment	RTC
0xEA	Volume Decrement	RTC

```
68  if (debounce[1].fell() == HIGH) {
69    play();
70  }
```

Listing 1: An example of how the media-controler executes a media command.

4.2 Mobile Application Implementation

The goal of the applications design was to be as simple as possible. For this reason, implementation of the user interface took very little time. The majority of the effort was spent on configuring the Bluetooth communication, ensuring the application still functioned in the event of an unexpected disruption to the Media Controller connection.

User Interface As can be seen in Figure 3, when the application is not connected to the Media Controller, the option available to the user is the connection button. This is done to prevent a user from attempting to use a function when the Media Controller is inaccessible and causing confusion.

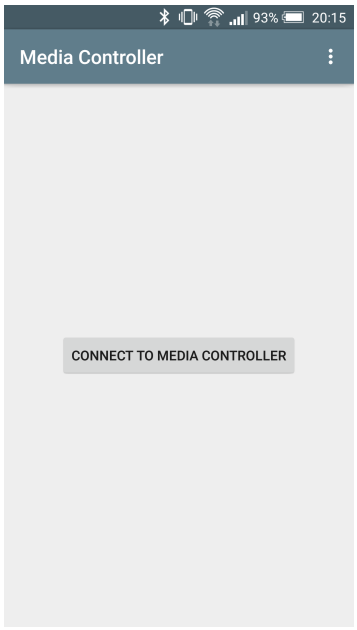


Figure 3: The mobile application interface before a connection has been established.

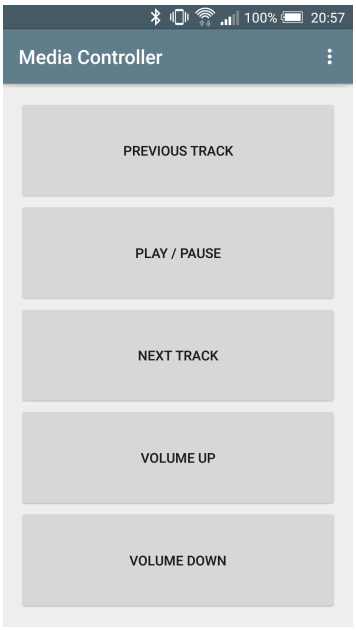


Figure 4: The mobile application interface after connecting to the Media Controller.

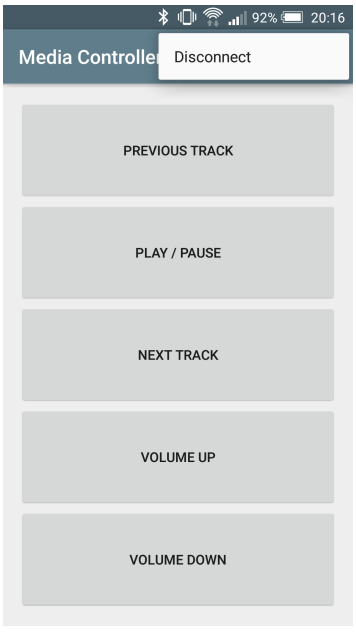


Figure 5: The disconnect button, accessible from the pop-up menu.

Once the user is connected to the physical system, they will be able to access several buttons - each corresponding to a button on the physical Media Controller (Figure 4). A user will also be able to disconnect from the bonded device (Figure 5), returning them to the starting stage of the application.

Bluetooth As previously mentioned, the majority of the applications development time was spent on implementing the Bluetooth connection process. Sample code from Google [15] was used to produce a basic Bluetooth connection class, with further error checking and handling added as the application was tested. The implemented Bluetooth tests can be seen in the included activity diagram [Appendix D].

Signal Transmission Having the application communicate a requested media function to the micro-controller was a simple affair. When a button on the applications user interface was click, a unique character was transmitted over the Bluetooth's output stream. Examples of the code used can be seen in Listings 2 and 3

```
31        //set up function buttons
32        final Button button0 = (Button) findViewById(R.id.main_button_0);
33        assert button0 != null;
34        button0.setOnClickListener(new View.OnClickListener() {
35            public void onClick(View v) {
36                mBluetoothHandler.writeValue("p");
37            }
38        });
```

Listing 2: A example of the Android's application button code.

```
191    //sends a value to connect device
192    public void writeValue(String val) {
193        try {
194            mOutput.write(val.getBytes());
195        } catch (IOException | NullPointerException e) {
196            e.printStackTrace();
197        }
198    }
```

Listing 3: The method called when transmitting a charcter to the Media Controller.

Signal Reception On the device side of the spectrum, the code to receive a signal from the application was also simple to implement. The Media Controller is constantly listening on its serial input stream and upon reading a character it calls the relevant function. It should be noted that pressing one of the physical buttons on the controller has a higher precedence than the application and will be executed before the application's requested command. The code required for this functionality is seen in Listing 4.

```
86  //check Bluetooth
87  while (mySerial.available()) {
88    //read char
89    myChar = mySerial.read();
90    Serial.print(myChar);
91
92    //execute command
93    switch (myChar) {
94      case't':
95        play();
96        break;
97      case'p':
98        prev();
99        break;
100     case'n':
101       next();
102       break;
103     case'u':
104       up();
105       break;
106     case'd':
107       down();
108       break;
109     default:
110       break;
111   }
```

Listing 4: The code utilized by the microcontroller to receive input from the Application.

5 Testing

Testing was of high importance in this project. Due to the nature of the two systems communicating between each other, it had to be verified that the correct data was being transferred and any communication disconnections did not cause any unexpected results.

To ensure all possible tests were conducted a test case table (Table 3) was constructed with the expected outcomes for each event. This allowed the systems to be systematic checked for any and allow potential bugs.

Table 3: Test cases used to validate the systems

Case	Expected Outcome	Outcome
No Bluetooth adapter	Tell user application cannot operate	TRUE
Bluetooth not enabled	Ask user to enable Bluetooth	TRUE
Device not powered	Tell user could not connect	TRUE
Connected to device	Inform user of connection	TRUE
Disconnect from device	Inform user of disconnection	TRUE
Function button pressed	Host device calls function	TRUE

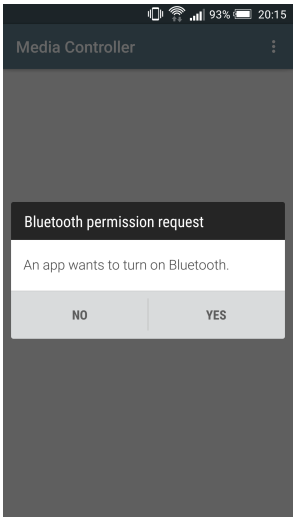


Figure 6: The message displayed to users when Bluetooth is not enabled.

During the testing stage of the project the Media Controller was tried on various host devices to check if the HID functionality was windows specific. It was found that the Media Controller was at least partially operational on the following devices:

- Mac OS laptop
- Debian Linux desktop
- Android tablet
- Samsung TV

Even though the MosCoW method only specified that the system should function on a Windows operating system, the device functions on a multitude of devices, an unforeseen feature of utilizing HID usage codes.

6 Results

The specification for the project presented in this report was to construct a microcontroller based system and companion application, with some form of communication between each of the systems. The Media Controller and remote controller application meet the requirements outlined in the specification, as well as making use of technologies and methodologies not requested in the specification.

6.1 Achievements

The main achievements of this project are as follows;

- Significantly expanded knowledge of microcontroller development.
- A greater understanding of USB Human Interface Devices and their operations.
- A fully functional C++ media command usage table.
- Further experience of Developing on the Android platform
- Expanded knowledge of wireless serial communication technologies and practices.

The open specification of the project allowed for investigation and implementation of specific areas of interest.

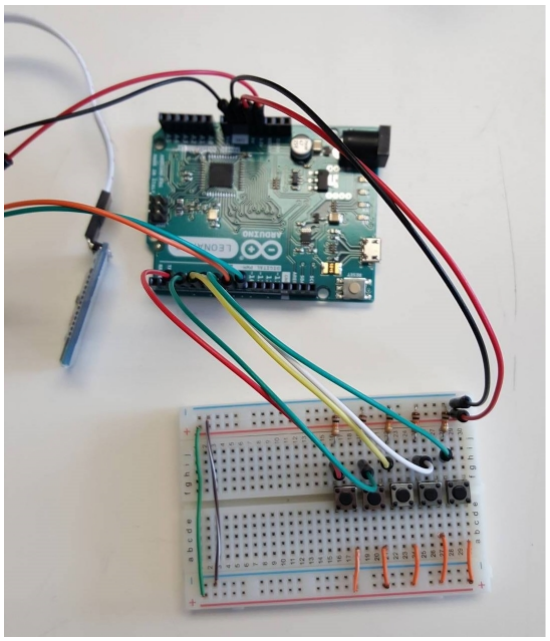


Figure 7: The final Media Controlle - microcontroller, Bluetooth module and push-buttons.

6.2 Recommendations

One of the key areas of consideration when undertaking a project similar to this is the limitations and specifications of the hardware intended to be used. Almost all microcontrollers available for purchase will be able to interface with a PC via serial communication. However only a handful of the systems can function as a USB Human Interface Device.

Another consideration for similar systems is the method of remote communication between the Media Controller and the user. For the purposes of this project Bluetooth provided an adequate service for linking the application to the device. It may be worth considering technologies like Wi-Fi or ZigBee to allow the system to be accessed by multiple user at once or over a wireless local area network.

6.3 Future Work

Though the project is complete and fulfilled all the aims and goals highlighted in the specification, there is still potential for additional functionality to be added to the deliverables.

One potential idea would be to incorporate different operation modes into the controller, allowing a user to change the functionality of the systems buttons, depending on which task they were working on. Examples include a mode for text processing where the buttons would provide short cuts for functions such as cut, copy and paste, or a mode for photo editing software where the buttons will correspond to an editing tool.

There is also potential to add additional functionality to the mobile application. One thought was to include a PC security feature where the host PC would automatically lock if the physical system lost connection with the mobile device, i.e. if a user walked away from their computer with their phone in their pocket. This would allow a user simply walk away from their PC, without the need to lock their PC or worry about others accessing it while they are away from their desk.

7 Conclusion

The aim of this project was to produce a Media Controller and companion application and investigate communication between both systems.

Agile development methods like MoSCoW and Gantt charts were utilized to ensure the project stayed on scheduled and time wasted developing non-crucial features of the systems.

Even though it was never intended during the project's inception, a great deal of effort went into the research of how USB devices work and how they are developed. While the use of HID technology was not the only way to complete this project, it appeared to be the most fit for purpose.

Overall, the project can be considered a success - all of the original goals that were set out were achieved. Though the development of the system was challenging at times, this kept the project engaging and entertaining.

8 Evaluation of Achievement

Having never worked with microcontrollers previously, the open ended specification of this project was extremely daunting. This is partially why I opted to construct a Media Controller. I felt that building a device that I could use everyday would motivate me to research the best methods for implementation.

While the specification for this project focused mainly communication methods, I got highly engrossed in learning about USB devices, their configuration and specifications. I found researching HIDs especially interesting, as they are devices that I use every day and yet had very little knowledge of how they operated.

One aspect of the project I found challenging was the lack of examples available to me. Many people have built Media Controllers but either their code was too outdated to learn from or they had gone about it in a completely different manner. While this did slow down the development process, it did make it all the more satisfying when the system was finally complete.

If I was redoing this project with the knowledge I have now, I would have started the development of the mobile application much earlier into the projects life cycle. While the application produced is entirely functional, I felt that could be improved greatly with just a little more time for polish.

I thoroughly enjoyed this project, I was given the freedom to research and develop a system that I found interesting, while learning about physical computing and communication technologies.

References

[1] “Arduino - home,” https://www.arduino.cc/, (Accessed on 04/16/2016).

[2] “Autohotkey,” https://autohotkey.com/, (Accessed on 04/16/2016).

[3] “Autoit - autoit,” https://www.autoitscript.com/site/autoit/, (Accessed on 04/16/2016).

[4] “Device class definition for human interface devices (hid),” http://www.usb.org/developers/hidpage/HID1_11.pdf, (Accessed on 04/12/2016).

[5] “Silicon labs - human interface device tutorial,” https://www.silabs.com/Support%20Documents/TechnicalDocs/AN249.pdf, (Accessed on 04/16/2016).

[6] “Arduino - arduinoboarduno,” https://www.arduino.cc/en/Main/ArduinoBoardUno, (Accessed on 04/10/2016).

[7] “Arduino - arduinoboardleonardo,” https://www.arduino.cc/en/Main/ArduinoBoardLeonardo, (Accessed on 04/10/2016).

[8] “Arduino - arduinoleonardomicro,” https://www.arduino.cc/en/Guide/ArduinoLeonardoMicro#toc5, (Accessed on 04/17/2016).

[9] “Hc-06 datasheet 201104201104 revised.doc,” https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf, (Accessed on 04/16/2016).

[10] “arduino/arduino: open-source electronics prototyping platform,” https://github.com/arduino/arduino, (Accessed on 04/10/2016).

[11] “Usb.org - hid tools,” http://www.usb.org/developers/hidpage/, (Accessed on 04/10/2016).

[12] “Arduino leonardo remote control and multimedia keys support in 1.0.5 — some thoughts,” http://stefanjones.ca/blog/arduino-leonardo-remote-multimedia-keys/, (Accessed on 04/10/2016).

[13] “Usb - hid usage tables,” http://www.usb.org/developers/hidpage/Hut1_12v2.pdf, (Accessed on 04/10/2016).

[14] “thomasfredericks/bounce2: Debouncing library for arduino or wiring,” https://github.com/thomasfredericks/Bounce2, (Accessed on 04/10/2016).

[15] “Bluetooth — android developers,” http://developer.android.com/guide/topics/connectivity/bluetooth.html, (Accessed on 04/20/2016).

Appendices

A Arduino Media Controller Source Code

```
#include <Bounce2.h>
#include <Media.h>
#include <SoftwareSerial.h>

SoftwareSerial mySerial(8, 9); // RX, TX

char myChar;

//declare array of buttons
const int button[] = {3, 4, 5, 6, 7};
//declare array of debouncers
Bounce debounce[sizeof(button)];
//declare interval for debouncers
const int del = 5;
const int volumeDelay = 50;

void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);

  //define buttons/bouncers attach a bouncer to each button
  for (int i = 0; i < sizeof(button); i++) {
    pinMode(button[i], INPUT);
    debounce[i] = Bounce();
    debounce[i].attach(button[i]);
    debounce[i].interval(del);
  }
}

void checkDebounce() {
  //update each debouncer i.e. check for presses
  for (int i = 0; i < sizeof(button); i ++) {
    debounce[i].update();
  }
}

void play() {
  Media.play();
  Media.clear();
}
void prev() {
  Media.previous();
  Media.clear();
}
void next() {
  Media.next();
  Media.clear();

}
void up() {
  Media.increase();
  Media.clear();
}
void down() {
  Media.decrease();
  Media.clear();
}

void loop() {

  //check all buttons
  checkDebounce();

  if (debounce[0].fell() == HIGH) {
    prev();
  }

  if (debounce[1].fell() == HIGH) {
    play();
  }

  if (debounce[2].fell() == HIGH) {
    next();
  }

```

```
76   if (debounce[3].read() == HIGH) {
77     up();
78     delay(volumeDelay);
79   }
80
81   if (debounce[4].read() == HIGH) {
82     down();
83     delay(volumeDelay);
84   }
85
86   //check Bluetooth
87   while (mySerial.available()) {
88     //read char
89     myChar = mySerial.read();
90     Serial.print(myChar);
91
92     //execute command
93     switch (myChar) {
94       case't':
95         play();
96         break;
97       case'p':
98         prev();
99         break;
100      case'n':
101        next();
102        break;
103      case'u':
104        up();
105        break;
106      case'd':
107        down();
108        break;
109      default:
110        break;
111    }
112    //reset char
113    myChar = ' ';
114  }
115
116 }
```

mediaController_debounce.ino

B Arduino Media Key Library Source Code

```
#ifndef MEDIA_h
#define MEDIA_h

#include "HID.h"

#if !defined(_USING_HID)

#warning "Using legacy HID core (non pluggable)"

#else
//================================================================================
//================================================================================
//  Media

#define MEDIA_CLEAR 0
#define VOLUME_UP 1
#define VOLUME_DOWN 2
#define VOLUME_MUTE 4
#define MEDIA_PLAY 8
#define MEDIA_PAUSE 16
#define MEDIA_STOP 32
#define MEDIA_NEXT 64
#define MEDIA_PREVIOUS 128
#define MEDIA_FAST_FORWARD 256
#define MEDIA_REWIND 512

class Media_
{
private:
public:
    Media_(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 Media_ Media;

#endif
#endif
```

Media.h

```
#include "Media.h"

#if defined(_USING_HID)

static const uint8_t _hidReportDescriptor[] PROGMEM = {
    0x09, 0x01,     //Usage (Consumer Control)
    0x05, 0x0c,     //Usage Page (Consumer Devices)
    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, 0xcd,     //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 remaining in byte
    0x81, 0x07,     //Input (Constant, Variable, Relative)
    0xc0            //End Collection
};

Media_::Media_(void)
{
    static HIDSubDescriptor node(_hidReportDescriptor, sizeof(_hidReportDescriptor));
    HID().AppendDescriptor(&node);
}

void Media_::begin(void)
{
}

void Media_::end(void)
{
}

void Media_::increase(void)
{
    u8 m[2];
    m[0] = VOLUME_UP;
    m[1] = 0;
    HID().SendReport(4,m,2);
}

void Media_::decrease(void)
{
    u8 m[2];
    m[0] = VOLUME_DOWN;
    m[1] = 0;
    HID().SendReport(4,m,2);
}
```

```
84
85 void Media_::mute(void)
86 {
87     u8 m[2];
88     m[0] = VOLUME_MUTE;
89     m[1] = 0;
90     HID().SendReport(4,m,2);
91 }
92
93 void Media_::play(void)
94 {
95     u8 m[2];
96     m[0] = MEDIA_PLAY;
97     m[1] = 0;
98     HID().SendReport(4,m,2);
99 }
100
101 void Media_::pause(void)
102 {
103     u8 m[2];
104     m[0] = MEDIA_PAUSE;
105     m[1] = 0;
106     HID().SendReport(4,m,2);
107 }
108
109 void Media_::stop(void)
110 {
111     u8 m[2];
112     m[0] = MEDIA_STOP;
113     m[1] = 0;
114     HID().SendReport(4,m,2);
115 }
116
117 void Media_::next(void)
118 {
119     u8 m[2];
120     m[0] = MEDIA_NEXT;
121     m[1] = 0;
122     HID().SendReport(4,m,2);
123 }
124
125 void Media_::previous(void)
126 {
127     u8 m[2];
128     m[0] = MEDIA_PREVIOUS;
129     m[1] = 0;
130     HID().SendReport(4,m,2);
131 }
132
133 void Media_::forward(void)
134 {
135     u8 m[2];
136     m[0] = 0;
137     m[1] = MEDIA_FAST_FORWARD >> 8;
138     HID().SendReport(4,m,2);
139 }
140
141 void Media_::rewind(void)
142 {
143     u8 m[2];
144     m[0] = 0;
145     m[1] = MEDIA_REWIND >> 8;
146     HID().SendReport(4,m,2);
147 }
148
149 void Media_::clear(void)
150 {
151     u8 m[2];
152     m[0] = 0;
153     m[1] = 0;
154     HID().SendReport(4,m,2);
155 }
156
157 Media_ Media;
158
159 #endif
```

Media.cpp

C Android Application Source code

```
1  package uk.co.sam.mediacontroller;
2
3  import android.content.Intent;
4  import android.os.Bundle;
5  import android.os.Handler;
6  import android.support.v7.app.AppCompatActivity;
7  import android.support.v7.widget.Toolbar;
8  import android.view.MotionEvent;
9  import android.view.View;
10 import android.view.Menu;
11 import android.view.MenuItem;
12 import android.view.ViewGroup;
13 import android.widget.Button;
14 import android.widget.RelativeLayout;
15
16 import java.util.ArrayList;
17
18 public class MainActivity extends AppCompatActivity {
19
20     private static ArrayList<Button> mFunctionButtons;
21     private static ArrayList<Button> mOtherButtons;
22     private BluetoothHandler mBluetoothHandler;
23
24     @Override
25     protected void onCreate(Bundle savedInstanceState) {
26         super.onCreate(savedInstanceState);
27         setContentView(R.layout.activity_main);
28         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
29         setSupportActionBar(toolbar);
30
31         //set up function buttons
32         final Button button0 = (Button) findViewById(R.id.main_button_0);
33         assert button0 != null;
34         button0.setOnClickListener(new View.OnClickListener() {
35             public void onClick(View v) {
36                 mBluetoothHandler.writeValue("p");
37             }
38         });
39         final Button button1 = (Button) findViewById(R.id.main_button_1);
40         assert button1 != null;
41         button1.setOnClickListener(new View.OnClickListener() {
42             public void onClick(View v) {
43                 mBluetoothHandler.writeValue("t");
44             }
45         });
46         final Button button2 = (Button) findViewById(R.id.main_button_2);
47         assert button2 != null;
48         button2.setOnClickListener(new View.OnClickListener() {
49             public void onClick(View v) {
50                 mBluetoothHandler.writeValue("n");
51             }
52         });
53         final Button button3 = (Button) findViewById(R.id.main_button_3);
54         assert button3 != null;
55         button3.setOnTouchListener(new View.OnTouchListener() {
56             private Handler mHandler;
57             @Override public boolean onTouch(View v, MotionEvent event) {
58                 switch(event.getAction()) {
59                     case MotionEvent.ACTION_DOWN:
60                         if (mHandler != null) return true;
61                         mHandler = new Handler();
62                         mHandler.postDelayed(mAction, 500);
63                         break;
64                     case MotionEvent.ACTION_UP:
65                         if (mHandler == null) return true;
66                         mHandler.removeCallbacks(mAction);
67                         mHandler = null;
68                         break;
69                 }
70                 return false;
71             }
72             Runnable mAction = new Runnable() {
73                 @Override public void run() {
74                     mBluetoothHandler.writeValue("u");
75                     mHandler.postDelayed(this, 50);
76                 }
77             };
78         });
79         final Button button4 = (Button) findViewById(R.id.main_button_4);
80         assert button4 != null;
```

```
81          button4.setOnTouchListener(new View.OnTouchListener() {
82              private Handler mHandler;
83              @Override public boolean onTouch(View v, MotionEvent event) {
84                  switch(event.getAction()) {
85                      case MotionEvent.ACTION_DOWN:
86                          if (mHandler != null) return true;
87                          mHandler = new Handler();
88                          mHandler.postDelayed(mAction, 500);
89                          break;
90                      case MotionEvent.ACTION_UP:
91                          if (mHandler == null) return true;
92                          mHandler.removeCallbacks(mAction);
93                          mHandler = null;
94                          break;
95                  }
96                  return false;
97              }
98              Runnable mAction = new Runnable() {
99                  @Override public void run() {
100                     mBluetoothHandler.writeValue("d");
101                     mHandler.postDelayed(this, 50);
102                 }
103             };
104         });
105
106         //put all buttons in an array for easy access
107         mFunctionButtons = new ArrayList<>();
108         mOtherButtons = new ArrayList<>();
109         ViewGroup layout = (ViewGroup) findViewById(R.id.main_button_layout);
110         assert layout != null;
111         int childCount = layout.getChildCount();
112         for (int i = 0; i < childCount; i++) {
113             View child = layout.getChildAt(i);
114             if (child instanceof Button) {
115                 mFunctionButtons.add((Button) child);
116             }
117         }
118
119         RelativeLayout mainLayout = (RelativeLayout) findViewById(R.id.main_layout);
120
121         //device connection buttons
122         final Button buttonConnect = new Button(this);
123         buttonConnect.setText(R.string.main_activity_connect_button);
124         assert mainLayout != null;
125         mainLayout.addView(buttonConnect);
126         RelativeLayout.LayoutParams layoutParams =
127                 (RelativeLayout.LayoutParams) buttonConnect.getLayoutParams();
128         layoutParams.addRule(RelativeLayout.CENTER_IN_PARENT, RelativeLayout.TRUE);
129         buttonConnect.setLayoutParams(layoutParams);
130         buttonConnect.setOnClickListener(new View.OnClickListener() {
131             public void onClick(View v) {
132                 if (mBluetoothHandler.isEnabled()) {
133                     mBluetoothHandler.findDevice();
134                 } else {
135                     mBluetoothHandler.enableBluetooth();
136                 }
137             }
138         });
139
140         mOtherButtons.add(buttonConnect);
141         hideButtons();
142
143         mBluetoothHandler = new BluetoothHandler(this, toolbar);
144     }
145
146     //method to hide all buttons
147     public static void hideButtons() {
148         for (Button button : mFunctionButtons) {
149             button.setVisibility(View.GONE);
150         }
151         for (Button button : mOtherButtons) {
152             button.setVisibility(View.VISIBLE);
153         }
154
155     }
156
157     //method to show all butoons
158     public static void showButtons() {
159         for (Button button : mFunctionButtons) {
160             button.setVisibility(View.VISIBLE);
161         }
162         for (Button button : mOtherButtons) {
163             button.setVisibility(View.GONE);
```

```
164         }
165     }
166
167     @Override
168     public boolean onCreateOptionsMenu(Menu menu) {
169         // Inflate the menu; this adds items to the action bar if it is present.
170         getMenuInflater().inflate(R.menu.menu_main, menu);
171         return true;
172     }
173
174     //bluetooth activation result
175     @Override
176     protected void onActivityResult(int requestCode, int resultCode, Intent data) {
177         if (requestCode == 1) {
178             mBluetoothHandler.onActivityResult(requestCode, resultCode);
179         }
180         super.onActivityResult(requestCode, resultCode, data);
181     }
182
183     //menu handler
184     @Override
185     public boolean onOptionsItemSelected(MenuItem item) {
186         int id = item.getItemId();
187         switch (id) {
188             case R.id.action_disconnect:
189                     mBluetoothHandler.disconnectDevice();
190         }
191         return super.onOptionsItemSelected(item);
192     }
193
194     //pause handler
195     @Override
196     public void onPause() {
197         super.onPause();  // Always call the superclass method first
198         mBluetoothHandler.disconnectDevice();
199     }
200
201     //resume handler
202     @Override
203     public void onResume() {
204         super.onResume();  // Always call the superclass method first
205         hideButtons();
206     }
207
208 }
```

MainActivity.java

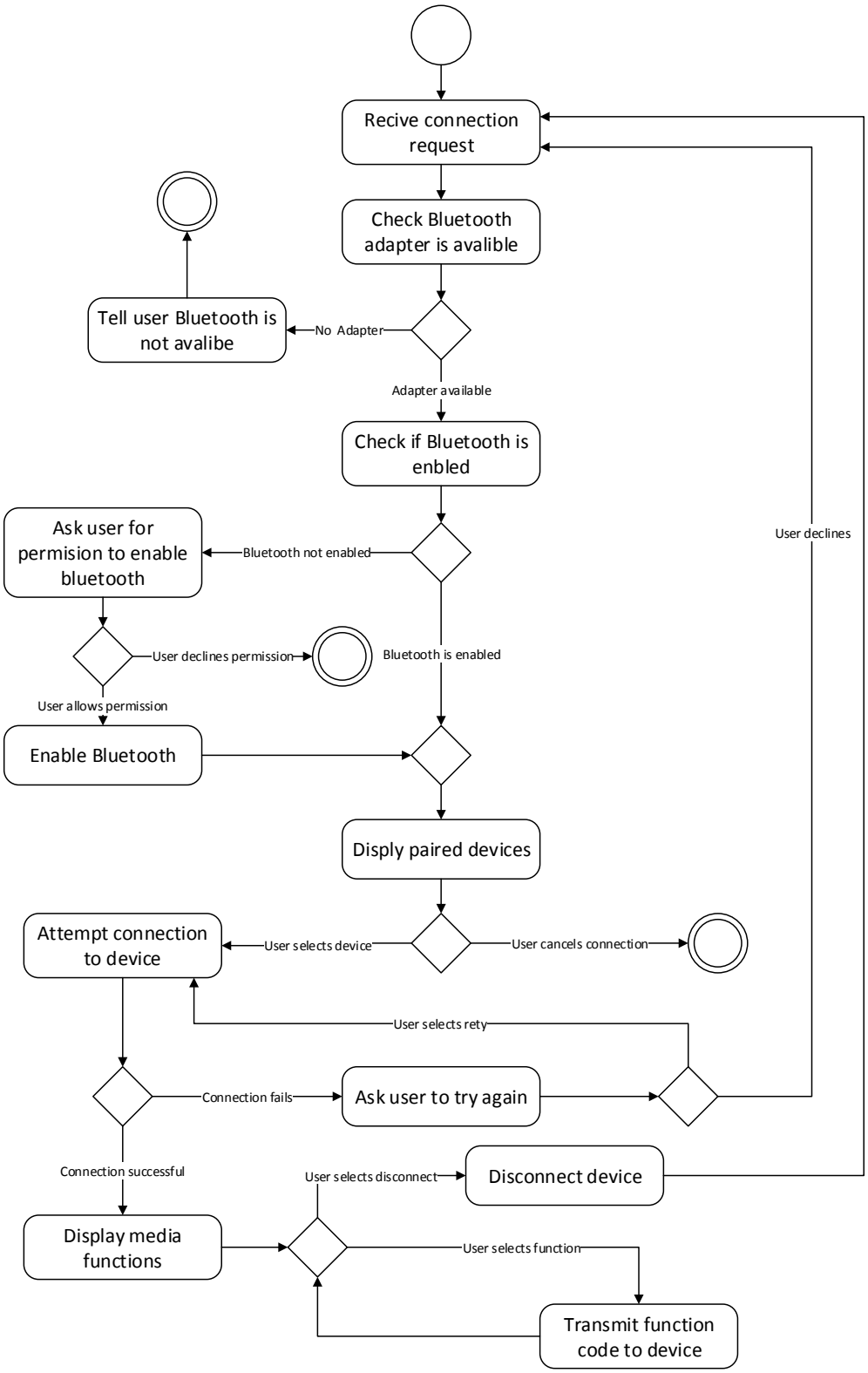
```java
package uk.co.sam.mediacontroller;

import android.app.Activity;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.DialogInterface;
import android.content.Intent;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AlertDialog;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

import java.io.IOException;
import java.io.OutputStream;
import java.util.Set;
import java.util.UUID;

public class BluetoothHandler {

    private static final UUID MY_UUID = UUID
            .fromString("00001101-0000-1000-8000-00805f9b34fb");
    private static final int REQUEST_ENABLE_BT = 1;
    private static final String DEVICE = "MEDIA_CONTROLLER";
    private BluetoothAdapter mBluetoothAdapter;
    private View mView;
    private Activity mActivity;
    private BluetoothDevice mDevice;
    private BluetoothSocket mSocket;
    private OutputStream mOutput;
    private ProgressDialog progress;


    //class constructor
    public BluetoothHandler(Activity activity, View view) {
        //get activity resources
        this.mView = view;
        this.mActivity = activity;
        //check mobile device has a Bluetooth adapter
        mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (mBluetoothAdapter == null) {
            //tell user no adapter is available
            Snackbar.make(this.mView, "Bluetooth unavailable on this device.", Snackbar.LENGTH_LONG).↩
     show();
        }
    }

    //turns on Bluetooth
    public void enableBluetooth() {
        //check if Bluetooth is already enabled
        if (!isEnabled()) {
            //start Bluetooth
            Intent btOnIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            //inform activity that bluetooth has been started
            mActivity.startActivityForResult(btOnIntent, REQUEST_ENABLE_BT);
        } else {
            //do nothing
        }
    }

    //returns whether bluetooth is on or off
    public boolean isEnabled() {
        return mBluetoothAdapter.isEnabled();
    }

    //result method as to whether bluetooth was successfully turned on
    protected void onActivityResult(int requestCode, int resultCode) {
        if (requestCode == REQUEST_ENABLE_BT) {
            //if bluetooth could not be enabled
            if (resultCode != Activity.RESULT_OK) {
                //tell user
                Snackbar.make(mView, "Could not enable Bluetooth", Snackbar.LENGTH_LONG).show();
            } else {
                //show previously paired devices to user
//                  showPairedDevicesDialog();
                findDevice();
            }
```

```
83          }
84      }
85
86      public void findDevice() {
87          //get all paired devices
88          final Set<BluetoothDevice> pairedDevices = mBluetoothAdapter
89                  .getBondedDevices();
90          //so long as there is one paired devices
91          if (pairedDevices != null) {
92              //loop through all paired devices
93              for (BluetoothDevice device : pairedDevices) {
94                  //if current device matches micro-controller
95                  if (DEVICE.equals(device.getName())) {
96                      //set current device as main device
97                      mDevice = device;
98                  }
99              }
100             try {
101                 //connect to selected device
102                 connectDevice();
103             } catch (IOException e) {
104                 e.printStackTrace();
105             }
106         } else {
107             //do nothing
108         }
109     }
110
111     //connect to a selected device
112     void connectDevice() throws IOException {
113         //create a progress dialog
114         progress = ProgressDialog.show(mActivity, null, "Connecting to " + mDevice.getName(), true);
115         //create new thread for connection process
116         new Thread(new Runnable() {
117             @Override
118             public void run() {
119                 try {
120                     //connect to device
121                     mSocket = mDevice.createRfcommSocketToServiceRecord(MY_UUID);
122                     mSocket.connect();
123                     //open output stream
124                     mOutput = mSocket.getOutputStream();
125                     //dismiss load dialog
126                     progress.dismiss();
127                     //tell user connection was successful
128                     Snackbar.make(mView, "Connection successful", Snackbar.LENGTH_SHORT).show();
129
130                     //create thread to update UI
131                     mActivity.runOnUiThread(new Runnable() {
132                         @Override
133                         public void run() {
134                             //reveal buttons on main page
135                             MainActivity.showButtons();
136                         }
137                     });
138
139                 } catch (IOException e) {
140                     e.printStackTrace();
141                     //dismiss load bar
142                     progress.dismiss();
143                     //inform user connection failed
144                     Snackbar snack = Snackbar.make(mView, "Connection failed", Snackbar.LENGTH_LONG);
145                     //add retry button to snack bar
146                     snack.setAction("Retry", new View.OnClickListener() {
147                         @Override
148                         public void onClick(View v) {
149                             try {
150                                 //try to connect again
151                                 connectDevice();
152                             } catch (IOException e) {
153                                 e.printStackTrace();
154                             }
155                         }
156                     });
157                     //show failed snack bar
158                     snack.show();
159                 }
160             }
161             //start thread
162         }).start();
163     }
164
165     //disconnect a connected device
```

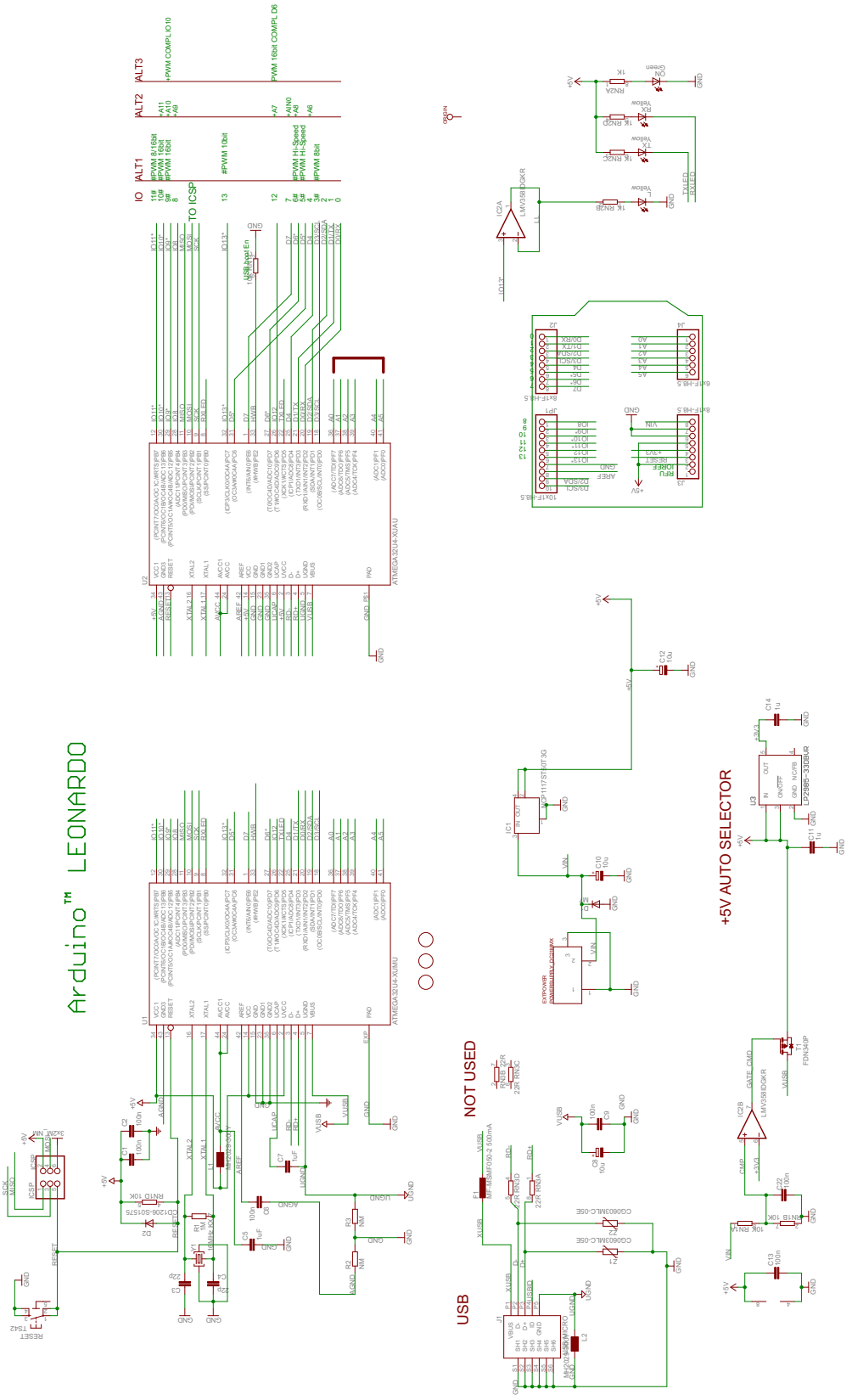
```
166     public void disconnectDevice() {
167         //check that the socket
168         if (mSocket != null) {
169             //check socket is in use
170             if (mSocket.isConnected()) {
171                 try {
172                     //close output stream
173                     mOutput.close();
174                     //close socket
175                     mSocket.close();
176                     //tell user disconnect was successful
177                     Snackbar.make(mView, "Disconnected from " + mDevice.getName(), Snackbar.↩
        LENGTH_SHORT).show();
178                     //hide buttons on UI
179                     MainActivity.hideButtons();
180 
181                 } catch (IOException e) {
182                     e.printStackTrace();
183                 }
184             } else {
185                 //tell user nothing to disconnect
186                 Snackbar.make(mView, "Not connected to a device", Snackbar.LENGTH_SHORT).show();
187             }
188         }
189     }
190 
191     //sends a value to connect device
192     public void writeValue(String val) {
193         try {
194             mOutput.write(val.getBytes());
195         } catch (IOException | NullPointerException e) {
196             e.printStackTrace();
197         }
198     }
199 
200 }
```

BluetoothHandler.java

D Bluetooth Connection Activity Diagram



E Arduino Leonardo Data Sheet



F Initial Project Proposal

USB / Bluetooth Media Controller

Initial Project Proposal

Name: Sam Dixon

Matric No: 40056761

Abstract

A Media Controller allows a user to interact with an entertainment device in a manner other than a traditional graphical interface or a keyboard and mouse. By providing physical controls a user is able to quickly alter media on a PC, without the need to change application window, or the use of a keyboard and mouse. A typical media controller allows a user to skip forward or backwards on a playlist, pause/resume media and control the volume the system. Additional features may include the ability to save and play favorited media or advance audio customization, such as bass or gain level.

In order to recreate a media controller, research in to pre-existing solitons must first take place. This will include devices such as DJ decks, USB keyboards, plug-and-play panels, etc. Once these technologies have been researched, their functionality can be reversed engineered and recreated on a smaller scale using a microcontroller. Once the physical system is fully functional a mobile application will be developed, to allow remote access to the media functions provided by the micro controller.

The goal of this project is to produce two deliverables, a microcontroller based USB media controller and a companion android application. With these systems working in tandem, the functionality of a media controller can be recreated, on a more personal, cost effective scale.

Introduction

Background

The rational for this project was simple; to provide a quick and efficient way for a user to alter media being played on a PC.

While many keyboard already offer inbuilt media keys, they can often be in hard-to-reach locations or require awkward key-combinations to use. Theses reason make a standalone media controller an attractive option to users, as the controller can be placed in an easy-to-reach location and a simple, analogue interface ensures that the correct function is always selected.

While the technologies that allow users to experience their media have grown ever more advance, the actual method of controlling said media remains relatively unchanged. The main benefits of an external controller include; it allows a user to augment their media without losing access to the keyboard and mouse, haptic feedback on button presses, simplified controls (useful for elderly/children) and access to media remotely or when the PC is locked.

Basing the system around an Arduino microcontroller allows for simple expansion of the project. For instance; a Bluetooth module will allow the media controller to be accessed remotely via a mobile application. The main benefit of this is that allows a user to control their media remotely for instance; at a party the computer can be locked to prevent undue access to the system but the host can still change the current song through the mobile application.

Aims and deliverables

The overall goal of the project is create a USB/Bluetooth media controller and mobile application.

The physical device will consist of Arduino microcontroller with an individual button for each media function, namely; previous track, next track, play/pause and mute. Volume control will also be present, either in the form of a rotary knob or two buttons; one to increase and the other decrease volume. A Bluetooth module will also be present, to allow a mobile device to connect to the system and control it remotely.

The mobile application's goal is to provide the same functionality as the hub, via a Bluetooth connection to the microprocessor. In addition to the media functions present on the hub, the application can include additional operations, such as locking the computer remotely.

In order to use the app, the user must first connect to the media control hub via Bluetooth. The user will then have access to several labelled buttons that offer the same functionality of the hub.

The goal for the appearance of the app is to keep it as simple as possible, in order to reduce the overall learning curve and keep the experience intuitive.

Design choices

Deliverables

The final deliverables of this project will be; an Arduino based media controller and Sketch, and an Android application. The controller will feature several buttons to allow physical input, and the Android application will provide the same functionality but with digital input.

Scope

In an ideal world, the media controller would work with all applications "out-of-the-box", but as there is not a unified standard for media control input, this is not possible. To compromise, the controller will be designed to work on a PC running a Windows operating system. In terms of what media

programs will be supported, Google's Play Music will be main focus, with other programs included if time allows.

On the application side of the project, only Android devices will be supported. The app is also being designed with mobile phones in mind, meaning that it will not be optimized for tablet sized screens.

Method

In order to provide the functionality required to complete this project, the micro controller needs to communicate the requested function to a media program in a manner that can be understood and carried out. The simplest way to achieve this is to have the microcontroller imitate a physical system that can already do this i.e. a USB keyboard.

Preliminary research indicates that an Arduino Leonardo board will provide the best start point for this project. The reason for the Leonardo, over the more popular Duo board is because the formers single processing capabilities, allowing for improved USB communication. The Leonardo board also typically retails at a lower price than the Duo, leading to more cost effective project.

Milestones

With this project primarily serving as learning initiative, there are serval milestones that aim to be met over the project lifecycle, namely:

- Correct wiring of a microcontroller / breadboard device.
- Using a microcontroller as a basic human interface device e.g. printing text in a document.
- Using a microcontroller to send media control signals to a PC, e.g. play/pause music.
- Develop an android application to interface with a microcontroller.

With the final goal in mind, these milestones should offer motivation and provide sufficient evidence

Resources

To complete this project, the following resources will be required:

- Android Studio (IDE) – for producing the Android application.
- Arduino Software (IDE) – for developing the Sketch for the microcontroller.
- Arduino Leonardo Starter kit – providing a microcontroller to run the system as well as the hardware required to implement all the features (breadboard, resistors, etc.).
- Arduino Bluetooth module – to allow communication between the microcontroller and a mobile device.
- Android mobile phone – for testing the system, in this case a HTC One M7 will be used for the majority of testing.
- Windows PC – for development and testing.
- Google Play Music – for testing functionality.

References

[1] Arduino.cc, "Arduino - MouseKeyboard", 2016. [Online]. Available: https://www.arduino.cc/en/Reference/MouseKeyboard. [Accessed: 18- Feb- 2016].

[2] Arduino.cc, "Arduino - ArduinoLeonardoMicro", 2016. [Online]. Available: https://www.arduino.cc/en/Guide/ArduinoLeonardoMicro. [Accessed: 18- Feb- 2016].

[3]B. Benchoff, "Turning an Arduino into a USB keyboard", *Hackaday*, 2012. [Online]. Available: http://hackaday.com/2012/06/29/turning-an-arduino-into-a-usb-keyboard/. [Accessed: 18- Feb- 2016].

[4] Instructables.com, "PC USB Media Volume Controller based on Arduino", 2016. [Online]. Available: http://www.instructables.com/id/PC-USB-Media-Volume-Controller-based-on-Arduino/. [Accessed: 18- Feb- 2016].

[5] Proto-PIC, "Proto-PIC Starter Kit for Arduino Leonardo", 2016. [Online]. Available: https://proto-pic.co.uk/proto-pic-starter-kit-for-arduino-leonardo/. [Accessed: 18- Feb- 2016].

G Interim Report

USB / Bluetooth Media Controller

Interim Report

Name: Sam Dixon

Matriculation No: 40056761

Abstract

This document is an interim report summarising the current state of the "USB / Bluetooth Media Controller". The project consists of the research, development and testing of a microcontroller based media controller and a companion mobile application. The purpose of this report is to catalogue the project's various stages of development and to describe future action.

Table of Contents

1 Introduction

1.1 Background

The rationale for this project was simple - to provide a quick and efficient way for a user to alter the way media is played on a PC.

While many keyboards already offer inbuilt media keys, they can often be in locations that are difficult to access, or they require the usage of awkward key-combinations. These reasons make a stand-alone media controller an attractive option to users, as the controller can be placed in a location that is easy to reach, and a simple analogue interface ensures that the correct function is always selected.

While the technologies that allow users to experience their media have grown ever more advanced, the actual method of controlling this media remains relatively unchanged. The main benefits of an external controller include allowing a user to augment their media without losing access to the keyboard and mouse; haptic feedback on button presses; simplified controls (something which may be particularly useful for elderly users or children); and access to media remotely, or when the PC is locked.

Basing the system around an Arduino microcontroller allows for simple expansion of the project. For instance, a Bluetooth module will allow the media controller to be accessed remotely via a mobile application. The main benefit of this is that it allows a user to control their media remotely. At a party, for instance, the host's computer could be locked to prevent undue access to the system, but the host can still change the current song through the mobile application.

1.2 Aims and Deliverables

The original goal of the project was to create a USB/Bluetooth media controller and mobile application. While this remains the ultimate goal, additional functionality in the form of Windows short cuts or hotkeys is likely to be included in the final build.

The physical device will consist of an Arduino microcontroller with an individual button for each media function. These functions will be as follows - previous track; next track; play/pause; mute. Volume control will also be present in the form of two buttons (one to increase and the other decrease volume). A Bluetooth module will also feature, to allow a mobile device to connect to the system and control it remotely.

The mobile application’s goal is to provide the same functionality as the hub, via a Bluetooth connection to the microprocessor. In addition to the media functions present on the hub, the application will include various Windows hotkeys, such as Lock PC and Log Off. At this point in time, it is currently unclear whether these functions will only be accessible via the app, or if the application will change the functionality of the physical buttons.

In order to use the app, the user must first connect to the media control hub via Bluetooth. The user will then have access to several labelled buttons that offer the same functionality of the hub.

The goal for the appearance of the app is to keep it as simple as possible, in order to reduce any potential learning curve involved in usage and keep the experience intuitive.

2 Current Status

2.1 Research and Analysis

From the preliminary research it was clear that this project was feasible, however, achieving the desired functionality without the necessary proprietary software running on the PC would make the task more complex.

By reviewing the datasheets for various microcontrollers [1] [2] and the projects others had produced with them [3], it was determined that an Arduino Leonardo had the capabilities to operate as a plug-and-play USB device. Had this not been the case, a script would have to be produced that would essentially 'listen' for serial input from the microcontroller and respond appropriately.

With the knowledge that a Leonardo could act as a suitable base for the project, further research as to how the microcontroller HID operations began. By examining the Leonardo's HID, USBAPI, Keyboard and Mouse libraries [4]a better understanding of the device's USB protocols was attained, as well as providing some base code for the bespoke Media Key library being produced.

Further understanding of how USB-HID devices operated was required to fully comprehend how the Leonardo communicates with a host PC. The official USB developers guidelines [5] provided a wealth of knowledge on the operation of USB devices, specifically HID Usage Tables [6]. In essence, a Usage Table is a list of all possible operations a USB HID device can perform. This list is sent to the PC when the device is plugged in and at any point, the device can request the host to carry out any of the functions provided.

The final step in producing a fully functional Media Key library was to acquire the HEX codes of each command that would be included in HID Usage Table. A three year-old personal blog post that had previously been dismissed for being too outdated provided a sample Usage Table for a host of different media commands [7]. The post in question had been analysed as potential solution to the project during the initial research phase. However, due to the age of the post and the fact the code was designed to interact with an IDE fourteen iterations out of date (as of time of writing), it was deemed unusable. Even a brief conversation on the official Arduino IRC with a lead developer determined that the libraries needed were no longer present in the IDE.

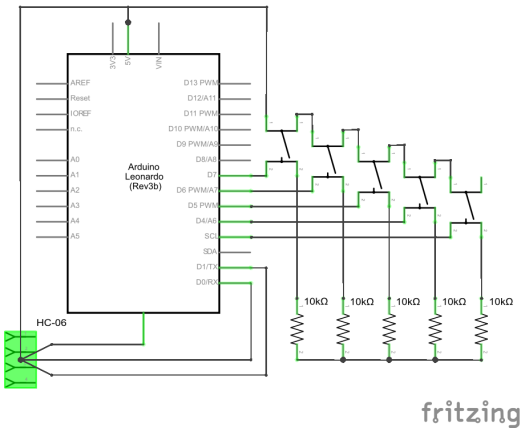


Figure 1- A circuit diagram of the complete system

After producing a working Media Library it was discovered that some of the HEX codes were outdated and were no longer supported by most media systems. This was a simple issue to solve, using the USB HID manual from earlier [6] allowed for quick look-up of the most update HEX codes and how they would be interpreted by the PC.

2.1 Practical and Experimental Work

With the second phase of research complete, the production of custom Media Key library was under taken and completed within the span of working week. This allowed for plenty of time to test, debug and fine tune the library against a number of media systems, including; Google Play Music, Windows Media Player, VLC, Winamp and Spotify.

With the successful completion of the required library, the first major Arduino Sprint has come to a close. As it stands at the time of writing, the system comprises of five push switches, each of which calls a different media control (namely previous, play, next, volume up and volume down). To ensure accurate readings from the switches, an external library was utilized. The Bounce2 library offers accurate edge detection and ensures that a function will only be called once per button press. While it could be said that utilizing an external library reduces the overall complexity of the system, the fact remains that main focus of the project is the media controls it offers. If time allows a bespoke debounce library may be produced but as it stands currently, the Bounce2 library provides the exact functionality required and allowed for more efficient testing.

2.3 Project Management

The previous report highlighted a list of milestones to be completed over the course of the project. The updated list can be seen below:

- ✓ Learn to correctly wire a microcontroller / breadboard device.
- ✓ Using a microcontroller as a basic human interface device. E.g. printing text in a document.
- ✓ Using a microcontroller to send media control signals to a PC. E.g. play/pause music.
- Add basic Bluetooth functionality.
- Develop an Android application to interface with a microcontroller.

A more detailed Gantt chart has been produced, to more display the time frame of the project accurately. The completed tasks are depicted in blue, while those yet to be started are in orange.

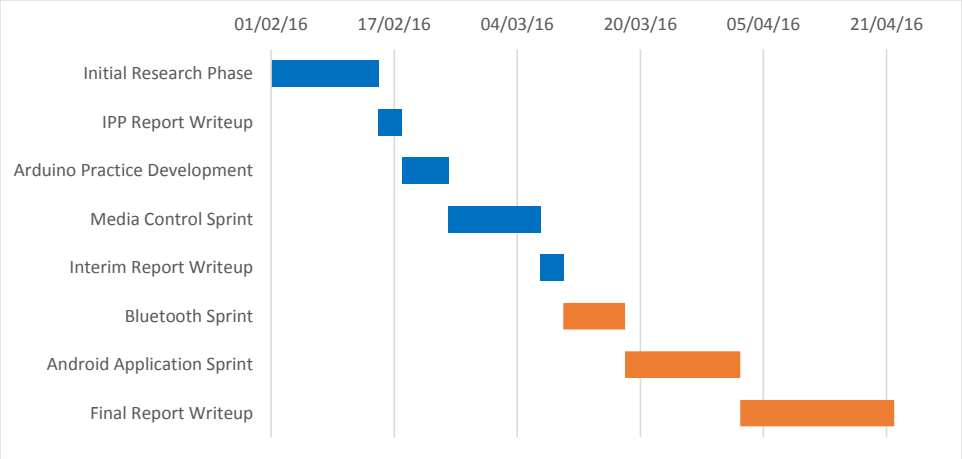


Figure 2 - A Gantt chart indicating the project timeline

As can be seen, the project is well underway and is on time to be complete for the deadline.

3 Future Programme of Work

3.1 Future Sprints

As one can see from the project timeline, two further sprints remain before completion of the project, namely the Bluetooth and Android Application sprints.

The goal of the next development sprint is to set up and configure basic Bluetooth communication with the Media controller. At the time of writing, the sprint has been delayed until the arrival of the

required hardware, namely a HC-06 Bluetooth Module. Once the appropriate resources are present, the scheduled sprint should be relatively short and the project will move into its final phase.

Once the Bluetooth module has arrived and been configured, the development of the companion Android application can begin. Mobile development can often be a time consuming process. Fortunately, the application to accompany the media control is of a minimalist design, in the hopes of keeping the operation and development as simple as possible. As seen in the timeline, the application sprint has been scheduled to be the longest. This is to ensure that application can be toughly tested, resulting in an enjoyable user experience.

3.2 Final Report

Once all the design, implementation and testing of the system is complete, all that remains of the project is the final report. At this time, the final report consists of rough notes and diagrams and the final section headers are yet to be finalized. While the report itself may currently be in a state of disarray, the notes being taken at stage of the project are thorough and should allow for a concise and balanced summary of the project.

4 Issues and Concerns

4.1 Hardware Concerns

As it stands, the only issue in terms of hardware is the lack of it. Once the required HC-06 module arrives, it should be a simple task to get it configured, thanks to prior experience with this technology.

4.2 Software Concerns

As previously stated, application development can often be a time consuming affair. The time scale for the project has left ample time to produce an application for the device. However, if more time is required to complete the final report, the overall quality of the application may suffer.

5 Conclusion

5.1 Summary

The project is well under way and is on track to be completed to a high standard for the deadline. The only constraint that currently exists is the time required for the Bluetooth module to arrive. Once Bluetooth implementation is complete, the only concern will be the time required to complete the Android application. While a very basic application can be created in a short amount of time, the usability and features may suffer if any major road-blocks are met during the sprint.

Compared to the time and effort required for the research and development of the project, the final report and presentation should prove to be a relatively simple affair. The hard work put in at the beginning of the project should allow for fast and efficient production of the final documentation.

6 References

6.1 References List

[1] “Arduino - ArduinoBoardLenoardo,” Arduino, [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardLeonardo. [Accessed 8 March 20016].

[2] “Arduino - ArduinoBoardUno,” Arduino, [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno. [Accessed 8 March 2016].

[3] M. Heironimus, “Add USB Game Controller to Arduino Leonardo/Micro,” instructables, [Online]. Available: http://www.instructables.com/id/Add-USB-Game-Controller-to-Arduino-LeonardoMicro/. [Accessed 8 March 2016].

[4] “arduino/Arduino: open-source electronics prototyping platform,” Arduino, [Online]. Available: https://github.com/arduino/arduino. [Accessed 8 March 2016].

[5] “USB.org - HID Tools,” USB Implementers Forum, [Online]. Available: http://www.usb.org/developers/hidpage/. [Accessed 8 March 2016].

[6] “Universal Serial Bus (USB) HID Usage Tables,” 21 October 2004. [Online]. Available: http://www.usb.org/developers/hidpage/Hut1_12v2.pdf. [Accessed 8 March 2016].

[7] S. Jones, “Arduino Leonardo Remote Control and Multimedia Keys Support in 1.0.5 | Some Thoughts,” [Online]. Available: http://stefanjones.ca/blog/arduino-leonardo-remote-multimedia-keys/. [Accessed 8 March 2016].