Senior Project Report: MIDIFlapper, a Leap Motion MIDI Controller

Mark Henry

Dr. Franz J. Kurfess, Advisor

California Polytechnic State University, San Luis Obispo

Software Engineering Department

College of Engineering

March 2014

Abstract

An application called MIDIFlapper was developed that translates data from the Leap Motion, an NUI device, into MIDI data suitable for use by a contemporary digital audio workstation. This allows electronic musicians to use the Leap Motion for musical creation and live performance.

Contents

1 Introduction and Background

1.1 About MIDIFlapper

MIDIFlapper is an application for live musical performance using the Leap Motion device. It is a virtual MIDI controller that emits CC messages in response to the user's interaction with the Leap Motion. The mappings between the user's actions and Flapper's resulting messages are user-editable.

The project goal is to create an app that is as useful as a hardware MIDI controller within a typical creative workflow or performance setup, while providing a direct, intuitive interface that can't be imitated with traditional hardware controllers (not to mention greatly surpassing them in "cool factor").

1.2 The Leap Motion

The Leap Motion controller is a hardware sensor by San Francisco-based startup Leap Motion Inc. It is a small USB device that is designed to sense the position of the user's hands and fingers roughly within a hemisphere of three feet.

The Leap's API delivers Frame objects to subscribed applications whenever objects enter its sensing range. These Frames contain data about the positions and velocities of fingers, hands and tools, as well as derived data such as descriptions of their relative positions, gesture recognition, and more. See Figure 1 to get a sense of what data is represented.

1.3 MIDI, CC Messages and DAWs

MIDI, Musical Instrument Digital Interface, is a standard for the interactions between instruments, keyboards, controllers and other musical hardware. The most relevant part of the standard *vis-à-vis* this project is the protocol for control change—or CC—events, which are the messages that knobs and sliders on hardware MIDI controllers generate.

CC messages are an ordered triple of three numbers: channel, control number, and a value between 0 and 127. The channel and control number form an "address," an ordered pair uniquely identifying which control was changed; the 0-127 value indicates the knob's new position.

The digital audio workstation, or DAW, houses synthesizers and audio effects and manages their interactions. Synthesizers generate audio in response to MIDI data from the host DAW. In the typical use case, this data originates from within the DAW, but the DAW is capable of accepting data from external sources and routing it to destinations of the user's choice. This is the use case we are concerned with, where CC events from MIDIFlapper are passed to synths and effects within the DAW. See Figure 2 for an at-a-glance view of the data flow.

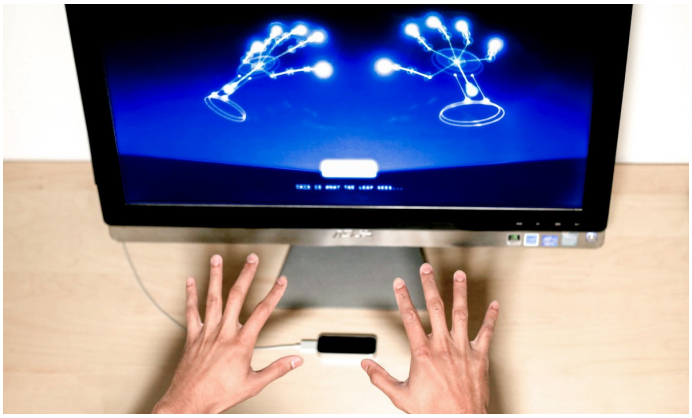

Figure 1: The Leap Motion in use, with onscreen graphic representation of data generated

2 Related Work

2.1 Previous Work

From April to June 2011, I worked with a team in CSC-486, Human-Computer Interaction, to create an application called Kinectamidia which was very similar to MIDIFlapper. Kinectamidia was also a virtual MIDI controller. It used the Microsoft Kinect instead of the Leap Motion (and consequently was written in C#), but was in principle identical.

I was not satisfied with Kinectamidia. For one thing, it had a limited feature set and didn't offer enough customization to the user to be useful as a utility in a musician's creative process. But the major, insurmountable problem was that the hardware was flawed. There is simply too much latency in the Microsoft Kinect—on the order of 100 ms [2]. For an application that demands 50 milliseconds of latency at an absolute maximum (and even less than that if the user wants to perform complex rhythms) it was completely unacceptable.

I was very excited to hear Leap Motion promise exceedingly slim latency times in its promo materials, and was very impressed when I received my dev kit. The Leap's response time is 50ms at its worst, and decreases from there depending on the Leap's settings and the capabilities of the user's video and USB hardware. 30ms is typical. [3]

2.2 Similar Work

At the time of the conception of this project, many applications for body-based musical performance existed. The pioneers for such applications included Chris Vik, Jonathan Hammond and the V Motion Project, who used the Microsoft Kinect and other NUI interfaces to create audiovisual displays that are

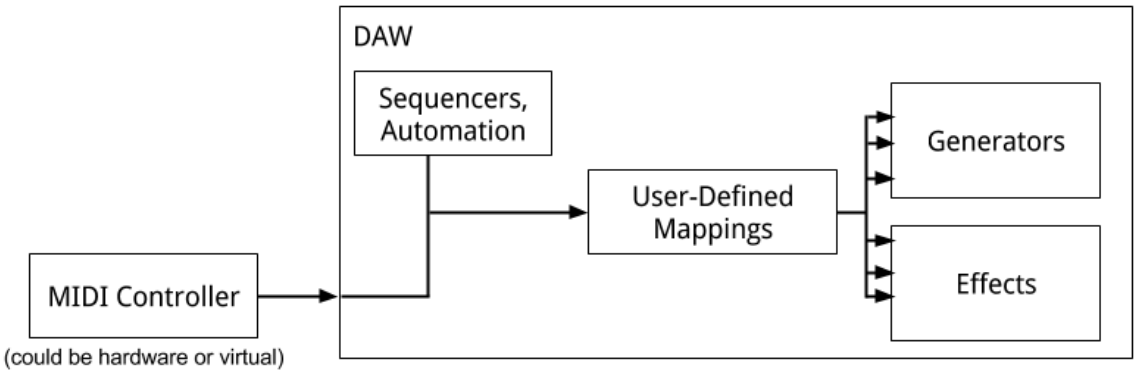

Figure 2: An overview of how MIDI is used by a DAW. The DAW mediates the flow of MIDI data from controllers to generators, which create audio in response.

controlled by the body.

However, this project was started in February 2012, when the Leap Motion dev kits had only just been released. As such, no similar work for the Leap Motion yet existed. For this reason, I can proudly claim that this project is part of the vanguard of Leap Motion-based musical applications.

2.3 Concurrent Work

As I predicted at the beginning of this project (c.f. my proposal), it is no longer unique, and similar projects do exist. Related work that has emerged since the start of this project include GecoMidi [5] and AeroMidi [1]. Steinberg have implemented native support for the Leap and other NUI in the latest version of Cubase, their flagship DAW [4].

3 Concept and Features

My intention was to design a tool for myself, one that would fit into my own preferred toolset and workflow as an electronic musician. Since my toolset is typical, I reasoned that if I created a tool that I liked using, many other musicians would find it useful as well.

My current solution for performance is a MIDI keyboard and a Korg nanoKontrol. I use the keyboard for triggering notes and the nanoKontrol as a general tool for recording automation, expression and "humanized" performances.

MIDIFlapper was conceived primarily as a replacement for the nanoKontrol and its array of knobs and sliders. I identified several deficiencies of the nanoKontrol which I hoped to improve with the MIDIFlapper solution. First, forming the mental connection between twisted

knob and heard effect is surprisingly non-intuitive. I wanted to see if binding sounds to my hand had a better "feel." Second, it is impossible to move more than two or three knobs/sliders simultaneously, whereas a Leap Motion-based interface can deliver simultaneous manipulation of ten times that amount of parameters. Thirdly, knobs and sliders are slow, precision instruments. Moving them in complex, quick rhythms is very difficult. The artist's unrestricted hand in the air should have no such limitations. Finally, it is not much fun to watch someone use the nanoKontrol. This is not a very important drawback from a UI standpoint, but to some users it is an essential feature nonetheless. I hope that a MIDIFlapper performance will be more engaging.

4 Requirements

4.1 How to Use a Hardware MIDI Controller

In order to understand how the app is intended to behave, you must first understand the hardware controllers that the app emulates. MIDIFlapper is a virtual MIDI controller. It emits MIDI messages identical to those that come from a "normal" controller, which is typically a piece of hardware. See section 1.3 for more information on MIDI controllers.

Each of the knobs and sliders on the controller seen in Figure 3 produce MIDI values when they are operated by the musician. These MIDI values are interpreted

Figure 3: The Korg NanoKontrol MIDI controller

by the DAW and used to set the values of user-mapped parameters to reflect the knob twist.

DAWs typically allow the user to assign a physical control (knob or slider) to an internal parameter (filter cutoff, decay length, etc.) using a "MIDI learn mode" function. The mapping process has three steps. First, the user enters MIDI learn mode. Then, the user selects an internal parameter to be modulated. Finally, the user sends a MIDI message from the control surface by e.g. turning the knob they wish to assign to the internal control.

4.2 Use Cases

4.3 Composition

I use MIDI controllers exclusively for composition. I do not perform live. When I use MIDI controllers, I record short sections of performance for single musical elements or layers, then stitch those separate performances together *post facto*.

As an artist, I want something that

facilitates creative expression, allows for happy accidents, and lets me explore sonic spaces. The key verb is explore. The antithesis of this exploration is spending a lot of time on boilerplate like setting up the device, rigging parameters, etc. It's important that the time required to use the device is absolutely minimal so as not to break my creative flow.

4.4 Live Performance

Others' use cases may include mine but may also include live performance. The multi-dimensional control afforded by the Leap Motion, the ability to control multiple parameters simultaneously, makes it ideal for complex live performance. With standard MIDI controllers, it's impossible to control more than one or two parameters with each hand. The new natural school of interfaces makes it possible to play computers like real instruments, controlling many aspects of the sound easily, intuitively, and emergently.

Live performance introduces several new requirements. The software must be completely reliable. It can't crash during a performance, and the risk of accidental clicks ruining somthing must be absolutely minimal. Reliability becomes the critical quality.

5 Architecture

5.1 Data Flow

The ultimate goal is to create MIDI data for consumption by the DAW. First, the user interacts with the Leap Motion hardware. This generates Frame objects which MIDIFlapper subscribes to. Based on the user's preferences, these data are then used to generate MIDI control-change messages which the DAW can interpret as a physical control.

6 Tools

The Leap API comes in C++ C#, Objective-C, Java, JavaScript, and Python. I chose to implement my app in Java because I am familiar with it, there is cross-platform support, and the standard libraries include MIDI operations.

For this project I used my version control solution of choice, git-plus-GitHub, and my Java IDE of choice, IntelliJ IDEA. The app's GUI is in Java's Swing. Tests were written in JUnit 4.

Virtual MIDI loopback cable utility LoopBe1 by nerds.de was used to provide an interface between the MIDIFlapper application and the DAW.

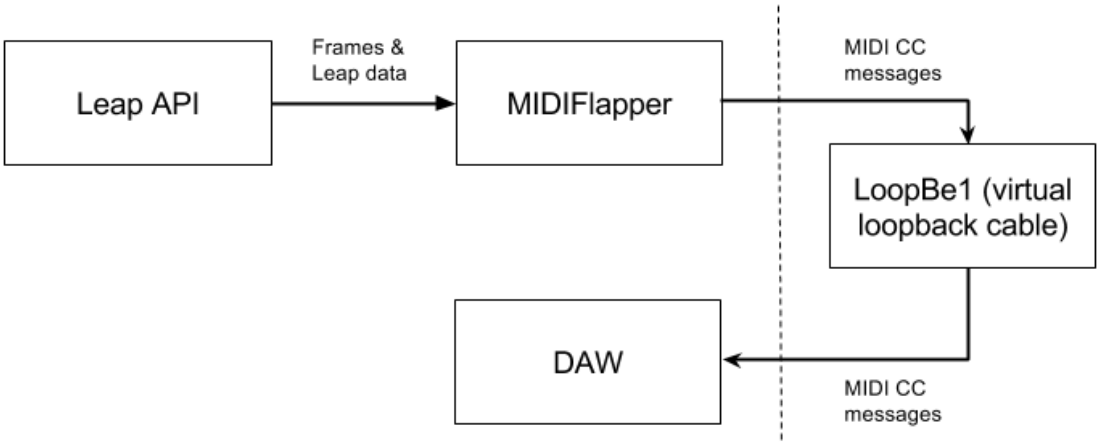

Figure 4: MIDIFlapper's place in the generation of MIDI data for consumption by the DAW.

7 Design Decisions

7.1 Data Model

7.1.1 Transform

Transforms are the heart of the application. A Transform is capable of accepting a Frame object (which originates from the Leap API) and returning a 0-127 value based on the Frame data. For example, a Transform based on the Finger X Axis takes a Finger from the Frame data, examines its position on the x-axis, and normalizes that position between a minimum and a maximum x-axis value to obtain a 0-127 value.

7.1.2 Options

Every Transform is capable of aggregating a list of Options. Options are user-adjustable parameters for the Transform's processing. To continue our earlier example, where the minimum and maximum x-axis values are in space can be adjusted to the user's preference with two Options.

7.1.3 Control

A Control encapsulates a Transform. It has a current state, reflecting the last 0-127 value that the Transform returned. It is an observable object; when its state changes, the view updates to match and new MIDI data can be emitted from the application.

7.2 View Architecture

Refer to figure 5 for this section.

One of the requirements for the project was great extensibility and modularity. There are two layers of modularity. New Controls can be easily "plugged in" to

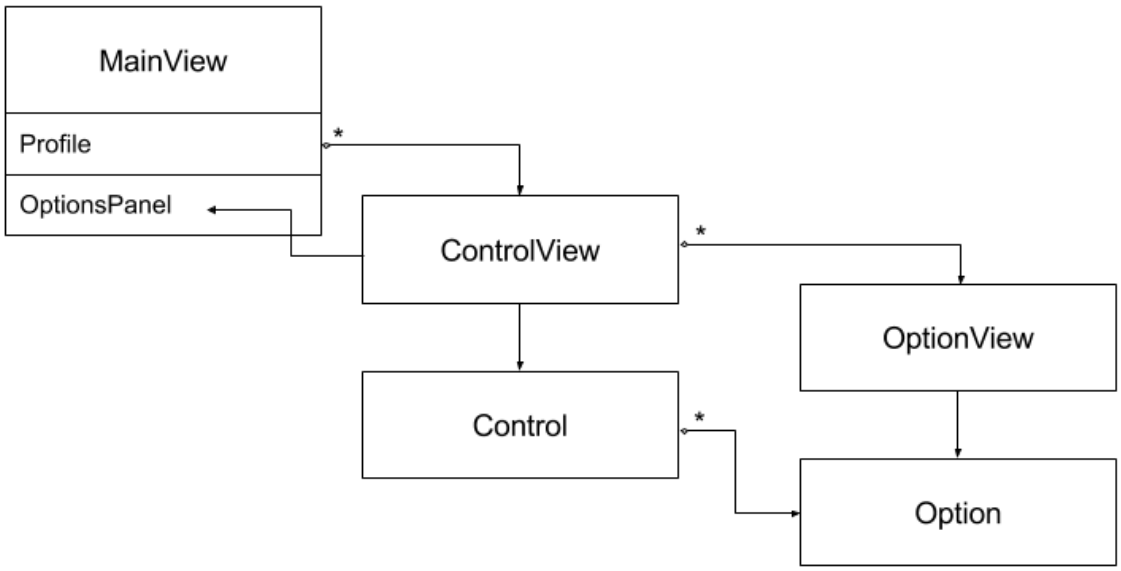

Figure 5: The structure of Views classes

the application, and new Options can be "plugged in" to Controls. Each of these elements—Controls and Options—have their own custom UI that they define for themselves. Every Control and Option knows not only its internal state, but also what it looks like.

This suggested that Controls be split into Control and ControlView classes (similarly, Options also have OptionViews). The view needs to know about the Control/Option, but the reverse is not true. Also, the main view needs only to know about its child Views.

There is one complication—the main view owns the UI element that Options appear in. So each ControlView has to keep a reference to a its parent, which is unfortunate.

7.3 Strings Class

I centralized the application's strings in the AppStrings class. I was inspired by Android's architecture, which encourages the developer to keep their strings in a resource file for internationalization/localization purposes. I don't think I'll ever use it for that reason; I just wanted to avoid hardcoding string literals into my UI.

8 Development Stories and Useful Findings

8.1 Interface or Abstract Class?

One lesson I learned and put to immediate concrete use was when to use an interface and when to use a concrete class. Obviously an abstract class can define data and behavior whereas an interface cannot, but what does this mean on a pragmatic level? An interesting and useful heuristic is that an abstract class represents an *is-a* relationship, whereas an interface represents a *can-do-this* relationship.

8.2 Throwing Out The GUI Designer

I spent a lot of time wrestling with IntelliJ's GUI designer. It is designed to work with Swing, but I found that I was spending far too much time trying to do very simple things. Ultimately I threw out all the automatically generated code and closed the WYSIWYG window, diagrammed out my UI on paper, and wrote the entire UI in code, with member fields and setup methods. This took much less time and it worked exactly as I expected it to.

8.3 Comments on the Leap API

I took many notes on the Leap Motion's API and specifically on the data it exposes in the Frame objects it generates every time hands enter its detection range. These objects contain descriptions of the hands, their positions, velocities, and shape; and the fingers and other "Pointables" (e.g. chopsticks and pencils) that are in the frame, with all their orientations, velocities etc.

Any of these data points could serve as a Control. It's just a matter of taking that data, performing processing on it, and turning it into a number between 0-127. But what kinds of processing make *good* Controls? The resulting data can't have much jitter in it, nor much extra lag introduced by extra processing on the LEAP's side—the control must feel like it is connected directly to the user's hand. Ideally, also, the data point is intuitive and useful for the user to manipulate and involves simple hand movements which lend themselves to precision and rhythm. Collectively, these two criteria are responsible for the mapping's "feel."

Here are my findings on what parts of the data yield good "feel." These notes apply to the v1.2 release of the LEAP API.

I found that:

- Finger X/Y/Z positions are exactly as you would expect. Conceptually simple, with a fast and precise response. It's clear there's no

latency-inducing post-processing between the movement and the control.

- Palm X/Y/Z positions are synthesized by the API from finger positions, and are consequently a bit unreliable. The palm detection can be a bit spotty as fingers flicker in and out of existence.

- Hand Roll is excellent. It responds quickly to the hand waggling back and forth, and feels great to map to sonic timbres. Synergizes well with hand height and left-to-right movement.

- Hand Pitch is similarly good.

- Hand Yaw is useless. Unresponsive and jittery, it doesn't feel like it's connected to my hand in any way.

- Hand Radius reflects how big a ball would be formed if the curve of the user's hand and fingers were extended. It is sluggish due to the processing required and what is clearly some averaging and anti-jitter introduced by the API processing; however, it could be very useful as a slow and imprecise timbral modifier.

- Stabilized Fingertips are for when you need a less reponsive but smoother fingertip experience I guess. I have known some parameters to be chaotically sensitive, so a steady hand is warranted at times.

9 Future Development

9.1 Known Bugs

Late in development, once I had added a large number of Controls to the control panel, my application began to frequently trigger LoopBe1's flood/short-circuit detection. LoopBe1[6], the virtual MIDI loopback cable my project relies on, mutes the cable if a certain bandwidth of MIDI data is exceeded. In order to unmute, the user has to click LoopBe1's taskbar icon and use a dialog box. In a working setting, this is an unacceptable annoyance; in a performance setting, this would bring the performance to a disastrous halt. The short-circuit detection was triggered almost immediately every time the user's hand entered the Leap's detection radius.

As a stopgap solution to the problem, I implemented a short queue in the MIDIInterface class. Instead of sending the message immediately, sendMessage() adds the message to the queue. This queue is flushed on a timer.

This adds latency to the data flow. I have not measured this latency and it doesn't feel significant to me, but I am not comfortable adding any quantity of latency to the application, as it is very latency-dependent. Ideally this would be replaced by another flood-prevention mechanism or LoopBe1 would be replaced by another system.

9.2 Features Wishlist

9.2.1 Better controls management

At the moment, every control I've created is in the active list. The long list is difficult to scan and navigate. This results in a huge volume of MIDI data any time the Leap is used. In combination with the anti-feedback-detection solution (see section 9.1), this may result in increased latency. A good feature to add, therefore, is a way to add and remove Controls.

9.2.2 Save/Load Profiles

Users should be able to save the changes they've made to Controls' options panes, the MIDI out they are using, and other state data as a Profile. This facilitates the usage of different profiles for different songs, each of which can have its own MIDI automation routing setup. Similarly, it is important that each Control points to the same MIDI channel and number, even after controls management is added.

9.2.3 More controls

Controls are the central feature of the application, and I think I've only scratched the surface of what's exposed by the Leap Motion's API.

9.2.4 More default Options

Each Control could have snap-to-default, deadzones, and other useful tools. The extensible nature of the architecture means these would be simple to add.

9.2.5 User-specified MIDI channels

The user may wish to bind a Control to the modwheel, pitchbend, aftertouch, pressure, pitchbend, or other special named channels. At the moment, channels are sequentialliy assigned to Controls, and if I wanted something to correspond to, say, the mod wheel specifically, I would be out of luck. In fact, the UI doesn't even display which Controls correspond to what channels.

9.2.6 Break dependence on LoopBe1

MIDIFlapper currently relies on LoopBe1 by nerds.de [6] to create a virtual loopback cable between MIDIFlapper and the user's DAW. Further research is needed to find whether this could be done within MIDIFlapper.

9.2.7 Update to the latest verion of the LEAP API

Since the conclusion of my senior project, the Leap Motion has announced its 2.0 public release. This probably changes my estimation of which data points are worthless and prompt the creation of new Controls.

9.2.8 User extensibility

The highly modular nature of the architecture means that users should be able to supply their own Controls or Options to be added to the existing set at runtime.

10 Conclusions

10.0.9 I'm impressed with the Leap Motion.

The Microsoft Kinect promised a lot and didn't deliver. In the wake of that disappointment, I was guarded about the claimed abilities of the Motion. However, I was surprised at the accuracy of the detection, the low latency, and the cleanliness of the API. It's ideal for this kind of project. However, it is billed as a replacement for more typical interaction interfaces, and I think that that's inappropriate. It cannot beat mouse, keyboard or touchscreen at their own games—nothing beats a mouse's pointing ability, and nothing types like a keyboard. However, nothing does 3D hand-tracking like the LEAP does, and that is why it succeeds for my application.

10.0.10 Swing is OK, I guess.

I hear a lot of criticism of Swing, and after my experience with it on this project I can see why. The WYSIWYG editor didn't work, I didn't understand the way Swing intended me to use it. Consequently, I feel I ended up writing a lot of code which may have "misused" Swing as a thin UI framework instead of leveraging its datamodel-view functionality.

10.0.11 I learned a lot.

I think this document is adequate proof of that. The scope of this project was just on the edge of my abilities as a software engineer, but I felt very competent throughout the design and implementation process. I felt that my education had prepared me really well for this project.

10.0.12 I should test more next time.

I know it's best practice and yet I need to develop my skills in the testing area. I wrote a lot of JUnit for the data models, but did not so much as research UI testing tools. The UI remains untested and the MIDI interface is untested, and as such there's no end-to-end tests that test whether the proper MIDI was output based on certain UI input.

References

[1] *AeroMidi on Leap Motion Airspace App Store*. URL: https://airspace.leapmotion.com/apps/aeromidi.

[2] Livingston et al. "Performance Measurements for the Microsoft Kinect Skeleton". In: (2012).

[3] Raffi Bedikian. *Understanding Latency: Part 1 - Leap Motion Developer Blog*. URL: https://developer.leapmotion.com/blog/understanding-latency-part-1/.

[4] *Cubase IC Air*. URL: http://www.steinberg.net/en/products/accessories/cubase_ic_air.html.

[5] *GecoMidi on Leap Motion Airspace App Store*. URL: https://airspace.leapmotion.com/apps/geco-midi.

[6] Daniel Schmitt. *LoopBe1 - A Free Virtual MIDI Driver - Nerds.de*. URL: http://www.nerds.de/en/loopbe1.html.

Appendices

A Source Code

A.1 AppStrings.java

```
1  package leapmidi;
2
3  import java.util.Dictionary;
4  import java.util.Hashtable;
5
6  /**
7   * AppStrings
8   */
9  public class AppStrings
10 {
11    private static Hashtable<String, String> strings =
12       new Hashtable<String, String>();
13
14    static {
15       strings.put("buttonSaveProfileText", "Save Profile...");
16       strings.put("buttonLoadProfileText", "Load Profile...");
17       strings.put("midiComboBoxLabelText", "MIDI Out Device:");
18       strings.put("MainWindowTitle", "MIDIFlapper");
19    }
20
21    public static String get(String key)
22    {
23       if (strings.containsKey(key))
24          return strings.get(key);
25       else
26          return "%" + key + "%";
27    }
28 }
```

A.2 Control.java

```
1  package leapmidi;
```

```
2
3   import com.leapmotion.leap.Frame;
4
5   import java.io.Serializable;
6   import java.util.ArrayList;
7   import java.util.List;
8   import java.util.Observable;
9
10  /**
11   * A Control is a CC channel and a Transform.
12   */
13  public class Control
14  {
15     private Transform transform;
16     private MIDIAddress address;
17     private String name;
18     private int value;
19     private List<ControlObserver> obesrvers;
20
21     public Control(String name, MIDIAddress address, Transform transform)
22     {
23        if (address == null)
24           address = new MIDIAddress(1, 1);
25
26        this.address = address;
27        this.transform = transform;
28        this.name = name;
29
30        this.obesrvers = new ArrayList<ControlObserver>();
31     }
32
33     public void setValue(int newValue)
34     {
35        newValue = Math.max(0, Math.min(newValue, MIDIInterface.MIDI_MAXVAL));
36        if (this.value != newValue) {
37           value = newValue;
38           notifyObservers();
39        }
```

```
40      }
41
42      public String getName()
43      {
44         return name;
45      }
46
47      public void setName(String name)
48      {
49         if (!name.equals(this.name)) {
50            this.name = name;
51            notifyObservers();
52         }
53      }
54
55      private void notifyObservers()
56      {
57         for (ControlObserver co : this.obesrvers) {
58            co.onControlChange(this);
59         }
60      }
61
62      public void setAddress(MIDIAddress address)
63      {
64         this.address = address;
65      }
66
67      public MIDIAddress getMIDIAddress()
68      {
69         return this.address;
70      }
71
72      public void acceptFrame(Frame frame)
73      {
74         int newValue = transform.getValue(frame);
75         if (newValue != -1)
76            this.setValue(newValue);
77      }
```

```
78
79      public int getValue()
80      {
81         return value;
82      }
83
84      public void addObserver(ControlObserver controlObserver)
85      {
86         this.obesrvers.add(controlObserver);
87      }
88   }
```

A.3 ControlObserver.java

```
1   package leapmidi;
2
3   /**
4    * Created by Mark Henry on 4/27/14.
5    */
6   public interface ControlObserver
7   {
8      public void onControlChange(Control control);
9   }
```

A.4 ControlView.java

```
1   package leapmidi;
2
3   import javax.swing.*;
4   import javax.swing.border.Border;
5   import javax.swing.event.ChangeEvent;
6   import javax.swing.event.ChangeListener;
7   import java.awt.*;
8   import java.awt.event.ActionEvent;
9   import java.awt.event.ActionListener;
10  import java.util.ArrayList;
11  import java.util.List;
12  import java.util.Observable;
```

```
13  import java.util.Observer;
14
15  /**
16   * ControlView
17   */
18  public class ControlView implements ControlObserver, ChangeListener
19  {
20     private JLabel nameLabel;
21     private JSlider slider;
22     private Control control;
23     private JButton showOptionsButton;
24     private JPanel optionsPanel;
25     private List<OptionView> optionViews = new ArrayList<OptionView>();
26
27     public ControlView(Control control)
28     {
29        this.control = control;
30        control.addObserver(this);
31        slider = new JSlider(0, 127, 0);
32        nameLabel = new JLabel(control.getName());
33        slider.addChangeListener(this);
34        showOptionsButton = new JButton("");
35        showOptionsButton.addActionListener(new ActionListener()
36        {
37           @Override
38           public void actionPerformed(ActionEvent e)
39           {
40              fillOptionsPanel();
41           }
42        });
43     }
44
45     public void setOptionsPanel(JPanel optionsPanel)
46     {
47        this.optionsPanel = optionsPanel;
48     }
49
50     public void setOptionViews(List<OptionView> views)
```

```
51      {
52         this.optionViews = views;
53      }
54
55      /**
56       * Called whenever the Control changes state, for any reason
57       */
58      @Override
59      public void onControlChange(Control control)
60      {
61         slider.setValue(control.getValue());
62         nameLabel.setText(control.getName());
63      }
64
65      /**
66       * Called only when slider is manually moved by the user.
67       *
68       * @param e a ChangeEvent object
69       */
70      @Override
71      public void stateChanged(ChangeEvent e)
72      {
73         control.setValue(slider.getValue());
74      }
75
76      private void fillOptionsPanel()
77      {
78         optionsPanel.removeAll();
79         for (OptionView optionView : optionViews)
80         {
81            JPanel subPanel = new JPanel();
82            subPanel.setMaximumSize(new Dimension(Short.MAX_VALUE, 30));
83
84            optionView.fillPanel(subPanel);
85            optionsPanel.add(subPanel);
86         }
87         optionsPanel.validate();
88      }
```

```
89
90      public void fillPanel(JPanel panel)
91      {
92         panel.removeAll();
93         panel.setLayout(new BorderLayout(5, 5));
94         panel.add(nameLabel, BorderLayout.WEST);
95         panel.add(slider, BorderLayout.CENTER);
96         panel.add(showOptionsButton, BorderLayout.EAST);
97      }
98
99      public Control getControl()
100     {
101        return control;
102     }
103  }
```

A.5 ControlViewFactory.java

```
1   package leapmidi;
2
3   import com.leapmotion.leap.Frame;
4
5   import java.util.Arrays;
6
7   /**
8    * ControlViewFactory
9    */
10  public class ControlViewFactory
11  {
12     public static ControlView
13        makeMinMaxControl(String name, int min,
14           int minInit, int maxInit, int max,
15           final ValueExtractor valueExtractor)
16     {
17        final OptionView minOptionView =
18           OptionViewFactory.makeSliderOption(min, minInit, max, "Min");
19        final OptionView maxOptionView =
20           OptionViewFactory.makeSliderOption(min, maxInit, max, "Max");
```

```
21
22          Transform minMaxTransform = new Transform()
23          {
24             @Override
25             public int getValue(Frame frame)
26             {
27                int min = minOptionView.getOption().getValue();
28                int max = maxOptionView.getOption().getValue();
29                if (frame.fingers().isEmpty())
30                   return -1;
31                else {
32                   int pos = valueExtractor.valueFromFrame(frame);
33                   if (max == min)
34                      return -1;
35                   else
36                      return (127 * (pos - min)) / (max - min);
37                }
38             }
39          };
40          Control minMaxControl = new Control(name, null, minMaxTransform);
41          ControlView minMaxControlView = new ControlView(minMaxControl);
42          minMaxControlView.setOptionViews(
43             Arrays.asList(minOptionView, maxOptionView));
44          return minMaxControlView;
45       }
46    }
```

A.6 DefaultProfile.java

```
1   package leapmidi;
2
3   import com.leapmotion.leap.FingerList;
4   import com.leapmotion.leap.Frame;
5
6   import java.util.ArrayList;
7   import java.util.List;
8
9   /**
```

```
10    * DefaultProfile
11    */
12   public class DefaultProfile
13   {
14      private static Profile defaultControls = null;
15
16      public static Profile getDefaultControls()
17      {
18         if (DefaultProfile.defaultControls == null)
19            DefaultProfile.defaultControls = constructDefaultControlsList();
20         return DefaultProfile.defaultControls;
21      }
22
23      private static Profile constructDefaultControlsList()
24      {
25         List<ControlView> views = new ArrayList<ControlView>();
26
27         views.add(ControlViewFactory.makeMinMaxControl(
28            "Hand X Axis", -200, -100, 100, 200,
29               new ValueExtractor()
30               {
31                  @Override
32                  public int valueFromFrame(Frame frame)
33                  {
34                     return (int) frame.hands().rightmost().palmPosition().getX();
35                  }
36               }
37         ));
38
39         views.add(ControlViewFactory.makeMinMaxControl(
40            "Hand Y Axis", 10, 100, 400, 800,
41               new ValueExtractor()
42               {
43                  @Override
44                  public int valueFromFrame(Frame frame)
45                  {
46                     return (int) frame.hands().rightmost().palmPosition().getY();
47                  }
```

```
48              }
49          ));
50
51          views.add(ControlViewFactory.makeMinMaxControl(
52             "Hand Z Axis", -200, -100, 100, 200,
53                new ValueExtractor()
54                {
55                   @Override
56                   public int valueFromFrame(Frame frame)
57                   {
58                      return (int) frame.hands().rightmost().palmPosition().getZ();
59                   }
60                }
61          ));
62
63          views.add(ControlViewFactory.makeMinMaxControl(
64             "Hand Pitch", -150, -50, 125, 150,
65                new ValueExtractor()
66                {
67                   @Override
68                   public int valueFromFrame(Frame frame)
69                   {
70                      float pitch = frame.hands().rightmost().direction().pitch();
71                      return (int) (100 * pitch);
72                   }
73                }
74          ));
75
76          views.add(ControlViewFactory.makeMinMaxControl(
77             "Hand Waggle", -300, -100, 100, 300,
78                new ValueExtractor()
79                {
80                   @Override
81                   public int valueFromFrame(Frame frame)
82                   {
83                      FingerList fingers = frame.hands().rightmost().fingers();
84                      float leftHeight = fingers.leftmost().tipPosition().getY();
85                      float rightHeight = fingers.rightmost().tipPosition().getY();
```

```
86                  return (int) (leftHeight - rightHeight);
87               }
88            }
89         ));
90
91         views.add(ControlViewFactory.makeMinMaxControl(
92            "Hand Radius", 0, 50, 100, 150,
93               new ValueExtractor()
94               {
95                  @Override
96                  public int valueFromFrame(Frame frame)
97                  {
98                     return (int) frame.hands().rightmost().sphereRadius();
99                  }
100            }
101        ));
102
103        views.add(ControlViewFactory.makeMinMaxControl(
104           "Stabilized Fingertip X", -200, -100, 100, 200,
105              new ValueExtractor()
106              {
107                 @Override
108                 public int valueFromFrame(Frame frame)
109                 {
110                    return (int) frame.pointables().frontmost().
111                       stabilizedTipPosition().getX();
112                 }
113              }
114        ));
115
116        views.add(ControlViewFactory.makeMinMaxControl(
117           "Stabilized Fingertip Y", 10, 100, 400, 800,
118              new ValueExtractor()
119              {
120                 @Override
121                 public int valueFromFrame(Frame frame)
122                 {
123                    return (int) frame.pointables().frontmost().
```

```
124                     stabilizedTipPosition().getY();
125               }
126            }
127         ));
128
129         views.add(ControlViewFactory.makeMinMaxControl(
130            "Stabilized Fingertip Z", -200, -100, 100, 200,
131               new ValueExtractor()
132               {
133                  @Override
134                  public int valueFromFrame(Frame frame)
135                  {
136                     return (int) frame.pointables().frontmost()
137                        .stabilizedTipPosition().getZ();
138                  }
139               }
140         ));
141
142         views.add(ControlViewFactory.makeMinMaxControl(
143            "Fingertip X", -200, -100, 100, 200,
144               new ValueExtractor()
145               {
146                  @Override
147                  public int valueFromFrame(Frame frame)
148                  {
149                     return (int) frame.pointables().frontmost()
150                        .stabilizedTipPosition().getX();
151                  }
152               }
153         ));
154
155         views.add(ControlViewFactory.makeMinMaxControl(
156            "Fingertip Y", 10, 100, 400, 800,
157               new ValueExtractor()
158               {
159                  @Override
160                  public int valueFromFrame(Frame frame)
161                  {
```

```
162                     return (int) frame.pointables().frontmost()
163                        .tipPosition().getY();
164                  }
165               }
166         ));
167
168         views.add(ControlViewFactory.makeMinMaxControl(
169            "Fingertip Z", -200, -100, 100, 200,
170               new ValueExtractor()
171               {
172                  @Override
173                  public int valueFromFrame(Frame frame)
174                  {
175                     return (int) frame.pointables().frontmost()
176                        .tipPosition().getZ();
177                  }
178               }
179         ));
180
181         return new Profile(views);
182      }
183
184      private static int valueFromFrame(Frame frame)
185      {
186         return (int)frame.fingers().leftmost().tipPosition().getY();
187      }
188   }
```

A.7 MIDIAddress.java

```
1   package leapmidi;
2
3   /**
4    * A MIDIAddress contains a CC channel and controller number. It's essentially an
5    */
6   public class MIDIAddress
7   {
8      private static final int MAX_CHANNEL = 15;
```

```
9       private static final int MAX_CONTROLLER = 127;
10      public int channel;
11       public int controller;
12
13       public MIDIAddress(int channel, int controller)
14       {
15           this.channel = channel;
16           this.controller = controller;
17       }
18
19      public void increment()
20      {
21         this.controller++;
22         if (this.controller > MAX_CONTROLLER) {
23            this.controller = 0;
24            this.channel++;
25         }
26         if (this.channel > MAX_CHANNEL) {
27            this.channel = 0;
28         }
29      }
30  }
```

A.8 MIDIInterface.java

```
1   package leapmidi;
2
3   import javax.sound.midi.*;
4   import java.util.LinkedList;
5   import java.util.Queue;
6   import java.util.concurrent.Executors;
7   import java.util.concurrent.ScheduledExecutorService;
8   import java.util.concurrent.ScheduledFuture;
9   import java.util.concurrent.TimeUnit;
10
11  /**
12   * MIDIInterface
13   */
```

```
14  public class MIDIInterface
15  {
16     public static final int MIDI_MAXVAL = 127;
17     private static final int MIDI_PERIOD_MILLISECONDS = 5;
18     private static final int MAX_MESSAGES_PER_PERIOD = 10;
19     public static int messagesThisPeriod = 0;
20     private static Queue<MidiMessage> messageQueue =
21        new LinkedList<MidiMessage>();
22     private static MidiDevice midiOutDevice = null;
23     private static final ScheduledExecutorService scheduler =
24        Executors.newScheduledThreadPool(1);
25     private static ScheduledFuture<?> periodHandle;
26
27     {
28        final Runnable midiPeriodRunner = new Runnable()
29        {
30           @Override
31           public void run()
32           {
33              everyPeriod();
34           }
35        };
36
37        periodHandle = scheduler.scheduleAtFixedRate(
38           midiPeriodRunner, 0, MIDI_PERIOD_MILLISECONDS, TimeUnit.MILLISECONDS);
39     }
40
41
42     public Object[] getAvailableMIDIDevices()
43     {
44        return MidiSystem.getMidiDeviceInfo();
45     }
46
47     public void setMidiOutDevice(Object newDevice)
48        throws MidiUnavailableException
49     {
50        this.midiOutDevice = MidiSystem.getMidiDevice(
51           (MidiDevice.Info) newDevice);
```

```
52          this.midiOutDevice.open();
53          System.out.println("Outputting to " + newDevice.toString());
54       }
55
56       public void sendMessage(MIDIAddress address, int value)
57       {
58          try
59          {
60             messageQueue.add(new ShortMessage(ShortMessage.CONTROL_CHANGE,
61                   address.channel, address.controller, value));
62          }
63          catch (InvalidMidiDataException e)
64          {
65             System.err.println("MIDI interface error: Invalid MIDI data: " +
66                e.getLocalizedMessage());
67          }
68       }
69
70       private static void everyPeriod()
71       {
72          messagesThisPeriod = 0;
73
74          if (midiOutDevice == null) {
75             return;
76          }
77
78          long timeStamp = -1;
79          while (!messageQueue.isEmpty() && messagesThisPeriod
80                <= MAX_MESSAGES_PER_PERIOD) {
81             MidiMessage message = messageQueue.remove();
82
83             try {
84                midiOutDevice.getReceiver().send(message, timeStamp);
85                messagesThisPeriod++;
86             }
87             catch (MidiUnavailableException e)
88             {
89                System.err.println("MIDI interface error: MIDI Unavailable: "
```

```
90                  + e.getLocalizedMessage());
91               return;
92            }
93         }
94      }
95
96      public static void close()
97      {
98         periodHandle.cancel(true);
99      }
100  }
```

A.9 MainWindow.java

```
1   package leapmidi;
2
3   import com.leapmotion.leap.*;
4
5   import javax.swing.*;
6   import java.awt.*;
7   import java.awt.event.*;
8   import java.util.Observable;
9   import java.util.Observer;
10  import javax.sound.midi.*;
11  import javax.swing.border.Border;
12
13  import com.leapmotion.leap.Frame;
14
15  /**
16   * MainWindow
17   */
18  public class MainWindow extends Listener implements ControlObserver
19  {
20     private JPanel windowPanel = new JPanel();
21     private JSplitPane windowSplitPane;
22        private JScrollPane optionsScrollPane;
23        private JPanel optionsPanel = new JPanel();
24        private JPanel leftPanel = new JPanel();
```

```
25            private JPanel topPanel = new JPanel();
26               private JButton buttonLoadProfile =
27                  new JButton(AppStrings.get("buttonLoadProfileText"));
28               private JButton buttonSaveProfile =
29                  new JButton(AppStrings.get("buttonSaveProfileText"));
30               private JLabel midiComboBoxLabel =
31                  new JLabel(AppStrings.get("midiComboBoxLabelText"));
32               private JComboBox midiComboBox = new JComboBox();
33            private JScrollPane controlsScrollPane;
34               private JPanel controlsPanel = new JPanel();
35
36      private Profile currentProfile;
37      private MIDIInterface midiInterface = new MIDIInterface();
38      private Controller controller;
39
40      private void createUIComponents()
41      {
42         topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.X_AXIS));
43         //topPanel.add(buttonLoadProfile);
44         //topPanel.add(Box.createRigidArea(new Dimension(2, 0)));
45         //topPanel.add(buttonSaveProfile);
46         //topPanel.add(Box.createRigidArea(new Dimension(7, 0)));
47         topPanel.add(midiComboBoxLabel);
48         topPanel.add(Box.createRigidArea(new Dimension(2, 0)));
49         topPanel.add(midiComboBox);
50         midiComboBox.setMinimumSize(new Dimension(100, 0));
51
52         controlsPanel.setLayout(
53            new BoxLayout(controlsPanel, BoxLayout.PAGE_AXIS));
54         controlsScrollPane = new JScrollPane(controlsPanel);
55
56         leftPanel.setLayout(new BorderLayout(5,5));
57         leftPanel.add(topPanel, BorderLayout.NORTH);
58         leftPanel.add(controlsScrollPane, BorderLayout.CENTER);
59
60         optionsPanel.setLayout(
61            new BoxLayout(optionsPanel, BoxLayout.PAGE_AXIS));
62         optionsScrollPane = new JScrollPane(optionsPanel);
```

```
63          optionsScrollPane.setPreferredSize(new Dimension(300, 0));
64
65          windowSplitPane = new JSplitPane(
66             JSplitPane.HORIZONTAL_SPLIT, leftPanel, optionsScrollPane);
67          windowPanel.setLayout(new BorderLayout());
68          windowPanel.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
69          windowPanel.add(windowSplitPane, BorderLayout.CENTER);
70       }
71
72       public MainWindow()
73       {
74          initMainWindow();
75          initController();
76       }
77
78       private void initController()
79       {
80          this.controller = new Controller(this);
81          controller.setPolicyFlags(
82             Controller.PolicyFlag.POLICY_BACKGROUND_FRAMES);
83       }
84
85       private void initMainWindow()
86       {
87          createUIComponents();
88          initMIDIDevices();
89          loadProfile(DefaultProfile.getDefaultControls());
90       }
91
92       private void initMIDIDevices()
93       {
94          // Init midiComboBox with available MIDI devices
95          DefaultComboBoxModel model = new DefaultComboBoxModel(
96             midiInterface.getAvailableMIDIDevices());
97          midiComboBox.setModel(model);
98
99          midiComboBox.addActionListener(new ActionListener()
100         {
```

```
101             @Override
102             public void actionPerformed(ActionEvent e)
103             {
104                 try
105                 {
106                     midiInterface.setMidiOutDevice(midiComboBox.getSelectedItem());
107                 }
108                 catch (MidiUnavailableException e1)
109                 {
110                     JOptionPane.showMessageDialog(windowPanel, e1.getMessage(),
111                         "MIDI Unavailable", JOptionPane.ERROR_MESSAGE);
112                 }
113             }
114         });
115     }
116
117     private void loadProfile(Profile profile)
118     {
119         this.currentProfile = profile;
120         for (ControlView cv : profile.getControlViews()) {
121             Control c = cv.getControl();
122             c.addObserver(this);
123
124             renderControlViewToControlsPane(cv);
125         }
126     }
127
128     private void renderControlViewToControlsPane(ControlView cv)
129     {
130         cv.setOptionsPanel(optionsPanel);
131
132         JPanel subPanel = new JPanel();
133         subPanel.setBorder(BorderFactory.createEmptyBorder(2,4,2,4));
134         subPanel.setMaximumSize(new Dimension(Short.MAX_VALUE, 30));
135
136         controlsPanel.add(subPanel);
137         cv.fillPanel(subPanel);
138     }
```

```
139
140     public static void main(String[] args)
141     {
142        MainWindow window = new MainWindow();
143        JFrame frame = new JFrame(AppStrings.get("MainWindowTitle"));
144        frame.setContentPane(window.windowPanel);
145        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
146        frame.pack();
147        frame.setVisible(true);
148     }
149
150     @Override
151     public void onFrame(Controller controller)
152     {
153        Frame frame = controller.frame();
154
155        for (ControlView cv : currentProfile.getControlViews()) {
156           cv.getControl().acceptFrame(frame);
157        }
158     }
159
160     @Override
161     public void onInit(Controller controller)
162     {
163        System.out.println("Controller Initialized");
164     }
165
166     @Override
167     public void onConnect(Controller controller)
168     {
169        System.out.println("Controller Connnected");
170     }
171
172     @Override
173     public void onExit(Controller controller)
174     {
175        System.out.println("Controller Exited");
176     }
```

```
177
178     @Override
179     public void onDisconnect(Controller controller)
180     {
181        System.out.println("Controller Disconnected");
182     }
183
184     @Override
185     public void onControlChange(Control control)
186     {
187        midiInterface.sendMessage(control.getMIDIAddress(), control.getValue());
188     }
189  }
```

A.10 Option.java

```
1   package leapmidi;
2
3   import javax.swing.*;
4
5   /**
6    * An Option encapsulates an integer
7    */
8   public class Option
9   {
10     private int value;
11
12     public Option(int initialValue)
13     {
14        this.value = initialValue;
15     }
16
17     public int getValue()
18     {
19        return value;
20     }
21
22     public void setValue(int value)
```

```
23      {
24         this.value = value;
25      }
26   }
```

A.11 OptionView.java

```
1    package leapmidi;
2
3    import javax.swing.*;
4
5    /**
6     * OptionView
7     */
8    public abstract class OptionView
9    {
10      private String name;
11      private Option option;
12
13      public OptionView(Option option, String name)
14      {
15         this.option = option;
16         this.name = name;
17      }
18
19      public abstract void fillPanel(JPanel panel);
20
21      public String getName()
22      {
23         return name;
24      }
25
26      public Option getOption() { return option; }
27   }
28
```

A.12 OptionViewFactory.java

```
1  package leapmidi;
2
3  import com.sun.javaws.exceptions.InvalidArgumentException;
4
5  import javax.swing.*;
6  import javax.swing.border.Border;
7  import javax.swing.event.ChangeEvent;
8  import javax.swing.event.ChangeListener;
9  import java.awt.*;
10
11 /**
12  * OptionViewFactory
13  */
14 public class OptionViewFactory
15 {
16    public static final int BOOLEAN_FALSE = 0;
17    public static final int BOOLEAN_TRUE = 1;
18
19    // A numerical Option inside of a slider View
20    public static OptionView makeSliderOption(
21       final int min, final int initial, final int max, String name)
22    {
23       final Option intOption = new Option(initial);
24
25       return new OptionView(intOption, name)
26       {
27          JSlider slider = new JSlider(min, max, initial);
28          JLabel label = new JLabel(getName());
29
30          {
31             slider.addChangeListener(new ChangeListener()
32             {
33                @Override
34                public void stateChanged(ChangeEvent e)
35                {
36                   intOption.setValue(slider.getValue());
37                }
```

```
38             });
39          }
40
41          @Override
42          public void fillPanel(JPanel panel)
43          {
44             slider.setValue(this.getOption().getValue());
45             label.setText(this.getName());
46
47             panel.removeAll();
48             panel.setLayout(new BorderLayout(5, 5));
49             panel.add(label, BorderLayout.WEST);
50             panel.add(slider, BorderLayout.CENTER);
51          }
52       };
53    }
54
55    public static OptionView makeBooleanOption(final int initial, String name)
56    {
57       final Option boolOption = new Option(initial);
58
59       if (initial != BOOLEAN_FALSE && initial != BOOLEAN_TRUE)
60          throw new IllegalArgumentException(
61             "Initial value for boolean option must be one of " +
62             "OptionViewFactory.BOOLEAN_TRUE or BOOLEAN_FALSE");
63
64       return new OptionView(boolOption, name)
65       {
66          JCheckBox checkBox = new JCheckBox(getName(), initial == BOOLEAN_TRUE);
67
68          {
69             checkBox.addChangeListener(new ChangeListener()
70             {
71                @Override
72                public void stateChanged(ChangeEvent e)
73                {
74                   if (checkBox.isSelected())
75                      boolOption.setValue(BOOLEAN_TRUE);
```

```
76                      else
77                         boolOption.setValue(BOOLEAN_FALSE);
78                   }
79                });
80             }
81             @Override
82             public void fillPanel(JPanel panel)
83             {
84                checkBox.setText(this.getName());
85                panel.removeAll();
86                panel.setLayout(new BorderLayout());
87                panel.add(checkBox, BorderLayout.CENTER);
88             }
89          };
90       }
91    }
```

A.13 Profile.java

```
1   package leapmidi;
2
3   import java.io.Serializable;
4   import java.util.ArrayList;
5   import java.util.Collection;
6   import java.util.List;
7
8   /**
9    * A Profile aggregates ControlViews, each of which aggregates
10   * a Control and a list of OptionViews.
11   */
12  public class Profile implements Serializable
13  {
14     private List<ControlView> controlViews;
15
16     public Profile(Collection<ControlView> controlViews)
17     {
18        this.controlViews = new ArrayList<ControlView>();
19        for (ControlView controlView : controlViews) {
```

```
20            this.add(controlView);
21         }
22      }
23
24      public void add(ControlView controlView)
25      {
26         MIDIAddress midiAddress = controlView.getControl().getMIDIAddress();
27
28         while (!midiAddressFree(midiAddress))
29            midiAddress.increment();
30
31         controlView.getControl().setAddress(midiAddress);
32         controlViews.add(controlView);
33      }
34
35      private boolean midiAddressFree(MIDIAddress midiAddress)
36      {
37         return midiAddressFree(midiAddress.channel, midiAddress.controller);
38      }
39
40      private boolean midiAddressFree(int CCChannel, int CCController)
41      {
42         for (ControlView controlView : controlViews)
43         {
44            MIDIAddress addr = controlView.getControl().getMIDIAddress();
45            if (addr.controller == CCController && addr.channel == CCChannel)
46               return false;
47         }
48
49         return true;
50      }
51
52      public List<ControlView> getControlViews()
53      {
54         return controlViews;
55      }
56   }
```

A.14 Transform.java

```
1  package leapmidi;
2
3  import com.leapmotion.leap.Frame;
4
5  import java.util.List;
6
7  /**
8   * Defines a transform between the data in a Leap Frame object
9   *  and a [0-127] value.
10  */
11 public abstract class Transform
12 {
13    /**
14     * Interprets the Frame data and returns the [0-127] that the Frame
15     * represents.
16     * @param frame the Frame to interpret
17     * @return [0-127], or -1 if no data is available.
18     */
19    abstract public int getValue(Frame frame);
20 }
```

A.15 ValueExtractor.java

```
1  package leapmidi;
2
3  import com.leapmotion.leap.Frame;
4
5  /**
6   * Created by Mark Henry on 3/9/14.
7   */
8  public interface ValueExtractor
9  {
10    int valueFromFrame(Frame frame);
11 }
```