BirdFish Documentation

Release 0.1a

Preston Holmes

October 30, 2013

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

BirdFish is a tool to programatically and interactively control the behavior of lights or animatronics for the purpose of entertainment.

The overall design goal is to allow the person with the creative idea, to focus more on the process of designing the way the lights behave at a high level, and less about how to generate the timing required by the lighting control hardware. Currently BirdFish is still for the technically inclined, and is more about making really hard things possible than making standard stuff easy.

The features provide a way to abstract the complexities of the high channel counts and varying protocols used by the electronics hardware that control the lights.

The this provides an alternative to the status-quo of software which typically represents each channel as a row, or set of collapsed rows, and slices time into a number of discreet chunks resulting in a grid or matrix, where each box represents the intensity value of that channel or group for that slice of time.

There are two problems with that approach which this software attempts to solve. The first is that with ever increasing channel counts, and lights that use multiple channels, it gets very tedious to fill in each block of the grid, even with helper tools. Second, modifications to the display arrangement or to the sequence result in completely reworking the channel values to effect the desired changes.

1.1 The Components of BirdFish

BirdFish consists of a set of Elements, combined with control inputs, connected to output networks, combined form a show.

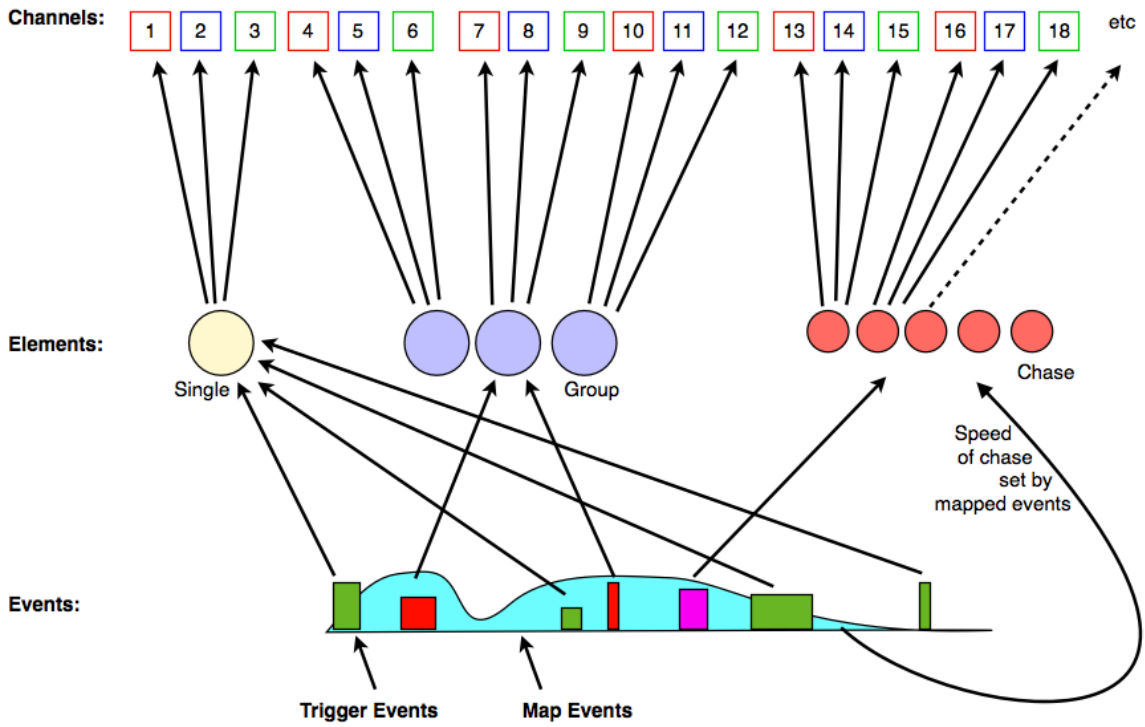
An Element is the fundamental building block of a show in Birdfish. In its most basic form, it is a single channel on a dimmer/controller. But even in this basic form, it offers some powerful features. Because a element is a object in software, it can have attributes that you can set or modify, or that can be programmed to change over time.

For an RGB light, an element wraps the three channels into one object. Chaning that light's "hue" attribute would automatically adjust the three RGB channels as needed.

Light Elements can be combined and nested in ways to create more complex elements. Many single lights can be combined into groups or chases, chases themselves can be grouped into other chases in a way that can provide for very complex effects, without worrying about how the low level channel data is generated.

1.2 Events control Light Elements

Events are the things that happen at a specific time to control an element. You can think of these roughly as the nubs on a music box wheel, or the notches in a player piano sheet.



There are two types of events:

Trigger Events These are a distinct on/off type of event, analogous to pushing and releasing a button. Just because they represent simple on or off state does not mean that the element's behavior isn't doing something complex while the trigger state is "on". For example think of a light that pulses while the state is "on", or a chase that repeats while its state is "on". Trigger events have an intensity value between 0-1, anything over 0 is an "on" trigger, while a trigger with value 0 is an "off" trigger.

Mapped Events These events change the value of some attribute of a element over time and might be connected to knobs, or sliders. For example you might have three knobs hooked up to an effect - one that controls the width of a pattern, one that controls the hue, and one that controls the cycle speed.

1.3 Birdfish takes events and renders them into channel output

Birdfish takes a sequence of events, either pre-recorded or live/interactive, and uses them to trigger and modify elements, which then, based on their attributes and behaviors, will render channel data over time, which is sent out to any connected lighting networks.

Lets work with an example. A basic light element supports an attribute called "attack". The value of this attribute is the number of seconds it takes a light to dim up from black to full intensity when it is triggered on - this is also known as a fade-in. In grid based software, this might be represented as a ramp of increasing height from left to right. In Birdfish, it is simply a property of the element. A input's trigger event will cause the element to start outputting first low values, and gradually brighter values over time until full intensity is reached until an off trigger event is received.

Now a sequence of several minutes might have dozens of these trigger events. If at a later time, you want to change the attack of a given light, you simply go to where that element is defined, and give it a different attack value. No changes need to be made to the sequence because the timing of the triggers is all still correct. When the sequence plays again, Birdfish will once again trigger the element, but now the element will render its channel output differently, using the

new attack value. This is just scratching the surface of what this separation between event timing and channel output can yield.

1.4 Groups, Chases, and Effects

LightElement is the base object in an inheritance tree (class tree). RGBLight is one that adds features for RGB, such as having a hue attribute that automatically adjusts the R,G,and B channels to match a color of a certain Hue in HSV colorspace (colorwheel).

In addition to subclasses for certain light types, there are also Groups and Chases. A group is simply a container object containing a set of LightElements. These can be nested so you can have groups like:

north_window is the name of a LightElement that represents a string of red lights around a particular window. You might combine this with other LightElements into the following groups:

- window_group
- red_lights

An all_house group might consist of [window_group, roof_group]

When a group is triggered, the software automatically triggers all its elements. If a group contains other groups, this trigger is propogated down to each light element that generates channel output.

A chase is a group that when triggered, will trigger each element in series over time. A chase has a number of attributes that control how those lights chase. What is recorded in the sequence is just the timing of the trigger, the complexity of the chase is all generated in software.

CHAPTER

TWO

INSTALLATION

Currently in this alpha release, the the best supported configuration OS X, though those experienced with Linux should also be able to get going by tweaking these directions. Both Linux and Win32 platforms will be fully supported in the future.

If you are at all experienced with Python development, you should use the virtualenv model of segregating your Python environments. If you're only plans are to install and experiment with this software, it is reasonable to just install onto your base system.

On OS X, ensure that your environment has a couple tools in place:

- The latest xcode from Apple (free in app store) or Developer tools CLI install (requires Apple developer ID)
- Homebrew
- git (for getting latest development version) **brew install git**
- portmidi for midi control **brew install portmidi**

These are my recommended steps for getting your Python environment setup - you will need your password to install some tools globally (type them into terminal):

```
cd /tmp
curl -O http://python-distribute.org/distribute_setup.py
sudo python distribute_setup.py
curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
sudo python get-pip.py
sudo pip install -U virtualenv
sudo pip install -U virtualenvwrapper
```

To finish the installation of the virtualenv wrapper tool, you will need to modify your shell environment. Open you .bash_profile file:

```
touch ~/.bash_profile
open -a textedit ~/.bash_profile
```

add these three lines to your .bashrc file in the root of your home folder.:

```
export PATH=/usr/local/bin:/usr/local/sbin:$PATH
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/Devel
source /usr/local/bin/virtualenvwrapper.sh
```

Now you need to make a Python virtualenvironment to execute the BirdFish code:

```
mkvirtualenv birdfish
pip install -e git+https://github.com/ptone/BirdFish.git#egg=birdfish-dev
```

```
pip install -e git+https://github.com/ptone/protomidi.git#egg=protomidi-dev
pip install -e git+https://github.com/ptone/Lumos.git#egg=lumos-dev
pip install -e git+https://github.com/ptone/pyosc.git#egg=pyOSC-dev
```

BirdFish is the main control software package, protomidi is the Python bindings to the PortMidi library which makes your system's Midi infrastructure available, and Lumos is a library for sending E1.31. pyOSC is a Python implementation of the OSC command protocol.

Download a small virtual midi keyboard called MidiKeys

If you want to use DMX USB hardware and not just E1.31 - you will need to install OLA, more information can be found on its home page. You need to make sure that the Python bindings are part of the binary - last I checked, this was not the case, so you need to build from source.

Anytime you want to execute you need to "activate" the BirdFish Python environment in a terminal session. Just type:

```
workon birdfish
```

You should then see (birdfish) in your shell prompt.

To open the examples (note you need Midikeys open):

```
cdvirtualenv
cd birdfish/examples
python <example name>
```

To update to the latest development version of BirdFish:

```
cdvirtualenv
cd birdfish
git pull
```

CHAPTER

THREE

INPUT AND OUTPUT

Birdfish provides a toolset to convert time based events, into light signalling output. This can be considered a core rendering engine that only knows about events in, and channel data out. Nothing about Birdfish's core assumes anything about what is generating the events, or where the channel data is being sent to. This is handled by a choice of one or more input and output modules. At this point, the number of options are relatively limited, but here is a list of what is planned:

3.1 Inputs

- MIDI signals
- OSC (Open Sound Control)
- Event File (A Birdfish specific format for recording events)
- Web (using realtime websockets) **
- MIDI file *
- PC Keyboard (currently handled by virtual midi keyboards)*
- Kinect **

3.2 Outputs

- E1.31 (pure Python via Lumos Library)
- OLA DMX (E1.31, USB)
- PixelNet *
- Event File **
- One or more sequence software formats *
- Renard, LOR (low priority)*

_ * not yet implemented _ ** partially implemented

CHAPTER

FOUR

ENVELOPES

Envelopes are tool used in Birdfish when designing effects. They allow you to describe how something changes over time or distance in a flexible and powerful way.

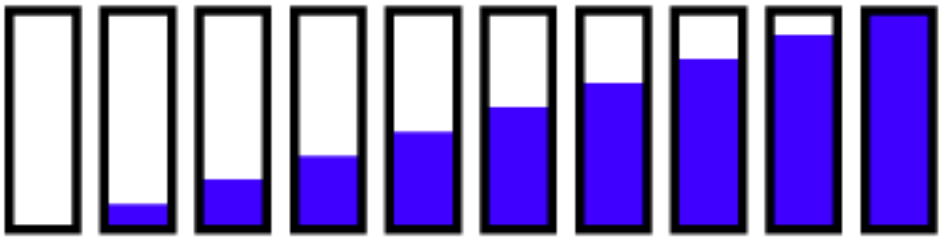
4.1 Introduction

Envelopes are used in many ways, and can be confusing in their abstraction, so lets start with a super simple example: a fade in.

We can describe a fade in as starting at zero, going to one, and taking one second.

Note: Most values in Birdfish are floating point numbers between 0-1 instead of 0-255 as in DMX, channel values are converted to 0-255 for network output.

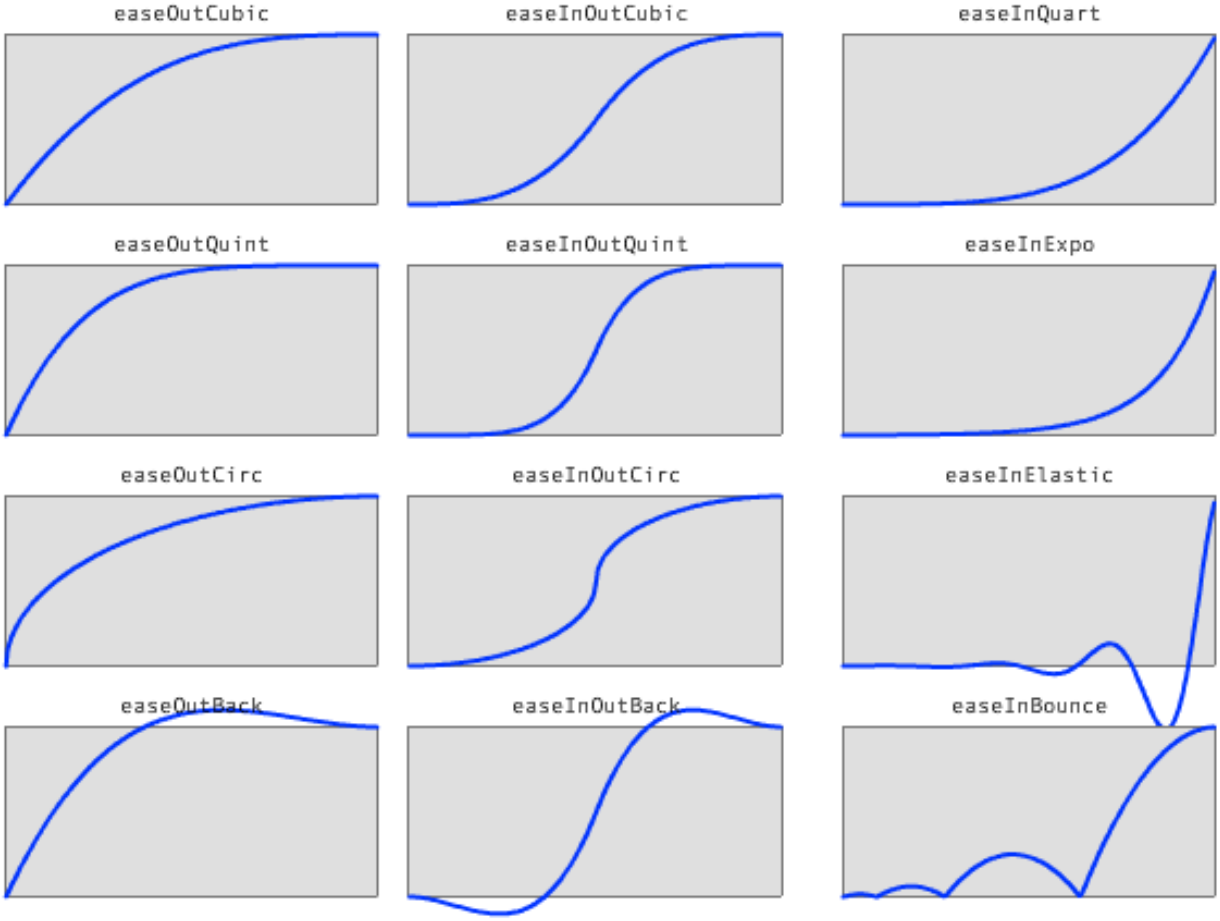
In a conventional grid based approach - where a second might be divided into ten slices, this might look something like this:



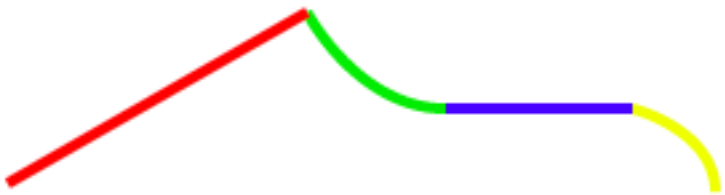
Whereas if time is divided into an infinite number of slices - this fade in would simply be represented as a line, a simple linear slope.

This line can be encapsulated as an envelope - and can be applied to attributes of elements, such as intensity, or hue. It can also be applied to the position of a chase, being a start and end position over time.

A linear change may be the most familiar, but far more dynamic results can be achieved with curves that travel from point A to point B in some non-linear way. These types of curves are often used in animations, and are sometimes referred to as easing curves or tweens and can be illustrated like this:



Many times, the behavior you are looking for can't be described by just one function. In these cases, curves are joined together in an evenlope, where each curve is a segment. For example, here is a more complex envelope consisting of four segments.



4.2 ADSR

The concept of envelopes is used frequently in music synthesizers. One classic envelope is the Attack-Decay-Sustain-Release, or ADSR, envelope. In synthesizers this describes how the volume of the sound changes over time, but we can apply this concept to light intensity (the color reference the segmented envelope figure above).

- Attack (red) - is the fade in.
- Decay (green) - is a fade to the held level.
- Sustain (blue) - is the level at which the intensity is held as long as the trigger is on.
- Release (yellow) is the fade out, which begins when the off trigger happens

This 4 part envelope is built into light elements directly, and can be modified simply be changing their attack, decay, sustain, or release attributes.

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*