


# Weather Synth

## Project Journal

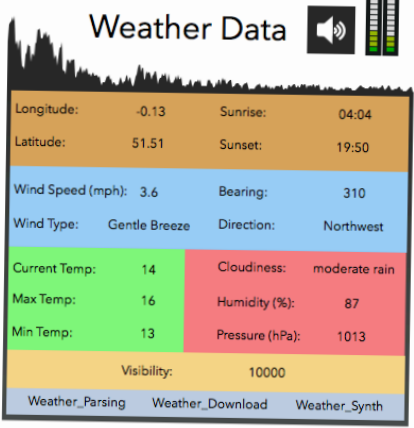


**Selected City:** London

Double click a city or type in the box above

Last Pinged: 17:45  Autoping (30m)  
Last Updated: 15:50

### Weather Data

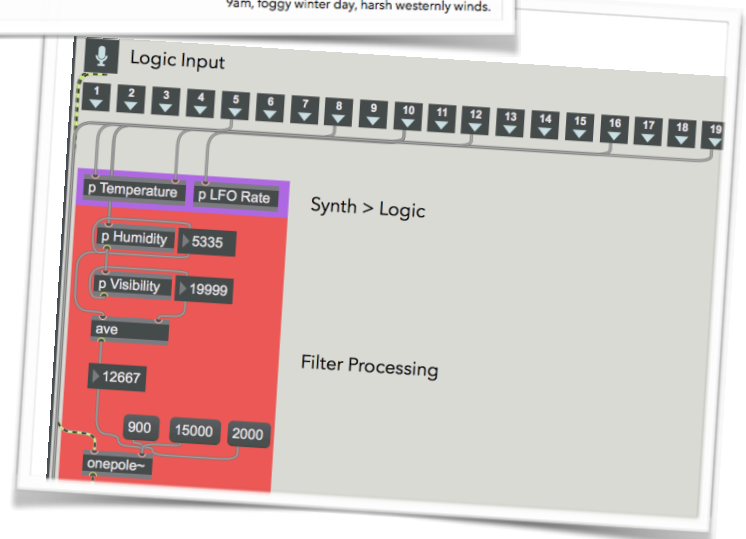
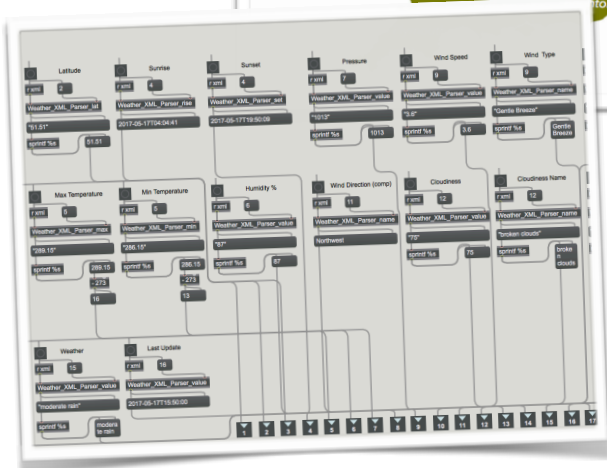


|                   |               |                 |               |
|-------------------|---------------|-----------------|---------------|
| Longitude:        | -0.13         | Sunrise:        | 04:04         |
| Latitude:         | 51.51         | Sunset:         | 19:50         |
| Wind Speed (mph): | 3.6           | Bearing:        | 310           |
| Wind Type:        | Gentle Breeze | Direction:      | Northwest     |
| Current Temp:     | 14            | Cloudiness:     | moderate rain |
| Max Temp:         | 16            | Humidity (%):   | 87            |
| Min Temp:         | 13            | Pressure (hPa): | 1013          |
| Visibility:       |               | 10000           |               |

Weather\_Parsing Weather\_Download Weather\_Synth

### Presets

Noon, clear hot day, gentle northerly breeze.  
Noon, crisp autumn day, cool easterly breeze.  
9am, foggy winter day, harsh westernly winds.



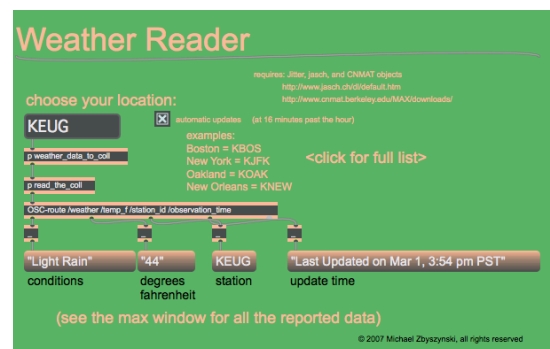
# Weather Synth

## Foreword

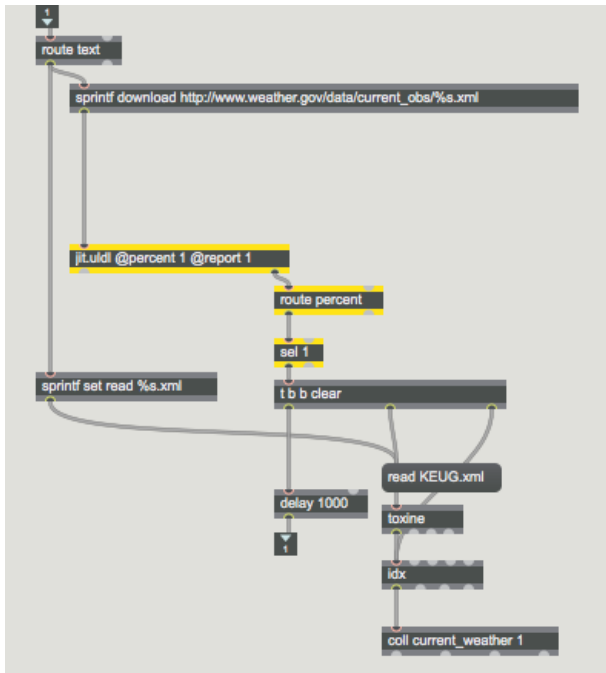
Ever since the Interactive Music Systems module, I have had a deep interest using VPLs (visual programming languages) to create systems that create music out of interesting and unusual data, data that you wouldn't normally attribute music to. After being shown a CNMAT Max patch created by Michael Zbyszynski that interprets XML weather data, I decided to focus on the weather as it is something that constantly changes over time and is never exactly the same in two places. *My patch downloads, parses and then uses Logic Pro X (eventually the patch will be self sufficient audio-wise) to synthesise weather data from an online API. The different weather conditions affect different parameters on the synthesiser. Below is my development process.*

## Development : Downloading

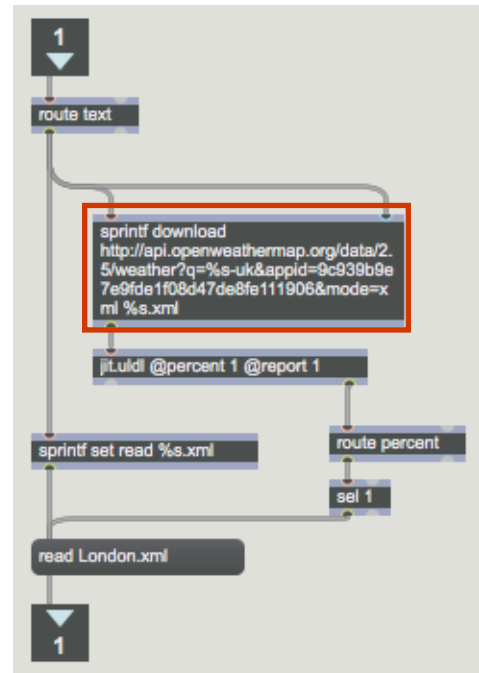
From the start of this project, I wanted to create a live installation device that played music continuously whenever it was turned on. I started my patch by using the Michael's Max patch that was created to interpret US weather station data. Originally, I had a lot of trouble decoding the objects that were being used as I had never used them. I got around this by going through the patch cord by cord and learning what each section of the patch did. I singled out the part that downloaded the XML data and edited it to download from OpenWeatherMap, the weather API service that I signed up to for this project. One inherent downside to my patch is that the weather is delayed by 2 hours due to me not paying for their premium service (\$180-\$2000/month). I don't view this as a downside to the listening experience, it is just something that the end-user has to consider.



Getting the patch to download the weather XML files from the OpenWeatherMap API service was the easiest part of the project, the difference between mine and Michael's subpatch are shown below:



Michael's download subpatch



Weather Synth's download subpatch

In my iteration, the user types in the city they want to download the weather for. It comes through the subpatch inlet and is substituted in the "%s" of the API call in the red highlighted object. Once the patch has downloaded the XML file containing the weather, and names the XML file "%s", where "%s" is the city name. For example, if the user types in London, the patch downloads the weather for London, and calls the file "London.xml". The patch then reads the file "%s.xml", and its contents are passed through the outlet of the subpatch.

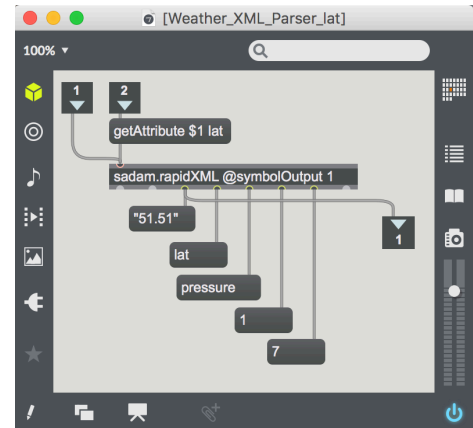
After having successfully made a patch that gets the XML file in the root folder of the project, I needed to get to the point where the patch could parse the information in the XML file. Below is a screenshot of the raw XML file.

```
London.xml
<?xml version="1.0" encoding="UTF-8"?>
<current><city id="2643743" name="London"><coord lon="-0.13" lat="51.51"></coord><country>GB</country><sun
rise="2017-04-08T05:18:46" set="2017-04-08T18:46:37"></sun></city><temperature value="290.14" min="288.15"
max="292.15" unit="kelvin"></temperature><humidity value="45" unit="%"></humidity><pressure value="1024"
unit="hPa"></pressure><wind><speed value="3.1" name="Light breeze"></speed><gusts></gusts><direction
value="230" code="SW" name="Southwest"></direction></wind><clouds value="0" name="clear sky"></clouds><
visibility value="10000"></visibility><precipitation mode="no"></precipitation><weather number="721" value="
haze" icon="50d"></weather><lastupdate value="2017-04-08T11:50:00"></lastupdate></current>
```

Raw XML file

## Development: Parsing

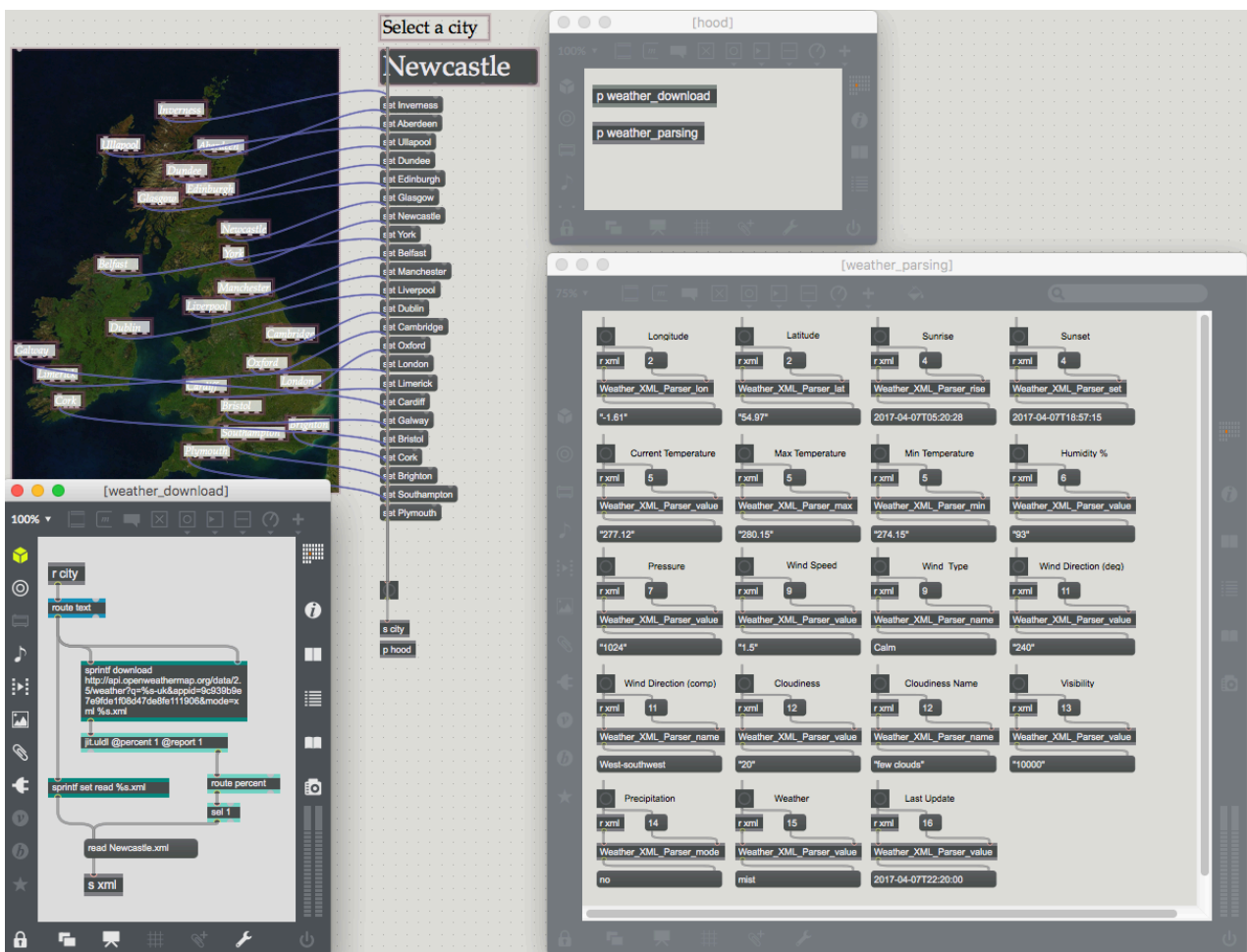
The XML was completely un-formatted, with a completely different tree hierarchy from the the US weather XML files used in the original weather reading patch. This was the hardest part of the project, as I'd never worked with XML at this level before. After about 2 weeks of trying, I finally found a way of parsing the XML into Max message boxes. On the right is one of the XML parsing abstractions. The weirdness of the unformatted XML files made it hard to make this part of the process look as neat as the others. All of the parsing happens in the [Weather\_Parsing] abstraction. For the actual parsing, I use an external made by \$Adam, a user on the Cycling '74 forums, called `sadam.rapidXML`. At this point in the project, I took a break from the technical side of patching to focus on the GUI development side of the patch.



Parsing sub-subpatch

## Development: GUI I

Below is the patch at this stage in the development, it was at this stage where I started focusing on visuals, not only for the end-user but also not to confuse myself when working on the project. I made use of the presentation mode in Max to toggle between seeing behind the scenes and seeing what the user would see.



Patch before I started working on GUI.



Choose a city or type in the box below  
Selected City: Newcastle

Last Pinged: 13:14  
Last Updated: 11:20

## Weather Data

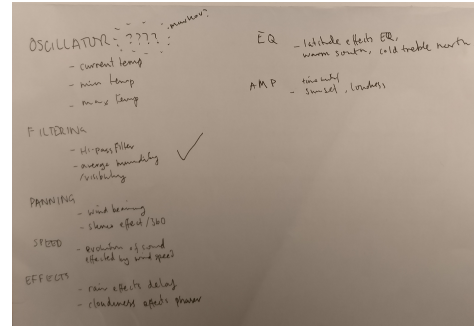
|                   |               |                |                 |
|-------------------|---------------|----------------|-----------------|
| Longitude:        | -0.14         | Latitude:      | 50.83           |
| Sunrise:          | 04:50         | Sunset:        | 19:08           |
| Wind Speed (mph): | 3.6           | Bearing:       | 240             |
| Wind Type:        | Gentle Breeze | Direction:     | North-northeast |
| Current Temp:     | 11            | Precipitation: | no              |
| Max Temp:         | 13            |                |                 |
| Min Temp:         | 10            | Cloudiness:    | clear sky       |
|                   |               | Weather Type:  | 0               |
| Humidity (%):     | 81            | Weather Desc.: | clear sky       |
| Pressure (hPa):   | 1020          | Visibility:    | 10000           |

Version 4 GUI

Above is the presentation patch after I worked on the GUI for about 2 days, I brought the weather data in to the main patch, as I thought the user would like to see as well as hear the weather. The user can click on some preset cities as well as type any town or city in the UK into the selected city field. I added a last pinged and last updated timer, the last updated timer comes from the XML, and is the latest weather data. Originally it was in the format of "YYYY-MM-DD T HH:MM:SS" but using the jasch external object [strcut], I isolated the hours and minutes as they are what are most important for the user. The last pinged time is the last time a request was sent to the API server by the user.

## Development: Synthesiser

After working on the GUI for about a week, I started work on the synthesiser. This was the most creatively challenging part of the project. I had to come up with musical features that lent themselves to weather conditions. This is the list I came up with originally:



Paper notes on parameters

**Oscillator:** Current Temperature

**Filtering:** Low-pass filter controlled by average humidity/visibility

**Panning:** Stereo panning effect controlled by wind bearing

**Speed:** Speed of oscillator arpeggio/scale set by wind speed

**Effects:** Rain affects delay? Cloudiness affects phaser?

**EQ:** Latitude affects EQ? Warm south, colder north?

**Amplification:** Time of day affects volume level of patch

I decided this was a lot of effects to add into the patch, so I settled with the following.

**Oscillator:** Current Temperature chooses scale, tempo is set by the wind speed

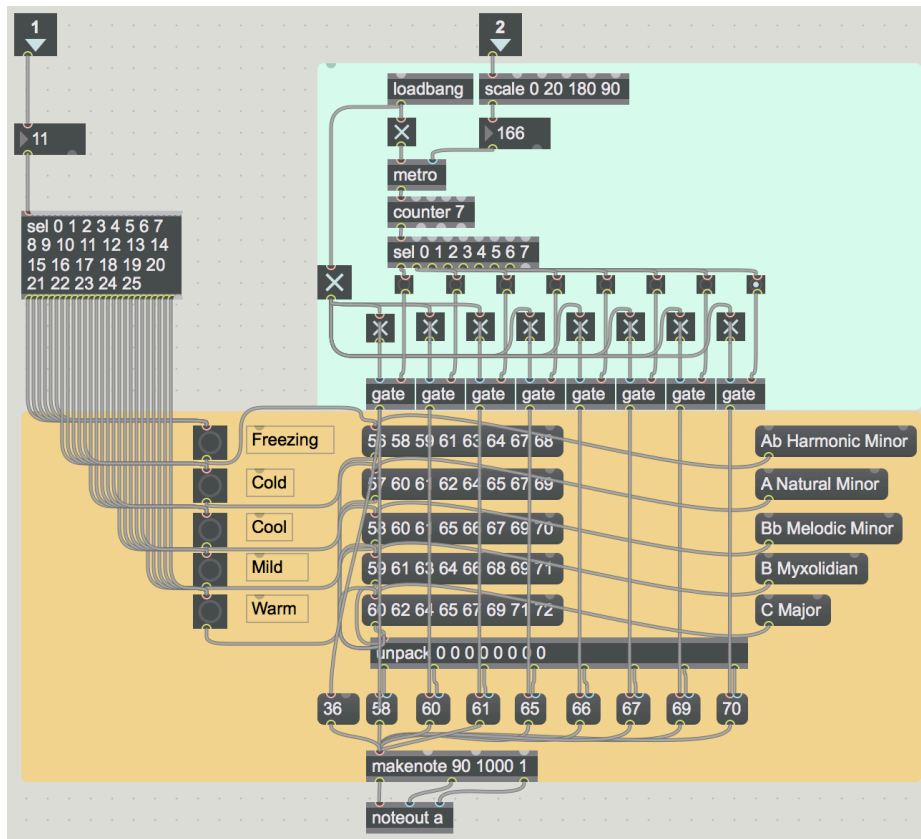
**Effects:** Wind speed also effects the speed of the LFO of the phasor effect in Logic

**Filtering:** Low-pass filter controlled by average humidity/visibility

**Amplification:** Time of day affects volume level of patch

**Panning:** Stereo panning effect controlled by direction of the wind (bearing)

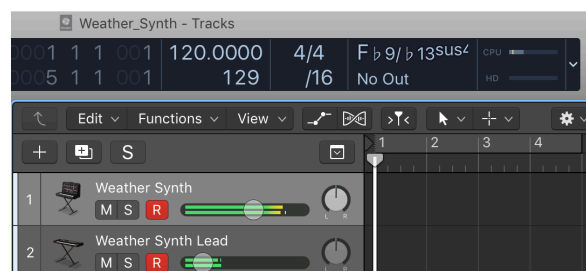
## Oscillator



[p Temperature] Subpatch

Above is the oscillator subpatch, [p Temperature]. Inlet 1 is current temperature. The temperature goes into the select object, where 0-5°C, 5-10°C, 10-15°C, 15-20°C, and 20-25°C each trigger different MIDI scales/modes in the yellow section. The sequencer in the blue part of the patch is controlled by Inlet 2, which sets the clock. The scale object scales 0-20 miles per hour wind speed between 180 and 90 milliseconds.

The output of the patch goes to the MIDI port "from Max 1" which is then picked up in Logic Pro X. Which plays the notes through a synthesiser I created in the stock Alchemy plugin. It then comes back into the Max patch via [ezadc~] and carries on through the signal path.

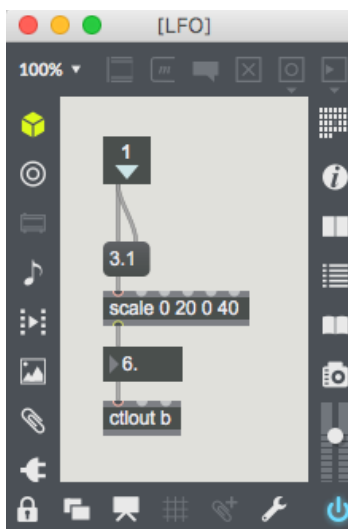


Logic Pro X Project

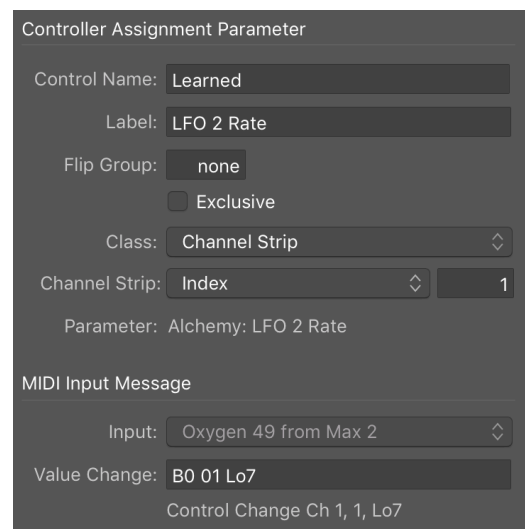


## Effects

In the [p LFO] subpatch, the windspeed is sent to Logic via the "from Max 2" port. The windspeed (0-20mph) is scaled to 0-40 in MIDI. In Logic, I have set up a default controller assignment which affects the LFO of the phasor effect on the synthesiser. This is to give the sound a more hectic feeling if it is very windy, and a more slow, cyclic rhythm if it is a gentle breeze.

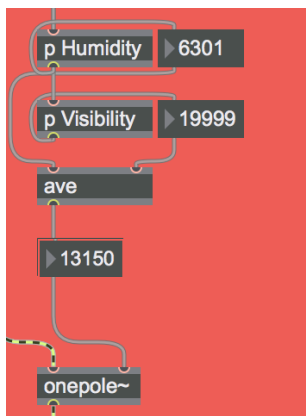


[p LFO] Subpatch



Logic Pro X Controller Assignment

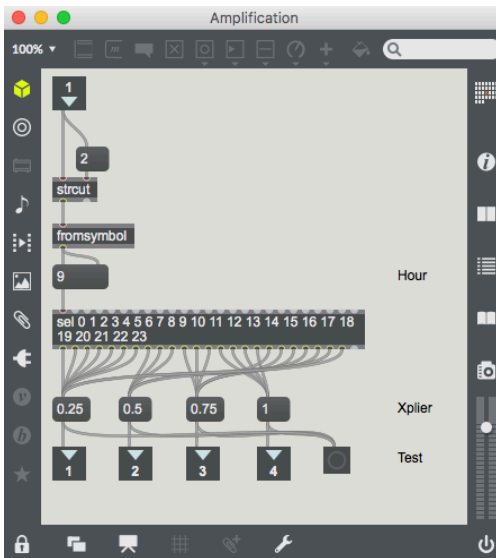
## Filtering



Filter section of [Weather\_Synth]

The filter was the first idea I had for the musical side of the patch. When I looked over the weather conditions that I would be dealing with (e.g. temperature, cloud cover, sunset etc), I immediately saw a connection between humidity and the idea of a low pass filter - the concept being that if it was more humid, and there was less visibility, the patch would sound dampened and the frequency range would be reduced. I carried this out by scaling the humidity and visibility to 0 - 20,000Hz and taking the average and putting the result into a low pass filter

## Amplification



[p Amplification] in [Weather\_Synth]

One of the last ideas I had with the patch was to implement a dynamic range of sorts. I did this by separating the HH component of the 'lastupdated' XML element (the hour that the weather data was taken from) with [strcut]. I then passed it through a select object, and made time brackets:

10pm - 7am is 0.25x volume level

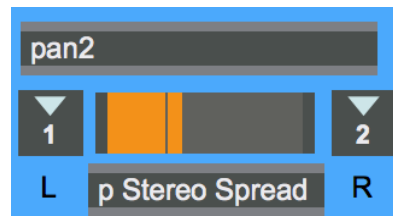
7am - 10am and 8pm-10pm is 0.5x volume level

10am - 12pm and 5pm - 8pm is 0.75x volume level

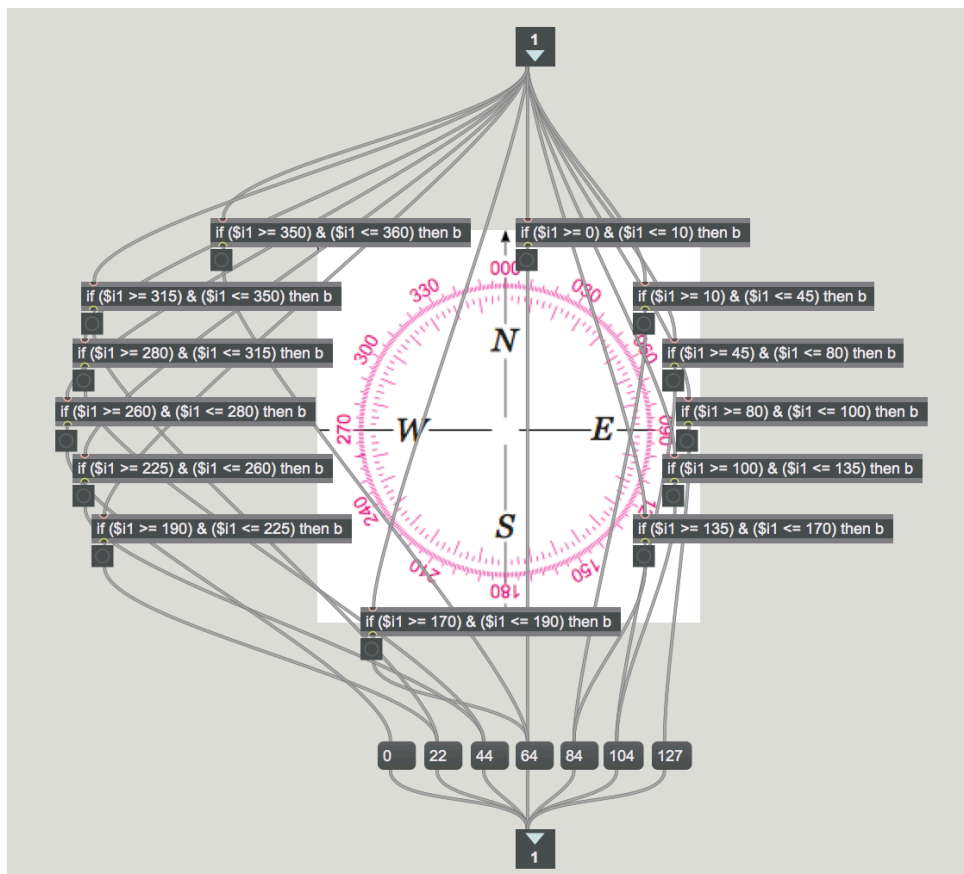
1pm - 5pm is full volume level

## Panning

The last feature I implemented into the patch in the development stage was the stereo panning. The idea was to create a sense of wind direction if the user was wearing headphones or using stereo speakers. The [p Stereo Spread] subpatch is responsible in doing this. The subpatch contains 13 expressions, which sorts the current wind bearing (coming through inlet 1) into whichever expression is correct. If the wind blows from the east at 90°, the expression `[if ($i1 >= 80) & (%i1 <= 100) then b]` will output a bang, which outputs 127 from the patch, this then goes into the right inlet of the [pan2] object tin [Weather\_Synth] which sets the corresponding volume levels for L and R output.



Panning section of [Weather\_Synth]



Behind the scenes in [p Stereo Spread]

## Development: GUI II

The final part of the project was working on the GUI to a point where I would be comfortable with someone else using the patch. Below is a screenshot of the GUI in the final version (5). Also shown are some features that I added to make the user experience more engaging.

The screenshot shows a GUI for a weather patch. On the left is a map of the United Kingdom with various cities labeled. On the right is a weather data panel for Brighton. The panel includes a 'Selected City' field, a 'Weather Data' section with a spectroscopic chart, and a 'Presets' section. Callouts point to specific features: 'Clearer map' points to the map; 'Clearer font patch wide' points to the city name; 'Autoping Function' points to the 'Autoping (30m)' checkbox; 'Synth On/Off and Levels' points to the volume icon and spectroscopic chart; 'Sleek looking spectroscope' points to the spectroscopic chart; 'Starkly different presets for demonstration' points to the 'Presets' section; and 'Colour grouped weather conditions, removed unnecessary conditions' points to the color-coded data table.

Clearer map

Clearer font patch wide

Autoping Function

Synth On/Off and Levels

Selected City: Brighton

Double click a city or type in the box above

Last Pinged: 12:51  Autoping (30m)  
Last Updated: 14:20

Weather Data

|                   |              |                 |                 |
|-------------------|--------------|-----------------|-----------------|
| Longitude:        | -0.14        | Sunrise:        | 04:06           |
| Latitude:         | 50.83        | Sunset:         | 19:48           |
| Wind Speed (mph): | 1.2          | Bearing:        | 0               |
| Wind Type:        | Light breeze | Direction:      | North-northeast |
| Current Temp:     | 24           | Cloudiness:     | few clouds      |
| Max Temp:         | 15           | Humidity (%):   | 0               |
| Min Temp:         | 14           | Pressure (hPa): | 1012            |
| Visibility:       |              | 10000           |                 |

Weather\_Parsing Weather\_Download Weather\_Synth

Presets

Noon, clear hot day, gentle northerly breeze.  
Noon, crisp autumn day, cool easternly breeze.  
9am, foggy winter day, harsh westerly winds.

Sleek looking spectroscope

Starkly different presets for demonstration

Colour grouped weather conditions, removed unnecessary conditions

## End note

The Weather Synth does have some errors. After some time open, the console will report the error "expected <". I've been aware of this for some weeks now, but have no idea why it occurs. I have narrowed it down to the parsing process but do not know how to get rid of it. After testing the error, it does not affect the sound in any way. During early development, the patch was getting errors pretty frequently, and at the time I didn't know why. It turned out it was because I wasn't allowed to ping the API more than 60 times an hour on my free subscription with OpenWeatherMap, this was solved by restarting the app, but it was nevertheless very annoying when I didn't know the source of the error. Currently, this error still occurs, but if the user does not ping the server more than 60 times in an hour then the patch behaves.

The [Weather\_Synth] subpatch contains a delay and reverb function that I plan on developing in the future. The reverb happens to distort and glitch the sound at the moment, which I quite enjoy the sound of, so I have left it in. The delay is turned off by default currently.

In the future, the patch will be completely self sufficient, with an in-patch synthesiser instead of relying on Logic Pro X. The audio routing is shown below. Eventually, the patch will use Miraweb to be browser based. Also, the patches directories will be more neat, unfortunately I did not think of this at the start of the project, and every attempt I've done to tidy it up has resulted in the patch breaking.

N.B. You will need the Jasch and \$adam external libraries to run this patch.

### **Max**

Out: Audio Interface -> Speakers

In: Soundflower (Or similar audio routing software)

### **Logic**

Out: Soundflower (Or similar audio routing software)

In: N/A (Doesn't affect the patch)