

## Abstract

The project deals mostly with the sonification of external, cloud based data sources, and contributing to aforementioned cloud-based data sources by using current web technologies including but not limited to Javascript, PHP, HTML media objects and RESTful API's with specific interest in regards to furthering the goals of the netChimes open source project by both lowering the barrier to entry and providing a monitoring system which can be used in a variety of ways and for varying purposes, with the base function of triggering sound under specific circumstances. At the time of publishing, the project is hosted at <http://dev.welikepie.com/MakeTheCosm>.

## Introduction

The project I chose is an extension of the netChimes project (<http://www.netchimes.org>) by Jason Geistweidt and Brock Craft, who is also my supervisor. This was a result of being interested in and having already done information sonification at hackdays before. The concept of an Internet of Things is a concept which has long been of interest, and I see this particular implementation very much as an extension of the concept. This project is both relevant to work and my degree program, incorporating important elements of both.

At the moment, the barrier to entry to sending and receiving information on the internet is relatively high; the programmatic structures and API's are usually only used by people who have the technical skillset to do so. While it is possible to learn how to send information to the cloud, the skillset required to do so oftentimes prevents people doing so. The first half of my project, namely making my own netChime sensor, accomplishes just this by using comparatively ubiquitous and easy to use technologies of a Makey Makey board coupled with a web interface which allows the user to configure what information is sent to the cloud.

The second half of my project is a project with its' roots firmly in the sonification of data for a purely acoustic effect. Along with extending and contributing to the netChimes project, it will also lower the barrier of entry to usefully using sound that has been produced by the cloud, as it will enable the user to search by keyword (and location based on the result presentation). This part of the project is intended to present a web frontend for the user to search a cloud data server for information according to keywords. Once the search results are returned, the user can select individual feeds along with individual data streams being returned. These streams can then be associated with a particular sound and a trigger specified by the end user. These streams will be sonified by sounds that the user is able to pick and choose at their own will.

In my part-time job, I am a web developer and occasionally work with Physical Computing projects. As a result of this, I am up to date on recent API's and tools to use to complete most web-based projects. This usually entails creating a front-end, user-directed side of the software and also developing the back end "nitty gritty" of the code to ensure an optimal and functional user experience. Additionally, thanks to the hardware component of my job, I have also learnt to create physical projects and work with some comparatively uncommon tools to reach the desired end. To this end, a project requiring server side scripting along with front-end user interfaces is well suited to my skillset.

In university, the courses that have taught me the most in relation to this project have to be Creative Projects and Perception and Multimedia Computing, along with Networking and Network Programming and Interaction Design. Procedures and Applications of Programming has taught me the basic principles of programming, language independent. Creative Projects has taught me proper project management on a large scale instead of the very "as-soon-as-possible" approach eminent in production level programming. Additionally, it also taught me where to go to look for most answers to problems and how to present a project to a wider audience, taking their needs into consideration to

produce a polished project. Perception and Multimedia Computing is relevant as it forces the developer to consider to think outside of the box in terms of accessibility and how the project is perceived on a literal level. This translates into a heightened consideration of what the end users may or may not be able to perceive or make sense of because of various different potential problems. Interaction Design has taught me about the needs of end users, and that the most efficient ways to make your system user friendly is to user test. This is especially significant in my project, as most of the interaction and intent is for the user, necessitating a good design. Networking and Network Programming was able to teach me the fundamentals of how software reacts and interacts with a network, which has helped me to write the back end and interact with the RESTful API's involved in the project.

### **Definition of terms**

#### *Frontend (Also Clientside)*

The frontend of a website is taken to mean the parts of the page that the user can reasonably interact with. In addition, it is also used to refer to code which executes on the local part of the webpage which has been accessed by following a link or typing one in to the browser.

#### *Backend (Also Serverside)*

The backend of the website is taken to mean the parts contributing to the webpage which the user does not usually see or have any interaction with, and as such are usually hosted on the server that the website is running on. Included in this would be files the user can download as well as PHP scripts.

#### *DOM*

The DOM is an acronym which is short for 'Document Object Model', which is a reference to the standards implemented by HTML, XML and XHTML to define how the syntax of the document is structured. A related concept is the DOM tree which refers to how the elements of the HTML, XML or XHTML relate to each other.

#### *Sonification*

Sonification is a blanket term, which in this specific case unless specifically stated otherwise is taken to mean the conversion of a set of data into sound by the methodologies of using data to fulfil conditions in which separate sounds are triggered.

#### *Data Object*

A data object in the scope of this project refers to anything which contains data relevant to the program. As such, JSON objects, Arrays, and even some strings could be seen to be data objects.

### **Background Literature<sup>1</sup>**

#### *Auditory Displays*

The basic of any sonification is to decide what kind of auditory display is going to be provided to the end user, and can be divided into four types : Audification, Speech Synthesis, Sonification and Speech Synthesis.

Sonification, in general, refers to any non-speech audio which can be used to convey information, a popular example of which being the chirp.io project.<sup>2</sup> The reason that sonification can be such a

---

<sup>1</sup> "Welcome | International Community for Auditory Display." 2007. 17 Mar. 2013

<<http://www.icad.org/home>>

<sup>2</sup> "chirp.io - Let's teach the machines to sing." 2007. 17 Mar. 2013 <<http://chirp.io/>>

popular method lies therein that humanity possesses a wide faculty of perceptual abilities, not least of which is hearing. As humans are able to perceive volume, pitch, timbre and speed of a sound, it makes sound an ideal addition to any standard visualisations for both entertainment and analytical purposes.

In Audification, a subset of Sonification, the principle is to directly correlate information received by either live-streaming or experimental results to a waveform, which can then be analysed acoustically and is incredibly useful for time-series data as signal processing which can then lead to further analysis being carried out on the data thanks to signal processing techniques.

Earcons are a special kind of auditory display, which are both for the visually impaired and visually abled alike. A pun on the phrase "Icon", this auditory display is a short audio cue to hint that certain things have happened. As this audio cue is often repeated, it can be ingrained in the memory of the user to the point of instant recognition, and is a feature employed heavily in most modern Operating Systems to indicate system state to the user.

Speech Synthesis is another kind of aid which can be used to aid those hard of sight, but also those with full visual acuity. In its simplest form, it is the conversion of data and information to a vocalised sound simulated by a computer which is understandable as speech to a person. To this end, text can be read to a user to create an additional interface to a device.

### *Sonifying Go<sup>3</sup>*

An example of making the physical world sound like something new and different to its original sounds, the Sonifying of the game Go by Peter Tigges represents an interesting step. Anybody with enough time and willingness can make a procedural task sound of something as long as enough data points are contributed.

In the case of this game, the sounds were determined procedurally by certain discrete criteria. The pitch and pan of the signal are determined by the distance and direction from the center of the board that the stone is placed, with the amount of reverb and amplitude of the signal being determined by the amount of liberties the given stone has. Each colour (read : each player) is allocated a different synth to differentiate between the two players and turns played. The rules of go can be found on Wikipedia<sup>4</sup>.

Instead of being an interactive live stream or live feed, this project is generated by inputting all of the pertinent information about the game into the source code manually, which then generates and outputs music. This leads the project to being very static in nature: the only interactive part of creating the music is the playing of the game itself, which even then needs to be tediously recorded.

### *Sonifying the Higgs Boson<sup>5</sup>*

The sonification of the Higgs Boson by the LHC Open Symphony group is one of the most recent productions of a long line of efforts to sonify data from the Large Hadron Collider and its physics experiments carried out by CERN.

---

<sup>3</sup> "Sonification of Go by Peter Tigges on SoundCloud - Hear the world's ..." 2012. 17 Mar. 2013  
<<http://soundcloud.com/peter-tigges/sonification-of-go>>

<sup>4</sup> "Rules of Go - Wikipedia, the free encyclopedia." 2005. 17 Mar. 2013  
<[http://en.wikipedia.org/wiki/Rules\\_of\\_Go](http://en.wikipedia.org/wiki/Rules_of_Go)>

<sup>5</sup> "The first Higgs boson data sonification! « LHC Open Symphony." 2012. 17 Mar. 2013  
<<http://lhcopensymphony.wordpress.com/the-first-higgs-boson-data-sonification/>>

In this piece, the data from the Higgs Boson experiment is ran through a specific algorithm which associates individual measurements in the data obtained from the experiment to a particular note in the standard western musical notation scale. This transposing follows a principle based in the assumption that patterns in nature often follow those evident in mathematics, and therefore in music equally, as music in western notation is based on mathematical principles.

In later incarnations of the melody, the track has been edited to have different synthesised instruments playing different sections of the derived melody, including a backing section, purely for melodious reasons.

In this sonification, the goal is clear: the intent is to impress the audience and, on a more important level, make science more accessible to the masses. Instead of being presented with a graph and arduously explained to where the Higgs Boson occurs and why it does so, the listener can hear in the melody where the Higgs Boson occurs instead, and learn through hearing.

### *Sonification of Scientific Data<sup>6</sup>*

The process of sonification of scientific data for the purpose of analysis is the main topic of this corpus. The methods of sonification analysed are from a wide range of disciplines including physics, sociology, Speech Communication and Neurology just to name a few.

The main goal of the over-reaching project was to develop a common sonification suite which would enable scientists and sociologists alike to analyze their data in real time with the aid of audio conversions to better visualise and group the information provided.

As the scientific worth and the results from data sonification analysis are not yet well understood, the corpus also urges that further research into the field is a necessity. Additionally, data sonification is not just limited to being useful in the scientific communities, as its' usefulness in broad analysis in other field such as neurology and sociology is often underrated.

### *Real-time Sonification of Physiological Data in an artistic performance context<sup>7</sup>*

For this particular sonification project, the medium of choice was decided to be a real-time artistic installation involving the human body as a proof of concept run, with emphasis being on as much of an aesthetically pleasing installation as pleasant sonically. The data sampling was carried out using three biosensors to gather the heartbeat, breathing and Thoracic volume expansion during breathing of each participant in the artistic installation.

The data points from this artistic installation were then taken separately and ran through differing conversion processes to translate the raw data to music. This, while also having analytical potential for each stream of data, chiefly was intended to produce sound which the listeners could potentially figure out the original source. Despite the interest in harmonious qualities to the sound, however, there was also heavy emphasis upon keeping the auditory phenomena heavily correlated with the data that was being received and processed from the performers.

To this end, the sound played by the installation has an unintentionally analytical quality; by being closely linked to the data, the sound can be re-analysed, and potentially produce a valid conclusion

---

<sup>6</sup> "ScienceByEar - Alberto de Campo." 2012. 17 Mar. 2013  
<[http://albertodecampo.net/uploads/ScienceByEar\\_diss\\_deCampo.pdf](http://albertodecampo.net/uploads/ScienceByEar_diss_deCampo.pdf)>

<sup>7</sup> Kessous, Loic et al. "Real-time sonification of physiological data in an artistic performance context." *Proceedings of the 14th international conference on auditory display* 2008.

after passing individual signals through discrete signal processing.

#### *Monitoring Real-Time data streams : A sonification approach<sup>8</sup>*

Here, the uses of human perception in sampling and analysing data streams are researched, along with the considerations required to adequately design audio interfaces for human use.

Part of the considerations of the study were the design of auditory displays, with specific interest in understanding the different kinds of auditory perception modes eminent in human perception. The modes boil down to the “Alert”, “Single Track”, “Relational” and “Global” modes. These modes differentiate in both specificity and quantity of data being usefully samples by the listener, with “Alert” and “Single Track” being the most specific; focusing for one particular sound. “Global” mode, on the other hand, is very much an indicator of general state of a system and hence the mode with the least data resolution necessary.

In experiments carried out, the listener was presented with different kinds of audio interfaces, including a spacial interface. This particular interface took advantage of the ability for humans to listen in multiple directions at once in order to pinpoint the location of the source of noise. The main outcome of this, as a result, was that the interface taking advantage of innate ability to pinpoint sound performed much better in tests, as it provided more than just one point of differentiation for the listener.

#### **Specification**

The first part of the project involves creating my own netChimes node from scratch. This will involve a web interface which can send information to the cloud service using AJAX requests and CURL requests for creating new feeds on the backend, with user input and a monitoring system being a necessity for the frontend. This frontend will be made with HTML and Javascript technologies, using Bootstrap CSS for styling. The frontend will involve inputs and menus regarding the live streaming of information being received by the MakeyMakey, giving the user choice and versatility in regards to the information being streamed.

As for the information retrieval / sound sonification, the interface relies on a slideshow effect of three slides, with each slide containing a different facet of the method required for sonification.

Slide one contains the search and information retrieval slide, which will have a backend of recurring AJAX requests, which then parse the JSON and write to a mapBox map embedded on site in the form of markers. These markers are then clickable, which introduces information to a panel displaying the information and offering the user the ability to select and view the streams updating in real time to select the best data for their purposes.

The second panel of the web interface will contain the datastreams that have been selected to stream the information and be sonified, along with options governing how the audio is being triggered and a graphical visualisation of the most recent data that has been received by the client. In this slide, the user will also have the option to remove the slide from the feed, to start and stop sonification, as well as to change the sound that is associated with the trigger event for each feed.

The third panel of the web-app is purely for selecting potential sounds and playing them back and adding them to a set for the user to use with their sonification. It will consist of a graphical interface

---

<sup>8</sup> Roginska, A. "MONITORING REAL-TIME DATA: A SONIFICATION ... - MIT." 2009.

<<http://www.mit.edu/~kimo/publications/sonification/icad2006.pdf>>

which enables the user to select from a range of sounds and sound sets which have been predetermined and stored on the server, with the potential to listen to each sound in turn to preview them and then add them to a personal list which can be viewed and edited in the same manner on the same page.

Controlling the slides will be a matter of either using the cursor to click on the slide buttons when they appear intelligently to show the user whether they can change slides or not, or use the left and right arrow keys to change slides.

## **Methods**

The first challenge of the project is to create a physical interface which will stream to Cosm (the Cloud data service of choice chosen by netChimes). To this end, I decided to write a web-based client for a Makey Makey to stream information to Cosm. It uses a Javascript front-end which takes information on a keyPressed event, resulting in a binary datastream. This datastream is then sent to cosm via a recurring Ajax POST request to fully take advantage of the rate limiting of Cosm to achieve as high a data resolution as possible.

The front end of the web app is achieved by implementing a basic instance of the Bootstrap CSS styling to give the web app consistent styling without having to manually style, and therefore focus more of my development time on the end product.

As for writing the client strictly for Makey Makey / keyboard input; the logic is to create a method for the end user to simply and easily stream useful data to the cloud with only a basic knowledge of circuits. This simple methodology means that the barrier of entry to streaming information to the cloud is lowered; enabling individuals of varying abilities to participate in the growing culture of circuit bending and hacking.

The other main part of the project will be pulling information from the cloud to sonify. However, in addition to solely sonifying the information returned by the cloud, the user will be able to select individual feeds based on a search term from a map view which covers the globe, and select individual datastreams to sonify based on which streams seem most interesting / have changed the most. Additionally, the sound produced will depend on different options such as when to trigger the sonification and which sounds are produced, which will be able to be selected and assigned in a separate screen inherent in the application to offer increased user customisability. The individual datastreams themselves will, in addition to being selectable, also display a graph of the data that has been retrieved from the server so the user has more information at his/her fingertips.

## **Technologies Chosen**

### *HTML / CSS*

The basis of any modern web project. HTML and CSS enable the end user to dynamically display and style content at a comparatively low effort. Is a long reaching standard and commonplace.

### *PHP*

A server-side language being used, in this case, to carry out some feed requests whereby information is being returned as a header which cannot be accessed in straight AJAX.

### *Javascript*

A web-based language which enables any developer to create code which runs solely on the client side of any web page. In this case, pure javascript is being used to modify pages on the fly.

### *jQuery*

An offshoot/extension to JavaScript making particular calls and edits much easier to carry out. In this project, it is being heavily used to interface between the client-side website and the COSM API to send and retrieve information using AJAX requests.

### *Mapbox API*

One of the simplest mapping solutions out there; it enables the developer, with little knowledge and a lot of examples, to achieve the desired end goal with a variety of predefined and provided marker interfaces. Additionally, onClick of markers is supported and such can be used for interactivity.

### *JSON*

JSON is a data storage format alongside CSV and XML which is heavily used to transmit data on the internet. In this case it is the format of my choosing for receiving information from the cloud, sending it, processing it, displaying it and also keeping a record of sonified datastreams.

### *COSM API*

Cosm are an instance of a cloud-based data warehouse which accepts connections and can accept API requests at 100 times a minute. In this case, chosen because they are what the netChimes project was set up to run in conjunction with.

### *Highcharts API*

A graphing API which will be used to display and update the graphs of the datastreams the user has chosen to sonify comparable to the graph provided on the cosm website.

## **Design**

### *Sonification interface*



The design of the sonification section of the project was mainly meant as a vehicle to try my hand at designing a project for public dissemination. To this end, I decided to try and emulate some popular applications as well as use some common sense logic in the<sup>9</sup> design process to formulate a distinguishable design. As skype is famed for having a logo that evokes a sense of friendliness and openness as well as an ease of use.

To this end, the final version of the sonification interface features rounded edges of otherwise angular corners. This rounding of corners evokes a sense of trust and friendliness from the end user owing to psychologically ingrained behavioural patterns from childhood<sup>10</sup>.

In addition to this, colour connotations also played an important role in my design choices for the styling of the application. The colour choice of blue was the most fitting to evoke and reinforce the feelings of friendliness and openness created by the rounded corners used originally<sup>11</sup>. In addition to this, the progression of the slides making the blue shades lighter also changes the emphasis and feel of the application, transitioning from the connotations of knowledge and seriousness of the map plot containing all the data to the tranquility and softness connoted from a light blue, which is apt for the sound selection window.

---

<sup>9</sup> Skype logo sourced from [https://commons.wikimedia.org/wiki/File:Skype\\_Logo.png](https://commons.wikimedia.org/wiki/File:Skype_Logo.png)

<sup>10</sup> <http://designmodo.com/rounded-corners/>

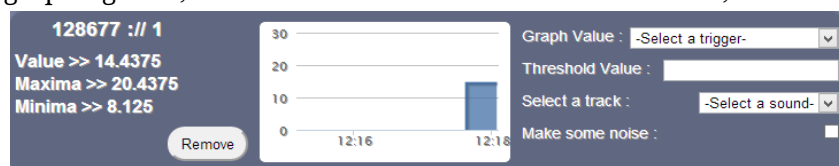
<sup>11</sup> <http://www.color-wheel-pro.com/color-meaning.html>



In addition to this, each panel of the web application sports a different layout, specifically chosen to purposefully not confuse end users as to their whereabouts in the project. This confusion could cause the end user to think that there is content that should be loading but that has failed to. This slightly different design is enough to differentiate between panels in the web application, and yet retain full functionality in regards to finding information.

In the mapping screen, the layout is of two elements, roughly at a 4:1 width ratio, with the larger of the two elements containing the mapping interface with pins being placed on the map, and the smaller of the two serving as an area to load data to temporarily for the user to view and decide whether they wish to use the information they have found. The map section, while comparatively simple, could also be design breaking owing to the colours chosen. To this end, the colour for the home icon was selected as purple to emphasise luxury and power, with the expectation that the user would collate this with a sense of home.

The sonification screen reverses the previous layout, with the thinner strip being on the left and showing the user, at a glance, what information they have selected and offering them the ability to remove that set from the data without scrolling through the other field in the slide, namely the graphing field, to find the data to be removed. In this case, both fields are presented as elements with



descriptions followed by lists which can be scrolled to remove. The graphing section takes the form of a lateral list of <sup>12</sup> properties, with the basic

information first, followed by the graph of the data values (following the colour connotations of the application), followed up by a form for selecting properties to trigger the sound sonification process.

The final screen, the sound <sup>13</sup> selection screen, features another two element layout, except this time ordered in the vertical instead of the horizontal. In the higher element of the two, the user can select sounds from the backend storage area and preview them, add them or remove them from the list which is displayed in the element underneath. The



higher element attempts to give the user an abbreviated interface to use to select sounds and find out more about them before deciding whether to use the sound for sonification or not. The plus and minus buttons are attempts at conveying understanding through symbolism instead of words, taking meaning from mathematics to mean add and subtract (from list) respectively. The right angular bracket ( > ) is an attempt to represent the equilateral triangle commonly used for play commands, reinforced by the green background, which is also a commonly found indication of a play button. Again, this taps into colour connotations also used in urban areas (specifically traffic lights) to indicate a 'go' command. The pause command is a double pipe with a space inbetween ( | | ) over a red background, again attempting to tap into both pictographic representation of the pause button and

<sup>12</sup> Detailed sound mapping element

<sup>13</sup> Sound selection screen



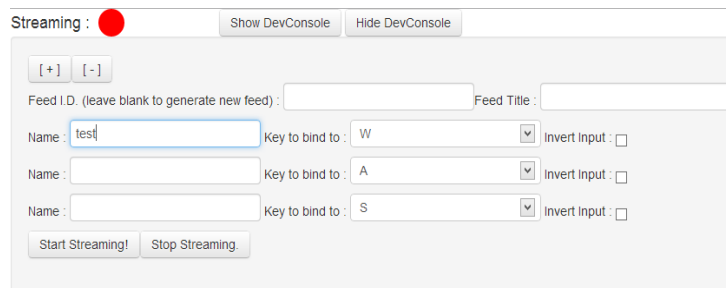
colour connotations associated with stopping.

The lower of the two elements serves as a grid styled list to present to the user to visualise the sounds that have been selected. In this element, the goal was clarity of vision and conciseness of content. To this end, each sound element contains the title of the sound, the description pertaining to the sound and rudimentary controls for each element, basing off of common pictographic depictions to convey meaning to the end user. The cross in the top right hand corner is a symbol for deleting the object, and is a common pictogram used in operating systems for computers across platforms.

### *Makey Makey to Cosm interface*

The design of the netChimes client, however, was strictly formatted by the Bootstrap CSS. This design choice was out of original planning to not make this a publicly available project to access and use. However, as time went on, it made more and more sense to make it publicly available, as searching for a similar project using a MakeyMakey as a physical input yielded no results.

The first screen that the user sees upon loading the application is the prompt to enter a valid API key for the Cosm service, along with a basic set of instructions and suggestions stemming from user testing. The logic behind placing the API prompt as the first page and not integrating it into the main

The screenshot shows a web interface for streaming data. At the top, there's a status bar with 'Streaming : ' followed by a red circular indicator, and two buttons: 'Show DevConsole' and 'Hide DevConsole'. Below this is a form with several fields. On the left, there are two buttons: '[+]' and '[-]'. The form has a 'Feed I.D. (leave blank to generate new feed) : ' field and a 'Feed Title : ' field. Below these are three rows of input fields. Each row has a 'Name : ' field, a 'Key to bind to : ' dropdown menu, and an 'Invert Input : ' checkbox. The first row has 'test' in the Name field and 'W' in the Key to bind to dropdown. The second row has an empty Name field and 'A' in the Key to bind to dropdown. The third row has an empty Name field and 'S' in the Key to bind to dropdown. At the bottom of the form are two buttons: 'Start Streaming!' and 'Stop Streaming!'.

application is to prevent users from thinking they might not need an API key, when, owing to the interactivity with the Cosm API of the application, they are intrinsically necessary.

This page then gives way to the main streaming page which

enables the user to input information pertinent to the stream to be sent. As the stream can be structured in the format of a form to fill with information, the structure chosen is a form which can be dynamically edited by the user to add or remove fields as necessary. The buttons in the form were kept in similar locations but differentiated vertically by function. This leads to the end user knowing that the form edit buttons are at the top of the form, with the form streaming controls at the bottom. The editing for each field is laterally placed, such that when adding a row and removing a row, the inputs form columns and can be easily navigated and changed. In addition, when streaming to Cosm, there are visual indicators showing both the state of the current form value being streamed, as well as the general status of the stream itself. The indicator lights were modeled after LED lights, ending up as circular indicators which change colours as the state of the application changes.

### **Implementation**

**Note:** MakeTheCosm refers to the root directory of the project.  
\* refers to all files contained in directory.

### Styling and dynamic DOM of the netChimes client.

*SRC : MakeTheCosm / MakeyMakey / css/ \**

*SRC : MakeTheCosm / MakeyMakey / index.html*

Both throughout the project and here specifically, the LESS syntax for CSS is being used. This main syntax has some advantages over CSS; Firstly, writing nested CSS (that is to say, having styling for elements explicitly inside containing elements) is much easier to understand than in CSS, and requires

the programmer to write nested code;

```
.element{
width:200px;
    .element2{
        width:10%;
    }
}
```

instead of the more formal CSS of;

```
.element{width 200px;}
.element .element2{width:10%};
```

Additionally, variables are an option within the LESS syntax, meaning that a particular colour can be changed globally once instead of needing to go through the entire style document. This is accomplished by adding the '@' symbol;

```
@PinkiePink = #FF69B4;
```

and then called by;

```
.element{
    background-color : PinkiePink;
}
```

After the code is written, the LESS compiler automatically parses the LESS file that has been written to CSS and outputs that CSS in a minified and streamlined form by removing duplicated style attributes and allowing for style inheritance.

The Bootstrap CSS module provides the styling for application of discrete classes to individual elements on the page. In theory, this makes styling a web-page as easy as including the Bootstrap CSS files in the right place and then adding the class names to each of the variables, as demonstrated by the Geocities Bootstrap theme<sup>14</sup>.

An example of the Bootstrap CSS is adding the classname 'well' to any attribute will apply the following CSS by default,

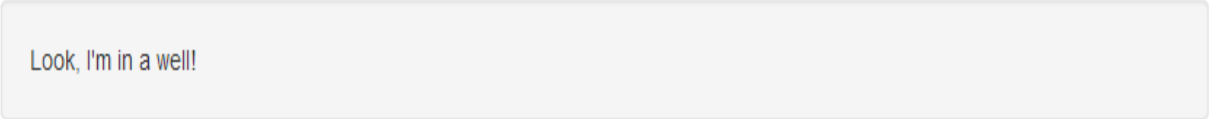
```
min-height: 20px;
padding: 19px;
margin-bottom: 20px;
background-color: #f5f5f5;
border: 1px solid #e3e3e3;
-webkit-border-radius: 4px;
    -moz-border-radius: 4px;
```

---

<sup>14</sup> Theme found here : <http://divshot.github.io/geo-bootstrap/>

```
border-radius: 4px;
-webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.05);
-moz-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.05);
box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.05);
```

which then gives the appearance;



Look, I'm in a well!

15

This gives programmers whose design and styling abilities are not as advanced as their programming the opportunity to create designs which are presentable and professional. In addition, there are multiple spin-off versions of bootstrap such as FlatUI<sup>16</sup> and modules which can be used to completely change the look and feel of the application.

In addition to this, the dynamic nature of the user input requires Javascript which can change the DOM. To this end, there are four important sets of code which are used;

```
document.getElementById("form").innerHTML;17,
```

```
document.getElementsByClassName("test");18,
```

```
var element = document.createElement('option');
element.setAttribute('value',"test");
element.appendChild(document.createTextNode("text");19
```

and

```
element = document.getElementById("test");
element.parentNode.removeChild(element);20
```

The first statement is a basic of Javascript and is used to swap the content of a <div> containing element by searching the document for a specific id attribute and then replacing content. Its' cousin is also in this list; the `getElementsByClassName` statement retrieves all elements in a page with a specific class attribute and returns an array of those values.

The third element is the formally correct way to add multiple elements to a page by creating the DOM structure as a DOM structure instead of a string, and then appending it to a page; this means that

---

<sup>15</sup> Image sourced from <http://twitter.github.io/bootstrap/components.html#misc>

<sup>16</sup> FlatUI can be found at <http://designmodo.github.io/Flat-UI/>

<sup>17</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/DOM/document.getElementById>

<sup>18</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/DOM/document.getElementsByClassName>

<sup>19</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/DOM/Node.appendChild>

<sup>20</sup> Explanation here : <http://stackoverflow.com/questions/3387427/javascript-remove-element-by-id>

contents of an entire DOM do not need to be re-written just to add a specific element, but rather the element can be appended at will without re-setting values such as checkboxes or drop-down menus.

The final code segment is one that enables the programmer to dynamically remove an element from the page by sourcing it by its' id attribute and then removing it from the parent element (the inverse of the .appendChild() operation).

This enables the programmer to dynamically change the content of the DOM, and, for example, to change the contents of a site from;

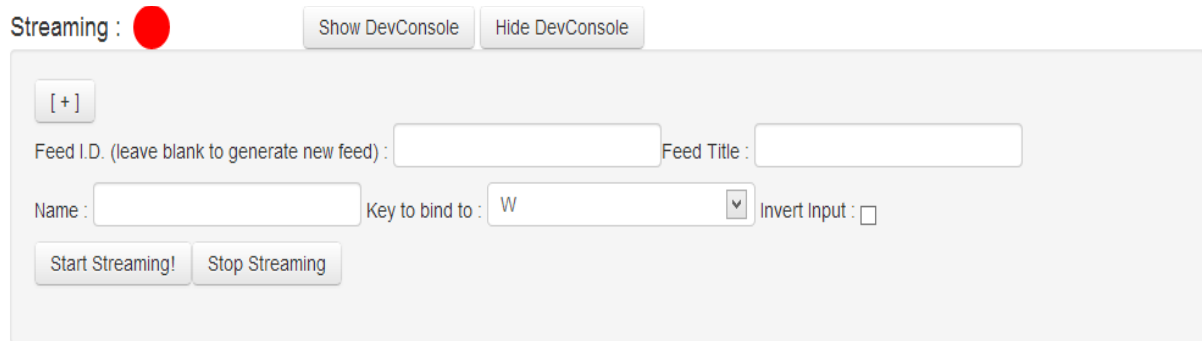


Streaming : ● Show DevConsole Hide DevConsole

API KEY :  Number of Inputs :  Submit

21

to a usable interface after the inputs have been validated ;



Streaming : ● Show DevConsole Hide DevConsole

[+]

Feed I.D. (leave blank to generate new feed) :  Feed Title :

Name :  Key to bind to :  Invert Input : ☐

Start Streaming! Stop Streaming

22

### Interface MakeyMakey with Javascript and record information.

SRC : MakeTheCosm / MakeyMakey / script / formGenerator.js

Accomplished using the MakeyMakey and cable as provided by the manufacturer to interface with the computer. Recording information was accomplished by a javascript front end registering keyPressed events, owing to the computer recognising the MakeyMakey as an external USB keyboard.

The technical background of this involves treating all inputs from the MakeyMakey as key presses, and registering listeners to those presses.

The first necessary task is to define which keys we want to listen to, and how these will be displayed to the user. In this case, the keys and their corresponding ASCII numbers are written to arrays to later be recalled in the following fashion;

```
var keys = [87, 65, 83, 68, 70, 71, 38, 40, 37, 39];  
var chars = ['W', 'A', 'S', 'D', 'F', 'G', 'UP', 'DOWN', 'LEFT', 'RIGHT'];23
```

<sup>21</sup> A screenshot of the first thing the user sees on

<sup>22</sup> A screenshot taken from the Cosm streaming webapp.

<sup>23</sup> Documentation here : [http://www.w3schools.com/js/js\\_obj\\_array.asp](http://www.w3schools.com/js/js_obj_array.asp)

The second important part of this is attaching the listener to the keys itself, which is accomplished by calling an anonymous function as soon as the document loads.

```
$(function() {  
    $(document).keydown(function(e) {  
        if (toStream == true) {  
            scanSubmitDown(charGet(e.keyCode));  
            console.log("KEYDOWN");  
            console.log(feedsValues);  
        }  
    });  
    $(document).keyup(function(e) {  
        if (toStream == true) {  
            scanSubmit(charGet(e.keyCode));  
            console.log(feedsValues);  
        }  
    });  
});
```

This adds listeners to the keydown and keyup events, which, when fired, send the ASCII keycode that was pressed to the defined methods; scanSubmitDown on the keydown event and scanSubmit on the keyup event. These keyCodes are then checked against the previously defined array of both keys that are expected and keys which have been specifically selected by the user.

```
function scanSubmitDown(info) {  
    for (var i = 0; i < formSubmitted[1].length; i++) {  
        if (info == formSubmitted[1][i]) {  
            if (formSubmitted[2][i] == true) {  
                //write a zero to feedsValues  
                feedsValues[i] = 0;  
            }  
            if (formSubmitted[2][i] == false) {  
                //write a one to feedsValues  
                feedsValues[i] = 1;  
            }  
            return true;  
        }  
    }  
    return false;  
}
```

This method is responsible for recording the key down events that the user has chosen (those in the formSubmitted[1] array) and writing them to the feedsValues array. The check on the formSubmitted[2] array is a check to see whether the user wanted to invert their input (for example,

write a zero on the keyPress event instead of a one).

The scanSubmit method, on the other hand, simply cycles through the user-selected keys and inverts the respective array value in the feedsValues array; a 1 becomes a 0 and vice versa.

Write backend code for personal web-based netChimes station to stream information to the cloud.

SRC : MakeTheCosm / MakeyMakey / backend / creator.php

SRC : MakeTheCosm / MakeyMakey / script / streaming.js

SRC : MakeTheCosm / MakeyMakey / script / location.js

Accomplished by using a PHP backend sending CURL requests to create new streams, with jQuery AJAX requests being made in order to send information to the cloud.

The location of the user, which is fed in to the stream as the origin, is retrieved by using the GeoLocation API, by making a call to the navigator.geolocation.getCurrentPosition() method.

```
navigator.geolocation.getCurrentPosition(GetLocation);
```

```
function GetLocation(location) {
    latitude = location.coords.latitude.toString().split(".");
    if(latitude[1].length >= 4){
        latitude = latitude[0]+"."+latitude[1].substring(0,4);
    }
    else{
        latitude = latitude[0]+"."+latitude[1];
    }
    longitude = location.coords.longitude.toString().split(".");
    if(longitude[1].length >= 4){
        longitude = longitude[0]+"."+longitude[1].substring(0,4);
    }
    else{
        longitude = longitude[0]+"."+longitude[1];
    }
    console.log("Got Co-ordinates to "+location.coords.accuracy+"accuracy.");
    console.log(latitude+", "+longitude);
}
```

A function is specified as the method argument, which then passes the returned value of the location straight in to the specified function. In this case, the Longitude and Latitude are shortened, as the COSM servers throw errors if there are more than a certain amount of numbers as part of the floating point number; 4 was decided upon as the limiter it still displays enough accuracy and is under the threshold at which COSM throws errors.

The next step after this is creating the new feed which the user wishes to create.

```
$apiKey = $_GET['APIkey'];
```

```

$thing = '"X-ApiKey : '.$apiKEY.'"' ;
$theurl = 'http://api.cosm.com/v2/feeds';
$ch = curl_init($theurl);

```

As API keys are tied to each account, the API key needs to be submitted to the php script as part of a GET submission form for the php script to create the new feed. This is then combined with a string to produce the header being sent to COSM for verification.

```

curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'POST'); // -X
curl_setopt($ch, CURLOPT_BINARYTRANSFER, TRUE); // --data-binary
curl_setopt($ch, CURLOPT_HTTPHEADER, array("X-ApiKey:".$apiKEY)); // -H
curl_setopt($ch, CURLOPT_HTTP_VERSION, CURL_HTTP_VERSION_1_0); // -0
curl_setopt($ch, CURLOPT_POSTFIELDS, '{"title":"Newly Created Feed With PHP"}');
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_VERBOSE, 1);
curl_setopt($ch, CURLOPT_HEADER, 1);

```

This next step sets up our CURL request, with the most important specifications being to set the request type as a POST request, to add the header containing the API key and to specify the content of the POST request being sent to the COSM servers.

```

$response = curl_exec($ch);
$header_size = curl_getinfo($ch, CURLINFO_HEADER_SIZE);
$header = substr($response, 0, $header_size);
$headers = get_headers_from_curl_response($response);

```

The final step of setting up the feed is set by executing the CURL statement, and handling the response by splitting the header section from the body of the response such that the information contained in the header is parsed into an array.

```

function get_headers_from_curl_response($response)
{
    $headers = array();
    $header_text = substr($response, 0, strpos($response, "\r\n\r\n"));
    foreach (explode("\r\n", $header_text) as $i => $line)
        if ($i === 0)
            $headers['http_code'] = $line;
        else
        {
            list ($key, $value) = explode(':', $line);
            $headers[$key] = $value;
        }
    return $headers;
}

```



This array is then parsed to JSON and returned to the program having passed in the original request.

The original request for a new feed is set up in a javascript function taking advantage of the jQuery AJAX module (to use instead of XMLHttpRequests).

```
$.ajax({
    type : "GET",
    url : "backend/creator.php",
    data : {
        APIkey : apiKEY
    },
    async : false,
    .
    .
    .
    .
}
```

Inside this function, there is the call to the php script we have set up, as well as the setup of the streaming method, which is called inside a setInterval method;

```
pushInterval = setInterval(pushToServer, 660);24
```

This also sets the pushInterval variable to the unique identifier created by the setInterval method such that it can later on be deleted. The pushToServer method specified as to be called every 660ms (to conform with the COSM rate limiting) then constructs a JSON object and sends the JSON object to the COSM servers to update the freshly created feed.

Create a user interface for the map/sonification part of the project.

SRC : MakeTheCosm / Mapping / index.html

SRC : MakeTheCosm / Mapping / script / history.js

SRC : MakeTheCosm / Mapping / script / transfers.js

Accomplished using HTML and jQuery to dynamically modify the CSS of the webpage to change which container elements filled the viewport of the browser. In addition, javascript was used to write the change slide commands.

The DOM structure in the main part of the project is formed as an unordered list with all standard list styling having been removed to enable the slideshow-esque treatment. In addition to this, the HTML5 history API is used to treat each slide in the slideshow as a separate webpage, such that they can also be navigated using the “return to previous page” and “return to next page” buttons in most web browsers and on some keyboards.

The current tag for the history is retrieved from the URL by using URL detecting methods to retrieve the substring of the whole URL by having the index of the hash which depicts the element to be used as the history.

---

<sup>24</sup> Documentation here : [http://www.w3schools.com/jsref/met\\_win\\_setinterval.asp](http://www.w3schools.com/jsref/met_win_setinterval.asp)

```
var hashThing = document.URL.substr(document.URL.indexOf('#'), document.URL.length);25
```

The next step is to push the current history string to the history stack. This pushing to the history stack is all that is necessary to create the history object to navigate to, with the changing the screen to the most current version using the `changeScreen()` method being necessary when using the backwards and forwards navigate through history keys.

The slide navigation interface is achieved by adding buttons and key events by using javascript. Click listeners are applied to the buttons using jQuery, as well as key events to achieve slide transfers.

```
$(document).keydown(function(e) {  
    if (e.keyCode == 37) {  
        backOne();  
    }  
    if (e.keyCode == 39) {  
        nextOne();  
    }  
});  
$('#prev-button').on('click', function() {  
    backOne();  
});  
$('#next-button').on('click', function() {  
    nextOne();  
});
```

The end goal in both cases is identical; to call either the `backOne()` function or the `nextOne()` function. These functions use `zepto.js` to cycle through the array of collected slides collected by the

```
$('.slideshow .slide.active');
```

jQuery. In addition to this, the pertinent class names are added to the specific list element that has been cycled to, triggering CSS animations.

The slide navigation is managed by applying the “active” class to the DOM element in question. This DOM element is then transitioned to using the CSS

```
-webkit-transform: translateX(100%);
```

to automatically transition from one slide to the next. In addition, the application detects when the user is on the first or last slide, and intelligently disables the navigation elements to navigate out of the bounds of the webpage.

```
if (active[0] != totalSlides[0]) {  
    document.getElementById("prev-button").className = "";
```

---

<sup>25</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/DOM/document.URL>

```

}
if (active[0] == totalSlides[totalSlides.length - 1]) {
    document.getElementById("next-button").className = "hidden";
}

```

This results in the buttons to navigate out of bounds being hidden from the user, rendering them inaccessible to the users.

Write feed retrieval backend for location interface and implement location interface.

SRC : MakeTheCosm / Mapping / script / maps.js

SRC : MakeTheCosm / Mapping / script / location.js

SRC : MakeTheCosm / Mapping / script / mapUpdater.js

SRC : MakeTheCosm / Mapping / backend / dataRetrieval.php

SRC : <http://api.tiles.mapbox.com/mapbox.js/v0.6.7/mapbox.css>

SRC : <http://api.tiles.mapbox.com/mapbox.js/v0.6.7/mapbox.js>

Accomplished using the Geolocation API, along with a recurring AJAX call to the cloud to retrieve information at as often as the API keys provided would allow. This was then mapped onto a Mapbox map object using the longitude and latitude coordinates of the stream. These markers were then made interactive and write the data into a separate element of the DOM on a click event.

The creation of the map is left solely to the mapbox API's, with a script and a stylesheet being included for the local creation of the map. The local content of the map is then created using the mapBox API and written to the DOM.

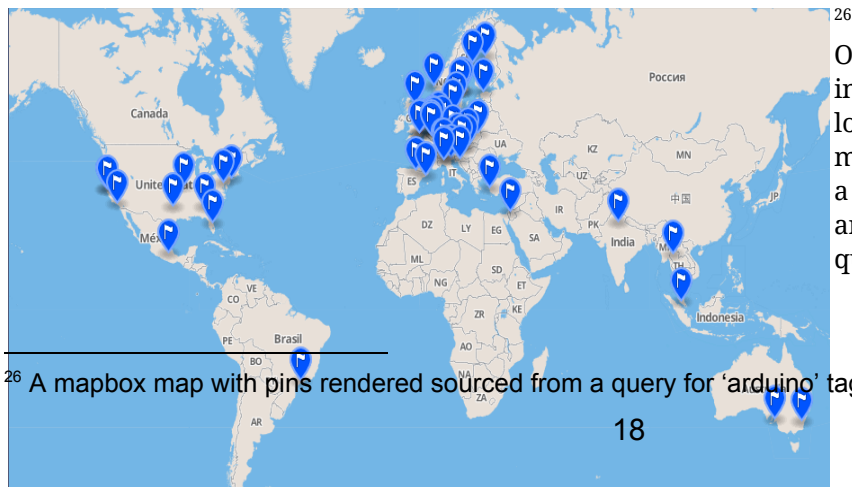
```

var layer = mapbox.layer().id('examples.map-vyofok3q');
var map = mapbox.map('map',layer,null,[]);

map.zoom(2).center({
    lat : 20,
    lon : 0
});

```

This particular snippet creates the globe map as seen in the 'map' div in the main DOM, and adds the mapbox layer specified as the layer argument in the map constructor on the creation of the map. After creation, the map is then zoomed to a specific layer and centered such that it presents a picture of the globe likely to be using the COSM service.



26 A mapbox map with pins rendered sourced from a query for 'arduino' tags.

Once the map has been loaded into the DOM as part of the page load along with the relevant map, more code is executed which adds a layer for the marker pins which are added when successful queries are returned.

Once the map has been created, listeners are then added for the mouseOver events of the individual marker pins such that tooltips can be created.

```
interacting.formatter(function(feature) {
    var o = '<h3><span id="feedId"'
+ 'style="display:none;">' + feature.properties.id + "</span><span"
+ 'id='feedName'>' + feature.properties.feedName + '</span></h3>' +
    '<ul>';
    var thing = 3;
    if(feature.properties.datastreams.length < 3){var thing =
    feature.properties.datastreams.length;}
    if(feature.properties.datastreams.length > 3){var thing = 2;}
    for(var i = 0; i < thing; i++){
        o +=
" <li>" + feature.properties.datastreams[i].id + ":" + feature.properties.datastreams[i].current
_value;
    }
    if(feature.properties.datastreams.length > 3){
        o += " <li>-> More than 3 inputs <-</li>";
    }
    o += "</ul>";
    return o;
});
```

The contents of the tooltip are created on the fly by the formatter function<sup>27</sup> which specifies the contents of the tooltip by cycling through the data retrieved from the COSM servers and places the pertinent information inside them as HTML content.

Another step taken while the map is initiating is to add a click listener to each element contained in the layer as follows;

```
layer.factory(function(feature) {
    var elem = mapbox.markers.simplestyle_factory(feature);
    MM.addEvent(elem, 'click', function(e) {
        var o = '<h3>' + feature.properties.feedName + '</h3>' +
            '<ul>';
        for(var i = 0; i < feature.properties.datastreams.length; i++){
            o +=
" <li>" + feature.properties.datastreams[i].id + ":" + feature.properties.datastreams[i].current
_value;
        }
        o += "</ul>";
        e.innerHTML = o;
    });
});
```

---

<sup>27</sup> Documentation here : <http://mapbox.com/mapbox.js/api/v0.6.7/#interaction.formatter>

```

        e.stopPropagation();
    });
    return elem;
});28

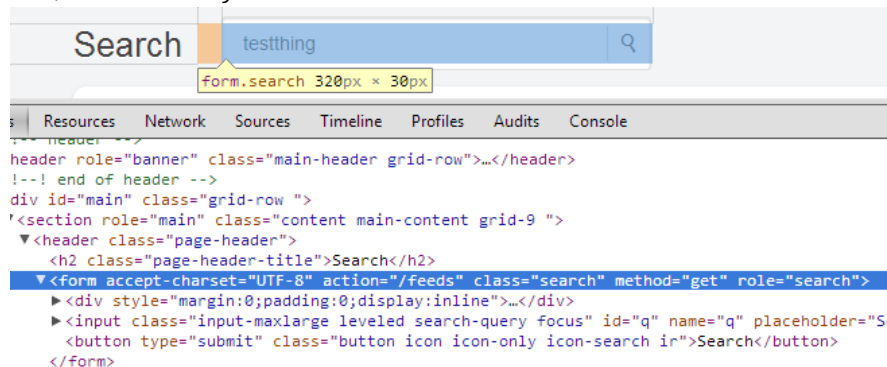
```

This adds a click listener to each new pin element added to the layer, passes information out of the mapBox instance to be used externally.

The information to populate the map with is retrieved by sending a request to a PHP script, which in turn sends a request to the COSM servers which spoofs COSM's own feed search and filtering requests. What this means in practicality is that the URL cosm produces for a certain set of results is ;

`https://cosm.com/feeds?utf8=%E2%9C%93&q=test&order=relevance&status=live`

We can discard the parts prior to the ? as the URL we need to send our query to, and focus on the second part, namely the string of `utf8=%E2%9C%93&q=test&order=relevance&status=live` and decode this so we can spoof it. If we break it up on the ampersands (&), we receive `utf8=%E2%9C%93`, `q=test`, `order=relevance` and `status=live`. In this case, the attribute for the UTF-8 is the key code for a checkbox, and notifies the server which character set is being used. The `q` attribute is the actual search query, with the `order` attribute being how the results are ordered, and in this case `relevance` retrieves the most accurate results. The `status` attribute follows up the rear and is set to `live` in this case, as the only useful information for sonification is from broadcasters which are actually



broadcasting. The final piece of information which we need is whether the request is a POST or GET request, which is found by using the “Inspect Element” tool in Google Chrome (though possible on any browser), and finding the “method” attribute for this particular form, which is a ‘GET’.

This combination of attributes also retrieves relatively consistent results, meaning that once the user selects a data stream for sonification, it is unlikely to vanish off of the user’s selected feeds while it is being used for sonifying.

This combination of attributes also means that we can send a jQuery powered AJAX request to a PHP script, which in turn sends a CURL request (specified in greater detail in the Feed Creation section of the netChimes node section of the technical description) to retrieve the information. This time, an extra field is added into the CURL request, namely the POSTFIELDS section.

```
curl_setopt($ch, CURLOPT_POSTFIELDS, "per_page=100&q=" . $q . "&order=created_at&status=live");
```

<sup>28</sup> Implementation guide : <http://mapbox.com/mapbox.js/example/centering-markers/>

<sup>29</sup> Screenshot of the Cosm search results page. The method type is decidedly a GET.

This section serves to send arguments as part of our request, which uses the previously discovered attributes to modify our query as well as the `per_page` attribute, which limits our request to 100 responses which satisfy our query; this is both to keep latency low and give the user as much information as (reasonably) possible to choose from to sonify.

Once the CURL request receives a response from the COSM server, the PHP script then returns the data to the client in the form of a JSON object containing as many valid objects as were returned by the query to be displayed on the global map.

On a successful response to the request sent to the PHP script, the current contents of the pin layer on the map are dropped using the

```
layer.features([]);30
```

command, which supplies an empty array to the layer object of the map; effectively clearing it. Then, the results from the query are parsed and for each usable result received,

```
json.push(response[i]);
layer.add_feature31({
  'geometry' : {
    'coordinates' : [response[i].location.lon, response[i].location.lat]
  },
  'properties' : {
    'marker-color' : colour,
    'marker-symbol' : 'embassy',
    'feedName' : response[i].title,
    'desc' : response[i].description,
    'datastreams' : response[i].datastreams,
    'id' : response[i].id
  }
});
```

is called, which firstly adds the current data point to the json global data object and then adds a feature, in this case a marker using the geographical co-ordinates of the data retrieved, to the map using the GeoJSON standard<sup>32</sup>.

Once the information is retrieved from COSM, and the markers slotted in to the map, a clickListener is added to each individual marker, such that the method 'getPertinentInfo' is called.

```
var markers = document.getElementsByClassName('simplestyle-marker');
for (var i = 0; i < markers.length; i++) {
  markers[i].addEventListener("click", getPertinentInfo);33
```

---

<sup>30</sup> Documentation here : <http://mapbox.com/mapbox.js/api/v0.6.7/#markers.features>

<sup>31</sup> Documentation here : [http://mapbox.com/mapbox.js/api/v0.6.7/#markers.add\\_feature](http://mapbox.com/mapbox.js/api/v0.6.7/#markers.add_feature)

<sup>32</sup> Specification here : <http://www.geojson.org/geojson-spec.html>

<sup>33</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/DOM/EventTarget.addEventListener>

```

    }

```

This is accomplished by using the `getElementsByClassName` method, then cycling through the array given to append the `clickListener` before the user is able to interact with the map to get information.

When the `getPertinentInfo` method is called, the first code executed is integral to the rest of the application.

```

if (document.getElementById("feedId") != null) {
    var idOfThing = document.getElementById("feedId").innerHTML;
}
for (var i = 0; i < json.length; i++) {
    if (idOfThing == json[i].id)
        .
        .
}

```

This piece relies on the logic that the tooltip from the pin has to be available as the tooltip is triggered on a `mouseover` event. This piece of code retrieves the ID of the feed based on the HTML contained in the tooltip, and then proceeds to use this to identify the data object necessary to replicate from the `json` global data object.



What then happens is the html that appears in the data investigation div element is generated based on the contents of the retrieved object and parsed into the DOM by using the `document.getElementById("").innerHTML` java script method as detailed in the 'styling and dynamic DOM of the netChimes node' section, to produce a functional location interface.<sup>34</sup>

Implement feed selection and

committing to a list

SRC : *MakeTheCosm / Mapping / script / mapUpdater.js*

SRC : *MakeTheCosm / Mapping / script / feedOrdering.js*

The feed selection creates dom elements from the selected feed in a neighbouring window, where the vital statistics of the feed are displayed using the information that is in the selected feeds data object to synthesise the dom elements on the fly. This is initiated when the data is posted to the map on the first response, which culminates with a click listener being added to each marker element

<sup>34</sup> Screenshot taken from a development version of the project

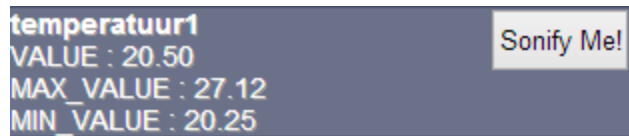


(which, with some element inspection, was found to have the class “simplestyle-marker”).

```
var markers = document.getElementsByClassName('simplestyle-marker');
for (var i = 0; i < markers.length; i++) {
    markers[i].addEventListener("click", getPertinentInfo);35
}
```

Once this clickListener is activated, the getPertinentInfo method is called. Once the method is called, the first action is to retrieve the ID of the feed that is being viewed thanks to having a DOM element in the tooltip for each marker containing the ID of the feed, which is displayed to the user. Once the program has the ID of the feed, the information is then used to cycle through the entire retrieved dataset in search for the ID of the dataset which matches the selected ID.

After the data object represented by the ID has been retrieved by the program, it then creates the content to fill the DOM space where the element is which will contain the information on the feed being pulled. This is carried out thanks to the rigid nature of the Datastream feeds, which are pulled from Cosm.<sup>36</sup> The information to be parsed into the DOM area is constructed by appending to a string with the equivalents in string form which represents the DOM structure that is being achieved. To this extent, the amount of individual data points in the data stream are added to a list, which is then styled and displayed to the user for selection. The information in the DOM element is updated every time the Cosm server replies to the Ajax requests as part of the “Information update” section of code.



Once the prior created DOM element is rendered, each data point has a button which gives the user the possibility to add the data to the next stage of the app. This button is also present in all other panes which contain the relevant information for the particular data point, giving the user an extreme amount of control over whether and how to use the data point for sonification.

The next stage of the app involves adding the data point we have just selected to another data object of active information, which, in its’ initial form looks as follows;

```
{
  "123597": [
    ["CO", "1012", 1683, 0, Highcharts Chart,
      [">>", 20, "2||chimes||Chimes_b", true]
    ],
    ["DTH-H", "20", 40, 0, Highcharts Chart,
      ["|==|", 10, "2||chimes||Chimes_b", true]
    ]
  ]
}
```

The data object representing the data that has been selected is roughly grouped by the ID of the feed that it comes from, and then with sub arrays which are unique per data point and have the data point’s name, the maximum, minimum and current values, additionally containing Graph objects and as the

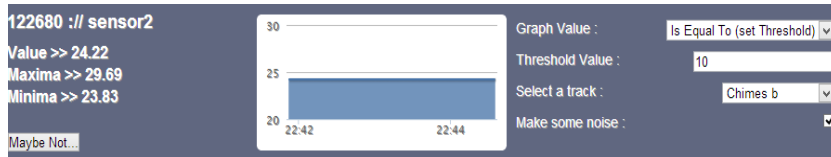
<sup>35</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/DOM/EventTarget.addEventListener>

<sup>36</sup> Example reproduced in Appendix B from the Cosm website : <http://cosm.com/docs/v2/>

final element another array which governs the sound playing.

```
[ "|==" , 10, "2||chimes||Chimes_b", true]
```

This array is used to determine the sound to be played by containing the check symbol, the threshold value, the unique sound string to identify the sound, and a boolean which dictates whether the sound is to be checked against or not.



The DOM elements are then created based on these data objects in the second screen,<sup>37</sup> with the element representing a given datapoint containing the Highcharts chart<sup>38</sup> to visualise

the data point. The generation itself is, again, a string being inserted directly into the DOM which happens to fulfil all the HTML markup constrictions necessary for valid HTML. The rows of information are created by adding list elements to an unordered list, with styling to discard the standard list styles of bullet points.

```
122680 :// sensor2
Value >> 24.22
Maxima >> 29.69
Minima >> 23.83
Maybe Not...
```

In addition to this, the user is presented with options for type of trigger, value to trigger on, sound to use and whether to start checking against the incoming information to see if an audio event is needed. There is also a smaller, less interactive<sup>39</sup> toolbar on the side which is created in much a similar way as

the main section of the page, except for sporting only the barest information about the information from the sensor, which updates in real time in much the same fashion as the other panes holding data throughout the project; a method is called every time the web application receives new information which handles the information being displayed.

Not only can the data points be added to the aforementioned data structure<sup>40</sup>, but they can also be removed from the data object and all DOM elements by passing the feed ID and particular data point name to a function. This function then cycles through the data object of selected data and removes the data entry dependant on the size of the datastream's object. If the object is not losing its' last element, the position of the element to remove is determined and then has a

```
chosenFeeds[feedId].splice(i, 1);41
```

command ran on it, which leads to the removal of the data element from the array. Another option is to delete the named array entirely, using the

```
delete chosenFeeds[feedId];42
```

<sup>37</sup> Image screenshot from project.

<sup>38</sup> Example reproduced in Appendix C from project source code. API available : <http://www.highcharts.com/>

<sup>39</sup> A screenshot of the data overview pane in the project.

<sup>40</sup> Selected data points array specification in Appendix D

<sup>41</sup> Documentation found here :

[https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global\\_Objects/Array/splice](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Array/splice)

<sup>42</sup> Documentation found here :

<https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Operators/delete>

command, which removes the property from the data object entirely, and is only used if the size of the resulting array would be zero, or rather, would be empty.

Implementing data point graphing.

*SRC : MakeTheCosm / Mapping / script / highcharts.js*

*SRC : MakeTheCosm / Mapping / script / feedOrdering.js*

On the data JSON object being updated, a new dom element will be created which will contain the visualisation of the most recently received information using the highCharts graphing API. This graphing API lets the programmer in question create any number of types of graph using a data management and rendering package. The full creation object needed for a highcharts chart is detailed in Appendix C. The creation process is managed by creating a graph object similar to any object oriented language by using the new keyword.

```
new Highcharts.chart({});
```

From this point, there are multiple object options for the chart, detailing both the appearance, type and functionality of the graph. In this instance, an area type of graph is used to clearer demonstrate the current values in the system. The period that the graph shows is governed by two variables : graphUpdateTime and dataResolution. The graphUpdateTime variable is a number dictating how many minutes of data the graph should show, and the dataResolution variable dictates how many data points should be displayed per minutes.

```
var data = [],
var time = (new Date()).getTime(), i;
    for (i = graphUpdateTime * graphResolution * -1; i <= 0; i++) {
        data.push({
            x: time + i * 1000,
            y: -999999
        });
    }
return data;43
```

What this function, contained in the “data” section of the object, serves to do is to pre-populate the graph with enough points such that the length of data to be displayed is held intact. The graph adds points at the -999999 mark such that there are points on the graph, but the user does not see these as existing, as practical information tends not to range to the negative one million mark.

As the information in the selected information data object is updated on a consistent basis when the data is updated by the server, a function can be called to conform with the data resolution constraints imposed by the user.

```
setInterval(function() {
    graphUpdate();
}, 1000 / graphUpdateTime);
```

---

<sup>43</sup> Source code taken from feedOrdering.js project file.

The first method to call the graphUpdate function reliably is to use a setInterval method, which runs in a predictable fashion. As the graphUpdateTime variable is in minutes, dividing 1000 by it results in the quantity of milliseconds needed to offset the function calls by to fill the graph completely over the graphUpdateTime specified.

```
function graphUpdate() {
    for (var key in chosenFeeds) {
        for (var i = 0; i < chosenFeeds[key].length; i++) {
            chosenFeeds[key][i][4].series[0].addPoint(
                [
                    (new Date()).getTime(),
                    parseFloat(chosenFeeds[key][i][1])
                ],
                true,
                true
            );
        }
    }
}
```

The graphUpdate function then cycles through the selected feeds data object, finding the discrete graph object and adding a new point to the graph by calling the addPoint()<sup>44</sup> highcharts method. This is called on the element in the data point object which stores the highchart graph to update the data points. As this method requires both a Y and an X co-ordinate, the time is fetched and parsed to insert in to the graph as the X coordinate, with the most recent current value in the selected data points object being used to create the Y coordinate.

#### Implement user selection of sounds and assignment of sounds.

SRC : MakeTheCosm / Mapping / script / audioSettings.js

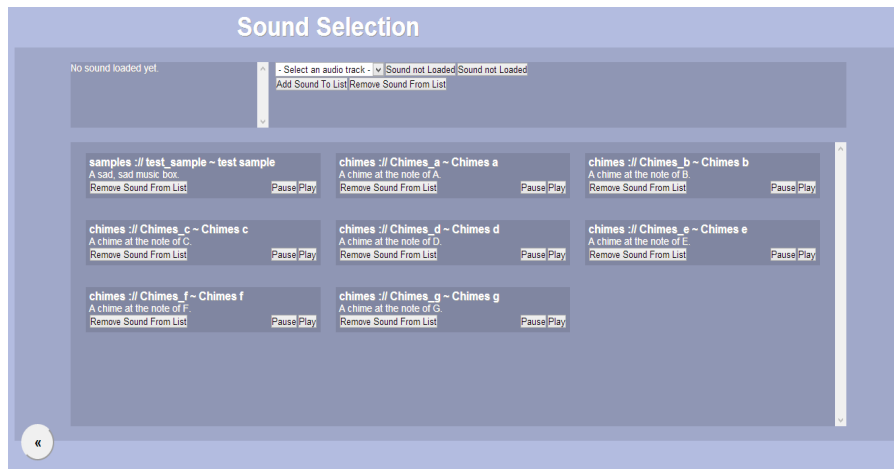
SRC : MakeTheCosm / Mapping / audio / audioList.json

The most important section of this is the information structure detailed in Appendix F. The data object contains three main sections, which are retrieved by the application using a jQuery powered AJAX request. The opts entry contains the data formats which should be parsed into the HTML5 audio elements being created to include all audio formats needed for a cross browser experience. Additionally, the init entry contains the information about which files should be pulled on start up with the format used for playing the sounds and for adding the ID to the HTML5 audio object.

```
"opts": [
    ["mp3", "mpeg"],
    ["wav", "wav"]
],
"init": ["0 | samples | test_sample", "0 | chimes | Chimes_a"],
"samples": [
    ["test_sample", "A sad, sad music box."]
]
```

---

<sup>44</sup> Documentation here : [http://api.highcharts.com/highcharts#Series.addPoint\(\)](http://api.highcharts.com/highcharts#Series.addPoint())



In addition to this, there are further entries which correspond directly to each subfolder in the audio directory which contains audio data. On loading the page, all of the sounds in the “init” array are pulled through to the main application and inserted into the DOM as a list in the final screen; the sound editing screen and

<sup>45</sup> additionally dropped in to

an array to keep track of the sounds that are being used. Once this has been added, the contents of the folders as detailed into the JSON object are added to a select box, which serves to let the user add new sounds into their selections to use for sonifying the data retrieved from Cosm by adding the entire list of sound files that can be selected from as options to a dropdown select menu.

On selecting a value from the dropdown box, an onchange function is triggered, which retrieves the value of the selected item and passes it to the addAudio function, which takes the value and cross references it against the values of audio that are already in use, and, if unfound, adds it to a preview element which is hidden. Additionally, the buttons in the top bar are rewritten to contain code instead of the filler text present when the page loads, which gives the user the option to preview, pause, add the sound and remove the sound from any lists which are present.

When the add to list command is given, the value is checked again against sound arrays to make sure the sound is not being added twice. After this, DOM values are created by appending strings, which are then appended to the DOM structure in the sound lists pane. Finally, the lists of sounds which have been added to the data point sonification panes need updating, and hence the updateAudioLists function is called with the parameters of “add” and the unique ID of the sound.

When the remove function is called, the inverse of the add function is carried out; namely, the elements pertinent to the sound including the HTML5 audio element and the detail in the sound pane is pulled out of the DOM using javascript. Additionally, the updateAudioLists function is called, but this time with the parameters of “remove” and the unique ID of the sound.

The updateAudioLists function is a utility function as it is able to deal with both the adding of sounds to the options of the data points pane and the removing of sound from the options of the data points pane. This is accomplished by distinguishing between the first parameter passed to the function in the form of the “add” or “remove” strings. The effect of this function is then to find elements to which the new sound should be added or removed, and then using the

```
document.getElementsByClassName("soundOptions")[i].appendChild(element);
```

or

<sup>45</sup> Screen selection taken from developmental version of application

```
document.getElementsByClassName("soundOptions")[i].removeChild(element);
```

functions respectively.

### Implement playing of sounds.

*SRC : MakeTheCosm / Mapping / script / audioSettings.js*

The important thing to consider when playing audio on the web is that each browser manufacturer such as Opera, Mozilla, Microsoft or Google will have a different specification of which audio format they require. As enumerated upon in Appendix E, in most cases a browser will have a file format it outright does not support, such as Internet Explorers 9 and above only supporting the mpeg file format. However, in some cases, a browser such as Google's Chrome will support and play the three major audio file types of mpeg, ogg and wav.

```
<audio id="2chimeschime_a">
  <source src="chimes/chime_a.ogg" type="audio/ogg">
  <source src="chimes/chime_a.mp3" type="audio/mpeg">
  <source src="chimes/chime_a.wav" type="audio/wav">
</audio>
```

To this end, the audio element in the DOM (and all audio files included in the project) need to be in these three file formats to ensure cross browser compatibility for the optimum user experience.

The playing of the sounds for the project is heavily reliant on dynamically generated HTML5 audio elements<sup>46</sup>, on which the `play()` method can be called once the particular data point satisfies checks which are validated using the discriminatory symbol used in the Appendix D. Depending on which discriminatory symbol has been selected depends on which checks are carried out; ">>" is the Greater Than symbol, denoted for the graph current data being greater than the threshold value set. The opposite value to this is the "<<", which corresponds to the Less Than symbol in programming. The final two sets of symbols used in the application are the 'value changes' symbol, and the 'is equal to' symbols. The 'value changes' symbol ("=|=") adds the current value to a check, which triggers once the current value of the data point changes from the set value. The final pair of symbols are "==" and "|==|", which do two entirely different things; the first symbol selects the current value to trigger if the feed value is equal to its' current value, while the second requires a value to be specified as the threshold value which is then used for triggers.

The final part of the audio being played is the background information for the sound that needs to be played being passed to a method which takes the unique ID for each sound and decodes it, re-encoding it into the format used to write ID's of the audio elements.

```
function playNoise(songId) {
  var index = songId.split("||")[0];
  var songFolder = songId.split("||")[1];
  var songName = songId.split("||")[2];
  console.log(songId);
```

---

<sup>46</sup> Documentation here : <https://developer.mozilla.org/en-US/docs/HTML/Element/audio>

```
document.getElementById(idPrefix + songFolder + songName).play();  
}
```

The unique ID being passed is of the structure “2||chimes||chimes\_a”, which is a representation of which point in the audio array the sound is, the folder the sound is contained in and the name of the sound. This information is re-encoded without the || characters to find the unique audio element, and then call the `.play()`<sup>47</sup> method, which plays the sound once and once only.

## Evaluation

### *Function in comparison to specification.*

According to the specifications set out previously, the final product does function according to the set out restrictions and technical requirements.

The first part of the project to produce a personal netChimes node, as outlined in the Implementation section, functions by verifying an API key, which then allows the user to create a new feed by filling out a form. This form gives the user the ability to add as many fields as the user needs (or until they reach the limit of the inputs of the Makey Makey), along with the ability to customise the feed generated. In this instance, the customisation consists of giving the data a name, selecting which key to map to, whether to invert the data and whether to stream the information or to pause the information. In addition, the information being streamed is displayed by elements which change colour, powered by Javascript.

The main, mapping section of the project also met the technical requirements and specifications set forth by the previous sections of the document: In short, it grants the end user the ability to select data and sonify it. The sonification is limited by only the amount of sounds which are in the pre-defined set and the triggering constraints. The first slide, as put forth previously, allows the user to select from a user selectable amount and category of data feeds sourcing from Cosm by clicking on plotted points on a global map representative of the data. The second slide consists of the meat of the project : the sound point is plotted on a graph for a visual inspection over time, and, as well as data being viewed, the sounds can also be triggered by way of a small form which is checked against criteria via a Javascript monitor. Additionally, there is a quick view bar which lets the user scroll through all the data points that have been added and quickly remove extraneous points from the list. The third screen is strictly audio based, and as a result handled the adding and removing from the list of selected tracks. These tracks can be previewed, removed and added to pre-existing lists and then used as a trigger for other data elements thanks to automatic list population.

### *Does the system fulfil original requirements?*

The requirements from the start of the project was that the end product needed to fill a void for a particular project. In this aspect, the project has been a definite success, owing to the implementation of all necessary features to complete the task at hand, namely, enabling the end user to sonify data. Additionally, this project has created a way for people with less technical expertise to take part in sending data to the cloud for other participants to use.

### *User Testing Results*

The user testing in this project was cursory, at best, however still yielded insightful results, owing to the major problem that the developer of the project is likely to be much too close to the project to objectively see usability issues with the interface presented with the project.

---

<sup>47</sup> Documentation here :

<http://www.position-absolute.com/articles/introduction-to-the-html5-audio-tag-javascript-manipulation/>



The testing format was extremely informal; the subject was briefed on the subject matter of the project, along with any questions answered that the respondent had. As these were so frequent, the testing devolved into a walkthrough of the software, with issues noted as they came up. The results of the testing were then grouped by each section of the projects, namely each screen of the sonification section of the project (Splash screen / API key entry, mapping page, sonification page and sound selection page) and the process of creating a unique feed using the secondary section of the project.

Despite grouping the results by section of the project, there was overarching feedback which influenced parts of the project. One such instance of feedback was the common repetition of the sentiment that there was little to no explanation in the web application. This lack of explanation extends both through the functionalities of the application and individual calls for information from the user.

To this end, the project could be further developed in light of this to include tooltips on certain elements which proved confusing to the respondent, making sure only to trigger after a time at which an experienced user would have already used the element. However, it would also be plausible to append a brief description to the splash screen for new users, or link to a video explanation.

This is evident first and foremost in the splash screen for the sonification side of the project : during testing, it became obvious that the calls for input of a search query and a Cosm API key confused the user. In addition, the browser based prompts to share the user's location with the app for localisation purposes was met with confusion.

To remedy this, the splash screen requiring an API key and search query will be modified to include an explanation to the user as to why they are supplying an API key and search query, as well as where to obtain an API key for Cosm. Additionally, a section will be appended to the explanation briefly detailing how the geolocation data is used and, more importantly, emphasise that the information will not be retained by the website for any reason.

The first screen that the user sees in the sonification section of the project resulted in different responses than just the common answer of a sentiment of a lack of clarity and purpose. In this case, the participant was unsure what icons on the map were supposed to represent. In addition to this, the participant was unsure about how to use the interface, despite instructions and options being presented on screen. Confusion also arose over the term 'sonify' being used for the button labels, as well as other terms such as "Datastreams rendered".

End users understanding what the markers rendered to the map are supposed to represent is an issue of understanding how individual pictograms are interpreted by end users. As however, in this case, the survey consisted of one respondent, this question is one which can be categorised as not unimportant, but rather warranting more study. The respondent being unsure of instructions despite their presence, however, is possible to be remedied by making the instructions more obvious on the screen, by potentially applying a CSS effect on them to make them more obvious to the user upon loading the page. As for the term confusion, this could be solved by simplifying the language instead of using technical, elaborate terms and making sure that, like the individual marker pictograms, the wording is understood the same way by the target demographic.

The sound mapping interface presents the user with yet another challenge : the ability to map discrete values selected in the previous pane to sound to be triggered by specific cases also set out by the user. Aside from the common complaint of a lack of explanation, the issues found here were a lack of

responsiveness of playing sounds and a lack of navigational aids for the end user. Another feature which was noted to be wanted would be the hypothetical ability for a user to upload their own sounds into the application to use with the pre-existing set. In this case, the applications' not noticing changes in the data and therefore having a low data resolution is down to the rate limiting inherent to the Cosm service.

A lack of description of the functionality and how to use will be resolved by appending a section to the introductory screen of the application comprising of a vague "how to" section on how to use the application. Sorting the issue of the respondent wanting to be able to upload their own sings is slightly more difficult, and would require re-tooling the custom backend solution, which is a task in the Future Work sub-section in the Conclusion and Future Work section.

The final screen of the sonification part of the application is dedicated to sound selection, where the user can select sounds to use to map to the data triggers specified in the sound mapping slide. When the respondent tested this slide, the response was not one of a strict user experience standpoint, but rather one which impacts the users experience by way of presentation. A styling bug had led to the title of the particular song breaking the layout of the element.

The other part of the project, namely the piece which enables the local user to stream to Cosm, raised much the same issues as the main project : the directions of how to use the web application were lacking, the user was unaware of why the geo location should be given permission to use location, and where to find a Cosm API key. In addition to this, it was noted that the errors in this sub project were unclear and the respondent needed to ask how to hook up hardware to the web application.

To remedy this, in much a similar fashion as the main sonification project, text will be included in the load up screen with the API verification. This paragraph will go in to detail as to how to obtain a Cosm API key, how to hook hardware up to the web application and briefly explaining why the web application needs the users' geolocation.

### *Project Shortcomings*

Exactly specifying how the project fails is a difficult matter; there is a fine line between asking how the project could have been expanded given more time and classifying these as failure, and where the project has obvious shortcomings in the time allotted for a basic completion, or elements that have not been considered. The distinction is an important one to make, as otherwise potential extensions are falsely classified, leaving the project to seem either incomplete or a failure.

Despite the otherwise successful aspects of the project technically, there was one main error which was unable to be fixed at the time of writing. Owing to the API request rate limitations of the Cosm API, proper real time data resolution is an issue. While 100 requests over one minute, or 300 requests over 3 minutes <sup>48</sup> seems like a fair amount, it causes data changes of less than 0.6 seconds not to be registered by any software using a standard Cosm API key. This has a direct result that the fluidity evident in the graphing section of the sonification project is, in all actuality, a way to divert from the data being received from Cosm not being properly real time to a useful data resolution. In addition to this, and also compounding the issue, is the server sent responses from the Cosm server. The content of the responses is not the source of the issue, but rather both the order in which they are sent and the timing in which they are sent leads to cause an issue. The requests sent to the server are responded to in roughly three second batches, returning a batch of responses equal to the queries sent in that time period. In laymans terms, instead of requests being returned every ~600 ms as sent by the application,

---

<sup>48</sup> <http://cosm.com/docs/v2/#rate-limit>

queries are returned in batches of ~5 every three seconds, sometimes out of order, owing to AJAX requests being asynchronous processes.

One of the other main shortcomings has been not being able to complete a full usability study on the project to verify my design choices based on my own opinions and personal sensibilities. This means that, potentially, the design of the project is heavily biased towards the demographic that I find myself in. Despite the lacking usability studies however, one extremely informal interview was able to happen with the responses being recorded. As a result, there were a few certifiable insights in regards to usability; only those which could be gleaned by common sense from a third party. However, these insights cannot be wholly substantiated owing to the lack of supporting evidence, which would come in the form of a full out usability study, where participants would be given questionnaires about both the design and the end functionality of the project. This testing would ideally occur in separate stages as the development of the project carries through, along with A/B testing for any major design choices.

## **Conclusions and Future Work**

### *How successful was my work?*

While considering how well I achieved the goals I set out to myself, I realise that most of the goals I set myself are goals I achieved, however falling short on a few ideal scenario achievements. Not having completed the stretch goals of this project, however, are not enough to categorise my project as a failure, as the main projects created are still feature complete and function as originally intended. That is to say, the projects function as intended, but are lacking a couple of features which might have made the user experience more enjoyable.

The original goal of the larger part was writing software which enables users to stream information from Cosm, and create sounds with it was a success, with a couple of additional features beyond the strict basics.

The smaller part was intended to enable the end user to stream to Cosm with the barest of technical hardware; namely a keyboard (or a Makey Makey), to attempt to lower the barrier to entry to physical hardware hacking for people with less technical knowledge. To this end, I feel that this part of the project was successful, as it makes it possible to create a very variable feed which is easy to create and use.

The web capability also extends the netChimes project, which has proposed the methods of communication of physical hardware as a basis to re-create wind chimes using http requests and Cosm as a proxy. This extension makes sheer hardware unnecessary for listening to a netChime feed from other people in other countries, and also makes it possible to listen to numerous netChime feeds at once and turn them into sounds which recreate the sound of wind chimes without the need for physical wind chimes.

The extension of creating an easy way to stream to Cosm and join the 'Internet of things' extends the netChimes project as well, allowing more people to take part and, again, lowering the barrier of entry to hardware hacking. The concept of lowering a barrier to entry is something that the Makey Makey excels at, as it runs on the underlying principle of "Attach point A to point B", which produces a binary signal when the resulting circuit (created using alligator clips and little else) is completed.

In addition, the end projects also have other uses than just being able to join in and use an open source project. The web based software, thanks to the ubiquity of machines which support Javascript and

HTML5, means that anybody with such a device (from Android to laptop computer) is able to monitor information of their own choosing. This does not only mean that the user can monitor other people's feeds, but also that they can set up hardware to monitor certain variables (such as the heat of a room, or the levels of carbon monoxide in a house) and trigger an audio alert once the value reaches a certain point.

The project has also expanded my knowledge of web technologies, including HTML5 audio and HTTP response headers, among many other things, such as running concurrent timed threads of a program and communication across these separate threads. As most of the mapping project runs on three separate recurring threads, communication between the three was vitally important to facilitate the application's abilities to map the feeds to a map at a comparatively regular pace, graph the data values and create music at regulated intervals. Also, learning to render individual and groups of points to a map was a new concept, largely relying on populating layers of a map and then clearing the map to update the layers with new information.

*How much of what you originally set out to achieve did you achieve?*

When counting how many features I originally intended to complete versus how many features are in the final product, the numbers are close to matching. The project demonstrates the core group of features that were intended for the project to function, including the ability to select streams from COSM, select individual elements of the stream, review the changes of the values of the stream, sonify said values with diverse criteria and selecting sounds at will. To complement the features, there are visual components which enable the user to review the feeds being streamed to the user in the form of a map plot, and additionally in the form of a graph of the value over the last two minutes.

The only features that did not get completed were those which were a strict bonus to the functionality, such as allowing the user to upload sounds and writing code to handle individual user profiles for extended customizability.

The streaming section of the web application functions both as desired and as intended; it fulfils the originally intended feature list of being able to plug a Makey Makey in to the computer, opening the webpage and streaming data. Additionally, the user is able to add or remove events to stream on, allowing for full customizability.

#### *Future Work*

If I had had more time to flesh out my project, an additional feature that would have been included would have been the ability to upload audio files from the end user to the server and store to constantly add to the collection of audio pre-defined and provided.

This would have been accomplished by using either a database backend such as MySQL, or keeping the organisational structure as it is in a JSON file.

The next step after this would have been to create individual user profiles accessible with a username and password server side authentication process, also using a database system such as MySQL (for the scaling of the end product in regards to access speeds) which would keep a record of the the sounds that the user has selected to use with the program. In addition to this, it would also store their recently searched selections for reference, and would add the ability to save sound profiles, as well as including the ability to upload the sounds to a server space.

Another interesting feature to add would be allowing the user more control on the sounds, adding

functionality which would make it programmable like a drum machine, such that the end user is able to choose where in the bar the note falls and is subsequently played. This would involve adding counters in to the thread governing the audio streams, as well as modifying the functions responsible for playing individual audio samples to enable the resulting piece of software to play sounds on particular beats of a bar.

However, this would not be a simple modular code addition; the organisation of the sounds would need to be overhauled completely, owing to the sheer quantity potentially overpopulating the list of sounds the user can select, making for an unwieldy selection process. Potentially, this would be mitigated by including an additional panel, which would serve to display the sounds uploaded by the user and the standard selectable sounds grouped by folder, and would allow the user to cycle through all sounds and preview, add or subtract them.

## Bibliography

"A programmer's first stop.", <http://stackoverflow.com>

"chirp.io - Let's teach the machines to sing." 2007. 17 Mar. 2013 <<http://chirp.io/>>

"Colour Meaning", <http://www.color-wheel-pro.com/color-meaning.html>

Cosm documentation, <http://cosm.com/docs/v2/>

Geo-JSON Specification, <http://www.geojson.org/geojson-spec.html>

Highcharts.js Documentation, <http://api.highcharts.com/>

HTML5 Audio Tags,

[www.position-absolute.com/articles/introduction-to-the-html5-audio-tag-javascript-manipulation](http://www.position-absolute.com/articles/introduction-to-the-html5-audio-tag-javascript-manipulation)

Javascript Documentation, <https://developer.mozilla.org/en-US/docs>

Kessous, Loic et al. "Real-time sonification of physiological data in an artistic performance context." *Proceedings of the 14th international conference on auditory display* 2008.

Mapbox Documentation, <http://mapbox.com/mapbox.js/api/v0.6.7/>

Mapbox Examples, <http://mapbox.com/mapbox.js/example/>

Roginska, A. "MONITORING REAL-TIME DATA: A SONIFICATION ... - MIT." 2009.  
<<http://www.mit.edu/~kimo/publications/sonification/icad2006.pdf>>

"Rules of Go - Wikipedia, the free encyclopedia." 2005. 17 Mar. 2013  
<[http://en.wikipedia.org/wiki/Rules\\_of\\_Go](http://en.wikipedia.org/wiki/Rules_of_Go)>

"Rounded Corners, and why they are here to stay.", <http://designmodo.com/rounded-corners/>

"ScienceByEar - Alberto de Campo." 2012. 17 Mar. 2013  
<[http://albertodecampo.net/uploads/ScienceByEar\\_diss\\_deCampo.pdf](http://albertodecampo.net/uploads/ScienceByEar_diss_deCampo.pdf)>

"Sonification of Go by Peter Tigges on SoundCloud - Hear the world's ..." 2012. 17 Mar. 2013  
<<http://soundcloud.com/peter-tigges/sonification-of-go>>

"The first Higgs boson data sonification! « LHC Open Symphony." 2012. 17 Mar. 2013  
<<http://lhcopensymphony.wordpress.com/the-first-higgs-boson-data-sonifcation/>>

Various documenation, <http://www.w3schools.com>

"Welcome | International Community for Auditory Display." 2007. 17 Mar. 2013  
<<http://www.icad.org/home>>