Module8: an open-source, modular, audio web-application

Ian Grant

MA in Creative Music Technologies

National University of Ireland, Maynooth

August 2013

The aims of this project are to provide an open-source, easily-extensible, modular alternative to costly Digital Audio Workstations. Due to the emergence of the Web Audio API for JavaScript and the many inherent conveniences of basing this project on the web, this project will be a web application. Thanks to the convenience of the faust2webaudio compiler the application is set up to convert the users' Faust creations into modules to be used in the application.

This report will take a look at the languages used in the writing of this application and the Faust language which the application works alongside. The implementation will be examined for the different features and functions of the application and their usage will be outlined. Finally, there will be an account of any known issues or deficiencies of the application in its current state and a discussion of proposed features to add.

Table of Contents

Acknowledgements:

This project would not have been possible without the input and support of the staff and postgraduate studentship of the National University of Ireland, Maynooth. Special thanks to Dr. Victor Lazzarini for his supervision and assistance, for his wealth of knowledge and for creating the opportunity to attend a course like this in Ireland.

I would also like to thank all of the developers who have worked and continue to work on the software tools that allow these kinds of projects to happen and the people who operate and contribute to codecademy.com for helping me to take advantage of those utilities.

Finally I would like to thank my friends, family and Enid Hutchins for never hesitating to drop other engagements to come rushing to my aid whenever I needed it most.

1. Languages

This project is a web application, hereafter referred to as a web-app, and as such is written in the preeminent web development languages: HTML5, CSS and JavaScript. The utility of these languages is extended by the use of some Application Programming Interfaces(APIs) including the Web Audio API, jQuery and jQuery UI. The web-app is designed to be easily extended by modules written in FAUST code and this is the basis for most of the signal processing done in the web-app. Following is a brief overview of each of the languages used.

1.1. FAUST

FAUST, a contraction of functional audio stream, is a functional programming language for real-time signal processing and synthesis. It was written in C++ by Yann Orlarey, Dominique Fober and Stéphane Letz. FAUST is developed by GRAME, Centre National de Création Musicale. Though it is a text-based language, it uses block-diagrams to display signal flow. Figure 1 shows a block-diagram of the implementation of the bi-quadratic equation on delay lines which is used to create audio filters in many FAUST examples.

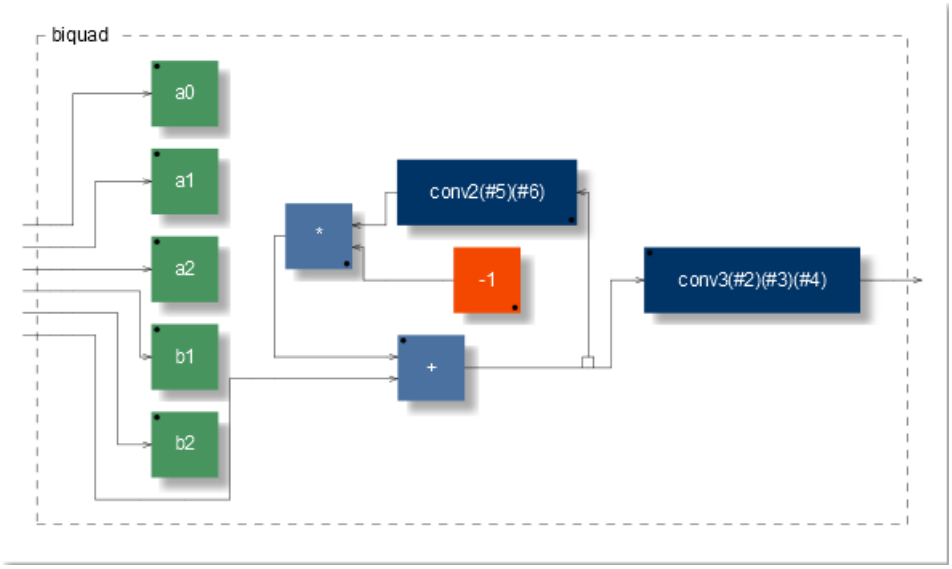


Figure 1: FAUST block-diagram of bi-quadratic filter implementation.

FAUST is compatible with other languages as the programs are fully compiled, not interpreted, into highly efficient C++. This allows FAUST programs to be compiled into several other languages efficiently, such as the Web Audio API. A high level off cross-compatibility is achieved because FAUST compiles the mathematical operation of the program at the sample level and so abstracts the program from its function as little as possible.[1] Though this has many advantages the Web Audio API has problems interpreting the oscillators and noise generators.

1.2. HTML5

The fifth revision of the Hypertext Mark-up Language by the World Wide Web Consortium(W3C) and the Web Hypertext Application Technology Working Group(WHATWG) is the first upgrade since HTML 4.01 released in 1999. While HTML is the ubiquitous mark-up language for web design HTML5 is not yet the official W3C standard and no browsers have full HTML5 support. Though it is still a work in progress it is supported by all of the major browsers and more HTML5 features are supported as browser updates are released.

Some of the core philosophies of HTML5 in particular make this project possible. There is a focus on having the language be device independent which makes application development much simpler. There is also an aim to reduce the reliance on external plug-ins like Adobe Flash which is notorious for its instability and is more device dependent. HTML5 offers full CSS3 support and access to local storage.[2]

1.3. CSS

Cascading Style Sheets are a method for changing the appearance of web pages. They are an

1 http://faust.grame.fr/index.php/documentation/what-faust

2 http://www.w3schools.com/html/html5_intro.asp

efficient method of abstracting the style attributes of HTML elements using selectors to target different sets of tags with high precision. Due to the cascading aspect of CSS the styles can inherit from each other with more specific selectors taking precedence over others. This way a HTML page can be styled by multiple different style sheets at once. CSS is also developed by W3C and the most recent version to be supported in most browsers is CSS3.[3] This is a huge step forward in allowing animations without external plug-ins and is part of the reason more work is being done on the audio aspect of web design to allow CSS and web audio to take the place of Flash and similar externals.

1.4. JavaScript

Designed by Brendan Eich and developed by Netscape Communications Corporation, JavaScript is an object-oriented scripting language. As a scripting language it is lightweight and is much easier to learn than the lower-level languages which influenced it such as C and Java. JavaScript can be inserted into HTML code and can be run on all modern browsers. This has made JavaScript one of the most widely used programming languages in the world. With JavaScript you can perform logical operations and add functions to be called on certain events in web pages. It can access the content of the HTML through the Document Object Model.[4] JavaScript operates primarily on the client-side, though server-side applications do exist.[5]

1.5. jQuery

Not a language in itself, jQuery is a library written in JavaScript by John Resig. It is a very lightweight library designed to simplify client-side scripting of HTML.[6] It allows the user to write

3 http://www.w3schools.com/css/css_intro.asp

4 http://www.w3schools.com/js/js_intro.asp

5 http://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance#awesm=~ofQtatRhnOgmWs

6 http://www.codeproject.com/Articles/157446/What-is-jQuery-and-How-to-Start-using-jQuery

less code to do more. One of the convenient advantages of jQuery over plain JavaScript it the support of CSS selectors for targeting HTML elements. This allows the user to apply interactivity to a page as efficiently as they can implement styles. jQuery supports all selectors that are supported by CSS3.[7]

For an example of how much neater and easier jQuery can make scripting see this comparison of script to change, remove or add classes in jQuery, JavaScript and legacy JavaScript made by Jeffrey Way:

jQuery:

```
$('#box').addClass('wrap');
$('#box').removeClass('wrap');
$('#box').toggleClass('wrap');
```

JavaScript:

```
var container = document.querySelector('#box');
container.classList.add('wrap');
container.classList.remove('wrap');
container.classList.toggle('wrap');
```

Legacy JavaScript:

```
var box = document.getElementById('box'),

    hasClass = function (el, cl) {
        var regex = new RegExp('(?:\\s|^)' + cl + '(?:\\s|$)');
        return !!el.className.match(regex);
    },

    addClass = function (el, cl) {
        el.className += ' ' + cl;
    },

    removeClass = function (el, cl) {
        var regex = new RegExp('(?:\\s|^)' + cl + '(?:\\s|$)');
        el.className = el.className.replace(regex, ' ');
    },

    toggleClass = function (el, cl) {
        hasClass(el, cl) ? removeClass(el, cl) : addClass(el, cl);

    };

addClass(box, 'drago');
removeClass(box, 'drago');
toggleClass(box, 'drago'); // if the element does not have a class of 'drago',[8]
```

7 http://jquery.com/

8 http://net.tutsplus.com/tutorials/javascript-ajax/from-jquery-to-javascript-a-reference/

Way describes the 'legacy' JavaScript as the “make browsers happy” version and the standard JavaScript as the modern “cool kids” method. The advantages of jQuery here are self evident.

The web-app also makes use of jQuery UI, another JavaScript library built on top of the jQuery library as part of the jQuery Project. It offers a selection of user interface interactions and effects which make it quick and efficient to implement otherwise complex interactivity.[9]

1.6. Web Audio API

The Web Audio application programming interface is a high-level JavaScript API which reduces the reliance on external plug-ins for online audio applications. It is developed by W3C. It provides methods through which user defined objects called AudioNodes can be connected in a signal chain. The actual signal processing is optimised to run on the lower-level implementation but does support processing directly at the JavaScript level.[10]

The API is still in development and is restricted by many limitations. Currently the only major browsers that support Web Audio(via webkit) by default are Google Chrome, Opera and Safari.[11] Mozilla's Firefox Aurora supports Web Audio[12] but at time of writing this version is in pre-Beta.[13]

There is currently no support for MIDI in the API but a Web MIDI API is in development by W3C.[14] The API currently has access to audio inputs and the nature of its implementation wreaks

9 http://jqueryui.com/about/

10 http://www.w3.org/TR/2012/WD-webaudio-20120315/

11 http://caniuse.com/audio-api

12 https://wiki.mozilla.org/WebAudio_API_Rollout_Status

13 http://www.mozilla.org/en-US/firefox/aurora/

14 http://www.w3.org/TR/2012/WD-webmidi-20121213/

havoc with the DSP operations compiled by faust2webaudio for oscillators and noise generators.[15] It also seems to have trouble implementing FAUST's bi-quadratic filter equations.

2. Usage

For cases where it may not yet be clear how the application can be utilised, a brief manual on the usage of the project follows.

2.1. Audio Looper

The bar across the top of the page is used to load samples and loop them. To do this just enter the URL of the desired audio file into the text input and click the 'Play' button. Audio format support varies depending on the browser used. Google Chrome(recommended) supports WAV, OCM, WebM, Ogg, Vorbis, MP4, MP3 and AAC.[16] For a complete account of which formats are supported on the major browsers see Table 1 attached in the appendix.

The file can take some time to load so it is best to be patient as clicking the play button multiple times can result in the audio file repeatedly cutting back to the start of the file, taking quite some time and most likely being undesirable. To stop the file playback click the 'Stop' button. To change the destination of the audio stream, choose an option from the 'Connect to:' drop-down menu. Once a selection is made you must click the connect button to reroute the audio. The drop-down menu is populated dynamically as modules which accept input are added.

15 http://faust.grame.fr/index.php/7-news/73-faust-web-art

16 https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats

2.2. Signal Generators

To use the signal generators, select the signal generators heading in the menu at the left of the page. This opens up a selection of signal generators featuring varying types of oscillators and physical string models. The oscillators highlighted in orange are all from FAUST but do not work with the Web Audio API; they are included in the hope that these issues in the API will be resolved. These oscillators are not only broken but they have adverse effects on any other modules and so should be avoided until fixed.

To create a signal generating module, click the name of the module you wish to use. Due to the lack of MIDI support, the practicality of the signal generators is limited; they exist here mostly for example purposes. The choice of working modules offers an oscillator with slider control over frequency and volume and a selection of four standard wave shapes. There are two instances of the Karplus-Strong physical model of a string, one is basic and the other uses an array of thirty-two resonators. These offer control over various aspects of the algorithm such as the excitation signal and the delay lines used to resonate. The harp module is another string model with a more hands on approach to activation.

2.3. Effects

Effects can be created in the same way as signal generators by expanding the effects heading and selecting a module from the list offered. Since there is less reliance on MIDI for these effects to employed they are the most practical aspect of the program in its current form. There are a number of effects including three different reverb options, a multi-band filter, a more lightweight single-band filter and a pitch shifter. These all work well but too many large modules running at once will slow the processing. The single-band filter can be used to take the place of filters such as low-pass, high-pass, all-pass, etc., until a better implementation of those can be made.

2.4. Utilities

This section is for mixing and routing tools that do not necessarily fall into the category of effects. Due to the way audio channels are implemented in the application, many of these modules do not yet work as desired. Unique among the standard modules, the capture module takes its input from the master mixer strip.

The master strip at the right hand side of the page controls the output volume of the whole application. It usually only takes input from other modules and sends only to the output but when a capture module is created the master strip's output is sent to both the output and the capture module. The module captures any audio it receives while the capture button is held down and begins looping that audio once the button is released.

2.5. Patchers

We now know how signal flows from the master strip to the capture module and how to reroute the output of the sample player. The rest of the signal flow direction is set under the patchers heading in the menu. When each module is added a corresponding drop-down menu is added to the patching section. These options are used to choose where to route the modules' outputs. All modules by default are routed to the output and start off with the option between routing to the output or disconnecting.

As modules such as effects and utilities are created there are corresponding options added to all of the drop-down patching menus. Signal generators are not added to the list of possible modules to route to because they have no methods for input. Unlike the audio file routing at the top of the page there is no need to hit a connect button, routing is done as soon as a selection is made from the drop-down patching menu.

2.6. DIY

The do it yourself tab in the menu offers a method of dynamically adding your own modules to the application on-the-fly. Modules are added by selecting the heading under which the module will fit, this also effects the patching as effect and utilities are implemented differently to signal generators. Then a type of code is chosen either 'Faust' or 'JS'. The 'JS' option is for adding your own JavaScript or related libraries. The 'Faust' option is more specific.

To add a module you have designed in FAUST you must first install the Faust2 branch of FAUST found at http://sourceforge.net/p/faudiostream/code/ci/faust2/tree/ . Save your module as a dsp file and call faust2webaudio on your module in the terminal inside the examples directory of the Faust2 install as in:

faust2webaudio mymodule.dsp

This will create a file called mymodule.html. Open this file in a text editor and copy the contents. Paste this code into the text box in the DIY panel. Click the 'Add' button and then switch to the panel that corresponds to the type of module you have created. Under the heading you will find a button labelled '-User Defined'. This will create the module you designed and the appropriate signal routing in the patching panel.

2.7. A Caution

Figure 2: All is well.

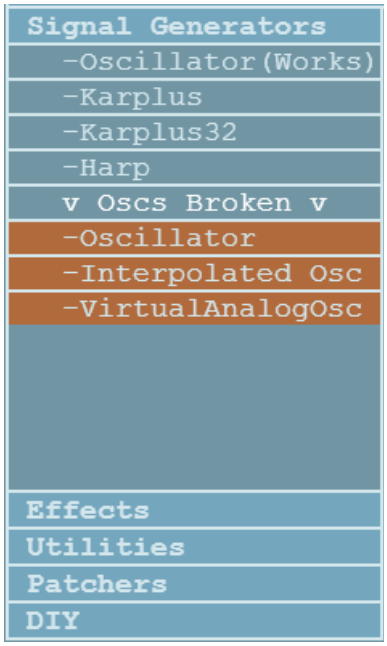
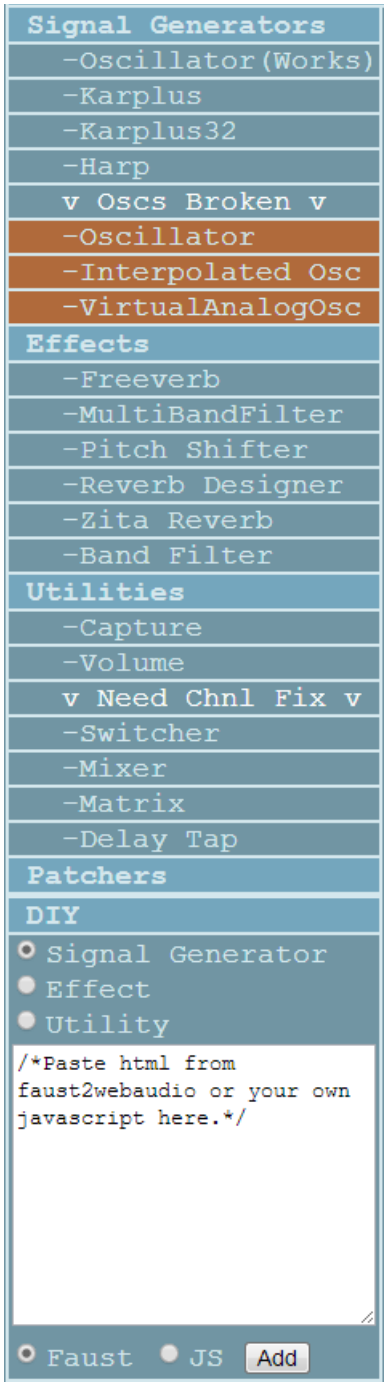


Figure 3: jQueryUI failed.



If your page resembles Figure 3 as opposed to Figure 2 this means that the jQueryUI library has not loaded properly. For issues like this or where the audio fails or the page only loads bare HTML then the user should first attempt to refresh the page. If the problem persists try launching the alternative version of the html file from the folder it came in with the JavaScript and CSS files in the same directory. If this does not resolve the issue ensure you are running the page on a browser version which supports Web Audio as listed in the appendix in Table 1. If issues continue feel free to contact me at granti@tcd.ie.

3. Implementation

This section will delve into the construction of the application in detail expanding into the practical aspect of the languages already discussed. The source code will be disseminated with examples given to demonstrate the architecture of the project.

3.1. Layout & Styling

Thanks to the adaptability of CSS it is practical to change the styling of a page as an afterthought without disrupting the functionality of the web page. Therefore in an early, proof-of-concept style program like this styling does not need to be a high priority; it can always be changed afterwards quite simply. Even though the styling was not the focus of this project it was one of the first things to be considered as it was the conceptual basis of the user interface.

The layout is outlined in Figure 4 with colour coded divisions, styled by their id attributes.

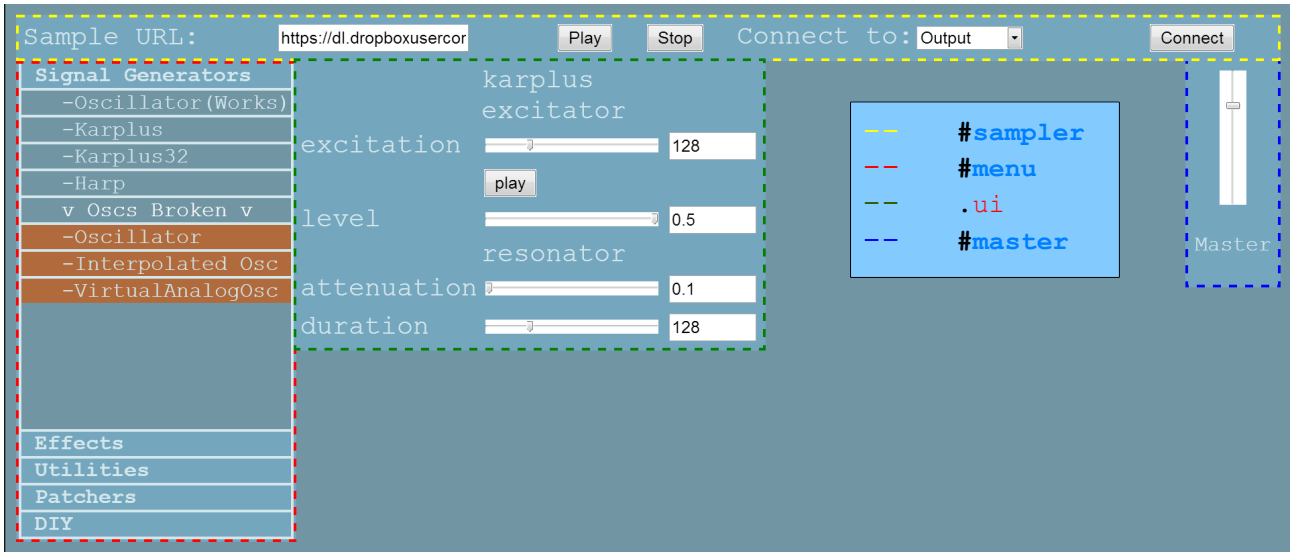


Figure 4: Layout

The `#sampler` bar at the top of the page is positioned relative to the document in the default position given a width of 100%. This way it stretches to fit the window size of whatever browser is used. The menu and ui elements are given the rule `float:left;` to make them align to the left hand side of the page in the order they are designated in the HTML5 document. The `#master` module is told to `float:right;` which allows room for creation of many modules in the middle of the page.

The colour scheme was chosen from a selection of colour palettes from colorcombos.com. The blues are not straining on the eyes and have seen wide success in modern web design and marketing.[17] The courier font was chosen to represent the open-source aspect of the project. Neither

17 http://www.colorcombos.com/color-schemes/428/ColorCombo428.html

of these decisions are final.

The majority of the rest of the CSS used is to line things up neatly and to make the borders appear in proportion. The `#gain` slider in the `#master` division is a special case though as it is vertical. This requires the `-webkit-appearance: slider-vertical;` rule.[18] This means that the slider will only appear vertical in browsers which support the webkit prefix. This is not a huge issue at present, though, as webkit is also currently the standard for implementing Web Audio. The `#FaustMeta` division is set to `display:none;` to help declutter the user interface to simplify the appearance.

Aside from CSS some of the interactivity and styling of the page comes from some jQueryUI in the linked script file:

```
$(document).ready(function() {
    $("#menu").accordion({collapsible: false, active: false});
    $('#FaustUI').sortable();
});
```

This tells the application to wait until the document is loaded then find the HTML element with the id `#menu` and give it the accordion functionality.[19] Then it says to make items in the division with the id `#FaustUI` sortable.[20] This is what makes the menu headings expand and contract to show one category at a time and allows the modules to be sorted by clicking and dragging.

3.2. Audio Looper

The ever-present bar at the top of the page used for sampling from a loaded URL is a vestige of the faust2webaudio architecture where the sample loading is used as an alternative to real-time input. In future versions adding this to the list of utilities instead will be entertained.

In its current state the audio playback is a re-writing of the faust2webaudio default in

18 http://jsfiddle.net/ArondeParon/SPjsf/15/

19 http://jqueryui.com/accordion/#default

20 http://jqueryui.com/sortable/

jQuery. The most notable difference in the implementation here is the audio routing. This function is called when the connect button is clicked:

```
$('#connect').click(function(){
        var out = $("#sample option:selected").val();
        patcher(source, out);
});
```

This is used to connect the output to the selection in the drop-down menu. Similar code is implemented for the play button and due to the construction of the `patcher()` function it changes the operation slightly from the default faust2webaudio version. This disconnects the output before starting another instance which prevents multiple layers of the sample being played asynchronously with the stop button only functioning to stop one of the instances.

3.3. Patcher

This will take a closer look at the `patcher()` function called in the above example. The `patcher()` accepts two parameters as input. The first is the audioNode object the output of which is to be connected. The second is a string, obtained from the value of a drop-down menu selection which is tested against a number of possible cases in a large swich statement. Within these cases the signal is connected to the audioNode corresponding with the string given as the second parameter:

```
function patcher(signal, destination){
        signal.disconnect(0);
        switch(destination){
            case "Freeverb":
                signal.connect(freeverby);
                break;
            case "output":
                signal.connect(gainNode);
                ...  etc.
```

3.4. Master Gain

As can be seen in the above extract from the code, the output option actually connects to a gain node. This takes input from the vertical gain slider on the right side of the page and uses it to scale

the amplitude of all signals before they reach the actual output.

```
gainNode = context.createGainNode();
gainNode.connect(context.destination);
$('#gain').change(function(){
    gainNode.gain.value = this.value;
});
```

This takes advantage of the `createGainNode()` function included in the Web Audio API.[21] The output is referenced by `context.destination` and the `#gain` slider is monitored for changes and used to update the gain value.

3.5. Signal Genrators

The signal generating modules are generated by a function which takes two parameters: a string to be used as a unique name and a function used to create the audioNode, usually used to reference the *create_FOO* methods created by faust2webaudio.[22] This function does all the work of creating a new signal generator:

```
function signalGenerator(name, creator){
    ui = new JUI(document.getElementById("FaustUI"));
    objName = new creator(context, ui, meta, 4096);
    objName.connect(gainNode);
    var options = $('#sample').html();
    $('#patchers').append('<div>'+name+'=><select id="'+name+'"
class="patch">'+options+'</select></div>');
    $('#'+name).change(function(){
        var out =  $('#'+name).val();
        patcher(objName, out);
    });
}
```

Firstly the user interface is created, this is essentially a blank canvas at this point. Then the *create_FOO* function parameter is called to create the audioNode and fill the user inteface with content. The audioNode is connected to the master gain control. The current content of the drop-down menu used to route the audio looper's output is copied to a local variable. This is then used to append a patching drop-down menu to the patching section of the application. The audio looper's

21 http://creativejs.com/resources/web-audio-api-getting-started/

22 http://faust.grame.fr/index.php/7-news/73-faust-web-art

output menu serves as a template for all subsequently created patchers. Because this template is updated every time any potential destination module is added, the new patchers are created with all available options already in their menu. An event listener is then added which calls the `patcher()` function whenever the selection in the menu is changed.

3.6. Working Oscillator

Since the oscillators generated by faust2webaudio are broken there is a working alternative included. This is unique among signal generators in that it is not built on the faust2webaudio architecture but instead deals only with the Web Audio API. This serves to demonstrate the way in which modules can be added without using the `signalGenerator()` function and can be written entirely in JavaScript.

Essentially an expansion upon the methods used to implement the master gain slider but wrapped in a click event handler, there is one notable difference: the user interface had to be generated each time the handler was called. This is achieved with a reasonably unwieldy function call:

```
$('#FaustUI').append('<table class="ui"><tr><th
colspan="2">Oscillator</th></tr><tr><td>Freq</td><td><input id="freq"
type="range" min="20" max="2200" step="1"
value="440"></td></tr><tr><td>Vol</td><td><input id="vol" type="range" min="0"
max="1" step="0.01" value="0.5"></td></tr><tr><td>Wave</td><td><select
id="wave"><option value="0" selected>Sine</option><option
value="1">Square</option><option value="2">Sawtooth</option><option
value="3">Triangle</option></select></td></tr></table>');
```

There are event listeners monitoring both sliders for change to update the values for volume and frequency. The drop-down menu is monitored in the same way as the patching menu. The oscillator itself is, like the gainNode, from a built in Web Audio function.[23]

23 http://www.phpied.com/webaudio-oscillator-in-js/

3.7. Effects & Utilities

The effects and utility modules call a function called `effect()` which operates in a very similar fasion to `signalGenerator()`. The difference between the two being that effects and utilities must be added to the patching drop-down menus to be able to receive input from other modules. To add the module as an option on all existing patching menus this snippet of code is added to the function:

```
$('.patch:not(#'+name+')').prepend('<option
value="'+name+'">'+name+'</option>');
```

This uses the `:not` selector to add this module as an option on all patching menus except for the module's own menu. This is to avoid explosive feedback though an option may be added to offer recursion in later versions.

Where the `signalGenerator()` function returns nothing the `effect()` function returns the audioNode object so that it can be used in the `switch()` statement in the `patcher()` function. The need to have the object already listed in the `switch()` statement, as opposed to being added dynamically, imposes limitations on the way user defined modules are implemented.

3.8. User Defined Modules

The 'JS' option for inputing user defined modules simply appends the text input to inside script tags to the document head:

```
$('<script>'+input+'</'+'script').appendTo(document.head);
```

This poses clear security issues though all user input carries at least some risk of breaking the program, whether intentional or otherwise. These issues will need to be addressed before an attempted release but will serve the purpose of this proof-of-concept.

The 'Faust' method of input is far more involved. This operation begins by logging the module type selection in a local variable. Next it utilises the `indexOf()` and `slice()` string functions to isolate the code necessary to execute the *`create_FOO`* function generated by

faust2webaudio and also the name of the function call itself and calls the `eval()` function on that string to render it useful as a function to pass to the module constructors:

```
var type = $('#sort:checked').val();
var start = input.indexOf("window.onload = init;")+21;
var end = input.indexOf("function loadSample(url)");
var guts = input.slice(start,end);
start = guts.indexOf("create_");
end = guts.length;
var creator = guts.slice(start,end);
end = start+creator.indexOf("(");
creator = guts.slice(start,end);
var create = eval(creator);
```

Once the input is parsed the `type` variable is passed to a **`switch()`** statement. This decides which heading the button to create the module will be appended to and whether to call the `effect()` or `signalGenerator()` functions using the `create` variable as a parameter.

```
$('#usrdef').click(function(){
    userdefined = effect("UserDefined",create);
});
```

Unfortunately, as can be seen above, the name string passed to the `effect()` function is not defined by the user but predefined as “UserDefined”. This is because the **`switch()`** statement in the `patcher()` function needs to know what to expect in terms of the value attribute it receives and the audioNode to connect to. This could be overcome by adding some more parameters to the `effect()` function's operation but until the issues with channels are resolved it would make little difference.

4. Known Issues

4.1. Web Audio/FAUST

In certain cases FAUST code, once compiled into JavaScript, is interpreted badly by Web Audio. The FAUST website lists known faults in how the oscillators and noise generators operate.[24] I have

24 http://faust.grame.fr/index.php/7-news/73-faust-web-art

found many filters which malfunction in a similar fashion. Including, from the FAUST examples folder, LPF.dsp, HPF.dsp, APF.dsp, BPF.dsp. These are all built in a similar fashion: by passing coefficients to a bi-quadratic process. I am not sure why these do not work properly but it appears to be an issue at the sample-to-sample, digital signal processing level. Hopefully these will be fixed as Web Audio continues to be developed.

4.2. Channels/Duplication

In its current state the project does not handle multiple channels properly and due to the naming of the audioNodes used in patching the modules behave improperly when there is more than one module of the same type. This effect also impairs the use of multiple user defined modules whether they are identical or different.

There is a possible fix for the patching issue of multiple different user defined modules by using an array of audioNodes with an index that is incremented each time a module is created. The issue arises when the modules are identical, as is the case with the built in example modules, there is cross-talk between the user interface elements. This issue requires an almost total rewrite of the user interface building functions from the faust.js file that is linked to by default in faust2webaudio HTML file.

5. Features to be Added

As this project is simply a proof-of-concept it is still quite far from being released. While many key features must await more complete support for Web Audio, Web MIDI and more powerful user interface tools, there are many improvements that can be added in the meantime. This section offers a brief account of these planned improvements.

5.1. Persistence

An essential feature to make this project practical is the ability to save not only the audio output to a file but also the content of users' designs. The audio could provisionally be bounced using a similar implementation to the capture node. This would be very time consuming and would need to be eventually be replaced by a bouncing method that runs above the sampling rate.

The most suitable method for keeping page content persistent would be to store the content server-side to be optionally made accessible to the public for an open-source approach to composition. This implementation would require an account based system so an alternative method of downloading a file as a method of saving could be offered to those without accounts. There should be additional pages added to create an entire website with a portal for sharing creations, a page for documentation, attributions, contact information, etc., all accessible through a navigation bar. A section to directly upload FAUST's .dsp files to and have them added as a module would be another welcomed improvement. Full VST support could hugely improve the project's extensibility also.

The obstacles to adding these features at this time include that it would be helpful to have Web MIDI support in a major browser before trying to implement such a system. The main obstacle is my own knowledge of server-side scripting, this is being remedied. Another obstacle is that it would be preferable to put the navigation bar where the audio looper is now: at the top of the page.

5.2. Modules

For this style of project there can never be too many modules and improvements made. Though the list of modules to add is practically unending there are three modules which are currently a higher priority.

The first is a full MIDI sequencer to offer control over signal generators. Ideally this would

have a smooth and fluid user interface and intelligent quantisation options. An arrangement panel would be necessary for editing structures on a large scale. A pitfall to avoid would be to encourage the use one time-signature over any others. Many digital audio workstations available at present offer 4/4 time as the default template. This tends to split most electronic music into those who write only in 4/4 and those who disregard time-signatures entirely. This module would also preferably implement support for n-tuplets. A great advantage to the modular aspect of this program is we are not limited to one sequencer design and a dedicated 4/4 looping sequencer could also be included for those who want to enjoy the convenience of that. It is not limited to one prototype.

Another module that should be added is an all encompassing sample editor or at the very least replace the functionality of the FAUST vestigial audio looper at the top of the page. Considering the time it currently takes to load audio files a helpful feature to add would be a progress bar from jQueryUI. Ideally this module would have midi support and an intuitive, highly graphical, sample editing interface. The user should be able to choose different methods of pitch-shifting to map the samples to MIDI input and select different types of playback including options like sample-and-hold to build realistic recreations of acoustic instruments from a library of samples. These effects and methods might be made to be modular so that savvy users can create their own methods of implementing sampling though this would be a difficult task to moderate.

The final module that is currently high priority, and certainly easiest to implement of the three, is a modular mixing strip. This would feature a vertical gain slider, a panning slider (or preferably a knob), mute and solo buttons. The mute and solo buttons can be set up with global and local variables to act as one connected mixer no matter how many instances of the module there are. There could be different versions for panning to different numbers of channels, for example, mono, stereo, 5.1, etc. Different modules could also be offered for 3D spatialisation with head related transfer functions and other panning methods. The spatialisations and panning across more than two speakers would call for a more advanced user interface.

5.3. General Improvements

One of the first general improvements that could be made to this program is in the implementation of the patching. In its current form multiple sources can be patched to the same output but each source can only go to one module. This could be quickly remedied by a splitter module but a more user friendly solution could be devised while the issues with channels are being resolved.

Patching may retain its own drop-down menu but moving the patching menus to the modules themselves will be considered. An 'x' option to close the modules should be added. Closing would consist of disconnecting the source and freeing the memory while removing the division from the window. The challenge in doing this is uniquely identifying the correct user interface division to remove. This might be solved along with the module duplication by incrementing an index to uniquely identify the division as it is created.

The current method of organising the modules is not ideal. While making the modules sortable with jQueryUI is helpful it is still not ideal to just have every division floated to the left, especially with modules of such variety. The solution to this may be found with a deeper understanding of new element positioning grid in CSS3.

There are some quick-fixes in the code that could be implemented more efficiently to shave down the weight of the program. For example the FAUST meta-data is simply hidden but all of the HTML and JavaScript written in regards to it still exists. It runs throughout the behind the scenes implementation of the modules in the linked JavaScript file. This kind of streamlining could have a greater importance once the project is at a larger scale.

6. Conclusion

Though the work of adding modules can never be done and there are many desirable additions yet to be made to this project, it successfully conveys the essence of the concept. Here we see the basis for a modular, web-based, open-source digital audio workstation. As development continues in providing support for the methods used the project will evolve to provide an all-encompassing basis to build a free cross platform DAW. Once a suitable foundation is laid and the Faust2 branch of FAUST becomes the standard the project will have great potential for extendibility.

If this project becomes interesting enough to garner attention from open-source audio developers who wish to democratise the production of music then it can grow exponentially. The trend in audio production is moving more and more towards home studios and inexpensive software. This project may grow to be a free option for beginning musicians to learn how to use DAWs. A project like this has the potential, with a strong community behind it, to eventually go beyond being a beginner's budget option and become an accepted studio standard.

7. Appendix

Table 1: Media formats supported by the HTML audio and video elements[25]

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	3.0	3.5 (1.9.1)	9.0	10.50	3.1
`<audio>`: WAVE, PCM	(Yes)	3.5 (1.9.1)	Not supported	Not supported	3.1
`<audio>`: WebM, Vorbis	(Yes)	4.0 (2.0)	Not supported	10.60	3.1 (must be installed separately)
`<audio>`: Ogg, Vorbis	(Yes)	3.5 (1.9.1)	Not supported	10.50	3.1 (must be installed separately, e.g.XiphQT)
`<audio>`: MP4, MP3	(Yes) (Not in Chromium)	Partial (see below)	9.0	Not supported	3.1
`<audio>`: MP4, AAC	(Yes) (Main only) (Not in Chromium)	Partial (see below)	9.0	Not supported	3.1
`<audio>`: Ogg, Opus	27.0	15.0 (15.0)	?	?	?

25 https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats

<video>: WebM, VP8, Vorbis	(Yes)	4.0 (2.0)	9.0 (must be installed separately, e.g.WebM MF)	10.60	3.1 (must be installed separately, e.g.Perian)
<video>: Ogg, Theora, Vorbis	(Yes)	3.5 (1.9.1)	Not supported	10.50	3.1 (must be installed separately, e.g.XiphQT)
<video>: MP4, H.264, MP3	(Yes) (Not in Chromium)	Partial (see below)	9.0	Not supported	3.1
<video>: MP4, H.264, AAC	(Yes) (Not in Chromium)	Partial (see below)	9.0	Not supported	3.1
any other format	Not supported	Not supported	Not supported	Not supported	3.1 (plays all formats available via QuickTime)

To avoid patent issues, support for MPEG 4, H.264, MP3 and AAC is not built directly into Firefox. Instead it relies on support from the OS or hardware. Firefox supports these formats on the following platforms:

Platform	Firefox version
Windows 7+	21.0
Windows Vista	22.0
Android	20.0
Firefox OS	15.0
OS X 10.7	22.0

8. Bibliography

http://caniuse.com/audio-api

http://creativejs.com/resources/web-audio-api-getting-started/

http://faust.grame.fr/index.php/7-news/73-faust-web-art

http://faust.grame.fr/index.php/documentation/what-faust

http://jqueryui.com/about/

http://jqueryui.com/accordion/#default

http://jqueryui.com/sortable/

http://jsfiddle.net/ArondeParon/SPjsf/15/

http://net.tutsplus.com/tutorials/javascript-ajax/from-jquery-to-javascript-a-reference/

http://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance#awesm=~ofQtatRhnOgmWs

http://sourceforge.net/p/faudiostream/code/ci/faust2/tree/

http://www.codecademy.com/

http://www.codeproject.com/Articles/157446/What-is-jQuery-and-How-to-Start-using-jQuery

http://www.colorcombos.com/color-schemes/428/ColorCombo428.html

http://www.mozilla.org/en-US/firefox/aurora/

http://www.phpied.com/webaudio-oscillator-in-js/

http://www.w3.org/TR/2012/WD-webaudio-20120315/

http://www.w3.org/TR/2012/WD-webmidi-20121213/

http://www.w3schools.com/css/css_intro.asp

http://www.w3schools.com/html/html5_intro.asp

http://www.w3schools.com/js/js_intro.asp

https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats

https://wiki.mozilla.org/WebAudio_API_Rollout_Status