Digital Music Synthesis: A VST Software Instrument

Gabriel Olochwoszcz
Advisor: Dr. Sophocles J. Orfanidis

Capstone Design: Digital Signal Processing
Department of Electrical and Computer Engineering
Rutgers University
May 2012

Acknowledgements

I would like to thank Dr. Sophocles J. Orfanidis for sharing just a fraction of his wealth of digital signal processing knowledge with me as I developed this instrument. I am particularly indebted to him for his parametric filter design which allowed me to achieve the sound I was seeking.

I am also grateful for the instruction of Dr. Michael Caggiano, Dr. Jeffery Walling, Dr. Lawrence Rabiner and Dr. David Daut. My understanding of the fundamental engineering concepts that enabled the creation of this instrument is rooted in the instruction received in their classes.

Thanks also to Eric Schreier for testing the first prototype and his articulate feedback from a performer's perspective.

Completing this project required many long hours spent in front of multiple computers spread across several desks instead of with my family – an arrangement that would not have been possible without the support, patience and love of my wife, Stacey, babysitting help from my mother, Mirtha, and the minutes of joy between frantic debugging sessions shared with my son, Neil. This project is dedicated to him in the hope that my work in engineering music products will give him the tools and inspiration to develop the same love of music that brought Stacey and I together and which we continue to share.

Table of Contents

I. Introduction

Early electronic synthesizers were large, modular instruments that were impractical for everyday use. They were unstable and unreliable and were subsequently used primarily in “academic” music but gradually became popular with rock and pop musicians. Dr. Robert Moog developed the first commercially available modular synthesizer in 1964; its most popular functions were later consolidated into the (relatively) portable Minimoog. The Minimoog and its contemporaries allowed working musicians to easily utilize synths both in the studio and while on tour and became associated with British progressive rock groups such as Yes, Emerson, Lake and Palmer and Genesis as well as with German electronic music pioneers Kraftwerk. These early instruments were portable but still not very stable or reliable.

Digital synthesis techniques were developed through the 1970s that allowed for stable pitch and superior reliability but at the expense of the characteristic “warm” sound of analog instruments. Regardless, digital synthesizers went on to dominate the New Wave and synthpop sounds of the 1980s. These styles eventually evolved into electronic dance music which is now the most common use of electronic synthesizers in popular music.

Analog synthesizers have also come back into fashion recently for their “fatter” sounds compared to digital instruments. Modern analog instruments are being designed to improve the stability and reliability of old analog designs and expand their functionality to include previously unavailable interfaces and tools. For example, the Minimoog is still being made new but now with modern components and features such as MIDI control.

A relatively recent development in electronic synthesis is what has come to be referred to as “virtual analog synthesis.” Based on physical modeling, virtual analog uses DSP techniques to model the underlying circuit behavior of analog instruments. The resulting instrument maintains all of the stability of digital devices as well as capturing much of the character of analog instruments.

Increased computing power in the 1990s enabled the development of software-based audio processing on home computers and laptops. One of the leading standards for this type of processing is Steinberg's Virtual Studio Technology (VST). VST allows a performer/producer to run multiple VST plugins on a single VST host, emulating the interconnected devices of a physical studio or pedalboard with no wires. In addition to effects processing, VST software instruments have been developed using various synthesis techniques.

II. Objective

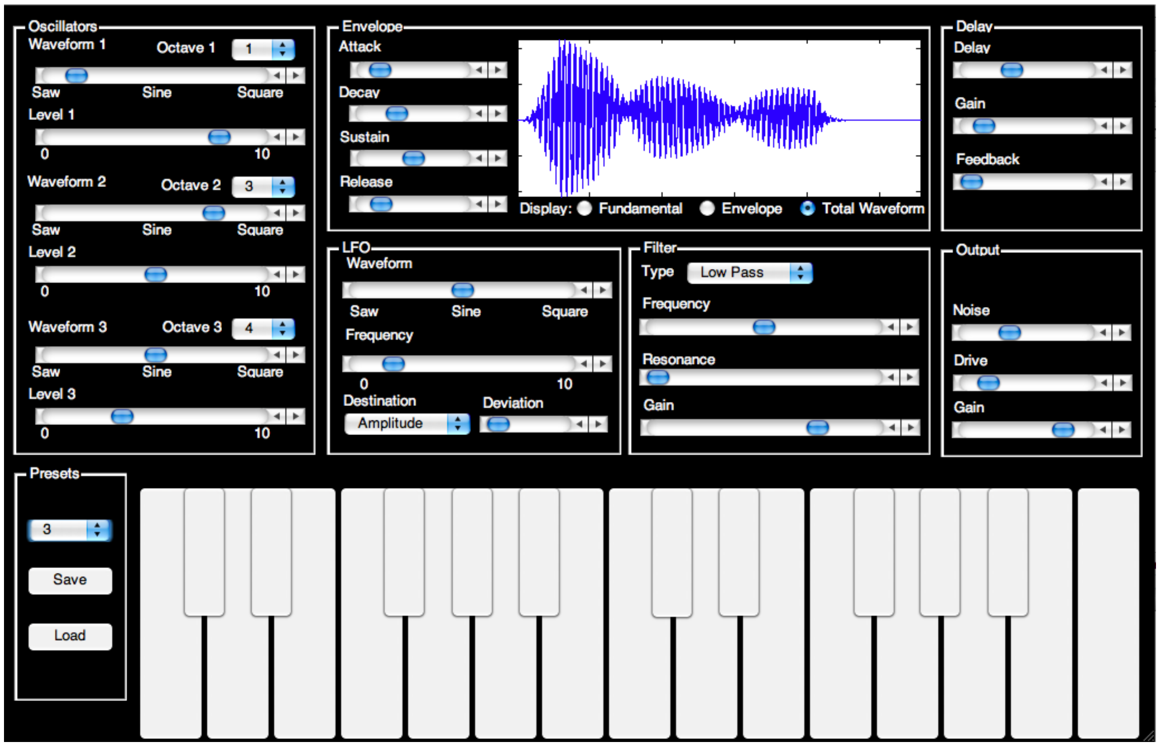
The objective for this project was to create a real-time digital instrument whose design would allow the opportunity to gain experience with industry standard music/audio processing platforms, improve algorithm design and optimization skills, and to develop C/C++ programming knowledge and ability. These objectives were met by continuing development of a previously developed MATLAB synthesizer as a VST instrument which improved computational efficiency and included additional and effects processing.

III. Development and Testing

The original proposal for this project was focused on developing techniques for use in virtual analog models of electronic music and audio circuits which would capture the process, voltage and temperature variations that contribute to analog gear's characteristic sound [4]. However, it was quickly apparent this was far too much to undertake in the available time and was better left as an area of future advanced study. Instead, the decision was made to look at other areas that were more strictly design based. Having previously designed and implemented a synthesizer in MATLAB for a previous course, pursing further development in synthesis to create a practical instrument was the best course of action.

Several platforms were considered for development of the software instrument: Apple's iOS for iPhone/iPad, dedicated DSP hardware and Steinberg's Virtual Studio Technology (VST) [6]. Experimentation with iOS programming revealed that the app design and interface programming would quickly eclipse any signal processing elements of the design. Similarly, using DSP hardware would have required managing the interface between MIDI inputs and audio outputs, again obscuring the intended signal processing focus.

VST, on the other hand, is a cross-platform (Windows and Mac OSX) technology which required no specialized hardware, was freely available – as were hosts – and was compatible with many open-source GUI creation tools. A simple, inexpensive interface could be developed and tested at no cost allowing the signal processing elements of the design to be at the forefront. For these reasons, VST was chosen as the development platform.



Octave 1: 1
Octave 2: 3
Octave 3: 4
Display: Total Waveform
Destination: Amplitude
Type: Low Pass
Presets: 3

Figure 1 - Original MATLAB synthesizer interface

With the development environment decided, the next step was to choose which synthesis techniques to implement. The original design used an inefficient combination of additive synthesis for signal generation combined with filtering to emulate the subtractive synthesis of analog instruments. This was done to ensure that the signal produced was band limited within the constraints of the sampling theorem; in order to ensure a band limited signal [5], the output *y(t)* of the oscillators must be of the form:

$$y(t)=\sum_{n=1}^{\log_2(f_s/f)} \frac{1}{n}\sin\left(2\pi n\frac{f}{f_s}t\right)$$

where f_s is the sampling rate and f is the frequency of the current pitch. This requirement was easily met in MATLAB by using a Fourier series summation to define band limited square and sawtooth waves. This method proved to be highly inefficient as it required processing the sine of $\log_2(f_s/f)$ angles for each of the three "oscillators" leading to significant time spent filling buffers for processing and an accompanying large block size. Improving the efficiency of signal generation through a more advanced synthesis technique was therefore the first priority for the practical instrument.

As described in the introduction, there are many different types of synthesis methods; of these, the area currently under greatest development (and of greatest personal interest) is physical modeling synthesis. In contrast to the spectrally-based subtractive synthesis methods used in analog synthesizers such as the Minimoog or in early digital synthesizers employing FM synthesis, physical modeling synthesis generates signals in a manner similar to formant speech synthesis. Mathematical acoustic models which describe the physical process occurring are developed and excited with an source that mimics the native source. The resulting sound can be remarkably accurate but the computations involved can be equally involved. With the time available to complete the design and implementation of a practical instrument once again the primary consideration, programming a physical modeling synthesizer from scratch was ruled out.

At this point, several experiments were carried out to explore unfamiliar – and hopefully novel – synthesis techniques. The goal of these experiments (primarily conducted in MATLAB) was to see what other methods might be used to generate controlled pitches. Since the characteristic sound of an instrument is described by the number of overtones and their relative intensities across its range, the spectral content of various sounds was considered as starting point. This led to the question of what happens if a representative frequency sample of an instrument is taken and used to generate a spectral model of the instrument's characteristics. Instead of sampling in time and interpolating the time signal as necessary to adjust pitch, the frequency sample would be scaled as necessary and used to build a resonant filter which matches the frequency content of the original instrument. The filter could then be excited with white noise to generate a "random" output.

This approach worked up to a point; it was possible to use the FFT of sampled sounds to generate a filter in MATLAB. However, using white noise as the signal source led to too many variations in the signal and produced an unmusical result. The same method was tried using an impulse train at the desired frequency which revealed the inherent problem with this approach; the impulse train only produced a useful signal when it was at a rate equal to the time sample size of the original sound and the signal produced was merely a continuously looped playback of the windowed time sample. The one-to-one relationship between time and frequency representations of the same signal had been neglected and the entire process amounted to nothing more than an extra step in time sampling – one which also removed the ability to change the frequency of the output. Given more time to refine the process, this method would likely function as intended but the result is likely no better than time sampling or other spectral synthesis methods.

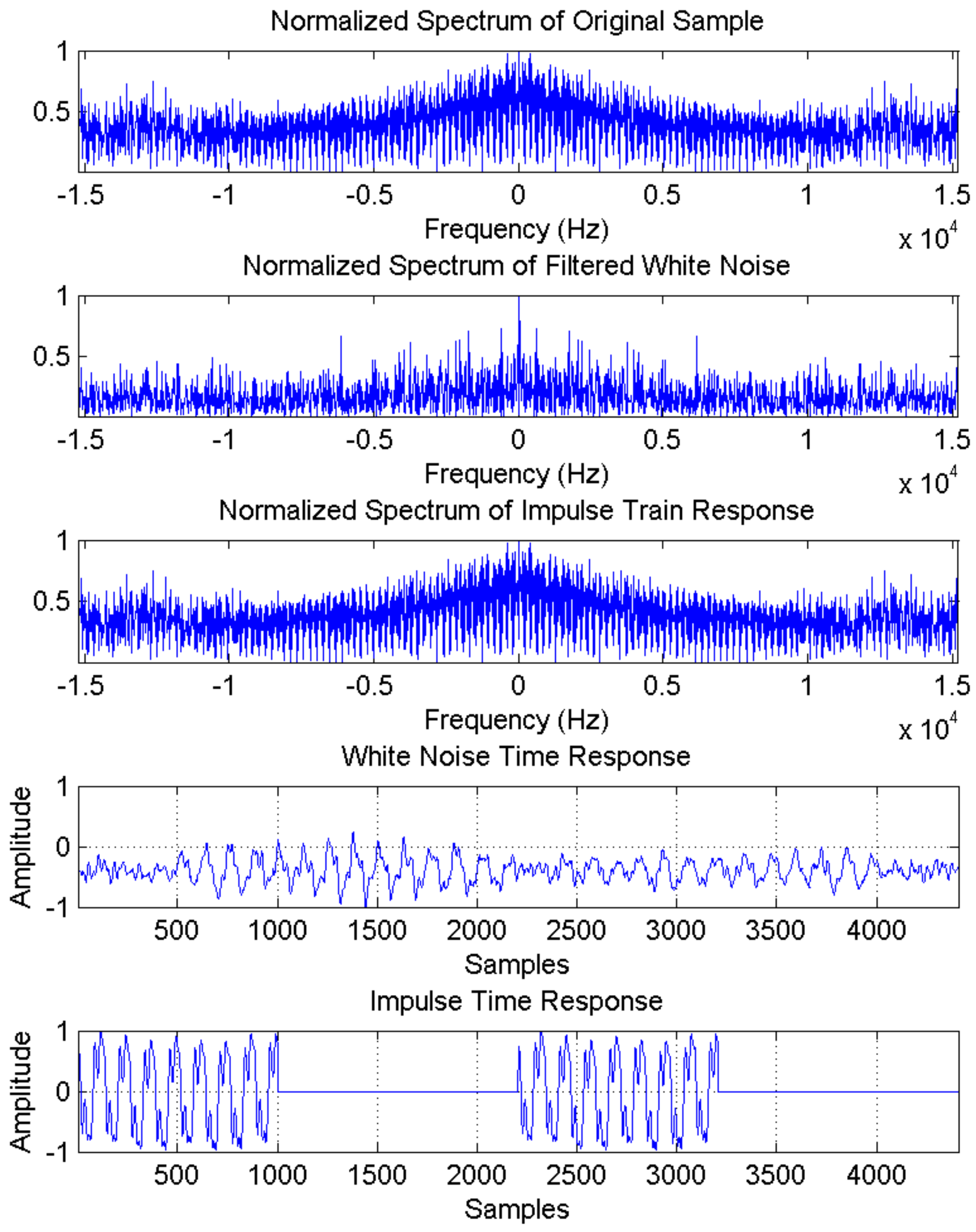


Figure 2 – Sample output of frequency sampling tests

Another approach based on resonant filters was considered that involved using a sudden change in the filter's Q factor to generate pitches. This was tested using Max, a visual design environment that allows the interconnecting of various DSP components using virtual “patch cords” [9]. Max is not a rigorous means of development and seems geared more towards musicians than engineers but it is still a useful prototyping tool for testing ideas quickly; it was particularly useful for this experiment due to the ease with which MIDI controls can be connected to a design. The “variable-Q” synthesizer consisted of a white noise generator filtered through a bank of 4 filters whose outputs were summed. The gain of the filters and their Q factors were tied to MIDI input velocity; when no keys are pressed, the filters were “zero Q” and their gain was also zero to mute the white noise generator. Striking a key set the gain to a positive value and, more importantly, abruptly changed the filters' Q factors in proportion to the velocity which caused the filter to resonate; the higher the velocity, the higher the Q and the longer the sound would ring. This design produced interesting percussive sounds that had a natural sounding overdrive and decay built into them. Unfortunately, the programming involved in developing such a system as a VSTi seemed far too complex at the time and this technique was not pursued further for this project.

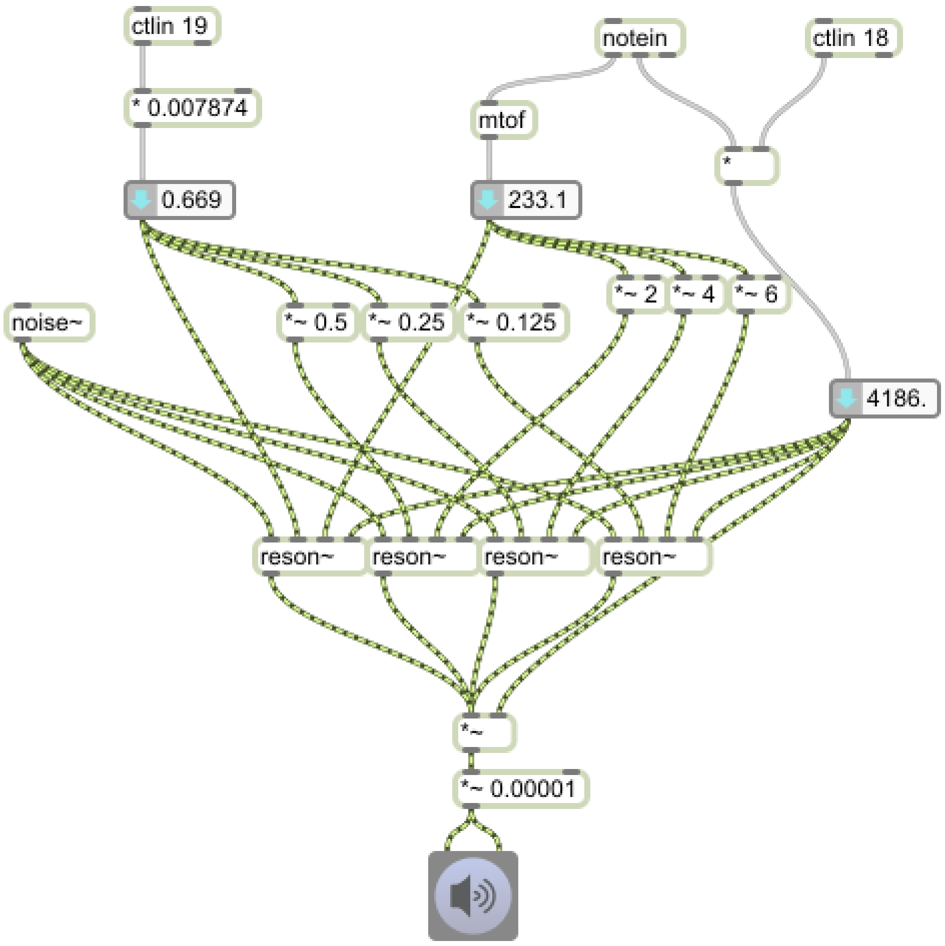


Figure 3 – Max patcher for “variable-Q” synthesizer

After evaluating these other options, it was clear that the best route to a practical instrument was to utilize the existing design from the MATLAB synthesizer and develop it further. Since the basic design would not change, the first step in developing a practical instrument was learning about the VST framework. Steinberg provides SDKs for both the current (VST3) and previous (VST2.4) versions, each of which includes sample plugins for both Windows and Mac OSX. Given that cross platform functionality was one of the main reasons for choosing VST, considerable time was spent attempting to compile the VST3 samples on both Windows and Mac OSX without success and with little support available online. Further research revealed that the majority of VST hosts and plugins available continue to support VST2.4 only instead of upgrading to VST3. Anecdotal evidence indicates that this is due to the increased complexity of VST3 over VST2.4 for minimal gains in performance as well as the cost involved for consumers to upgrade to VST3 compatible software. Based on this experience and evidence, VST2.4 was chosen as it seemed to provide the easiest access for learning about the standard and the most potential users.

Unfortunately, VST2.4 was not exactly "simple" either so templates and tutorials were consulted as a means to get up and running with actual signal processing development. This led to the biggest breakthrough of the early development process: finding JUCE (Jules' Utility Class Extensions) [7]. JUCE is a set of platform-independent GPL C++ classes which provides a wrapper to the VST framework. It also includes project generators for both Apple's Xcode and Microsoft's Visual Studio, GUI design tools and wrappers to other audio plugin formats such as Apple's Audio Unit. This easily allowed for simultaneous development in both Xcode and Visual C++ to verify that cross platform operation remained identical as changes were made – something that proved invaluable during troubleshooting as the behavior of the VST hosts and operating systems made it possible to detect errors more readily.

With a means to easily connect to the VST host/plugin system and design a GUI, the focus shifted to rewriting the synthesizer's MATLAB functions in C++. A major priority was improving the signal generation method to allow a block size small enough for a reasonable response to performer inputs. Previous research into physical modeling had led to the discovery of the Synthesis Toolkit (STK), a set of open-source C++ classes that implement typical DSP functions used in synthesizer design [8]. Several physical models are also included but since the implementations were already complete they were ruled out for use in this project. STK's library of oscillator, filter, and delay functions, however, provided a straight-forward way to implement many of the same algorithms used in the MATLAB synthesizer in an efficient manner.

The last step necessary to begin development was to determine what tools would be used to test the plugin as it was developed. With no commercial host available, free host applications were found and used for testing through the development process. VSTHost [10] and SAVIHost [11] were used on Windows, Ugly VSTi Interface was used on Mac OS X [12] and REAPER [13] was used on both operating systems.

In order to learn about JUCE, STK, and C++ before tackling development of the final product, several small JUCE projects were created. The first, `STKTest`, was used to test if it was even possible to use STK within JUCE. The example project included in the JUCE distribution was modified to replace the direct calculations used by the original author with one of the physical model classes from STK. One of the limitations of using JUCE and STK together became immediately apparent; due to the different mechanisms used to manage blocks of data, there would be no way to use STK's built-in `STKFrames` class to automatically handle data in blocks. Instead, all operations would have to be performed on a sample-by-sample basis – negating one of the primary initial attractions of STK. However, STK still provided enough benefits through its working implementations of the main synthesizer components to justify its use. This project was also useful for learning about the particular compiler settings required for each operating system and how to configure them in both development environments.

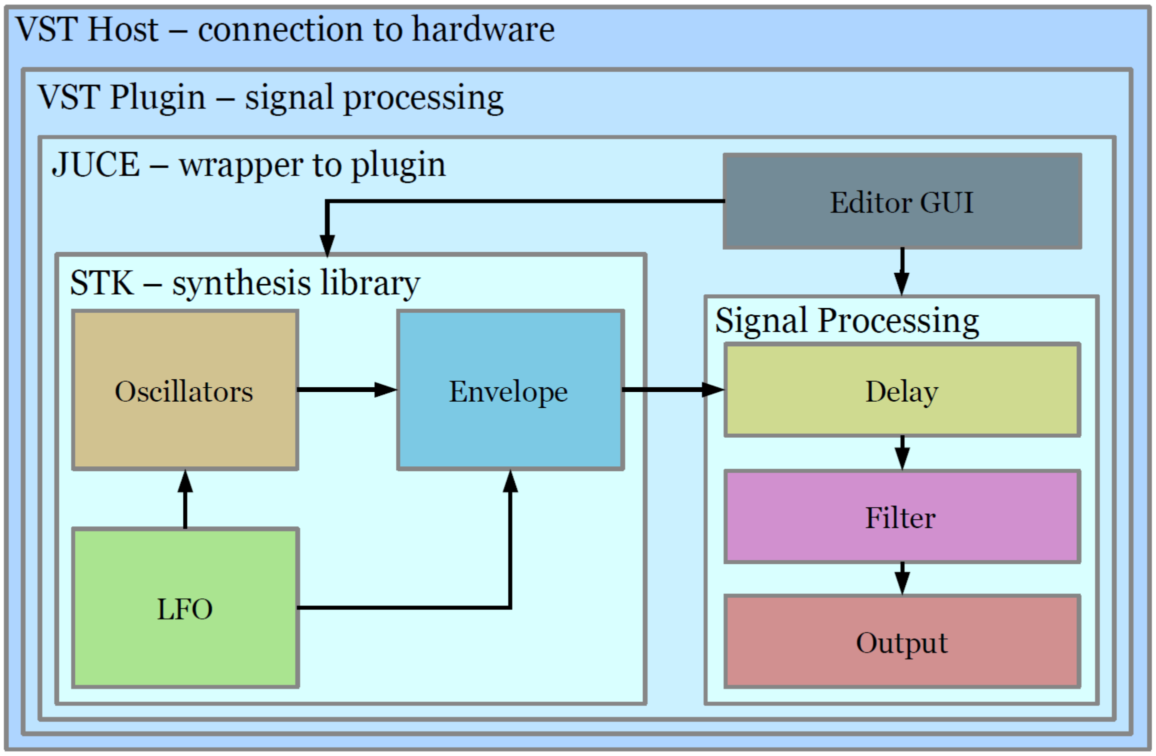


Figure 4 – Hierarchy of VST, JUCE and STK

The second test project, `miniSynth`, was written to practice developing a GUI using the tools included in JUCE as well as to test the processing structure to be used in the final project. This project was built from scratch using JUCE's project management tool, The Introjucer. The Introjucer generates the required Xcode and Visual Studio projects necessary to build audio plugins and other supported JUCE applications based on user-selected options. Again, STK classes were used as the signal source and no additional effects processing was added. Both physical models and signal generator classes (sine, square and sawtooth) were experimented with to learn how to manage them in the context of JUCE. The classes were also split into separate files for this project (as opposed to the included example projects) to verify that the desired file structure could be used. A simple GUI was developed and successfully added to the project using The Jucer, JUCE's component builder application. The Jucer provides a visual GUI editor that generates the necessary code for the plugin's editor class and was used to develop the final interface for the finished product.

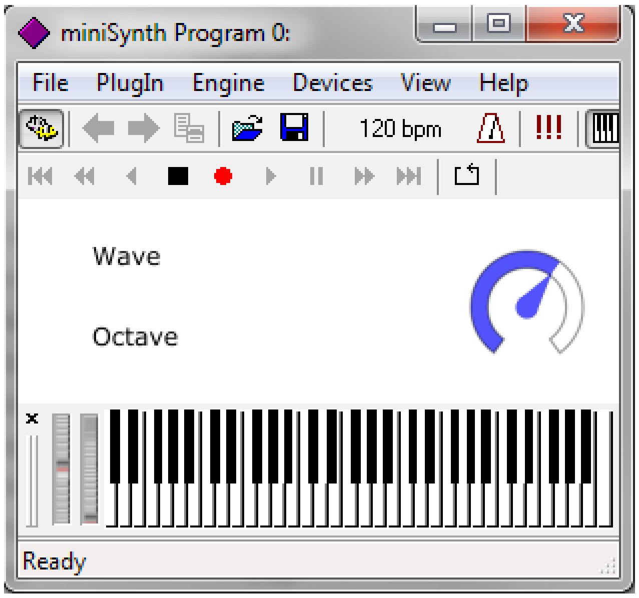


Figure 5 – miniSynth GUI running in SAVIHost

Using `miniSynth` as a model, development began on the actual synthesizer project. Unfortunately, several inadvertent bugs – born of inexperience – were carried over from the earlier design. The most significant was that the buffer was being passed over multiple times unnecessarily while being filled with oscillator data. As a result, latency increased as operations were added until the instrument was not playable. The error was ultimately found by returning to the now hugely simpler `miniSynth` and analyzing the code that had been copied into the final project until the unnecessary nested loop was found.

Despite the original intent to use STK to provide oscillator, filter and delay classes, the issue with block processing mentioned earlier led to difficulties carrying this out. In the configuration originally tested, the filter implementations provided in STK did maintain internal states correctly; ultimately, this proved moot as the rolloff was not steep enough and another filter design was used. Similarly, the delay line from STK did not work as intended so a circular buffer was created instead. These errors were likely the result of inexperience in managing C++ objects; future work will evaluate if it is possible to use these classes after all.

The filter design used in the final design was a high-order parametric equalizer filter design by Orfanidis. [2] This design was chosen to meet the desired rolloff characteristics for the filter, particularly in the bandpass case. The filter design was tested and verified in MATLAB prior to implementation in C++; this initially proved troublesome due to instability in one of the cascaded sections. Ultimately, the instability was a result of a missing pole/zero cancellation and was remedied by using two different sets of coefficients for the low/highpass cases and bandpass case that carried out the appropriate cancellation.

An early working version of the synthesizer was made available to musician colleagues to receive their feedback on its sound and interface. Based on this feedback, changes were made to the UI and output levels to improve performance.

IV. Technical Details

The synthesizer is divided into six processing sections: oscillators, low frequency oscillator (LFO), envelope, delay, filter and output. There are three oscillators with four parameters which are independently controlled for each oscillator. The output of the oscillators is of the form

$$y_{osc_n}(t) = E(t)[G_{osc_n} \sum_{m}^{\log_2(f_s/f)} \sin(2\pi m \frac{f}{f_s} t + L(t)) + G_{noise} N(t)]$$

where *E(t)* is the envelope output, G_{oscn} is the output level for oscillator *n*, *L(t)* is the LFO output (used here for vibrato effect) and G_{noise} is the noise generator gain, *N(t)* is the output of a pseudo-random noise generator. The number and nature of the harmonics, *m*, is dependent on the waveform selected but is bounded in all cases by $\log_2(f_s/f)$ to ensure a band limited signal as required to prevent aliasing.

The envelope, *E(t)*, describes an ADSR (Attack, Decay, Sustain and Release) envelope – a model used to describe the various phases of a musical sound from its initiation to conclusion. In this model, the attack phase determines the time for the amplitude of a sound to rise from zero to its maximum level, the decay phase determines the time to fall from the maximum to the level described by the sustain parameter and the release phase is the time for the sound to return to zero after the sound is no longer being sustained. The envelope generator provided in STK produces a linear envelope which responds to MIDI note on commands to begin the attack phase which is then followed by the decay and sustain phases until a note off command initiates the release phase. The total envelope can be described as:

$$E(t) = \begin{cases} \frac{t - t_{noteOn}}{t_{attack}}, t_{noteOn} \leq t \leq t_{attack} \\ 1 + \frac{E_{sustain} - 1}{t_{decay} - t_{attack}}(t - t_{attack}), t_{attack} \leq t \leq t_{decay} \\ E_{sustain}, t_{decay} \leq t \leq t_{noteOff} \\ E_{sustain} + \frac{-E_{sustain}}{t_{release} - t_{noteOff}}(t - t_{noteOff}), t_{noteOff} \leq t \leq t_{release} \end{cases}$$

where t_{noteOn} is the time at which the MIDI note on command is received, t_{attack} is the end of the attack phase, t_{decay} is the end of the decay phase, $E_{sustain}$ is the sustain level, $t_{noteOff}$ is the time at which the MIDI note off command is received and $t_{release}$ is the end of the release phase. This operation currently occurs in each of the three oscillators independently; this was done to generalize the oscillator class for use in other projects. Since this is a monophonic instrument, a new key press begins a new attack phase which interrupts any remaining release phase running to completion from a previous note. In a polyphonic synthesizer, each note has its own envelope; this allows for layered effects as keys are released at different times and will be enabled in a future polyphonic version.

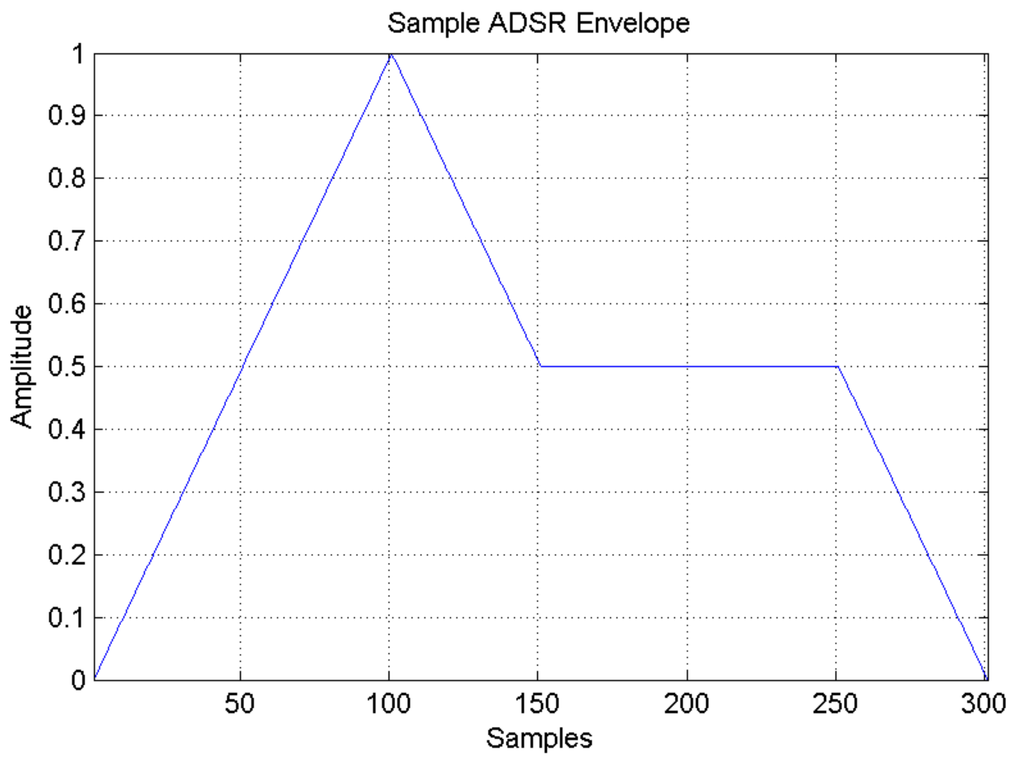


Figure 6 – Example of an ADSR envelope

The low frequency oscillator (LFO) output

$$L(t)=A_{LFO}\sin\left(2\pi\frac{f_{LFO}}{f_s}t\right)$$

is used internally in the oscillator to vary the frequency of the output. The amplitude of the LFO, A_{LFO}, limits the variation of the pitch frequency to $f \pm A_{LFO}$. In general, LFOs are used to generate time varying effects such as filter sweeps and phase effects. In this instrument, the available effects are vibrato (frequency modulation) and tremolo (amplitude modulation).

The outputs of each of the three audio frequency oscillators are sequentially added to the processing buffer to produce the total oscillator output:

$$y_{osc}(t)=\sum_{n=1}^{3} y_{osc_n}(t) \quad .$$

Once the oscillator output has been added to the processing buffer, various effects processing is applied to the buffer. As with the oscillator functions, effects processing code was generalized to allow for the reuse of the included effects code in other plugins by simply loading other data into the buffer. Effects processing occurs on a sample-by-sample basis with all effects being applied to given sample during a single pass through the buffer.

First, tremolo is applied to the buffer containing $y_{osc}(t)$ using the output of an LFO:

$$y_{trem}(t) = [1 + L(t)]\, y_{osc}(t) \quad .$$

In theory, this LFO and the LFO used to generate the vibrato effect within the oscillators could be the same; in practice, it was easier to have two separate LFO objects controlled by the same parameters. This, however, introduced bugs where both effects can be enabled at the same time but with only the currently selected effect responding to parameter changes. While the results can be interesting, completely turning off both vibrato and tremolo can be difficult since it requires zeroing the amplitude and frequency of both LFOs independently. Future versions will either provide separate controls for each effect or use a single oscillator to eliminate this error.

Following the application of the tremolo effect, the buffer is passed through a hyperbolic tangent function to introduce a simple form of non-linear distortion resulting in $y_{drive}(t)$:

$$y_{drive}(t) = \tanh(G_{drive}\, y_{trem}(t)) \quad .$$

The amount of distortion is controlled by the overdrive gain, G_{drive}, as illustrated below. Currently, the overdrive is always on with $\min(G_{drive}) = 1$ so some distortion is always present. Future versions will allow performers to completely disable the overdrive as well as improve upon the methods used to generate distortion to more closely resemble the output of analog devices.

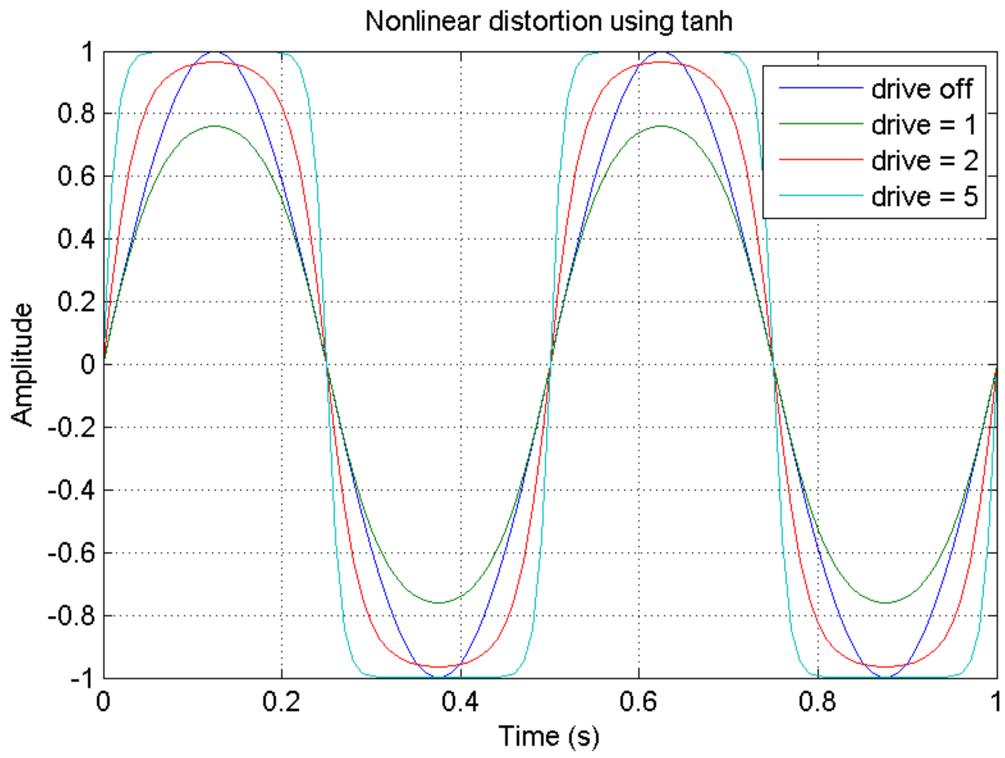


Figure 7 – Nonlinear distortion using hyperbolic tangent

The last effect prior to filtering is a linear delay with delay *D* samples and gain G_{delay} represented by

$$y_{effects}(t) = y_{drive}(t) + G_{delay}\, y_{delay}(t)$$
$$y_{delay}(t) = y_{effects}(t - D)$$

where $y_{effects}(t)$ is the total output of the effects processing (tremolo, overdrive and delay). The delay, $y_{delay}(t)$, is implemented as a circular buffer of the same length as the processing buffer plus *D* samples using separate read and write pointers. The position of the pointers is checked as they are advanced and wrapped to the beginning of the buffer when appropriate.

With the effects processing complete, the buffer is then filtered using a cascade of three second-order (fourth-order for bandpass) sections with a fixed -18 db per octave rolloff. The first two sections implement a 3rd order Butterworth filter with digital transfer functions:

$$H_0(z) = \frac{b_{00} + b_{01} z^{-1}}{1 + a_{01} z^{-1}} \frac{b_{10} + b_{11} z^{-1} + b_{12} z^{-1}}{1 + a_{11} z^{-1} + a_{12} z^{-2}}$$

$$H_1(z) = \frac{b_{00} + b_{01} z^{-1} + b_{02} z^{-2}}{1 + a_{01} z^{-1} + a_{02} z^{-2}} \frac{b_{10} + b_{11} z^{-1} + b_{12} z^{-1} + b_{13} z^{-3} + b_{14} z^{-4}}{1 + a_{11} z^{-1} + a_{12} z^{-2} + a_{13} z^{-3} + a_{14} z^{-4}} \quad .$$

The filter coefficients are derived using Orfanidis' high-order parametric equalizer design described in [2]. This method enables the filter to have a steeper rolloff than possible with a first- or second-order filter, a feature found to be necessary after lower order filters failed to produce the desired attenuation outside of the pass band. The parametric equalizer defined by the first and second sections of the cascade has fixed gain of -6 dB at its cutoff frequencies; further, the bandwidth of the equalizer in the bandpass case is proportional to the center frequency of the filter such that the bandwidth of the filter is approximately two octaves across its frequency range.

The third stage of the filter is a second-order resonator section with digital transfer function:

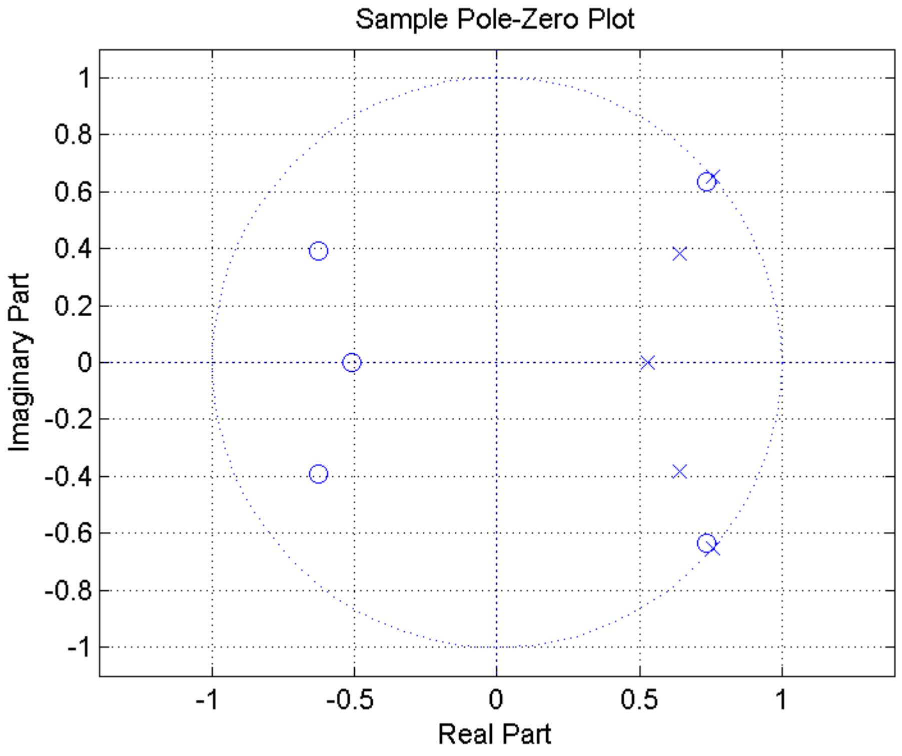
$$H_2(z) = \frac{b_{20} + b_{21} z^{-1} + b_{22} z^{-1}}{1 + a_{21} z^{-1} + a_{22} z^{-2}}$$

whose coefficients are derived from [1]. The resonator bandwidth, $\Delta\omega_2$, reference gain, G_{02}, and gain, G_2, are kept in fixed relationships to the parametric equalizer passband gain, $G_{(0+1)}$, in order to ensure that the resonant behavior is unchanged as overall gain levels and frequency are changed:

$$\Delta\omega_2 = 0.01\, G_{(0+1)}$$
$$G_{0_2} = G_{(0+1)}$$
$$G_2 = R\, G_{(0+1)}$$

where *R* is the gain of the resonator above the passband gain $G_{(0+1)}$.

The pole-zero plot and frequency response of an example lowpass filter with $G_{(0+1)} = 0.1$, $R = 100$, and $f_c = 5000$ Hz are show below.



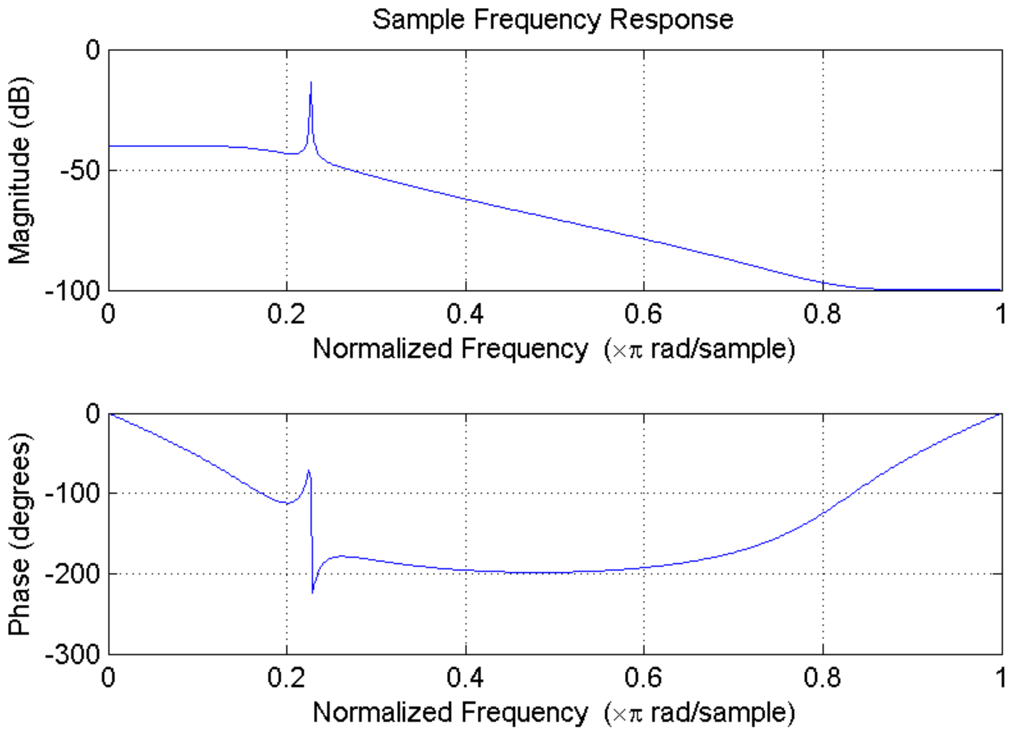


Figure 8 – Example lowpass filter pole-zero plot and frequency response
$G_{(0+1)} = 0.1$, $R = 100$, $f_c = 5000$ Hz

The overall filtering operation is then

$$Y_{filtered}(z) = Y_{effects}(z) H_0(z) H_1(z) H_2(z) \quad .$$

Finally, the output gain, G_{output}, is applied to the filtered buffer:

$$y_{output}(t) = G_{total} \, y_{filtered}(t) \quad .$$

All of the above processing happens in mono to reduce the number of calculations required. At this point, the buffer is copied to both channels to produce a stereo output. A future panning effect could be easily implemented at this stage.

From here, the remainder of the operations necessary to connect to the VST host application are carried out through JUCE's connections to the VST plugin functions. The operations described above, along with the relevant connections to the VST system and GUI provided by JUCE, are organized into five files:

- vstSynthEditor.cpp – an automatically generated file from JUCE's GUI designer. This file contains all of the graphic layout and design information as well as handling callbacks from each of the controller components.

- vstSynthProcessor.cpp – the default file created by JUCE to implement audio plugins. This file contains the instrument initialization, controls the number of voices available to the synthesizer, stores parameter values, provides MIDI connectivity and is used to generate the VST block processing instructions. The delay, overdrive and tremolo effects are implemented directly in this class while the oscillators and filter are implemented through calls to other classes.

    - vstSynth.cpp – user created file containing the JUCE SynthesiserVoice for the synthesizer. Local parameters for each voice are stored here (for later use in a polyphonic instrument) and the methods to initially fill the output buffer with the oscillator output are part of this class.

        - vstSynthOscillator.cpp – outputs a single sample of STK oscillator data based on parameters passed from the calling SynthesiserVoice. The ADSR envelope is applied during this step as is vibrato, if enabled.

    - vstSynthFilter.cpp – high-order parametric filter implementation called during buffer processing. Consists of a cascade of three fourth-order sections: two for the shaping filter and one for the superimposed resonant peak.

V. Usage

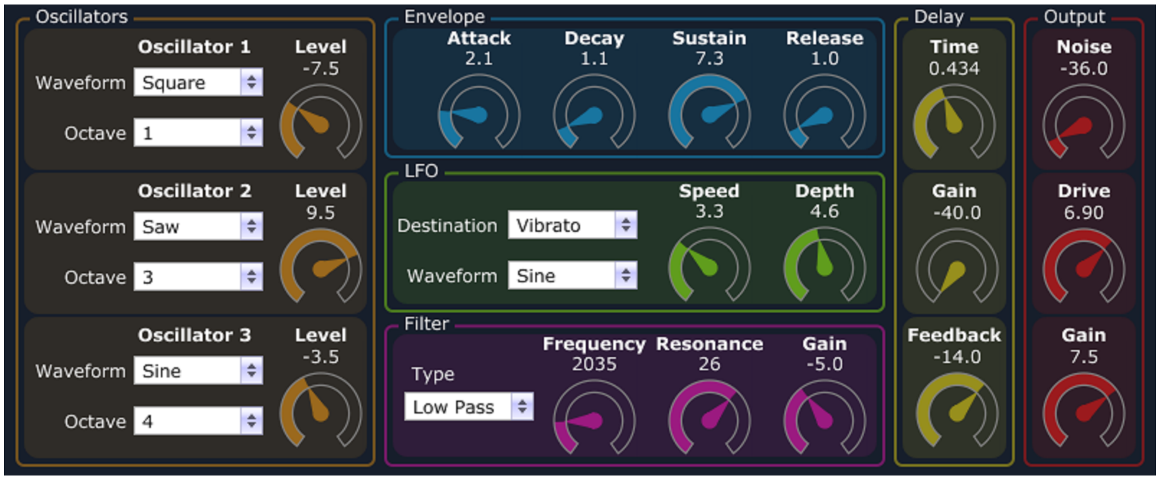

Oscillators
Oscillator 1
Waveform Square
Octave 1
Level -7.5
Oscillator 2
Waveform Saw
Octave 3
Level 9.5
Oscillator 3
Waveform Sine
Octave 4
Level -3.5
Envelope
Attack 2.1
Decay 1.1
Sustain 7.3
Release 1.0
LFO
Destination Vibrato
Waveform Sine
Speed 3.3
Depth 4.6
Filter
Type Low Pass
Frequency 2035
Resonance 26
Gain -5.0
Delay
Time 0.434
Gain -40.0
Feedback -14.0
Output
Noise -36.0
Drive 6.90
Gain 7.5

Figure 9 – VST synthesizer user interface

The synthesizer's interface was designed to be familiar and intuitive for performers with experience using other synthesizers. The GUI is divided into color-coded sections that correspond to the parameters for each processing section so that performers can quickly locate the controls they wish to adjust. These sections and their associated controls are:

- Oscillators (orange) – three separate oscillators driven by STK classes
  - Waveform – choice of sine wave and band-limited sawtooth and square waves
  - Octave – multiplier applied to note frequency derived from MIDI note number
  - Level – output gain (dB) for each oscillator section

- Envelope (blue) – ADSR envelope parameters
  - Attack – time from initiation of note to peak
  - Decay – time from peak to sustained level
  - Sustain – level of sustain until note off command
  - Release – time from key release to note end

- LFO (green) – Low Frequency Oscillator for effects processing
  - Destination – vibrato (angle modulation) or tremolo (amplitude modulation)
  - Waveform – not implemented
  - Speed – LFO Frequency
  - Depth – LFO Amplitude

- Filter (magenta) – parametric filter cascade with resonant peak
  - Type – choose between lowpass, bandpass, and highpass filtering
  - Frequency – corner/center frequency of filter. Resonant peak is located here
  - Resonance – output level (dB) of resonant peak above shaping filter gain
  - Gain – gain (dB) of shaping filter

- Delay (yellow) – circular delay buffer
    - Time – time (seconds) of delay. Resets buffer when changed to avoid addressing errors
    - Gain – not implemented
    - Feedback – controls the amount of delayed signal mixed back into the output
- Output (red) – final output stage
    - Noise – gain (dB) of noise generator
    - Drive – multipler applied to signal to control amount of distortion from tanh function
    - Gain – overall output gain (dB)

VI. Example

The following example traces a 20 Hz square wave through the synthesizer signal path. First, the band-limited signals are generated in the oscillator objects and vibrato is applied.

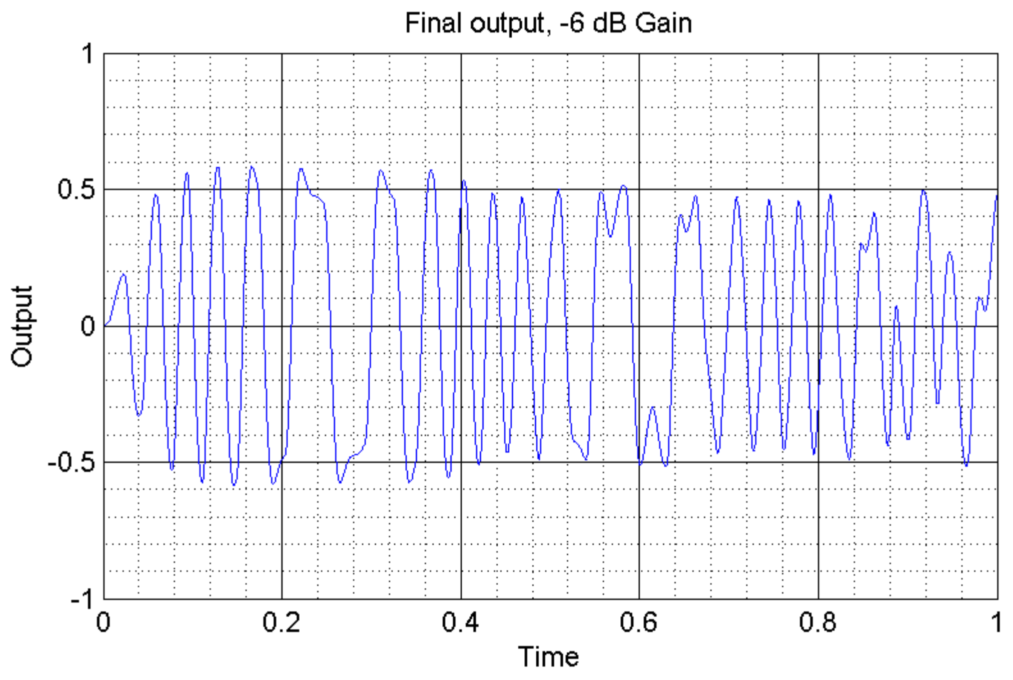


Figure 10 – Signal Generation

Next, the delay and overdrive are applied to the oscillator output. Note that tremolo is disabled in this example.

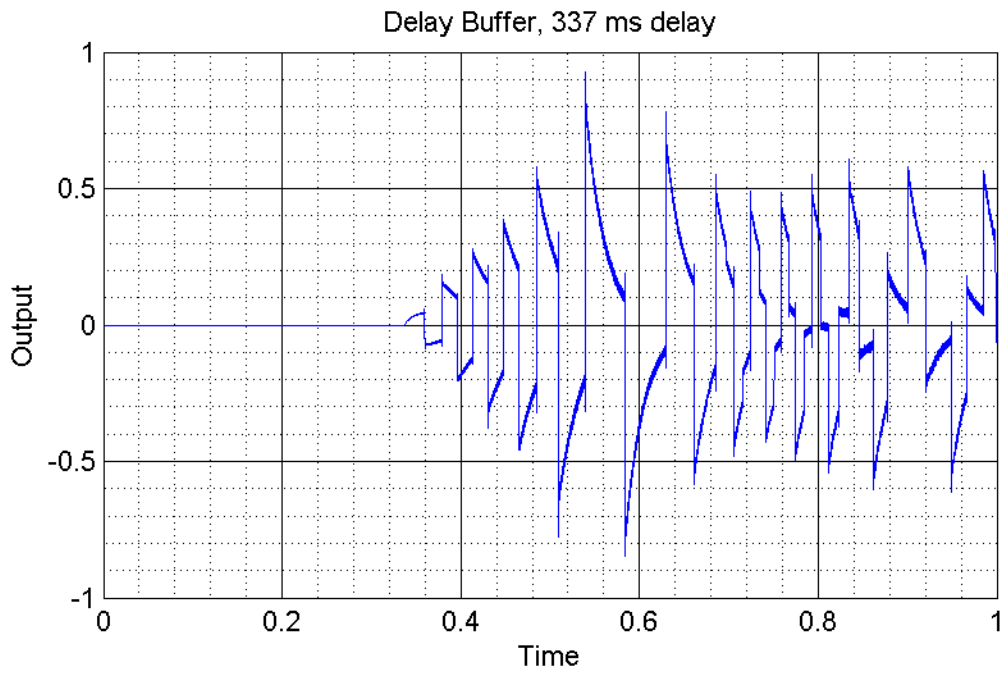


Figure 11 – Delay buffer contents

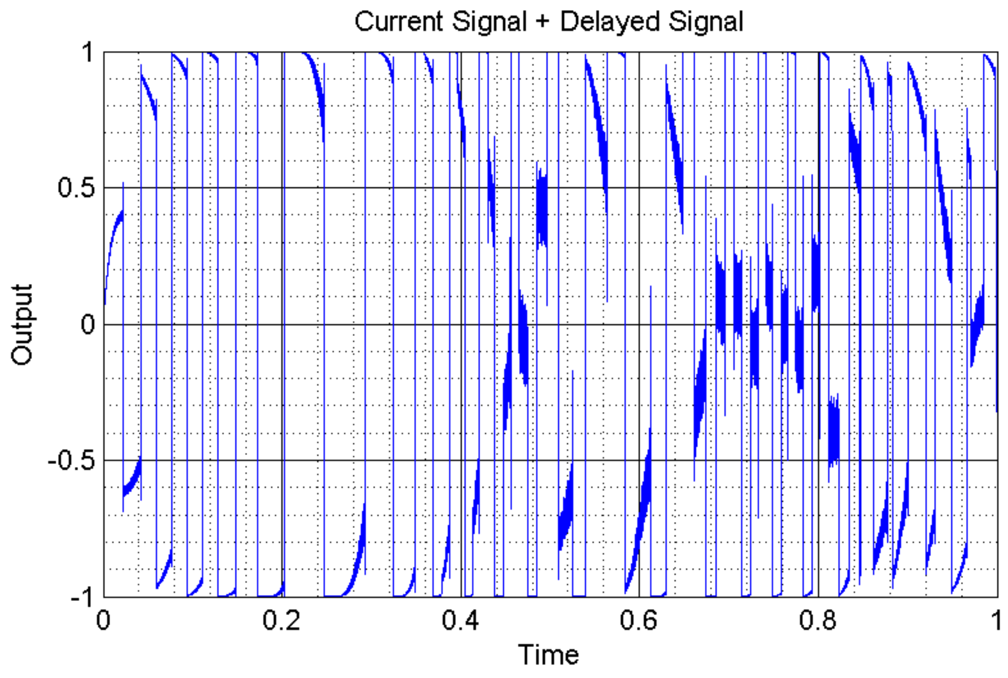


Figure 12 – Sum of delay buffer and current oscillator output

The output of the effects processing stage is then filtered, in this case a lowpass filter with 50 Hz cutoff with no resonant peak.

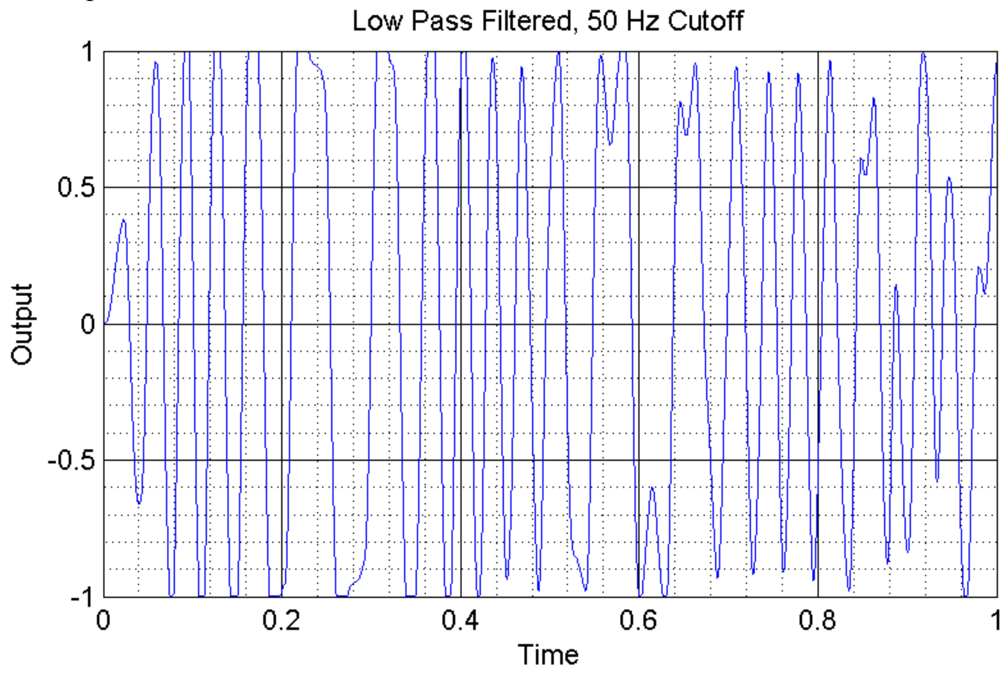


Figure 13 – Filter output

Finally, the buffer is scaled by the final output gain before being passed to the appropriate VST operations by JUCE. The process then repeats for the next block of data.

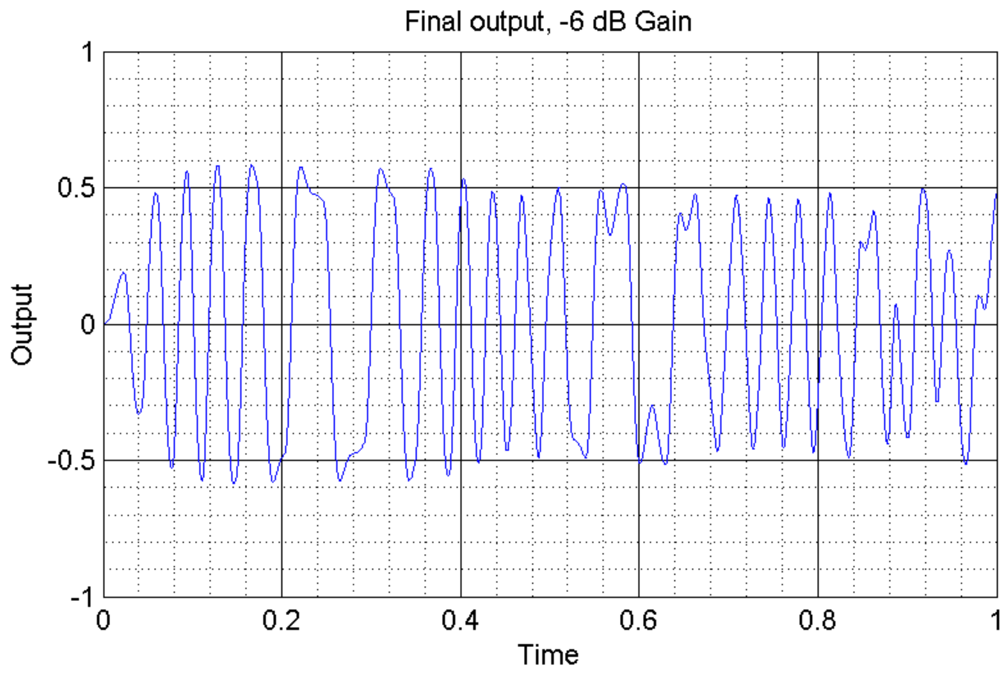


Figure 14 – Final output

VII. Future Development

Planned areas of development include features such as polyphony, chorus effects, filter frequency sweep, master tuning and full MIDI control of all parameters. The source code has been released under the GPL and is available at http://code.google.com/p/vstsynth.

VIII. References

[1] Sophocles J. Orfanidis, (2010) “IIR Digital Filter Design,” in *Introduction to Signal Processing* [Online]. Available: http://eceweb1.rutgers.edu/~orfanidi/intro2sp/orfanidis-i2sp.pdf

[2] Sophocles J. Orfanidis, "High-Order Digital Parametric Equalizer Design," *J. Audio Eng.Soc.*, vol. 53, pp. 1026-1046, Nov. 2005.

[3] Richard Boulanger and Victor Lazzarini, Eds., *The Audio Programming Book.* Cambridge, MA: MIT Press, 2011.

[4] Jyri Pakarinen, Vesa Välimäki, Federico Fontana, Victor Lazzarini and Jonathan S Abel, (2011) *Recent Advances in Real-Time Musical Effects, Synthesis, and Virtual Analog Models* [Online]. Available: http://asp.eurasipjournals.com/content/2011/1/940784

[5] Tim Stilson and Julius Smith, (1997) *Alias-Free Digital Synthesis of Classic Analog Waveforms* [Online]. Available: https://ccrma.stanford.edu/~stilti/papers/blit.pdf

[6] Steinberg Media Technologies GmbH, (2006) *Virtual Studio Technology (VST) SDK* [Online]. Available: http://www.steinberg.net/en/company/developer.html

[7] Julian Storer, (2012) *Jules' Utility Class Extensions (JUCE)* [Online]. Available: http://www.rawmaterialsoftware.com/juce.php

[8] Perry R. Cook and Gary P. Scavone, (2011) *The Synthesis ToolKit in C++ (STK)* (4.4.3) [Online]. Available: https://ccrma.stanford.edu/software/stk/

[9] Cycling '74 (2012) *Max6* [Online]. Available: http://cycling74.com/products/max/

[10] Hermann Seib (2012), *VSTHost* [Online]. Available: http://www.hermannseib.com/english/vsthost.htm

[11] Hermann Seib (2012), *SAVIHost* [Online]. Available: http://www.hermannseib.com/english/savihost.htm

[12] reFuse Software, *Ugly VSTi Interface* [Online]. Available: http://www.refusesoftware.com/products/feature/3

[13] Cockos Incorporated (2012), *REAPER* [Online]. Available: http://www.reaper.fm/

Appendix

`noisetest.m`

A resonant filter excited once by white noise and again by an impulse used to verify understanding of the principles of resonant filters. This was used as preparation for the "frequency scaling" tests carried out during development.

```
close all
clear all
clc

Fs = 44100;
f0 = 1760;
df = 1;
dw = 2*pi*df/Fs;
w0 = 2*pi*f0/Fs;
N = 256;
M = floor(Fs/2);
Q = f0/df;

beta = dw/4;
R = (1-beta)/(1+beta);
a0 = R;
a1 = -2*R*cos(w0);
a2 = R;
G = 1;
b0 = 1;
b1 = R;
b2 = R/10;

Num = G*[b0,b1,b2];
Den = [a0,a1,a2];

freqz(Num,Den)

x = rand(Fs,1);
X = fft(x,N);
u = [1,zeros(1,Fs-1)];
% u = [ones(1,floor(Fs/M)),zeros(1,floor(Fs/M))];
% u = repmat(u,1,M/2);
% U = fft(u,N);

y = zeros(1,Fs);
z = zeros(1,Fs);

for n = 3:Fs
    y(n)  = G*b0*x(n)+G*b1*x(n-1)+G*b2*x(n-2)-a1*y(n-1)-a2*y(n-2);
    z(n)  = G*b0*u(n)+G*b1*u(n-1)+G*b2*u(n-2)-a1*z(n-1)-a2*z(n-2);
end

Y = fft(y,N);
Z = fft(z,N);

t = [0:Fs-1]/Fs;
```

```
f = [-Fs/2:Fs/(N-1):Fs/2];

figure
subplot(121)
plot(t,x); axis tight
xlabel('Time (s)'); ylabel('Amplitude');
subplot(122)
plot(f,abs(fftshift(X)/max(abs(X)))); axis tight
xlabel('Frequency (Hz)'); ylabel('Normalized FFT');

% soundsc(x,Fs)

figure
subplot(221)
plot(t,y/max(abs(y))); axis tight
title('White Noise Response')
xlabel('Time (s)'); ylabel('Amplitude')

subplot(222)
plot(f,abs(fftshift(Y)/max(abs(Y)))); axis tight
title('White Noise Spectrum')
xlabel('Frequency (Hz)'); ylabel('Normalized FFT');

soundsc(y,Fs)

% figure
subplot(223)
plot(t,z/max(abs(z))); axis tight
title('Impulse Response')
xlabel('Time (s)'); ylabel('Amplitude')

subplot(224)
plot(f,abs(fftshift(Z)/max(abs(Z)))); axis tight
title('Impulse Spectrum')
xlabel('Frequency (Hz)'); ylabel('Normalized FFT');

% soundsc(z,Fs)

% set(findobj('type','axes'),'fontsize',16,'fontname','Times')
figureHandle = gcf;
%# make all text in the figure to size 14 and bold
% set(findall(figureHandle,'type','text'),'fontSize',20,'FontName','Times')
% set(gcf, 'PaperPosition', [0 0 6.5 5]);
saveas(figure(3),'noisetest.png')
```

```
freqsample.m
```

Used to test the idea of sampling a representative sound and scaling its frequency response to create a filter at various frequencies with similar character to the original sound.

```
close all
clear all
clc

[x,Fs] = wavread('wav/partialnote.wav');
x = x/max(max(x));
% w = hamming(length(x));
% x = w.*x(:,1);

N = 20;
frame = ceil(Fs/N);

% % artificial square wave

% n = [0:frame-1];
% f = 110;

% x = 0;
% for j = 1:2:13
%     x = x+1/j*sin(2*pi*j*f/Fs*n);
% end

% x = sign(sin(2*pi*f/Fs*n));
% x = filter(H,x);

% Initialise playrec
if playrec('isInitialised')
    playrec('reset');
end
playrec('init',Fs,4,-1);

i = 10;

M = frame;

X = fft(x,M);
Z1 = 0;
z1 = [];
Z2 = 0;
z2 = [];

while i+frame < Fs/10+frame
    f1 = rand(1,frame);
    f2 = [1,zeros(1,frame-1)];
%    f = [90+20*rand(1,1),zeros(1,frame-1)];
    F1 = fft(f1,M);
    F2 = fft(f2,M);
    Y1 = X'.*F1;
    Y2 = X'.*F2;
    Z1 = Z1+Y1;
    Z2 = Z2+Y2;
    y1 = ifft(Y1',M);
    y2 = ifft(Y2',M);
    z1 = [z1;y1];
    z2 = [z2;y2];
    i = i+frame;
%     playrec('play',[y y],[1 2]);
end

% soundsc(z1,Fs)
% soundsc(z2,Fs)

f = (-Fs/2:Fs/(frame-1):Fs/2);

n = M/2.9;
f = f(end/2-n:end/2+n+1);

X = X(end/2-n:end/2+n+1);
Z1 = Z1(end/2-n:end/2+n+1);
Z2 = Z2(end/2-n:end/2+n+1);

figure
subplot(511)
plot(f,abs(fftshift(X))/max(abs(X))), axis tight
title('Normalized Spectrum of Original Sample')
xlabel('Frequency (Hz)')

subplot(512)
plot(f,abs(fftshift(Z1))/max(abs(Z1))), axis tight
title('Normalized Spectrum of Filtered White Noise')
xlabel('Frequency (Hz)')

subplot(513)
plot(f,abs(fftshift(Z2))/max(abs(Z2))), axis tight
title('Normalized Spectrum of Impulse Train Response')
xlabel('Frequency (Hz)')
% saveas(figure(1),'freq_sample_spec.png')
%
% figure
subplot(514)
plot(z1/max(abs(z1))); grid; axis tight; ylim([-1 1])
title('White Noise Time Response')
xlabel('Samples'); ylabel('Amplitude')

subplot(515)
plot(z2); axis tight; grid; ylim([-1 1])
title('Impulse Time Response')
xlabel('Samples'); ylabel('Amplitude')
set(gcf, 'PaperPosition', [0 0 6 7.5]);
saveas(figure(1),'freq_sample.png')
```

hpeqfilters.m

Used to verify that the filter design parameters for the high-order parametric equalizer were correct. Implemented nearly identically in C++ for the actual instrument.

```
close all
clear all
clc

%fixed values
fs = 44100;

G0 = 0.0001; % reference gain

%filter parameters
G = 0.1; % filter gain
R = 100; % resonant peak
Gb = max(G,G0)/2; % transition gain
fc = 5000; % from slider
type = 'lp';

%filter design
N = 3; r = 1; L = 1;

Wc = 2*pi*fc/fs;

% Select filter type

switch type
    % lowpass
    case 'lp'
        W0 = 0;
        W1 = 0;
        W2 = Wc;

        % Define parameters (move to separate function in C++)
        deltaW = W2-W1;

        g = G^(1/N); g0 = G0^(1/N); phi = pi/6;

        epsilon = sqrt((G*G - Gb*Gb)/(Gb*Gb - G0*G0));
        beta = epsilon^(-1/N)*tan(deltaW/2);

        c0 = cos(W0); s1 = sin(phi);
        % end parameter definition

        % Second order lowpass section
        D0 = beta + 1;
        b00 = (g*beta + g0)/D0;
        b01 = (g*beta - g0)/D0;
        b02 = 0;
        a01 = (beta - 1)/D0;
        a02 = 0;

        % Fourth order lowpass section
        D1 = beta^2 + 2*s1*beta + 1;
        b10 = (g^2*beta^2 + 2*g*g0*s1*beta + g0^2)/D1;
        b11 = 2*(g^2*beta^2 - g0^2)/D1;
        b12 = (g^2*beta^2  -  2*g*g0*s1*beta + g0^2)/D1;
        b13 = 0;
        b14 = 0;
        a11 = 2*(beta^2 - 1)/D1;
        a12 = (beta^2 - 2*s1*beta + 1)/D1;
        a13 = 0;
        a14 = 0;

        % Second order section coeffs
        b0 = [b00,b01,b02]; a0 = [1,a01,a02];

        % Fourth order section coeffs
        b1 = [b10,b11,b12,b13,b14]; a1 = [1,a11,a12,a13,a14];


    % bandpass - approx two octave bandwidth
    case 'bp'
        W0 = Wc;
        W1 = Wc/2;
        W2 = 2*atan(tan(Wc/2)^2 / tan(W1/2));

        % Define parameters (move to separate function in C++)
        deltaW = W2-W1;

        g = G^(1/N); g0 = G0^(1/N); phi = pi/6;

        epsilon = sqrt((G*G - Gb*Gb)/(Gb*Gb - G0*G0));
        beta = epsilon^(-1/N)*tan(deltaW/2);

        c0 = cos(W0); s1 = sin(phi);
        % end parameter definition

        % Second order bandpass section
        D0 = beta + 1;
        b00 = (g0 + g*beta)/D0;
        b01 =  -2*g0*c0/D0;
        b02 = (g0 - g*beta)/D0;
        a01 =  -2*c0/D0;
        a02 = (1 - beta)/D0;

        % Fourth order bandpass section
        D1 = beta^2 + 2*s1*beta + 1;
        b10 = (g^2*beta^2 + 2*g*g0*s1*beta + g0^2)/D1;
        b11 =  -4*c0*(g0^2 + g*g0*s1*beta)/D1;
        b12 = 2*(g0^2*(1 + 2*c0^2) - g^2*beta^2)/D1;
        b13 =  -4*c0*(g0^2 - g*g0*s1*beta)/D1;
        b14 = (g^2*beta^2  -  2*g*g0*s1*beta + g0^2)/D1;
        a11 =  - 4*c0*(1 + s1*beta)/D1;
        a12 = 2*(1 + 2*c0^2 - beta^2)/D1;
        a13 =  -4*c0*(1 - s1*beta)/D1;
        a14 = (beta^2 - 2*s1*beta + 1)/D1;

        % Second order section coeffs
        b0 = [b00,b01,b02]; a0 = [1,a01,a02];
```

```
        % Fourth order section coeffs
        b1 = [b10,b11,b12,b13,b14]; a1 = [1,a11,a12,a13,a14];

    % highpass
    case 'hp'
        W0 = pi;
        W1 = Wc;
        W2 = pi;

        % Define parameters (move to separate function in C++)
        deltaW = W2-W1;

        g = G^(1/N); g0 = G0^(1/N); phi = pi/6;

        epsilon = sqrt((G*G - Gb*Gb)/(Gb*Gb - G0*G0));
        beta = epsilon^(-1/N)*tan(deltaW/2);

        c0 = cos(W0); s1 = sin(phi);
        % end parameter definition

        % Second order highpass section
        D0 = beta + 1;
        b00 = (g*beta + g0)/D0;
        b01 = -(g*beta - g0)/D0;
        b02 = 0;
        a01 = -(beta - 1)/D0;
        a02 = 0;

        % Fourth order highpass section
        D1 = beta^2 + 2*s1*beta + 1;
        b10 = (g^2*beta^2 + 2*g*g0*s1*beta + g0^2)/D1;
        b11 = -2*(g^2*beta^2 - g0^2)/D1;
        b12 = (g^2*beta^2  -  2*g*g0*s1*beta + g0^2)/D1;
        b13 = 0;
        b14 = 0;
        a11 = -2*(beta^2 - 1)/D1;
        a12 = (beta^2 - 2*s1*beta + 1)/D1;
        a13 = 0;
        a14 = 0;

        % Second order section coeffs
        b0 = [b00,b01,b02]; a0 = [1,a01,a02];

        % Fourth order section coeffs
        b1 = [b10,b11,b12,b13,b14]; a1 = [1,a11,a12,a13,a14];


end

% resonator section
c0r = cos(Wc);
Gr = R * G;
Gbr = Gr/2;
G0r = G;
dWr = 0.001;
betaR = sqrt((Gbr^2-G0r^2)/(Gr^2-Gbr^2))*tan(dWr/2);
```

```
b2 = [(G0r+Gr*betaR)/(1+betaR), -2*(G0r*c0r/(1+betaR)), (G0r-Gr*betaR)/(1+betaR)];
a2 = [1, -2*c0r/(1+betaR), (1-betaR)/(1+betaR)];

% Coefficients are convolved into one polynomial for plotting using freqz
b = conv(b0,b1); a = conv(a0,a1);
b = conv(b,b2); a = conv(a,a2);

% Various input signals for sound example
f = 1000;
x = [ones(1,floor(fs/f)),-ones(1,floor(fs/f))];
x = repmat(x,1,f/2);
% t = 0:length(x)-1;
% x = cos(2*pi*f*t/fs);
% x = [1,zeros(1,fs-1)];
% x = rand(1,fs);

% y = filter(b,a,x);

y = zeros(1,length(x));
z = zeros(1,length(y));


% Second order section
for i = 3:length(x)
    y(i) = b00*x(i) + b01*x(i-1) + b02*x(i-2) - a01*y(i-1) - a02*y(i-2);
end
sound(y,fs)


% Fourth order section
for i = 5:length(y)
    z(i) = b10*y(i) + b11*y(i-1) + b12*y(i-2) + b13*y(i-3) + b14*y(i-4) - a11*z(i-
1) - a12*z(i-2) - a13*z(i-3) - a14*z(i-4);
end
sound(z,fs)
```

```
/*
STKTestEditor.h - header file for STKTestEditor.cpp
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

    This file was auto-generated by the Introjucer!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#ifndef __PLUGINEDITOR_H_255A72F0__
#define __PLUGINEDITOR_H_255A72F0__

#include "../JuceLibraryCode/JuceHeader.h"
#include "PluginProcessor.h"


//==============================================================================
/**
*/
class StktestAudioProcessorEditor  : public AudioProcessorEditor
{
public:
    StktestAudioProcessorEditor (StktestAudioProcessor* ownerFilter);
    ~StktestAudioProcessorEditor();

    //==============================================================================
    // This is just a standard Juce paint method...
    void paint (Graphics& g);
};


#endif  // __PLUGINEDITOR_H_255A72F0__
```

```
/*
STKTestEditor.cpp - automatically generated editor class for STKTest synthesizer
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

    This file was auto-generated by the Introjucer!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"


//==============================================================================
StktestAudioProcessorEditor::StktestAudioProcessorEditor (StktestAudioProcessor* ownerFilter)
    : AudioProcessorEditor (ownerFilter)
{
    // This is where our plugin's editor size is set.
    setSize (400, 300);
}

StktestAudioProcessorEditor::~StktestAudioProcessorEditor()
{
}

//==============================================================================
void StktestAudioProcessorEditor::paint (Graphics& g)
{
    g.fillAll (Colours::white);
    g.setColour (Colours::black);
    g.setFont (15.0f);
    g.drawFittedText ("testing",
                      0, 0, getWidth(), getHeight(),
                      Justification::centred, 1);
}
```

```
/*
STKTestProcessor.h - header file for STKTestProcessor.cpp
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#ifndef __PLUGINPROCESSOR_H_22ABA2A3__
#define __PLUGINPROCESSOR_H_22ABA2A3__

#include "SineWave.h"
#include "Blit.h"
#include "BlitSquare.h"
#include "BlitSaw.h"
#include "BeeThree.h"
#include "Rhodey.h"
#include "Clarinet.h"
#include "../JuceLibraryCode/JuceHeader.h"


//==============================================================================
/**
*/
class StktestAudioProcessor  : public AudioProcessor
{
public:

    //==============================================================================
    StktestAudioProcessor();
    ~StktestAudioProcessor();

    //==============================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock);
    void releaseResources();

    void processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages);

    //==============================================================================
    AudioProcessorEditor* createEditor();
    bool hasEditor() const;

    //==============================================================================
    const String getName() const;

    int getNumParameters();

    float getParameter (int index);
    void setParameter (int index, float newValue);

    const String getParameterName (int index);
    const String getParameterText (int index);

    const String getInputChannelName (int channelIndex) const;
    const String getOutputChannelName (int channelIndex) const;
    bool isInputChannelStereoPair (int index) const;
    bool isOutputChannelStereoPair (int index) const;

    bool acceptsMidi() const;
    bool producesMidi() const;

    //==============================================================================
    int getNumPrograms();
    int getCurrentProgram();
    void setCurrentProgram (int index);
    const String getProgramName (int index);
    void changeProgramName (int index, const String& newName);

    //==============================================================================
    void getStateInformation (MemoryBlock& destData);
    void setStateInformation (const void* data, int sizeInBytes);

private:
    //==============================================================================
       Synthesiser additiveSynth;
       MidiKeyboardState keyboardState;


       JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (StktestAudioProcessor);

};

#endif  // __PLUGINPROCESSOR_H_22ABA2A3__
```

```
/*
STKTestProcessor.cpp - uses STK signal generation or physical model classes to
test compatibility of JUCE and STK
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"

// added to try and implement a synth
class additiveSynthSound : public SynthesiserSound
{
public:
	additiveSynthSound()
	{
	}

	bool appliesToNote (const int /*midiNoteNumber*/)	{return true;}
	bool appliesToChannel (const int /*midiChannel*/)	{return true;}
};

class additiveSynthVoice : public SynthesiserVoice
{
public:
	additiveSynthVoice()
	{
		output = new stk::Rhodey();
	}

	~additiveSynthVoice()
	{
		delete output;
	}

	bool canPlaySound (SynthesiserSound* sound)
	{
		return dynamic_cast <additiveSynthSound*> (sound) != 0;
	}

	void startNote(const int midiNoteNumber, const float velocity, SynthesiserSound*
/*sound*/, const int /*currentPitchWheelPosition*/)
	{
		freq = MidiMessage::getMidiNoteInHertz(midiNoteNumber);

		//output->setFrequency (freq);
		output->noteOn (freq, 0.5f);

	}

	void stopNote(const bool allowTailOff)
	{
			output->noteOff(0.5f);
			//output->setFrequency(0);
			clearCurrentNote();
		}

		void pitchWheelMoved (const int /*newValue*/)
		{

		}

		void controllerMoved (const int /*controllerNumber*/, const int /*newValue*/)
		{
		}

		void renderNextBlock (AudioSampleBuffer& outputBuffer, int startSample, int numSamples)
		{
			float* currentBlock = new float[numSamples];

			for (int currentSample = 0; currentSample < numSamples; currentSample++)
			{
				currentBlock[currentSample] = (float) output->tick();
			}

			for (int i = outputBuffer.getNumChannels(); --i >= 0;)
			{
				outputBuffer.addFrom(i, startSample, currentBlock, numSamples, 0.5f);
			}

			delete [] currentBlock;


		}


private:
		double freq;
		stk::Rhodey* output;
};


//==============================================================================
StktestAudioProcessor::StktestAudioProcessor()
{
		additiveSynth.addVoice (new additiveSynthVoice());
		additiveSynth.addSound (new additiveSynthSound());

}

StktestAudioProcessor::~StktestAudioProcessor()
{
}

//==============================================================================
const String StktestAudioProcessor::getName() const
{
    return JucePlugin_Name;
}
```

```
int StktestAudioProcessor::getNumParameters()
{
    return 0;
}

float StktestAudioProcessor::getParameter (int index)
{
    return 0.0f;
}

void StktestAudioProcessor::setParameter (int index, float newValue)
{
}

const String StktestAudioProcessor::getParameterName (int index)
{
    return String::empty;
}

const String StktestAudioProcessor::getParameterText (int index)
{
    return String::empty;
}

const String StktestAudioProcessor::getInputChannelName (int channelIndex) const
{
    return String (channelIndex + 1);
}

const String StktestAudioProcessor::getOutputChannelName (int channelIndex) const
{
    return String (channelIndex + 1);
}

bool StktestAudioProcessor::isInputChannelStereoPair (int index) const
{
    return true;
}

bool StktestAudioProcessor::isOutputChannelStereoPair (int index) const
{
    return true;
}

bool StktestAudioProcessor::acceptsMidi() const
{
#if JucePlugin_WantsMidiInput
    return true;
#else
    return false;
#endif
}

bool StktestAudioProcessor::producesMidi() const
{
#if JucePlugin_ProducesMidiOutput
    return true;
#else
```

```
    return false;
#endif
}

int StktestAudioProcessor::getNumPrograms()
{
    return 0;
}

int StktestAudioProcessor::getCurrentProgram()
{
    return 0;
}

void StktestAudioProcessor::setCurrentProgram (int index)
{
}

const String StktestAudioProcessor::getProgramName (int index)
{
    return String::empty;
}

void StktestAudioProcessor::changeProgramName (int index, const String& newName)
{
}

//==============================================================================
void StktestAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    // Use this method as the place to do any pre-playback
    // initialisation that you need..
       additiveSynth.setCurrentPlaybackSampleRate(sampleRate);
       keyboardState.reset();

       stk::Stk::setSampleRate(sampleRate);
       stk::Stk::setRawwavePath("../rawwaves/");

}

void StktestAudioProcessor::releaseResources()
{
    // When playback stops, you can use this as an opportunity to free up any
    // spare memory, etc.
       keyboardState.reset();
}


void StktestAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages)
{
    // This is the place where you'd normally do the guts of your plugin's
    // audio processing...

       const int numSamples = buffer.getNumSamples();

       keyboardState.processNextMidiBuffer(midiMessages, 0, numSamples, true);
       additiveSynth.renderNextBlock(buffer, midiMessages, 0, numSamples);
```

```
      //for (channel = 0; channel < getNumInputChannels(); ++channel)
 //   {
 //       float* channelData = buffer.getSampleData (channel);
      //
 //       // ..do something to the data...
 //   }



    // In case we have more outputs than inputs, we'll clear any output
    // channels that didn't contain input data, (because these aren't
    // guaranteed to be empty - they may contain garbage).
    for (int i = getNumInputChannels(); i < getNumOutputChannels(); ++i)
    {
        buffer.clear (i, 0, buffer.getNumSamples());
    }
}

//==============================================================================
bool StktestAudioProcessor::hasEditor() const
{
    return true; // (change this to false if you choose to not supply an editor)
}

AudioProcessorEditor* StktestAudioProcessor::createEditor()
{
    return new StktestAudioProcessorEditor (this);
}

//==============================================================================
void StktestAudioProcessor::getStateInformation (MemoryBlock& destData)
{
    // You should use this method to store your parameters in the memory block.
    // You could do that either as raw data, or use the XML or ValueTree classes
    // as intermediaries to make it easy to save and load complex data.
}

void StktestAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
{
    // You should use this method to restore your parameters from this memory block,
    // whose contents will have been created by the getStateInformation() call.
}

//==============================================================================
// This creates new instances of the plugin..
AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
    return new StktestAudioProcessor();
}
```

```
/*
miniSynthEditor.h - header file for miniSynthEditor.cpp
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

  This is an automatically generated file created by the Jucer!

  Creation date:  4 Apr 2012 1:04:01am

  Be careful when adding custom code to these files, as only the code within
  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
  and re-saved.

  Jucer version: 1.12

  ------------------------------------------------------------------------------

  The Jucer is part of the JUCE library - "Jules' Utility Class Extensions"
  Copyright 2004-6 by Raw Material Software ltd.

  ==============================================================================
*/

#ifndef __JUCER_HEADER_MINISYNTHAUDIOPROCESSOREDITOR_PLUGINEDITOR_2D2EC4AF__
#define __JUCER_HEADER_MINISYNTHAUDIOPROCESSOREDITOR_PLUGINEDITOR_2D2EC4AF__

//[Headers]     -- You can add your own extra header files here --
#include "../JuceLibraryCode/JuceHeader.h"
#include "PluginProcessor.h"
//[/Headers]



//==============================================================================
/**
                                                                    //[Comments]
    An auto-generated component, created by the Jucer.

    Describe your class and how it works here!
                                                                    //[/Comments]
*/
class MiniSynthAudioProcessorEditor  : public AudioProcessorEditor,
                                       //public ComboBoxListener,
                                       public SliderListener
{
public:
    //==============================================================================
    MiniSynthAudioProcessorEditor (MiniSynthAudioProcessor* (ownerFilter));
    ~MiniSynthAudioProcessorEditor();

    //==============================================================================
    //[UserMethods]     -- You can add your own custom methods in this section.
    //[/UserMethods]

    void paint (Graphics& g);
    void resized();
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged);
    void sliderValueChanged (Slider* sliderThatWasMoved);
    //==============================================================================
    juce_UseDebuggingNewOperator

private:
    //[UserVariables]   -- You can add your own custom variables in this section.
    //[/UserVariables]

    //==============================================================================
//    ComboBox* waveBox;
//    ComboBox* octaveBox;
    Slider* levelSlider;
    Label* label;
    Label* label2;

       MiniSynthAudioProcessor* getProcessor() const
    {
        return static_cast <MiniSynthAudioProcessor*> (getAudioProcessor());
    }


    //==============================================================================
    // (prevent copy constructor and operator= being generated..)
    MiniSynthAudioProcessorEditor (const MiniSynthAudioProcessorEditor&);
    const MiniSynthAudioProcessorEditor& operator= (const MiniSynthAudioProcessorEditor&);
};


#endif   // __JUCER_HEADER_MINISYNTHAUDIOPROCESSOREDITOR_PLUGINEDITOR_2D2EC4AF__
```

```
/*
miniSynthEditor.cpp - automatically generated file for miniSynth GUI
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

  This is an automatically generated file created by the Jucer!

  Creation date:  4 Apr 2012 1:04:01am

  Be careful when adding custom code to these files, as only the code within
  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
  and re-saved.

  Jucer version: 1.12

  ------------------------------------------------------------------------------

  The Jucer is part of the JUCE library - "Jules' Utility Class Extensions"
  Copyright 2004-6 by Raw Material Software ltd.

  ==============================================================================
*/

//[Headers] You can add your own extra header files here...
#include "PluginProcessor.h"
//[/Headers]

#include "PluginEditor.h"


//[MiscUserDefs] You can add your own user definitions and misc code here...
//[/MiscUserDefs]

//==============================================================================
MiniSynthAudioProcessorEditor::MiniSynthAudioProcessorEditor (MiniSynthAudioProcessor*
(ownerFilter))
    : AudioProcessorEditor (ownerFilter),
//      waveBox (0),
//      octaveBox (0),
      levelSlider (0),
      label (0),
      label2 (0)
{
//    addAndMakeVisible (waveBox = new ComboBox (L"new combo box"));
//    waveBox->setEditableText (false);
//    waveBox->setJustificationType (Justification::centredLeft);
//    waveBox->setTextWhenNothingSelected (String::empty);
//    waveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
//    waveBox->addItem (L"Sine", 1);
//    waveBox->addItem (L"Saw", 2);
//    waveBox->addItem (L"Square", 3);
//    waveBox->addListener (this);

//    addAndMakeVisible (octaveBox = new ComboBox (L"new combo box"));
//    octaveBox->setEditableText (false);
//    octaveBox->setJustificationType (Justification::centredLeft);
//    octaveBox->setTextWhenNothingSelected (String::empty);
//    octaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
//    octaveBox->addItem (L"1", 1);
//    octaveBox->addItem (L"2", 2);
//    octaveBox->addItem (L"3", 3);
//    octaveBox->addItem (L"4", 4);
//    octaveBox->addListener (this);

    addAndMakeVisible (levelSlider = new Slider (L"new slider"));
    levelSlider->setRange (0, 10, 0);
    levelSlider->setSliderStyle (Slider::Rotary);
    levelSlider->setTextBoxStyle (Slider::NoTextBox, false, 80, 20);
    levelSlider->addListener (this);

    addAndMakeVisible (label = new Label (L"new label",
                                          L"Wave"));
    label->setFont (Font (15.0000f, Font::plain));
    label->setJustificationType (Justification::centredLeft);
    label->setEditable (false, false, false);
    label->setColour (TextEditor::textColourId, Colours::black);
    label->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label2 = new Label (L"new label",
                                           L"Octave"));
    label2->setFont (Font (15.0000f, Font::plain));
    label2->setJustificationType (Justification::centredLeft);
    label2->setEditable (false, false, false);
    label2->setColour (TextEditor::textColourId, Colours::black);
    label2->setColour (TextEditor::backgroundColourId, Colour (0x0));


    //[UserPreSize]
    //[/UserPreSize]

    setSize (300, 100);


    //[Constructor] You can add your own custom stuff here..
    //[/Constructor]
}

MiniSynthAudioProcessorEditor::~MiniSynthAudioProcessorEditor()
{
    //[Destructor_pre]. You can add your own custom destruction code here..
    //[/Destructor_pre]

//    deleteAndZero (waveBox);
//    deleteAndZero (octaveBox);
    deleteAndZero (levelSlider);
    deleteAndZero (label);
    deleteAndZero (label2);


    //[Destructor]. You can add your own custom destruction code here..
    //[/Destructor]
}

//==============================================================================
void MiniSynthAudioProcessorEditor::paint (Graphics& g)
{
    //[UserPrePaint] Add your own custom painting code here..
    //[/UserPrePaint]

    g.fillAll (Colours::white);

    //[UserPaint] Add your own custom painting code here..
    //[/UserPaint]
}

void MiniSynthAudioProcessorEditor::resized()
{
//    waveBox->setBounds (96, 16, 110, 24);
//    octaveBox->setBounds (160, 56, 46, 24);
    levelSlider->setBounds (224, 24, 60, 60);
    label->setBounds (32, 16, 56, 24);
    label2->setBounds (32, 56, 112, 24);
    //[UserResized] Add your own custom resize handling here..
    //[/UserResized]
}

//void MiniSynthAudioProcessorEditor::comboBoxChanged (ComboBox* comboBoxThatHasChanged)
//{
//    //[UsercomboBoxChanged_Pre]
//    //[/UsercomboBoxChanged_Pre]
//
//    if (comboBoxThatHasChanged == waveBox)
//    {
//        //[UserComboBoxCode_waveBox] -- add your combo box handling code here..
//            getProcessor()->setParameterNotifyingHost(MiniSynthAudioProcessor::octaveParam,
(float) waveBox->getSelectedItemIndex());
//        getProcessor()->voice1->osc1Wave = pow(2,getProcessor()-
>getParameter(MiniSynthAudioProcessor::waveParam));
//        //[/UserComboBoxCode_waveBox]
//    }
//    else if (comboBoxThatHasChanged == octaveBox)
//    {
//        //[UserComboBoxCode_octaveBox] -- add your combo box handling code here..
//            getProcessor()->setParameterNotifyingHost(MiniSynthAudioProcessor::octaveParam,
(float) octaveBox->getSelectedItemIndex());
//        getProcessor()->voice1->osc1Octave = pow(2,getProcessor()-
>getParameter(MiniSynthAudioProcessor::octaveParam));
//        //[/UserComboBoxCode_octaveBox]
//    }
//
//    //[UsercomboBoxChanged_Post]
//    //[/UsercomboBoxChanged_Post]
//}

void MiniSynthAudioProcessorEditor::sliderValueChanged (Slider* sliderThatWasMoved)
{
    //[UsersliderValueChanged_Pre]
    //[/UsersliderValueChanged_Pre]

    if (sliderThatWasMoved == levelSlider)
    {
        //[UserSliderCode_levelSlider] -- add your slider handling code here..
            getProcessor()->setParameterNotifyingHost (MiniSynthAudioProcessor::levelParam,
(float) levelSlider->getValue());
```

```
        getProcessor()->voice1->osc1Level = getProcessor()-
>getParameter(MiniSynthAudioProcessor::levelParam);
        //[/UserSliderCode_levelSlider]
    }

    //[UsersliderValueChanged_Post]
    //[/UsersliderValueChanged_Post]
}



//[MiscUserCode] You can add your own definitions of your custom methods or any other code
here...
//[/MiscUserCode]


//==============================================================================
#if 0
/*  -- Jucer information section --

    This is where the Jucer puts all of its metadata, so don't change anything in here!

BEGIN_JUCER_METADATA

<JUCER_COMPONENT documentType="Component" className="MiniSynthAudioProcessorEditor"
                 componentName="" parentClasses="public Component"
constructorParams="MiniSynthAudioProcessor* (ownerFilter)"
                 variableInitialisers="AudioProcessorEditor (ownerFilter)" snapPixels="8"
                 snapActive="1" snapShown="1" overlayOpacity="0.330000013" fixedSize="1"
                 initialWidth="300" initialHeight="100">
  <BACKGROUND backgroundColour="ffffffff"/>
  <COMBOBOX name="new combo box" id="6927baa27b4d7190" memberName="waveBox"
            virtualName="" explicitFocusOrder="0" pos="96 16 110 24" editable="0"
            layout="33" items="Sine&#10;Saw&#10;Square" textWhenNonSelected=""
            textWhenNoItems="(no choices)"/>
  <COMBOBOX name="new combo box" id="8cde068ff566198e" memberName="octaveBox"
            virtualName="" explicitFocusOrder="0" pos="160 56 46 24" editable="0"
            layout="33" items="1&#10;2&#10;3&#10;4" textWhenNonSelected=""
            textWhenNoItems="(no choices)"/>
  <SLIDER name="new slider" id="fd658d33abd04e32" memberName="levelSlider"
          virtualName="" explicitFocusOrder="0" pos="224 24 60 60" min="0"
          max="10" int="0" style="Rotary" textBoxPos="NoTextBox" textBoxEditable="1"
          textBoxWidth="80" textBoxHeight="20" skewFactor="1"/>
  <LABEL name="new label" id="bd7adefa67a2a34" memberName="label" virtualName=""
         explicitFocusOrder="0" pos="32 16 56 24" edTextCol="ff000000"
         edBkgCol="0" labelText="Wave" editableSingleClick="0" editableDoubleClick="0"
         focusDiscardsChanges="0" fontname="Default font" fontsize="15"
         bold="0" italic="0" justification="33"/>
  <LABEL name="new label" id="1961a2ddc0f9f596" memberName="label2" virtualName=""
         explicitFocusOrder="0" pos="32 56 112 24" edTextCol="ff000000"
         edBkgCol="0" labelText="Octave" editableSingleClick="0" editableDoubleClick="0"
         focusDiscardsChanges="0" fontname="Default font" fontsize="15"
         bold="0" italic="0" justification="33"/>
</JUCER_COMPONENT>

END_JUCER_METADATA
*/
#endif
```

```
/*
miniSynthProcessor.h - header file for miniSynthProcessor.cpp
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#ifndef __PLUGINPROCESSOR_H_5B699C9A__
#define __PLUGINPROCESSOR_H_5B699C9A__

#include "MiniSynth.h"
#include "../JuceLibraryCode/JuceHeader.h"


//==============================================================================
/**
*/
class MiniSynthAudioProcessor  : public AudioProcessor
{
public:
    //==============================================================================
    MiniSynthAudioProcessor();
    ~MiniSynthAudioProcessor();

    //==============================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock);
    void releaseResources();

    void processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages);

    //==============================================================================
    AudioProcessorEditor* createEditor();
    bool hasEditor() const;

    //==============================================================================
    const String getName() const;

    int getNumParameters();

    float getParameter (int index);
    void setParameter (int index, float newValue);

    const String getParameterName (int index);
    const String getParameterText (int index);

    const String getInputChannelName (int channelIndex) const;
    const String getOutputChannelName (int channelIndex) const;
    bool isInputChannelStereoPair (int index) const;
    bool isOutputChannelStereoPair (int index) const;

    bool acceptsMidi() const;
    bool producesMidi() const;
    //==============================================================================
    int getNumPrograms();
    int getCurrentProgram();
    void setCurrentProgram (int index);
    const String getProgramName (int index);
    void changeProgramName (int index, const String& newName);

    //==============================================================================
    void getStateInformation (MemoryBlock& destData);
    void setStateInformation (const void* data, int sizeInBytes);

       enum Parameters
       {
              waveParam,
              octaveParam,
              levelParam,

              totalNumParams
       };

       float wave, octave, level;
       MiniSynthVoice* voice1;

private:
    //==============================================================================
       Synthesiser MiniSynth;
       MidiKeyboardState keyboardState;

       JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (MiniSynthAudioProcessor);
};

#endif  // __PLUGINPROCESSOR_H_5B699C9A__
```

```
/*
miniSynthProcessor.cpp - test of JUCE GUI and synthesizer code structure
Copyright (C) 2012 Gabriel Olochwoszcz
  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#include "PluginProcessor.h"
#include "PluginEditor.h"


//==============================================================================
MiniSynthAudioProcessor::MiniSynthAudioProcessor()
{
       wave = 0;
       octave = 0;
       level = 0;

    for (int i = 10; --i>0;)
        MiniSynth.addVoice (new MiniSynthVoice());
       voice1 = (MiniSynthVoice*) MiniSynth.getVoice(0);
    MiniSynth.addSound(new MiniSynthSound());
}

MiniSynthAudioProcessor::~MiniSynthAudioProcessor()
{
}

//==============================================================================
const String MiniSynthAudioProcessor::getName() const
{
    return JucePlugin_Name;
}

int MiniSynthAudioProcessor::getNumParameters()
{
    return 0;
}

float MiniSynthAudioProcessor::getParameter (int index)
{
       switch (index)
       {
              case waveParam:            return wave;
              case octaveParam:   return octave;
              case levelParam:    return level;

              default:                   return 0.0f;
       }
}

void MiniSynthAudioProcessor::setParameter (int index, float newValue)
{
       switch (index)
       {
       case waveParam:                   wave = newValue; break;
       case octaveParam:          octave = newValue; break;
       case levelParam:           level = newValue; break;

       default:                          break;
       }
}

const String MiniSynthAudioProcessor::getParameterName (int index)
{
    switch (index)
       {
              case waveParam:            return "wave";
              case octaveParam:   return "octave";
              case levelParam:    return "level";

              default:                   break;
       }

       return String::empty;
}

const String MiniSynthAudioProcessor::getParameterText (int index)
{
    return String (getParameter (index), 2);
}

const String MiniSynthAudioProcessor::getInputChannelName (int channelIndex) const
{
    return String (channelIndex + 1);
}

const String MiniSynthAudioProcessor::getOutputChannelName (int channelIndex) const
{
    return String (channelIndex + 1);
}

bool MiniSynthAudioProcessor::isInputChannelStereoPair (int index) const
{
    return true;
}

bool MiniSynthAudioProcessor::isOutputChannelStereoPair (int index) const
{
    return true;
}

bool MiniSynthAudioProcessor::acceptsMidi() const
{
#if JucePlugin_WantsMidiInput
    return true;
#else
    return false;
#endif
}
```

```
bool MiniSynthAudioProcessor::producesMidi() const
{
#if JucePlugin_ProducesMidiOutput
    return true;
#else
    return false;
#endif
}

int MiniSynthAudioProcessor::getNumPrograms()
{
    return 0;
}

int MiniSynthAudioProcessor::getCurrentProgram()
{
    return 0;
}

void MiniSynthAudioProcessor::setCurrentProgram (int index)
{
}

const String MiniSynthAudioProcessor::getProgramName (int index)
{
    return String::empty;
}

void MiniSynthAudioProcessor::changeProgramName (int index, const String& newName)
{
}

//==============================================================================
void MiniSynthAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    // Use this method as the place to do any pre-playback
    // initialisation that you need..
       MiniSynth.setCurrentPlaybackSampleRate(sampleRate);
       //stk::Stk::setSampleRate(sampleRate);
       keyboardState.reset();
}

void MiniSynthAudioProcessor::releaseResources()
{
    // When playback stops, you can use this as an opportunity to free up any
    // spare memory, etc.
       keyboardState.reset();
}

void MiniSynthAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer&
midiMessages)
{
    // This is the place where you'd normally do the guts of your plugin's
    // audio processing...
//    for (int channel = 0; channel < getNumInputChannels(); ++channel)
//    {
        //float* channelData = buffer.getSampleData (channel);
        // ..do something to the data...
            const int numSamples = buffer.getNumSamples();

            keyboardState.processNextMidiBuffer(midiMessages, 0, numSamples, true);
            MiniSynth.renderNextBlock(buffer, midiMessages, 0, numSamples);
//    }

    // In case we have more outputs than inputs, we'll clear any output
    // channels that didn't contain input data, (because these aren't
    // guaranteed to be empty - they may contain garbage).
    for (int i = getNumInputChannels(); i < getNumOutputChannels(); ++i)
    {
        buffer.clear (i, 0, buffer.getNumSamples());
    }
}

//==============================================================================
bool MiniSynthAudioProcessor::hasEditor() const
{
    return true; // (change this to false if you choose to not supply an editor)
}

AudioProcessorEditor* MiniSynthAudioProcessor::createEditor()
{
    return new MiniSynthAudioProcessorEditor (this);
}

//==============================================================================
void MiniSynthAudioProcessor::getStateInformation (MemoryBlock& destData)
{
    // You should use this method to store your parameters in the memory block.
    // You could do that either as raw data, or use the XML or ValueTree classes
    // as intermediaries to make it easy to save and load complex data.
}

void MiniSynthAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
{
    // You should use this method to restore your parameters from this memory block,
    // whose contents will have been created by the getStateInformation() call.
}

//==============================================================================
// This creates new instances of the plugin..
AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
    return new MiniSynthAudioProcessor();
}
```

```
/*
miniSynth.h - header file for miniSynth.cpp
Copyright (C) 2012 Gabriel Olochwoszcz
*/

#include "../JuceLibraryCode/JuceHeader.h"
#include "StkIncludes.h"
#include "additiveSynthOscillator.h"

class MiniSynthSound : public SynthesiserSound
{
public:
    MiniSynthSound();

    bool appliesToNote(const int /*midiNoteNumber*/);
    bool appliesToChannel(const int /*midiChannel*/);
};

class MiniSynthVoice : public SynthesiserVoice
{
public:
    MiniSynthVoice();
    ~MiniSynthVoice();

    bool canPlaySound (SynthesiserSound* sound);

    void startNote (const int midiNoteNumber, const float velocity, SynthesiserSound*
/*sound*/, const int /*currrentPitchWheelPosition*/);

    void stopNote (const bool allowTailOff);

    void pitchWheelMoved (const int /*newValue*/);

    void controllerMoved (const int /*controllerNumber*/, const int /*newValue*/);

    void renderNextBlock (AudioSampleBuffer& outputBuffer, int startSample, int numSamples);

       int osc1Wave, osc1Octave;
       float osc1Level;


private:
       additiveSynthOscillator oscillator0;
       additiveSynthOscillator oscillator1;
       additiveSynthOscillator oscillator2;

       //AudioSampleBuffer& processBlock;
       //AudioSampleBuffer& outputBlock;



    double freq, freq1;
       double keyLevel;
       float outputLevel;
};
```

```
/*
miniSynth.cpp - JUCE SynthesiserSound and SynthesiserVoice objects for miniSynth project
Copyright (C) 2012 Gabriel Olochwoszcz
*/

#include "MiniSynth.h"

//additiveSynthOscillator oscillators[3];

MiniSynthSound::MiniSynthSound()
{
}

bool MiniSynthSound::appliesToNote(const int /*midiNoteNumber*/)
{
       return true;
}
bool MiniSynthSound::appliesToChannel(const int /*midiChannel*/)
{
       return true;
}

//==========================================================================
MiniSynthVoice::MiniSynthVoice()
{
}

MiniSynthVoice::~MiniSynthVoice()
{
}

bool MiniSynthVoice::canPlaySound (SynthesiserSound* sound)
{
       return dynamic_cast <MiniSynthSound*> (sound) != 0;
}

void MiniSynthVoice::startNote (const int midiNoteNumber, const float velocity,
SynthesiserSound* /*sound*/, const int /*currrentPitchWheelPosition*/)
{
       freq = MidiMessage::getMidiNoteInHertz(midiNoteNumber);
       //outputLevel = osc1Level*velocity/10;
       //freq1 = freq*osc1Octave;
       //sine1Out->setFrequency(freq);

       //oscillators[0].noteOn(freq,1,.2,0);
       //oscillators[1].noteOn(freq,1,.2,0);
       //oscillators[2].noteOn(freq,1,.2,0);
       oscillator0.noteOn(freq,3,.2,0);
       oscillator1.noteOn(freq,2,.2,1);
       oscillator2.noteOn(freq,1,.2,2);
       //maybe check if each oscLevel > 0
       //for each oscillator
       //switch (osc1Wave)
       //{
       //     case 0: sine1Out->setFrequency(freq1);          break;
       //     //case 1: saw1Out->setFrequency(freq1);         break;
       //     //case 2: square1Out->setFrequency(freq1);      break;
       //       default: sine1Out->setFrequency(freq1);              break;
```

```
        //   }
}

void MiniSynthVoice::stopNote (const bool allowTailOff)
{

        //oscillators[0].noteOff(0);
        //oscillators[1].noteOff(0);
        //oscillators[2].noteOff(0);
        oscillator0.noteOff(0);
        oscillator1.noteOff(0);
        oscillator2.noteOff(0);
        outputLevel = 0;
        clearCurrentNote();
}

void MiniSynthVoice::pitchWheelMoved (const int /*newValue*/)
{
}

void MiniSynthVoice::controllerMoved (const int /*controllerNumber*/, const int /*newValue*/)
{
}

void MiniSynthVoice::renderNextBlock (AudioSampleBuffer& outputBuffer, int startSample, int
numSamples)
{
        float* osc1Output = new float[numSamples];


        for (int currentSample = 0; currentSample < numSamples; currentSample++)
        {
                //osc1Output[currentSample] = (float) oscillators[0].tick() + (float)
oscillators[1].tick() + (float) oscillators[2].tick();
                osc1Output[currentSample] = (float) oscillator0.tick() + (float)
oscillator1.tick() + (float) oscillator2.tick();
        }

        for (int numChannels = outputBuffer.getNumChannels(); --numChannels >= 0;)
        {
                outputBuffer.addFrom(numChannels, startSample, osc1Output, numSamples, 0.5f);
        }

        delete [] osc1Output;

        //    while (--numSamples >= 0)
        //    {
        //        for (int numChannels = outputBuffer.getNumChannels(); --numChannels >= 0;)
        //        {
        //            *outputBuffer.getSampleData (numChannels, startSample) += (float)
sine1Out->tick();
        //            //outputBuffer.applyGain(startSample, numSamples, outputLevel);
        //        }
        //    }
        //    ++startSample;


}
```

```
/*
vstSynthEditor.h - header file for vstSynthEditor.cpp
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
  ==============================================================================

  This is an automatically generated file created by the Jucer!

  Creation date:  26 Mar 2012 12:35:47am

  Be careful when adding custom code to these files, as only the code within
  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
  and re-saved.

  Jucer version: 1.12

  ------------------------------------------------------------------------------

  The Jucer is part of the JUCE library - "Jules' Utility Class Extensions"
  Copyright 2004-6 by Raw Material Software ltd.

  ==============================================================================
*/

#ifndef __JUCER_HEADER_vstSynthAUDIOPROCESSOREDITOR_PLUGINEDITOR_11520947__
#define __JUCER_HEADER_vstSynthAUDIOPROCESSOREDITOR_PLUGINEDITOR_11520947__

//[Headers]     -- You can add your own extra header files here --
#include "../JuceLibraryCode/JuceHeader.h"
#include "vstSynthProcessor.h"
//[/Headers]



//==============================================================================
/**
                                                                    //[Comments]
    An auto-generated component, created by the Jucer.

    Describe your class and how it works here!
                                                                    //[/Comments]
*/
```

```
class vstSynthAudioProcessorEditor  : public AudioProcessorEditor,
                                          public SliderListener,
                                          public ComboBoxListener//,
//                                                                     public Timer
{
public:
    //==============================================================================
    vstSynthAudioProcessorEditor (vstSynthAudioProcessor* ownerFilter);
    ~vstSynthAudioProcessorEditor();

    //==============================================================================
    //[UserMethods]     -- You can add your own custom methods in this section.
    //[/UserMethods]

//    void timerCallback();
    void paint (Graphics& g);
    void resized();
    void sliderValueChanged (Slider* sliderThatWasMoved);
    void comboBoxChanged (ComboBox* comboBoxThatHasChanged);



    //==============================================================================
    juce_UseDebuggingNewOperator

private:
    //[UserVariables]   -- You can add your own custom variables in this section.
    //[/UserVariables]

    //==============================================================================
    GroupComponent* filterGroup;
    Slider* filterGainSlider;
    GroupComponent* lfoGroup;
    Slider* lfoDevSlider;
    GroupComponent* oscGroup;
    Label* label21;
    GroupComponent* envelopeGroup;
    Slider* sustainSlider;
    Slider* osc1lvlSlider;
    GroupComponent* delayGroup;
    Slider* delayFBSlider;
    Slider* delayGainSlider;
    GroupComponent* outputGroup;
    Label* label8;
    Slider* decaySlider;
    Slider* attackSlider;
    Slider* delayTimeSlider;
    Label* label;
    Label* label2;
    Label* label3;
    Label* label4;
    Label* label5;
    Label* label6;
    Label* label7;
    Slider* noiseSlider;
    Slider* driveSlider;
    Slider* outputGainSlider;
    Label* label11;
    Label* label12;
    Label* label13;
    Slider* releaseSlider;
    ComboBox* osc3WaveBox;
    ComboBox* osc3OctaveBox;
    Label* label23;
    Label* label24;
    Label* label25;
    Label* label27;
    Label* label29;
    Slider* filterFreqSlider;
    Label* label31;
    Label* label32;
    Label* label33;
    Slider* lfoFreqSlider;
    ComboBox* lfoWaveBox;
    ComboBox* lfoDestBox;
    Label* label9;
    Slider* osc3lvlSlider;
    Slider* filterResSlider;
    ComboBox* filterTypeBox;
    Label* label10;
    Label* label26;
    ComboBox* osc2WaveBox;
    ComboBox* osc2OctaveBox;
    Label* label14;
    Label* label15;
    Label* label16;
    Slider* osc2lvlSlider;
    Label* label17;
    ComboBox* osc1WaveBox;
    ComboBox* osc1OctaveBox;
    Label* label18;
    Label* label19;
    Label* label20;

    vstSynthAudioProcessor* getProcessor() const
    {
        return static_cast <vstSynthAudioProcessor*> (getAudioProcessor());
    }


    //==============================================================================
    // (prevent copy constructor and operator= being generated..)
    vstSynthAudioProcessorEditor (const vstSynthAudioProcessorEditor&);
    const vstSynthAudioProcessorEditor& operator= (const vstSynthAudioProcessorEditor&);
};


#endif   // __JUCER_HEADER_vstSynthAUDIOPROCESSOREDITOR_PLUGINEDITOR_11520947__
```

```
/*
vstSynthEditor.cpp - synth interface generated from JUCE
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
  ==============================================================================

  This is an automatically generated file created by the Jucer!

  Creation date:  26 Mar 2012 12:35:47am

  Be careful when adding custom code to these files, as only the code within
  the "//[xyz]" and "//[/xyz]" sections will be retained when the file is loaded
  and re-saved.

  Jucer version: 1.12

  ------------------------------------------------------------------------------

  The Jucer is part of the JUCE library - "Jules' Utility Class Extensions"
  Copyright 2004-6 by Raw Material Software ltd.

  ==============================================================================
*/

//[Headers] You can add your own extra header files here...
//[/Headers]

#include "vstSynthProcessor.h"
#include "vstSynthEditor.h"


//[MiscUserDefs] You can add your own user definitions and misc code here...
//[/MiscUserDefs]


//==============================================================================
vstSynthAudioProcessorEditor::vstSynthAudioProcessorEditor (vstSynthAudioProcessor*
ownerFilter)
    : AudioProcessorEditor (ownerFilter),
      filterGroup (0),
      filterGainSlider (0),
      lfoGroup (0),
      lfoDevSlider (0),
      oscGroup (0),
      label21 (0),
      envelopeGroup (0),
      sustainSlider (0),
      osc1lvlSlider (0),
      delayGroup (0),
      delayFBSlider (0),
      delayGainSlider (0),
      outputGroup (0),
      label8 (0),
      decaySlider (0),
      attackSlider (0),
      delayTimeSlider (0),
      label (0),
      label2 (0),
      label3 (0),
      label4 (0),
      label5 (0),
      label6 (0),
      label7 (0),
      noiseSlider (0),
      driveSlider (0),
      outputGainSlider (0),
      label11 (0),
      label12 (0),
      label13 (0),
      releaseSlider (0),
      osc3WaveBox (0),
      osc3OctaveBox (0),
      label23 (0),
      label24 (0),
      label25 (0),
      label27 (0),
      label29 (0),
      filterFreqSlider (0),
      label31 (0),
      label32 (0),
      label33 (0),
      lfoFreqSlider (0),
      lfoWaveBox (0),
      lfoDestBox (0),
      label9 (0),
      osc3lvlSlider (0),
      filterResSlider (0),
      filterTypeBox (0),
      label10 (0),
      label26 (0),
      osc2WaveBox (0),
      osc2OctaveBox (0),
      label14 (0),
      label15 (0),
      label16 (0),
      osc2lvlSlider (0),
      label17 (0),
      osc1WaveBox (0),
      osc1OctaveBox (0),
      label18 (0),
      label19 (0),
      label20 (0)
{
    addAndMakeVisible (filterGroup = new GroupComponent (L"new group",
                                                          L"Filter"));
    filterGroup->setColour (GroupComponent::outlineColourId, Colour (0x88c6159a));
    filterGroup->setColour (GroupComponent::textColourId, Colours::white);

    addAndMakeVisible (filterGainSlider = new Slider (L"new slider"));
//    filterGainSlider->setRange (0, 2, 0.01);
    filterGainSlider->setRange (-20, 20, 0.5);
    filterGainSlider->setSliderStyle (Slider::Rotary);
    filterGainSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    filterGainSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc6159a));
    filterGainSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    filterGainSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    filterGainSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    filterGainSlider->setColour (Slider::textBoxHighlightColourId, Colour (0x1111ee));
    filterGainSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    filterGainSlider->addListener (this);

    addAndMakeVisible (lfoGroup = new GroupComponent (L"new group",
                                                       L"LFO"));
    lfoGroup->setTextLabelPosition (Justification::centredLeft);
    lfoGroup->setColour (GroupComponent::outlineColourId, Colour (0x8874c615));
    lfoGroup->setColour (GroupComponent::textColourId, Colours::white);

    addAndMakeVisible (lfoDevSlider = new Slider (L"new slider"));
    lfoDevSlider->setRange (0, 10, 0.1);
    lfoDevSlider->setSliderStyle (Slider::Rotary);
    lfoDevSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    lfoDevSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xff74c615));
    lfoDevSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    lfoDevSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    lfoDevSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    lfoDevSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    lfoDevSlider->addListener (this);

    addAndMakeVisible (oscGroup = new GroupComponent (L"new group",
                                                       L"Oscillators"));
    oscGroup->setColour (GroupComponent::outlineColourId, Colour (0x88c67e15));
    oscGroup->setColour (GroupComponent::textColourId, Colours::white);

    addAndMakeVisible (label21 = new Label (L"new label",
                                            L"Oscillator 1"));
    label21->setFont (Font (15.0000f, Font::bold));
    label21->setJustificationType (Justification::centred);
    label21->setEditable (false, false, false);
    label21->setColour (Label::textColourId, Colours::white);
    label21->setColour (TextEditor::textColourId, Colours::black);
    label21->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (envelopeGroup = new GroupComponent (L"new group",
                                                            L"Envelope"));
    envelopeGroup->setColour (GroupComponent::outlineColourId, Colour (0x88158ec6));
    envelopeGroup->setColour (GroupComponent::textColourId, Colours::white);

    addAndMakeVisible (sustainSlider = new Slider (L"new slider"));
    sustainSlider->setRange (0, 10, 0.1);
    sustainSlider->setSliderStyle (Slider::Rotary);
    sustainSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    sustainSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xff158ec6));
    sustainSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    sustainSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    sustainSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    sustainSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    sustainSlider->addListener (this);

    addAndMakeVisible (osc1lvlSlider = new Slider (L"new slider"));
//    osc1lvlSlider->setRange (0, 2, 0.01);
    osc1lvlSlider->setRange (-20, 20, 0.5);
    osc1lvlSlider->setSliderStyle (Slider::Rotary);
    osc1lvlSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    osc1lvlSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc67e15));
    osc1lvlSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    osc1lvlSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    osc1lvlSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    osc1lvlSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    osc1lvlSlider->addListener (this);

    addAndMakeVisible (delayGroup = new GroupComponent (L"new group",
                                                         L"Delay"));
    delayGroup->setColour (GroupComponent::outlineColourId, Colour (0x88c1b115));
    delayGroup->setColour (GroupComponent::textColourId, Colours::white);

    addAndMakeVisible (delayFBSlider = new Slider (L"new slider"));
//    delayFBSlider->setRange (0, 1, 0.01);
    delayFBSlider->setRange (-40, 0, 0.5);
    delayFBSlider->setSliderStyle (Slider::Rotary);
    delayFBSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    delayFBSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc1b115));
    delayFBSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    delayFBSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    delayFBSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xfaf7f7));
    delayFBSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    delayFBSlider->addListener (this);

    addAndMakeVisible (delayGainSlider = new Slider (L"new slider"));
//    delayGainSlider->setRange (0, 1, 0.01);
    delayGainSlider->setRange (-40, 0, 0.5);
    delayGainSlider->setSliderStyle (Slider::Rotary);
    delayGainSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    delayGainSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc1b115));
    delayGainSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    delayGainSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    delayGainSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    delayGainSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    delayGainSlider->addListener (this);

    addAndMakeVisible (outputGroup = new GroupComponent (L"new group",
                                                          L"Output"));
    outputGroup->setColour (GroupComponent::outlineColourId, Colour (0x88c61515));
    outputGroup->setColour (GroupComponent::textColourId, Colours::white);

    addAndMakeVisible (label8 = new Label (L"new label",
                                           L"Destination"));
```

```
label8->setFont (Font (15.0000f, Font::plain));
label8->setJustificationType (Justification::centredRight);
label8->setEditable (false, false, false);
label8->setColour (Label::backgroundColourId, Colour (0xb70101));
label8->setColour (Label::textColourId, Colours::white);
label8->setColour (TextEditor::textColourId, Colours::black);
label8->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (decaySlider = new Slider (L"new slider"));
decaySlider->setRange (0, 10, 0.1);
decaySlider->setSliderStyle (Slider::Rotary);
decaySlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
decaySlider->setColour (Slider::rotarySliderFillColourId, Colour (0xff158ec6));
decaySlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
decaySlider->setColour (Slider::textBoxTextColourId, Colours::white);
decaySlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
decaySlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
decaySlider->addListener (this);

addAndMakeVisible (attackSlider = new Slider (L"new slider"));
attackSlider->setRange (0, 10, 0.1);
attackSlider->setSliderStyle (Slider::Rotary);
attackSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
attackSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xff158ec6));
attackSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
attackSlider->setColour (Slider::textBoxTextColourId, Colours::white);
attackSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
attackSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
attackSlider->addListener (this);

addAndMakeVisible (delayTimeSlider = new Slider (L"new slider"));
delayTimeSlider->setRange (0, 1, 0.001);
delayTimeSlider->setSliderStyle (Slider::Rotary);
delayTimeSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
delayTimeSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc1b115));
delayTimeSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
delayTimeSlider->setColour (Slider::textBoxTextColourId, Colours::white);
delayTimeSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
delayTimeSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
delayTimeSlider->addListener (this);

addAndMakeVisible (label = new Label (L"new label",
                                      L"Attack"));
label->setFont (Font (15.0000f, Font::bold));
label->setJustificationType (Justification::centred);
label->setEditable (false, false, false);
label->setColour (Label::textColourId, Colours::white);
label->setColour (TextEditor::textColourId, Colours::black);
label->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label2 = new Label (L"new label",
                                       L"Sustain"));
label2->setFont (Font (15.0000f, Font::bold));
label2->setJustificationType (Justification::centred);
label2->setEditable (false, false, false);
label2->setColour (Label::textColourId, Colours::white);
label2->setColour (TextEditor::textColourId, Colours::black);
label2->setColour (TextEditor::backgroundColourId, Colour (0x0));
```

```
    addAndMakeVisible (label3 = new Label (L"new label",
                                           L"Decay"));
    label3->setFont (Font (15.0000f, Font::bold));
    label3->setJustificationType (Justification::centred);
    label3->setEditable (false, false, false);
    label3->setColour (Label::textColourId, Colours::white);
    label3->setColour (TextEditor::textColourId, Colours::black);
    label3->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label4 = new Label (L"new label",
                                           L"Release"));
    label4->setFont (Font (15.0000f, Font::bold));
    label4->setJustificationType (Justification::centred);
    label4->setEditable (false, false, false);
    label4->setColour (Label::textColourId, Colours::white);
    label4->setColour (TextEditor::textColourId, Colours::black);
    label4->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label5 = new Label (L"new label",
                                           L"Feedback"));
    label5->setFont (Font (15.0000f, Font::bold));
    label5->setJustificationType (Justification::centred);
    label5->setEditable (false, false, false);
    label5->setColour (Label::textColourId, Colours::white);
    label5->setColour (TextEditor::textColourId, Colours::black);
    label5->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label6 = new Label (L"new label",
                                           L"Gain"));
    label6->setFont (Font (15.0000f, Font::bold));
    label6->setJustificationType (Justification::centred);
    label6->setEditable (false, false, false);
    label6->setColour (Label::textColourId, Colours::white);
    label6->setColour (TextEditor::textColourId, Colours::black);
    label6->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label7 = new Label (L"new label",
                                           L"Time"));
    label7->setFont (Font (15.0000f, Font::bold));
    label7->setJustificationType (Justification::centred);
    label7->setEditable (false, false, false);
    label7->setColour (Label::textColourId, Colours::white);
    label7->setColour (TextEditor::textColourId, Colours::black);
    label7->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (noiseSlider = new Slider (L"new slider"));
//    noiseSlider->setRange (0, 0.01, 0.0001);
    noiseSlider->setRange (-40, -0, 0.5);
    noiseSlider->setSliderStyle (Slider::Rotary);
    noiseSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    noiseSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc61515));
    noiseSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    noiseSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    noiseSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    noiseSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    noiseSlider->addListener (this);
    addAndMakeVisible (driveSlider = new Slider (L"new slider"));
    driveSlider->setRange (1, 10, 0.05);
    driveSlider->setSliderStyle (Slider::Rotary);
    driveSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    driveSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc61515));
    driveSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    driveSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    driveSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xc61515));
    driveSlider->setColour (Slider::textBoxOutlineColourId, Colour (0xc61515));
    driveSlider->addListener (this);

    addAndMakeVisible (outputGainSlider = new Slider (L"new slider"));
//    outputGainSlider->setRange (0, 2, 0.01);
    outputGainSlider->setRange (-20, 20, 0.5);
    outputGainSlider->setSliderStyle (Slider::Rotary);
    outputGainSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    outputGainSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc61515));
    outputGainSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    outputGainSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    outputGainSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    outputGainSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    outputGainSlider->addListener (this);

    addAndMakeVisible (label11 = new Label (L"new label",
                                            L"Gain"));
    label11->setFont (Font (15.0000f, Font::bold));
    label11->setJustificationType (Justification::centred);
    label11->setEditable (false, false, false);
    label11->setColour (Label::textColourId, Colours::white);
    label11->setColour (TextEditor::textColourId, Colours::black);
    label11->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label12 = new Label (L"new label",
                                            L"Drive"));
    label12->setFont (Font (15.0000f, Font::bold));
    label12->setJustificationType (Justification::centred);
    label12->setEditable (false, false, false);
    label12->setColour (Label::textColourId, Colours::white);
    label12->setColour (TextEditor::textColourId, Colours::black);
    label12->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label13 = new Label (L"new label",
                                            L"Noise"));
    label13->setFont (Font (15.0000f, Font::bold));
    label13->setJustificationType (Justification::centred);
    label13->setEditable (false, false, false);
    label13->setColour (Label::textColourId, Colours::white);
    label13->setColour (TextEditor::textColourId, Colours::black);
    label13->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (releaseSlider = new Slider (L"new slider"));
    releaseSlider->setRange (0, 10, 0.1);
    releaseSlider->setSliderStyle (Slider::Rotary);
    releaseSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    releaseSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xff158ec6));
    releaseSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    releaseSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    releaseSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
```

```
releaseSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
releaseSlider->addListener (this);

addAndMakeVisible (osc3WaveBox = new ComboBox (L"new combo box"));
osc3WaveBox->setEditableText (false);
osc3WaveBox->setJustificationType (Justification::centredLeft);
osc3WaveBox->setTextWhenNothingSelected (String::empty);
osc3WaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
osc3WaveBox->addItem (L"Sine", 1);
osc3WaveBox->addItem (L"Saw", 2);
osc3WaveBox->addItem (L"Square", 3);
osc3WaveBox->addListener (this);

addAndMakeVisible (osc3OctaveBox = new ComboBox (L"new combo box"));
osc3OctaveBox->setEditableText (false);
osc3OctaveBox->setJustificationType (Justification::centredLeft);
osc3OctaveBox->setTextWhenNothingSelected (String::empty);
osc3OctaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
osc3OctaveBox->addItem (L"1", 1);
osc3OctaveBox->addItem (L"2", 2);
osc3OctaveBox->addItem (L"3", 3);
osc3OctaveBox->addItem (L"4", 4);
osc3OctaveBox->addListener (this);

addAndMakeVisible (label23 = new Label (L"new label",
                                        L"Octave"));
label23->setFont (Font (15.0000f, Font::plain));
label23->setJustificationType (Justification::centredRight);
label23->setEditable (false, false, false);
label23->setColour (Label::textColourId, Colours::white);
label23->setColour (TextEditor::textColourId, Colours::black);
label23->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label24 = new Label (L"new label",
                                        L"Waveform"));
label24->setFont (Font (15.0000f, Font::plain));
label24->setJustificationType (Justification::centredRight);
label24->setEditable (false, false, false);
label24->setColour (Label::textColourId, Colours::white);
label24->setColour (TextEditor::textColourId, Colours::black);
label24->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label25 = new Label (L"new label",
                                        L"Level"));
label25->setFont (Font (15.0000f, Font::bold));
label25->setJustificationType (Justification::centred);
label25->setEditable (false, false, false);
label25->setColour (Label::textColourId, Colours::white);
label25->setColour (TextEditor::textColourId, Colours::black);
label25->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label27 = new Label (L"new label",
                                        L"Speed"));
label27->setFont (Font (15.0000f, Font::bold));
label27->setJustificationType (Justification::centred);
label27->setEditable (false, false, false);
label27->setColour (Label::textColourId, Colours::white);
label27->setColour (TextEditor::textColourId, Colours::black);
```

```
label27->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label29 = new Label (L"new label",
                                        L"Depth"));
label29->setFont (Font (15.0000f, Font::bold));
label29->setJustificationType (Justification::centred);
label29->setEditable (false, false, false);
label29->setColour (Label::textColourId, Colours::white);
label29->setColour (TextEditor::textColourId, Colours::black);
label29->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (filterFreqSlider = new Slider (L"new slider"));
filterFreqSlider->setRange (50, 11025, 5);
filterFreqSlider->setSliderStyle (Slider::Rotary);
filterFreqSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
filterFreqSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc6159a));
filterFreqSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
filterFreqSlider->setColour (Slider::textBoxTextColourId, Colours::white);
filterFreqSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
filterFreqSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
filterFreqSlider->addListener (this);

addAndMakeVisible (label31 = new Label (L"new label",
                                        L"Resonance"));
label31->setFont (Font (15.0000f, Font::bold));
label31->setJustificationType (Justification::centred);
label31->setEditable (false, false, false);
label31->setColour (Label::textColourId, Colours::white);
label31->setColour (TextEditor::textColourId, Colours::black);
label31->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label32 = new Label (L"new label",
                                        L"Frequency"));
label32->setFont (Font (15.0000f, Font::bold));
label32->setJustificationType (Justification::centred);
label32->setEditable (false, false, false);
label32->setColour (Label::textColourId, Colours::white);
label32->setColour (TextEditor::textColourId, Colours::black);
label32->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label33 = new Label (L"new label",
                                        L"Gain"));
label33->setFont (Font (15.0000f, Font::bold));
label33->setJustificationType (Justification::centred);
label33->setEditable (false, false, false);
label33->setColour (Label::textColourId, Colours::white);
label33->setColour (TextEditor::textColourId, Colours::black);
label33->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (lfoFreqSlider = new Slider (L"new slider"));
lfoFreqSlider->setRange (0, 10, 0.1);
lfoFreqSlider->setSliderStyle (Slider::Rotary);
lfoFreqSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
lfoFreqSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xff74c615));
lfoFreqSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
lfoFreqSlider->setColour (Slider::textBoxTextColourId, Colours::white);
lfoFreqSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
lfoFreqSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
```

```
    lfoFreqSlider->addListener (this);

    addAndMakeVisible (lfoWaveBox = new ComboBox (L"new combo box"));
    lfoWaveBox->setEditableText (false);
    lfoWaveBox->setJustificationType (Justification::centredLeft);
    lfoWaveBox->setTextWhenNothingSelected (String::empty);
    lfoWaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
    lfoWaveBox->addItem (L"Sine", 1);
    lfoWaveBox->addItem (L"Saw", 2);
    lfoWaveBox->addItem (L"Square", 3);
    lfoWaveBox->addListener (this);

    addAndMakeVisible (lfoDestBox = new ComboBox (L"new combo box"));
    lfoDestBox->setEditableText (false);
    lfoDestBox->setJustificationType (Justification::centredLeft);
    lfoDestBox->setTextWhenNothingSelected (String::empty);
    lfoDestBox->setTextWhenNoChoicesAvailable (L"(no choices)");
    lfoDestBox->addItem (L"Off", 1);
    lfoDestBox->addItem (L"Vibrato", 2);
    lfoDestBox->addItem (L"Tremolo", 3);
    lfoDestBox->addItem (L"Envelope", 4);
    lfoDestBox->addListener (this);

    addAndMakeVisible (label9 = new Label (L"new label",
                                           L"Waveform"));
    label9->setFont (Font (15.0000f, Font::plain));
    label9->setJustificationType (Justification::centredRight);
    label9->setEditable (false, false, false);
    label9->setColour (Label::textColourId, Colours::white);
    label9->setColour (TextEditor::textColourId, Colours::black);
    label9->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (osc3lvlSlider = new Slider (L"new slider"));
//    osc3lvlSlider->setRange (0, 2, 0.01);
    osc3lvlSlider->setRange (-20, 20, 0.5);
    osc3lvlSlider->setSliderStyle (Slider::Rotary);
    osc3lvlSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    osc3lvlSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc67e15));
    osc3lvlSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    osc3lvlSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    osc3lvlSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    osc3lvlSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808000));
    osc3lvlSlider->addListener (this);

    addAndMakeVisible (filterResSlider = new Slider (L"new slider"));
//    filterResSlider->setRange (1, 100, 1);
    filterResSlider->setRange (0, 40, 1);
    filterResSlider->setSliderStyle (Slider::Rotary);
    filterResSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    filterResSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc6159a));
    filterResSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    filterResSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    filterResSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    filterResSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    filterResSlider->addListener (this);

    addAndMakeVisible (filterTypeBox = new ComboBox (L"new combo box"));
    filterTypeBox->setEditableText (false);
```

```
filterTypeBox->setJustificationType (Justification::centredLeft);
filterTypeBox->setTextWhenNothingSelected (String::empty);
filterTypeBox->setTextWhenNoChoicesAvailable (L"(no choices)");
filterTypeBox->addItem (L"Low Pass", 1);
filterTypeBox->addItem (L"Band Pass", 2);
filterTypeBox->addItem (L"High Pass", 3);
filterTypeBox->addListener (this);

addAndMakeVisible (label10 = new Label (L"new label",
                                        L"Type"));
label10->setFont (Font (15.0000f, Font::plain));
label10->setJustificationType (Justification::centredLeft);
label10->setEditable (false, false, false);
label10->setColour (Label::textColourId, Colours::white);
label10->setColour (TextEditor::textColourId, Colours::black);
label10->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label26 = new Label (L"new label",
                                        L"Oscillator 3"));
label26->setFont (Font (15.0000f, Font::bold));
label26->setJustificationType (Justification::centred);
label26->setEditable (false, false, false);
label26->setColour (Label::textColourId, Colours::white);
label26->setColour (TextEditor::textColourId, Colours::black);
label26->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (osc2WaveBox = new ComboBox (L"new combo box"));
osc2WaveBox->setEditableText (false);
osc2WaveBox->setJustificationType (Justification::centredLeft);
osc2WaveBox->setTextWhenNothingSelected (String::empty);
osc2WaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
osc2WaveBox->addItem (L"Sine", 1);
osc2WaveBox->addItem (L"Saw", 2);
osc2WaveBox->addItem (L"Square", 3);
osc2WaveBox->addListener (this);

addAndMakeVisible (osc2OctaveBox = new ComboBox (L"new combo box"));
osc2OctaveBox->setEditableText (false);
osc2OctaveBox->setJustificationType (Justification::centredLeft);
osc2OctaveBox->setTextWhenNothingSelected (String::empty);
osc2OctaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
osc2OctaveBox->addItem (L"1", 1);
osc2OctaveBox->addItem (L"2", 2);
osc2OctaveBox->addItem (L"3", 3);
osc2OctaveBox->addItem (L"4", 4);
osc2OctaveBox->addListener (this);

addAndMakeVisible (label14 = new Label (L"new label",
                                        L"Octave"));
label14->setFont (Font (15.0000f, Font::plain));
label14->setJustificationType (Justification::centredRight);
label14->setEditable (false, false, false);
label14->setColour (Label::textColourId, Colours::white);
label14->setColour (TextEditor::textColourId, Colours::black);
label14->setColour (TextEditor::backgroundColourId, Colour (0x0));

addAndMakeVisible (label15 = new Label (L"new label",
                                        L"Waveform"));
```

```
    label15->setFont (Font (15.0000f, Font::plain));
    label15->setJustificationType (Justification::centredRight);
    label15->setEditable (false, false, false);
    label15->setColour (Label::textColourId, Colours::white);
    label15->setColour (TextEditor::textColourId, Colours::black);
    label15->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label16 = new Label (L"new label",
                                            L"Level"));
    label16->setFont (Font (15.0000f, Font::bold));
    label16->setJustificationType (Justification::centred);
    label16->setEditable (false, false, false);
    label16->setColour (Label::textColourId, Colours::white);
    label16->setColour (TextEditor::textColourId, Colours::black);
    label16->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (osc2lvlSlider = new Slider (L"new slider"));
//    osc2lvlSlider->setRange (0, 2, 0.01);
    osc2lvlSlider->setRange (-20, 20, 0.5);
    osc2lvlSlider->setSliderStyle (Slider::Rotary);
    osc2lvlSlider->setTextBoxStyle (Slider::TextBoxAbove, true, 80, 20);
    osc2lvlSlider->setColour (Slider::rotarySliderFillColourId, Colour (0xffc67e15));
    osc2lvlSlider->setColour (Slider::rotarySliderOutlineColourId, Colours::grey);
    osc2lvlSlider->setColour (Slider::textBoxTextColourId, Colours::white);
    osc2lvlSlider->setColour (Slider::textBoxBackgroundColourId, Colour (0xffffff));
    osc2lvlSlider->setColour (Slider::textBoxOutlineColourId, Colour (0x808080));
    osc2lvlSlider->addListener (this);

    addAndMakeVisible (label17 = new Label (L"new label",
                                            L"Oscillator 2"));
    label17->setFont (Font (15.0000f, Font::bold));
    label17->setJustificationType (Justification::centred);
    label17->setEditable (false, false, false);
    label17->setColour (Label::textColourId, Colours::white);
    label17->setColour (TextEditor::textColourId, Colours::black);
    label17->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (osc1WaveBox = new ComboBox (L"new combo box"));
    osc1WaveBox->setEditableText (false);
    osc1WaveBox->setJustificationType (Justification::centredLeft);
    osc1WaveBox->setTextWhenNothingSelected (String::empty);
    osc1WaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
    osc1WaveBox->addItem (L"Sine", 1);
    osc1WaveBox->addItem (L"Saw", 2);
    osc1WaveBox->addItem (L"Square", 3);
    osc1WaveBox->addListener (this);

    addAndMakeVisible (osc1OctaveBox = new ComboBox (L"new combo box"));
    osc1OctaveBox->setEditableText (false);
    osc1OctaveBox->setJustificationType (Justification::centredLeft);
    osc1OctaveBox->setTextWhenNothingSelected (String::empty);
    osc1OctaveBox->setTextWhenNoChoicesAvailable (L"(no choices)");
    osc1OctaveBox->addItem (L"1", 1);
    osc1OctaveBox->addItem (L"2", 2);
    osc1OctaveBox->addItem (L"3", 3);
    osc1OctaveBox->addItem (L"4", 4);
    osc1OctaveBox->addListener (this);
    addAndMakeVisible (label18 = new Label (L"new label",
                                            L"Octave"));
    label18->setFont (Font (15.0000f, Font::plain));
    label18->setJustificationType (Justification::centredRight);
    label18->setEditable (false, false, false);
    label18->setColour (Label::textColourId, Colours::white);
    label18->setColour (TextEditor::textColourId, Colours::black);
    label18->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label19 = new Label (L"new label",
                                            L"Waveform"));
    label19->setFont (Font (15.0000f, Font::plain));
    label19->setJustificationType (Justification::centredRight);
    label19->setEditable (false, false, false);
    label19->setColour (Label::textColourId, Colours::white);
    label19->setColour (TextEditor::textColourId, Colours::black);
    label19->setColour (TextEditor::backgroundColourId, Colour (0x0));

    addAndMakeVisible (label20 = new Label (L"new label",
                                            L"Level"));
    label20->setFont (Font (15.0000f, Font::bold));
    label20->setJustificationType (Justification::centred);
    label20->setEditable (false, false, false);
    label20->setColour (Label::textColourId, Colours::white);
    label20->setColour (TextEditor::textColourId, Colours::black);
    label20->setColour (TextEditor::backgroundColourId, Colour (0x0));


    //[UserPreSize]
    //[/UserPreSize]

    setSize (800, 328);


    //[Constructor] You can add your own custom stuff here..
//      startTimer(50);

       vstSynthAudioProcessor* ourProcessor = getProcessor();

       osc1WaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::osc1WaveParam), false);
       osc2WaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::osc2WaveParam), false);
       osc3WaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::osc3WaveParam), false);

       osc1OctaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::osc1OctaveParam), false);
       osc2OctaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::osc2OctaveParam), false);
       osc3OctaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::osc3OctaveParam), false);

       osc1lvlSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::osc1LevelParam)), false); //dB
       osc2lvlSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::osc2LevelParam)), false); //dB
       osc3lvlSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::osc3LevelParam)), false); //dB

	//osc1lvlSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::osc1LevelParam), false);
	//osc2lvlSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::osc2LevelParam), false);
	//osc3lvlSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::osc3LevelParam), false);

	noiseSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::noiseParam)), false); //dB
	//noiseSlider->setValue(ourProcessor->getParameter(vstSynthAudioProcessor::noiseParam),
false);

    attackSlider->setValue(ourProcessor->getParameter(vstSynthAudioProcessor::attackParam),
false);
	decaySlider->setValue(ourProcessor->getParameter(vstSynthAudioProcessor::decayParam),
false);
	sustainSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::sustainParam), false);
	releaseSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::releaseParam), false);

	lfoDestBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::lfoDestParam), false);
	lfoWaveBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::lfoWaveParam), false);
	lfoFreqSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::lfoFreqParam), false);
	lfoDevSlider->setValue(ourProcessor->getParameter(vstSynthAudioProcessor::lfoDevParam),
false);

	float sampleRate = getProcessor()->getSampleRate();

	filterFreqSlider->setValue(sampleRate / (2 * PI) * ourProcessor-
>getParameter(vstSynthAudioProcessor::filterCutoffParam), false);
	filterTypeBox->setSelectedItemIndex (ourProcessor-
>getParameter(vstSynthAudioProcessor::filterTypeParam), false);
	filterResSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::filterResonanceParam)), false); //dB
	filterGainSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::filterGainParam)), false); //dB
//	filterResSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::filterResonanceParam), false);
//	filterGainSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::filterGainParam), false);

	delayTimeSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::delayTimeParam)/sampleRate, false);
	delayFBSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::delayFeedbackParam)), false); //dB
	delayGainSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::delayGainParam)), false); //dB
//	delayFBSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::delayFeedbackParam), false);
//	delayGainSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::delayGainParam), false);
```

```
       driveSlider->setValue(ourProcessor->getParameter(vstSynthAudioProcessor::driveParam),
false);

    outputGainSlider->setValue(20 * log10(ourProcessor-
>getParameter(vstSynthAudioProcessor::outputGainParam)), false); //dB
//     outputGainSlider->setValue(ourProcessor-
>getParameter(vstSynthAudioProcessor::outputGainParam), false);

    //[/Constructor]
}

vstSynthAudioProcessorEditor::~vstSynthAudioProcessorEditor()
{
    //[Destructor_pre]. You can add your own custom destruction code here..
    //[/Destructor_pre]

    deleteAndZero (filterGroup);
    deleteAndZero (filterGainSlider);
    deleteAndZero (lfoGroup);
    deleteAndZero (lfoDevSlider);
    deleteAndZero (oscGroup);
    deleteAndZero (label21);
    deleteAndZero (envelopeGroup);
    deleteAndZero (sustainSlider);
    deleteAndZero (osc1lvlSlider);
    deleteAndZero (delayGroup);
    deleteAndZero (delayFBSlider);
    deleteAndZero (delayGainSlider);
    deleteAndZero (outputGroup);
    deleteAndZero (label8);
    deleteAndZero (decaySlider);
    deleteAndZero (attackSlider);
    deleteAndZero (delayTimeSlider);
    deleteAndZero (label);
    deleteAndZero (label2);
    deleteAndZero (label3);
    deleteAndZero (label4);
    deleteAndZero (label5);
    deleteAndZero (label6);
    deleteAndZero (label7);
    deleteAndZero (noiseSlider);
    deleteAndZero (driveSlider);
    deleteAndZero (outputGainSlider);
    deleteAndZero (label11);
    deleteAndZero (label12);
    deleteAndZero (label13);
    deleteAndZero (releaseSlider);
    deleteAndZero (osc3WaveBox);
    deleteAndZero (osc3OctaveBox);
    deleteAndZero (label23);
    deleteAndZero (label24);
    deleteAndZero (label25);
    deleteAndZero (label27);
    deleteAndZero (label29);
    deleteAndZero (filterFreqSlider);
    deleteAndZero (label31);
    deleteAndZero (label32);
    deleteAndZero (label33);
    deleteAndZero (lfoFreqSlider);
    deleteAndZero (lfoWaveBox);
    deleteAndZero (lfoDestBox);
    deleteAndZero (label9);
    deleteAndZero (osc3lvlSlider);
    deleteAndZero (filterResSlider);
    deleteAndZero (filterTypeBox);
    deleteAndZero (label10);
    deleteAndZero (label26);
    deleteAndZero (osc2WaveBox);
    deleteAndZero (osc2OctaveBox);
    deleteAndZero (label14);
    deleteAndZero (label15);
    deleteAndZero (label16);
    deleteAndZero (osc2lvlSlider);
    deleteAndZero (label17);
    deleteAndZero (osc1WaveBox);
    deleteAndZero (osc1OctaveBox);
    deleteAndZero (label18);
    deleteAndZero (label19);
    deleteAndZero (label20);


    //[Destructor]. You can add your own custom destruction code here..
    //[/Destructor]
}

//==============================================================================
void vstSynthAudioProcessorEditor::paint (Graphics& g)
{
    //[UserPrePaint] Add your own custom painting code here..
    //[/UserPrePaint]

    g.fillAll (Colour (0xff131a26));

    g.setColour (Colour (0x22c1b115));
    g.fillRoundedRectangle (624.0f, 18.0f, 72.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22c67e15));
    g.fillRoundedRectangle (14.0f, 18.0f, 238.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22c67e15));
    g.fillRoundedRectangle (14.0f, 118.0f, 238.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22c67e15));
    g.fillRoundedRectangle (14.0f, 218.0f, 238.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22158ec6));
    g.fillRoundedRectangle (270.0f, 18.0f, 336.0f, 84.0f, 10.0000f);

    g.setColour (Colour (0x2274c615));
    g.fillRoundedRectangle (270.0f, 124.0f, 336.0f, 85.0f, 10.0000f);

    g.setColour (Colour (0x22c6159a));
    g.fillRoundedRectangle (270.0f, 230.0f, 336.0f, 85.0f, 10.0000f);

    g.setColour (Colour (0x22c61515));
    g.fillRoundedRectangle (714.0f, 18.0f, 72.0f, 98.0f, 10.0000f);
    g.setColour (Colour (0x22c61515));
    g.fillRoundedRectangle (714.0f, 118.0f, 72.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22c61515));
    g.fillRoundedRectangle (714.0f, 218.0f, 72.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22c1b115));
    g.fillRoundedRectangle (624.0f, 218.0f, 72.0f, 98.0f, 10.0000f);

    g.setColour (Colour (0x22c1b115));
    g.fillRoundedRectangle (624.0f, 118.0f, 72.0f, 98.0f, 10.0000f);

    //[UserPaint] Add your own custom painting code here..
    //[/UserPaint]
}

void vstSynthAudioProcessorEditor::resized()
{
    filterGroup->setBounds (262, 215, 352, 109);
    filterGainSlider->setBounds (540, 240, 60, 80);
    lfoGroup->setBounds (262, 109, 352, 108);
    lfoDevSlider->setBounds (540, 134, 60, 80);
    oscGroup->setBounds (6, 2, 254, 322);
    label21->setBounds (15, 20, 235, 20);
    envelopeGroup->setBounds (262, 2, 352, 108);
    sustainSlider->setBounds (460, 28, 60, 80);
    osc1lvlSlider->setBounds (190, 35, 60, 80);
    delayGroup->setBounds (616, 2, 88, 322);
    delayFBSlider->setBounds (632, 235, 60, 80);
    delayGainSlider->setBounds (632, 135, 60, 80);
    outputGroup->setBounds (706, 2, 88, 322);
    label8->setBounds (268, 144, 80, 20);
    decaySlider->setBounds (380, 28, 60, 80);
    attackSlider->setBounds (300, 28, 60, 80);
    delayTimeSlider->setBounds (630, 35, 60, 80);
    label->setBounds (286, 14, 88, 20);
    label2->setBounds (446, 14, 88, 20);
    label3->setBounds (366, 14, 88, 20);
    label4->setBounds (526, 14, 88, 20);
    label5->setBounds (616, 220, 88, 20);
    label6->setBounds (616, 120, 88, 20);
    label7->setBounds (616, 20, 88, 20);
    noiseSlider->setBounds (720, 35, 60, 80);
    driveSlider->setBounds (720, 135, 60, 80);
    outputGainSlider->setBounds (720, 235, 60, 80);
    label11->setBounds (706, 220, 88, 20);
    label12->setBounds (706, 120, 88, 20);
    label13->setBounds (706, 20, 88, 20);
    releaseSlider->setBounds (540, 28, 60, 80);
    osc3WaveBox->setBounds (90, 245, 90, 20);
    osc3OctaveBox->setBounds (90, 280, 90, 20);
    label23->setBounds (10, 280, 80, 20);
    label24->setBounds (10, 245, 80, 20);
    label25->setBounds (190, 220, 60, 20);
    label27->setBounds (446, 120, 88, 20);
    label29->setBounds (526, 120, 88, 20);
    filterFreqSlider->setBounds (380, 240, 60, 80);
    label31->setBounds (446, 226, 88, 20);
    label32->setBounds (366, 226, 88, 20);
    label33->setBounds (526, 226, 88, 20);
    lfoFreqSlider->setBounds (460, 134, 60, 80);
    lfoWaveBox->setBounds (350, 179, 90, 20);
    lfoDestBox->setBounds (350, 144, 90, 20);
    label9->setBounds (268, 179, 80, 20);
    osc3lvlSlider->setBounds (190, 235, 60, 80);
    filterResSlider->setBounds (460, 240, 60, 80);
    filterTypeBox->setBounds (278, 270, 90, 20);
    label10->setBounds (278, 245, 88, 24);
    label26->setBounds (15, 220, 235, 20);
    osc2WaveBox->setBounds (90, 145, 90, 20);
    osc2OctaveBox->setBounds (90, 180, 90, 20);
    label14->setBounds (10, 180, 80, 20);
    label15->setBounds (10, 145, 80, 20);
    label16->setBounds (190, 120, 60, 20);
    osc2lvlSlider->setBounds (190, 135, 60, 80);
    label17->setBounds (15, 120, 235, 20);
    osc1WaveBox->setBounds (90, 45, 90, 20);
    osc1OctaveBox->setBounds (90, 80, 90, 20);
    label18->setBounds (10, 80, 80, 20);
    label19->setBounds (10, 45, 80, 20);
    label20->setBounds (190, 20, 60, 20);
    //[UserResized] Add your own custom resize handling here..
    //[/UserResized]
}

void vstSynthAudioProcessorEditor::sliderValueChanged (Slider* sliderThatWasMoved)
{
    //[UsersliderValueChanged_Pre]
    //[/UsersliderValueChanged_Pre]

       // Oscillators ========================
    if (sliderThatWasMoved == osc1lvlSlider)
    {
        //[UserSliderCode_osc1lvlSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::osc1LevelParam, (float) pow(10, 0.05 * osc1lvlSlider->getValue()));
//            getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::osc1LevelParam, (float) osc1lvlSlider->getValue());
        //[/UserSliderCode_osc1lvlSlider]
    }
    else if (sliderThatWasMoved == osc2lvlSlider)
    {
        //[UserSliderCode_osc2lvlSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::osc2LevelParam, (float) pow(10, 0.05 * osc2lvlSlider->getValue()));
//            getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::osc2LevelParam, (float) osc2lvlSlider->getValue());
        //[/UserSliderCode_osc2lvlSlider]
    }
    else if (sliderThatWasMoved == osc3lvlSlider)
    {
        //[UserSliderCode_osc3lvlSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::osc3LevelParam, (float) pow(10, 0.05 * osc3lvlSlider->getValue()));
//            getProcessor()->setParameterNotifyingHost
```

```
(vstSynthAudioProcessor::osc3LevelParam, (float) osc3lvlSlider->getValue());
        //[/UserSliderCode_osc3lvlSlider]
    }

       // LFO =====================================
    else if (sliderThatWasMoved == lfoDevSlider)
    {
        //[UserSliderCode_lfoDevSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::lfoDevParam,
(float) lfoDevSlider->getValue());
        //[/UserSliderCode_lfoDevSlider]
    }
    else if (sliderThatWasMoved == lfoFreqSlider)
    {
        //[UserSliderCode_lfoFreqSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::lfoFreqParam,
(float) lfoFreqSlider->getValue());
        //[/UserSliderCode_lfoFreqSlider]
    }

       // Noise ==================================
    else if (sliderThatWasMoved == noiseSlider)
    {
        //[UserSliderCode_noiseSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::noiseParam,
(float) pow(10, 0.05 * noiseSlider->getValue()));
//            getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::noiseParam,
(float) noiseSlider->getValue());
        //[/UserSliderCode_noiseSlider]
    }

       // Envelope =================================
    else if (sliderThatWasMoved == attackSlider)
    {
        //[UserSliderCode_attackSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::attackParam,
(float) attackSlider->getValue());
        //[/UserSliderCode_attackSlider]
    }
    else if (sliderThatWasMoved == sustainSlider)
    {
        //[UserSliderCode_sustainSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::sustainParam,
(float) sustainSlider->getValue());
        //[/UserSliderCode_sustainSlider]
    }
    else if (sliderThatWasMoved == decaySlider)
    {
        //[UserSliderCode_decaySlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::decayParam,
(float) decaySlider->getValue());
        //[/UserSliderCode_decaySlider]
    }
    else if (sliderThatWasMoved == releaseSlider)
    {
        //[UserSliderCode_releaseSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::releaseParam,
(float) releaseSlider->getValue());
```

```
        //[/UserSliderCode_releaseSlider]
    }

       // Filter ======================================
    else if (sliderThatWasMoved == filterGainSlider)
    {
        //[UserSliderCode_filterGainSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::filterGainParam, (float) pow(10, 0.05 * filterGainSlider-
>getValue()));
//            getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::filterGainParam, (float) filterGainSlider->getValue());
              getProcessor()->hpeqFilter.setGain(getProcessor()-
>getParameter(vstSynthAudioProcessor::filterGainParam));
        //[/UserSliderCode_filterGainSlider]
    }
    else if (sliderThatWasMoved == filterFreqSlider)
    {
        //[UserSliderCode_filterFreqSlider] -- add your slider handling code here..
              float sampleRate = getProcessor()->getSampleRate();
        // Convert from Hz to rads/sample
        float radsPerSample = 2 * PI * (float) filterFreqSlider->getValue()/sampleRate;
        getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::filterCutoffParam,
radsPerSample);
        getProcessor()->hpeqFilter.setFrequency(getProcessor()-
>getParameter(vstSynthAudioProcessor::filterCutoffParam));
        //[/UserSliderCode_filterFreqSlider]
    }
    else if (sliderThatWasMoved == filterResSlider)
    {
        //[UserSliderCode_filterResSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::filterResonanceParam, (float) pow(10, 0.05 * filterResSlider-
>getValue()));
//            getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::filterResonanceParam, (float) filterResSlider->getValue());
        getProcessor()->hpeqFilter.setResonance(getProcessor()-
>getParameter(vstSynthAudioProcessor::filterResonanceParam));
        //[/UserSliderCode_filterResSlider]
    }

       // Delay ======================================
    else if (sliderThatWasMoved == delayFBSlider)
    {
        //[UserSliderCode_delayFBSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::delayFeedbackParam, (float) pow(10, 0.05 * delayFBSlider-
>getValue()));
//            getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::delayFeedbackParam, (float) delayFBSlider->getValue());
        //[/UserSliderCode_delayFBSlider]
    }
    else if (sliderThatWasMoved == delayGainSlider)
    {
        //[UserSliderCode_delayGainSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::delayGainParam, (float) pow(10, 0.05 * delayGainSlider->getValue()));
//            getProcessor()->setParameterNotifyingHost
```

```
(vstSynthAudioProcessor::delayGainParam, (float) delayGainSlider->getValue());
        //[/UserSliderCode_delayGainSlider]
    }

       else if (sliderThatWasMoved == delayTimeSlider)
    {
              //[UserSliderCode_delayTimeSlider] -- add your slider handling code here..
              float sampleRate = getProcessor()->getSampleRate();
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::delayTimeParam, floor(sampleRate * (float) delayTimeSlider-
>getValue()));
        //[/UserSliderCode_delayTimeSlider]
    }

       // Output ====================================
    else if (sliderThatWasMoved == driveSlider)
    {
        //[UserSliderCode_driveSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost (vstSynthAudioProcessor::driveParam,
(float) driveSlider->getValue());
        //[/UserSliderCode_driveSlider]
    }
    else if (sliderThatWasMoved == outputGainSlider)
    {
        //[UserSliderCode_outputGainSlider] -- add your slider handling code here..
              getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::outputGainParam, (float) pow(10, 0.05 * outputGainSlider-
>getValue()));
//            getProcessor()->setParameterNotifyingHost
(vstSynthAudioProcessor::outputGainParam, (float) outputGainSlider->getValue());
        //[/UserSliderCode_outputGainSlider]
    }

    //[UsersliderValueChanged_Post]
    //[/UsersliderValueChanged_Post]
}

void vstSynthAudioProcessorEditor::comboBoxChanged (ComboBox* comboBoxThatHasChanged)
{
    //[UsercomboBoxChanged_Pre]
    //[/UsercomboBoxChanged_Pre]

    // Oscillator 1 ===================
       if (comboBoxThatHasChanged == osc1WaveBox)
    {
        //[UserComboBoxCode_osc1WaveBox] -- add your combo box handling code here..
              getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::osc1WaveParam,
(float) osc1WaveBox->getSelectedItemIndex());

        //[/UserComboBoxCode_osc1WaveBox]
    }
    else if (comboBoxThatHasChanged == osc1OctaveBox)
    {
        //[UserComboBoxCode_osc1OctaveBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::osc1OctaveParam,
(float) osc1OctaveBox->getSelectedItemIndex());

        //[/UserComboBoxCode_osc1OctaveBox]
    }

    // Oscillator 2 ===================
    else if (comboBoxThatHasChanged == osc2WaveBox)
    {
        //[UserComboBoxCode_osc2WaveBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::osc2WaveParam,
(float) osc2WaveBox->getSelectedItemIndex());
        //[/UserComboBoxCode_osc2WaveBox]
    }
    else if (comboBoxThatHasChanged == osc2OctaveBox)
    {
        //[UserComboBoxCode_osc2OctaveBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::osc2OctaveParam,
(float) osc2OctaveBox->getSelectedItemIndex());
        //[/UserComboBoxCode_osc2OctaveBox]
    }

    // Oscillator 3 ===================
       else if (comboBoxThatHasChanged == osc3WaveBox)
    {
        //[UserComboBoxCode_osc3WaveBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::osc3WaveParam,
(float) osc3WaveBox->getSelectedItemIndex());
        //[/UserComboBoxCode_osc3WaveBox]
    }
    else if (comboBoxThatHasChanged == osc3OctaveBox)
    {
        //[UserComboBoxCode_osc3OctaveBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::osc3OctaveParam,
(float) osc3OctaveBox->getSelectedItemIndex());
        //[/UserComboBoxCode_osc3OctaveBox]
    }

       // LFO ==============================
    else if (comboBoxThatHasChanged == lfoWaveBox)
    {
        //[UserComboBoxCode_lfoWaveBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::lfoWaveParam, (float)
lfoWaveBox->getSelectedItemIndex());
        //[/UserComboBoxCode_lfoWaveBox]
    }
    else if (comboBoxThatHasChanged == lfoDestBox)
    {
        //[UserComboBoxCode_lfoDestBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::lfoDestParam, (float)
lfoDestBox->getSelectedItemIndex());
        //[/UserComboBoxCode_lfoDestBox]
    }

       // Filter Type
    else if (comboBoxThatHasChanged == filterTypeBox)
    {
        //[UserComboBoxCode_filterTypeBox] -- add your combo box handling code here..
        getProcessor()->setParameterNotifyingHost(vstSynthAudioProcessor::filterTypeParam,
(float) filterTypeBox->getSelectedItemIndex());
              getProcessor()->hpeqFilter.setType((int) getProcessor()-
>getParameter(vstSynthAudioProcessor::filterTypeParam));
```

```
        //[/UserComboBoxCode_filterTypeBox]
    }

    //[UsercomboBoxChanged_Post]
    //[/UsercomboBoxChanged_Post]
}


//[MiscUserCode] You can add your own definitions of your custom methods or any other code
here...
//[/MiscUserCode]


//==============================================================================
#if 0
/*  -- Jucer information section --

    This is where the Jucer puts all of its metadata, so don't change anything in here!

BEGIN_JUCER_METADATA

<JUCER_COMPONENT documentType="Component" className="vstSynthAudioProcessorEditor"
                 componentName="" parentClasses="public Component"
constructorParams="vstSynthAudioProcessor* ownerFilter"
                 variableInitialisers="AudioProcessorEditor (ownerFilter)" snapPixels="8"
                 snapActive="1" snapShown="1" overlayOpacity="0.330000013" fixedSize="1"
                 initialWidth="800" initialHeight="328">
  <BACKGROUND backgroundColour="ff131a26">
    <ROUNDRECT pos="624 18 72 98" cornerSize="10" fill="solid: 22c1b115" hasStroke="0"/>
    <ROUNDRECT pos="14 18 238 98" cornerSize="10" fill="solid: 22c67e15" hasStroke="0"/>
    <ROUNDRECT pos="14 118 238 98" cornerSize="10" fill="solid: 22c67e15" hasStroke="0"/>
    <ROUNDRECT pos="14 218 238 98" cornerSize="10" fill="solid: 22c67e15" hasStroke="0"/>
    <ROUNDRECT pos="270 18 336 84" cornerSize="10" fill="solid: 22158ec6" hasStroke="0"/>
    <ROUNDRECT pos="270 124 336 85" cornerSize="10" fill="solid: 2274c615" hasStroke="0"/>
    <ROUNDRECT pos="270 230 336 85" cornerSize="10" fill="solid: 22c6159a" hasStroke="0"/>
    <ROUNDRECT pos="714 18 72 98" cornerSize="10" fill="solid: 22c61515" hasStroke="0"/>
    <ROUNDRECT pos="714 118 72 98" cornerSize="10" fill="solid: 22c61515" hasStroke="0"/>
    <ROUNDRECT pos="714 218 72 98" cornerSize="10" fill="solid: 22c61515" hasStroke="0"/>
    <ROUNDRECT pos="624 218 72 98" cornerSize="10" fill="solid: 22c1b115" hasStroke="0"/>
    <ROUNDRECT pos="624 118 72 98" cornerSize="10" fill="solid: 22c1b115" hasStroke="0"/>
  </BACKGROUND>
  <GROUPCOMPONENT name="new group" id="ea3069c632e89929" memberName="filterGroup"
                  virtualName="" explicitFocusOrder="0" pos="262 215 352 109"
outlinecol="88c6159a"
                  textcol="ffffffff" title="Filter"/>
  <SLIDER name="new slider" id="ec58c63b4ccff8af" memberName="filterGainSlider"
          virtualName="" explicitFocusOrder="0" pos="540 240 60 80" rotarysliderfill="ffc6159a"
          rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
          textboxhighlight="1111ee" textboxoutline="808080" min="0" max="10"
          int="0.1" style="Rotary" textBoxPos="TextBoxAbove" textBoxEditable="0"
          textBoxWidth="80" textBoxHeight="20" skewFactor="1"/>
  <GROUPCOMPONENT name="new group" id="9a03f50615df1c9f" memberName="lfoGroup"
                  virtualName="" explicitFocusOrder="0" pos="262 109 352 108"
outlinecol="8874c615"
                  textcol="ffffffff" title="LFO" textpos="33"/>
  <SLIDER name="new slider" id="9b2e1431fcd2f805" memberName="lfoDevSlider"
          virtualName="" explicitFocusOrder="0" pos="540 134 60 80" rotarysliderfill="ff74c615"
          rotarysslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
          textboxoutline="808080" min="0" max="100" int="0.1" style="Rotary"
          textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
          textBoxHeight="20" skewFactor="1"/>
  <GROUPCOMPONENT name="new group" id="65a0feb597479c50" memberName="oscGroup"
                  virtualName="" explicitFocusOrder="0" pos="6 2 254 322" outlinecol="88c67e15"
                  textcol="ffffffff" title="Oscillators"/>
  <LABEL name="new label" id="55fd3e4f75cff325" memberName="label21" virtualName=""
         explicitFocusOrder="0" pos="15 20 235 20" textCol="ffffffff"
         edTextCol="ff000000" edBkgCol="0" labelText="Oscillator 1" editableSingleClick="0"
         editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
         fontsize="15" bold="1" italic="0" justification="36"/>
  <GROUPCOMPONENT name="new group" id="7ace822ca98bda71" memberName="envelopeGroup"
                  virtualName="" explicitFocusOrder="0" pos="262 2 352 108"
outlinecol="88158ec6"
                  textcol="ffffffff" title="Envelope"/>
  <SLIDER name="new slider" id="8f89739b8fee464b" memberName="sustainSlider"
          virtualName="" explicitFocusOrder="0" pos="460 28 60 80" rotarysliderfill="ff158ec6"
          rotarysslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
          textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
          textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
          textBoxHeight="20" skewFactor="1"/>
  <SLIDER name="new slider" id="fd2273ee05f26609" memberName="osc1lvlSlider"
          virtualName="" explicitFocusOrder="0" pos="190 35 60 80" rotarysliderfill="ffc67e15"
          rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
          textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
          textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
          textBoxHeight="20" skewFactor="1"/>
  <GROUPCOMPONENT name="new group" id="1ff1c17feeb8ec2a" memberName="delayGroup"
                  virtualName="" explicitFocusOrder="0" pos="616 2 88 322"
outlinecol="88c1b115"
                  textcol="ffffffff" title="Delay"/>
  <SLIDER name="new slider" id="a75d50ad64e53127" memberName="delayFBSlider"
          virtualName="" explicitFocusOrder="0" pos="632 235 60 80" rotarysliderfill="ffc1b115"
          rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="faf7f7"
          textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
          textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
          textBoxHeight="20" skewFactor="1"/>
  <SLIDER name="new slider" id="2b67fbd26d4a51e1" memberName="delayGainSlider"
          virtualName="" explicitFocusOrder="0" pos="632 135 60 80" rotarysliderfill="ffc1b115"
          rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
          textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
          textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
          textBoxHeight="20" skewFactor="1"/>
  <GROUPCOMPONENT name="new group" id="32ee0e26eb09be5a" memberName="outputGroup"
                  virtualName="" explicitFocusOrder="0" pos="706 2 88 322"
outlinecol="88c61515"
                  textcol="ffffffff" title="Output"/>
  <LABEL name="new label" id="ee0c7e762e5270f5" memberName="label8" virtualName=""
         explicitFocusOrder="0" pos="268 144 80 20" bkgCol="b70101" textCol="ffffffff"
         edTextCol="ff000000" edBkgCol="0" labelText="Destination" editableSingleClick="0"
         editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
         fontsize="15" bold="0" italic="0" justification="34"/>
  <SLIDER name="new slider" id="f5aabb99a95e5062" memberName="decaySlider"
          virtualName="" explicitFocusOrder="0" pos="380 28 60 80" rotarysliderfill="ff158ec6"
          rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
          textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
          textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
```

```
        textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="95bb1bdea3cb3d24" memberName="attackSlider"
        virtualName="" explicitFocusOrder="0" pos="300 28 60 80" rotarysliderfill="ff158ec6"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="d31354e1003f6da9" memberName="delayTimeSlider"
        virtualName="" explicitFocusOrder="0" pos="630 35 60 80" rotarysliderfill="ffc1b115"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<LABEL name="new label" id="501a157c94ed15a8" memberName="label" virtualName=""
       explicitFocusOrder="0" pos="286 14 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Attack" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="220b9859b66f7f8d" memberName="label2" virtualName=""
       explicitFocusOrder="0" pos="446 14 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Sustain" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="dfcf9cca27e240f5" memberName="label3" virtualName=""
       explicitFocusOrder="0" pos="366 14 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Decay" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="bb0184db111ca28" memberName="label4" virtualName=""
       explicitFocusOrder="0" pos="526 14 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Release" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="c5b8dd30e3f72c57" memberName="label5" virtualName=""
       explicitFocusOrder="0" pos="616 220 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Feedback" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="33f990a20ca36ef7" memberName="label6" virtualName=""
       explicitFocusOrder="0" pos="616 120 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Gain" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="efd42dad964489a3" memberName="label7" virtualName=""
       explicitFocusOrder="0" pos="616 20 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Time" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<SLIDER name="new slider" id="50603b8782394472" memberName="noiseSlider"
        virtualName="" explicitFocusOrder="0" pos="720 35 60 80" rotarysliderfill="ffc61515"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="bad6de7cc47b0108" memberName="driveSlider"
        virtualName="" explicitFocusOrder="0" pos="720 135 60 80" rotarysliderfill="ffc61515"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="c61515"
        textboxoutline="c61515" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="74d03ea84f25ef02" memberName="outputGainSlider"
        virtualName="" explicitFocusOrder="0" pos="720 235 60 80" rotarysliderfill="ffc61515"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<LABEL name="new label" id="6f104a8e37375aa0" memberName="label11" virtualName=""
       explicitFocusOrder="0" pos="706 220 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Gain" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="49c1ace250c4f8ee" memberName="label12" virtualName=""
       explicitFocusOrder="0" pos="706 120 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Drive" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="6b139137b8b0fd39" memberName="label13" virtualName=""
       explicitFocusOrder="0" pos="706 20 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Noise" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<SLIDER name="new slider" id="9dabcece093dca00" memberName="releaseSlider"
        virtualName="" explicitFocusOrder="0" pos="540 28 60 80" rotarysliderfill="ff158ec6"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<COMBOBOX name="new combo box" id="c37948d0c3bd9c5b" memberName="osc3WaveBox"
          virtualName="" explicitFocusOrder="0" pos="90 245 90 20" editable="0"
          layout="33" items="Sine&#10;Saw&#10;Square" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<COMBOBOX name="new combo box" id="4ae71ef44644cda7" memberName="osc3OctaveBox"
          virtualName="" explicitFocusOrder="0" pos="90 280 90 20" editable="0"
          layout="33" items="1&#10;2&#10;3&#10;4" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<LABEL name="new label" id="912baf86a1a0ad06" memberName="label23" virtualName=""
       explicitFocusOrder="0" pos="10 280 80 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Octave" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="0" italic="0" justification="34"/>
<LABEL name="new label" id="332fa9d8aa556539" memberName="label24" virtualName=""
       explicitFocusOrder="0" pos="10 245 80 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Waveform" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="0" italic="0" justification="34"/>
<LABEL name="new label" id="9824114c301fcea0" memberName="label25" virtualName=""
       explicitFocusOrder="0" pos="190 220 60 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Level" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="43d19f5393ba8ba0" memberName="label27" virtualName=""
       explicitFocusOrder="0" pos="446 120 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Frequency" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="200d41631cc9ffa5" memberName="label29" virtualName=""
```

```
       explicitFocusOrder="0" pos="526 120 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Deviation" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<SLIDER name="new slider" id="c6e3b536dabbe181" memberName="filterFreqSlider"
        virtualName="" explicitFocusOrder="0" pos="380 240 60 80" rotarysliderfill="ffc6159a"
        rotarysslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<LABEL name="new label" id="dfbe4933e3b8ec07" memberName="label31" virtualName=""
       explicitFocusOrder="0" pos="446 226 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Resonance" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="ac6f6b11384ff2b" memberName="label32" virtualName=""
       explicitFocusOrder="0" pos="366 226 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Frequency" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<LABEL name="new label" id="a0b29c1bc98035a1" memberName="label33" virtualName=""
       explicitFocusOrder="0" pos="526 226 88 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Gain" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<SLIDER name="new slider" id="c40e18508b0c7437" memberName="lfoFreqSlider"
        virtualName="" explicitFocusOrder="0" pos="460 134 60 80" rotarysliderfill="ff74c615"
        rotarysslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<COMBOBOX name="new combo box" id="9ef0d8a2b2849ba7" memberName="lfoWaveBox"
          virtualName="" explicitFocusOrder="0" pos="350 179 90 20" editable="0"
          layout="33" items="Sine&#10;Saw&#10;Square" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<COMBOBOX name="new combo box" id="8eb3a72193ed6930" memberName="lfoDestBox"
          virtualName="" explicitFocusOrder="0" pos="350 144 90 20" editable="0"
          layout="33" items="Off&#10;Vibrato&#10;Tremolo&#10;Envelope"
          textWhenNonSelected="" textWhenNoItems="(no choices)"/>
<LABEL name="new label" id="4995cb78101163ca" memberName="label9" virtualName=""
       explicitFocusOrder="0" pos="268 179 80 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Waveform" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="0" italic="0" justification="34"/>
<SLIDER name="new slider" id="4c6a01a02cb62be7" memberName="osc3lvlSlider"
        virtualName="" explicitFocusOrder="0" pos="190 235 60 80" rotarysliderfill="ffc67e15"
        rotarysslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808000" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<SLIDER name="new slider" id="5809b2c616366c36" memberName="filterResSlider"
        virtualName="" explicitFocusOrder="0" pos="460 240 60 80" rotarysliderfill="ffc6159a"
        rotarysslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<COMBOBOX name="new combo box" id="a6246f5792d2062f" memberName="filterTypeBox"
          virtualName="" explicitFocusOrder="0" pos="278 270 90 20" editable="0"
          layout="33" items="Low Pass&#10;Band Pass&#10;High Pass" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<LABEL name="new label" id="3c636708a43de103" memberName="label10" virtualName=""
       explicitFocusOrder="0" pos="278 245 88 24" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Type" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="0" italic="0" justification="33"/>
<LABEL name="new label" id="8eb30b7b5e9d77bd" memberName="label26" virtualName=""
       explicitFocusOrder="0" pos="15 220 235 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Oscillator 3" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<COMBOBOX name="new combo box" id="5c54f1a8a395239b" memberName="osc2WaveBox"
          virtualName="" explicitFocusOrder="0" pos="90 145 90 20" editable="0"
          layout="33" items="Sine&#10;Saw&#10;Square" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<COMBOBOX name="new combo box" id="2fc2e28ae0fde6a3" memberName="osc2OctaveBox"
          virtualName="" explicitFocusOrder="0" pos="90 180 90 20" editable="0"
          layout="33" items="1&#10;2&#10;3&#10;4" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<LABEL name="new label" id="b5a440e9f025ebd2" memberName="label14" virtualName=""
       explicitFocusOrder="0" pos="10 180 80 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Octave" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="0" italic="0" justification="34"/>
<LABEL name="new label" id="35e567332bb5c0d5" memberName="label15" virtualName=""
       explicitFocusOrder="0" pos="10 145 80 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Waveform" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="0" italic="0" justification="34"/>
<LABEL name="new label" id="f452e44f60e3b2ec" memberName="label16" virtualName=""
       explicitFocusOrder="0" pos="190 120 60 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Level" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<SLIDER name="new slider" id="e7671b1dcd8ad512" memberName="osc2lvlSlider"
        virtualName="" explicitFocusOrder="0" pos="190 135 60 80" rotarysliderfill="ffc67e15"
        rotaryslideroutline="ff808080" textboxtext="ffffffff" textboxbkgd="ffffff"
        textboxoutline="808080" min="0" max="10" int="0.1" style="Rotary"
        textBoxPos="TextBoxAbove" textBoxEditable="0" textBoxWidth="80"
        textBoxHeight="20" skewFactor="1"/>
<LABEL name="new label" id="1b877b32ef164692" memberName="label17" virtualName=""
       explicitFocusOrder="0" pos="15 120 235 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Oscillator 2" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
<COMBOBOX name="new combo box" id="f77e2df6e32fe9bb" memberName="osc1WaveBox"
          virtualName="" explicitFocusOrder="0" pos="90 45 90 20" editable="0"
          layout="33" items="Sine&#10;Saw&#10;Square" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<COMBOBOX name="new combo box" id="8160e144bbe9cd9c" memberName="osc1OctaveBox"
          virtualName="" explicitFocusOrder="0" pos="90 80 90 20" editable="0"
          layout="33" items="1&#10;2&#10;3&#10;4" textWhenNonSelected=""
          textWhenNoItems="(no choices)"/>
<LABEL name="new label" id="356b6b7cf57e370a" memberName="label18" virtualName=""
       explicitFocusOrder="0" pos="10 80 80 20" textCol="ffffffff" edTextCol="ff000000"
       edBkgCol="0" labelText="Octave" editableSingleClick="0" editableDoubleClick="0"
       focusDiscardsChanges="0" fontname="Default font" fontsize="15"
       bold="0" italic="0" justification="34"/>
<LABEL name="new label" id="532eb7a22ea08ac9" memberName="label19" virtualName=""
       explicitFocusOrder="0" pos="10 45 80 20" textCol="ffffffff" edTextCol="ff000000"
       edBkgCol="0" labelText="Waveform" editableSingleClick="0" editableDoubleClick="0"
       focusDiscardsChanges="0" fontname="Default font" fontsize="15"
       bold="0" italic="0" justification="34"/>
<LABEL name="new label" id="99c889db831c4574" memberName="label20" virtualName=""
       explicitFocusOrder="0" pos="190 20 60 20" textCol="ffffffff"
       edTextCol="ff000000" edBkgCol="0" labelText="Level" editableSingleClick="0"
       editableDoubleClick="0" focusDiscardsChanges="0" fontname="Default font"
       fontsize="15" bold="1" italic="0" justification="36"/>
</JUCER_COMPONENT>

END_JUCER_METADATA
*/
#endif
```

```
/*
vstSynthProcessor.h - header file for vstSynthProcessor.cpp
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
  ==============================================================================

    This file was auto-generated!

    It contains the basic startup code for a Juce application.

  ==============================================================================
*/

#ifndef __PLUGINPROCESSOR_H_AEB64BCF__
#define __PLUGINPROCESSOR_H_AEB64BCF__

#include "vstSynth.h"
#include "vstSynthFilter.h"


//==============================================================================
/**
*/
class vstSynthAudioProcessor  : public AudioProcessor
{
public:
    //==============================================================================
    vstSynthAudioProcessor();
    ~vstSynthAudioProcessor();

    //==============================================================================
    void prepareToPlay (double sampleRate, int samplesPerBlock);
    void releaseResources();

    void processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages);

    //==============================================================================
    AudioProcessorEditor* createEditor();
    bool hasEditor() const;

    //==============================================================================
    const String getName() const;
```

```
int getNumParameters();

float getParameter (int index);
void setParameter (int index, float newValue);

const String getParameterName (int index);
const String getParameterText (int index);

const String getInputChannelName (int channelIndex) const;
const String getOutputChannelName (int channelIndex) const;
bool isInputChannelStereoPair (int index) const;
bool isOutputChannelStereoPair (int index) const;

bool acceptsMidi() const;
bool producesMidi() const;

//==============================================================================
int getNumPrograms();
int getCurrentProgram();
void setCurrentProgram (int index);
const String getProgramName (int index);
void changeProgramName (int index, const String& newName);

//==============================================================================
void getStateInformation (MemoryBlock& destData);
void setStateInformation (const void* data, int sizeInBytes);

//==============================================================================

/*
 oscWave: oscillator output waveform. 0 = SineWave, 1 = BlitSquare, 2 = BlitSaw
 oscOctave: osciallator octave (1-4). output frequency = Midi key frequency * 2^oscOctave
 oscLevel: gain applied to individual oscillator.
 */

enum Parameters
{
    // single oscillator
    osc1WaveParam,
    osc1OctaveParam,
    osc1LevelParam,

    osc2WaveParam,
    osc2OctaveParam,
    osc2LevelParam,

    osc3WaveParam,
    osc3OctaveParam,
    osc3LevelParam,

          // envelope
    attackParam,
    decayParam,
    sustainParam,
    releaseParam,

    // LFO
```

```
        lfoDestParam,
        lfoWaveParam,
        lfoFreqParam,
        lfoDevParam,

        // filter
        filterTypeParam,
        filterCutoffParam,
        filterResonanceParam,
        filterGainParam,

        // delay
        delayTimeParam,
        delayFeedbackParam,
        delayGainParam,

        // output
        noiseParam,
        driveParam,
        outputGainParam,

        totalNumParams
    };

       float parameters[27];

    //==============================================================================
    Synthesiser vstSynth;
       MidiKeyboardState keyboardState;
    AudioSampleBuffer delayBuffer;
       vstSynthFilter hpeqFilter;
       Modulate tremolo;

    float* delayWritePtr;
    float* delayReadPtr;

       float gain;

    //==============================================================================
    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR (vstSynthAudioProcessor);
};

#endif  // __PLUGINPROCESSOR_H_AEB64BCF__
```

```
/*
vstSynthProcessor.cpp - synth implementation using:
              - STK (https://ccrma.stanford.edu/software/stk/)
                   for "oscillators" and envelope generators
              - JUCE (http://rawmaterialsoftware.com)
                   for connection to VST/AU, MIDI and GUI
              - high order parametric filters (http://www.ece.rutgers.edu/~orfanidi/hpeq/)
              - circular delay buffer

Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
==============================================================================

This file was auto-generated!

It contains the basic startup code for a Juce application.

==============================================================================
*/

#include "vstSynthProcessor.h"
#include "vstSynthEditor.h"

//==============================================================================
vstSynthAudioProcessor::vstSynthAudioProcessor()
       : delayBuffer(1,1)

{
    // Initialize all parameters
       setParameterNotifyingHost(osc1WaveParam, 1);
       setParameterNotifyingHost(osc2WaveParam, 1);
       setParameterNotifyingHost(osc3WaveParam, 1);

       setParameterNotifyingHost(osc1OctaveParam, 0);
       setParameterNotifyingHost(osc2OctaveParam, 0);
       setParameterNotifyingHost(osc3OctaveParam, 0);

       setParameterNotifyingHost(osc1LevelParam, 1);
       setParameterNotifyingHost(osc2LevelParam, 1);
       setParameterNotifyingHost(osc3LevelParam, 1);

       setParameterNotifyingHost(noiseParam, 0.0001);
```

```
	setParameterNotifyingHost(attackParam, 0);
	setParameterNotifyingHost(decayParam, 0);
	setParameterNotifyingHost(sustainParam, 5);
	setParameterNotifyingHost(releaseParam, 0);

	setParameterNotifyingHost(lfoDestParam, 0);
	setParameterNotifyingHost(lfoWaveParam, 0);
	setParameterNotifyingHost(lfoFreqParam, 0);
	setParameterNotifyingHost(lfoDevParam, 0);

	setParameterNotifyingHost(filterTypeParam, 0);
	setParameterNotifyingHost(filterCutoffParam, 0.25 * PI);
	setParameterNotifyingHost(filterResonanceParam, 1);
	setParameterNotifyingHost(filterGainParam, 1);

	setParameterNotifyingHost(delayTimeParam, 0);
	setParameterNotifyingHost(delayFeedbackParam, 0.0001);
	setParameterNotifyingHost(delayGainParam, 0.0001);

	setParameterNotifyingHost(driveParam, 1);
	setParameterNotifyingHost(outputGainParam, 1); //log

	// Initialise the synth.
	for (int numVoices = 1; --numVoices >= 0;)
	{
		// A reference to global parameters is passed to each
	 // voice so that it can reference them locally.
	 vstSynth.addVoice (new vstSynthVoice(parameters));
	}
	vstSynth.addSound(new vstSynthSound());
}

vstSynthAudioProcessor::~vstSynthAudioProcessor()
{
}

//==============================================================================
const String vstSynthAudioProcessor::getName() const
{
	return JucePlugin_Name;
}

int vstSynthAudioProcessor::getNumParameters()
{
	return totalNumParams;
}

float vstSynthAudioProcessor::getParameter (int index)
{
	switch (index)
	{
		// single oscillator
	case osc1WaveParam:          return parameters[0];
	case osc1OctaveParam:        return parameters[1];
	case osc1LevelParam:         return parameters[2];

	case osc2WaveParam:          return parameters[3];
	case osc2OctaveParam:		return parameters[4];
	case osc2LevelParam:		return parameters[5];

	case osc3WaveParam:		return parameters[6];
	case osc3OctaveParam:		return parameters[7];
	case osc3LevelParam:		return parameters[8];

		// envelope
	case attackParam:		return parameters[9];
	case decayParam:		return parameters[10];
	case sustainParam:		return parameters[11];
	case releaseParam:		return parameters[12];

		// LFO
	case lfoDestParam:		return parameters[13];
	case lfoWaveParam:		return parameters[14];
	case lfoFreqParam:		return parameters[15];
	case lfoDevParam:		return parameters[16];

		// filter
	case filterTypeParam:		return parameters[17];
	case filterCutoffParam:		return parameters[18];
	case filterResonanceParam:	return parameters[19];
	case filterGainParam:		return parameters[20];

		// delay
	case delayTimeParam:		return parameters[21];
	case delayFeedbackParam:	return parameters[22];
	case delayGainParam:		return parameters[23];

		// output
	case noiseParam:		return parameters[24];
	case driveParam:		return parameters[25];
	case outputGainParam:		return parameters[26];

	default:			return 0.0f;
	}
}

void vstSynthAudioProcessor::setParameter (int index, float newValue)
{
	switch (index)
	{
		// single oscillator
	case osc1WaveParam:		parameters[0] = newValue; break;
	case osc1OctaveParam:		parameters[1] = newValue; break;
	case osc1LevelParam:		parameters[2] = newValue; break;

	case osc2WaveParam:		parameters[3] = newValue; break;
	case osc2OctaveParam:		parameters[4] = newValue; break;
	case osc2LevelParam:		parameters[5] = newValue; break;

	case osc3WaveParam:		parameters[6] = newValue; break;
	case osc3OctaveParam:		parameters[7] = newValue; break;
	case osc3LevelParam:		parameters[8] = newValue; break;

		// envelope
	case attackParam:		parameters[9] = newValue; break;
```

```
        case decayParam:            parameters[10] = newValue; break;
        case sustainParam:          parameters[11] = newValue; break;
        case releaseParam:          parameters[12] = newValue; break;

               // LFO
        case lfoDestParam:          parameters[13] = newValue; break;
        case lfoWaveParam:          parameters[14] = newValue; break;
        case lfoFreqParam:          parameters[15] = newValue; break;
        case lfoDevParam:           parameters[16] = newValue; break;

               // filter
        case filterTypeParam:       parameters[17] = newValue; break;
        case filterCutoffParam:     parameters[18] = newValue; break;
        case filterResonanceParam:  parameters[19] = newValue; break;
        case filterGainParam:       parameters[20] = newValue; break;

               // delay
        case delayTimeParam:        parameters[21] = newValue; break;
        case delayFeedbackParam:    parameters[22] = newValue; break;
        case delayGainParam:        parameters[23] = newValue; break;

               // output
        case noiseParam:            parameters[24] = newValue; break;
        case driveParam:            parameters[25] = newValue; break;
        case outputGainParam:       parameters[26] = newValue; break;

        default:                    break;
        }
}

const String vstSynthAudioProcessor::getParameterName (int index)
{
        switch (index)
        {
               // single oscillator
        case osc1WaveParam:          return "osc1Wave";
        case osc1OctaveParam:        return "osc1Octave";
        case osc1LevelParam:         return "osc1Level";

        case osc2WaveParam:          return "osc2Wave";
        case osc2OctaveParam:        return "osc2Octave";
        case osc2LevelParam:         return "osc2Level";

        case osc3WaveParam:          return "osc3Wave";
        case osc3OctaveParam:        return "osc3Octave";
        case osc3LevelParam:         return "osc3Level";

               // envelope
        case attackParam:           return "attack";
        case decayParam:            return "decay";
        case sustainParam:          return "sustain";
        case releaseParam:          return "release";

               // LFO
        case lfoDestParam:          return "lfoDest";
        case lfoWaveParam:          return "lfoWave";
        case lfoFreqParam:          return "lfoFreq";
        case lfoDevParam:           return "lfoDev";
```

```
            // filter
        case filterTypeParam:       return "filterType";
        case filterCutoffParam:     return "filterCutoff";
        case filterResonanceParam:  return "filterResonance";
        case filterGainParam:       return "filterGain";

            // delay
        case delayTimeParam:        return "delayTime";
        case delayFeedbackParam:     return "delayFeedback";
        case delayGainParam:        return "delayGain";

            // output
        case noiseParam:            return "noiseGain";
        case driveParam:            return "drive";
        case outputGainParam:       return "outputGain";

        default:                    break;
        }
        return String::empty;
}

const String vstSynthAudioProcessor::getParameterText (int index)
{
        return String (getParameter (index), 2);
}

const String vstSynthAudioProcessor::getInputChannelName (int channelIndex) const
{
        return String (channelIndex + 1);
}

const String vstSynthAudioProcessor::getOutputChannelName (int channelIndex) const
{
        return String (channelIndex + 1);
}

bool vstSynthAudioProcessor::isInputChannelStereoPair (int index) const
{
        return true;
}

bool vstSynthAudioProcessor::isOutputChannelStereoPair (int index) const
{
        return true;
}

bool vstSynthAudioProcessor::acceptsMidi() const
{
#if JucePlugin_WantsMidiInput
        return true;
#else
        return false;
#endif
}

bool vstSynthAudioProcessor::producesMidi() const
{
```

```
#if JucePlugin_ProducesMidiOutput
        return true;
#else
        return false;
#endif
}

int vstSynthAudioProcessor::getNumPrograms()
{
        return 0;
}

int vstSynthAudioProcessor::getCurrentProgram()
{
        return 0;
}

void vstSynthAudioProcessor::setCurrentProgram (int index)
{
}

const String vstSynthAudioProcessor::getProgramName (int index)
{
        return String::empty;
}

void vstSynthAudioProcessor::changeProgramName (int index, const String& newName)
{
}

//==============================================================================
void vstSynthAudioProcessor::prepareToPlay (double sampleRate, int samplesPerBlock)
{
    // Set sample rate for both JUCE and STK
    vstSynth.setCurrentPlaybackSampleRate(sampleRate);
        Stk::setSampleRate(sampleRate);

        delayBuffer.clear();
        keyboardState.reset();

    // Initialize filter with starting parameters
        hpeqFilter.setCoefficients(getParameter(filterTypeParam), getParameter(filterGainParam),
getParameter(filterCutoffParam), getParameter(filterResonanceParam));
}

void vstSynthAudioProcessor::releaseResources()
{
        // When playback stops, you can use this as an opportunity to free up any
        // spare memory, etc.
        keyboardState.reset();
}

void vstSynthAudioProcessor::processBlock (AudioSampleBuffer& buffer, MidiBuffer& midiMessages)
{
    // number of samples in current buffer
        const int numSamples = buffer.getNumSamples();
    /*
     Checks to see if delay size has changed since the last block. If it has,
     the delay buffer is resized and cleared (to prevent garbage in the output)
     The read and write pointers are also reset to their starting positions and
     the saved filter states are removed to reduce transients.
     */

    // delayTimeParam controlled by vstSynthEditor::delayTimeSlider
    if (delayBuffer.getNumSamples() != getParameter(delayTimeParam) + numSamples)
       {
              delayBuffer.setSize(1, getParameter(delayTimeParam) + numSamples);
              delayBuffer.clear();
              delayWritePtr =  delayBuffer.getSampleData(0) + (int)
getParameter(delayTimeParam);
              delayReadPtr = delayBuffer.getSampleData(0);
              //hpeqFilter.reset();
       }

       // Receives MIDI data from host
    keyboardState.processNextMidiBuffer(midiMessages, 0, numSamples, true);

    // Call to vstSynthVoice::renderNextBlock where buffer is filled with raw oscillator data
    vstSynth.renderNextBlock(buffer, midiMessages, 0, numSamples);

    // Pointer to beginning of buffer
       float* bufferPtr = buffer.getSampleData(0, 0);

    // Performs tremolo (AM) if enabled, overdrive and delay operation
       for (int currentSample = 0; currentSample < numSamples; currentSample++)
       {
              // Apply tremolo if enabled
        if (getParameter(lfoDestParam) == 2) // Controlled by vstSynthEditor::lfoDestComboBox
              {
                     tremolo.setVibratoRate(getParameter(lfoFreqParam)); // Controlled by
vstSynthEditor::lfoFreqSlider
                     tremolo.setVibratoGain(getParameter(lfoDevParam)/10); // Controlled by
vstSynthEditor::lfoDevSlider
                     *bufferPtr *= (float) (1+tremolo.tick()); // Modulate amplitude with
tremolo output
              }

        // Push signal through tahn to introduce nonlinear distortion
              *bufferPtr = tanhf(getParameter(driveParam) * *bufferPtr); // Controlled by
vstSynthEditor::driveSlider

        // Process delay if enabled
              if (getParameter(delayTimeParam) > 0) // Controlled by
vstSynthEditor::delayTimeSlider
              {
            // Add existing delay data into buffer
                     *bufferPtr += getParameter(delayFeedbackParam) * *delayReadPtr; //
Controlled by vstSynthEditor::delayFBSlider

            // Save current output data into delay buffer
                     *delayWritePtr = *bufferPtr;

            // Increment pointers
            delayWritePtr++;
```

```
                    delayReadPtr++;

            // Circular buffer implementation: reset pointers to beginning of buffers when end
is reached
                    if (delayReadPtr > delayBuffer.getSampleData(0) +
delayBuffer.getNumSamples())
                    {
                           delayReadPtr = delayBuffer.getSampleData(0);
                    }

                    if (delayWritePtr > delayBuffer.getSampleData(0) +
delayBuffer.getNumSamples())
                    {
                           delayWritePtr = delayBuffer.getSampleData(0);
                    }
             }

        // Increment pointer
             bufferPtr++;
      }

    // Send buffer to vstSynthFilter where it is replaced with filtered data
      hpeqFilter.processSamples(buffer.getSampleData(0, 0), numSamples);

    // All processing happens in only one channel for speed; the other channel is filled here.
      buffer.addFrom(1, 0, buffer, 0, 0, numSamples);

    // Apply overall output gain to buffer before playback
      buffer.applyGain(0, numSamples, 10 * getParameter(outputGainParam)); // Controlled by
vstSynthEditor::outputGainSlider

      // In case we have more outputs than inputs, we'll clear any output
      // channels that didn't contain input data, (because these aren't
      // guaranteed to be empty - they may contain garbage).
      for (int i = getNumInputChannels(); i < getNumOutputChannels(); ++i)
      {
             buffer.clear (i, 0, buffer.getNumSamples());
      }
}

//==============================================================================
bool vstSynthAudioProcessor::hasEditor() const
{
      return true; // (change this to false if you choose to not supply an editor)
}

AudioProcessorEditor* vstSynthAudioProcessor::createEditor()
{
      return new vstSynthAudioProcessorEditor (this);
}

//==============================================================================
void vstSynthAudioProcessor::getStateInformation (MemoryBlock& destData)
{
      // You should use this method to store your parameters in the memory block.
      // You could do that either as raw data, or use the XML or ValueTree classes
      // as intermediaries to make it easy to save and load complex data.
}
```

```
void vstSynthAudioProcessor::setStateInformation (const void* data, int sizeInBytes)
{
	// You should use this method to restore your parameters from this memory block,
	// whose contents will have been created by the getStateInformation() call.
}

//==============================================================================
// This creates new instances of the plugin..
AudioProcessor* JUCE_CALLTYPE createPluginFilter()
{
	return new vstSynthAudioProcessor();
}
```

```
/*
vstSynth.h - header for vstSynth.cpp
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/



#include "StkIncludes.h"
#include "vstSynthOscillator.h"

class vstSynthSound : public SynthesiserSound
{
public:
    vstSynthSound();

    bool appliesToNote(const int /*midiNoteNumber*/);
    bool appliesToChannel(const int /*midiChannel*/);
};

class vstSynthVoice : public SynthesiserVoice
{
public:
    vstSynthVoice(float* parameters);
    ~vstSynthVoice();
    float getParameter(int index);

    bool canPlaySound (SynthesiserSound* sound);

    void startNote (const int midiNoteNumber, const float velocity, SynthesiserSound*
/*sound*/, const int /*currrentPitchWheelPosition*/);

    void stopNote (const bool allowTailOff);

    void pitchWheelMoved (const int /*newValue*/);

    void controllerMoved (const int /*controllerNumber*/, const int /*newValue*/);

       void renderNextBlock (AudioSampleBuffer& outputBuffer, int startSample, int numSamples);

private:
    enum Parameters
    {
        // single oscillator
        osc1WaveParam,
        osc1OctaveParam,
        osc1LevelParam,

        osc2WaveParam,
        osc2OctaveParam,
        osc2LevelParam,

        osc3WaveParam,
        osc3OctaveParam,
        osc3LevelParam,

              // envelope
        attackParam,
        decayParam,
        sustainParam,
        releaseParam,

        // LFO
        lfoDestParam,
        lfoWaveParam,
        lfoFreqParam,
        lfoDevParam,

        // filter
        filterTypeParam,
        filterCutoffParam,
        filterResonanceParam,
        filterGainParam,

        // delay
        delayTimeParam,
        delayFeedbackParam,
        delayGainParam,

        // output
        noiseParam,
        driveParam,
        outputGainParam,

        totalNumParams
    };

    float* localParameters;

       vstSynthOscillator oscillator1;
       vstSynthOscillator oscillator2;
       vstSynthOscillator oscillator3;

       stk::ADSR envelope;
    bool vibratoOn;

    float oscOutput;

    double freq;
       double keyLevel;
};
```

```
/*
vstSynth.cpp - JUCE Voice and Sound classes for synthesiser implementation
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

#include "vstSynth.h"

vstSynthSound::vstSynthSound()
{
}

bool vstSynthSound::appliesToNote(const int /*midiNoteNumber*/)
{
       return true;
}
bool vstSynthSound::appliesToChannel(const int /*midiChannel*/)
{
       return true;
}

//===========================================================================

vstSynthVoice::vstSynthVoice(float* parameters)
{
       localParameters = parameters;
}

vstSynthVoice::~vstSynthVoice()
{
}

float vstSynthVoice::getParameter (int index)
{
       switch (index)
       {
            // single oscillator
        case osc1WaveParam:          return localParameters[0];
        case osc1OctaveParam:        return localParameters[1];
        case osc1LevelParam:         return localParameters[2];

        case osc2WaveParam:          return localParameters[3];
        case osc2OctaveParam:        return localParameters[4];
        case osc2LevelParam:         return localParameters[5];
}

        case osc3WaveParam:          return localParameters[6];
        case osc3OctaveParam:        return localParameters[7];
        case osc3LevelParam:         return localParameters[8];

            // envelope
        case attackParam:           return localParameters[9];
        case decayParam:            return localParameters[10];
        case sustainParam:          return localParameters[11];
        case releaseParam:          return localParameters[12];

            // LFO
        case lfoDestParam:          return localParameters[13];
        case lfoWaveParam:          return localParameters[14];
        case lfoFreqParam:          return localParameters[15];
        case lfoDevParam:           return localParameters[16];

            // filter
        case filterTypeParam:       return localParameters[17];
        case filterCutoffParam:     return localParameters[18];
        case filterResonanceParam:  return localParameters[19];
        case filterGainParam:       return localParameters[20];

            // delay
        case delayTimeParam:        return localParameters[21];
        case delayFeedbackParam:    return localParameters[22];
        case delayGainParam:        return localParameters[23];

            // output
        case noiseParam:            return localParameters[24];
        case driveParam:            return localParameters[25];
        case outputGainParam:       return localParameters[26];

        default:                    return 0.0f;
       }
}

bool vstSynthVoice::canPlaySound (SynthesiserSound* sound)
{
       return dynamic_cast <vstSynthSound*> (sound) != 0;
}

void vstSynthVoice::startNote (const int midiNoteNumber, const float velocity,
SynthesiserSound* /*sound*/, const int /*currrentPitchWheelPosition*/)
{
       freq = MidiMessage::getMidiNoteInHertz(midiNoteNumber);
       keyLevel = velocity/127;

       oscillator1.noteOn(freq, getParameter(osc1OctaveParam), keyLevel *
getParameter(osc1LevelParam), getParameter(osc1WaveParam));
       oscillator2.noteOn(freq, getParameter(osc2OctaveParam), keyLevel *
getParameter(osc2LevelParam), getParameter(osc2WaveParam));
       oscillator3.noteOn(freq, getParameter(osc3OctaveParam), keyLevel *
getParameter(osc3LevelParam), getParameter(osc3WaveParam));

       envelope.setAllTimes(getParameter(attackParam), getParameter(decayParam),
getParameter(sustainParam)/10, getParameter(releaseParam));
       envelope.keyOn();
}
```

```
void vstSynthVoice::stopNote (const bool allowTailOff)
{
       oscillator1.noteOff(0.0001);
       oscillator2.noteOff(0.0001);
       oscillator3.noteOff(0.0001);
       envelope.keyOff();
    clearCurrentNote();
}

void vstSynthVoice::pitchWheelMoved (const int /*newValue*/)
{
}

void vstSynthVoice::controllerMoved (const int index, const int newValue)
{
}

void vstSynthVoice::renderNextBlock (AudioSampleBuffer& outputBuffer, int startSample, int
numSamples)
{
    envelope.setAllTimes(getParameter(attackParam), getParameter(decayParam),
getParameter(sustainParam)/10, getParameter(releaseParam));

       oscillator1.update(getParameter(osc1OctaveParam), keyLevel *
getParameter(osc1LevelParam), getParameter(osc1WaveParam));
       oscillator2.update(getParameter(osc2OctaveParam), keyLevel *
getParameter(osc2LevelParam), getParameter(osc2WaveParam));
       oscillator3.update(getParameter(osc3OctaveParam), keyLevel *
getParameter(osc3LevelParam), getParameter(osc3WaveParam));

    if ((int) getParameter(lfoDestParam) == 1)
    {
        vibratoOn = true;
    }
    else
    {
        vibratoOn = false;
    }

       oscillator1.fillBuffer(outputBuffer, startSample, numSamples, &envelope,
getParameter(noiseParam), getParameter(lfoFreqParam), getParameter(lfoDevParam), vibratoOn);
       oscillator2.fillBuffer(outputBuffer, startSample, numSamples, &envelope,
getParameter(noiseParam), getParameter(lfoFreqParam), getParameter(lfoDevParam), vibratoOn);
       oscillator3.fillBuffer(outputBuffer, startSample, numSamples, &envelope,
getParameter(noiseParam), getParameter(lfoFreqParam), getParameter(lfoDevParam), vibratoOn);
}
```

```
/*
vstSynthOscillator.h - header file for vstSynthOscillator.cpp
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
STK LICENSE:

STK WWW site: http://ccrma.stanford.edu/software/stk/

The Synthesis ToolKit in C++ (STK)
Copyright (c) 1995-2011 Perry R. Cook and Gary P. Scavone

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

Any person wishing to distribute modifications to the Software is
asked to send the modifications to the original developer so that they
can be incorporated into the canonical version.  This is, however, not
a binding provision of this license.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

#include "StkIncludes.h"
#include "../JuceLibraryCode/JuceHeader.h"

using namespace stk;

class vstSynthOscillator
```

```
{
public:
    vstSynthOscillator();
    ~vstSynthOscillator();

    void setFrequency(StkFloat newFrequency);
       float getFrequency();

       void setOctave(int newOctave);
    int getOctave();

       void setGain(float newGain);
       float getGain();

       void setWaveform(int newWaveform);
       int getWaveform();

       void noteOn(float newFrequency, int newOctave, float newGain, int newWaveform);
       void update(int newOctave, float newGain, int newWaveform);
    void noteOff(float newAmplitude);

    StkFloat tick();
       void fillBuffer(AudioSampleBuffer& buffer, int startSample, int numSamples, ADSR*
envelope, float noiseGain, float lfoFreq, float lfoDev, bool vibratoOn);

private:
    float currentSample;

       float frequency;
       int octave;
       int waveform;
       float gain;
    float noiseLevel;
       float vibratoDepth;
       float vibratoSpeed;

       SineWave sineOut;
       BlitSaw sawOut;
       BlitSquare squareOut;
       Noise noise;
    Modulate vibrato;
};
```

```
/*
vstSynthOscillator.cpp - generates waveforms for synth from STK classes
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
STK LICENSE:

STK WWW site: http://ccrma.stanford.edu/software/stk/

The Synthesis ToolKit in C++ (STK)
Copyright (c) 1995-2011 Perry R. Cook and Gary P. Scavone

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

Any person wishing to distribute modifications to the Software is
asked to send the modifications to the original developer so that they
can be incorporated into the canonical version.  This is, however, not
a binding provision of this license.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR
ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

/*
 This class is used to generate the raw waveforms used by the synth.
 Waveforms are generated using the STK SineWave, BlitSquare and BlitSaw
 classes.
*/
```

```
#include "vstSynthOscillator.h"

vstSynthOscillator::vstSynthOscillator()
{
       // Initial parameters
    frequency = 220;
       octave = 1;
       waveform = 0;
       gain = 0;
    noiseLevel = 0;
}

vstSynthOscillator::~vstSynthOscillator()
{

}

void vstSynthOscillator::setFrequency(StkFloat newFrequency)
{
    frequency = newFrequency; // Controlled by MIDI key number
}

float vstSynthOscillator::getFrequency()
{
       return this->frequency;
}

void vstSynthOscillator::setOctave(int newOctave)
{
       this->octave = 1<<newOctave; // Controlled by vstSynthEditor::oscXOctaveSlider
}

int vstSynthOscillator::getOctave()
{
       return this->octave;
}

void vstSynthOscillator::setGain(float newGain)
{
    this->gain = newGain; // Controlled by vstSynthEditor::oscXLevelSlider
}

float vstSynthOscillator::getGain()
{
       return this->gain;
}

void vstSynthOscillator::setWaveform(int newWaveform)
{
       this->waveform = newWaveform; // Controlled by vstSynthEditor::oscXWaveComboBox
}

int vstSynthOscillator::getWaveform()
{
       return this->waveform;
}
```

```
// Allows for setting of all parameters in one command. Called on by vstSynth on MIDI note on
void vstSynthOscillator::noteOn(float newFrequency, int newOctave, float newGain, int
newWaveform)
{
       this->setWaveform(newWaveform);
       this->setOctave(newOctave);
       this->setFrequency(newFrequency);
       this->setGain(newGain);
}

// Checks current settings against any potential changes and updates as necessary
void vstSynthOscillator::update(int newOctave, float newGain, int newWaveform)
{
    if (newOctave != this->getOctave())
       {
              this->setOctave(newOctave);
       }
       if (newWaveform != this->getWaveform())
       {
              this->setWaveform(newWaveform);
       }
       if (newGain != this->getGain())
       {
              this->setGain(newGain);
       }
}

// Silences output after MIDI key release signal received
void vstSynthOscillator::noteOff(float newGain)
{
       this->setGain(newGain);
}

// Generate a single output of the appropriate waveform.
StkFloat vstSynthOscillator::tick()
{
       switch (waveform)
       {
        case 0:
              {
                     return this->getGain() * this->sineOut.tick();
              }
        case 1:
              {
                     return this->getGain() * this->sawOut.tick();
              }
        case 2:
              {
                     return this->getGain() * this->squareOut.tick();
              }
        default:
              {
                     return 0;
              }
       }
}

// Fills an AudioSampleBuffer with the oscillators output and applies vibrato effect if
```

```
enabled.
void vstSynthOscillator::fillBuffer(AudioSampleBuffer& buffer, int startSample, int numSamples,
ADSR* envelope, float noiseGain, float lfoFreq, float lfoDev, bool vibratoOn)
/*
 buffer
 startSample
 numSamples
 envelope - reference to ADSR envelope generator from calling vstSynthVoice
 noiseGain - level of noise generator
 lfoFreq - used to set vibratoSpeed
 lfoDev - used to set vibratoDepth
 vibratoOn - boolean set by caller to enable/disable vibrato
 */

{
    // Pointer to buffer
    float* bufferPtr = buffer.getSampleData(0, startSample);

    // Updates vibrato parameters if enabled
       if (vibratoOn)
    {
        if (vibratoSpeed != lfoFreq) // Controlled by vstSynthEditor::lfoFreqSlider
        {
            vibratoSpeed = lfoFreq;
            vibrato.setVibratoRate(vibratoSpeed);
        }

        if (vibratoDepth != lfoDev) // Controlled by vstSynthEditor::lfoDevSlider
        {
            vibratoDepth = lfoDev;
            vibrato.setVibratoGain(vibratoDepth);
        }
    }
    // Reset parameters to 0 when disabled
    else
    {
        vibratoSpeed = 0;
        vibratoDepth = 0;
    }

    if (noiseGain <= 0.01)
    {
        noiseLevel = 0;
    }
    else
    {
        noiseLevel = 0.01 * noiseGain;
    }

    /*
     Fills buffer with output from generators with appropriate envelope applied.
     Noise is also added to each oscillator to allow for proper enveloping with each key
press/release.
     */
    for (int currentSample = 0; currentSample < numSamples; currentSample++)
       {
        switch (waveform)
              {
```

```
            case 0: //Sine
                    {
                sineOut.setFrequency(octave * (frequency + vibrato.tick()));
                            bufferPtr[currentSample] += (float) (envelope->tick() * (noiseLevel
* noise.tick() + gain * sineOut.tick()));
                            break;
                    }
            case 1: //Saw
                    {
                sawOut.setFrequency(octave * (frequency + vibrato.tick()));
                            bufferPtr[currentSample] += (float) (envelope->tick() * (noiseLevel
* noise.tick() + gain * sawOut.tick()));
                            break;
                    }
            case 2: //Square
                    {
                squareOut.setFrequency(octave * (frequency + vibrato.tick()));
                            bufferPtr[currentSample] += (float) (envelope->tick() * (noiseLevel
* noise.tick() + gain * squareOut.tick()));
                            break;
                    }
            default:
                    {
                            break;
                    }
             }
       }
}
```

```
/*
vstSynthFilter.h - header file for vstSynthFilter.cpp
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

# include "StkIncludes.h"
#include "../JuceLibraryCode/JuceHeader.h"

using namespace stk;

class vstSynthFilter// : public IIRFilter
{
public:
       vstSynthFilter();
       ~vstSynthFilter();

       void setCoefficients(int newType, float newGain, float newFrequency, float
newResonance);

       void setType(int newType);
       void setGain(float newGain);
    void setFrequency(float newFrequency);
    void setResonance(float newResonance);

    void processSamples(float* samples, int numSamples) noexcept;

       void reset();

private:
    void defineParameters(); // calculates g, g0, and epsilon
       void updateFilter(); // calculates all coefficients

       float G0;           //reference gain
       float G;                   //filter gain
       float Gb;           //gain at deltaW

       float Wc;           //center/cutoff frequency
       int filterType;     //0 = LP, 1 = BP, 2 = HP

       float N;                   //analog filter order
    float W0;       //center frequency
       float W1;           //lower transition frequency
       float W2;           //upper transition frequency
       float dW;           //W2-W1
       float g;             //G^(1/N)
    float g0;        //G0^(1/N)
       float phi;           //pi/6

    float epsilon;
       float beta;     // hpeq beta
       float c0;            //cos(W0)
       float s1;            //sin(phi)

       float gSq;
       float betaSq;
       float g0Sq;
       float c0Sq;

       float D0;
    // second order coefficients
    float b00, b01, b02;
    float a00, a01, a02;
    // second order internal states
    float x01, x02;
    float y01, y02;

       float D1;
    // fourth order coefficients
    float b10, b11, b12, b13, b14;
    float a10, a11, a12, a13, a14;
    // fourth order internal states
    float x11, x12, x13, x14;
    float y11, y12, y13, y14;
    float w11, w12, w13, w14;


    float Gr;       //resonator gain
    float G0r;      //resonator reference gain
    float dWr;      //resonator width 0.001
    float betaR;    //resonator beta
    float c0r;      //resonator center freq cosine
    float Gbr;      //resonator gain at dWr

    // resonator coefficients
    float b20, b21, b22;
    float a20, a21, a22;
    // resonator internal states
    float x21, x22;
    float y21, y22;

};
```

```
/*
vstSynthFilter.cpp - implementation of high order parametric filter
                     and resonant parametric equalizer
Copyright (C) 2012 Gabriel Olochwoszcz

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

#include "vstSynthFilter.h"

vstSynthFilter::vstSynthFilter()
{
	// Initialize fixed design parameters
    G0 = 0.0001; // reference gain -120db
	N = 3.0;
	phi = PI/6;
	s1 = sin(phi);

    // Initialize internal filter states
    // 2nd order
	x01 = x02 = 0;
	y01 = y02 = 0;

    // 4th order
	x11 = x12 = x13 = x14 = 0;
	y11 = y12 = y13 = y14 = 0;
	w11 = w12 = w13 = w14 = 0;

    // resonator
	x21 = x22 = 0;
	y21 = y22 = 0;
}

vstSynthFilter::~vstSynthFilter()
{
}

// Set all parameters and update filter with new coefficients
void vstSynthFilter::setCoefficients(int newType, float newGain, float newFrequency, float
newResonance)
{
	this->setType(newType);
	this->setGain(newGain);
	this->setFrequency(newFrequency);
	this->setResonance(newResonance);
	this->updateFilter();
```

```
}

// Set filter type - 0=lp,1=bp,2=hp
void vstSynthFilter::setType(int newType)
{
       this->filterType = newType;
       this->updateFilter();
}

// Set 2nd and 4th order gain
void vstSynthFilter::setGain(float newGain)
{
       this->G = newGain;
       this->updateFilter();
}

// Set cutoff frequency
void vstSynthFilter::setFrequency(float newFrequency)
{
       this->Wc = newFrequency;
       this->updateFilter();
}

// Set resonator gain above 2nd/4th order gain
void vstSynthFilter::setResonance(float newResonance)
{
       this->Gr = newResonance * G;
       this->updateFilter();
}

// Reset internal states after changes
void vstSynthFilter::reset()
{
       x01 = x02 = 0;
       y01 = y02 = 0;

       x11 = x12 = x13 = x14 = 0;
       y11 = y12 = y13 = y14 = 0;
    w11 = w12 = w13 = w14 = 0;

       x21 = x22 = 0;
       y21 = y22 = 0;
}

// Calculate g, g0, and epsilon
void vstSynthFilter::defineParameters()
{
       // Filter gain > reference gain
    // Set cutoff gain to 6 dB below passband gain
       if (G >= G0)
       {
              Gb = G/2;
       }

    // Filter gain < reference gain
    // Set cutoff gain to 6 db below reference gain
       if (G < G0)
       {
              Gb = G0/2;
        }

    // Resonator reference gain is the passband gain
       G0r = G;

    // 6 dB below resonator peak
       Gbr = Gr/2;
       g = pow(G,1.0f/N);
       g0 = pow(G0,1.0f/N);

       epsilon = sqrt((G*G - Gb*Gb)/(Gb*Gb - G0*G0));

    c0r = cos(Wc);

       // squared values
       gSq = g*g;
       g0Sq = g0*g0;
}


// Calculate all coefficients
void vstSynthFilter::updateFilter()

// All resonator coefficients have "r" appended
{
    reset();
       defineParameters();

       // Set frequency parameters for each filter type
       switch (filterType) {

    // lowpass
       case 0:
              {
                     W0 = 0;
                     W1 = 0;
                     W2 = Wc;
                     dW = W2 - W1;
                     beta = pow(epsilon,-1.0f/N) * tanf(dW/2);
                     betaSq = beta*beta;

                  c0 = cos(W0);
                     c0Sq = c0*c0;

                     // Lowpass second order section
                     D0 = beta + 1;
                     b00 = (g*beta + g0)/D0;
                     b01 = (g*beta - g0)/D0;
                     b02 = 0;
                     a01 = (beta - 1)/D0;
                     a02 = 0;

                     // Lowpass fourth order section
                     D1 = betaSq + 2*s1*beta + 1;
                     b10 = (gSq*betaSq + 2*g*g0*s1*beta + g0Sq)/D1;
                     b11 = 2*(gSq*betaSq - g0Sq)/D1;
                     b12 = (gSq*betaSq  -  2*g*g0*s1*beta + g0Sq)/D1;
```

```
                b13 = 0;
                b14 = 0;
                a11 = 2*(betaSq - 1)/D1;
                a12 = (betaSq - 2*s1*beta + 1)/D1;
                a13 = 0;
                a14 = 0;

                break;
            }

// bandpass
    case 1:
            {
        // Approx two octave bandwidth
                if (Wc < 0.025)
        {
            Wc = 0.025;
        }
        W0 = Wc;
                W1 = Wc/2;
                float tanSq = tan(Wc/2)*tan(Wc/2);
                W2 = 2*atan(tanSq / tan(W1/2));
                dW = W2 - W1;

                beta = pow(epsilon,-1.0f/N) * tanf(dW/2);
                betaSq = beta*beta;

                c0 = cos(W0);
                c0Sq = c0*c0;


                // second order section coefficients
                D0 = beta + 1;
                b00 = (g0 + g*beta)/D0;
                b01 =  -2*g0*c0/D0;
                b02 = (g0 - g*beta)/D0;
                a01 =  -2*c0/D0;
                a02 = (1 - beta)/D0;

                // fourth order section coefficients
                D1 = betaSq + 2*s1*beta + 1;
                b10 = (gSq*betaSq + 2*g*g0*s1*beta + g0Sq)/D1;
                b11 =  -4*c0*(g0Sq + g*g0*s1*beta)/D1;
                b12 = 2*(g0Sq*(1 + 2*c0Sq) - gSq*betaSq)/D1;
                b13 =  -4*c0*(g0Sq - g*g0*s1*beta)/D1;
                b14 = (gSq*betaSq - 2*g*g0*s1*beta + g0Sq)/D1;
                a11 =  - 4*c0*(1 + s1*beta)/D1;
                a12 = 2*(1 + 2*c0Sq - betaSq)/D1;
                a13 =  -4*c0*(1 - s1*beta)/D1;
                a14 = (betaSq - 2*s1*beta + 1)/D1;

                break;

            }

// highpass
    case 2:
            {
```

```
                        W0 = PI;
                        W1 = Wc;
                        W2 = PI;
                        dW = W2 - W1;

                        beta = pow(epsilon,-1.0f/N) * tanf(dW/2);
                        betaSq = beta*beta;

                        c0 = cos(W0);
                        c0Sq = c0*c0;

                        // Highpass second order section
                        D0 = beta + 1;
                        b00 = (g*beta + g0)/D0;
                        b01 = -(g*beta - g0)/D0;
                        b02 = 0;
                        a01 = -(beta - 1)/D0;
                        a02 = 0;

                        // Highpass fourth order section
                        D1 = betaSq + 2*s1*beta + 1;
                        b10 = (gSq*betaSq + 2*g*g0*s1*beta + g0Sq)/D1;
                        b11 = -2*(gSq*betaSq - g0Sq)/D1;
                        b12 = (gSq*betaSq  -  2*g*g0*s1*beta + g0Sq)/D1;
                        b13 = 0;
                        b14 = 0;
                        a11 = -2*(betaSq - 1)/D1;
                        a12 = (betaSq - 2*s1*beta + 1)/D1;
                        a13 = 0;
                        a14 = 0;

                        break;
                }

        default:
                break;
        }

     // Resonator bandwidth
        // fixed ratio to gain to keep shape of resonator as gain changes
        dWr = 0.01 * G;

        // resonator section coefficients
        betaR = sqrt((Gbr*Gbr - G0*G0)/(Gr*Gr - Gbr*Gbr))*tanf(dWr/2);

        b20 = (G0r + Gr*betaR)/(1 + betaR);
        b21 = -2*(G0r*c0r/(1 + betaR));
        b22 = (G0r - Gr*betaR)/(1 + betaR);
        a20 = 1;
        a21 = -2*c0r/(1 + betaR);
        a22 = (1 - betaR)/(1 + betaR);
}

void vstSynthFilter::processSamples(float* const samples, const int numSamples) noexcept
{
        for (int currentSample = 0; currentSample < numSamples; ++currentSample)
        {
                // Get data from buffer
```

```
            const float x00 = samples[currentSample];

                  // second order section
                  float y00 = b00 * x00
                         + b01 * x01
                         + b02 * x02
                         - a01 * y01
                         - a02 * y02;

            // Update internal states
                  x02 = x01;
                  x01 = x00;

                  y02 = y01;
                  y01 = y00;

                  // fourth order section
            const float w10 = y00
                - a11 * w11
                - a12 * w12
                - a13 * w13
                - a14 * w14;

            float y10 = b10 * w10
                + b11 * w11
                + b12 * w12
                + b13 * w13
                + b14 * w14;

            // Update internal states
            w14 = w13;
            w13 = w12;
            w12 = w11;
            w11 = w10;

                  // resonator section
                  const float x20 = y10;

                  float y20 = b20 * x20
                         + b21 * x21
                         + b22 * x22
                         - a21 * y21
                         - a22 * y22;

            // Update internal states
                  x22 = x21;
                  x21 = x20;

                  y22 = y21;
                  y21 = y20;

            // Write cascaded output to buffer
                  samples[currentSample] = y20;
           }
    }
```