(:wizoo:)

FORTUNE/SCHOFFHAUZER/HAUPT

VISUAL VST/i PROGRAMMING

A Comprehensive Guide to Creating VST-FX and Instruments with Synthedit

- The cookbook for Synthedit maniacs
- Build your VST-FX or VSTi without higher programming languages
- The easiest way to start successful VST/i programming

H. G. Fortune (Editor), Peter Schoffhauzer, and David Haupt

Visual VST/i-Programming

A Comprehensive Guide to Creating
VST-FX and Instruments with Synthedit

Publisher Peter Gorges

Authors H. G. Fortune (Editor), Peter Schoffhauzer, and David Haupt

Cover art Complete Design, www.cmplt.com
Interior design and layout Uwe Senkler

ISBN 978-3-934903-59-3

Foreword by the Editor

As Stevie Winwood put it, “While you see a chance, take it.” Indeed: A chance meeting at the supermarket next door sparked an idea that resulted in the book you hold in your hands.

SynthEdit, in turn, sparked a VST/i boom. It offers even novices the opportunity to create a VST/i of their own simply by connecting modules in the SE environment—no programming languages such as C++ required. This makes creating labyrinthine synthesizers and effects for use in any VST-compatible audio software or sequencer kid’s stuff rather than rocket science. Simply stack the building blocks—that is, modules. What’s more, to beef up its already formidable powers, SynthEdit accepts third-party modules.

Mastering SynthEdit is fun, and what better way to learn than by doing. But without a guide to show hidden shortcuts and steer you clear of obstacles and dead-ends, you may find the going too tough for your liking.

With the inside scoop on audio processing provided in this book, you will better understand the basics and background of audio effects and synthesis. Every turn of a page takes you that much closer to realizing your vision of an effect or synthesizer in the virtual realm.

This book is your compass; use it to explore the seductive world of VST/i. Seasoned sound-sculptor or newbie, you will soon find yourself creating exciting musical tools.

As this book evolved, I often had cause to call on third-party developers’ help. Though an obliging spirit prevails in the SynthEdit community, I was dumbfounded to discover how generously these kind people shared the fruits of their labor.

Specifically, I wish to express my heartfelt gratitude to the following gentlemen, princes among men all, for sharing their modules by way of this book:

Ralph Gonzales, Dave Haupt, Rob Herder, Rick Jelliffe, Butch Kratzer, Simonluca Laitempergher, Oli Larkin, Marc Lindahl, Kelly D. Lynch,

Etric van Mayer, Lance Putnam, Peter Schoffhauzer, Guido Sconamiglio, and Dan Worrall.

A big round of applause and kudos go to Kelly D. Lynch for contributing modules unavailable to the public.

Thank you Lance Putnam for proofreading, Vera Kinter for providing free GUI sets, and Hermann Seib for contributing the VST Host on www.wizoobooks.com/synthedit for free.

Jeff McClintock provided SynthEdit, a phenomenally flexible tool for creating anything from basic to advanced VST/i, even for C++ programmers using the SE SDK. And he merits special thanks for it.

The Chapters 1 to 4 are written by Peter Schoffhauzer, Chapter 5 which was done by David Haupt, and the foreword and the appendix (A Brief History of Synthesizers) by H. G. Fortune.

H. G. Fortune

Alfter/Bonn, October 2006

Table of Contents

1

Welcome to the Wonderful World of SynthEdit

What's SynthEdit?

Decades ago, synthesizers were ponderous beasts with a nest of wires sprouting from their panels. To create sounds, one would plug these patch cords into different modules such as oscillators, filters, envelope generators, and amplifiers. The signal path—that is, the order in which one connected modules—shaped the sound. Hard-wired synths, far easier to port and use, later won the day. Modular synthesis reared its head again with the arrival of digital technology. SynthEdit is a software application enabling today's users to build audio applications taking the modular approach of yesteryear. Its modules put many processing options at your fingertips. Few who practiced the cumbersome chore of juggling real patch cords would contend that plugging in virtual patch cords is anything but a quantum leap in convenience.

VST Technology

Storability is what makes SynthEdit stand out in the crowd of modular software applications. Create a synth or an effect, save it in VST or VST/i format, and you may share it with or sell it to people owning VST/i-compatible hosts.

A trademark of Steinberg Media Technologies GmbH, VST stands for Virtual Studio Technology. Steinberg rolled out Cubase VST for PCs in 1996. VST instruments and effects are separate modules rather than features of the main application. In 1997, Steinberg released the format as an open standard, inviting third-party developers to market their own plug-ins.

Figure 1.1

SynthEdit is a fast, convenient tool for creating VST/i. Put it in the hands of people lacking programming experience or deep DSP knowledge, and they can still deliver the goods. Courtesy of custom sub-controls, you can skin and customize the GUI just about any way you see fit. And with the many built-in and third-party modules, you have beau coup sound-sculpting tools at hand.

What's a VST Plug-in?

Typically, a VST plug-in is a Windows dll module that loads into the host application, whose numbers are growing at a scary rate. Instruments usually produce sound, and lack audio inputs. MIDI events such as note on, note off, control change, pitchbend and mod wheel messages serve control purposes. Effects plug into individual audio tracks or master bus' inserts or sends (depending on host). Most feature one or more inputs and outputs, and a set of adjustable parameters.

A VST plug-in comprises a DSP module that performs the actual processing, and the graphic user interface, or GUI for short. Some VST plug-ins lack a GUI, relying on the host panel for parameter adjustment, though most offer one. It lets you tweak and toggle to your heart's content various graphical controls such as knobs, sliders, buttons, list boxes, text entry boxes, and so forth. SynthEdit VST plug-ins always feature a GUI. The Save As VST command automatically creates the VST parameters the host and plug-in need to interact. End-users rarely see anything other than the GUI.

The SynthEdit Structure window opens by default when you launch a new project. It shows the DSP module, that is, the plug-in's internal structure. The Panel Edit window opens when you right-click the Structure window and select Panel Edit. It lets you edit this GUI. The section "What Are Sub-controls?" from page 193 onwards discusses graphical features and their properties in detail. The following sections deal mainly with DSP effects and synths' underlying modular structure.

X-Raying SynthEdit's Hierarchic Structure

Modules

Modules are your building blocks. They appear in the Structure window as boxes with various input and output plugs. Figure 1.2 shows a Moog Filter module with three in plugs (Signal, Pitch, and Resonance) on the left, and one out plug on the right. The Signal Out plug of a slider (Insert > Controls > Slider) connects to the Pitch plug, so the slider controls the filter's cutoff frequency.

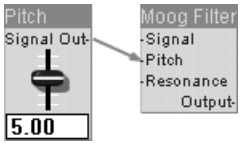


Figure 1.2

Heads up: Add modules using the Insert menu in the menu bar or the Structure window's right-click menu. Some modules are built in; others appear in the default SynthEdit folder's "modules" subdirectory. SynthEdit automatically lists all folders and modules found in the "modules" directory, so you can create folders for third-party modules and copy the modules there.

Plug Types

Voltage Plugs

Most plugs are blue voltage pins. They serve mainly for audio signals and control voltages. The standard range for control voltages is 0 to 10 volts, and –10 to +10 volts for audio signals. Plugging two or more cords into the same input adds the signals.

List Plugs

Some modules plugs' colors vary. For example, an Oscillator (Insert > Waveform > Oscillator) has a green Waveform plug. These plugs reference list selections, meaning that green input plugs only accept list plugs. These include List Entry, List Entry2, (Insert > Controls) and Voltage To List (Insert > Conversion) modules. A list input accepts one input only. A list output, in turn, connects to more list inputs, but only if they are of the same type, such as the Waveform selector of two Oscillator modules.

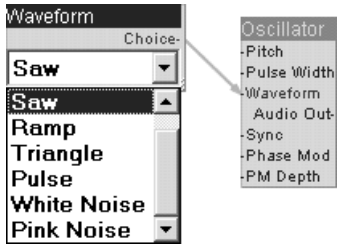


Figure 1.3

Float Plugs

Serving mainly to communicate with GUI modules, these plugs come in a different shade of blue. Float denotes the data type. They may have any real number value.

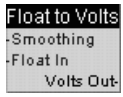


Figure 1.4

MIDI Plugs

The green MIDI in and MIDI out plugs shuttle MIDI data such as note on, note off, pitch bend, mod wheel, aftertouch, program change, and control change events to and fro.

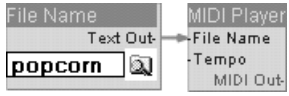


Figure 1.5

Text Plugs

Though these crimson text plugs enable mainly filename entry as shown in figure 1.5, they also serve other purposes. GUI text plugs, for example, provide caption titles for sub-controls. See the section “What Are Sub-controls?” on page 193 for details.

Spare Plugs

One SynthEdit plug type automatically clones itself when connected to another module, for instance, a switch (Insert > Flow Control > Switch (Many → 1)). Connect a module's output to a spare input plug of the switch, and another spare input plug appears. This means you may connect as many inputs as you wish. Spare plugs may appear either as inputs or outputs.

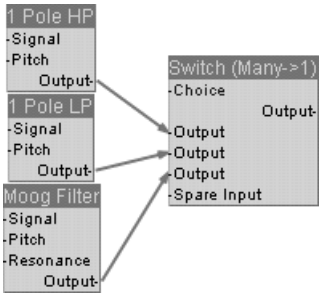


Figure 1.6

GUI Plugs

Some plugs' background is blue rather than gray. Called GUI plugs, they serve chiefly to communicate with the GUI and call sub-control modules home. What sets GUI plugs apart from regular plugs is that they are updated less frequently, about 20 times a second. To learn more, see the sections on GUI plugs and Sub-controls.

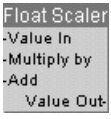


Figure 1.7

Module Properties

Every module offers a Properties window. Right-click a module and select Properties; a window much like this appears.

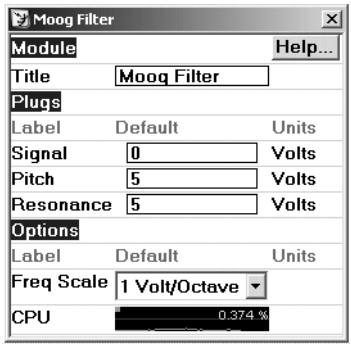


Figure 1.8: Module properties window

Module: Rename the module any way you see fit. The new name appears in the Structure window.

Plugs: This lists input plugs and their default values. Feel free to set a default value for any plug. Note that connecting an input to the plug overrides the values entered here.

Options: Some modules' advanced settings may only be changed in the Properties window, for instance, frequency scale (1 Volt/Octave or 1 Volt/kHz) and resolution settings.

CPU: Appearing below the options is a graph showing CPU use and history. The green dots at the upper left show signal polyphony. See the section "Polyphony" on page 21 to learn more.

Prefabs

SynthEdit lets you load a full set of modules called a prefab. Consisting of several modules—anything from a flanger with an LFO to an entire synth goes—prefabs are usually large and easy to insert. They often hold modules in a container for easier handling. You will find them in the "prefabs" folder. The Insert menu lists se1 files; copy frequently used setups to it for easy, quick loading.

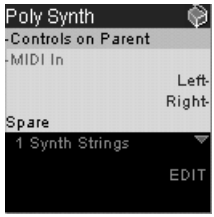


Figure 1.9

Containers

Containers are key building blocks in SynthEdit. They round up several modules, herding them into one logical corral to simplify the structure of a synth, effect, or control feature. Every container affords you a view of its internal structure and a panel window. Below you see the structure of a flanger prefab loaded to a container:

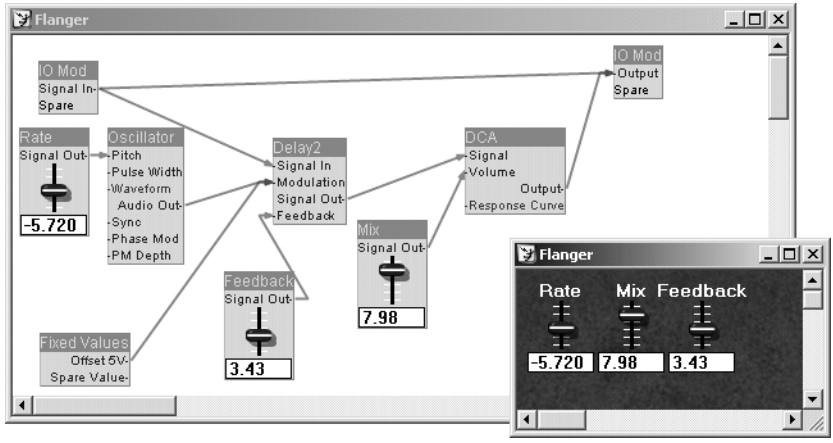


Figure 1.10

An IO Mod module feeds the audio signal into the container. The Delay2 and the DCA modules delay the incoming singal and adjust the wet signal level. Another IO Mod module routes it out of the container. The original dry and wet signals blend to create a typical flanger effect. Check out the section "Conjuring a Flanger" on page 69 for more on flangers.

After creating a container (Insert > Container), double-click it to view its structure. The application automatically generates an IO Mod. Feel free to use the same module for outputs, but note that creating another IO Mod devoted to outputs makes for a tidier setup.

Controls on Module/Parent

Right-click a container and select Properties. The following panel pops up:

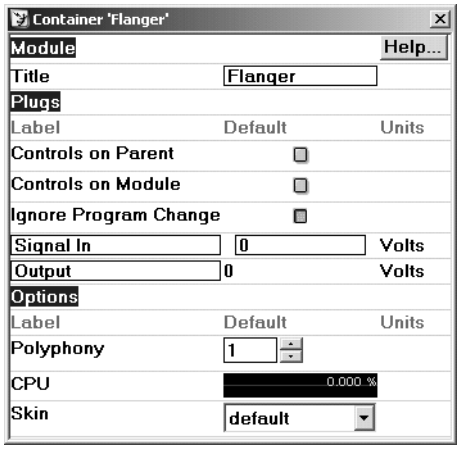


Figure 1.11: Container properties

Enable Controls on Module, and the Structure window's container box shows the panel, as in figure 1.12.

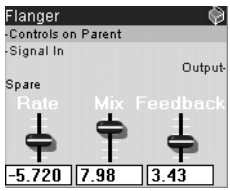


Figure 1.12

Enable Controls on Parent, and these controls also appear on the main container. You can handle them as one group as shown in figure 1.13. This method goes for all control features with sub-controls (Knob, Pitch Bender, List Entry2, Joystick, LED2, and VU Meter).

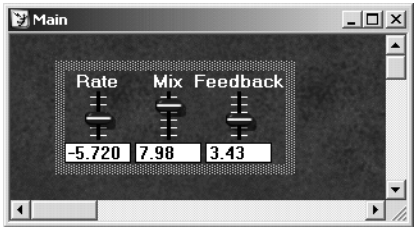


Figure 1.13

Locked Containers

Lock a container if you wish to prevent further editing. Simply clicking the Lock icon in the toolbar locks the active window. Or right-click the container module and tick Locked. The spare plugs disappear when a container locks, deterring you from adding connections. The box icon in the container heading shows the status. An open green box stands for unlocked; a closed gray box locked.

Polyphony

Containers are the key to managing polyphony in SynthEdit, so let's back up and look at what polyphony is all about. A MIDI to CV module automatically converts incoming MIDI data into control voltages. Press several keys at once and the MIDI to CV module produces a polyphonic signal. The application creates how ever many clones all the downstream modules need. Generated internally, these clones are invisible to us. When the signal leaves the container, the application shoehorns it back into monophonic format.

A container's default polyphony is six voices. Give figure 1.11 another gander and note the option called Polyphony. You may set it to any number up to 128, the MIDI maximum for voices. This of course limits the number of voices and clones created during processing.

Heads up:

- The polyphony setting affects MIDI to CV modules. A container or sub-container holds just one MIDI to CV module, so you cannot dial in different polyphony settings within the same container.
- The Properties window shows the number of active clones for each module. Figure 1.14 shows the bottom of an SV Filter's Properties window. The green dots at the CPU graph's top left signal the specified clone is active. In figure 1.14's scenario, max polyphony is four voices, with three currently active.

Figure 1.14

- Use the Special/Voice Combiner module to manually convert a polyphonic into monophonic a signal, for instance, to elicit guitar stomp box-like distortion from a Waveshaper module.
- Polyphony is more complicated. Some modules force the signal into mono format, for example, Delay2, Level Adj and Pan, when every clone shares the same settings and these modules sit at the end of the chain. Apply the effect to each voice and combine them, or combine them and apply the effect—the outcome is the same. But inserting these effects between polyphonic modules imposes polyphony on them. Say you're dealing with the following setup:

  VCA (poly) → Pan (mono) → Delay2 (mono)

  You decide to add a Waveshaper after Delay2, forcing all modules (including Pan and Delay2) into polyphonic operation. This wastes CPU resources, so economize by inserting a Special/Voice Combiner module before Pan. The distortion will of course sound different.

Skins

Use your main container to select a skin for your plug-in. Choose either the Skin option in the container's Properties window, or right-click the container's Panel window and select the Skin menu from the list.

Automation

This right-click property lists all automation parameters used in the container. You may define automation parameters specifically for one patch or globally for all, and configure them as VST parameters or MIDI continuous controllers for user to access.

Module/Prefab Categories

Add modules and prefabs in the toolbar's Insert menu or right-click menu. They are grouped in different submenus, the default categories being:

Controls

This menu comprises mostly parameter controls such as knobs, sliders, switch, list and text entry boxes, mod wheel, pitch bend, joystick, and keyboard as shown in figure 1.15.

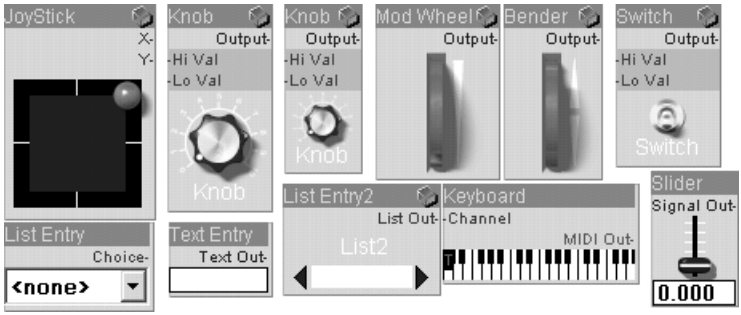


Figure 1.15: Control modules

Note that the Slider offers an Appearance parameter listing various sliders, knobs, and buttons for selection. Predating sub-controls, these are the original SynthEdit controls as shown in figure 1.16. You can skin all the controls shown here and create new types using sub-controls. See the section "What Are Sub-controls?" on page 193 for details.

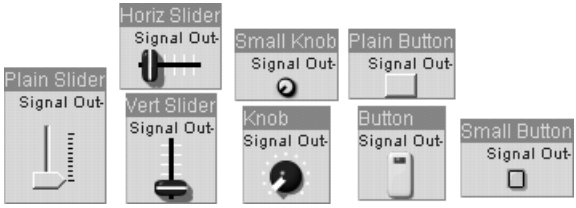


Figure 1.16: Original SynthEdit controls

Visual feedback modules also fall into this category. This group includes the frequency analyzer, LED indicators, peak and VU meters, and a volt meter as shown in figure 1.17.

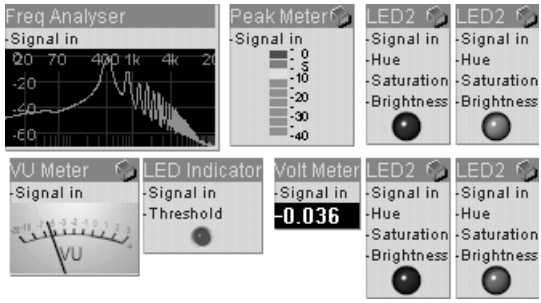


Figure 1.17: Visual feedback modules

Two other modules merit mention, Fixed Values and Image. The former produces constant output voltages; the latter displays bmp or png images on the GUI.

Conversion

This category's modules convert data types, for example, floats to volts and vice versa.

Effects

This category's defaults are delay, flanger, chorus, ring modulation and hard-clipping distortion modules and prefabs. A Clipper module lets you restrict control voltages to any range of your choosing.

Examples

This category offers sample prefabs serving to do things like create low frequency oscillators or a paged panel. You are sure to find these examples are edifying, so be sure to check them out.

Filters

These components cut frequencies from a signal. Filters come in various guises with different characteristics. The most significant difference is the number of poles (the more poles, the steeper its cutoff slope), and the frequencies they let pass.

Low pass: Allows only low frequencies to pass, filtering out frequencies above the cutoff frequency

High pass: Allows only high frequencies to pass, filtering out frequencies below the cutoff frequency

Band pass: Allows only a narrow band of frequencies around the cutoff to pass, filtering out both low and high frequencies

Band stop: Filters out a narrow band of frequencies around the cutoff to, allowing all other frequencies to pass

All pass: Allows all frequencies to pass, but changes the incoming signal's phase. Phaser effects usually feature this filter.

Single-pole low-pass filters also serve to smooth parameter changes and create portamento effects. Use negative voltages for smoother transitions. See portamento_example.se1 on the disk to learn how to create portamento by applying a one-pole LP filter to an oscillator's pitch.

A one-pole high-pass filter set to 20–30 Hz serves to remove DC. Check out the DC_filter.se1 example for more on this.

Flow Control

Switches let the user configure patch cords and choose sound-shaping options, for example, to select an LFO's destination and a signal processor. Figure 1.18 shows how a switch works using the DC filter prefab from the preceding example. The Choice list plug lets you select cutoff frequencies in 0.02, 0.025, and 0.03 increments (20 Hz, 25 Hz. and 30 Hz in 1 Volt/kHz scale).

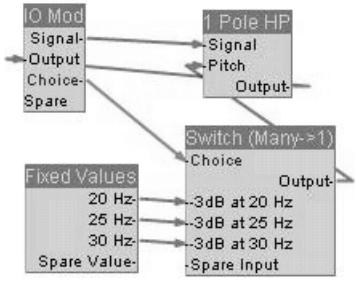


Figure 1.18: The DC filter, again

Input/Output

Use the Sound In and Sound Out modules to access your audio card's input and output channels. Note that the unregistered version confines you to just two output channels.

Wave Player and Wave Recorder modules play files from disk and record any audio signal in different wav formats.

Logic

SynthEdit comes with logic modules that use control voltages to transfer data. 5 volts is the high value denoting true; 0 volts is the low value denoting false.

This category also features different counters, a shift register, and a Monostable module for creating a pulse with a fixed length.

Math

This group contains basic math modules such as Multiply, Subtract and Divide. An Add module is not on board because simply connecting any signals to the same input adds them. The Floor and Ceil modules round the voltage input down and up to the nearest whole number. For example, Floor rounds 1.45 volts down to one volt; Ceil rounds it up to 2 volts.

MIDI

This category's modules mainly handle MIDI data, filtering it, playing it from files, and converting it to control voltages and back to MIDI. The menu also features MIDI sequencer modules, a MIDI Soundfont Player, a Patch Select module for adding patches to your plug-in, and a MIDI Automator module that lets you automate controls using a MIDI mod wheel or controller commands.

Modifiers

This broad category features modules for adjusting levels (VCA—Voltage Controlled Amplifier and Level Adj), cross-fading (X-Mix), panning, and inverting. Two Waveshapers distort signals or impose a customized transfer function on a control voltage, say to use a specific velocity curve. Waveshaper lets you draw the transfer curve manually by dragging points, while Waveshaper2 accepts a mathematical equation. A Rectifier inverts voltages' charge from negative to positive. A Comparator compares two levels, and a Peak Follower tracks a signal's envelope. The Quantizer module constrains the number of input steps, much like bit reduction. Use this to do things like confine a slider or a knob's output value to whole numbers. Sample And Hold holds the incoming level until it is retriggered.

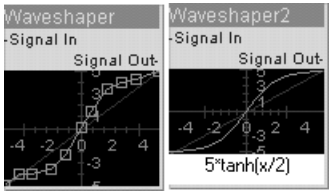


Figure 1.19: Waveshapers

Obsolete

This category contains an ancient version of Delay for compatibility. Use Effects > Delay2 instead.

Special

Many different modules call this group home. Two modules detect and clean denormal numbers generated by some third-party delays, filters, and other modules' feedback loops, unnecessarily consuming CPU power. Monitor watches over MIDI signals and plugs' status changes. The OS Command module carries out system commands. Voice Combiner converts polyphonic signals to mono. Random Voltage generates—you guessed it—random voltages.

SynthEdit does not enable feedback, bar one exception. You can use the Special > Feedback (delayed) module to create feedback loops. Connect the Feedback module's output to any module's input, and the latter's output to the Feedback module's input to configure feedback loops for delays and the like. Note that this feedback is not sample based, and its latency amounts to about 2 milliseconds at 44 kHz sample rate, depending on audio buffer size.

Sub-Controls

Use these modules to build controls and GUI features. An entire chapter of this book is devoted to these sophisticated tools.

Synths

This folder contains two drum modules, a subtractive synth, phase distortion synth, and an FM synth example.

VST Plug-ins

SynthEdit lists all VST/i plug-ins found at the location you specified in Edit > Preferences > File Locations > VST Plugins. Subfolders may be created in that directory, with their contents listed as submenus.

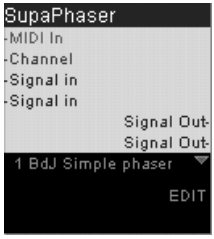


Figure 1.20: A VST effect

Though SynthEdit lets you load and embed VST/i plug-ins in your structure, the GUI does not appear if you embed it in your VST. If the embedded VST/i plug-in is MIDI-enabled, use MIDI controller data to manipulate it. Check the plug-in's documentation to learn which controllers you may use.

Heads up: Before releasing a plug-in with an embedded VST/i, check if the license permits you to do this, or contact the author to get permission.

Waveform

This menu lists sound-generating modules. There are three types of oscillators.

The Oscillator module generates band-limited sine, saw, ramp, triangle, and pulse waveforms offering oodles of modulation options. It also serves as a low-frequency oscillator, and generates white and pink noise. White noise's frequency spectrum is flat, whereas pink noise's frequency spectrum drops by 3 dB per octave and yields smoother highs.

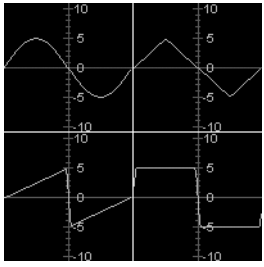


Figure 1.21: Sine, triangle, saw, and pulse waves

Phase Dist Osc distorts a sine wave's phase to create complex waveforms. Introduced by Casio with the CZ synth line in 1984, phase distortion is much like FM synthesis.

The Soundfont Oscillator loads and plays selected banks and patches from Soundfont (.sf2) files. It renders sample data only, without envelope, filter, and velocity settings.

The ADSR module creates amplitude, filter, pitch, and user envelopes. A, D, S, and R stand for attack, decay, sustain and release, respectively.

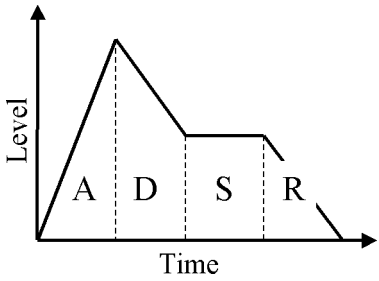


Figure 1.22: ADSR envelope

Attack: Controls how fast the signal reaches peak level when the player presses a key.

Decay: Controls how fast the signal falls from peak to sustain level.

Sustain: Controls the level at which the signal remains until the player releases the key.

Release: Controls how fast the level returns to 0 when the player releases the key.

Third-party Modules

You can create what in synth-speak is called a third-party module using the SynthEdit SDK (Software Development Kit) and, say, a C++ compiler. Whatever your heart desires—oscillators, filters, effects, logic operators, and many more—someone somewhere offers a third-party module to fit the bill. The accompanying data on www.wizoobooks.com/synthedit offers a rich selection of these. Again, SynthEdit automatically lists all folders and modules found in the "modules" directory, so you can create more folders and copy third-party modules. See the web on www.wizoobooks.com/synthedit for details.

2

Designing VST Effects in SynthEdit

Meet the Family of VST Effects

This chapter examines the different types of VST effects, reviewing the ins and outs of creating all common signal processors. Beginning with the basics, it describes how to assemble plug-ins featuring different parameters and application possibilities. Many tips and much inspiration for fine-tuning plugs await, so let's get to it.

Kicking Off a VST Effect Project

A general rule for both instruments and effects: A main container in the Structure window must hold an effect before you can save it as a VST plug-in. Every container comprises one or more IO Mod modules that pipe signal and data flows in and out. The number of channels hinges on the number of inputs and outputs. For example, a main container with one input and one output creates a mono-to-mono VST plug-in. A container with one input and two outputs is a mono-to-stereo effect. A container with two inputs and two outputs is a stereo-to-stereo effect. The registered version of SynthEdit gives you more inputs and outputs to juggle, letting you configure stereo side-chains (say, two times two inputs), surround plug-ins (5 + 1 channels), multiple outputs, and so forth, the VST host allowing.

Cooking Up a Simple Filter Plug-in

Create and open a container in the Structure window. Insert a Moog Filter (Insert > Filters) and connect the IO module's Spare pin to the filter's Signal pin. You could connect the filter's Output pin to the IO Mod's Spare pin. But for an uncluttered setup, your better bet is to create another IO Mod (Insert > IO Mod) for the outputs and drag it to the window's right edge. Now slap some controls on the filter (Insert > Controls > Slider). Conveniently, the slider takes on the input pin's name as you connect the two. Your Structure window should look like something like this:

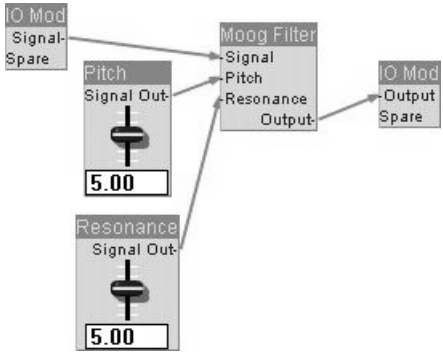


Figure 2.1: The structure of a simple filter (filter.se1)

Right-click the Structure window and select Panel Edit to open the interface, which will be appear on the plug-in. The window's size determines GUI's size. Drag and move the sliders to the window's top left corner, and adjust window size so it holds the sliders as seen in figure 2.2.

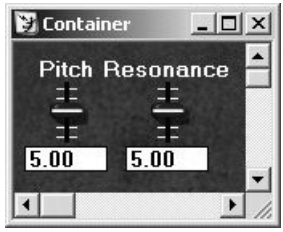


Figure 2.2: A simple filter's panel window

Now you can save the plug-in. Open the File menu and select Save As VST. Enter a name, say, Moog Filter. This header appears in the VST host. Assign a unique, four-character ID to your plug-in so the host can identify the plug-in. Click OK to save the VST effect. It should be ready and waiting in your host. In Cubase SX, it looks like this:

Figure 2.3: The filter's GUI in a host

Heads up:

- You may need to restart the host program or rescan the VST plug-ins folder for your VST to appear.
- You can register your plug-in with the ID at the following official Steinberg website: http://service.steinberg.de/databases/plug-in.nsf/plug-in

Go-to files: Effects\Our first effect\filter.se1

Double Up for Stereo

Our first module is a mono-to-mono plug-in. For a stereo version, clone this setup to both the left and right channels. To this end, add another Moog Filter, and connect the input IO Mod's Spare pin to the filter's Signal pin, and the filter's Output pin to the output IO Mod's Spare pin. Recall that you may connect all inputs and outputs to the same IO Mod, but adding another gives you tidier setup. New input and output plugs appear on the container as you connect new pins to the IO Mod. The sequence in which you connect plugs fixes their order. Now connect the sliders to the second filter's Pitch and Resonance pins. Presto—your stereo version of the plug-in should be good to go.

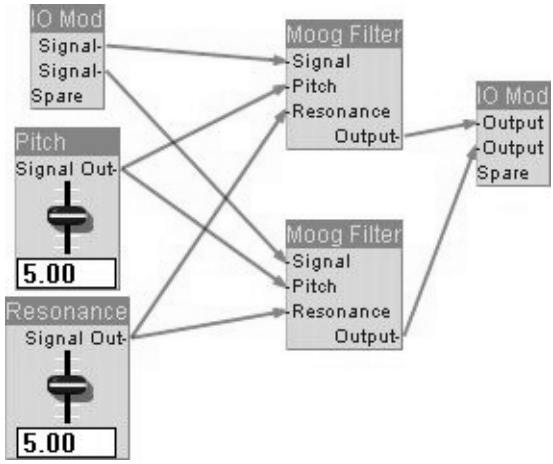


Figure 2.4: A stereo version of the filter (filter_stereo.se1)

Heads up:

- You may find building a more complex stereo version easier if you put both the left and right channel setups into a container.
- To test your plug-in in the SynthEdit environment, plug an audio source into the input and wire the outputs to a Sound Out module or a Freq Analyser as shown in figure 2.5. Saving a plug-in as a VST/i stores only the main container's contents; the test signal and Sound Out modules are not part of the plug-in.

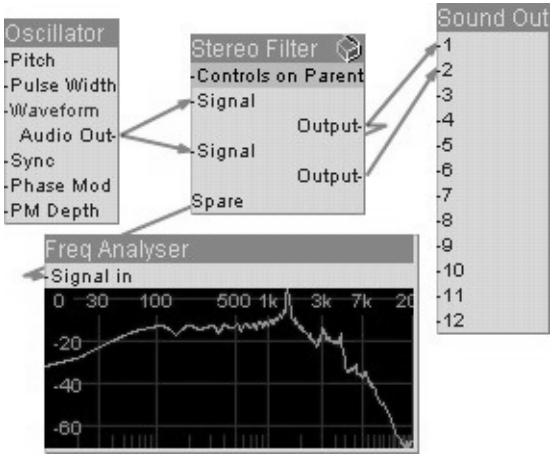


Figure 2.5: Testing the stereo filter in an SE environment

Fun with Auto Filters

Most plug-ins offer modulation sources, say, low frequency oscillators (LFO) and envelope followers, to tweak audio parameters. This section explains how to add these to the filter using a third-party module, Dave Haupt's DH_MultiFilter2. Download and copy DH_MultiFilter2.sem to any "modules" subfolder in your SynthEdit folder, naming it modules\Filters\, modules\3rdparty\Filters, or something similarly imaginative. The module appears in the Insert menu's folder selection. Create a container and add two copies of DH_MultiFilter2 to its structure. Now configure a setup similar to the stereo filter above. Connect a List Entry (Insert > Controls > List) to the filter's Filter Type selector plug. You can connect the same list entry to both plugs. Your structure should look like figure 2.6.

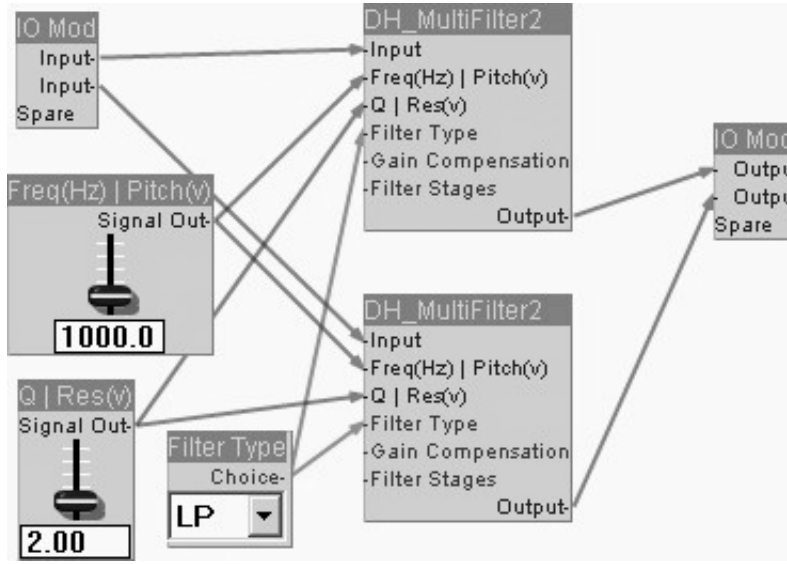


Figure 2.6: The filter's basic structure (autofilter1.se1)

You must tweak some parameters before you can use MultiFilters. Open the filter's Properties window and adjust the following parameters:

Set Gain Compensation to On. This normalizes the output signal and prevents clipping by setting the filter's resonance peak to 0 dB.

Set Filter Stages to 1. This feature cascades several filters to create steeper slopes, a service we can do without for the moment.

Turn your attention to the Input Mode. Some native and third-party plug-ins feature variable input ranges or scales for controlling pitch. Select Pitch/Res to set the Freq(Hz) | Pitch(v) plug to 1 Volt/Octave mode. This ensures the Q | Res(v) plug controls resonance in a way that serves our purposes. Be sure to adjust both filters.

Heads up: At times you may need several copies of a module sharing the same parameter settings. If so, configure the parameters of one module, and copy it as often as you like using Edit menu copy and paste commands or key shortcuts. The application also copies your settings, sparing you the tedium of mindless repetition.

Time to adjust the Slider modules: Open the Pitch Slider's Properties window. Select Cutoff to rename it. Note the slider's default low and high settings have changed from 0 and 10 volts to 10 and 20,000 volts. That's high voltage, so set it back to 0 and 10 volts, as the input scale is 1 volt per octave with 5 volts equaling 440 Hz. Rename the resonance plug Reso and set the Lo and Hi values to 0 and 10 volts, respectively. If you prefer a knob's look and feel to a slider's, simply select Knob from the list. Now when you patch in a mono test signal, you should get a mono output signal.

Installing Dry/Wet and Gain Knobs

The balance of dry and wet signals usually calls for fine adjustment. That's what an X-Mix module does (Insert > Modifiers > X-Mix). Add two copies, one for each channel. Connect the main input to the cross-faders' Input B plugs, and the filters' outputs to the Input A plugs. Create a new knob or copy and paste an existing knob, and connect its output to the cross-fader's Mix plug to dial in a smooth transition. Only the dry (unfiltered) signal passes at –5 volts, while only the wet (filtered) signal passes at 5 volts. The two signals blend at values between the two extremes, creating a phaser-like effect at high resonance settings.

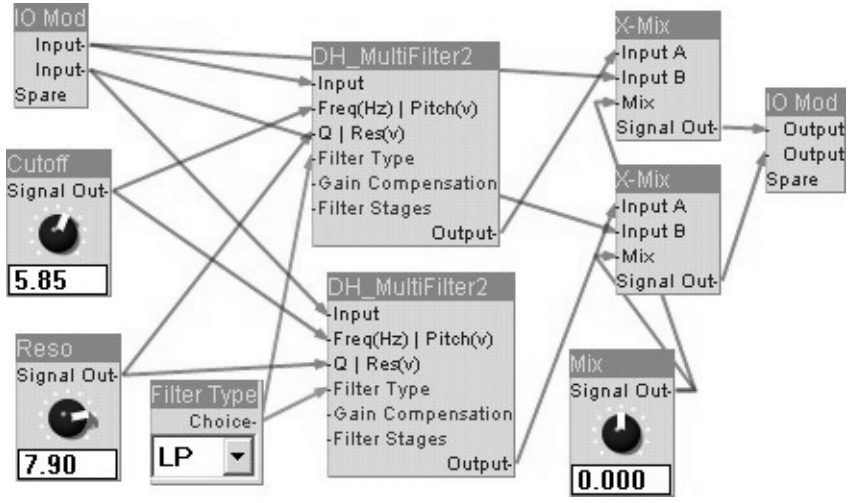


Figure 2.7: Installing a dry/wet knob (autofilter2.se1)

You need a VCA (voltage controlled amplifier) module (Insert > Modifiers > VCA) to adjust levels, one for each channel. Open the VCA's Properties window and set Response Curve to Decibel to control the input voltage level in decibel increments. For details on VCA response curves, see the section on converting signal levels in the SynthEdit manual.

Heads up: Third-party converter modules offer precise decibel values for adjusting levels. Some examples in this chapter use DH_dBTo-Voltage to this end.

Go-to files:

- Effects\Filters\autofilter2.se1
- Effects\Filters\autofilter3.se1

Follow Up with an Envelope Follower

Filters often feature strange beasts called envelope followers. Soundsculptors use envelope followers to detect a signal's contour. They rectify and filter the incoming signal to extract the envelope curve, which provide the control voltage for compressors or auto-wah effects.

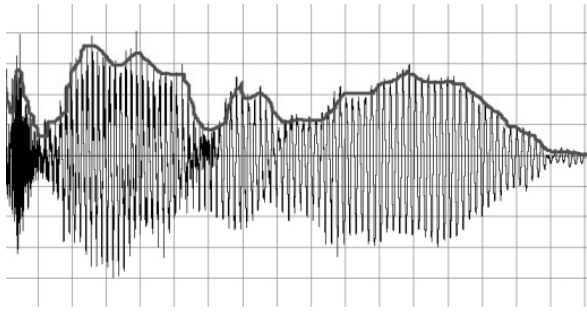


Figure 2.8: A wave file in an editor with its envelope marked

SynthEdit offers you other choices for extracting a signal's envelope. Perhaps the most straightforward is to use a Peak Follower module (Insert > Modifiers > Peak Follower). It has a Signal-in plug, and Attack and Decay parameters to control the smoothing amount. These values decide how quickly the envelope follower responds to sudden peaks. One volt equals 20 milliseconds. A Level Adj module lets you handily control the effect amount. Place it after the Peak Follower; then feed its output signal into the filter's Pitch plug. It adds up incoming signals, so the envelope modulates the filters' cutoff frequency.

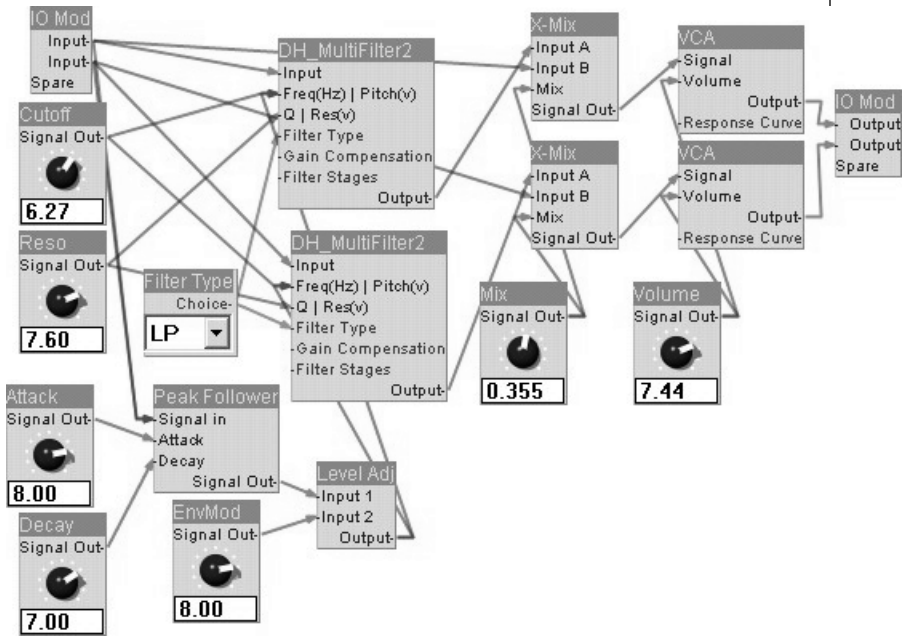


Figure 2.9: Inserting an envelope follower (autofilter4.se1)

Heads up:

- Both input plugs connect to the Peak Follower's Signal-in plug, converting them to mono and taking the average of the two signals. You may tap the left and right channels' envelope individually.
- To test the effect in the SynthEdit environment, you may find it necessary to load a loop or other wave file using a Wave Player (Insert > Input/Output > Wave Player).
- Fastest attack and decay values may over-excite the modulation effect, causing it to spin too fast, the sound to warble, and aliasing noise in the filter. Enter one volt as the low attack and decay value to fix that problem.
- Equal to 600 milliseconds, 30 volts is fine for higher attack and decay limits.
- Setting the range of the EnvMod knob to –10 Volts ... 10 Volts enables negative modulation. Negative voltages invert the envelope.

Go-to files: Effects\Filters\autofilter4.se1

Go Low by Adding an LFO

An LFO is a low-frequency oscillator that modulates a given parameter value at a slow rate. Though similar to an oscillator that produces audible waveforms, its frequency usually lies in the subsonic range between 0.01 Hz and 30 Hz. The most common waveforms are sine and triangle as shown in figure 2.10.

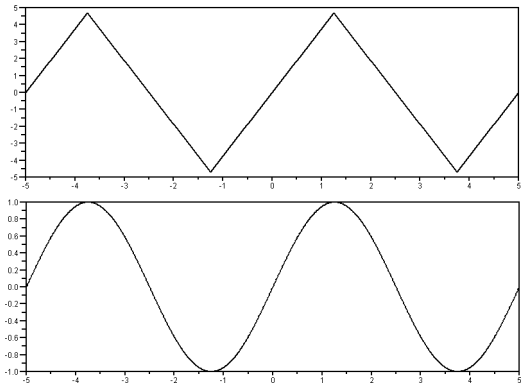


Figure 2.10: Commonly used LFO waveforms: sine and triangle

Use an oscillator module (Insert > Waveform > Oscillator) to create an LFO in SynthEdit. Open an oscillator's Properties window and you will see two modes for setting the frequency, 1 Volt/Octave and 1 Volt/kHz. In 1 Volt/Octave mode, 5 volts equals 440 Hz, and every 1-volt change doubles or halves the frequency. The conversion formula is:

$$\text{Frequency} = 440 * 2^{\text{Volts}-5} = 13.75 * 2^{\text{Volts}}$$

Therefore, the default 0-to-10 volt range extends from 13.75 to 14080 Hz. The formula for converting frequency to voltage is:

$$\text{Voltage} = \frac{\log(\text{Freq}) - \log(440)}{\log(2)} + 5 = \frac{\log(\text{Freq}) - \log(13.75)}{\log(2)}$$

Reach for your trusty calculator and confirm that 30 Hz equals 1.1255 V, and 0.01 Hz equals –10.4252 V. Connect a knob to an oscillator's Pitch plug, and enter these as high and low values. This affords you exponential control over the LFO rate within the range of 0.01 Hz and 30 Hz. Name this knob Rate.

An oscillator's default output range is +5 to –5 volts. Drop a Level Adj module in after the oscillator's Audio Out plug so you can adjust the LFO's depth. This example uses a triangle waveform selected in the Properties window. So far, the LFO structure looks much like figure 2.11. Connect the Level Adj module's Output plug to the filters' Pitch plug. The LFO then modulates the filters' cutoff frequencies as shown in the autofilter5.se1 prefab.

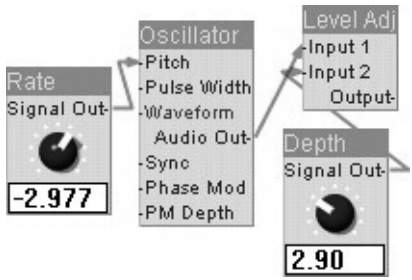


Figure 11: A simple LFO

Go-to files: Effects\Filters\autofilter5.se1

Super-size the Signal with a Stereo LFO

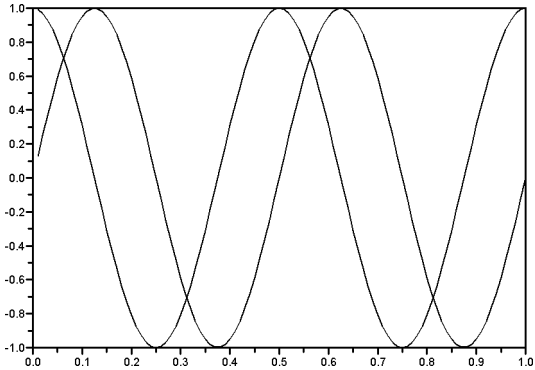


Figure 2.12: Two sine waves with different phase

To conjure sumo-sized stereo effects, build an LFO for both channels and shift the two waveforms' phases. To do this, you need another oscillator with a Level Adj module, using the same knob to control the

respective rate and depth. The oscillators' Phase Mod plug controls the waveform's phase. Change this value for one oscillator. Setting Phase Mod to 10 volts puts the two waveforms in opposite phase, so 0 to 10 volts is a good choice of range. Simply connect a knob to this plug and name it Stereo. Figure 2.13 outlines the resulting structure. Now connect a Level Adj's output to plug a filter's Pitch plug, and the other Level Adj's output to the other filter's Pitch plug.

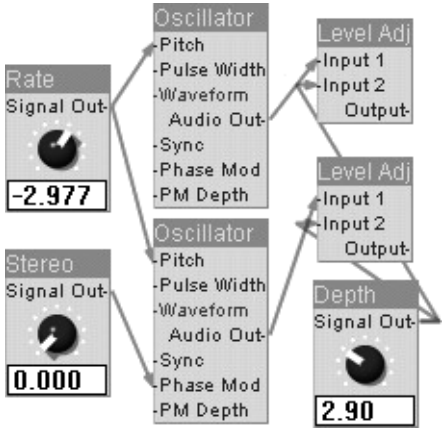


Figure 2.13: This is what a stereo LFO looks like.

Go-to files: Effects\Filters\autofilter6.se1

Adding a Tempo Sync LFO

Many plug-ins let you synchronize the LFO to the host's tempo. Though a bit tricky, this is not exactly rocket science. The Tempo Sync LFO prefab in the Insert\Examples folder makes this chore even easier. Check out figure 2.14 to reference the structure.

The tempo sync LFO bases on the BPM Clock2 module in the Insert > Special folder. Its main purpose is to detect the host's tempo and issue a pulse to kick off downbeats. The Tempo Out plug provides tempo in beats per minutes (BPM). One minute comprises 60 seconds, so dividing the tempo in BPM by 60 gives you the number of beats per second, which equals frequency in Hz. The oscillator is in 1 Volt/kHz mode, so multiply the dividend by 1000 to get a kHz result. Now you know why

we divide 60,000 to obtain the frequency. Your Division choice divides or multiplies 60,000 by a number provided by the Fixed Values (Insert > Controls > Fixed Values) modules. The result is the frequency for the given time period.

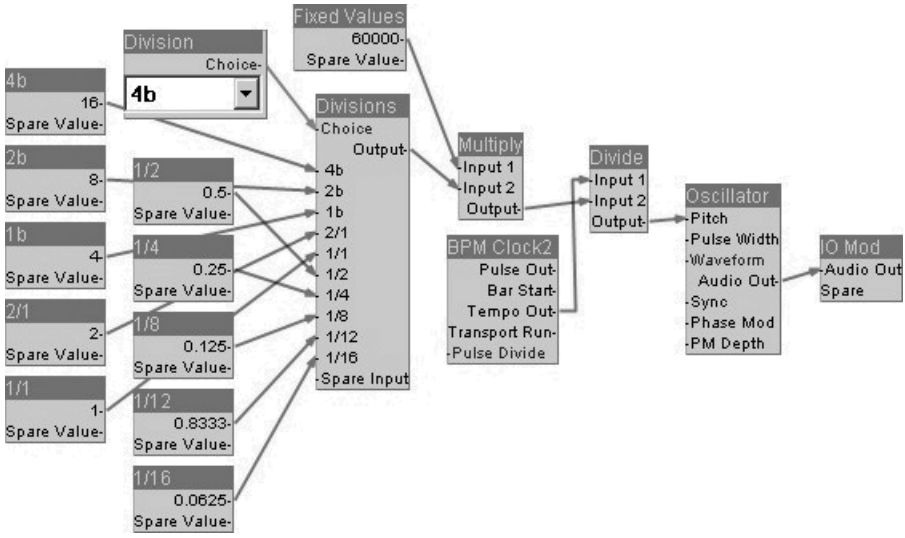


Figure 2.14: Complex but not confusing—a tempo sync LFO's structure

To whip up a stereo version of this LFO, insert another oscillator with the same pitch and an adjustable phase. Connect the Spare plug to an oscillator's Phase Mod plug to create an input plug for the container. Right-click the IO Mod module and select Properties to rename the plug. Label lets you change the input and output plugs' names. Be sure to select a triangle for the oscillators' waveform and set their Freq Scale to 1 Volt/kHz. If you wish to hide the Spare plug, lock it by clicking the Lock icon in the toolbar or right-clicking the container and ticking Locked.

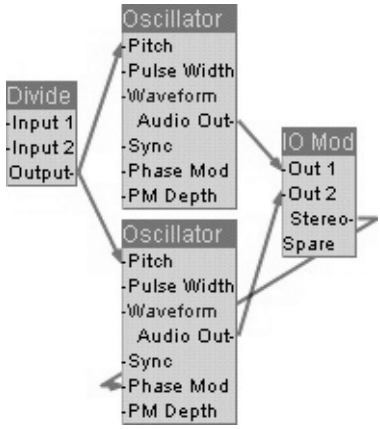


Figure 2.15: Going stereo with twin oscillators

Use a switch (Many → 1) to toggle conveniently between a free-running oscillator and a tempo sync LFO. Connect the oscillator's Audio Out and the tempo sync LFO's output to the switch's spare plug; ditto for the other channel. Be sure to do this in the same order in each setup, configuring the free-running oscillator first and the tempo sync LFO second, or vice versa. Connect a List Entry module to both switches' Choice plug. Open the first switch's Properties window to change the list entries. Editing labels also changes the list entries. If the List Entry module connects to several List inputs, the first connected switch defines the labels in the list.

Heads up:

- A List Entry module comes in many guises, including Combo Box, Led Stack, Labeled Led Stack, Selector, Button Stack, Rotary Switch and Up/Down Selector. Figure 2.16 gives you a glimpse of the default visuals. Bear in mind that you can skin all these selectors.

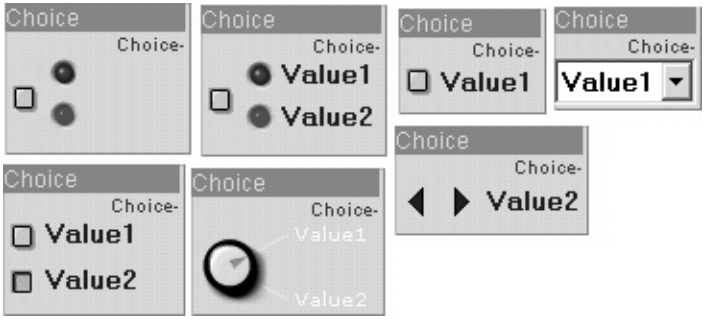


Figure 2.16: A List Entry module's default visuals look like this

❖ Figure 2.17 maps out the final auto-filter structure with a switch sitting between the free-running oscillator and the tempo sync LFO.

Go-to files: Effects\Filters\autofilter7.se1

Finalizing the Filter

Once you have finished configuring the plug-in's structure, it is time to design the user interface. Open the Panel window and arrange knobs in a practical array. The Panel Group serves no other purpose but to help you lay out GUI features (Insert > Controls > Panel Group). To rename (right-click and select Properties) and resize features, simply drag them to the bottom right corner.

Tips:

❖ Select several features by dragging the mouse or clicking modules while holding the control key down.

❖ Disable Edit > Snap to Grid to position features more precisely.

❖ Pressing Ctrl-A selects all features.

Readout boxes' information is not always relevant. To hide these boxes, open the slider or knob's Properties window and disable Show Readout. Congratulations are in order if your panel resembles figure 2.18.

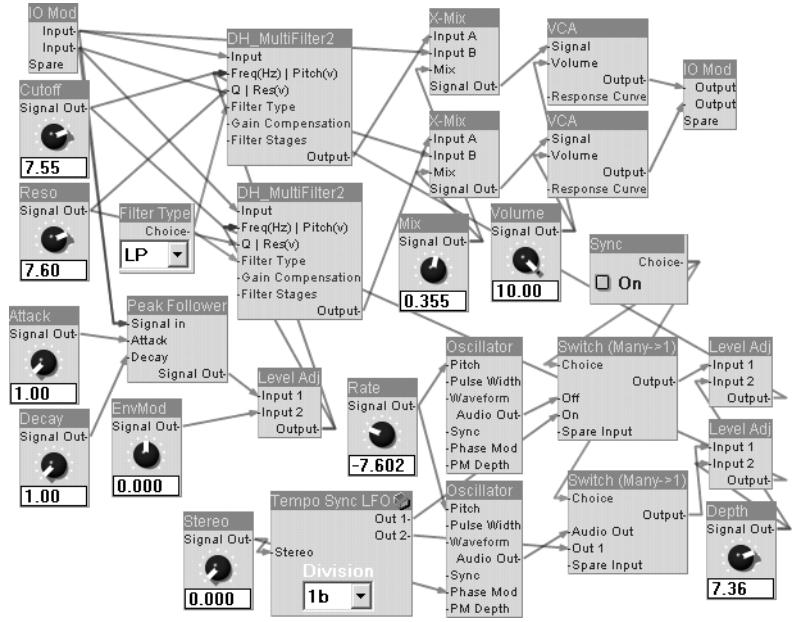


Figure 2.17: The whole kit and caboodle—an auto-filter with an LFO and envelope follower

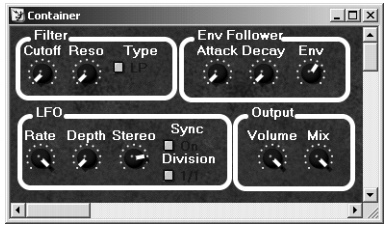


Figure 2.18: An auto-filter panel with a default skin

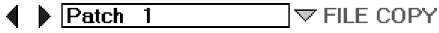
If this look and feel fails to float your boat, use a skin on www.wizoo-books.com/synthedit or design one to taste. To apply a skin, you must load its files to a subfolder of the SynthEdit\skins folder. Named VK_Mini-Grey, the skin in figure 2.19 came courtesy of Vera Kinter.

Select a skin in one of two ways. Open the Main container's Properties window and select the skin from the list at the bottom, or right-click the panel window and select the skin from the Skin menu.

Figure 2.19: An auto-filter with a groovy custom skin (VK_Mini-Grey)

Go-to files:
Effects\Filters\autofilter7gui.se1
Effects\Filters\autofilter7gui2.se1

Adding Patches and Presets



One more chore awaits before saving this filter as a VST plug-in. In order to enable patches, you must insert a patch selector. Drop a Patch Select (Insert > MIDI > Patch Select) module anywhere into the structure. It automatically handles patches and presets. The Panel window opens with a Patch Select bar at the top. Serving solely to create patches, it will not appear in the final plug-in. Browse the patch bank, configure different patches, and give them meaningful names. Once you have created plenty of presets for your plug-in, click File > Save as VST.

Heads up: 16 patches is the limit for plug-ins saved using the unregistered version of SynthEdit.

Go-to files: Effects\Filters\autofilter8.se1

Delay Effects

Basic delay effects load the input signal to a buffer and render it after a brief interval. Most employ a feedback loop, cycling the output signal back to the input. The signal repeats infinitely as its amplitude gradually decreases. Some sonic scientists insert a filter to the feedback loop to simulate high- and low-frequency damping. Others use more delay lines (taps) to conjure complex delays. And all this merits our further investigation.

Devising a Simple Delay

SynthEdit features a module devoted to creating delay effects. Called Delay2 (Insert > Effects > Delay2), it is a delay line of variable length with built-in feedback. Figure 2.20 depicts its Properties window.

Delay Time (secs) sets maximum delay time in seconds. 10 seconds is the buffer limit for delay. The Modulation plug determines the actual delay time contingent on the Delay Time parameter. That is, if Delay Time is 1.0 seconds, 10 volts modulation yields a one-second delay, 5 volts 500 ms delay, and so forth. For modulated delay lines such as flanger and chorus effects, you should enable the Interpolate Output to achieve smoother transitions between delay times.

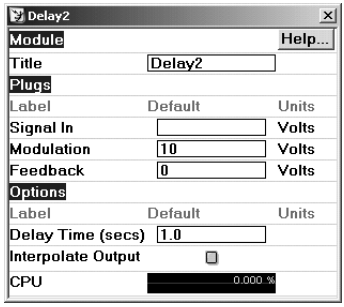


Figure 2.20: A look at a Delay2 module's Properties window

Figure 2.21 maps the simplest delay structure. Each input channel feeds a delay line, with a VCA adjusting output levels. The VCA's Response Curve plug is in Decibel mode, so the delayed signal's level adjusts on a decibel scale. The wet processed signal mixes with the dry

input signal to produce the composite output signal. The Length slider controls delay time, while the Feedback slider adjusts feedback amount. The channels share the same settings, so this is a mono delay imposed on a stereo signal.

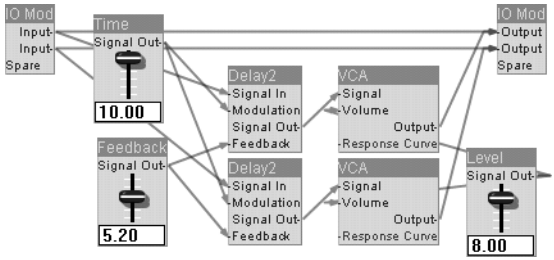


Figure 2.21: A streamlined delay structure (delay1.se1)

Max delay time is one second. To display and enter delay time in milliseconds, set the slider's high value to 1,000 and divide the slider's output by 100, or use a Divide module (Insert > Math > Divide) to do this. Connect the slider's output to the Divide module's Input 1. This is the numerator. Open the Divide module's Properties window and set Input 2 to 100. This divides the slider's output by 100 for an output range of 0 to 10 volts, suitable for the Modulation plug. Add another slider with a Divide module and you can control the left and right channels' delay times independently. See figure 2.22.

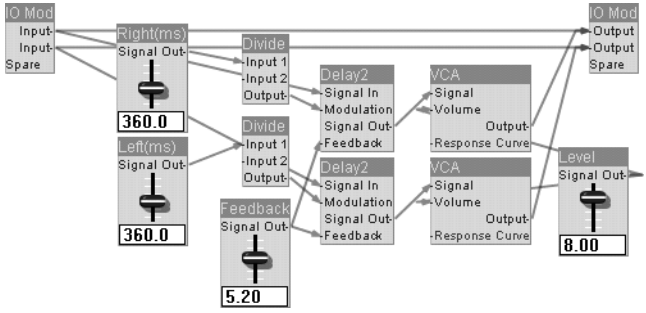


Figure 2.22: A stereo delay metered in milliseconds (delay2.se1)

Heads up: The module's Properties window lists every plug's default. If no wires are connected to the plug, SynthEdit assumes the default value as input, sparing you the effort of wiring up the Divide modules' Input 2 plugs. The principle at work here is the same as connecting a Fixed Values module (Insert > Controls > Fixed Values). Though this merely provides fixed voltages to the selected plugs, the values appear in the plug's label in the Properties window.

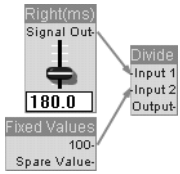


Figure 2.23: Fixed values

These prefabs use a common Feedback control for both channels. An independent Feedback control for each channel is often more practical.

The Delay module in the Insert > Obsolete folder uses a –5 to +5 voltage range for the Modulation plug. Use Delay2 instead.

Go-to files:
Effects\Delay\delay1.se1
Effects\Delay\delay2.se1

Adding Dry/Wet Controls

You may use plug-ins as insert or as send effects. Figure 2.24 is a schematic diagram of inserts and sends. Delay, reverb, flanger, chorus, and other effects comprising a mix of dry and wet signals can usually serve as sends. A send effect patches only the wet signal out. The host mixes it with the dry signal, producing the output. Send effects are great for processing any number of channels without burdening the CPU. If you want to enable a plug-in as a send effect, you must find a way to mute the dry signal, usually by adding dry/wet and gain controls or separate gain controls for dry and wet signals. The following prefabs use separate dry and wet gain controls.

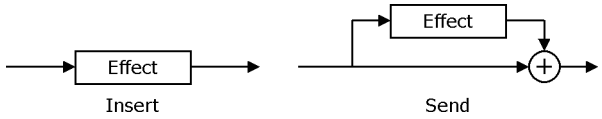


Figure 2.24: A schematic view of inserts and sends

Slapping a Filter on the Wet Signal

Delay plug-ins often use filters to shape the delayed signal. Types vary, though usually low-pass and high-pass filters simulate high and low frequency damping. The delay3.se1 example uses a—adjective alert—two-pole, resonant, low-pass, state variable filter in the wet signal chain to dampen high frequencies (Insert > Filters > SV Filter). Two VCAs adjust dry signal levels.

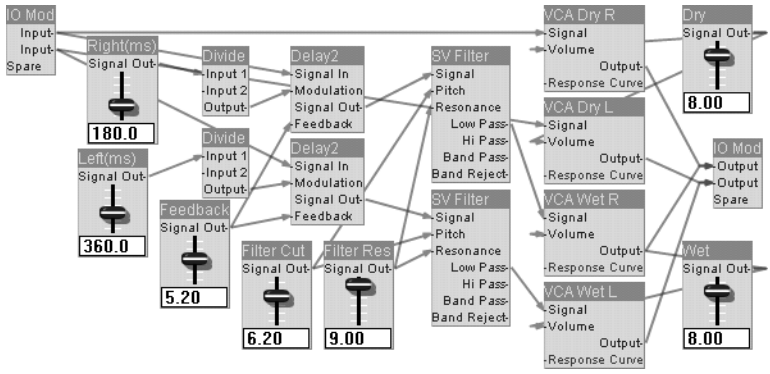


Figure 2.25: A low-pass filter in the wet chain dampens high frequencies (delay3.se1).

Heads up:

- "State-variable" means a two-pole filter with a 12 dB/octave slope. The application calculates low-pass, high-pass, band-pass, and band-reject filtering simultaneously, so you are free to choose any output signal.
- When resonance approaches 10 volts, the SV filter begins to self-oscillate, which is uncool in a delay filter. Confine the resonance slider's range to 0 to 9 or 9.5 volts to nip this problem in the bud.
- Savvy designers set the filter cutoff plug's low value to about 3 volts, or 110 Hz, the likelihood of lower frequencies seeing use is slim.

Roughly equivalent to 22,000 Hz, 10.6439 volts is an excellent choice of high value. Anything beyond that only the user's dog will hear.

- To simulate low frequency damping, insert another SV filter after the low-pass filter, connecting its Hi Pass plug to the VCA module.

Synchronizing Delay Time to Tempo

Users often wish to synchronize delay to song tempos, so do them a favor and afford them the opportunity to enter delay length in beats rather than milliseconds. Time to don your math cap: The BPM Clock2 (Insert > Special > BPM Clock2) module provides the tempo in volts. Set the Delay2 modules' max delay time to 10 seconds so one volt of modulation means one second of delay. One minute comprises 60 seconds, so divide 60 by the tempo in BPM to get the voltage required for one beat. Take, for instance, 120 BPM: 60/120 = 0.5, so a beat is 500 milliseconds long. Now what if users wish to synchronize delay to halves or quarters rather than whole beats? Give them that alternative by dividing the beat's length by 2, 4, and so on to arrive at 250, 125, or another increment in milliseconds. Figure 2.26 outlines the structure for determining unit length.

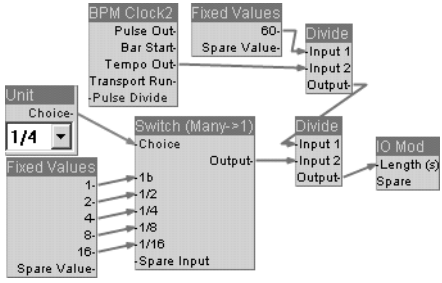


Figure 2.26: Determining unit length in seconds

Connect the Fixed Values module's Spare plug (Insert > Controls > Fixed Values) to the Switch module's Spare plug (Insert > Flow Control > Switch (Many → 1)), thereby creating selection options. The labels defined in the switch's Properties window determine which labels the list box will display. This structure resides in its own container. To show the Unit list box on the container and the GUI, open the Container's Properties window and enable Controls on Parent and Controls on Module. If necessary, open the unit container's panel window and resize the Unit list box.

Heads up:

- Every container comes with a panel holding graphical features. You can edit and move them around as you would the main GUI's features. Tick the Controls on Module box to show them on the main structure's container. Tick the Controls on Parent box to show them on the main GUI. This option serves chiefly to create custom control modules. See the section "What Are Sub-controls?" from page 193 onwards for details.

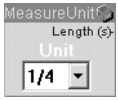


Figure 2.27: Controls on module

- When using this structure, be sure to set the Delay2 modules' Delay Time parameter to 10 so one volt of modulation equals one second of delay.

Once you have chosen the unit and calculated its length, you need a control feature to set delay length in that unit of measurement. This length is always the unit length multiplied by a whole number, so you need a control feature that puts out whole voltages, say a switch with fixed whole number values. You could also opt for a slider with an output value rounded to whole numbers, as in this example.

A slider or knob produces a floating point number within the given low and high limits. Two modules round off a signal voltage: Ceil (Insert > Math > Ceil) rounds the input up to the nearest whole volt (2.5 yields 3 volts; –2.5 yields –2 volts.) Floor (Insert > Math > Floor) rounds the input down to the nearest whole volt (2.5 yields 2;–2.5 yields –2). Set a slider's lower and upper limits to 1 and 8 volts, and the Floor module will round the output signal down to whole numbers between 1 and 8. This is fine for selecting delay length in a scale based on beats.

The problem is the slider's readout shows the floating point value rather than the rounded number. The solution is to use sub-controls. Switch the slider or knob's readout box off in the Properties window by disabling Show Readout. Instead, task a Text Entry2 sub-control module (Insert > Sub-Controls > Text Entry2) to display the value. The input plug's blue background tells you this is a GUI Text plug, so you must convert the signal plug into a GUI Text plug.

The Volts to Float (Insert > Conversion > Volts to Float) module converts voltage into a float plug, which a Patch Mem–Float Out (Insert > Sub-Controls > Patch Mem–Float Out) module can then transform into a GUI float value. In this case, its only purpose is to convert voltage. Then convert the GUI float value to GUI text using the Text To Float (Insert > Sub-Controls > Text To Float) module. Bear in mind that GUI module plugs' flow may be bidirectional. Here the float value enters the module on the right, and the text value exits to the left. Open the Text To Float module's Properties window to specify how many digits you wish to see displayed. Set it to 0 to show only the whole number without decimals. Figure 2.28 affords you a view of this structure.

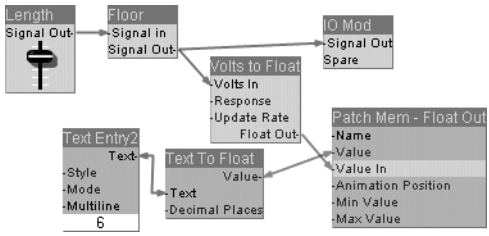


Figure 2.28: Showing readouts using sub-controls

Now you can dump this structure into a container to create a control prefab for selecting length. Open the container's panel window and drag the Text Entry box under the slider. Enable the Controls on Parent and Controls on Module options for the container. Rename the slider to Length. Lock the container if no further changes are necessary. For more on sub-controls, refer to the section "What Are Sub-controls?" from page 193 onwards.

Figure 2.29

Heads up: Embedding a structure couldn't be easier: Press the keyboard's Control key or drag the mouse to select the target modules, and then select the Containerise Selection command from the Edit menu.

All that remains for you to do now is select a Multiply (Insert > Math > Multiply) module that multiplies the unit length by the slider or knob length, and patch its output to the Delay2 module' Modulation plug. Multiplication is commutative, so the order of wires is irrelevant. Figure 2.30 diagrams the resulting structure.

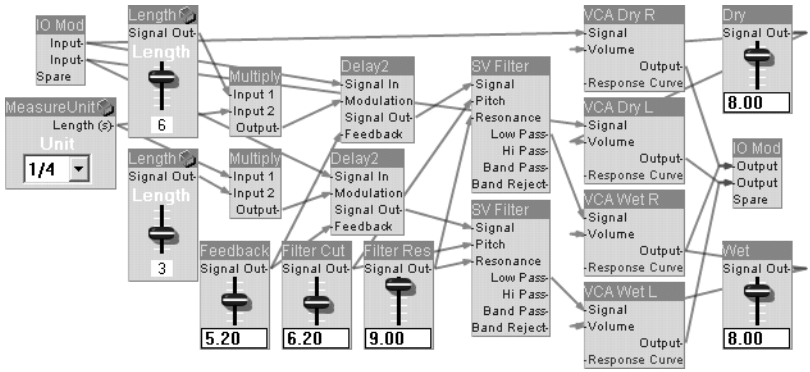


Figure 2.30: Structure of a tempo sync delay (delay4.se1)

Go-to files: Effects\Delay\delay4.se1

Serve and Volley with Cross Delays (Ping-pong Delays)

A cross delay is much like a feedback delay, except that its signal feeds back to the other delay line's input, bouncing the signal between the left and right channels. Hence the term ping-pong delay. Figure 2.31 is a schematic diagram of a cross delay.

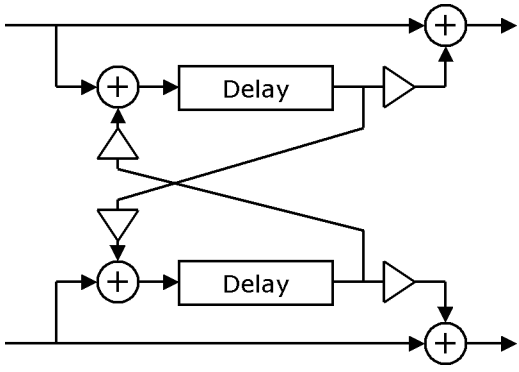


Figure 2.31: Schematic diagram of a stereo cross delay

We must use an external feedback path rather than the Delay2 module's internal feedback circuit. However, if you try to create this structure with a feedback path, you will get the error message:

"This patch contains a FEEDBACK path, Please remove."

The reason for this is SynthEdit's internal structure, in which modules process audio in buffers rather than by samples. You can hurdle this obstacle using a Feedback module (Insert > Special > Feedback (delayed)) to create feedback paths. There is a minor catch, though. Feedback is not instantaneous; the module introduces a touch of latency. Its amount hinges on buffer size, usually around 90 to 100 samples. This comes to about 2 milliseconds at 44 kHz, and one millisecond at 96 kHz sampling rate. Though the effect is negligible in some applications, timing may suffer in others.

Heads up: Say a patch comprises two chains, one with, the other without feedback. Latency may elicit flanging, leaving undesirable artifacts in the signal. Insert a Feedback (delayed) module to the other chain to compensate. On the upside, the two chains are now in sync sans flanging. On the downside, you end up with about 2 milliseconds global latency. Latency compensation prevents flanging in this patch.

Figure 2.32 outlines the basic cross-feedback delay structure. It bases on a tempo sync patch, meaning delay length selectors are synchronized to tempo. The top two Feedback (delayed) modules live in the dry signal path, delaying the signal to compensate for the feedback loop's latency. The bottom two feedback modules' outputs connect to the Delay2 modules' Signal in plugs. Level Adj modules adjust their output levels, thus determining feedback level. The scaled signal feeds back into the other channel's Feedback module, creating the cross-delay effect. VCA modules scale the Delay2 modules' output signals and determine the wet (delayed) signal level.

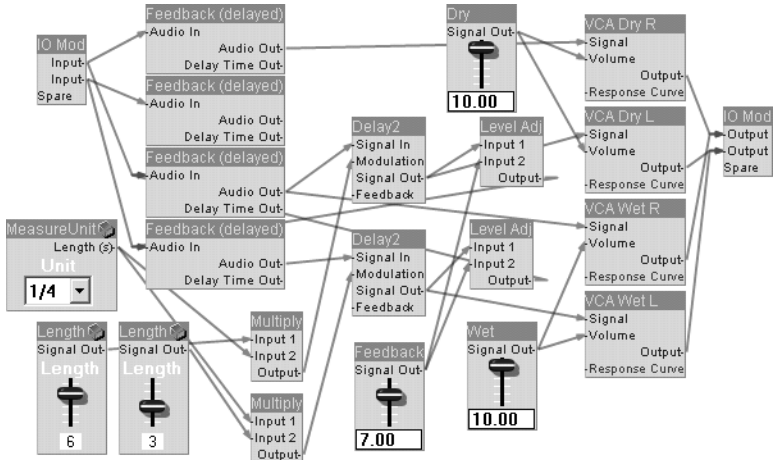


Figure 2.32: A basic cross-feedback delay structure (crossdelay1.se1)

Heads up:

- Using the Delay2 modules' internal feedback is a no-go, so be sure to set their Feedback plugs to 0.
- Simply add a Switch (1 → Many) to transform this cross delay structure into a conventional feedback delay. Choose between sending the signal back to the same or the other channel to let you toggle between conventional and cross feedback. You could also add another Level Adj module that feeds back to the same channel. This lets you adjust normal and cross feedback amounts independently.
- The signal recycles infinitely at 100 % (10 volts) feedback level. Levels above 100 % (> 10 V) continually ramp up the volume, eventually distorting the signal and "exploding" the effect. Set feedback to 100 % or lower, unless an endless loop is what you want.
- You can set feedback levels in decibels using a DH_dBToVoltage module.

Go-to files: Delay\crossdelay1.se1

Lining the Feedback Path with Filters

In the patches we have worked with so far, the Delay2 module's limitations prevented us from inserting filters in every feedback loop. Instead, we filtered the global wet output signal. But adding filters to the feedback path puts far more sounding-shaping power at users' fingertips. It enables realistic simulations of damping, where the effect

cumulates with every feedback loop. Resonance and gain pile up quick in resonant, EQ, and shelving filters with positive gain, soon blowing the effect up. To prevent this, use filters without resonance or with gain normalization (DH_MultiFilter2), or allow negative gain in shelving filters only. The following example employs cascaded one-pole low-pass and high-pass filters with a smooth 6 dB/octave slope for high and low frequency damping. Place filters pre Delay2 effects so the filtering effect accumulates. Figure 2.33 illustrates the structure with the filter.

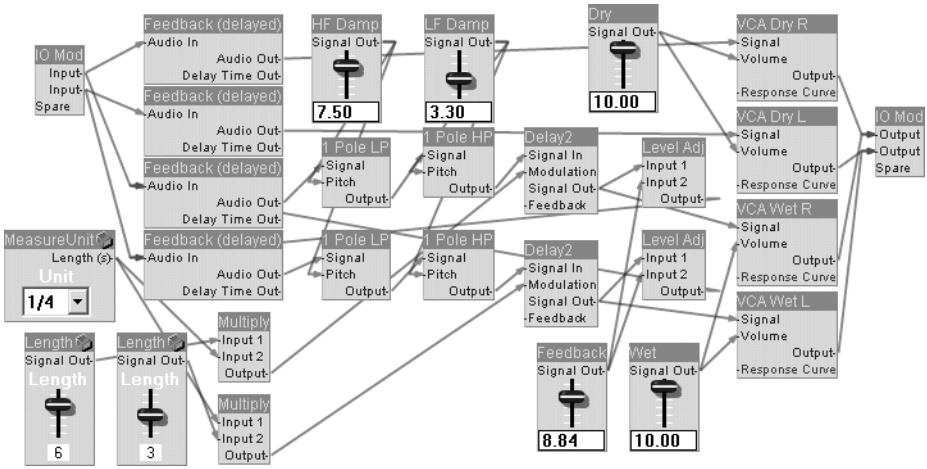


Figure 2.33: A cross-delay with filters lining the feedback path

Heads up:

- Filters in the feedback path are nifty, but many settings cut the signal's amplitude. Feel free to use feedback levels a touch over 100 % for experimental effects. This lets you create endless ambient textures, but mandates careful adjustment of the feedback level. A peak limiter comes in handy for limiting high feedback levels.
- Drop Pan modules in front of the amplifiers to stereo pan the two channels and create leaner effects.
- Place any filter or effect in the feedback chain, say, flangers and phasers, to conjure striking soundscapes. Again, proceed with caution when using level-boosting prefabs and adding resonance or feedback, as output levels may soon spike.

Caution:
Before experimenting with structures containing external feedback paths and potentially high feedback levels, back off speakers and headphones' volume to protect your equipment and hearing.

Go-to files: Delay\crossdelay2.se1

Doing the Multi-tap Dance with Delays

Many delays feature several delay lines called taps for greater flexibility. Adding Delay2 modules does this trick neatly. To keep things reasonably simple, this example converts the incoming signal to mono, adding the two channels and multiplying by 0.5 to get the average. This composite signal feeds into four delay lines. A low-pass filter follows each delay to dampen high frequencies. A dedicated Pan module pans mono signals (Insert > Modifiers > Pan). And a value range of –5 to +5 volts determines panning positions. Like a Level Adj module, the Volume plug adjusts levels.

A third-party conversion module called DH_dBToVoltage adjusts levels in decibel increments. Built by David Haupt, it comes with his BasicModulePack. DH_dBToVoltage converts precise decibel values to voltage for use with linear level adjustment modules, for example Level Adj or the Pan module's Volume plug. With the Vref plug set to the default 10 volts, the value is in the standard 0 to 10 volts scale. The default range of the dB in plug is –100 to 0 volts, so if you connect a slider or knob to it, its low and high values adapt automatically. A decibel range of –40 to 0 will do for a tap volume.

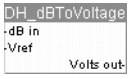


Figure 2.34: DH_dBToVoltage

Now adjust the dry signal's level, and mix it with the four taps' output signals. The delay lines' delay time is the one second default, so the sliders' high value is 1,000. Divide each slider's output signal by 100 to enable millisecond delay length settings. All taps sport a dedicated feedback control. Figure 2.35 shows one tap's structure with the dry signal.

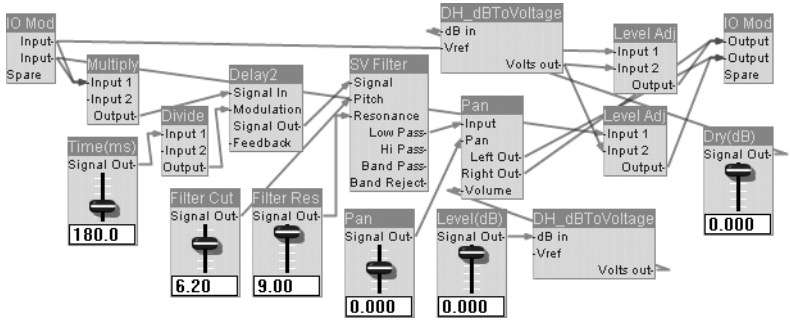


Figure 2.35: Structure of one tap with the dry signal (multitap.se1)

Drop the tap into a container to tidy up. Hold the Control key to mark the Divide, Delay2, SV Filter, Pan, and DH_dBToVoltage modules for the tap, and then select Containerise Selection from the Edit menu. Change the new container's input labels. The dBToVoltage and Level Adj modules for the dry signal are also containerized for your convenience. Figure 2.36 maps this structure.

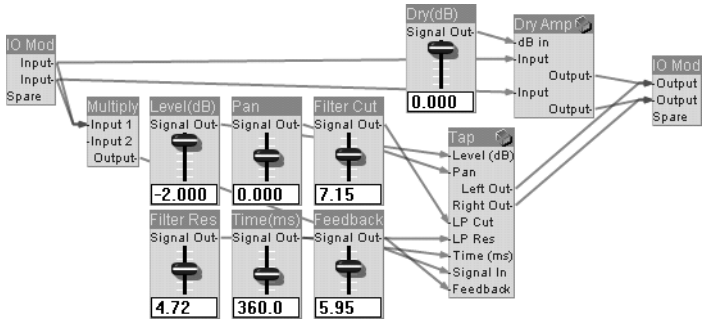


Figure 2.36: Containerized Tap and Level Adj modules (multitap2.se1)

To add more taps, simply copy the Tap container with its controls. Connect the averaged input to the tap's Signal in plug, and connect its outputs to the IO Mod module. Figure 2.37 shows the entire structure.

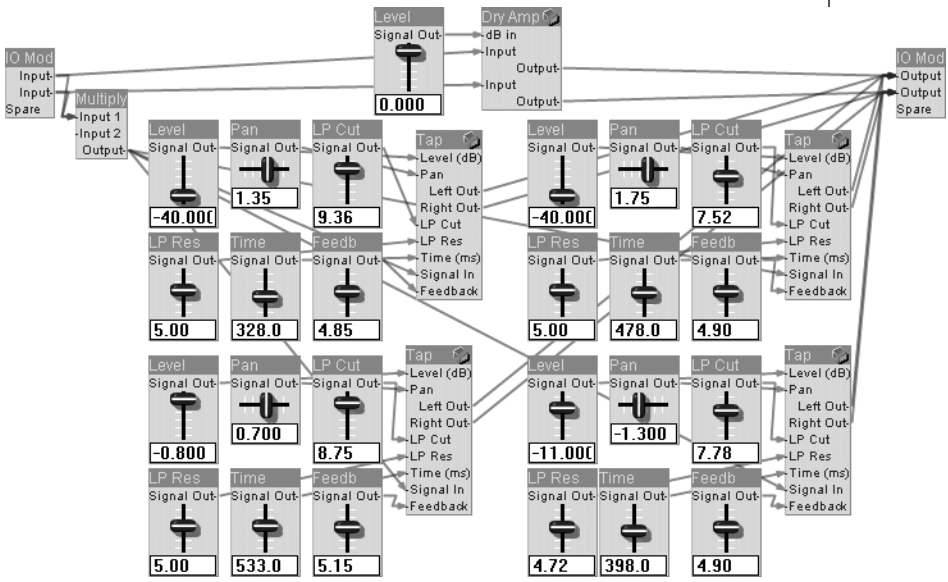


Figure 2.37: A delay structure with four taps (multitap3.se1)

Go-to files:
Delay\multitap1.se1
Delay\multitap2.se1
Delay\multitap3.se1

Finalizing the Multi-tap Delay

The effect is nearing completion; only the GUI awaits your attention. Open the panel window and behold a jumbled mess at its center. Panels housing so many control features call for some tidying tricks. Grab a control feature and drag it away from the rest. Leave it selected and go to the Structure window, where it is also selected. Select all control features in that group by holding the Control key or dragging the mouse. Now you can move them collectively in the panel window by dragging the isolated feature. This method is also useful for identifying which control feature is selected on the panel.

Here's another approach: First lay out one tap's control features in the panel window; then copy and paste them in the Structure window. Select the Tap container to do this. Copying also clones control features' panel position, so you can move them collectively.

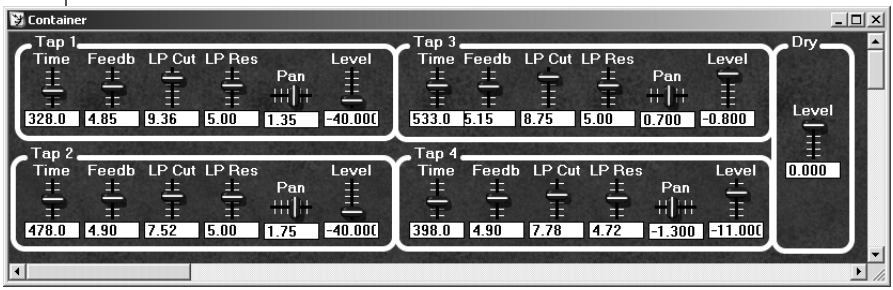


Figure 2.38: A possible layout for the four-tap delay using the default skin

Give 'Em Some Room with Reverb

Perhaps you noticed that when you tweak the four-tap delay's parameters, certain settings evoke diffuse delay patterns reminiscent of reverb. Though primitive, early digital algorithms relied on similar structures in combination with all-pass filters to simulate diffuse reverbs. So let's briefly recap how reverb works.

Every room or reverberant space reflects sound differently. An impulse is a sudden burst of sound, like an electrical arc, a sharp hand clap, a popping balloon, or a gunshot. Figure 2.39 charts a typical room's response to such an impulse.

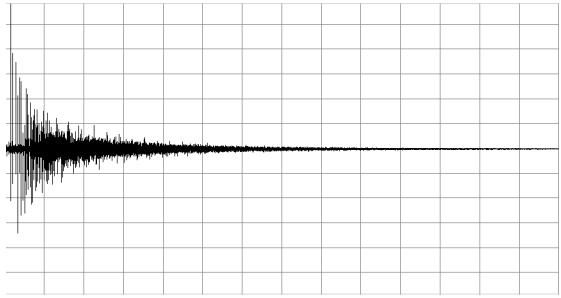


Figure 2.39: A reverb's impulse response

The immediate response is distinct echoes bouncing off walls called early reflections (ER). The pattern gradually grows more diffuse as amplitude decays exponentially. Compare this to the impulse response of a comb filter, in essence a delay line with feedback. Figure 2.40 plots a comb filter's impulse.

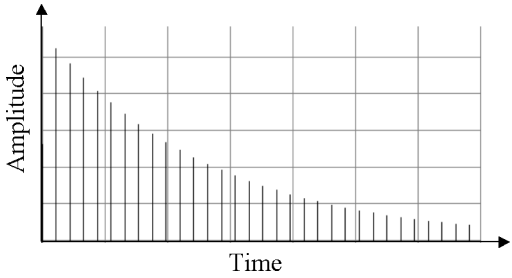


Figure 2.40: A comb filter's impulse response

Exponentially decaying impulses follow the first impulse. Though this is similar to an exponentially decaying reverb, in the frequency response peaks occur at equally spaced frequencies like the teeth of a comb, hence the name comb filter. This elicits a ringing metallic sound. Digital pioneer Manfred Schroeder proposed using parallel comb filters in combination with all-pass filters for digital reverb. Figure 2.41 is a schematic diagram of an all-pass filter.

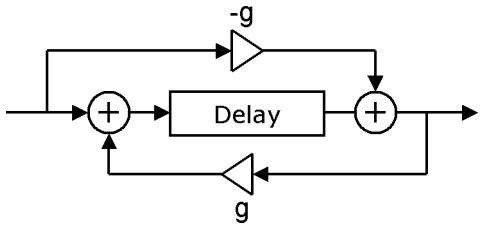


Figure 2.41: Schematic diagram of an all-pass filter

An all-pass filter comprises a feedback and a feed-forward path, yielding flat frequency response. Nonetheless, it serves to shape transients and simulate diffusion courtesy of its decaying impulse response. Figure 2.42 graphs an all-pass filter's impulse response.

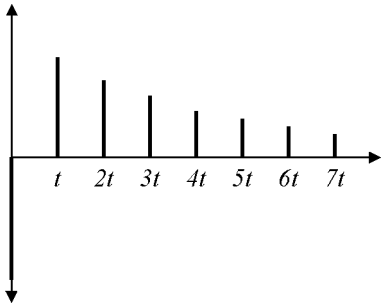


Figure 2.42: An all-pass filter's impulse response

The Schroeder Model

Schroeder proposed a structure with four parallel comb filters and two serial all-pass filters. The comb filters simulate the reflections, and the all-pass filters add density to the sound by smearing transients and making the reverb more diffuse.

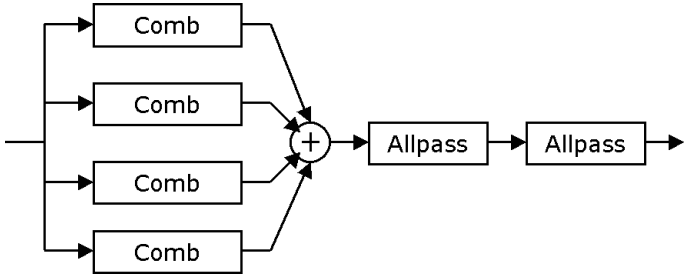


Figure 2.43: A Schroeder reverberator

Below you see the structure of a Schroeder reverberator in SynthEdit. The input signal feeds into a container holding four parallel delay lines wired to two serially cascaded all-pass filters.

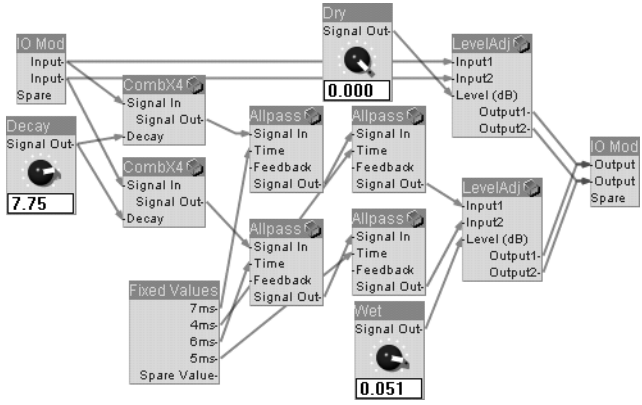


Figure 2.44: The structure of a Schroeder reverberator in SynthEdit (schroeder1.se1)

Figure 2.45 outlines the four comb filters. In this structure, Delay2 modules serve as comb filters, with delay times of 52, 63, 79, and 83 ms. Though seemingly random, textbook reverb design calls for these values to be mutually prime numbers. The comb filters' length determines the simulated room's size. Moorer suggests using linearly distributed values over a ratio of 1 to 1.5. Different delay values color reverb in different ways, so fine-tune the delay times manually. Changing the length of just one comb filter can have a tremendous impact on the reverb's overall sound. The Multiply module divides the result by the number of comb filters so the reverb and dry signal level are equal.

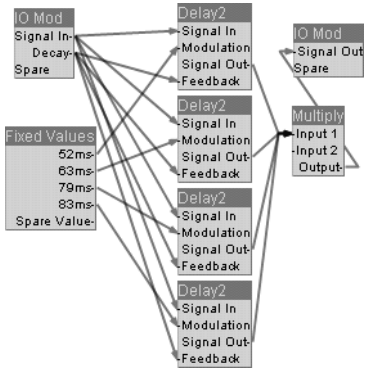


Figure 2.45: The structure of the CombX4 container

The reverb time is defined as the time for the level to drop from the initial level to –60 dB. Schroeder's equation for calculating a comb filter's time goes like this:

$$T = \frac{60}{-20 \log |g|} \qquad t = \frac{-3}{\log |g|} t$$

g denotes feedback level, and t delay length. Here's how to calculate the feedback level for a given delay time and total reverb time:

$$g = \pm e^{3t/T}$$

Take, for example, 52, 63, 79, and 83 ms and one second total reverb length. This equation yields ±0.8555, ±0.8277, ±0.7889 and ±0.7795 (±8.555, ±8.277, ±7.889 and ±7.795 volts) for the feedback levels. The structure above is simplified, with the same feedback level for all delay times. Though this makes it easier to control reverb time, on the downside, comb filters with longer delays are slower to fade. Jezar Freeverb employs this method, with satisfying results. Setting feedback to a value of one (decay to 10 volts) elicits infinite reverb.

Figure 2.46 shows the all-pass filter structure. It is the SynthEdit equivalent of the all-pass scheme in figure 2.38. A Feedback (delayed) module feeds a delay module's output back to its input. It adds the inverted and scaled input signal to the Delay2 module's output signal, lending it all-pass characteristics. The Inverter module changes the signal's charge. The all-pass filters smear the transients in the reverb, creating a diffuse sound. The longer the delays, the more diffuse the reverb. True enough, but Moorer recommends some 6 ms for the all-pass' delay time because longer delays produce audible reiterations. The left and right channels delay times' differ, yielding a stereo effect. If the input signal is mono, or the input's stereo width is irrelevant, you can convert the two channels to mono by averaging. Then only one CombX4 structure feeding both left and right all-pass filters yield the same results for mono signals, with the benefit of a lighter CPU load.

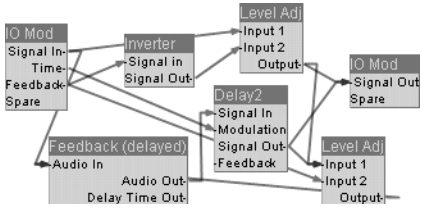


Figure 2.46: An all-pass filter in all its glory

Heads up: Conserve CPU resources by using sc:RevAllpass modules. They consume only about third of the processing power devoured by this all-pass structures.

Go-to files: GoReverb\schroeder.se1

The Moorer Model

You can across James Moorer's name earlier in the book. He improved Schroeder's model, which suffers the drawback of a metallic sound. Moorer suggested using low-pass filters in the comb filters' feedback path to simulate high frequency damping. He also simulated early reflections using tapped delay lines. Moorer's model employs six parallel comb filters and just one all-pass filter. Figure 2.47 outlines its schematic.

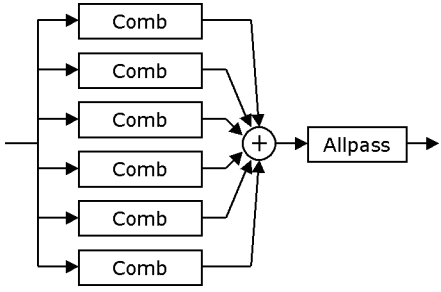


Figure 2.47: The Moorer model

Each comb filter's feedback loop sports a one-pole low-pass filter as shown in figure 2.48.

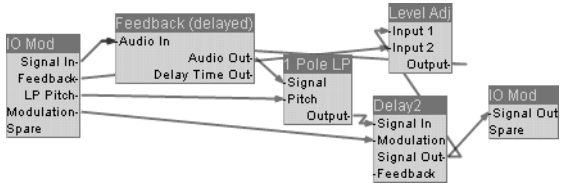


Figure 2.48: A comb filter with a low-pass filter in the feedback loop

This structure is similar to our cross delay. Here the one-pole filters are set to 1 Volt/kHz mode. Called Delay3 in the prefab, a CombX6 container holds six of these in parallel array. Figure 2.49 depicts the main structure. Divide the Damping knob's output by 100 to show the damping frequency in Hz.

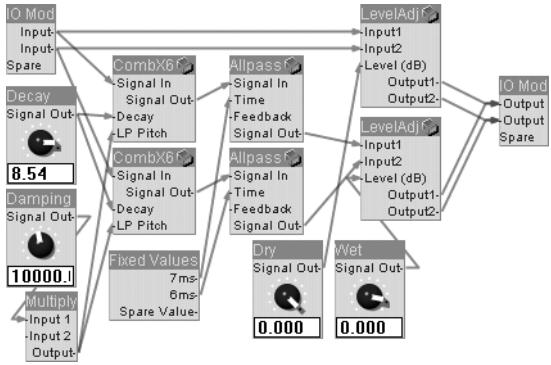


Figure 2.49: Moorer's remarkable reverberator (moorer1.se1)

This prefab employs delay times proposed by Moorer—50, 53, 61, 68, 72, and 78 ms. To simulate different room sizes, scale these values using Level Adj modules. Check out moorer2.se1, in which the Size parameter scales all delay times. To avoid conjuring a ringing metallic sound, limit the Size parameter to at least one volt.

Go-to files:
Delay\moorer1.se1
Delay\moorer2.se1

Good-to-Know Facts about Reverb

Though many reverb plug-ins employ the Schroeder/Moorer model, ours is of the simple sort. It simulates neither early reflections nor escalating density, so do this to improve it:

- Use individual delay taps with varying amplitude to simulate early reflections; then feed them into the comb/all-pass structure.
- Use reverb algorithms that mix the left and right input channel, adding about 25 % of each channel to the other. This emulates naturally occurring blending.
- Moorer holds that more than six comb filters or more than one all-pass filter do not improve audio quality markedly. Some reverbs beg to differ. A case in point is the oft-reused Jezar Freeverb code sporting eight parallel comb filters and four serial all-pass filters per channel.
- Throw in a high-pass filter to simulate low-frequency damping.
- Apply equalization to shape the reverb's sound.
- If you hang with having less control over the sound, try using third-party reverb modules based on the Jezar Freeverb code such as DH_Reverb and UD-Reverb, or EVM Rev-8 and EVM Rev-12 modules.

Modulated Delay Effects (Flanger, Chorus)

This section is aptly named because the primary building blocks of modulated delay effects are delay lines, or comb filters, if you prefer, as used in the previous sections. This section pairs them various modulations, usually generated by low-frequency oscillators. They add depth and stereo breadth to a signal, or lend it a distinct flavor. Flanger and chorus are the two main categories of modulated delay effects. The big difference between the two is that a flanger thrives on short delay times, usually less than 20 ms, while a chorus usually sweeps across a wider range. Read on to learn more ...

Conjuring a Flanger

Mixing two signals, one with several milliseconds of variable delay, creates a flanging effect. The two signals are out of phase, notching the frequency spectrum at linearly spaced intervals. As delay time changes, the notches sweep across the frequency spectrum, conjuring a comb fil-

ter effect. For more emphasis, the output is often fed back to the input to give rise to sharp resonant peaks in the frequency spectrum. Figure 2.50 is a spectrogram image of a flanger's effect on white noise. The light stripes are notches; the dark stripes resonant peaks.

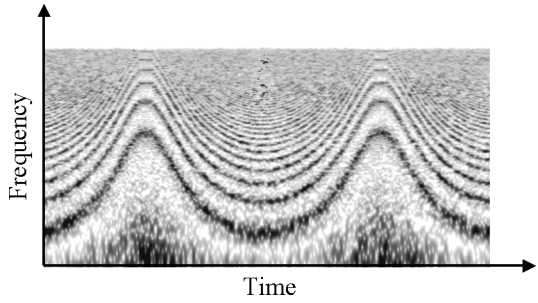


Figure 2.50: A spectrogram of a flanger applied to white noise

The term flanger was coined in the mid 20th century, when two tape machines played the same signal at the same time. An engineer placed a finger on a tape reel's flange, slowing one tape and throwing the two out of sync. When the engineer released the reel, its speed gradually returned to normal, evincing a groovy psychedelic whoosh.

Figure 2.51 shows the most rudimentary stereo flanger structure. Each input channel feeds into a delay line, which then mixes with the dry signal and notches the spectrum.

The delay lines' delay time is 0.01, or 10 milliseconds. An oscillator module puts out –5 to +5 volts by default. This setup adds 5 volts to arrive at 0 to 10 volts to match the input range of the Delay2 module's Modulation plug. The Level Adj module scales the output signal, thereby determining modulation depth. The Modulation plug's value changes constantly, so be sure to enable the delays' Interpolate Output option. This prevents the dreaded zipper noise and ensures a smoother output signal. This prefab employs a triangle waveform for the LFO.

Tweak one oscillator's Phase Mod plug to go stereo. Shifting the two waveforms' phases creates a wide-body stereo effect. 5 volts translate to 90 degrees, 10 to 180 degrees phase shift.

The oscillator's pitch determines the modulation rate. In 1 Volt/Octave mode, the voltage equivalent of 0.01 Hz is –10.4252 V, and 10 Hz is –0.4594. So, the Rate plug's lower and upper limits stake out a range of 0.01 to 10 Hz. Feedback slider's range is –10 to +10 volts to accommodate negative feedback levels.

Heads up: For more frequency-to-voltage conversion rules, see the sections "Follow Up with an Envelope Follower" from page 38 onwards and "Go Low by Adding an LFO" from page 40 onwards in this book, or the section on voltage conversion in SynthEdit's Help.

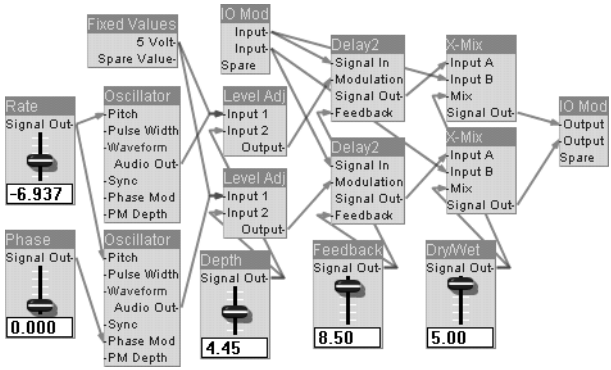


Figure 2.51: As simple stereo flanger (flanger1.se1)

Go-to files: flanger1.se1

Adding a Waveform Selector

Sine and triangle waves are the two most commonly used LFO waveforms. Saw and ramp waves also see some use, but pulse and noise waveforms are rarely employed for LFOs. you may narrow down oscillator waveform options. The Insert > Examples > Limiting List Choices prefab is an example of how to limit an oscillator's waveform selector options using sub-control modules. Figure 2.52 maps the structure.

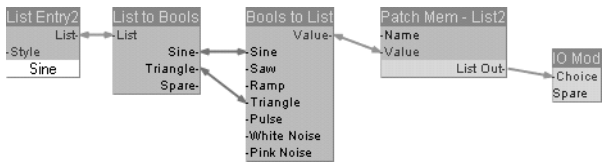


Figure 2.52: A prefab for limiting list choices

The Patch Mem–List2 module converts the list input into a GUI list value. The list is then converted to Boolean values (recall GUI plugs' bidirectional data flow). The List to Bools module converts the Boolean values back to a list plug, but addresses only options linked to it. If desired, rename options by opening the Properties window of the List to Bools module and changing the labels. The GUI list output connects to a Dropdown List module, which is the actual GUI control feature. We renamed the module in the example prefab List Entry2.

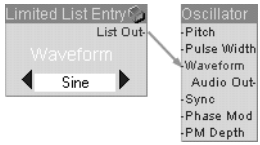


Figure 2.53

To add a left/right arrow similar to those found in the Controls > List Entry2 module, simply copy the Bitmap Image, Float to Bool and Increment2 modules from that prefab, and connect the Increment2 module to the List to Bools module's List plug. Once you have configured the container's panel, the arrows let you quickly change options. We connected a Text Entry2 module to the Name plug of the Patch Mem–List2 to show a label for the list entry. If you wish, open the Properties window and select Read-Only mode to make the label readable only. The Style option—as determined by the current skin's global.txt settings—changes the text's appearance. Figure 2.54 illustrates the structure. The flanger2.se1 prefab uses this waveform selector.

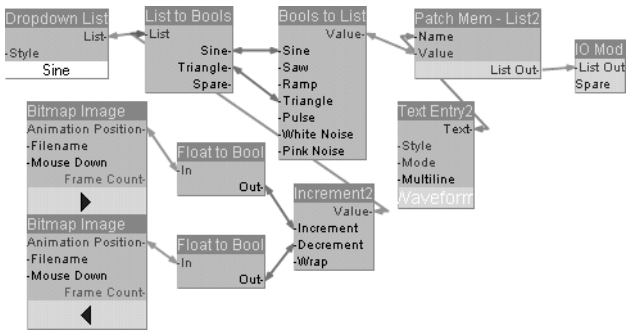


Figure 2.54: Limiting list choices with label and arrows

Go-to files: flanger2.se1

Making Modulation More Variable

The lowest modulation value in the previous prefabs is 0 volts. Some users may hanker for different modulation ranges. Fulfill their wish by adding a constant voltage to delays' Modulation plug. Proceed with caution, for the two voltages' sum may not exceed the Modulation plug's 10-volt threshold. So, if you choose 5 volts as the Depth and Min sliders' upper limit and double the Delay2 modules' max delay time, pulling the Min slider down to 0 yields the same results. But the Modulation plug's value will not exceed 10 volts even with the Min slider pulled all the way up, ensuring the plug-in remains stable at all settings.

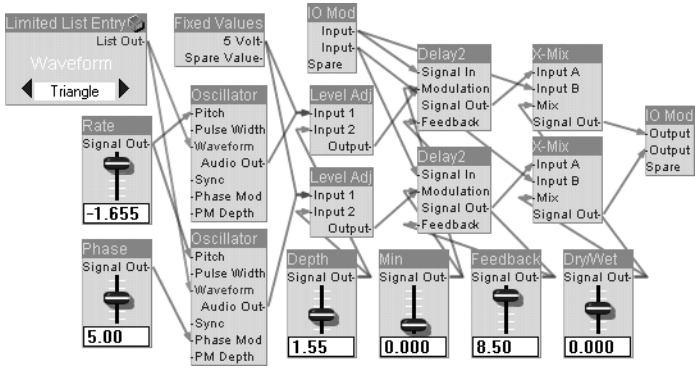


Figure 2.55: Variable minimum delay time (flanger3.se1)

Go-to files: flanger3.se1

More About Flangers

Flanger modules may produce a loud low frequency hum at high feedback levels. Squeeze in a high-pass filter after the delay lines to fix this problem. If the effect's tone is too trenchant, tame a delayed signal's top end with a low-pass filter.

Switch the phase of the the delayed signal using an Inverter module to conjure different sounds.

Tweaking the GUI

So far, we used mostly standard Slider modules to control parameters. The following example looks at how to fine-tune knobs and sliders' readouts using sub-control modules. Figure 2.56 shows the outcome.

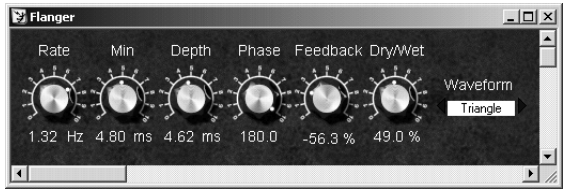


Figure 2.56: A sexier GUI with a more sophisticated readout (flanger4.se1)

The knobs base on the Knob prefab (Controls > Knob) depicted in figure 2.54. The heart of the prefab is the Patch Mem–Float module. The Patch Mem module's Min Value and Max Value plugs define the knobs' low and high limits. The Animation Position plug connects to Bitmap Image modules providing the knob view. The actual value goes to two plugs—a GUI Float Value plug on the left, and a Float Value Out plug on the right. The prefab employs a float Value Out plug to send the signal to a Float to Volts (Insert > Conversion > Float to Volts) module, which converts the value to voltage. The Text Entry2 module merely brandishes the label.

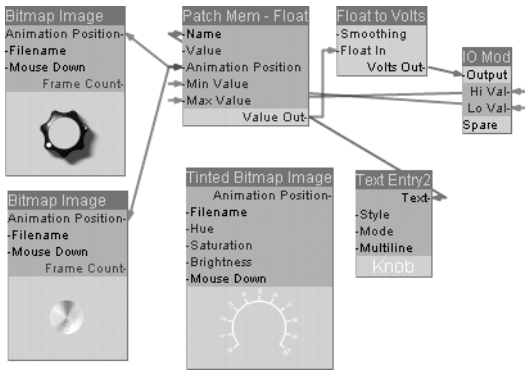


Figure 2.57: A knob sporting sub-controls

Use the Patch Mem's GUI float Value plug to show readouts. Figure 2.59 illustrates the Depth knob's structure. This setup first scales the Value plug; then converts it to text for display. The knob's range is 0 to 5 volts, or 0 to 10 milliseconds. Use a Float Scaler (Insert > Sub-Controls > Float Scaler) module to scale the knob. The rule for scaling is:

Value Out= Value In ∗ Multiply by + Add

This module is bidirectional, with the inverse function being:

$$\text{Value In} = \frac{\text{Value Out} - \text{Add}}{\text{Multiply by}}$$

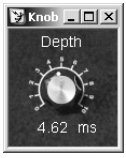


Figure 2.58

We are using the inverse option. To multiply the Value Out by 2, go to the Float Scaler's Properties window and enter the inverse of 2, 0.5, for the Multiply by plug. The Value In plug's range is thus 0 to 10. A Text To Float module converts these values to text displayed by a Text Entry2 module. Don't forget the decimal places. Another Text Entry2 module shows the unit of measure. You need a Patch Mem–Text module to store the Text plug's value, which is now ms. Then arrange labels and readout on the container's panel as depicted in figure 2.58. If necessary, disable Edit > Snap to Grid to enable more precise positioning.

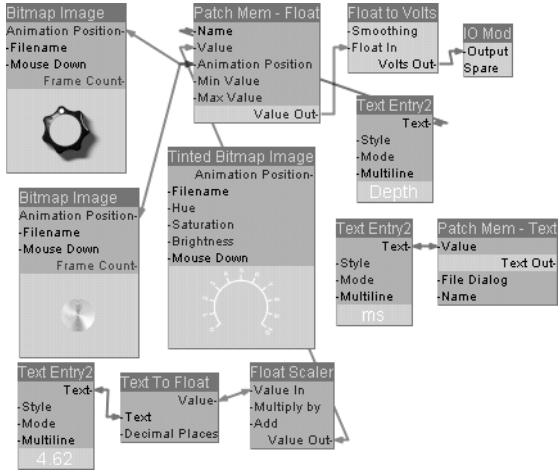


Figure 2.59: The Depth knob's structure

The same goes for all other knobs in this prefab—scale them; then display the GUI float value plug. The Rate plug is the sole exception; it uses a third-party module to convert volts to frequency. Figure 2.60 outlines the structure. KDL Volts2Hz converts the output voltage to Hz. Insert > Convert > Volts to Float then converts the Hz value to float, and a Patch Mem–Float Out sub-control converts it to GUI float. This sub-control provides the value in GUI float plug format for conversion to text and display. The unit may connect to the Patch Mem–Float Out module's Name plug. Once you configure the readout, it shows you the LFO's exact frequency in Hertz.

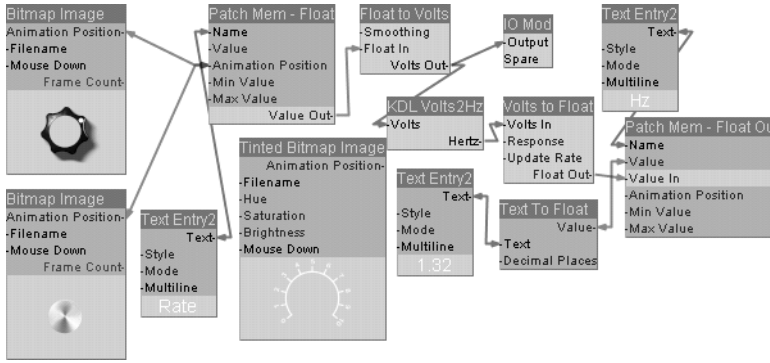


Figure 2.60: Structure of the Rate knob

Go-to files: flanger4.se1

Cooking Up a Chorus Effect

A chorus' structure is much the same as a flanger's, apart from a few differences. For one, the chorus' delay is longer, usually around 20 to 30 ms. For another, chorus plugs usually do without feedback, though you are free to create experimental effects with longer delays and feedback. Finally, flangers often use one (mono) or two (stereo) delay lines; choruses may use four, six, or even more delay lines (voices) to fatten up to the sound.

Our flanger prefab is a good place to start. First, we'll change delay modules' delay time to 0.06, or 60 ms. Now adjust the Min and Depth knobs accordingly so they give us the right readout. Setting the Multiply by value to 0.166667 multiplies the 0-to-5 volts range by 6, yielding a readout of 0 to 30 ms.

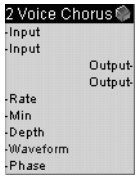


Figure 2.61

For the sake of convenience, we dumped the delay lines with the oscillators in a container called 2 Voice Chorus. Copying this container adds two voices to the chorus. Figure 2.61 pictures the container; figure 2.62 the structure.

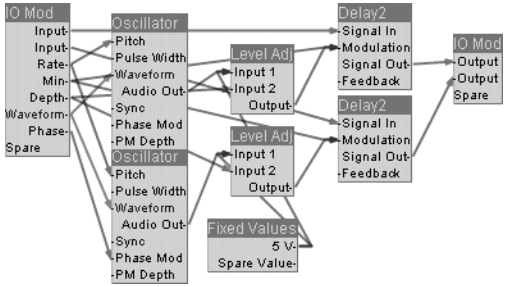


Figure 2.62: The two delay lines with LFOs inside a container

The controls connect to container plugs rather than modules. Check out chorus2.se1 for the full structure.

Go-to files:
chorus1.se1
chorus2.se1

Adding Two More Voices

More voices add girth to the sound. To add two, copy the 2 Voice Chorus container and connect the controls to the appropriate plugs. You'll find this structure in figure 2.63. The inputs connect to both two-voice chorus containers' inputs, with their outputs being mixed in the cross-faders' Input A plugs.

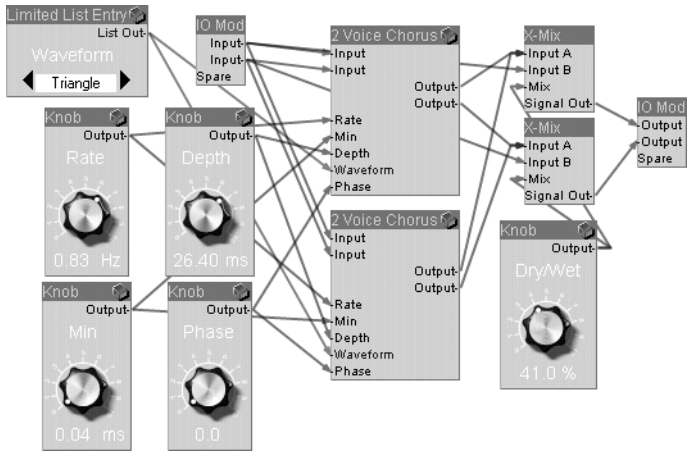


Figure 2.63: A chubby four-voice chorus

Different phases beef up the sound. Add a constant voltage to the oscillators' Phase Mod plug in the second dual-voice chorus. We added 10 volts in this prefab, as highlighted in figure 2.64. This shifts the phase 180 degrees, putting the two-voice pairs in opposite phase. Feel free to adjust the value or add a manual control.

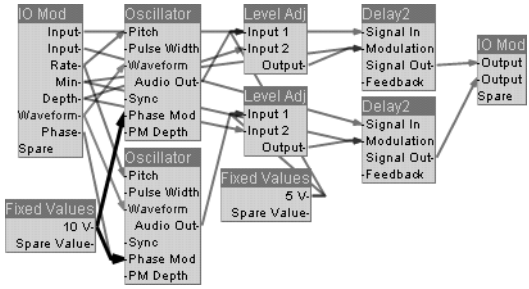


Figure 2.64: Shifting the LFOs' phase

Go-to files: chorus3.se1

Switching Voices Off

Sometimes, four voices is overkill, so a switch between two and four voices comes in handy. Though you have several options, the best way to conserve CPU resources is to switch voices off *before* they enter the container. We will use Switch (Many→1) modules for the second pair of voices. When "2" is selected from the list, a Fixed Values module feeds a constant 0 volts to the container's input, putting the container to sleep. When "4" is selected, the main input provides the signal, adding two voices to the wet sound. That's all it takes to conveniently switch between two and four voices.

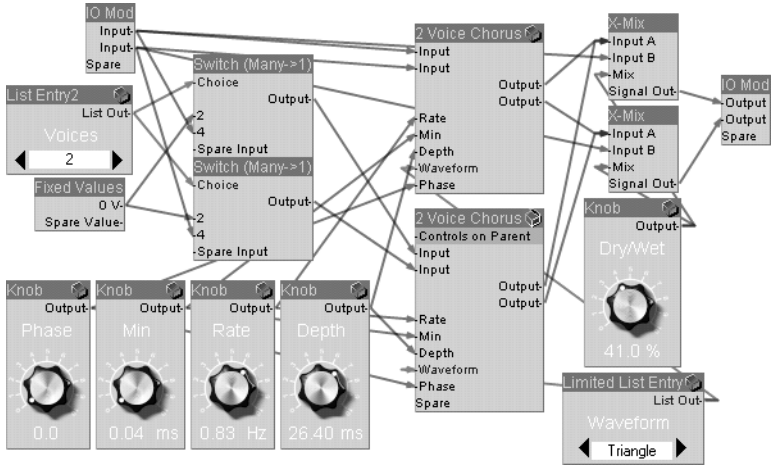


Figure 2.65: Here's how to silence the voices in a chorus effect

Go-to files: chorus4.se1

Phaser Effects

Phasers are like flangers, the main difference being that all-pass rather than comb filters carve notches and peaks into the spectrum. Though built-in and third-party all-pass filters are available, SynthEdit's constraints preclude real feedback. The good news is that third-party phaser modules and other filter types can simulate phasers' peaks and notches.

Phaser Variation 1

Our first version employs all-pass filters. Rather than changing the frequency spectrum, all-pass filters distort the incoming signal's phase. Take, for example, a sine wave. Applying an all-pass filter to it yields a sine wave of the same amplitude, but with a different phase determined by the sine wave's frequency. Different frequencies' phases shift differently. So, how do we notch the frequency spectrum if it remains unchanged? By mixing the wet and dry signal. Frequencies with opposite phases cancel each other out, cutting notches into the spectrum. The number of all-pass filters in the chain determines the number of notches.

The first example features a third-party module, EVM All-pass. On the upside, it does not hog as much CPU as the in-built all-pass filter. On the downside, it produces aliasing noise when modulated too fast because it updates filter coefficients less frequently. It will do for standard phaser modulation rates, though. A dual-stage phaser unit looks something like the setup in figure 2.66. The signal passes through two serial all-pass filters, and then mixes with the original signal.

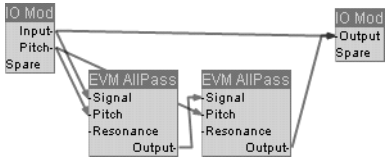


Figure 2.66: A two-stage phaser

We created similar containers with two, four, six, and eight serial all-pass filters to enable stage selection. A 1→Many switch selects the number of stages. Figure 2.67 shows this structure.

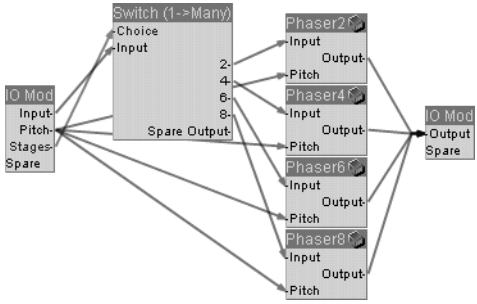


Figure 2.67: Selecting stages using a switch

Figure 2.68 outlines the main structure with two LFOs and controls much like those in the previous prefabs. A center knob with a 0 to 5 V range selects the center frequency. The two LFOs modulate the all-pass filters' pitch, creating the sweeping effect.

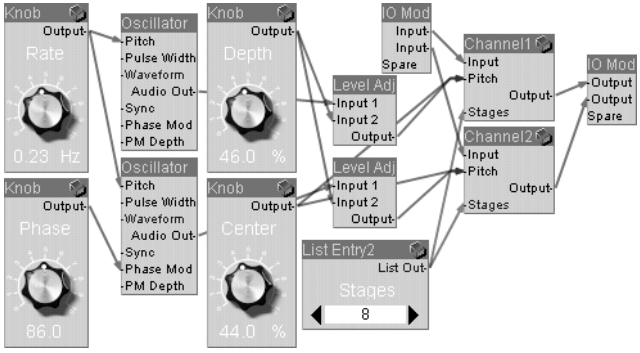


Figure 2.68: The phaser prefab's main container

Heads up: Swap the Channel1 and Channel2 containers for EVM Phaser modules, if you wish. They offer all-pass filters and let you select stages and control other parameters. See phaser2.se1 to learn more.

Go-to files:
Phaser\phaser1.se1
Phaser\phaser1.se1

Phaser Variation 2

The above prefabs simply notch the spectrum; they can't generate real feedback. But there are other ways to fake it. This example uses state variable filters to put notches and peaks in the spectrum, serving up a sound very much like a phaser. The trick is to subtract the high-pass output from the state variable filter's low-pass output, creating a resonant peak as well as phase distortion. Then if you route more of these filters in parallel, the phase cancellations create notches between the peaks as shown in figure 2.69—great for simulating a phaser's whoosh.

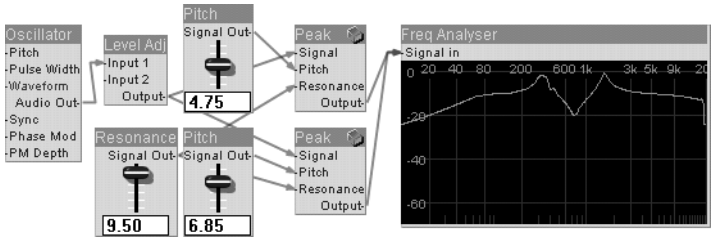


Figure 2.69: Parallel peak filters

Figure 2.70 pictures the Peak container's plumbing, whereby the Hi Pass out is subtracted from the Low Pass out.

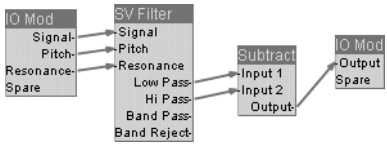


Figure 2.70: The Peak container's internal structure

A switch similar to the previous prefab's determines the number of stages. Each container holds parallel peak filters as depicted in figure 2.71. The fixed values add offset to the pitch, spreading the peaks in the frequency spectrum. The Multiply module scales the output, as the parallel filters boost the level.

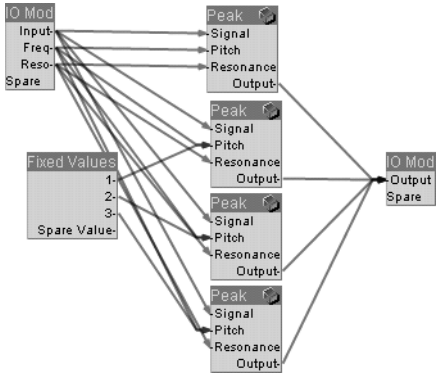


Figure 2.71: Parallel peak filters spread

The external structure mirrors the previous prefab's, except that here a Feedback knob controls the filters' resonance amount. Open the Feedback knob's structure and you will see how Subtract and Level Adj modules shape the output signal. All they do is calculate $1-(1-x)^2$ to change the resonance curve. Though not strictly necessary, this makes it easier to adjust the feedback amount. Figure 2.72 graphs this curve for you.

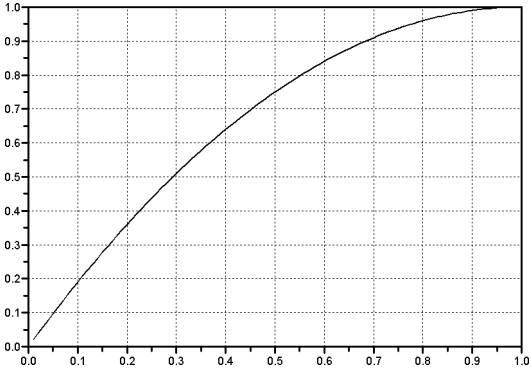


Figure 2.72: Resonance curve

See phaser3.se1 for the full structure. This effect's feedback emulation sounds rather sweet. Note that the notches' depth corresponds to the feedback amount.

Heads up: State variable filters are all-pole filters, meaning they boost high frequencies near Nyquist, possibly eliciting high-frequency ringing. Prevent this by limiting the phaser's maximum pitch.

Go-to files: Effects\Phaser\phaser3.se1

Equalization

Equalizers feature prominently in most audio applications. The term dates back to bygone days when filters compensated certain frequencies' attenuation in electronic equipment to elicit linear frequency response. Today equalizers serve many different purposes. They do things like adjust a boom-box's tone, boost certain frequencies of an instrument to make it stand out in the mix, and cut low frequency hum or noise. Shelving and peaking filters are the most common types. Shelving filters boost or attenuate frequencies below or above a specified cutoff frequency. Audiophiles call them low-pass shelving or low-shelf and high-pass shelving or high-shelf filters. Peaking or peak EQ filters attenuate or boost a narrow frequency band surrounding the cutoff frequency, leaving the remainder untouched. The Q factor or filter bandwidth determines the frequency band's width. Figure 2.73 charts some typical response curves. Conventional low-pass, high-pass, and notch filters also serve to equalize.

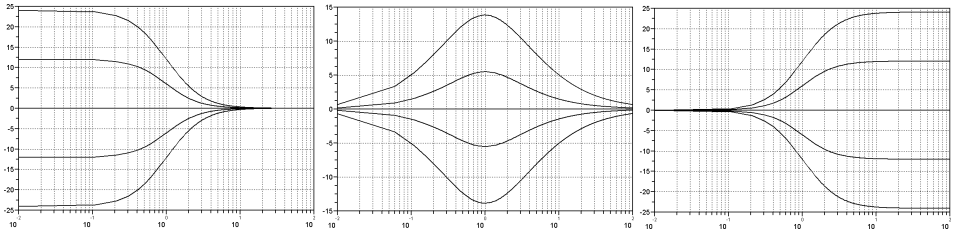


Figure 2.73: Low-shelf, peaking and high-shelf filters' response curves

Though there are many types of equalizers, for our purposes they come in four major categories. One comprises tone controls with fixed frequency bands for bass, midrange and highs like the equalizers on mixing desks and DJ gear.

Graphic equalizers make up the second category. They have several—usually from seven to 31—equally spaced, fixed frequency bands. On the upside, they offer far greater flexibility for shaping tone; on the downside, their bands' frequency and width are fixed.

Parametric or paragraphic equalizers are the third category. Featuring several bands with adjustable center frequency and bandwidth, they are by far the most versatile.

In the fourth category we find dynamic equalizers. With one foot planted in the dynamic processor camp and the other among equalizers, they respond differently to different input levels.

Now let's look closer at these different breeds of EQ.

Three-band Tone Controls

Welcome to tone controls, our first group of equalizers. The least flexible of the bunch, they offer just three fixed bands—low, mid, and high. Some mixing consoles feature adjustable midrange frequencies, that is, a sweepable (semi-parametric) equalizer. Frequencies vary with application and manufacturer. The low band is usually a 40 to 80 Hz shelving filter, the mid band is a peak EQ around 1.3 to 2.6 kHz, and the high band is a shelving filter around 12 to 15 kHz. We opted for 80 Hz, 2.5 kHz, and 12 kHz. Sadly, SynthEdit version 1.015 lacks shelving/peak filters; happily, great third-party modules are available. We'll borrow David Haupt's DH_BiquadFilter. It features shelving and peak filters alongside conventional low-pass, band-pass, high-pass, and notch filters.

DH_BiquadFilter sports two input modes, selectable in the preferences window. SE 0–10 volts mode enables the tried-and-true 0-to-10 volt range. The Hz/Octave/dB range specifies values in Hertz, octaves, and decibels. The last sounds good to us.

Creating the three bands is sooner done than said. First line up three Bi-quad filters in series. Set the low band filter to a low-shelf, 80 Hz frequency. Set the mid band to EQ Peak, and its frequency to 2500. The midrange's bandwidth is a matter of taste; this prefab uses 4 V (4 octaves) for a 625-to-10,000 Hz range. The effect is strongest at 2500 Hz, with a bell-like curve dropping off towards the band's corner frequencies. Set the third filter to high-shelf, and its frequency to 12,000. The Gain plugs determine boost and attenuation in decibels. Figure 2.74 outlines the structure of a mono three-band equalizer.

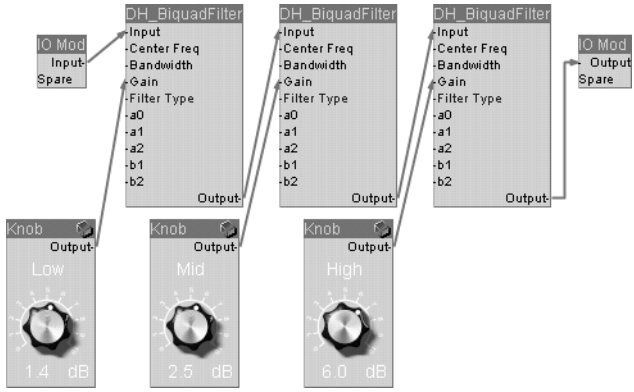


Figure 2.74: A simple 3-band tone control

Heads up: The DH_BiquadFilter module has a parameter named Quality with two settings, Economy and Fast Modulation. In Economy mode, filter coefficients are rarely updated, sparing CPU power. Modulating settings very fast can cause aliasing noise. We used Economy mode for this structure, but please use the other setting for prefabs enabling fast modulation.

Go-to files: 3band1.se1

Graphic Equalizers

You may add bands for greater flexibility. Graphic equalizers usually feature seven to 31 equally-spaced frequency bands. The more bands, the more precisely you can shape sound. Different equalizers use different frequency settings, but frequency bands are usually spaced equally along a logarithmic scale. Two methods determine the ratio of two neighboring bands. The base 2 method in 1/N octave mode is:

$$\text{UpperBand} = \text{BaseFreq} * 2^{1/N}$$

$$\text{LowerBand} = \frac{\text{BaseFreq}}{2^{1/N}}$$

The other method arrives at the center frequency using powers of 10. The 1/N octave mode rule is:

$$\text{UpperBand} = \text{BaseFreq} * 10^{N/10}$$

$$\text{LowerBand} = \frac{\text{BaseFreq}}{10^{N/10}}$$

Though the two methods yield slightly different results, they are practically the same. An ISO standard lists a number of preferred center frequencies. Many graphic equalizers, both software and hardware, use them. Table 2.1 lists these frequencies.

16	31.5	63	125	250	500	1000	2000	4000	8000	16000

Table 2.1/a. ISO preferred frequencies for one octave mode

16	22.4	31.5	63	90	125	180	250	355	500
710	1000	1400	2000	2800	4000	5600	8000	11200	16000

Table 2.1/b. ISO preferred frequencies for 1/2 octave mode

16	20	25	31.5	40	50	63	80	100	125
160	200	250	315	400	500	630	800	1000	1250
1600	2000	2500	3150	4000	5000	6300	8000	10000	12500
16000	20000								

Table 2.1/c. ISO preferred frequencies for 1/3 octave mode

In the following example, we will create a ten-band, one-octave graphic equalizer. Though you can use band-pass and peak filters to do this, our example employs peaking EQ filters. Each channel uses ten DH_BiquadFilter modules in serial array. Input modes are set to Hz/Octave/dB, and center frequencies are set according to Table 2.1/a. On both sides, a Slider connects to the Gain plug, with a range of –12 to +12 V. A Level Adj module for global gain adjustment sits in front of the output. Figure 2.75 shows the structure.

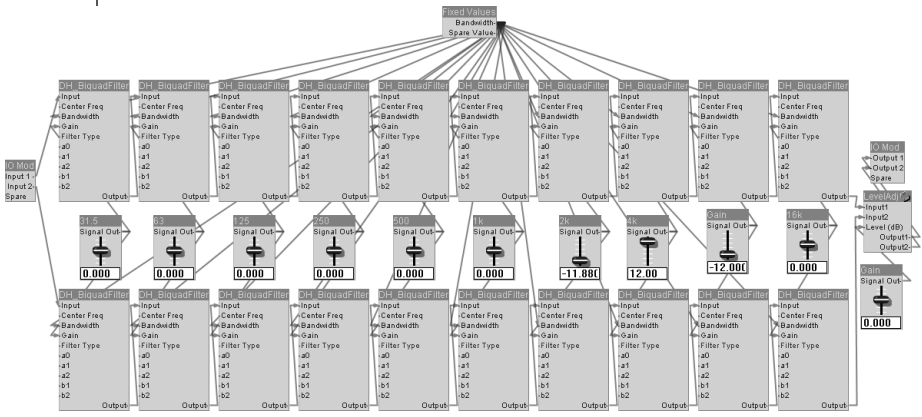


Figure 2.75: Ten-band equalizer (eq10-1.se1)

Heads up:

- Chose peak filters' bandwidth carefully. Different methods serve to set bandwidths, the most common being proportional-q (variable-q) and constant-q. The constant-q method adjusts the bandwidth for different gain settings to curb cross-talk between bands. The above structure is a classic proportional-q equalizer. If bandwidth is too narrow, a high-gain boost will yield a resonant sound. When boosting adjacent bands, narrower bandwidth may introduce ripple. If bandwidth is too broad, you will get more band interaction. 1.3 octaves is a good choice of amount. That way, when you boost two adjacent sliders by +12 dB, band interaction boosts total gain to about +17 dB.
- A greater slider range also increases ripple, so don't go beyond a 12-dB boost.

Go-to files: eq10-1.se1

Adding Stereo Controls with Link Switch

Add another group of sliders to the second channel if you wish to build a stereo EQ. Many plug-ins offer automatic channel linking for convenience. SL Slider Linker is a third-party sub-control module that does this. Setting the Link GUI Bool plug to True links Slider 1 and Slider 2 plugs' values. Change one, and the other changes accordingly. Figure 2.76 depicts a structure with two linked sliders.

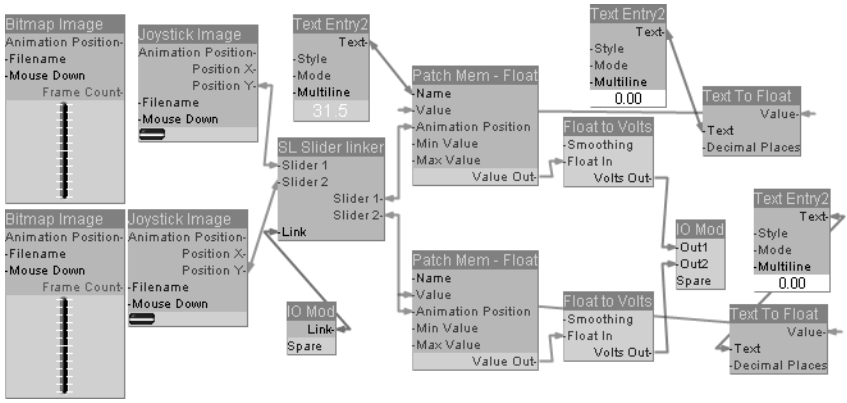


Figure 2.76: A structure with two sliders using SL Slider linker

Each slider has a Patch Mem–Float module assigned, but the SL Slider linker connects the Animation Position knob to the Joystick Image. Setting the container's Link plug to True prompts the Slider Linker to link the two sliders' position.

A Joystick Image lets you create a slider animation in SynthEdit. It provides the slider's knob animation, while the Bitmap Image supplies the background. This prefab requires two custom skin features, vslider_med_handle.png and vslider_med_back2.png, found on www.wizoobooks.com/synthedit. Please copy them to the skin folder before loading this prefab. These images base on the default vertical slider; skin and customize them as you wish. For more information on how to use the Joystick Image, please refer to the section "Joystick Image" on page 213.

The usual suspects make up the other features in this prefab. Float to Volts sliders' connects the Position sliders to voltage. Text To Float and Text Entry2 modules display the readout and frequency. We placed one slider below another in the GUI. Figure 2.74 shows their structure.

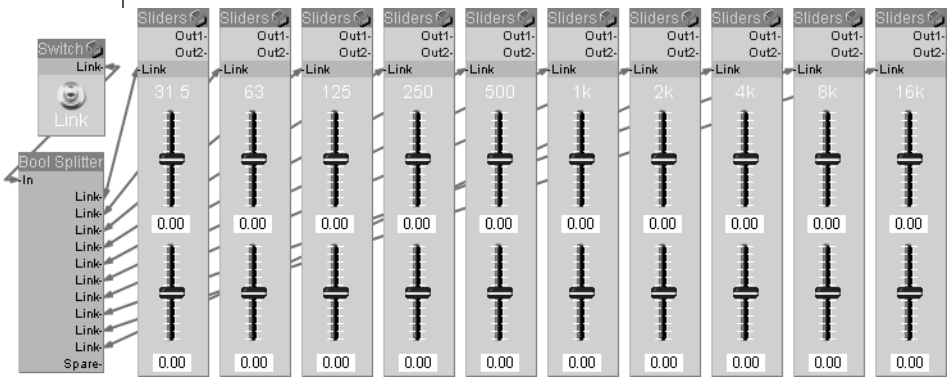


Figure 2.77: Linked sliders

The switch bases on the Controls > Switch prefab. Figure 2.75 maps its structure. Rather than using a Patch Mem module for animation, we opted for a Float to Bool module to convert the switch's status to a GUI Bool value. The Text Entry2 with the Patch Mem merely labels the switch. GUI plugs connect to just one GUI Bool plug. This means we need a Bool Splitter (Insert > Sub-Controls > Bool Splitter) module to connect the Link plug to all the sliders.

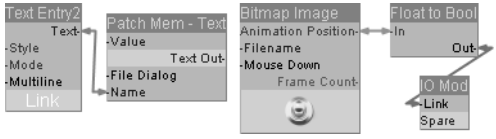


Figure 2.78: Link switch

Linkable sliders connect to the left and right channels' Gain plugs. Figure 2.79 illustrates the user interface.

Heads up: Add meter modules to monitor input/output signal levels (Controls > Peak Meter), if you wish.

Go-to files: Effects\Equalizer\eq10-2.se1
Effects\Equalizer\Skin files\vslider_med_handle.png
Effects\Equalizer\Skin files\vslider_med_back2.png

Be sure to copy the png files to the actual skin folder.

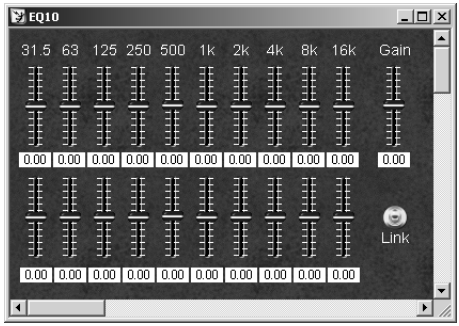


Figure 2.79: A stereo ten-band equalizer's interface

Parametric Equalizers

Parametric equalizers afford users great flexibility, enabling them to adjust equalizer bands' center frequency and bandwidth. Some parametric equalizers also feature different filter types ranging from low-pass and high-pass to shelving and peaking filters. The following example shows you how to create a four-band parametric equalizer with one low shelf, one high shelf and two peaking bands.

This example uses DH_BiquadFilter modules much like those in the previous prefabs. Four filters are arrayed serially for each channel, all operating in Hz/Octave/dB mode. One is configured as a low shelf, another as a high shelf, and two more in EQ Peak mode. The controls are also akin to the previous prefabs'. Gain ranges from –18 to 18 dB; bandwidth is specified in octaves. Frequencies range from 20 Hz to 20 kHz, which equals low and high knob values of 0.5405684 and 10.506353, respectively. The KDL Volts2Hz module converts these values to Hz. Figures 2.80 and 2.81 show the structure and user interface.

Heads up:

- Add bands for more precise control. This ups the CPU load, so if you use many bands, equip them with On/Bypass switches. See the section "Optimizing Effects" on page 185 to learn more.
- Add filter type selectors for each band if you wish to afford users even greater flexibility.

Go-to files: Effects\Equalizer\eq_para4.se1

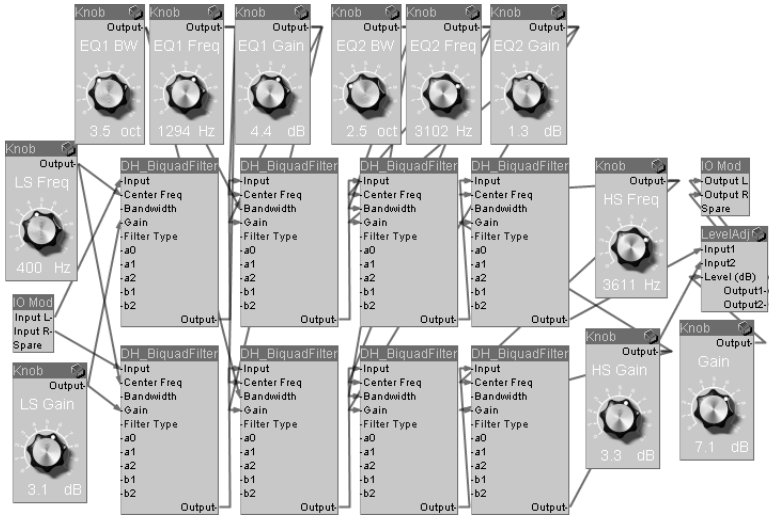


Figure 2.80: A four-band parametric equalizer's structure

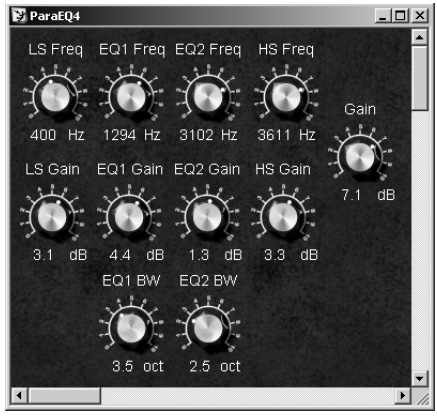


Figure 2.81: A four-band parametric equalizer's user interface

Dynamic Processing

These processors shape a signal's dynamics and dynamic range. Compressors, limiters, expanders, and gates are most common types. This book's scope is limited to explaining compressors and limiters, but expanders and gates' structures are very similar.

Compressors limit a signal's dynamic range, reducing its amplitude above a given threshold value. Sometimes called make-up gain, a gain control boosts the softer parts of the signal. A ratio control adjusts the compression amount by determining the ratio between input and output levels. Set ratio to 2:1, and the compressor halves signal levels above the threshold. Figure 2.82 graphs some compressor/limiter transfer curves. Any ratio above 10:1 constitutes limiting. Peak limiters' ratio is generally infinite to 1. This equates to hard clipping, strictly maximizing the output level.

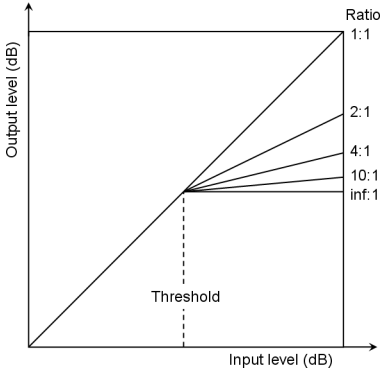


Figure 2.82: Compressor/limiter transfer curves

Figure 2.83 outlines the general scheme of a dynamic processor. You see two paths in the diagram. The detection path determines the peak or RMS level, and calculates the amount of gain reduction, which the processor then applies to the main path.

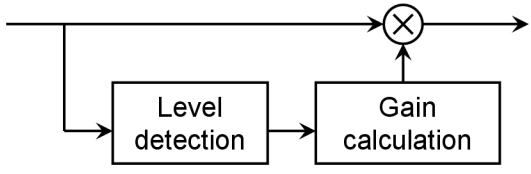


Figure 2.83: Dynamic processors work like this

Compressors' level detectors sport two controls, attack and release. Attack determines how fast the compressor cuts the gain when it detects a signal level exceeding the threshold. The release value determines how fast the signal returns to its original level once it drops back below the threshold. Peak limiters' response to sudden peaks must be instantaneous, so hard limiters only have a release control.

Setting Up a Simple Peak Limiter

Figure 2.84 captures the structure of a simple peak limiter in all its signal-routing glory. The Pregain knob adjusts the incoming signal level. A DH_Max module available from David Haupt's BasicModulePak takes the louder of the two channels, linking the left and right channels' gain reduction to preserve the stereo image. This signal then goes to the level detector, a Peak Follower (Insert > Modifiers > Peak Follower). The Attack plug's value is 0 volts, ensuring it responds immediately to sudden peaks. The Release knob adjusts the limiter's decay rate. The Attack/Decay knobs' range is 1 V/20 ms, and the readout is scaled accordingly.

A word on the Release knob: The Multiply module squares the voltage before feeding it out. This means the knob's scale is exponential; well, sort of. In any case, it brings greater definition to low release times. To set the high and low values, enter the target values' square roots to the Min Value and Max Value plugs.

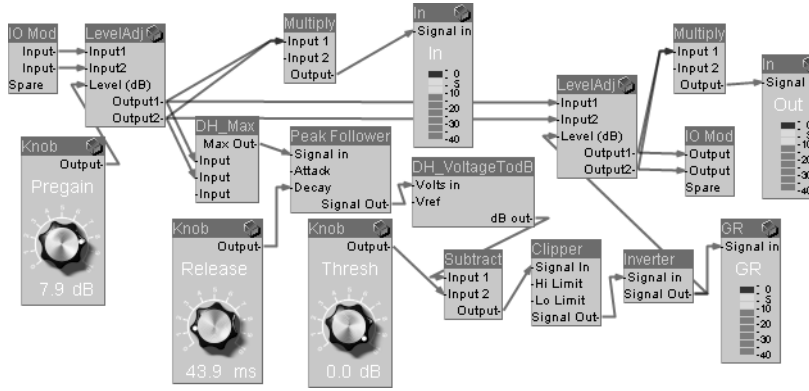


Figure 2.84: A simple peak limiter's structure (limiter1.se1)

Once the compressor detects the peak, David Haupt's DH_VoltageTodB module converts the level to decibels. Dynamic processors generally work with decibels rather than linear voltages. Though not compulsory for peak limiters, it does make levels easier to handle.

Then the compressor subtracts the threshold value from the decibel level. If the level lies above the threshold, positive values give the gain reduction in decibels. Negative voltages are a no-go, so a Clipper (Insert > Effects > Clipper) module clips negative voltages to 0. The compressor then inverts the gain reduction value to apply it to the main signal. The formula for this is:

$$GR_{dB} = -\max(In_{dB} - Thr_{dB}, 0)$$

The peak meters show the actual levels to help users keep track of the signal's status. We used the Peak Meter2 prefab from the Insert > Controls menu. Note that the prefab's Volts to Float module converts the level to animation. Open this module's Properties menu and you will see two settings, Response and Update Rate. In/Out peak meters employ a dB peak setting with fast response. The Gain Reduction meter uses volts DC (Fast), because the signal is already in decibels. The prefab converts the two channels to mono before displaying meters.

Heads up:

- Lowering the release value increases harmonic distortion. Set release to 0, and the limiter behaves like a hard-clipping distortion module. You may want to limit the release parameter's low value to prevent this. This prefab's low value is 1 ms. Extreme settings may elicit serious distortion.
- The Peak Limiter in version v1.0150 may "leak" signals at very high levels and low release rates, possibly causing more clipping and distortion.

Go-to files: Effects\Dynamics\limiter1.se1

Putting Together a Peak Compressor

A hard-knee peak compressor is much like the limiter shown above. Figure 2.85 depicts the structure. You'll find some differences, though. An Attack knob adjusts how fast the compressor responds to signals above the threshold. The Gain knob connects to the Level Adj module for the main signal, which lets the user dial in post compression make-up gain. The Ratio knob adjusts the amount of compression.

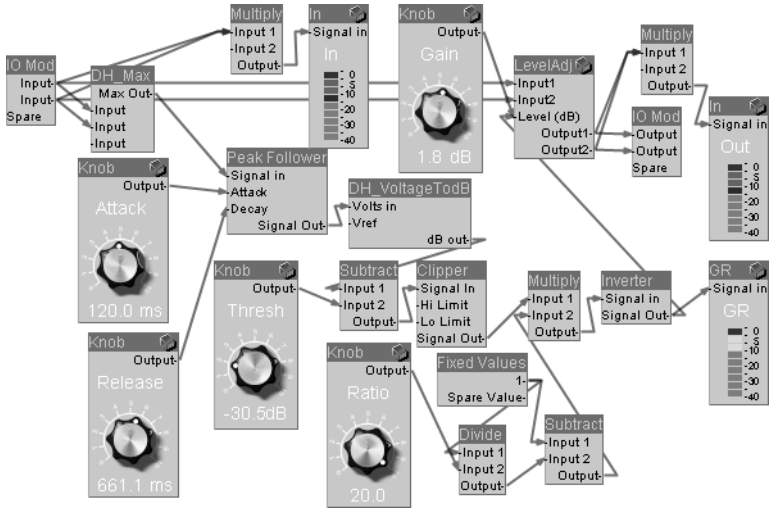


Figure 2.85: A hard-knee peak compressor (compressor1.se1)

The compressor does nothing to the sound at a 1:1 Ratio knob setting. Set it to 1:20, and you will dial in some serious compression. This structure takes the ratio value's reciprocal, and then subtracts it from one, yielding a value ranging from 0 (1:1) to 0.95 (20:1). The compressor multiplies the difference between input level and threshold by this value. This is how the Ratio setting determines the amount of gain reduction. The equation goes like this:

$$GR_{dB} = -\max(In_{dB} - Thr_{dB}, 0) * \left(1 - \frac{1}{Ratio}\right)$$

Go-to files: Effects\Dynamics\compressor1.se1

Adding an RMS Level Detector

Up to this point, our level detector focused on signal peaks to create peak limiters and peak compressors. But some users may wish to detect overall loudnes rather than peaks. RMS (Root Mean Square) frequently serves this end, making the level detector behave more like the human ear. RMS entails taking the square of N input samples, the mean (average), and then the square root. Stated mathematically, this is

$$x_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \ldots + x_N^2}{N}}$$

How to Average

Two methods serve to compute a signal's average, moving average, and infinite impulse response filters. A moving average filter takes the weighted sum of N inputs. An infinite response filter takes the weighted sum of inputs and outputs. We'll use the latter.

The impulse response of a first-order IIR filter like the 1 Pole LP filter in SynthEdit is an exponentially decaying signal, so recent samples are given more weight. In reality, though, response is endless, hence the name infinite impulse response. A one-pole filter's time constant is the time it takes to reach the level of 1/e, or about 37 % of the original level.

Now let's do the math for a one-pole filter. The equation for calculating a signal is:

$$y[n] = (1-a) * x[n] + a * y[n-1]$$

The relationship between a and the one-pole filter's time constant:

$$a = \frac{-1}{e^{f_s t}}$$

Here t is time constant in seconds, and f_s the sampling frequency. The relationship between a and the cutoff frequency is:

$$a = e^{(-2 * pi * \frac{f}{fs})}$$

It follows that the mathematical relationship between the filter's time constant and frequency is:

$$f = \frac{1}{2\pi t}$$

Here t is the time constant in seconds. So, if we want an averaging filter with a time constant of 1 ms, we can use a 1 Pole LP filter with a 159-Hz cutoff frequency.

Figuring Out RMS

The setup in figure 2.86 performs an RMS calculation. It squares the signal, and then averages it using a 1 Pole LP filter. Equation 1 is the reference for setting the filter's frequency. Then it takes the square root. Figure 2.87 shows the square root structure.

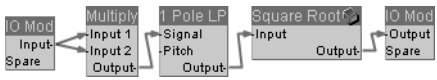


Figure 2.86: Calculating RMS

We used a Waveshaper2 module to calculate the square root, which can apply any transfer curve to a signal. In essence, it is a wavetable with a arbitrary mathematical function. However, its input signal range is limited to –5 to +5 volts. The compressor's input value usually ranges from –10 to +10 volts, so the square will lie in the 0-to-100-volt range. Dividing it by 10 and subtracting 5 scales it to –5 to +5 volts. After this, we use the function *sqrt((x + 5)/10) ∗ 10*. The *x* variable represents the input value; the other operations serve to scale it. This structure provides the square root of voltages between 0 and 100 volts.

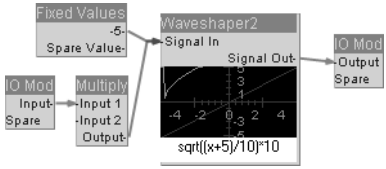


Figure 2.87: Calculating the square root

Heads up: Feel free to also use Oli Larkin's OL_Squareroot module to compute a signal's square root.

Adding an RMS Level Detector to the Compressor

Now that we have prefabs for calculating RMS level, we can add a selector for detecting peak/RMS levels. See figure 2.88 for its structure. Depending on the choice, the peak or RMS value goes to the DH_Max module, and then to the Peak Follower module for adjusting Attack/Release settings.

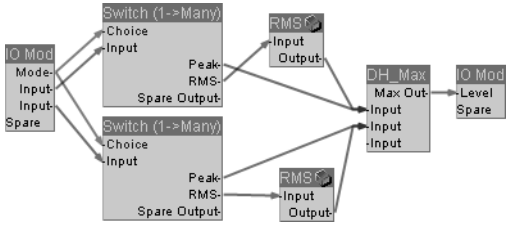


Figure 2.88: A switchable peak/RMS level detector

The RMS detector's time constant is somewhat of an arbitrary choice. Plug-ins use values ranging from one to 30 ms. The lower the time constant, the more sensitive the compressor is to sudden peaks. At very low time constants, the RMS detector behaves almost like a peak detector. This structure uses a 3-ms time constant. To this end, we set the one-pole filters' cutoff frequency to $1/(2 * \pi * 0.003) = 53$ Hz.

Go-to files: Effects\Dynamics\compressor2.se1

Creating a Soft-knee Compressor

Though we've discussed hard-knee dynamic processors, let's back up a bit to find out what a knee is and how it affects sound. Check out the two curves in figure 2.89. One bends hard at the threshold level. Compressors with this type of transfer function are called hard-knee compressors. The dotted line is a smooth curve called a soft knee. Compressors with this transfer function are called soft-knee compressors. A soft-knee compressor's gain reduction kicks in at a very low ratio a few dBs below the threshold. As the signal approaches the threshold, the

ratio increases until it arrives at the threshold. This transfer function comes courtesy of analog circuitry, where diodes do this smoothly. Many plug-ins offer soft-knee compression because it sounds more musical. So let's create a soft-knee compressor.

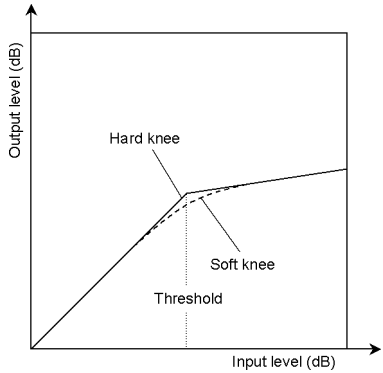


Figure 2.89: Soft and hard knee compression

In our earlier examples, the Clipper module shaped the transfer curve shown in figure 2.90. Above 0, the output level equals the input level. Below 0, the output level is 0. Smoothing out this curve with a knee replacing the sharp corner changes the compressor's characteristic.

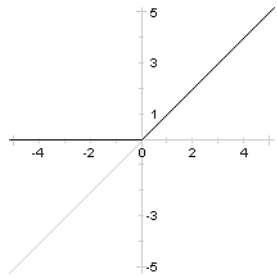


Figure 2.90: Transfer curve of a Clipper module

Let's use a Waveshaper2 module to this end. Note figure 2.91. We employed an exponential function in conjunction with min( ) and abs( ) functions to create the curve. Use this structure in the Clipper module's stead to endow the compressor with soft knee. The Multiply mod-

ules scale levels so that one volt equals 10 dB. The knee spans from about –15 to +10 dB around the threshold. Though not much of a range, it is just enough to smooth the sound a touch. Subtract this function from *x* to arrive at the transfer curve shown in figure 2.91.

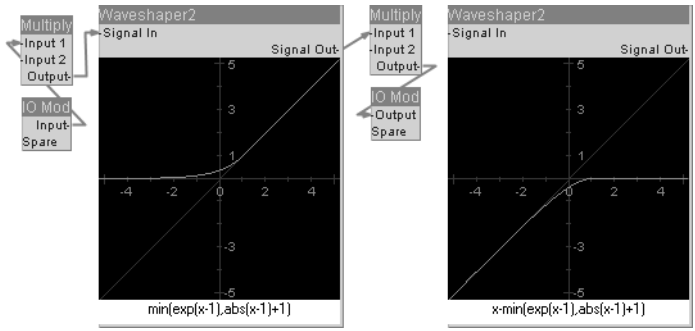


Figure 2.91: A soft-knee prefab showing the transfer curve

To let users choose between a soft and a hard knee, add a structure with switches rather than the Clipper module. See figure 2.92 for a structure that selects the type of knee. It sports two switches, 1 → Many and Many → 1. Here's why: When the 1 → Many switch selects a hard knee, the modules in the soft-knee container go to sleep. Yet they continue to issue residual constant voltage from previous calculations. A second switch prevents this constant voltage from influencing the sound. Usually 1 → Many switches are preferable, this being an exception. To learn more about optimizing patches, see the chapter "Making the Most of Performance" from page 181 onwards.

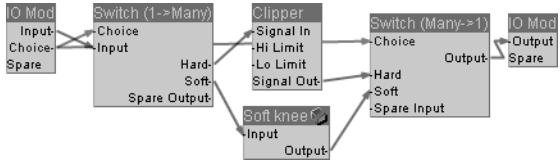


Figure 2.92: Soft/hard knee selector

Now you are the proud owner of a flexible compressor/limiter plug-in with variable knee and switchable peak/RMS level detection. Figure 2.93 pictures the GUI.

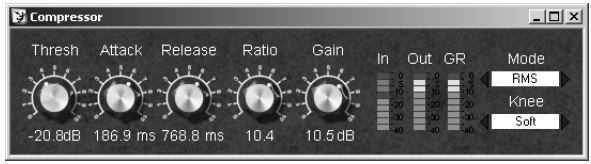


Figure 2.93: The compressor's user interface

Go-to files: Effects\Dynamics\compressor3.se1

Heads up: Use a Waveshaper module in place of Waveshaper2, and you can create just about any transfer function.

Getting Down and Dirty with Distortion Effects

For decades, musicians have been throwing some grit in pop music's gears with distortion and overdrive. This usually involves slapping some kind of transfer function (wave-shaping) on the incoming signal to generate new harmonics. The most common distortion types are hard-clipping, soft distortion (or overdrive), and fold-back distortion. Hard clipping limits signal to a certain level, chopping off whatever lies above it. This is common practice among transistors and operational amplifiers. Soft saturation also affects high signal levels, but the waveform retains some of its original characteristic. In analog circuits, vacuum tubes or diodes create this type of saturation. Vacuum tubes' distortion is often asymmetrical, meaning positive and negative voltages are affected differently.

Figure 2.94 charts a sine wave. The dotted line represents the same wave with soft distortion. The dashes represent the wave clipped hard at 0.5 and –0.5.

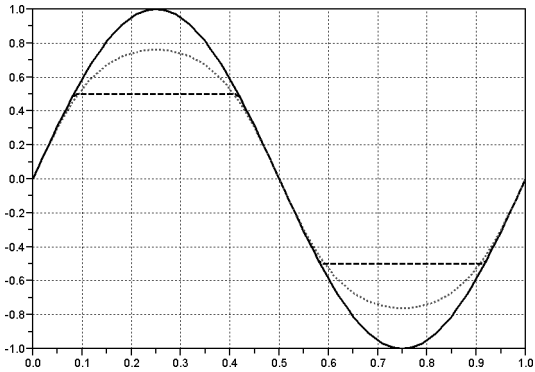


Figure 2.94: Distortion curves

Hard-clipping generates oodles of harmonics; the waveform all but resembles a square wave. Figure 2,95 shows these harmonics in a frequency analyzer. Here a 1 kHz sine wave is clipped at 2.5 and –2.5 V, creating beau coup harmonic content above 1 kHz.

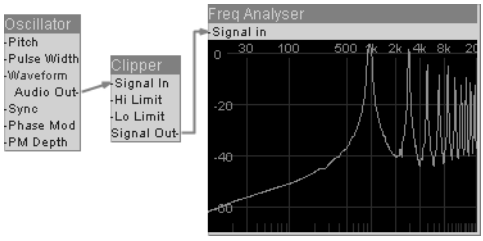


Figure 2.95: Harmonic distortion generated by hard-clipping

Hard Clipping

A Clipper (Insert > Effects > Clipper) module like the one above can serve to clip signals hard. Hi Limit and Lo Limit plugs determine maximum and minimum levels. The basic setup in figure 2.96 yields hard-clipped distortion. This setup boosts the level high enough to clip the signal. The Threshold knob adjusts the Clipper module's high and low limits. It converts the dB value to voltage using a DH_dBToVoltage module, and inverts it for the negative limit.

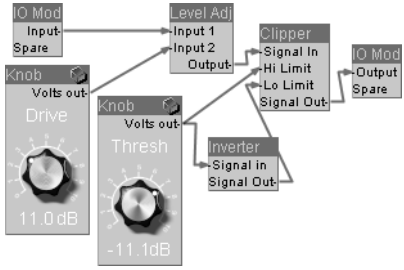


Figure 2.96: Hard-clipping

Go-to files: Effects\Distortion\hardclip1.se1

Soft Clipping

SynthEdit affords you many options for configuring a soft-clipping setup. Your choices are dedicated third-party modules such as sc:Soft-Drive and DH_SoftDist, Waveshaper modules, and polynomial distortion. Figure 2.97 portrays one soft-clipping setup. A Waveshaper2 module provides distortion by way of a tanh() trigonometric function. This setup is very similar to the internal structure of the DH_SoftDist module. You may change the transfer function or replace the Waveshaper2 module with a Waveshaper enabling any transfer function, including asymmetrical.

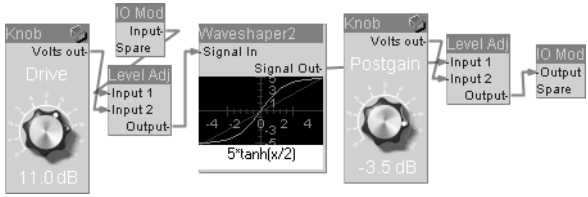


Figure 2.97: Soft clipping

Go-to files:
Effects\Distortion\overdrive1.se1
Effects\Distortion\overdrive2.se1

Fold-back Distortion

Those of us who are slavishly devoted to digital synths love fold-back distortion waveshapers because they are so easily realized with software. A fold-back effect inverts or folds back signal levels extending beyond a certain threshold. This conjures a distinct sound with flavor much like frequency modulation or phase modulation synthesis. Though third-party modules (RH-Fold-back, RH-Fold-back2) do this, using a Waveshaper2 module is an easy option. Enter

$$3.75 * (\text{abs}(-\text{abs}(-\text{abs}(x+1.25)+2.5)+2.5)-1.25)$$

to Waveshaper2 to fold back all signals twice. Figure 2.98 shows the structure with the transfer curve. Note the absence of distortion in the –1.25 to 1.25 V range. Multiplying the input by 0.25 scales –5 to 5 volts to this range. Subtracting 12 dB from the Drive amount yields similar results because $10^{-12/20} = 0.2511$.

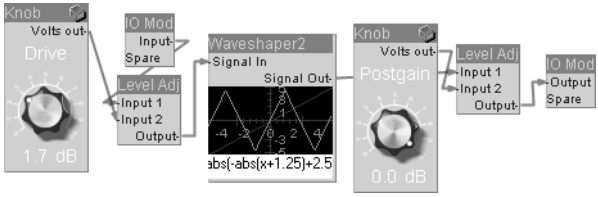


Figure 2.98: Fold-back distortion

You may use a sine function to soften the fold-back. If you introduce the function

$$5 * \sin(x * 1.5)$$

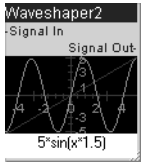


Figure 2.99

... the Waveshaper first provides soft distortion, then folds the signal back in a soft curve twice, and finally clips it to ±5 volts. Presto, there you have your all-purpose sound softener. Figure 2.99 charts the transfer curve. To fold back the signal softly just once before clipping, use:

$$5 * \sin(x * 0.9)$$

Go-to files:
Effects\Distortion\fold-back1.se1
Effects\Distortion\fold-back2.se1
Effects\Distortion\fold-back3.se1

What's Up with Aliasing?

Let's look at what happens to high frequencies during distortion. The curly line in figure 2.100 shows the spectrum of a 16 kHz hard-clipped sine wave, creating lots of upper harmonics. These harmonics should all lie above 16 kHz, so you wouldn't expect to see anything below this frequency. But the frequency analyzer detects plenty of frequencies with significant amplitude below 16 kHz, down to about 400 Hz.

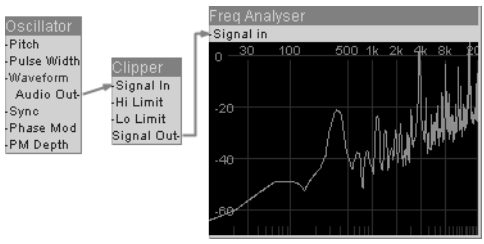


Figure 2.100: Aliasing

So, where do these frequencies come from? The answer lies in the Nyquist-Shannon sampling theorem. It states that the frequency bandwidth equals half of the sampling frequency. Dividing the sampling rate by 2 gives you the Nyquist frequency. The highest frequency at 48 kHz is thus 24,000 Hz. Now if you sample a signal with a frequency higher than the Nyquist, this frequency aliases back to the spectrum. In other words, frequencies above the Nyquist are mirrored below it, creating a signal with another frequency. Case in point: If we wish to sample a 28 kHz signal at a 48 kHz rate (and 24 kHz bandwidth), we will create a 20 kHz signal instead.

Picture a John Wayne movie. He's on a stagecoach, with the wagon wheels turning. As the wagon speeds up, there comes a point when the wheels appear to spin backwards, yet the wagon continues moving forward. The same phenomenon is at work here. The wheels are spinning faster than the camera's 30 frames per second can capture. The same goes for signals above the theoretical limit. They are simply mirrored back below the Nyquist frequency.

Though the process is called aliasing, to us it means noise. It adds nasty high-end frequencies to the original signal. Generally undesirable, aliasing commonly occurs when distorting or shaping the wave of a signal. In the above example, clipping generated harmonics above 16 kHz. Many lay above the theoretical 22 kHz limit, so they were mirrored back to the frequency spectrum, creating that mess below 16 kHz in figure 2.100.

There are two prevailing ways of reducing aliasing. One entails re-sampling the signal to a higher rate using interpolation filters. Then you do your wave-shaping, filter out the harmonics above the original Nyquist, and finally down-sample the signal to the original rate. Neither SynthEdit nor most native and custom modules offered this option at the time of writing. So you'll have to fall back on another approach.

Aliasing largely takes place in the high frequency spectrum; using a low-pass filter to cut high-end frequencies helps reduce aliasing noise. Applying distortion judiciously also limits this noise. A post-distortion low-pass is an option, but rather than preventing aliasing noise outright it merely filters some of it.

Adding Filters to the Sonic Equation

You may add filters to shape timbre. Removing low frequencies often improves overall performance, and lends the sound a different flavor. Filtering out high frequencies is a good option because this reduces aliasing noise. In the following example, we will use simple cascaded one-pole, low-pass and high-pass filters to cut out the low end and smooth out the top end. A peak EQ boosts a selected frequency pre distortion to add a pinch of sonic spice similar to a resonant filter effect. This prefab features three distortion types—hard-clipping, overdrive and fold-back distortion, switchable via the Type switch. The Drive knob adjusts the distortion amount; the Post Gain knob adjusts the output level.

Heads up:

- ❖ Add a low-pass post-filter to smooth high frequency harmonics created by distortion.
- ❖ Some plug-ins drop a band-pass filter in front of the distortion unit to shape timbre. It's also good for reducing aliasing.
- ❖ Some plug-ins use combinations of different distortion types to conjure distinct sounds.
- ❖ Post-equalization gives users another sound-sculpting tool.

Go-to files: Effects\Distortion\distortion1.se1

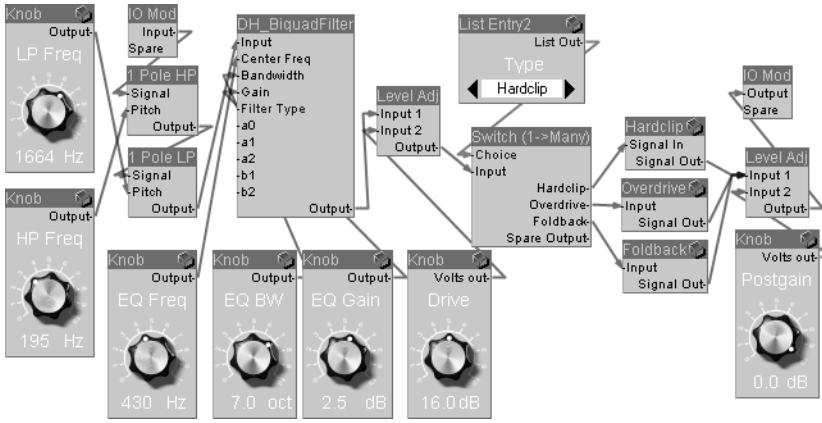


Figure 2.101: Distortion with filters (distortion.se1)

Getting Ugly with Lo-fi Effects

With up to 192 kHz sample rate and 24 bit depth, modern-day workstations sculpt sound with surgical precision. Back in the digital audio Stone Age, hardware and software capabilities were far more primitive. Sampling rates of 16, 10, or 8 kHz, sometimes with 12 or 8 bit depth were the norm. If you want to let your users get down dirty, help them recreate these lo-fi sounds with re-sampling or bit reduction.

Figure 2.102 depicts a simple re-sampler based on a Sample and Hold module from the Insert > Modifiers group. It samples and holds the input voltage, triggering the Hold plug. An Oscillator module controls the triggering rate; its rate determines re-sampling rate.

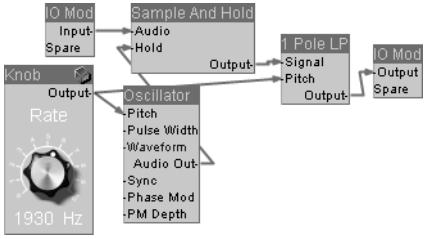


Figure 2.102: Lo-fi re-sampling (resampler2.se1)

This structure generates considerable aliasing noise above the re-sampling frequency. We may clean up the mess a little using filters. Figure 2.102 also shows a simple one-pole low-pass filter used to smooth top-end frequencies after re-sampling.

A Quantizer (Insert > Modifiers > Quantizer) module is great for simulating low bit rates. It constrains the input signal to voltage ranges specified by the Step Size plug. So, if you set step size to 5 volts, the output values will be –10, –5, 0, 5, and 10 volts. This is similar to 2-bit resolution—no pun intended—enabling values of $2^2 = 4$ different value. A switch with fixed values lets users select step size for different bit resolutions. Figure 2.103 illustrates a bit crusher effect with a list for selecting bit depth. In the fixed-value pyramid, every number is half the value below it. This means every step increases depth by one bit.

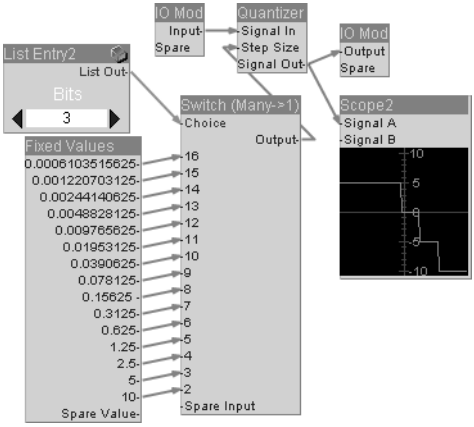


Figure 2.103: A bit crusher's structure (bitcrusher1.se1)

Heads up:

- An sc:Quantizer module is fully compatible with the in-built Quantizer, but consumes far less CPU power.
- Lance Putnam's LP-BitCrush module creates the same lo-fi effects, re-sampling, and bit reduction.

Go-to files:
Effects\Lo-fi\resampler1.se1
Effects\Lo-fi\resampler2.se1
Effects\Lo-fi\bitcrusher1.se1

Vocoders

Vocoders were designed in the 1930s to encode and transmit voice signals via phone lines, and then decode, synthesize, and render the original voice. The name is a portmanteau of *voice* and *encoder*. They usually analyze the spectrum of the input signal by splitting the input signal—called the modulator—into different bands using band-pass filters. Then they transmit this spectral data to another signal called the carrier using voltage-controlled amplifiers. This effect lends a vocal quality to synthesizers, guitars, drum loops, and the like.

Robert Moog introduced vocoders to music in the 1970s when he developed a ten-band device used by Wendy Carlos in the soundtrack to Clockwork Orange. A Moog synth provided the carrier signal; a microphone the modulator signal. Many artists since have used vocoders in pop music. Countless Hollywood robot voices came courtesy of vocoders.

Many analog vocoder units surfaced in the '70s and '80s, sporting from ten to 32 frequency bands for analysis and synthesis. Most used four- or six-pole band-pass filters with steep 24 to 36 dB/octave slopes to separate frequency bands. Steep filters reduce overlapping and interference between bands, improving encoded speech intelligibility. Figure 2.104 is a diagram of a vocoder.

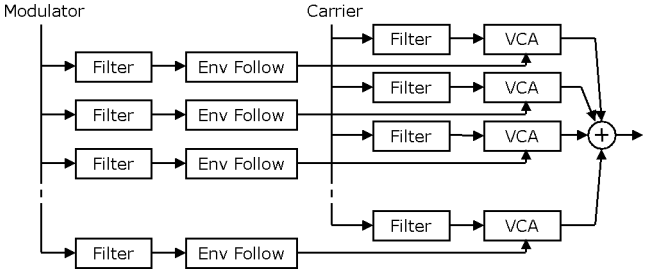


Figure 2.104: Vocoder schematic

Creating a Vocoder

DH_MultiFilter2 is a fine choice of band-pass filter for many reasons. For one, the Filter Stages switch makes its steepness easy to adjust. A two-pole band-pass filter has a 12 dB/oct. slope. With every added stage, the number of poles increases by two. So, three does the trick for a 36 dB/oct. slope. For the other, peak gain is normalized to 0 dB in BP2 mode. This lets you adjust the filter's Q factor and, by extension, its bandwidth, without dialing in big differences in gain. The third reason is that its internally cascaded filter stages are CPU-friendly.

Even if you're not obsessively tidy, it's wise to put each band in its own container. Figure 2.105 maps the structure of one vocoder band. First, the filters process both the modulator and carrier signal. Then a Peak Follower module extracts the modulator signal's envelope, whose voltage controls the carrier band's amplitude. The Level Adj module acts as a VCA, adjusting the carrier's volume. We set the DH_MultiFilter2 modules to BP2 mode, and filter stages to three. The input mode is Pitch/Res, making it easy to space the bands equally in the logarithmic frequency domain.

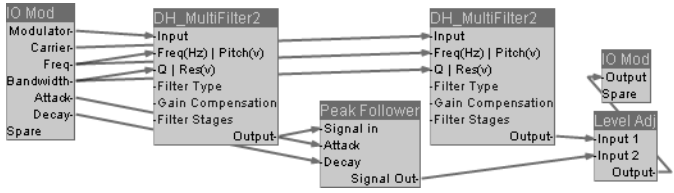


Figure 2.105: One vocoder band's structure

Figure 2.106 shows a vocoder structure with ten bands. Modulator, Carrier, Bandwidth, Attack, and Decay plugs connect to all the bands. Bands are spaced $^{3}/_{4}$ octave apart. Pitch voltages are 3.25, 4, 4.75, 5.5, 6.25, 7, 7.45, 8.5, and 9.25, roughly translating to 130, 220, 370, 620, 1050, 1760, 2400, 4500, and 8400 Hz, respectively.

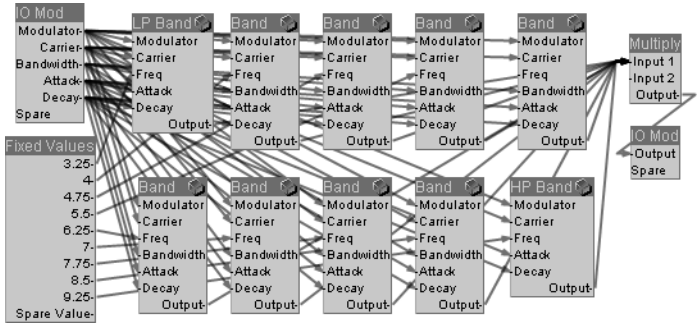


Figure 2.106: A ten-band vocoder's structure

We named the lowest band LP Band, and used a two-stage (24 dB/oct.) low-pass filter rather than a band-pass filter. HP Band designates the highest band, and as the name suggests, the container uses a 24 dB/octave high-pass filter. Analog gear often uses the same ten-band setup to cover the full frequency spectrum. Despite the few bands, you'll find the speech reasonably intelligible. Detecting voiced and unvoiced sounds helps improve intelligibility, as you will soon discover.

Figure 2.107 outlines a rudimentary stereo vocoder's structure. This prefab's carrier is an oscillator-generated saw wave. Though this is a common analog setup, you could replace it with something as complex as a full-fledged polyphonic internal synth engine. The stereo input signal is the modulator.

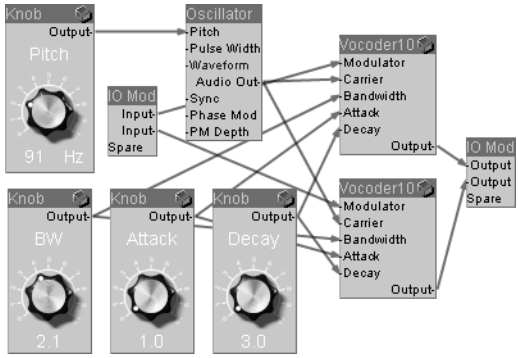


Figure 2.107: A basic stereo vocoder (vocoder1.se1)

The BW knob adjusts bandwidth. Dial in greater bandwidth, and cross-talk between bands increases. Backing off the bandwidth separates bands better. Above a certain point, bands become individual resonant peaks, sounding like a fixed bank of a resonant phaser.

Two methods serve to process an external modulator and external carrier signal. In many plug-ins, one input (for example the left channel) is the modulator, and the other is the carrier, which means settling for a mono sound. Side-chaining with four input channels is a stereo alternative. Two channels provide the stereo carrier, and another two the stereo modulator. Not all hosts support this configuration, however.

Heads up: Etric van Mayer has two third-party modules for creating vocoders, EVM DBV, a band-pass filter bank for one band, and EVM Vocoder.

Go-to files: Effects\Vocoder\vocoder1.se1

Improving Intelligibility

Vocoded speech sounds lifeless. Analog vocoder units often feature a white noise generator to simulate unvoiced sounds. Most unvoiced sounds, like the consonants s, t, k, and f, comprise high-frequency hiss with some resonant frequencies. Detecting the level of frequencies above 4 kHz serves to assess the sound's fricative quality. Our example does this by filtering frequencies below 4 kHz, and then using this control signal's envelope to adjust the white noise level. To see how this works, look no further than figure 2.108.

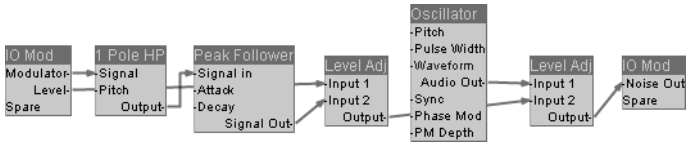


Figure 2.108: Detecting unvoiced sounds

Adding this noise to the carrier signal notably improves speech intelligibility. Figure 2.105 is a snapshot of this structure.

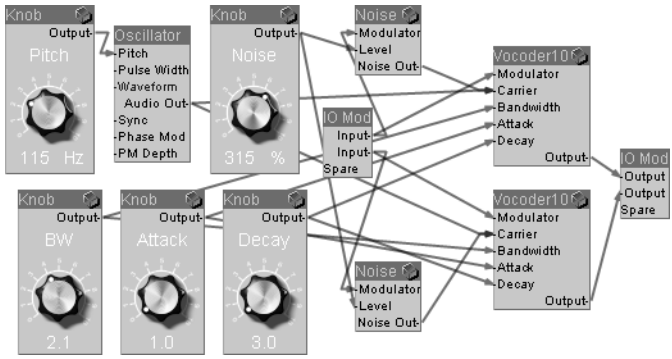


Figure 2.109: Vocoder with white noise (vocoder2.se1)

The Noise knob adjusts the level of noise used to fake unvoiced and fricative sounds. Employ only a noise source for the carrier signal, and you will get something akin to robotic whispering.

Go-to files: Effects\Vocoder\vocoder2.se1

More Pitch-shifting Fun

- Change the carrier bands' frequencies to cook up a pitch-shifting effect. Shifting carrier bands simulates positive or negative pitch changes. Some vocoders let you adjust the modulator bands' frequency; many let you adjust individual bands' levels.
- To configure a simple pitch-tracking setup, filter the modulator at around 250 Hz using a low-pass, and patch the signal to the oscillator's Sync plug. This synchronizes the oscillator to the 0 crossings in the modulator, making it roughly track the modulator signal's pitch.
- You can add a Freeze option by using Sample and Hold modules that retain modulator bands, status to let your users freeze the sound timbre.

More Mischief with Multi-band Processing

A vocoder is not the only application that calls for splitting the input signal into separate frequency bands. Occasionally, users wish to process low, middle, and high frequencies independently. A multi-band dynamic processor, frequently used for mastering, is one example. Splitting the input signal into bands lets you treat these bands with different compressor settings. And it reduces interference between frequency regions, for example, so that the kick drum doesn't encroach on the hi-hat's tone and dynamics.

Multi-band distortion is another popular application. Rather than subjecting the entire signal to a waveshaper, it is first sliced up into bands. This cuts cross-talk between bands, and gives you greater leeway for shaping sound. You could, say, leave the low-end untouched, squeeze the mid-range hard, and add a pinch of overdrive to the high-end. This is akin to harmonic exciter that shapes just a slice of the frequency pie and creates new harmonics.

Other applications include applying a different chorus effect to different frequency regions. Almost all effects combine with multi-band processing. Figure 2.110 is a schematic diagram of a multi-band processor. It shows three bands, though some dynamic processors use four or five. Crossover filters like those found in speaker cabinets split the input signal into individual bands. First the processor shapes bands separately, and then blends them to a composite output signal.

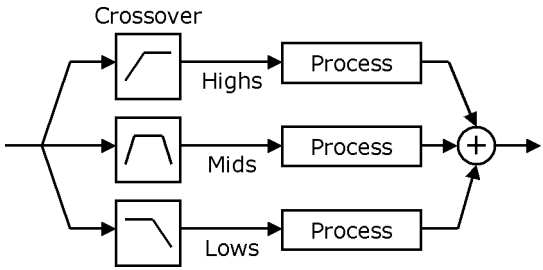


Figure 2.110: Multi-band processing

Crossovers

Crossover filters are key ingredients of a multi-band processor. Choose the filters carefully because they go a long way towards determining the processor's characteristic and transparency. A crossover filter separates frequency bands, yet it must retain a flat output when the two bands mix. Its phase response is also important for retaining transient transparency and minimizing band interaction.

Crossovers generally come in two categories, finite impulse response (FIR) and infinite impulse response (IIR) filters. FIR filters' linear phase is an advantage because it benefits transparent band separation. At the time of writing, FIR crossover filters were unavailable in SynthEdit. Most built-in and third-party filters—one-pole, state variable, biquad and Moog—are recursive IIR filters. Now let's look at the most important IIR crossover filters, hopefully without getting too tangled in filter theory.

One-pole Filters

The simplest crossover uses one-pole filters to separate the two bands. Surely the most straightforward option is to apply a one-pole low-pass filter to a signal, and then subtract the low-pass output from the original input. This yields the high-pass output, and also ensures the low-pass and high-pass filters' sum equates to the original sound. Figure 2.111 graphs a one-pole low-pass and high-pass filter's frequency response.

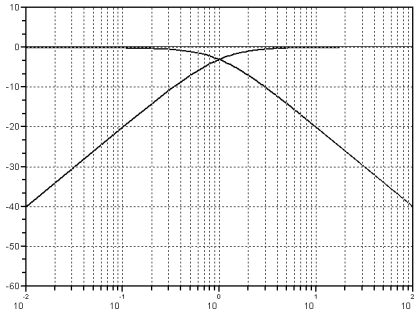


Figure 2.111: One-pole crossover frequency response

One-pole filters' phase and transient responses are strong, but with just 6 dB/octave slope, their band separation is weak. Figure 2.112 depicts a rudimentary one-pole crossover.

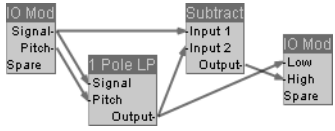


Figure 2.112: One-pole crossover structure

Linkwitz-Riley Filters

The speaker industry values Linkwitz-Riley filters for their flat summed output and good band separation. Let's see if we can second that emotion by cascading two Butterworth filters, each with –3 dB gain at the cutoff. The cascaded filters' gain at the crossover frequency is –6 dB, which comes to $10^{-6/20} = 0.5$. If the low-pass and high-pass filters' gain is 0.5 each at the cutoff and you add them, their sum equals one, that is, a flat frequency spectrum. Figure 2.113 tracks the frequency response of 24 dB/oct. L-R crossovers. The dashed line runs at –6 dB. This is where the low-pass and high-pass responses meet, yielding a flat sum.

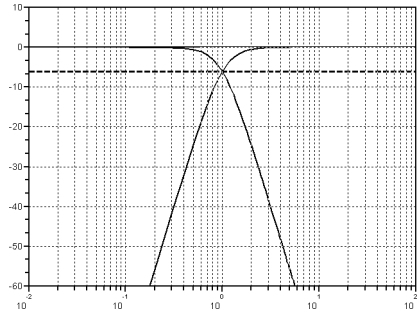


Figure 2.113: A 24 dB/oct. Linkwitz-Riley crossover's frequency response

12 dB/Octave L-R Filters

Cascading two one-pole low-pass or high-pass filters creates a two-pole (12 dB/octave) Linkwitz-Riley crossover because one-pole filters have a Butterworth characteristic. Using a DH_MultiFilter2 with a Q setting of 0.5 (that is, –6 dB) achieves the same result. Don't forget to set the Filter Stages to one, and the Input mode to Pitch/Q or Hz/Q. Remember also to invert the polarity of one band owing to each pole's 90-degree phase shift. Two poles add up to a 180-degree phase shift, so you must invert one band's polarity to put the two in phase. Figure 2.114 pictures two options that yield largely the same results.

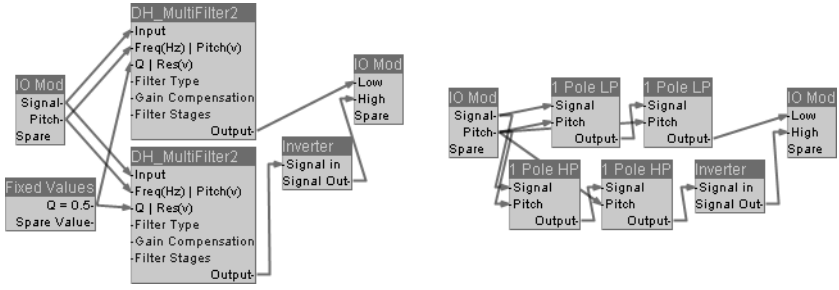


Figure 2.114: 12 dB/oct. Linkwitz-Riley crossovers

24 dB/Octave L-R Filters

Cascading two two-pole Butterworth filters creates four-pole Linkwitz-Riley filters. Though their band separation is steep, their transient response is less impressive. The simplest solution is to use a DH_MultiFilter2 in Hz/Q or Pitch/Q mode. Set Q to 0.7071, which

equals –3 dB. Then set Filter Stages to two to cascade two filters in serial array. No need to invert polarity here. The four poles yield a sum 360-degree phase shift, so the two outputs are in phase. Figure 2.115 shows the structure.

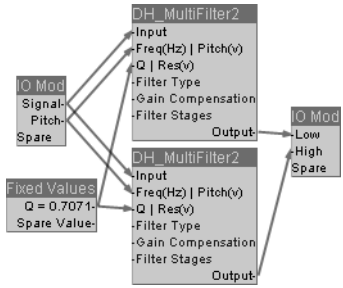


Figure 2.115: 24 dB/oct. Linkwitz-Riley crossover

This should cover the basics, for more math behind Linkwitz-Riley crossovers, click to http://www.linkwitzlab.com/crossovers.htm.

Go-to files: Effects\Multiband\crossovers.se1

Putting Crossover Filters into Practice

Now that you've fought your way through this windy introduction, you're probably eager to learn how to use these filters. Figure 2.116 is a schematic diagram of a three-way crossover separating the signal in low, mid, and high bands. F1 and F2 represent the crossover frequencies. The low band has a low-pass filter with the F1 cutoff. The mid band has a high-pass filter with the same frequency, and a low-pass filter at the second crossover frequency, that is, F2. All the high band needs is a high-pass filter with an F2 cutoff. Feel free to add more middle bands using a pair of low-pass and high-pass filters.

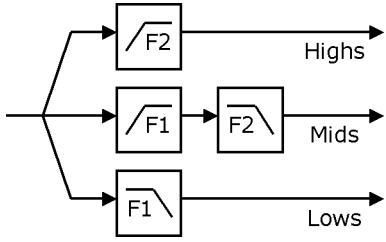


Figure 2.116: A schematic view of a three-band crossover

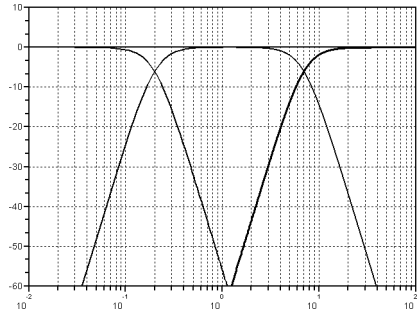


Figure 2.117: A 24 dB/oct. three-way crossover's frequency bands

Building a Two-band Compressor

Multi-band processing applications abound. Figure 2.118 shows one, a simple two-band compressor. Two-pole crossovers separate both the left and right input signals into low and high bands. Identical to the prefabs we used for dynamic processing, the two compressors process the bands separately. We enabled Controls on Parent in both compressor containers' Properties window to show their user interfaces in the main panel. This lets users handle an individual compressor's interface as a group in the main interface. Figure 2.118 gives you a view to how this works.

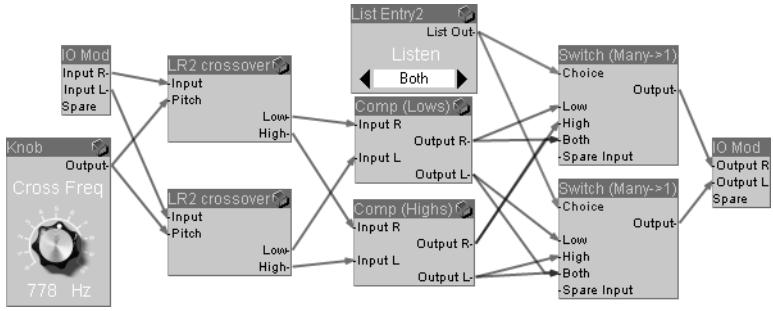


Figure 2.118: A two-band compressor's structure

Processing multiple bands is a complex chore, and it helps the user to hear band adjustments in isolation. Though inessential and often lacking in plug-ins, this option does come in handy. And it's easily implemented using switches that route signals to the output or cut them off

at the pass, so to speak. Users can select Lows to patch only the low compressor's outputs signal out, and Highs to patch only the high compressor's signal out. Both compressors' outputs connect to the Both plug, which enables users to tap a composite of the two bands.

Figure 2.119: The two-band compressor's GUI

Go-to files: Effects\Multiband\twoband_comp.se1

3

Stepping Up to Synthesis

Our trek through modular audio processing has arrived at an interesting juncture—sound synthesis. Synthesis entails generating audio signals electronically or digitally. Some synthesis techniques aim to reproduce the sound of real-world instruments. Others strive to create unprecedented, unique, or outright bizarre signals. The most common synthesis technologies are subtractive, frequency and phase modulation, wavetable, physical modeling, additive, and phase distortion.

Subtractive synthesis may well be the most widespread of the bunch. It uses different filters to selectively cull frequencies from a wave produced by oscillators and other sound sources. Digital Yamaha synths like the DX7 popularized frequency modulation synthesis. Ironically, they actually used phase modulation, where a waveform modulates the phase of another oscillator, to do this. Later more synthesis technologies surfaced, some of which featured fixed wavetables stored in memory; others mimicking the sound of physical instruments through physical modeling. Phase distortion synthesis is akin to FM synthesis, where changing a sine wave's phase shapes timbre. Additive synthesis creates sound by adding sine partials rather than subtracting ingredients of a spectrally rich signal. This chapter deals with the two key methods of synthesis, subtractive and FM synthesis, explaining their theory and implementation in SynthEdit.

Heads up: See also the Appendix for a brief history of synthesizers.

Less Is More with Subtractive Synthesis

Recapping Subtractive Synthesis

Subtractive synthesis is probably the most common approach to synthesizing sounds. Usually, oscillators create basic waveforms with rich spectral content. Figure 3.1 shows the most common waveforms—saw, pulse, and triangle.

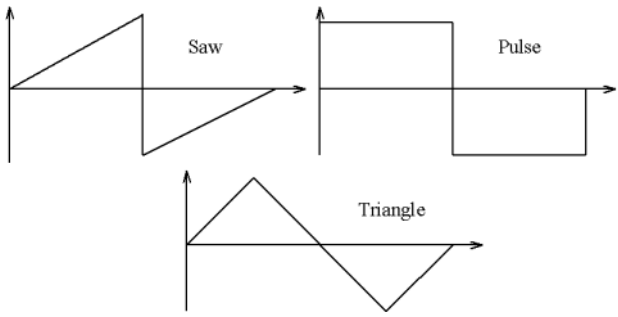


Figure 3.1: Basic waveforms

Saw and pulse waves' spectral content is very rich. For example, a saw wave with a base frequency of 500 Hz comprises harmonics spaced at equal intervals, with exponentially decaying amplitude as shown in figure 3.2. However, many physical instruments' harmonics' amplitude is lower at high frequencies. Filtering the waveform's high frequency content simulates this, conjuring a smoother, darker sound. Figure 3.3 shows the same waveform filtered at 2 kHz using a 24 dB/octave resonant low-pass filter. This resonance boosts harmonics at around 2 kHz. Beyond that, the amplitude now decays at a rate of 30 dB per octave, that is, faster than in the original spectrum.

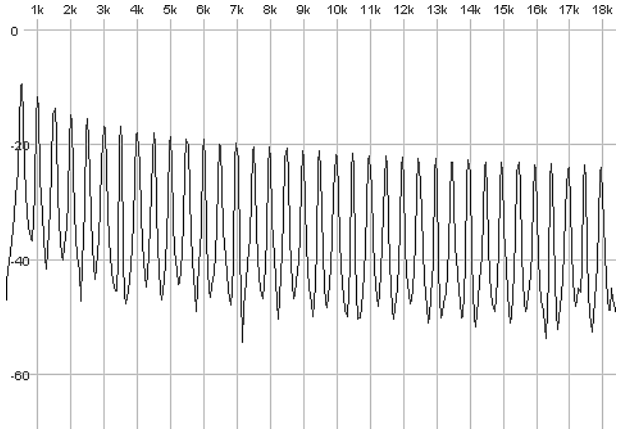


Figure 3.2: Spectral content of a 500-Hz saw wave

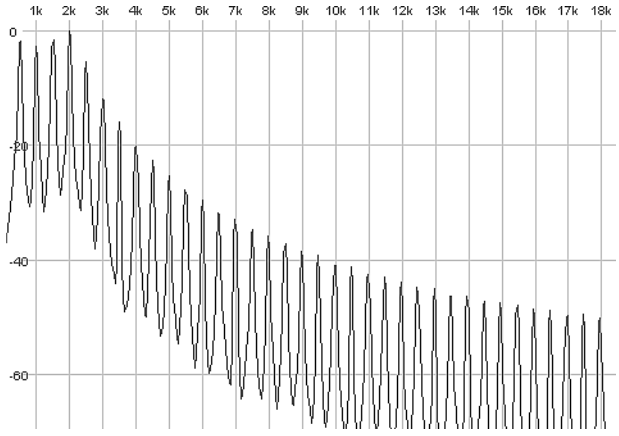


Figure 3.3: Spectral content of a 500-Hz saw wave filtered at 2 kHz

Modulating the filter's frequency with an envelope lets you vary spectral content over time, say to simulate damping. Usually, another envelope modulates the main amplitude contouring the volume curve. This, in turn, serves to mimic fast percussive instruments and slow strings and pads. Figure 3.4 shows a filtered signal's waveform and spectrogram. We applied an envelope to both the amplitude and filter cutoff frequency. You'll find the amplitude envelope in the waveform graph, and the filter envelope in the spectrogram.

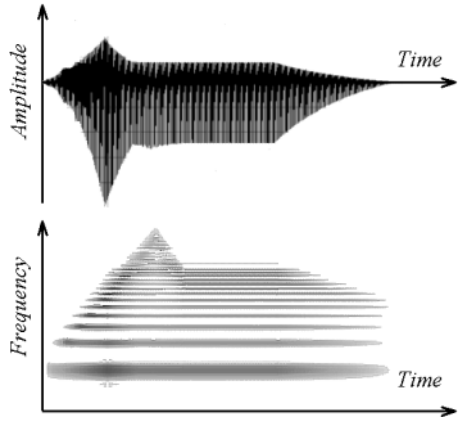


Figure 3.4: A filtered sound's waveform and spectrogram

That pretty much sums up subtractive synthesis. Most subtractive synths give you more modulation tools such as low frequency oscillators and other envelopes that vary different parameters over time. And most synthesists enrich sounds by blending several waveforms, or processing them with added effects. Versatile subtractive synths offer various filters with different characteristics and slopes that let you sculpt sounds' timbre and shape in many ways.

More on MIDI

Before we delve into how to create a subtractive synthesizer, let's look at the way instruments communicate. Arriving in the early 1980s, the MIDI (Musical Instrument Digital Interface) standard aimed to standardize communication between instruments and synthesizers. Today musicians everywhere use it for digital communication among instruments, and to control sequencers, synths and effects. MIDI messages use 16 independent channels to pipe messages to and fro. These messages include note-on, note-off, pitch bend, mod wheel, control change, program change, aftertouch, and SysEx messages. Encoded as integer numbers, they represent notes. MIDI note number 69 corresponds to A4 (440 Hz), note 70 to A#4, 71 to B4, and so on.

Though both instrument and effect plug-ins can receive MIDI data, effects lacking a MIDI input run just fine. SynthEdit offers the Plug-in is Synth option in the Save as VST panel only if the main container sports a MIDI input plug. Otherwise, it assumes the plug-in is an effect.

SynthEdit offers various modules for converting MIDI data to control voltages. Serving as a synth's main control module, the MIDI > MIDI to CV module is the king of this hill. Figure 3.5 shows a rudimentary example of a MIDI to CV module's application. Both a Keyboard (Insert > Controls > Keyboard) and a MIDI In (Insert > MIDI > MIDI In) module connect to the MIDI to CV module's MIDI In plug. This means you can use an external device like a MIDI keyboard to generate messages, and pipe them in via the default MIDI port specified in Edit > Preferences > Audio & MIDI > MIDI In.

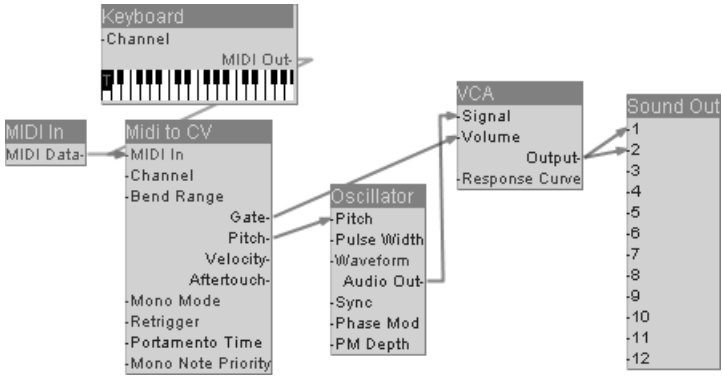


Figure 3.5: A simple example of a MIDI to CV in action

You could mouse-click the tiny keyboard on your screen to play notes. The Keyboard module also responds to keystrokes on a keyboard by producing MIDI notes. This layout resembles two rows of piano keyboards mapped to letters. Z to M and Q to P signify white keys, while S, D, G, H, J, 2, 3, 5, 6, 7, 9, and 0 represent the black keys. You'll find a chart showing how notes are mapped in figure 3.6.

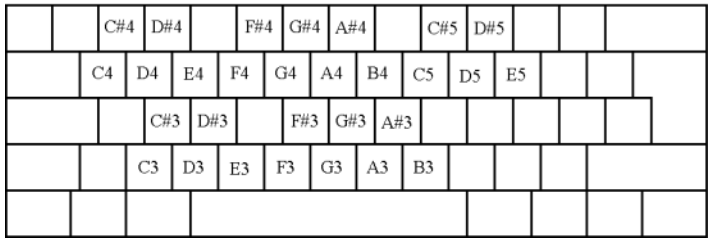


Figure 3.6: Keyboard-to-MIDI note mapping

Pressing a key elicits a saw wave with the assigned pitch. Be sure to turn your speakers down before doing this, as the saw wave's volume may be very high. The synth converts the MIDI note to pitch at 1 volt/octave. This pitch controls the oscillator's frequency. At 10 volts, the Gate's output remains high until you press any note, and returns to its low 0-volt value when you release the last key. Pressing a key triggers envelopes and other events. In our example, it controls the VCA's volume, which adjusts the oscillator's amplitude. Pressing any key triggers the signal at peak level; releasing all keys mutes the signal. This primitive on/off design is the cave dweller among synth controllers.

MIDI to CV Properties

Now let's examine the MIDI to CV module's controls.

Channel: By default, the module responds to all MIDI channels. You may limit it to one by picking a MIDI channel from Channel the selector.

Bend Range: The pitch-bend default setting is 12 semitones. Many synths offer different pitch-bend ranges, including two semitones. Specify the range here. If you connect a control to this plug and slap this feature on the GUI interface, your users can select the pitch bend wheel's bend range.

Mono Mode: This plug shoehorns the synth into mono mode to spare CPU power. Simply set it to On if you wish to conjure a monophonic synth.

Retrigger: Designed to work in mono mode, it retriggers envelopes when playing legato. Legato is a fancy term for playing another note before releasing the previous one.

Portamento Time: Use this control to create portamento effects in legato mode. Portamento means a continuous gliding movement from one tone to another. So if you hit a note before releasing its predecessor, the pitch gradually morphs over a defined time. Often called glide, you can create a similar effect by squeezing in a one-pole low-pass filter after the Pitch plug.

Mono Note Priority: Offering Off, Low, High, and Last settings, it controls the synth's response in mono mode. Play two or more notes in Low mode, and it uses the lowest pitch. Set to High mode, it uses the highest pitch. Last plays the first note again if you press and hold one key, while pressing and releasing another.

Heads up:

- MIDI to CV is territorial; only one can live in one container. Drop another MIDI to CV module into the container or a sub-container, and you will get an error message:

  You have several "MIDI to CV," "Soundfont Player," or "Drum Trigger" modules together. Put each in its own container.

- When mono mode is disabled, the Polyphony setting in the container's Properties window defines the polyphony for the MIDI to CV module in that container.

- SynthEdit converts a signal to mono as it leaves the container. This is why we mustn't place the MIDI to CV module in a separate container. If we did, the control voltages for polyphonic voices would add up, creating some very seasick off-pitch notes.

Building a Basic Polyphonic Synth

Now let's put together a basic synth structure. Figure 3.7 shows a dead-simple layout with one oscillator and one envelope in a container, and no filter. Our synth converts MIDI input to pitch and gate CV signals. The pitch signal controls the oscillator's frequency. The gate signal feeds an ADSR module (Insert > Waveform > ADSR), which responds to each incoming note by producing an envelope. This ADSR envelope controls the VCA's volume, which modulates the oscillator's sound. This processed signal goes to both output channels.

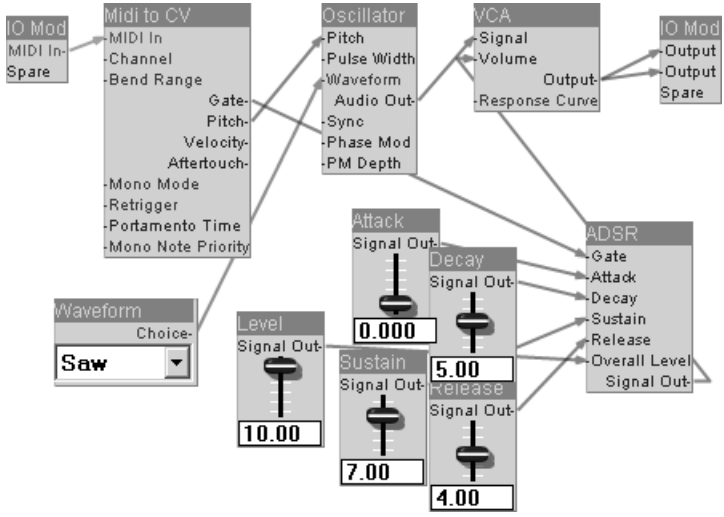


Figure 3.7: A dead-simple synth structure

To test our little rig, connect a Keyboard or MIDI in module to the container's MIDI In plug so you can play notes. Wire its outputs to a Sound Out module so you can audition the results. Figure 3.8 shows what this looks like.

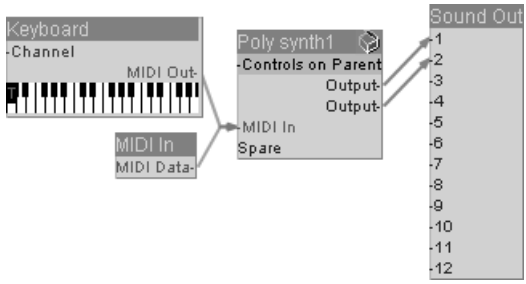


Figure 3.8: Test-driving your shiny new synth

Play several notes at the same time, and our synth will generate them. Open the oscillator's Properties window to view the number of voices and active oscillator clones. The green dots at the oscillator's top left corner point out polyphony. Figure 3.9 shows six voices, with three currently active.

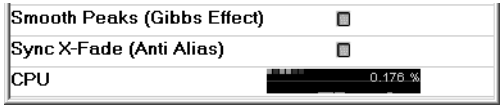


Figure 3.9: Look here to see how many voices are active

Heads up: The main container's default polyphony is six voices. To change the number, open the main container's Properties window and adjust the Polyphony value.

Go-to prefabs: Synthesis > PolySynth1

Sending Off Envelopes

Envelopes mostly serve to contour amplitude and manipulate the filter's cutoff, though they can modulate other parameters—an oscillators' pitch, an LFO's depth, and so on. The most common envelope comes with four sections, attack, decay, sustain, and release, or ADSR for short. Figure 3.10 gives you a view to an ADSR envelope's curve. The envelope's default level is 0. When the Gate plug's signal changes from 0 to a positive value, it triggers the envelope. The signal rises to the peak level as defined by the Overall Level plug. If this plug's value is negative, the envelope flips over. The shape remains the same, but the negative voltages turn it upside down.

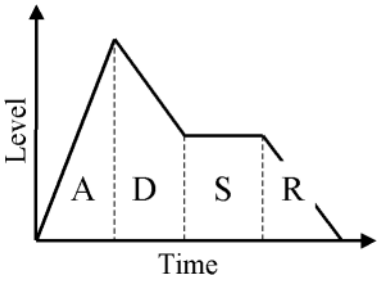


Figure 3.10: ADSR envelope

Again, the Overall Level plug defines the envelope's peak level. The attack value decides how long it takes for the signal to reach this peak. It then decays to a level specified by the Sustain plug, remaining constant until the Gate plug's voltage is high. The Decay plug determines how long this transition takes. Release the key—the gate signal's voltage dips—and the envelope's level tapers off to 0. The Release plug determines how long this takes.

Attack, decay, and release plugs' scale is exponential. The rules for conversion are:

$$\text{Time} = 2^{Volts - 6.666666}$$

$$Volts = \frac{\log(Time)}{\log(2)} + 6.6666$$

Time is specified in seconds. This means the 0-to-10 volts range equals 9.8 ms to 10.07 seconds. Negative voltages also work. Fast, percussive attacks may mandate even shorter times. But if the time is too short, pressing and releasing keys may elicit clicks. This is why some synths limit the shortest time to about 1 ms. Some filters also produce clicks at hair-trigger release times. A good, flexible scale ranges from –3.2991 (1 ms) low to 9.9885 (10 sec.) high.

David Haupt's BasicModulePak offers a free module that converts milliseconds to voltage. A voltage-to-millisecond converter module was unavailable at the time of writing, but you can easily whip one up with Waveshaper2. Figure 3.11 shows a structure that converts voltage to time. The Waveshaper's input range is –5 to +5 volts, so the input must first be multiplied by 0.5. Using $x * 2$ instead of x in the equation restores the original value.

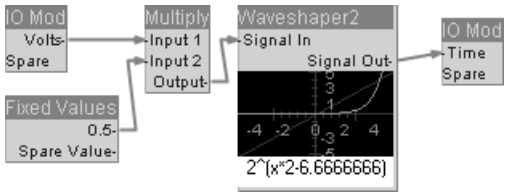


Figure 3.11: A nifty voltage-to-time converter

Go-to prefabs: Synthesis > VoltageToTime

Once you have built a voltage-to-time converter prefab, you can crown you efforts with a readout showing the value in seconds. You may recognize the structure; it leans heavily on the knobs we used in the Effects chapter. The prefab in figure 3.11 converts voltage to time. The setup below converts time to a GUI float value, sending it to a Text Entry2 module for display. Figure 3.12 shows this structure.

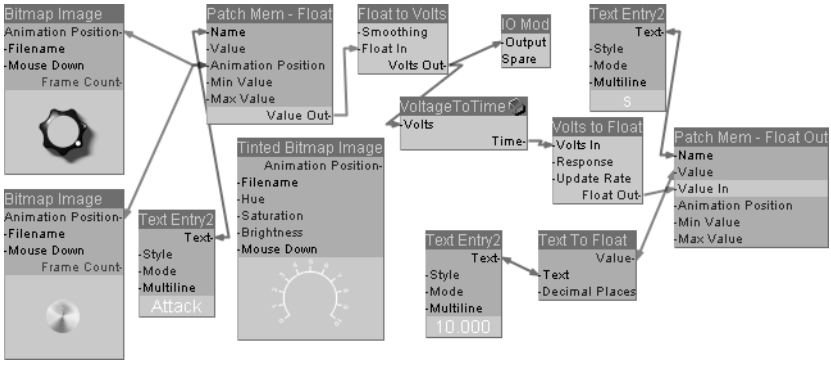


Figure 3.12: Time knob with readout

Twist this knob and the exact values appear on the GUI as in the example in figure 3.13.

Figure 3.13: VCA with readout

Go-to prefabs: Synthesis > PolySynth2

Adding Oscillators

Our first example featured an unsophisticated design sporting a lone oscillator. Most subtractive synths offer two or three oscillators for each voice to fatten up sounds. Most oscillators are de-tunable, meaning you can pitch one oscillator frequency slightly higher or lower than the other for a bubblier, chubbier sound. Many synths also offer octave or coarse tuning.

You'll find a Detuner prefab in the Insert > Controls group. Figure 3.14 shows its structure. Two List plugs are on board; one selects octaves, the other notes. The Octave plug offers –2, –1, 0, 1, and 2 volts. If you open the fixed values for Note, you will see 0, 0.0833333333, 0.166666666666, 0.25, and so on. Let's see what these numbers are all about: Oscillators usually work with one volt per octave. This means adding one volt to a pitch detunes the note by one octave. Say you wish to detune a pitch by one semitone. An octave comprises 12 notes. To detune a pitch by one semitone, you must add 1/12 to the original pitch, or roughly 0.083333333 volts. Add 2/12, or about 0.1666666 volts, to detune by two semitones; 3/12 or 0.25 volts for three steps, 4/12 or 0.3333333 volts for four, and so on. There you have the Fixed Values' decimal numbers' origins.

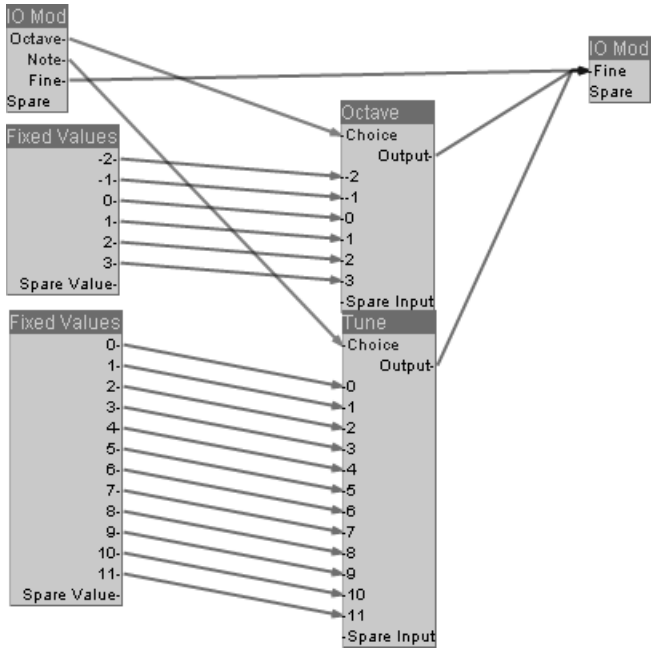


Figure 3.14: The Detuner prefab

In this prefab, you'll find a knob wired to the Fine plug. Its values run from 0 to 0.08333333333, or 0 to 1 semitones for fine-tuning. You can extend the control range from –0.08333333333 to 0.08333333333, or –1 to 1 semitone.

Now feast your eyes on a structure with two oscillators, each featuring detuning controls, in figure 3.15. A detune prefab connects to each of the oscillator's Pitch plugs. This lets users tune the two oscillators individually and relative to the base pitch provided by the MIDI to CV module. We added two knobs for adjusting the oscillators' level. The Level Adj modules send these signals to the VCA's Signal plug, where they merge. You're familiar with rest of the structure (the ADSR), so we spared you a diagram. Though this structure still lacks a filter, you can conjure some simple lead and pad sounds. Play a couple of notes and fiddle with the detuning and envelope settings to see what you get.

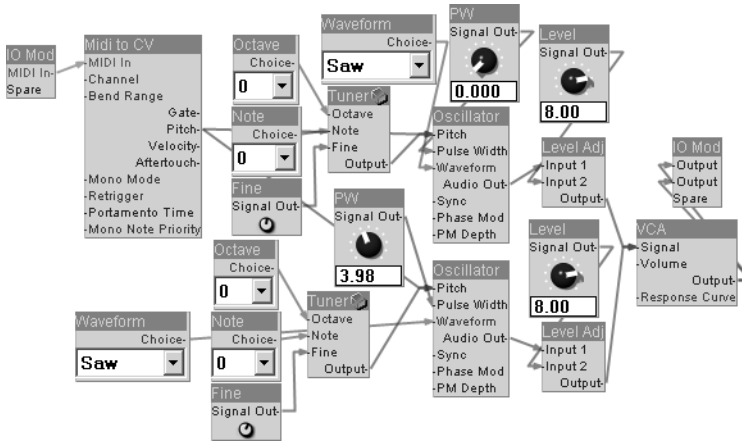


Figure 3.15: Two oscillators with detuning controls

Go-to prefabs: Synthesis > PolySynth3

Pulse Width

Note that we added two knobs for adjusting pulse width, although they only shape the waveform when a pulse waveform is selected. Telecom and electronics engineers call this the pulse wave's duty cycle. Figure 3.16 shows pulse waves at different pulse width settings.

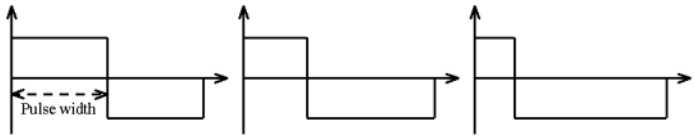


Figure 3.16: Pulse waveforms with varying pulse width

Different pulse widths' spectral content varies, lending the wave different characteristics. Subtractive synths often use LFOs to modulate pulse width, achieving a rich flavor similar to the sound of two detuned oscillators.

In SynthEdit, 0 volts yields a symmetrical waveform. As you jack up the pulse width value, the waveform grows increasingly asymmetrical. Approaching 10 volts, it's more of a short spike. Feel free to use a Scope2 module—its oscilloscope analyzes signals—from the Insert > Controls menu to check the waveform's shape.

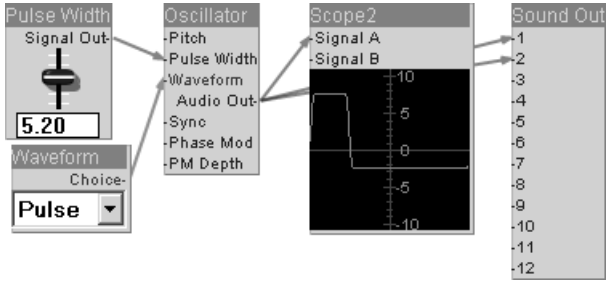


Figure 3.17: Analyzing waveforms with a Scope2 module

More on Waveforms

The oscillator offers sine, saw, ramp, triangle, pulse, white noise, and pink noise waveforms. A ramp sounds much like a saw wave because they share the same shape, though the ramp's is inverted. Used as an LFO, it provides falling rather than rising voltage.

The two flavors of noise are white and pink. White noise's power spectral density is flat, yielding a sharp, bright sound. It serves to create percussive instruments such as snare drums and hi-hats, and spectral effects. Pink noise's power density decays at –3 dB per octave for a softer, darker sound.

Two noise oscillators are usually one too many. Apart from noise type, there are generally no other parameters to tweak for a noise generator. You can build a more versatile synth by confining the oscillators' waveforms to sine, triangle, saw, and pulse, and adding a separate noise oscillator with a dedicated level knob. Figure 3.18 shows how this works.

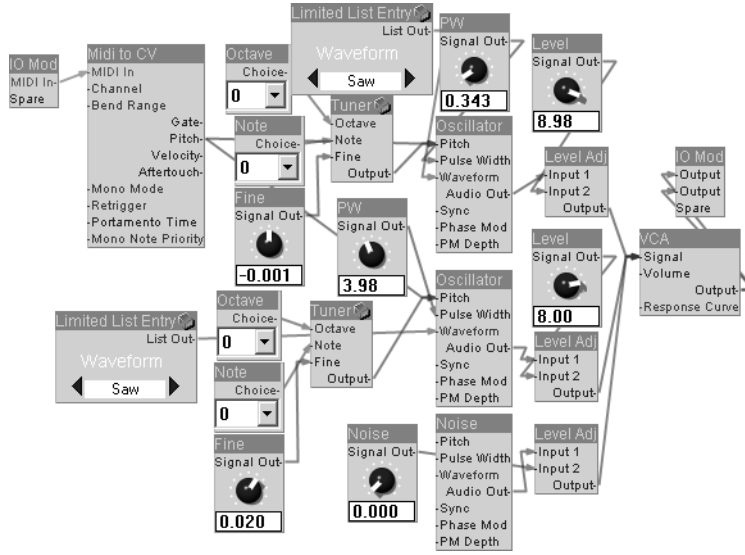


Figure 3.18: Adding a separate noise oscillator

Aptly named Noise, the third oscillator's waveform is now a white noise generator. It features a dedicated Level Adj module sporting a Noise knob. If the Limited List Entry prefab that selects the two main oscillators' waveform looks familiar, you may remember it from the "Effects" chapter. Its list offers sine, triangle, saw, and pulse waves. To see the full setup, give figure 3.19 a gander. The List to Bools module's Spare plug connects to the Bools to List module's Sine, Triangle, Saw, and Pulse plugs in that order to create the list.

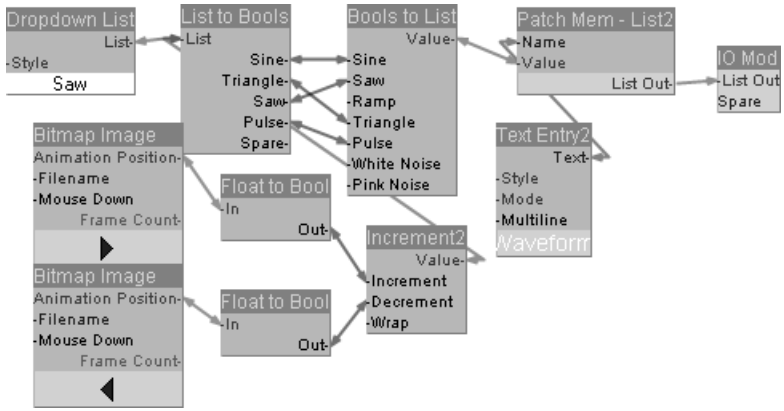


Figure 3.19: Limiting waveforms to sine, triangle, saw, and pulse looks like this

Go-to prefabs: Synthesis > PolySynth4

Get Smooth with the Gibbs Effect

Open an oscillator's Properties window, and you will see an advanced option called Smooth Peaks (Gibbs Effect). Now compare the two oscillators' waveform in a scope, and you will discover they are different.

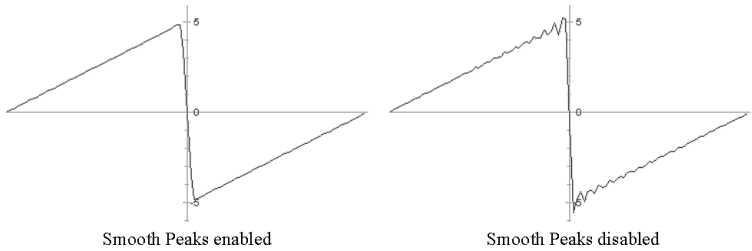


Figure 3.20: The Smooth Peaks option

Enable Smooth Peaks and the waveform will resemble a saw wave, more or less. Disable it, and a big ripple appears at the edge. How come? Put simply, the waveform is band-limited, meaning that its frequencies range no further than from 0 Hz to half the sampling rate. The Fourier series tells us summing an infinite number of sine waves creates a band-limited saw wave. Summing a limited number of sine waves creates what scientific types call the Gibbs effect; we'd call it rip-

ples at the edges. The same happens to other waveforms with sharp edges—ramp and pulse come to mind. So, this rippling waveform is a fixture in the digital domain. Smooth Peaks curtails the effect, but also diminishes high frequency content above 4 kHz, as a look at the frequency analyzers in figure 3.21 will confirm.

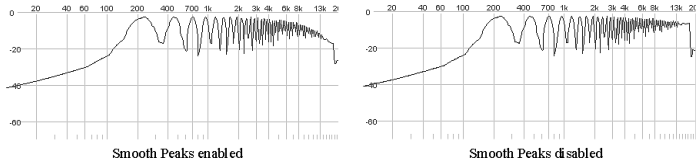


Figure 3.21: Smooth Peaks' high-frequency damping

Enabling Smooth Peaks is a good idea for LFOs; otherwise the Gibbs effect may introduce weird artifacts near the waveform's edges. If you prefer brighter-sounding oscillators, disable this option.

Sizing Up Filters

A subtractive synth without a filter is an exercise in blandness—a curry dish without spices. Filters lend sounds their flavor by boosting and cutting the oscillators' frequencies. So, let's first review the different breeds of filter.

The State Variable Filter

SynthEdit offers two breeds of filters suitable for use with subtractive synths. One is the SV Filter, a two-pole state variable—or SV, for short—filter with resonance. It works in low-pass, high-pass, band-pass, and band-reject (also called notch and band-stop) modes. And all at the same time, if you wish. Low-pass and high-pass modes' slopes are 12 dB per octave slopes; band-pass mode's is 6 dB per octave. Figure 3.22 graphs a few of the different modes' transfer curves.

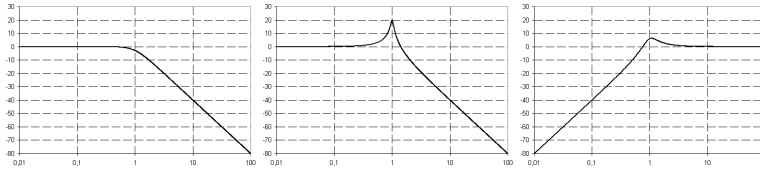


Figure 3.22: A small selection of SV filter transfer curves (LP and HP mode)

Selecting Type

A simple method of selecting the filter type is using a Many → 1 switch. Look no further than figure 3.23 for an example. But simplest is not always smartest. You'll find a 1 → Many switch is a more efficient tool. See the Optimization chapter's SV Filter section to learn more.

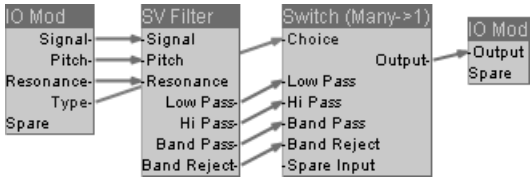


Figure 3.23: Selecting the mode

Resonance Levels

Try adjusting an SV Filter's pitch and resonance parameters and you may discover the filter's resonance spikes somewhere around 10 volts. This is rarely desirable, and may cause clipping. Also, the resonant response is nonlinear in the dB scale, as a glance at figure 3.24 attests. Around 10 volts, the resonance value skyrockets close to the point of self-oscillation. To prevent this, you may want to confine the resonance knob's highest value to about 9.8 volts, or roughly 30 dB resonance.

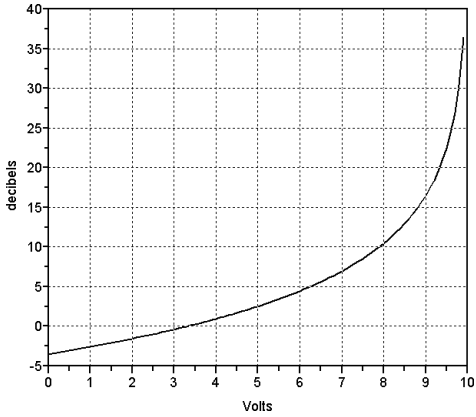


Figure 3.24: An SV filter's resonance levels

Heads up: Scoofster SVF, a third-party SV filter, uses the more convenient decibel scale for adjusting resonance. This filter also spares CPU power in some modes.

Cascading More Filter Stages

Cascading two SV filters in series creates a steeper low-pass filter. To do this, wire the first filter's low-pass output to the second filter's Signal plug. When pitch settings are identical, this structure behaves like a four-pole 24 dB/oct. low-pass filter. Figure 3.23 shows an example of such a structure. Note that the second filter's resonance value is two, which is low indeed. This means only the first filter adds resonance to the sound.

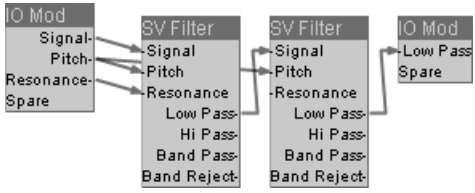


Figure 3.25: Do this to cascade two two-pole SV filters

Heads up: Cascading many band-pass or high-pass outputs is not a good idea. It may elicit annoying high-frequency ringing because state variable filters boost high frequencies with high cutoffs even at low resonance settings. You do have a tradeoff option, though. The third-party sc:SVF gives you less high-frequency ringing at the expense of a more limited frequency range.

Normalizing Output Level

High resonance levels can kick the output level up a few notches, causing clipping or huge inconsistencies between output levels. Some filters let you normalize levels to prevent this problem. This handy function attenuates the signal level as the resonance level rises. Sadly, SV Filter lacks such a feature. Happily, it is easily added. See figure 3.26 for an example. A Level Adj module sited in front of the filter adjusts the incoming signal. 10 volts equals 100 %; lower values attenuate the signal accordingly. A Waveshaper2 module calculates a value between 10 and 4 volts, depending on what the Resonance plug is up to. Resonance ranges from 0 to 10 volts. Subtracting 5 volts yields a –5 to 5 volts range. Using $x + 5$ in the equation in lieu of x yields the original resonance value. The resonance level, multiplied by 0.6, is subtracted from 10, resulting in a descending slope. Waveshaper2 does not do any dampening at 0 volts resonance. At 10 volts, it attenuates the signal by 60 %. If you want more or less aggressive normalizing, simply change the 0.6 in the equation. Higher values boost the normalizing factor.

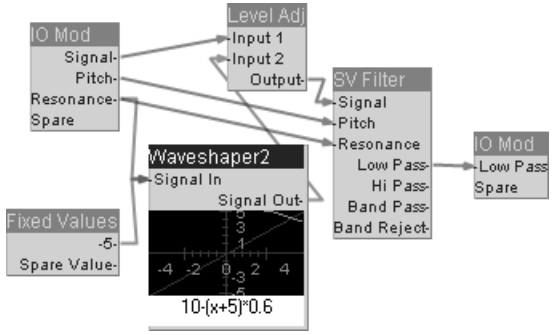


Figure 3.26: Normalizing output level

Heads up: Normalizing applies to low-pass, high-pass, and band-pass modes only. You could normalize the signal in band-reject mode, but to little effect. Its output levels are low enough as it is.

Go-to prefabs: Synthesis > SV Filter Norm

Mixing Outputs

Some synths let users mix a state variable filter's low-pass, band-pass, and high-pass outputs, which beats having to settle for just one type. Users may then merge various filter outputs and mix their levels to conjure unique sounds. The only drawback in SynthEdit is the more outputs you connect, the slower SV Filter runs. Besides, the Level Adj modules also devour CPU resources. Simply using low-pass, band-pass, high-pass and band-reject modes is more efficient but less fun. Figure 3.27 shows an example prefab.

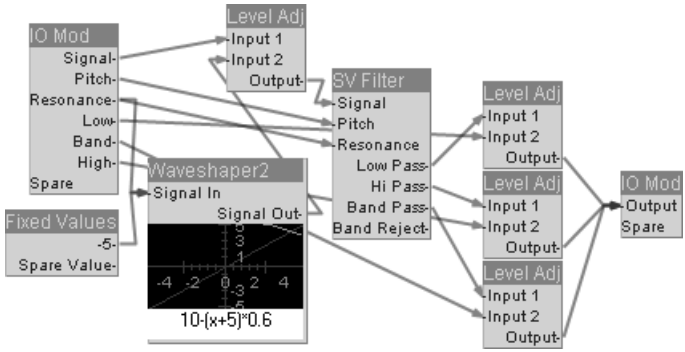


Figure 3.27: Mixing low-pass, band-pass, and high-pass outputs

Based on the preceding prefab, this version first normalizes the output level to compensate for resonance gain. The Low, Band, and High plugs patch the low-, band-, and high-pass signals out. The Level Adj modules tweak these signals' levels and feed them to the IO Mod, which blends the batch. Users may dial in unique filter characteristics by varying these levels. The three outputs change the signal's phase in different ways, sometimes creating notches. Mixing Low Pass and Hi Pass outputs creates a band-reject filter. Negative voltages are good to go, so entering something like 10 and –10 volts for the low and high values compels the prefab to subtract the high-pass from the low-pass output. This evokes a flat response with a resonant peak at the cutoff frequency, great for adding resonant peaks to a signal.

Heads up: State variable filters are all-pole filters, meaning they boost high frequencies even when the resonance value is low. This affects band-pass and high-pass outputs more than the low-pass output. And mixing these outputs adds a touch more gain.

The Moog Filter

A digital emulation of Moog's celebrated four-pole transistor ladder filter, the Moog Filter is a has a 24 dB/octave slope. Its internal saturation circuit simulates analog components' nonlinearities. Responding differently to different input levels, it self-oscillates when resonance is cranked. It behaves very differently to an SV filter at high resonance levels. The Moog Filter features enhanced resonance levels for high frequencies, adding high-end gloss, and moderate resonance for low frequencies. At extreme resonance settings, the saturation circuit may cause top-end aliasing.

On the third-party front, Marc Lindahl's Moog VCF Ladder Filter is an emulation sharing the same code as the Moog filter found in Steinberg Model E. Rick Jelliffe's RJ LP Filter 2 and 3 offer various saturation modes, characteristics, and even high-pass versions.

Biquad Filters

Although SynthEdit version v1.0150 lacks biquad filters, they are commonly used in digital filter design. Biquad is short for biquadratic, a second order filter comprising two poles and two zeros. Many third-party filters feature this topology, including DH_BiquadFilter, DH_MultiFilter2, and EVM LP Filter. Scoofster Low-pass also uses a modified version of a biquad structure.

A biquad filter's transfer function is much like a state variable's. What sets the former apart is that it contains so-called zeros. They eliminate the high-frequency ringing state variable filters are so notorious for. Biquad filters' high-frequency performance is excellent, highly stable all the way up to Nyquist. Most are faster than state variables, and much faster than Moog filters. On the downside, a biquad filter's low-frequency performance is not exactly a model of stability. At high resonance levels, low frequencies tend to destabilize with excessive gain. This can boost levels to extremes and cause clipping, especially during fast filter modulations. Here's a quick-fix: The structure in figure 3.28 adjusts the resonance levels for low cutoff frequencies to stabilize performance. The curve approaches 0 at around –5 volts, though the true value is 0 volts because the Waveshaper's input is scaled to –5 volts. This attenuates resonance for low frequencies. To adjust the amount, simply replace the 0.1 in the exponent with another value.

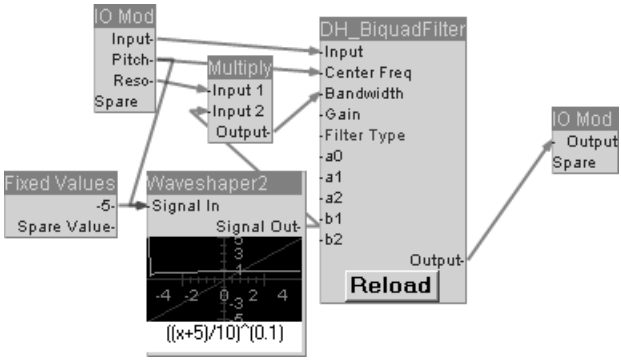


Figure 3.28: Here's how to stabilize biquad filters' low-frequency performance

Go-to prefabs: Synthesis > Biquad Stable

Biquad filters' other drawback is the amount of calculation it takes to modulate the filter. A biquad filter expends far more processing power than a variable state filter. Some filters calculate less to save more. For example, DH_MultiFilter2 calculates filter coefficients merely for every fourth sample to improve performance. Bear in mind, though, that this invites aliasing noise and unwanted artifacts when modulating the cutoff frequency at speeds approaching the audio rate.

How Different Filter Types Compare

Table 3.1 summarizes the three most common filters' strengths and weaknesses. If your filter is destined for modulation at the audio rate, then an SV Filter may well be a better choice than a biquad because it modulates without consuming extra CPU power. Its low-end performance remains stable under all conditions. If high-frequency performance is more important to you, or you wish to cascade multiple stages, then a biquad may be your better bet because it rules out high-frequency ringing. Both filters offer low-pass, band-pass, high-pass, and band-reject filters, though you may find their flavor a touch too digital compared to analog filter emulations. Users value Moog filters for their distinct sound and timbre. Their internal saturation stages need more calculations, so they impose a bigger burden on the processor. Most offer low-pass mode only; some offer high. On the upside, you can count on their modulation and resonance stablility.

	SV Filter	*Moog*	*Biquad*
Speed	++	+	+++
Modulation	+++	+++	+
Highs	+	++	+++
Lows	+++	++	+
Types	+++	+	+++
Resonance	++	++	++
Sound	digital	analog	digital

+ = Weak, ++ = Medium, +++ = Good

Table 3.1: Comparison of different filter types

Slapping a Filter on a Synth

Now that we know all about the different filters' pros and cons, we'll learn how to add a filter to the synth structure. Conventional wisdom dictates placing the filter before the VCA. You could drop it in after the VCA, but then if the filter self-oscillates, the VCA will not mute it as the notes fade out. Figure 3.29 shows a schematic diagram of a basic dual-oscillator synth with a filter placed before the VCA. The MIDI to CV

module controls the two oscillators and the ADSR. The subsequent stage adjusts the noise generator and oscillators' levels, mixes these signals, and feeds them to the filter. The VCA applies a volume envelope before the signal leaves the synth.

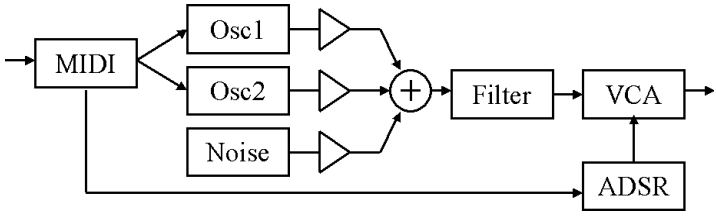


Figure 3.29: Schematic diagram of a two oscillator synth with filter

Before we build this structure, let's streamline the synth we created earlier. First we'll introduce a simple Mix prefab for mixing the oscillators' outputs. This is nothing more exciting than an empty container with an input and output as shown in figure 3.30. If you patch several signals into the In plug, the prefab mixes them and routes the composite through to the Out. This prefab may seem superfluous now, but it will make wiring easier later. Before you can make this connection, you must first wire up the IO Mod's Spare plug. Then connect the input to the output, and delete the other wires.

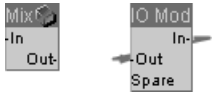


Figure 3.30: A simple Mix prefab's module and structure

Go-to prefabs: Synthesis > Mix

Dropping the oscillators with the Tuners and Level Adj modules into a separate container makes the structure easier to work with. To do this, press and hold Shift, and select the Tuner, Oscillator, and Level Adj modules for an oscillator via mouse-click. Then execute the Edit menu's Containerise Selection command. You may have to change labels or rearrange the plugs' order. If you like the results, repeat the procedure for the second and third oscillators. Feel free to reference figure 3.31; it shows the final structure without the VCA envelope. With all the components apart from GUI features tidied up in containers, you'll find the going much easier.

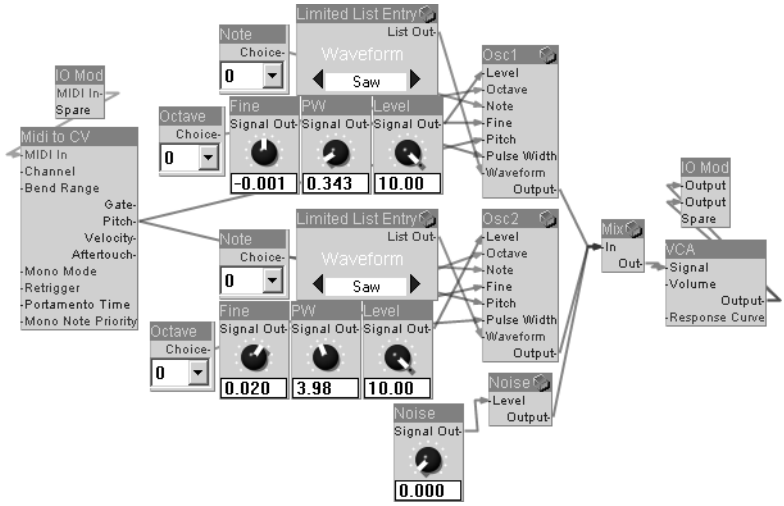


Figure 3.31: Make mom proud and put those oscillators away in containers

Adding a No-frills Filter

Adding a filter to a tidy structure is a piece of cake. Simply site the filter between the mixer and VCA as shown in figure 3.32. This setup uses the same VCA envelope as the previous prefabs, so we'll spare you the details. Our example bases on a normalized multimode state variable filter similar to the prefabs in the Sizing Up Filters section. Figure 3.33 x-rays the SV Filter prefab's internal structure.

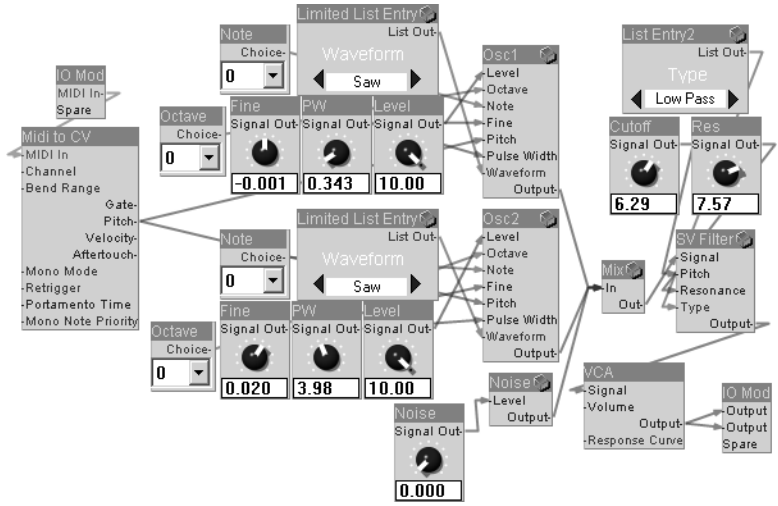


Figure 3.32: Adding a filter

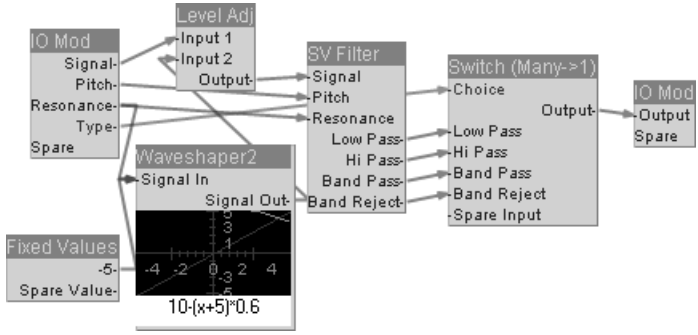


Figure 3.33: A map of the SV Filter prefab

Go-to prefabs: Synthesis > PolySynth6

Adding a Filter Envelope

The only option for adjusting the filter's pitch in the previous prefab was the Cutoff knob. Most synths feature a dedicated envelope, usually an ADSR, serving to modulate the filter's cutoff and create filter sweeps. Figure 3.34 shows you a schematic diagram.

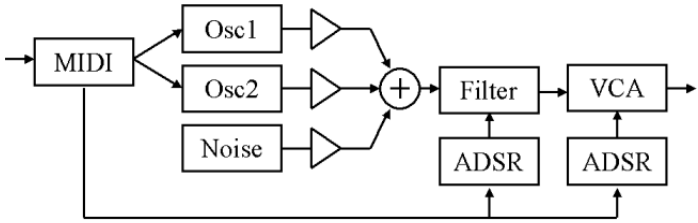


Figure 3.34: Schematic diagram of a filter envelope with keyboard tracking

To add a filter envelope, select the VCA ADSR with its control knobs, copy the whole kit, and paste it into the picture. Connect the MIDI to CV module's Gate plug to the newly created ADSR's Gate plug, and the envelope's Signal Out plug to the SV Filter's Pitch plug. Now every note triggers an envelope that the synth applies to the filter's cutoff frequency, thereby modulating it. Figure 3.35 shows the resulting envelope section.

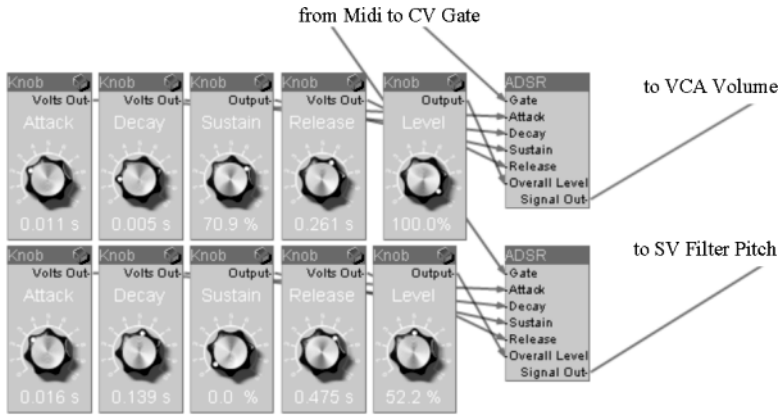


Figure 3.35: An Envelope section with dedicated volume and filter envelopes

Go-to prefabs: Synthesis > PolySynth7

Negative Envelopes

The Overall Level plug's value adjusts the amount of modulation applied to the cutoff, with one volt equaling one octave. Some synths also offer negative modulation. If you enter a negative voltage to the Overall Level plug, the envelope flips as shown in figure 3.36. You have two options for adding positive and negative envelopes. One is to set the Level knob's control range to –10 to 10 volts. The 12 o'clock position is passive, meaning that it prompts no modulation. Twisting the knob clockwise dials in a positive envelope; counterclockwise dials in a negative envelope.

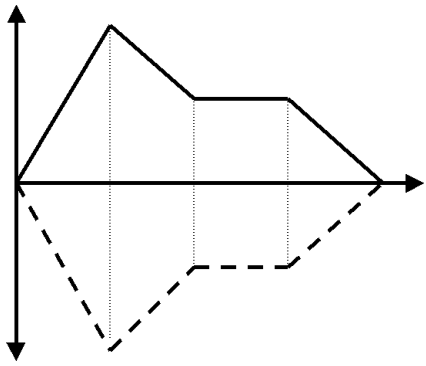


Figure 3.36: An inverted envelope

Here's the other method for adding negative envelopes: Stake out a range of 0 to 10 volts, and add a 1→Many switch with an Inverted module to one of the chains. Its setting determines if the knob's value is inverted, with a neutral or zero setting precluding modulation in both modes. Figure 3.37 gives an example of this structure. Setting the Invert switch to On flips the envelope over.

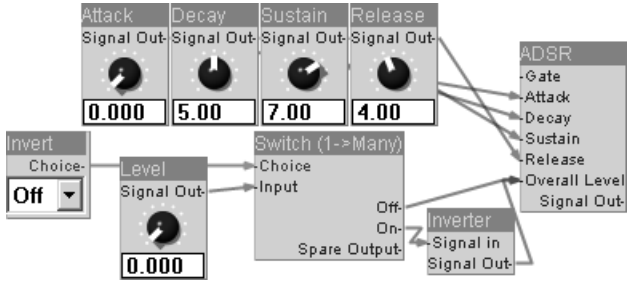


Figure 3.37: Inverting an envelope

Go-to prefabs: Synthesis > ADSR Invert

Creating an Exponential Envelope

The ADSR module automatically creates a linear envelope, that is, voltage rises and falls along a straight trajectory. This may sound artificial because the level drops to 0 fairly abruptly. You have many options for transforming an envelope from linear to exponential, and smoothing out slopes a touch. One is to place a VCA after the ADSR and set its Response Curve to Exponential or Decibel. Another is to use a Waveshaper2 and customize its transfer curve. Perhaps the simplest is to set the ADSR's Overall Level to 10 volts, and connect the Signal Out plug to Level Adj module's two input plugs as shown in figure 3.38.

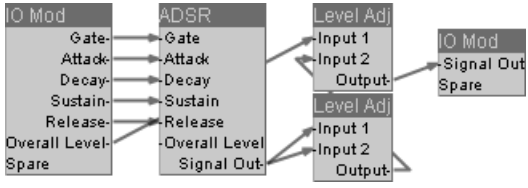


Figure 3.38: Fake an exponential envelope with this prefab

This creates a mock exponential envelope with smoother curves. Placing another Level Adj module after the first lets users adjust the envelope's level. Figure 3.39 illustrates the resulting envelope curve. The Level Adj module multiplies the two inputs and divides the result by 10, thereby reducing sustain levels. Case in point: If the Sustain plug's setting is 5 volts, the sustain level will be $5 \times 5/10 = 2.5$ volts.

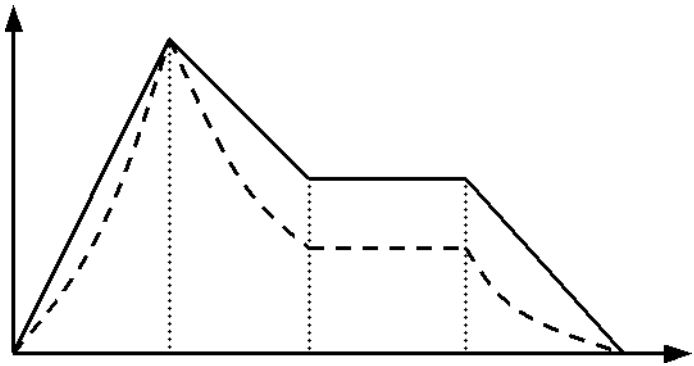


Figure 3.39: A linear vs a mock exponential envelope

Go-to prefabs:
Synthesis > ADSR Exp
Synthesis > PolySynth8

Heads up: David Haupt's DH_EnvSeg module lets you create envelopes with any number of sections. It also lets you specify the response curve's individual sections. For more on this, see the module's documentation.

Adding Keyboard Tracking

Many synths offer a keyboard tracking function for the filter, and yours can too. What happens here is when the user enables key-tracking, the note's pitch affects the filter's cutoff frequency. Say the key-tracking value is one. If the user plays a note an octave higher, the filter's cutoff also climbs by an octave. You'll find a schematic diagram in figure 3.40.

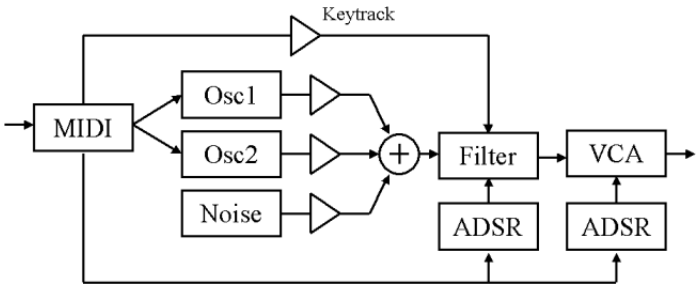


Figure 3.40: Key-tracking

A simple way of adding keyboard tracking is to wire the MIDI to CV module's Pitch plug to the filter's pitch. Drop a Multiply module into the signal chain to adjust the key-tracking amount. Multiplying the pitch by values ranging from 0 to one changes the keyboard tracking value. The problem is that this limits the filter's cutoff range. Say a user plays the middle A note. The pitch is 5 volts, which the synth adds to the filter's pitch. If the Cutoff knob's range is 0 to 10 volts, then adding the pitch shifts the range from 5 to 15 volts, with the lowest cutoff being 440 Hz. Figure 3.41 shows a prefab which compensates for this by dividing five times the amount of key-tracking from the detune amount. This ensures the middle A remains fixed regardless of the key-

tracking value. You may wish to extend the cutoff range beyond 10 volts so key-tracking does not unduly affect the highest frequency when playing low notes. You'll find a streamlined structure depicting how to connect the Keytrack prefab to the filter in figure 3.42.

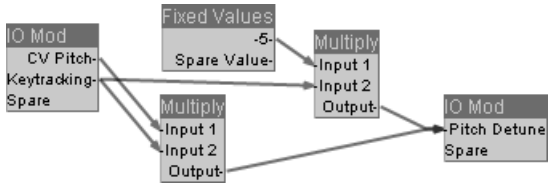


Figure 3.41: Key-tracking prefab

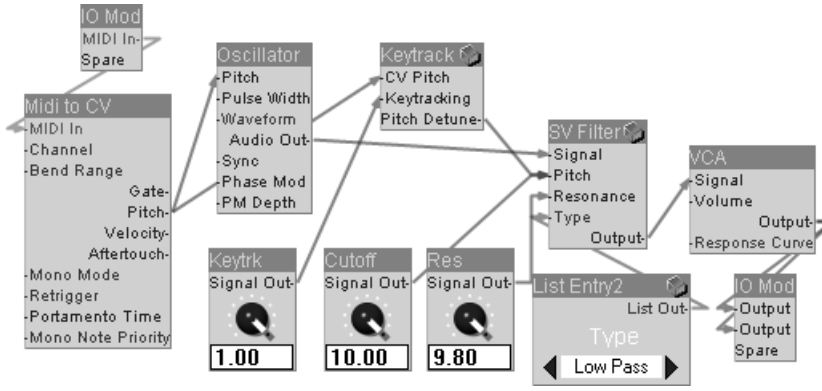


Figure 3.42: Connecting the Keytrack prefab

Heads up: Key-tracking values normally range from 0 to 1. Bear in mind, though, that negative voltages are permissible, so higher notes will lower the cutoff.

Go-to prefabs:
Synthesis > Keytrack
Synthesis > PolySynth9

More About Filters

The previous sections explained how to create a basic multimode filter with a dedicated envelope and keyboard tracking. Many advanced software and hardware synths feature two separately adjustable filter sections. They let you route filters in parallel or in series; often with a

switch that changes the routing mode. And they let you do things like use a stereo filter to create big, bold sounds by panning the oscillators sited in front this filter. Frequently a saturation stage follows the filter, spicing it up with unique sonic flavors. Be sure to bear aliasing in mind when you give users the tools to saturate or distort signals.

Modulation

A subtractive synth without proper modulation possibilities is a car without tires—it'll go, but not very far. Most synths feature one or more low frequency oscillators. Some boast added envelopes and various MIDI modulation sources, such as mod wheel, velocity, aftertouch, and control change messages. You can usually route these to filter cutoff, pulse width or oscillator pitch, filter resonance, modulation depth, and sometimes to other destinations. A 1 → Many switch can serve to select the modulation sources' destination, but a modulation matrix is more convenient for mapping modulation sources. A mod matrix lets you choose sources and destinations, and adjust amounts. Some mod matrixes offer several destinations for each source with a dedicated amount knob for each modulation routing. So let's look at these modulation sources.

LFOs

Synths' and effects' low-frequency oscillators are much the same. Most synths' LFOs offer frequencies ranging from 0.01 to 30 Hz. Some offer Sample and Hold so that pressing a key retriggers the LFO. This lets users do stuff like trigger a random constant voltage at every note-on message using a noise waveform. If you want two LFOs, simply add two oscillator modules. Enable Smooth Peaks in the Properties window—it is the default, but check to confirm this. This option prevents ripples at the waveform edges. Connect a List selector to each Waveform selector. Set Pulse Width to 0 to conjure a symmetrical pulse wave, or insert a dedicated pulse width knob for each LFO. The Rate knobs used in this synth are the same as those in the Effects chapter, so we'll spare you another structural diagram. The only difference is that these knobs' high value is 30 Hz (1.1255 volts). The modulation circuit's level adjustment determines modulation depth, as you shall soon see.

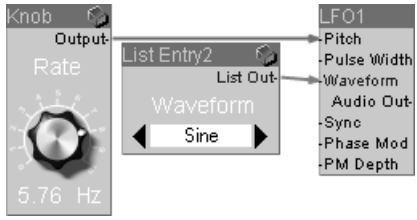


Figure 3.43: Low-frequency oscillators

Heads up:

- By default, an LFO is monophonic unless a polyphonic signal arrives at an input. If you wish to assign an LFO for each voice, you could connect the MIDI to CV module's Gate plug to the oscillator's PM Depth plug and set the Phase Mod plug to 0. This does not affect the LFO waveform, but it does coerce the LFO into doing the polyphonic thing. Be aware that this polyphonic capability takes a bite out of the CPU pie even when the LFO isn't modulating.
- If you wish to retrigger the LFO's waveform at every note, patch the MIDI to CV module's Gate signal to the oscillator's Sync plug. This is another case of coerced polyphony. If you prefer a monophonic LFO, drop a Voice Combiner module from the Insert > Special group in front of the Sync plug. It squeezes the signal into mono format.
- Combining an LFO and a Sample and Hold module can conjure interesting effects. Try this on for size: Insert a Sample and Hold module after the LFO and connect the Gate signal to the S&H module's Hold plug. Then it retriggers only when you hit a note. The LFO puts out a polyphonic signal. If you prefer a monophonic signal, use a Voice Combiner module to convert the Gate signal. Use an S&H module on a noise waveform to trigger random constant voltages in response to every note-on message. Figure 3.44 pictures a monophonic S&H LFO structure.

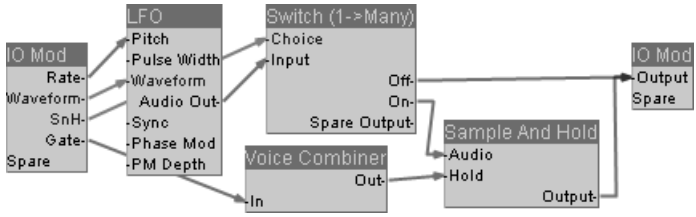


Figure 3.44: A monophonic S&H LFO

Some synths offer a delay parameter for the LFO to determine the time the LFO modulation takes to reach peak level from 0. You can do this too by applying a separate envelope, triggered by the Gate signal, to the LFO that controls signal level. Keep an eye on those polyphonic signals, though.

Go-to prefabs: Synthesis > SnH LFO

Envelopes

A little structural modification is all it takes to use the filter envelope as another modulation source. Figure 3.45 shows a modified version of the mock exponential envelope. The EnvMod plug controls the envelope's level, and sends the signal to the Mod Out plug. It modulates the filter's pitch much like in the unmodified version. The difference here is that the Env Out plug takes its source from the full level envelope, before the level adjustment. This means it can serve as a modulation source.

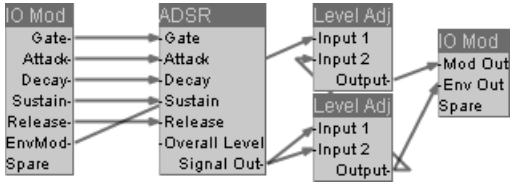


Figure 3.45: A souped-up filter envelope

Adding more envelopes is easy; simply connect the Gate signal to an ADSR module so every note attack triggers an envelope. Your users and their CPUs may rejoice if you insert a Level Adj module before the ADSR's Gate plug with a Switch wired to its other input. This lets users switch unused envelopes off to save CPU power.

MIDI Messages

MIDI commands come in many guises. Called velocity, mod wheel, pitch bend, control change, and aftertouch messages, all serve as modulation signals. The MIDI to CV module delivers velocity and aftertouch values direct. SynthEdit's MIDI folder offers a Controller module that affords access to the others. It sports default outputs for aftertouch and pitchbend, as well as four user-definable outputs for MIDI CC (control change) messages. An individual Type selector for each lists

available control change messages. This list also describes the typical functions of some CC messages, like 1—Mod Wheel, 2—Breath, 7—Volume, and so on. You can set these to a fixed CC number, or let your users freely select the number of CC messages used as modulation sources.

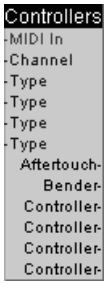


Figure 3.46

Making a Modulation Matrix

Featuring source and destination selectors, a modulation matrix maps one or more routes. The schematic diagram in figure 3.47 shows one mod matrix route. It adjusts the selected source's level according to modulation depth, and routes the signal to the specified destination.

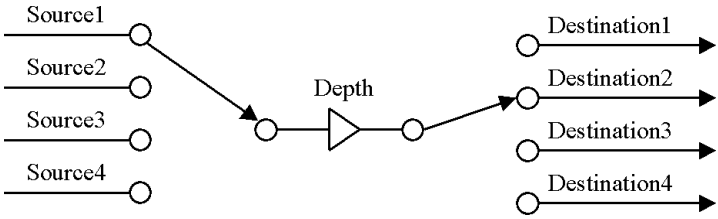


Figure 3.47: Schematic diagram of mod matrix signal routing

Figure 3.48 shows how this diagram translates to a SynthEdit structure. The prefab maps two modulation routes. The first route's sources are None, LFO1, Filter Env, Velocity, Aftertouch, and Modwheel. The second's are the same, though LFO2 is used rather than LFO1. These sources were connected to the Spare Input plug of the switch that selects the modulation source. The signal runs from the switch to a Level Adj module, with the Depth knob adjusting its level. It then feeds a 1→Many switch that selects the modulation destination. To add des-

tinations, simply connect the Spare Output plug to the desired destination plugs. They will then appear in the list. The Pulse Width and Pitch plugs connect to the Pulse Width and Pitch plugs of both oscillators, meaning that a source can be routed to two or more destination plugs at the same time. If you wish to rename the source and destination slots, open the switches' Properties window and change the labels.

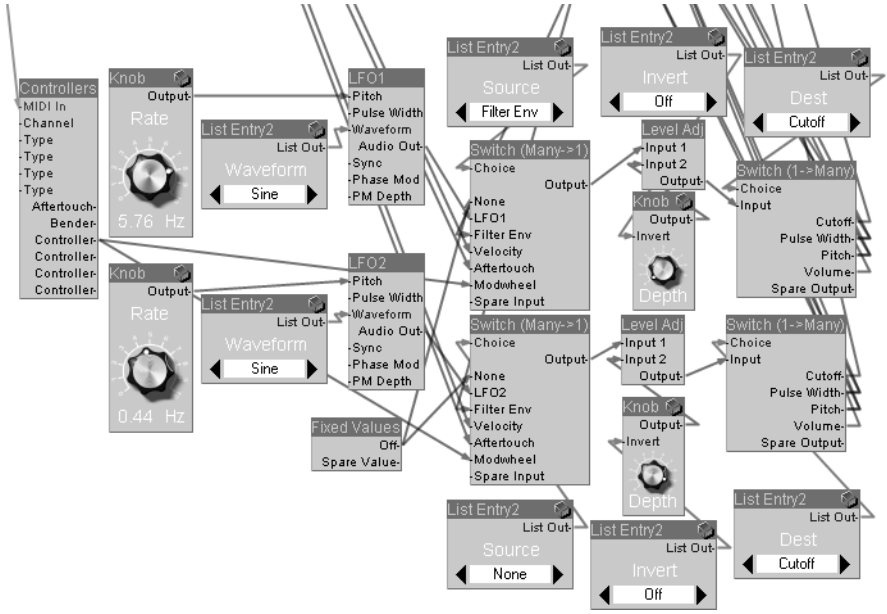


Figure 3.48: The modulation matrix's structure

The Depth knob bases on the Knob Sm prefab in the Controls folder. Figure 3.49 delineates its structure. The knob's range is 0 to 10 volts. Converted to volts, the signal feeds both inputs of a Level Adj module. Much like the Level Adj used in the envelope, it transforms the linear scale into a mock exponential scale for greater tweaking precision, particularly for low values. Next comes a switch enabling users to invert the signal. Now, what is that good for? Say the envelope addresses the oscillators' pitch. By default, the modulation invokes a positive pitch change. The Invert switch lets your users select a negative envelope. Of course, assigning the knob a control range of –10 to 10 volts also does the trick, but then the Level Adj module converts all negative levels to positive levels. Using an invert switch is quite convenient for setting the modulation depth.

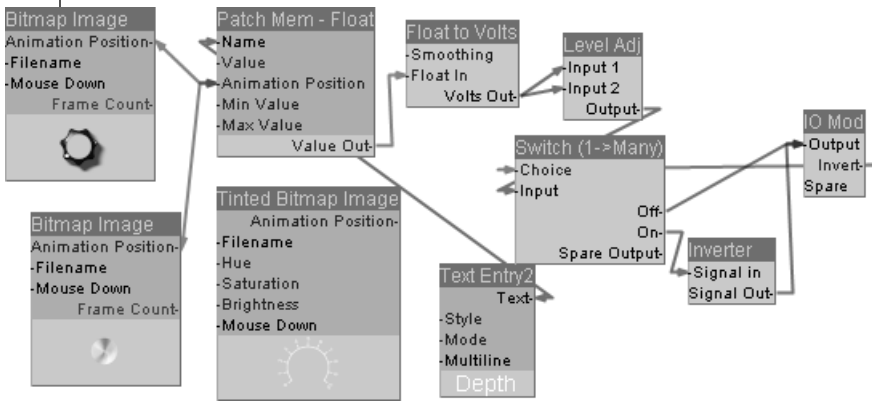


Figure 3.49: The Depth knob's structure

Take a moment to examine the source plugs in figure 3.48 and find the None setting. To enable it, a connected Fixed Values module produces a constant 0 volts. This switches off all modules in the path, which lightens the CPU load. We created the Velocity and Aftertouch sources by connecting these MIDI to CV plugs to the switch's Spare Input plug. The filter envelope's Env Out plug supplies the Filter Env source. The Controllers module's first controller, which is the Mod Wheel by default, delivers the modwheel input signal. Looking for inspiration for your GUI layout? Then look no further than the modulation matrix in figure 3.50.

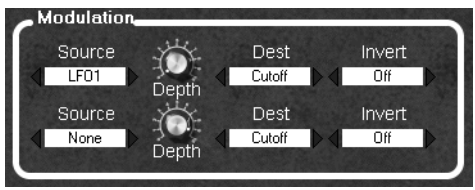


Figure 3.50: A potential GUI layout for the mod matrix

Go-to prefabs: Synthesis > PolySynth10

Heads up: David Haupt offers a module pack for creating versatile modulation matrices. It lets users adjust the modulation amount for each target individually. You'll find a screenshot in figure 3.51. See the module's pack documentation for details.

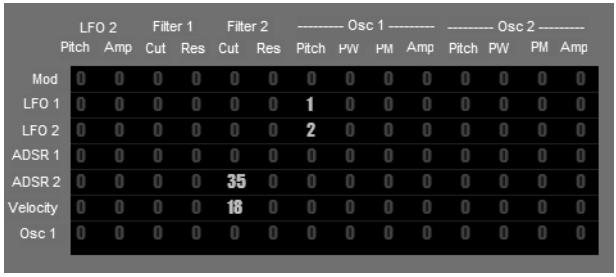


Figure 3.51: DH_MatrixPak screenshot

Finalizing Your Synth

Revealing with Readouts

You may recall from the Effects chapter that you can design readouts by assigning sub-controls to all knobs. The EQ prefabs' structure provides a good platform for building a cutoff readout. Use a KDL Volts2Hz module to convert voltage to Hz. An excellent choice of control range, 0.540568 and 10.64385 volts equal 20 and 22,000 Hz, respectively. We limited resonance to 0 to 9.8 volts. To convert this range to a 0-to-100 % scale, divide it by 0.098 using a Float Scaler so 9.8 volts equals 100 %. To convert the keyboard tracking amount to percentages, simply divide by 0.01. Your control features' layout may end up looking something like figure 3.52. For more on these knobs' internal structure, consult the included prefabs.

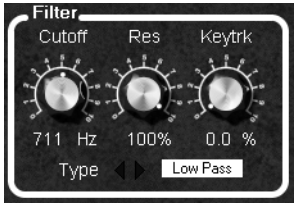


Figure 3.52: A filter section's control panel

You can create similar controls for oscillators. Divide the value by 0.083333 to show the fine-tuning amount in semitones. Feel free to edit the List Entry2 prefab's panel layout. To this end, open the List Entry2 container's panel window. Then arrange the features—arrows, labels, and list box—any way you see fit, perhaps like the example in figure 3.53.

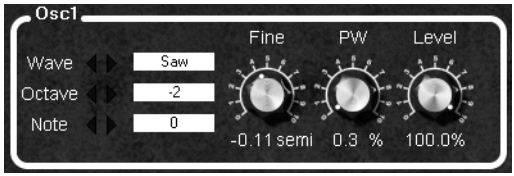


Figure 3.53: An oscillator section's control panel

Heads up: Clicking a list box opens a drop-down menu. If you wish to prevent this and use the Dropdown List sub-control strictly for readout, cover it with a transparent bitmap.

Adding Effects

Many digital and VSTi synths offer effects ranging from simple delay and chorus to elaborate racks brimming with saturation, delay, reverb, flanger, and phaser effects. Place saturation and distortion effects after the filter if you want to saturate each voice. Insert a global distortion effect after the VCA and a Voice Combiner module. Most other effects perform best when placed after the VCA. If you use nonlinear modules in the effect chain, cloning creates one for each voice, possibly wasting CPU power. A Voice Combiner module prevents this problem.

The Effects chapter x-rays the effects' internal structure, so we won't cover the same ground here. To ensure your synth delivers the best performance with effects, be sure to get a handle on sleep mode, the various routings, and polyphonic cloning. The "Making the Most of Performance" chapter covers these topics in details; be sure to read it carefully. Also note that effects routing influences tone; for example, delay and chorus configured in a parallel array sound different than in a serial loop.

Adding Patches

Adding patches to your synth is easy: Load a Patch Select module from the Insert > MIDI folder, and connect the synth's MIDI input to the MIDI In plug. Though the wiring is invisible, this module connects to all sliders, knobs, and other GUI features, and automatically adjusts them in response to a MIDI patch change message. If you insert a Patch Select module, a patch select bar appears at the top of the panel window, which figure 3.54 shows so well. It lets you create and navigate synth patches. Note that this bar does not appear in the exported plug-in. The registered version of SynthEdit stores up to 128 patches for your synth; the shareware version stores 16 patches.

Figure 3.54: A patch select bar

MIDI Automation

Now you know how to assign MIDI control change messages to different plugs. SynthEdit also offers a convenient way of assigning MIDI messages to sliders, knobs, and other GUI features—MIDI automation. To take advantage of it, you must first load a MIDI Automator module to the prefab, and connect the MIDI input to the Automator's MIDI In plug. A MIDI Automator, like the Patch Select module, connects to the control features on the GUI via invisible "wires".

Three types of MIDI messages serve to automate controls—control change (CC), registered parameters (RPN), and non-registered parameters (NRPN). Control changes are MIDI channel messages comprising a controller number and a data value used to adjust a control feature. The MIDI standard calls for 128 controllers; Table 3.2 lists their standard definitions.

0	Bank Select
1	Modulation Wheel
2	Breath Controller
3	Undefined
4	Foot Controller
5	Portamento Time
6	Data Entry MSB
7	Main Volume
8	Balance
9	Undefined
10	Pan
11	Expression
12	Effect Control 1
13	Effect Control 2
14–15	Undefined
16–19	General Purpose 1–4
20–31	Undefined
32	Bank Select (LSB)
33	Modulation Wheel (LSB)
34	Breath controller (LSB)
36	Foot Pedal (LSB)
37	Portamento Time (LSB)
38	Data Entry (LSB)
39	Volume (LSB)
40	Balance (LSB)
42	Pan position (LSB)
43	Expression (LSB)
44	Effect Control 1 (LSB)
45	Effect Control 2 (LSB)
46–63	LSB for Controllers 14–31
64	Hold Pedal
65	Portamento
66	Sostenuto
67	Soft Pedal
68	Legato Footswitch
69	Hold 2 Pedal
70	Sound Variation
71	Sound Timbre (Resonance)
72	Release Time
73	Attack Time
74	Sound Brightness (Cutoff)
75–79	Sound Controllers 6–10
80–83	General Purpose 5–8
84	Portamento Control
85–90	Undefined
91	Effect 1 (Reverb) Level
92	Effect 2 (Tremolo) Level
93	Effect 3 (Chorus) Level
94	Effect 4 (Detune) Level
95	Effect 5 (Phaser) Level
96	Data Increment
97	Data Decrement
98	NRPN LSB
99	NRPN MSB
100	RPN LSB
101	RPN MSB
102–119	Undefined
120	All Sound Off
121	Reset All Controllers
122	Local Keyboard
123	All Notes Off
124	Omni Mode Off
125	Omni Mode On
126	Mono Mode On
127	Poly Mode On

Table 3.2: Standard MIDI controller numbers

Heads up:

- The table above lists standard definitions, but manufacturers' MIDI implementations may differ.
- Controller data ranges from 0 to 127. Controllers 0 to 31 denote the MSB, or most significant bits; controllers 32 to 63 denote LSB, or least significant bits. MSB usually suffice to define a controller; if not LSB data are also sent to provide a more precise definition. Not all manufacturers use LSB.
- Avoid using bank select, data entry, data increment/decrement, RPN, NRPN, and controllers higher than 120 for controlling a synth.

RPN and NRPN messages consist of control change messages. The controller transmits a pair of CC messages to select the RPN/NRPN number. It sends the parameter number's higher 7 bits as CC 101/99, and the lower 7 bits as CC 100/98. Then it sends the actual data as CC 6 (data entry), or sometimes as CC 6 and CC 38 (data entry MSB and LSB). Sending 127 to both CC 101/99 and CC 100/98 usually resets the RPN/NRPN parameter number. SynthEdit handles such messages automatically; you need only set the parameter number to handle NRPN/RPN messages.

You'll deal with two general types of GUI features. Slider and List Entry are individual modules, and the other GUI features use sub-controls to create the control prefab. You can automate Slider and List Entry modules in the Properties window. Figure 3.55 shows a Slider's Properties window. The MIDI Controller ID option assigns a MIDI controller from the list. MIDI NRPN does the same for NRPN messages.

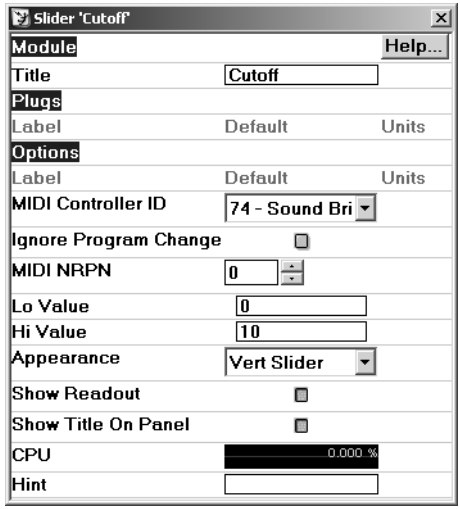


Figure 3.55: Assigning a MIDI CC to a slider

Features with sub-controls work differently. Right-click a container to view its Automation menu options. The Automation function detects Patch Mem modules inside the container, and lets you assign a MIDI controller or RPN or NRPN message number to the Patch Mem. Here's an example: Figure 3.56 shows the structure of the Mod Wheel prefab found in the Insert > Controls folder.

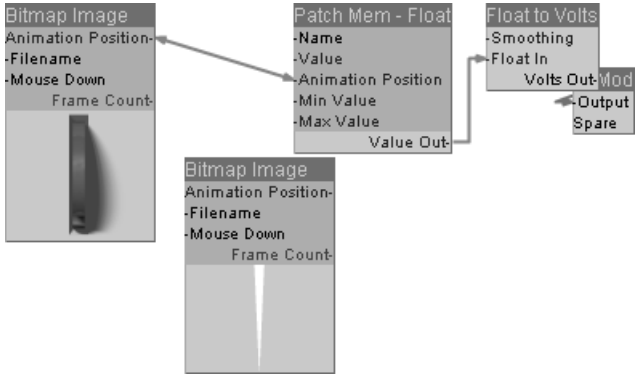


Figure 3.56: The Mod Wheel prefab's structure

The structure features a Patch Mem–Float module that stores the mod wheel image's position. Right-click the Mod Wheel prefab's container, select Automation, and the panel shown in figure 3.57 appears.

Figure 3.57: The Mod Wheel prefab's Automation panel

This Automation panel lists all Patch Mem modules, their values, and its assigned MIDI message. Label indicates the Name plug's value. If you connect a Text Entry2 module to the Patch Mem, it will show the name you chose instead, perhaps something like Cutoff. The Controller default is <none>. In the Mod Wheel prefab, it is CC number 1, which is the mod wheel. Click the Controller box and the Learn and a Set options appear. Learn assigns the Patch Mem to a controller automatically. Click Set, and a panel appears that lets you choose the MIDI message type (CC, RPN, or NRPN) and number.

Ignore PC (program change) works much like it does for Slider and List Entry modules. When enabled, an incoming program change does not affect the control feature's value. When Private is enabled, SynthEdit hides controls earmarked private from the host, though the host will show all other control features.

Heads up:

- You must load a MIDI Automation module into the structure to enable automation.
- Some prefabs in this book use a Patch Mem–Text to display labels. If you wish to configure automation, be sure to assign the MIDI message to the right Patch Mem. Setting the Private option to True hides the controls from the host.
- The Automation panel also lists Waveshaper and Waveshaper2 modules. This means if you don't set Private to True for ADSR knobs' Waveshapers, the host will list them as text parameters.

If you configure automations, do your users a favor and mention the MIDI controller numbers in the synth's documentation.

David Haupt's DH_BasicModulePak offers a MIDI learn module set called DH_MIDIControlMeister. It lets you endow your synth with learning capabilities. See the documentation for more on this.

Getting Funky with FM Synthesis

Introduction

What happens when a sine LFO modulates a sine wave's frequency? Right, you get a vibrato effect. Now what happens if you boost the modulator's frequency so much that it is an audible frequency instead of an LFO? The vibrato effect evaporates in a poof of sonic smoke, and an intricate new timbre appears in its place. That's the short version of how John Clowing discovered FM synthesis in the 1960s. Once he refined and sold it to Yamaha, dozens of synths powered by FM technology surfaced, the most famous being the DX7.

Let's look at how FM works. Figure 3.58 depicts a rudimentary FM setup in SynthEdit. Two oscillators produce sine waves. One oscillator's output connects to the other's Phase Mod plug, making it very easy to conjure frequency modulation. Rather than modulating the oscillator's pitch, this setup modulates the waveform's phase to simulate frequency modulation. The DX7 used the same method. When one oscillator is wired to the other's Phase Mod plug, PM depth controls the phase/frequency modulation amount.

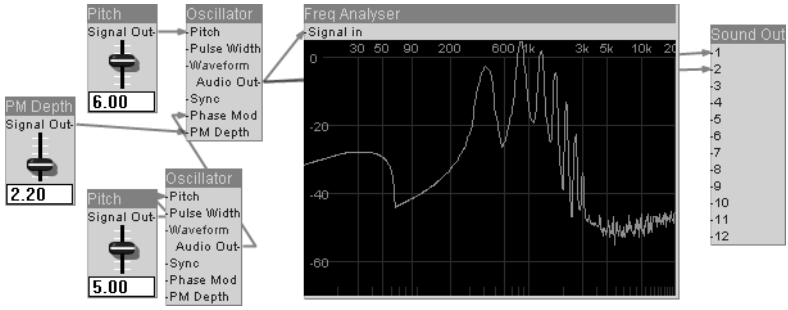


Figure 3.58: The Mod Wheel prefab's Automation panel

In FM jargon, the modulated operator is the carrier, and the other is the modulator. The terms and technology are the same as in broadcasting, expect in the latter the carrier frequency usually lies in the MHz range. In FM synthesis, the carrier is an audible waveform.

Figure 3.58 tells you that the carrier's pitch is 6 volts, or 880 Hz. A pure sine wave's frequency spectrum is merely a single spike. The analyzer shows a spike at 880 Hz with neighboring harmonics called sidebands. Created by the frequency modulation effect, sidebands' amplitude and relationship are determined by the modulator's frequency and FM depth. Varying frequency and depth creates different timbres ranging from metallic sounds to brass and bell-like sounds.

Heads up: Perhaps some of these tones conjure fond (or not so fond) memories if you owned a Sound Blaster, Sound Blaster Pro, or Sound Blaster 16 audio card. Their audio cards featured an FM synthesizer chip (OPL2/OPL3) licensed from Yamaha that produced MIDI sounds of a similar flavor.

Go-to prefabs: Synthesis > Simple FM Example

Experimenting with Modulator and Carrier Algorithms

Yamaha DX series synths usually offer four or six operators for use in many different configurations. And they enable oodles of options for defining operators as modulators and carriers. What's more, a modulator can even modulate itself by way of a feedback loop. With so many routing and feedback alternatives, the possibilities for connecting these operators are endless. Yamaha chose the most musical setups, and called them algorithms. The six-operator DX7 features 32 different algorithms, each comprising a unique operator routing.

Figure 3.59 shows the Yamaha DX7's #1 algorithm. The six operators are grouped in two stacks, each with one carrier and one or more modulators. The lowest operator in each stack is a carrier; those above it are the modulators.

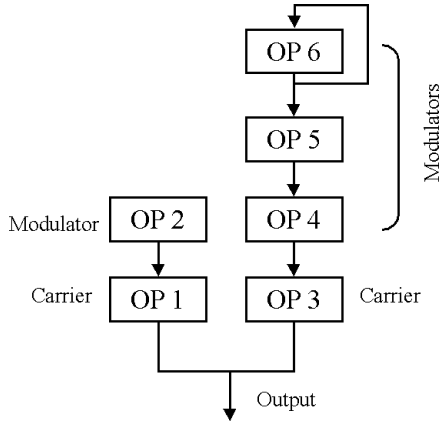


Figure 3.59: #1 with a bullet: A Yamaha DX7's algorithm

In this setup, only operator 1 and operator 3 are audible; the others merely modulate these two operators' frequencies. Operator 2 modulates operator 1's frequency. The other stack is less straightforward. Operator 6 modulates both operator 5 and itself via a feedback loop. Operator 5 modulates the frequency of operator 4, which modulates the carrier, operator 3. This sophisticated modulation scheme creates complex timbres. See the Yamaha DX7 manual if a desire to discover more algorithms overwhelms you.

SynthEdit has a couple of—noun alert—modulation configuration limitations. One is that it lacks sample feedback, ruling out feedback loops. The other is that if you pre-wire a structure for a given algorithm, it limits the synth's sonic capabilities. You can overcome this barrier by using a sophisticated modulation matrix like the mod matrix used in many FM synth plug-ins. Case in point: Say operator 2 modulates operator 1. If you try configuring a setup where operator 1 modulates operator 2, you will a create feedback loop, which SynthEdit won't tolerate. However, operator 3 can still modulate operators 1 and 2. The no-brainer rule is any operator can only modulate operators below it, so operator 5 could modulate operators 1, 2, 3, and 4.

This four-operator setup features in figure 3.60. All operators can modulate any operators below them, and their signals may be routed to the master output. This means all operators can act as carriers, and all operators bar operator 1 can act as modulators. Though the setup lacks feedback loops, it is still flexible.

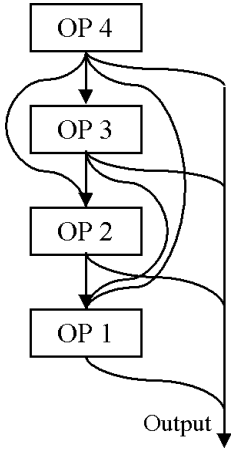


Figure 3.60: A versatile algorithm with four operators

Figure 3.61 depicts the modulation matrix of the setup in figure 3.60. Lacking the main diagonal, this matrix's form is triangular. The circles signify potential modulations. For instance, the circle in row 3, column 2 means operator 3 can modulate operator 2. The missing main diagonal would entail a feedback loop, and that's a no-no.

OP	1	2	3	4
1				
2	○			
3	○	○		
4	○	○	○	

Figure 3.61: Modulation matrix for the flexible algorithm

Sidling Up to Sidebands

Let's define a carrier's frequency as C, and the modulator's as M. The modulator introduces harmonics to the carrier frequency. The upper sidebands always appear at the following frequencies:

C + M, C + 2M, C + 3M, C + 4M, and so on

And the lower sidebands appear at these frequencies:

C – M, C – 2M, C – 3M, C – 4M, and so on

Say the carrier is a 1,000-Hz sine wave, and the modulator is a 200 Hz sine wave. The carrier's ascending sideband frequencies are 1,200 Hz, 1,400 Hz, 1,600 Hz, and so on. Its descending sidebands are 800 Hz, 600 Hz, 400 Hz, eventually reaching 0. Sidebands in the negative domain are reflected, that is, they bounce back from 0. We can treat them as positive numbers and represent them as such with absolute value signs. For example, if C = 1 and M = 2, the first sideband is at |C – M| = |1 – 2| = |–1| = 1. A carrier and a modulator sharing the same frequency generate a harmonic at 0 Hz, and by extension a constant DC offset that is visible on the analyzer. A simple one-pole high-pass filter applied at around 20 Hz prevents this.

We often express carrier and modulator frequencies as C:M or M:C ratio, because they define the sidebands. Lower sidebands do not occur when the C:M ratio is 1:M (the carrier is the fundamental frequency), provided that M ≥ 2 or M = 1. If M = 1, then the first lower sideband is 1 – 1 = 0 Hz, which is a DC offset. The second lower sideband is |1 – 2 * 1| = |–1| = 1, or a frequency identical to the carrier's. If M is greater than 2, then all lower sidebands lie above the carrier frequency, which rules out lower sidebands. Table 3.3 lists some C : M ratios and their first ten sideband frequencies.

C	*M*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	1	2	3	4	5	6	7	8	9	10	11
1	2	3	5	7	9	11	13	15	17	19	21
1	3	2	4	5	7	8	10	11	13	14	16
1	4	3	5	7	9	11	13	15	17	19	21
1	5	4	6	9	11	14	16	19	21	24	26
1	6	5	7	11	13	17	19	23	25	29	31
1	7	6	8	13	15	20	22	27	29	34	36
1	8	7	9	15	17	23	25	31	33	39	41

Table 3.3: A chart full of C:M ratios and their first ten sideband frequencies

If a sideband's frequency lies above the Nyquist frequency, it is mirrored—that is, aliased—back into the 0-to-Nyquist range. This effect increases with the FM depth setting. You can confine it somewhat by limiting the FM depth amount. Waveforms other than a sine tend to exacerbate aliasing.

Sidebands' amplitude hinges on modulation depth. They are 0 without frequency modulation, leaving only the carrier. Ever more sidebands appear as FM depth increases. The FM depth amount determines the number of audible sidebands and thus the modulated signal's bandwidth. Vary this amount over time using an envelope, and you can shape the modulated sound's spectral component. Put simply, the carrier envelope determines the sound's amplitude over time, and the modulator envelope determines the harmonic content (timbre) over time, behaving much like a filter envelope.

Fabricating a Four-operator FM Synth

Assembling Operators

Now that we know what important ingredients envelopes are, let's look at how we can use them to assemble an operator. In DX7, every operator features a dedicated five-stage envelope that lets us fine-tune volume and timbre curves. So grouping an oscillator and an envelope generator together to form an operator seems logical enough. Figure 3.62 shows a structure containing an oscillator, a tuner, an envelope, and a VCA. Imaginatively named Operator, you can copy and paste this container to create clones.

Go-to prefabs: Synthesis > FM Operator

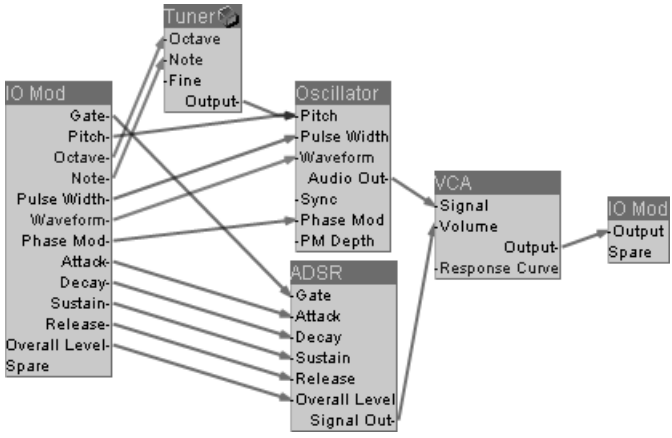


Figure 3.62: The Operator's structure with a tuner, envelope, and VCA

This structure employs a four-stage ADSR envelope. To create more sophisticated envelopes, use one of the third-party envelope generator modules with more stages.

Once you have created this structure and made three copies, add a MIDI to CV module for triggering the envelopes and controlling the oscillators' pitch. Connect the Gate and Pitch plugs to the operator's Gate and Pitch plugs as pictured in figure 3.63. If velocity-sensitive operators like DX7's sound good to you, connect Velocity to the operator's Overall Level plug. Though in a real-world synth conjuring different velocity response curves requires velocity level processing, this example sticks with a streamlined setup to keep things simple.

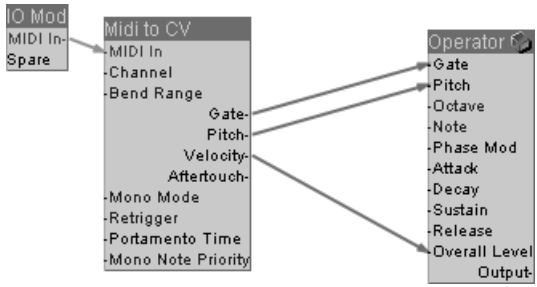


Figure 3.63: Connecting MIDI to CV and an Operator

The next chore is adding sliders and list entries to control the detuner and envelope. Connect list entries to the Octave and Note plugs, and a slider to the Pitch plug for fine pitch tuning. This prefab's Fine slider has a range of –0.0833333 to 0.083333, as in the previous prefabs. Again, we used basic sliders for the Attack/Decay/Sustain/Release plugs for simplicity's sake. Now drop a Level Adj module in after the Output plug to let your users adjust the operator's output level. This output is destined for the main output mixer as shown in figure 3.64. All you need to do now to create a simple additive synth is stack four of these operators and connect the outputs to a mixer. Check out the Synthesis > FM1 prefab to see the full structure.

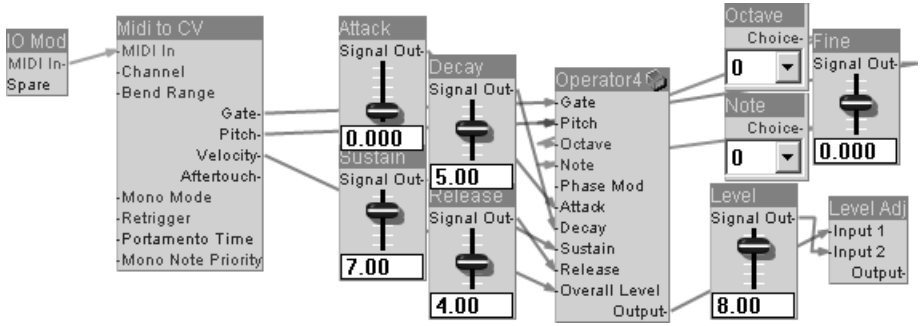


Figure 3.64: Adding controls for the operator

Go-to prefabs: Synthesis > FM1

Adding a Modulation Matrix

Figures 3.60 and 3.61 show a nifty modulation structure. If you wish to build one too, create a separate container for a tidier setup. Connect operators 2, 3 and 4 to the container. Be sure to tap the operator's initial output signal, and not the post Level Adj signal. The latter determines this operator's output level only. If it is set to 0, the operator serves as a modulator only, without issuing an actual output signal. You need six inputs to determine modulation amount (2 > 1, 3 > 1, 3 > 2, 4 > 1, 4 > 2 and 4 > 3). Figure 3.65 outlines the modulation matrix's structure. Six Level Adj modules adjust the modulators' levels according to the depth values. The first Level Adj sets operator 2's level according to 2>1 Depth, and sends it to operator 1's phase modulator. The second Level Adj sets operator 3's level according to 3>1 Depth, and also sends it to operator 1's phase modulator. The third Level Adj sets operator 3's level according to 3>2 Depth, and sends it to PM operator 2, and so forth. The six Level Adj modules thus trace the six modulation paths shown in figure 3.60. Be sure to connect the PM Op1, PM Op2, and PM Op3 outputs to the operator 1, 2, and 3's Phase Mod plugs.

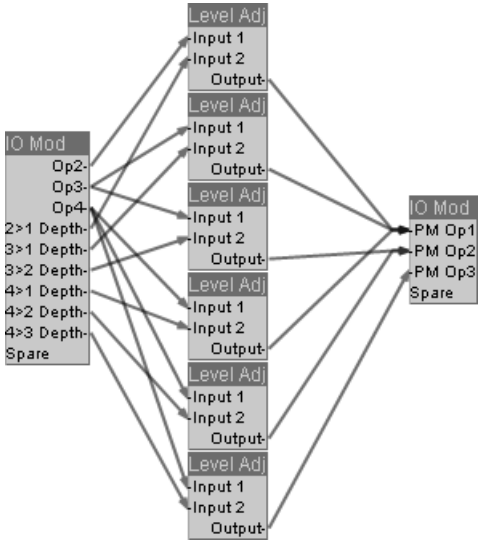


Figure 3.65: The modulation matrix's structure

Of course you need a fistful of knobs to actually use the modulation matrix. Wire six to the Depth knobs, preferably in the order shown in figure 3.61. Figure 3.66 depicts the mod matrix container with the knobs for easy reference. Once you have connected all the components,

your synth's basic FM structure is up and ready to run. You may wish to drop in a one-pole HP filter between the mixer and main output. Set to about 20 Hz, it removes any DC offset generated by 0 frequency sidebands. To see the complete structure, check out the FM2 prefab. Note that the PM Depth knob adjusts Phase Mod depth. This means a PM Depth setting of 5 volts and modulator Level Adj setting of 10 volts yield the same amount of FM as a PM Depth setting of 10 volts and a modulator Level Adj setting of 5 volts. If you wish to extend the modulation depth range, either increase the PM Depth value for all operators, or assign a wider range to the modulation depth knobs. And if you wish to convert the linear range to an exponential scale using a Level Adj module, setting the Depth knobs' range to 0 to 10 volts and increasing the PM Depth knob's amount for the operators sounds like a very good idea.

Go-to files: Synthesis > FM2

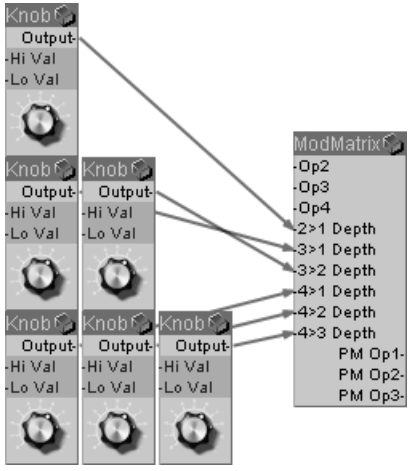


Figure 3.66: The modulation matrix's structure, again

Designing the User Interface

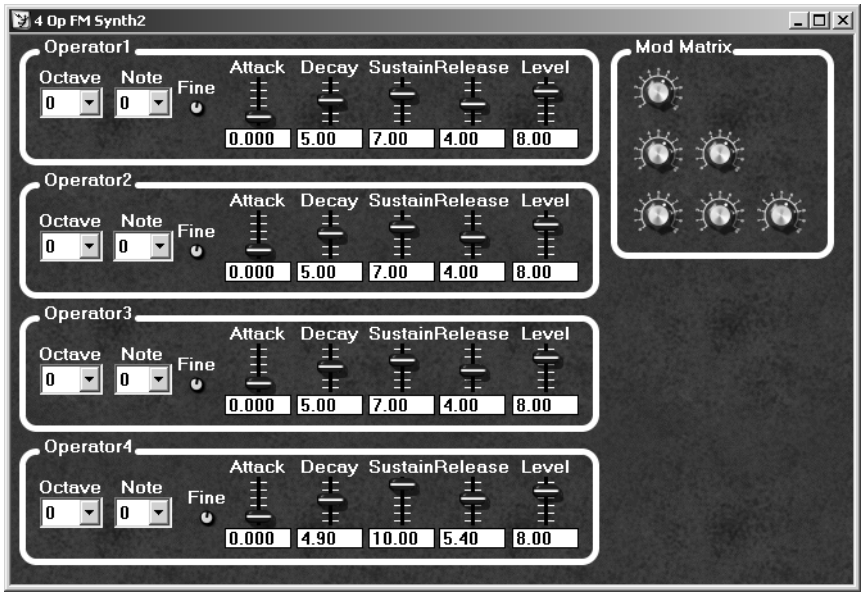


Figure 3.67: A streamlined user interface with the default skin

Figure 3.67 shows a basic user interface featuring the default GUI. Feel free to customize and skin it, and replace standard controls with subcontrols. This prefab lacks modulation matrix labels. Your users are sure to appreciate some signposts identifying modulation knobs, so why not create a skin or background image with the operator numbers written at the top and left of the modulation matrix. Note that the operators' order in the structure is reversed, that is, operator 4 is at the top and operator 1 at the bottom.

Go-to prefabs: Synthesis > FM3

More Good Things to Do

The setup above is a basic four-operator FM synth structure. Extend it to six or more operators if you wish, but be sure to abide by the no-feedback rule.

Add modulation tools such as LFOs to your heart's content. You'll find exhaustive—if not exhausting—discussions of these features in other chapters, so we'll spare you a rehash here.

Some synths slap added filters on the main output signal. Try it; you may like this hybrid synthesis.

Insert after the mixer any external effect that floats your boat—delays, reverbs, chorus effects, ad infinitum.

4

Making the Most of Performance

A processor's resources are finite, so the burden a plug-in places on the CPU matters. Polyphony hinges on CPU usage, and high loads limit the resources the host program can devote other synths and effects. Though modular synthesis confines your options for optimizing plug-ins—certainly in comparison to hand-coded plug-ins—you'll be delighted to learn some improvement is possible.

Early versions of SynthEdit were gluttons for CPU performance, but its author curbed its appetite in later versions. Now when a signal is inactive, it goes to sleep to conserve CPU. Introducing GUI modules enabled further improvements. GUI modules process signals at about 20 Hz rather than the audio rate, significantly boosting GUI performance.

What's Sleep Mode?

Signals generally come in two categories, constant and changing voltages. An inert slider produces constant voltage; an oscillator generating sound produces changing voltage. Most built-in modules detect if incoming voltage is constant or changing. If it's constant, they shut down audio processing to save CPU. The module's Properties window signals this by showing CPU usage has dropped to 0 % as shown in figure 4.1.

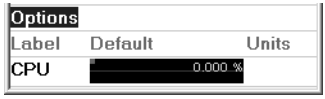


Figure 4.1: A module in sleep mode

Modules in sleep mode usually issue a status signal telling the subsequent module in the chain that it's bedtime. Soon all the modules in the chain go to sleep. The drawback is that the CPU load briefly spikes when a static signal changes, possibly provoking an audio glitch if the system is running at peak load. If the signal doesn't demand updating at the audio rate, use GUI modules instead to prevent glitches. And if suitable GUI modules are available, it's always wise to calculate control signals on the GUI side.

There are many ways to detect if a module's output signal is static. A Monitor (Insert > Special > Monitor) module shows ST_STATIC to indicate static signals, and ST_RUN for changing signals as depicted in figure 4.2.

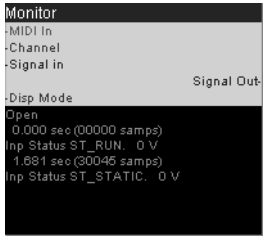


Figure 4.2: A Monitor module

Another method is to connect an Inverter module to the plug, and check if the Inverter module's CPU use drops to 0 %. If so, the Inverter module is in sleep mode, indicating a signal with a static status. The third-party module sc:Status also determines a signal's status. It puts out 10 volts if the signal is active, and 0 volts if it is not. Connect it to an LED indicator, and the signal's status is easily seen. Figures 4.3 and 4.4 map out this prefab.

Figure 4.3

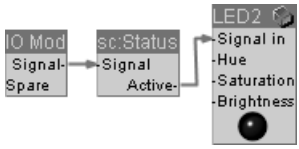


Figure 4.4: Determining status using sc:Status

Go-to prefabs: Optimization > Active Detector

Native modules detect sleep mode, but some third-party modules may not because the function is not compulsory for SE. If you suspect a patch is consuming more than its fair share of CPU power, a sleep mode detector may help track down the culprits.

Go with Better Flow Control

Of the two types of switches, a 1 → Many switch almost always beats a Many → 1 switch. Figure 4.4 shows two configurations that select an effect to process a signal. In the first setup, the active input signal enters the switch. The switch sends this signal to the output specified by the Choice plug. What's more, it automatically sends a sleep mode signal to all other output plugs to switch off unused effects, provided that the modules in the container support sleep mode.

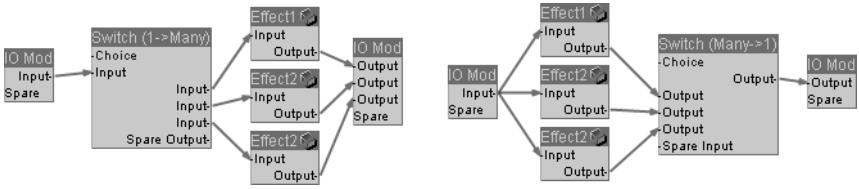


Figure 4.5: Possible flow control routings

In the second setup, the active signal enters through the Input pin. It then goes to all three effects for processing. This wastes beau coup CPU because the Many → 1 switch routes just one signal to the output. So, whenever possible, use the 1 → Many switch rather than the Many → 1 switch. The same goes for SV Filters. Figures 4.6 and 4.7 depict two routing options for choosing a state variable filter mode. The second uses less than half the CPU power of the first. The reason is, the SV Filter works fine for one output, but the processing load doubles

with several outputs connected. The switch in figure 4.7 selects the best SV Filter for the task at hand, and puts all other filters to sleep to conserve CPU. And you can optimize SV Filters further, as you shall soon discover.

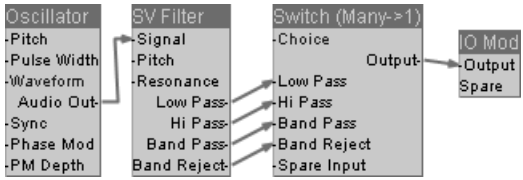


Figure 4.6: Possible SV Filter choices

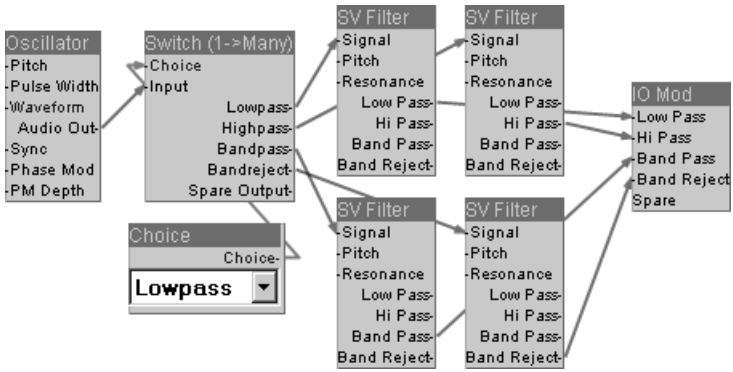


Figure 4.7: Suggested SV Filter choice

Switches also serve to bypass effects. The switch in figure 4.8 routes the signal through unchanged, sets the effect to sleep mode, or sends the signal through the effect container.

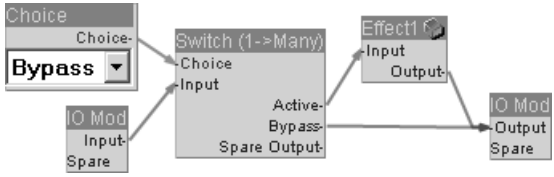


Figure 4.8: A bypass switch

Figure 4.9 maps out a similar structure that lets users adjust levels for delay, reverb, chorus, and flanger effects. Adjusting the level *before* the effect—that is, backing the Level slider off to 0—turns the effect off. When a Level Adj module receives zero input, it puts out a static zero signal and goes to sleep, switching the effect off. If you want an on/off switch, replace the slider with a 0/10 volts switch or button.

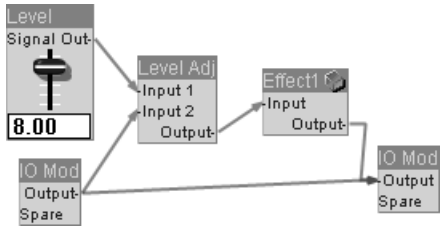


Figure 4.9: A Level Adj employing sleep mode

Optimizing Effects

VST effects pose two problems that beg optimization. One is that a plug-in always interprets its input as a changing signal, so it consumes the same amount of CPU whether it receives a signal of not. Placing a Scoofster AutoSleeper module after the input inside the main container fixes the problem. It waits for an incoming audible signal, and if it doesn't detect one over a reasonable period, puts all subsequent modules to sleep to conserve CPU power.

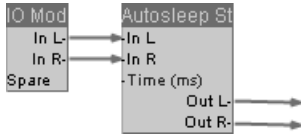


Figure 4.10: A stereo Autosleeper module

Silence poses the other problem. Sometimes a spike or constant load occurs when a sound fades out, sending CPU consumption sky high. Denormal numbers, that is, inaudible signals with very low value, cause this surge. When the processor detects such small numbers, it processes them with greater precision. The elaborate math slows calculations to a crawl, and wastes processing power because the signals are inaudible. Delay, filter, and other modules with feedback circuits can

produce denormal numbers. Native modules remove denormal samples, but third-party modules may have denormal issues. If you come across a likely suspect, use a Special > Denormal Detector module to detect denormal numbers, and a Special > Denormal Cleaner module to eliminate them. An AutoSleeper may also curtail denormal effects. However, if CPU performance spikes after the sound fades and nothing you do resolves the issue, a module may have an inherent denormal number problem. If you have good cause to believe you have discovered a denormal problem in a third-party module, try contacting the developer to report it.

Optimizing Synths

Polyphony

Nothing whets synths' appetite for CPU like polyphony, as each new voice begets clones clamoring for added processing. So, chose your polyphony wisely. Too many voices tax your CPU unnecessarily. Changing the number of voices on the fly is not an option in SynthEdit v1.0150, so define the polyphony in the main container. It's a good idea to slap a Mono mode switch on the interface to let your users lighten the CPU load by opting for mono mode.

Envelope Length

A voice usually remains active until the VCA envelope fades to 0. Longer envelopes means voices are slower to fade, and more simultaneously active voices means higher CPU loads. To define a load threshold, simply set the amplitude envelope's release time accordingly. If you limit release time to 9 volts rather than 10, a voice fades for no longer than 5 rather than 10 seconds. Bear in mind, though, that this also limits the synth's sonic potential somewhat.

Linear vs Non-linear Modules

You may recall from the introductory chapter the difference between linear and non-linear modules: A linear effect's processing sequence is irrelevant. It can first process voices separately and then mix the signals together, or mix the voices together first and process the composite signal, yielding the exact same result. Linear modules include Delay2, Pan, and Level Adj. Not so with non-linear modules: Their clones process each voice separately. Take a VCA's envelope. It modulates every

voice individually, making it non-linear; ditto for waveshaper modules. The rule of thumb is that linear modules at the end of a signal path are not cloned, so they don't consume as much power as cloned modules. Figure 4.11 shows an example.

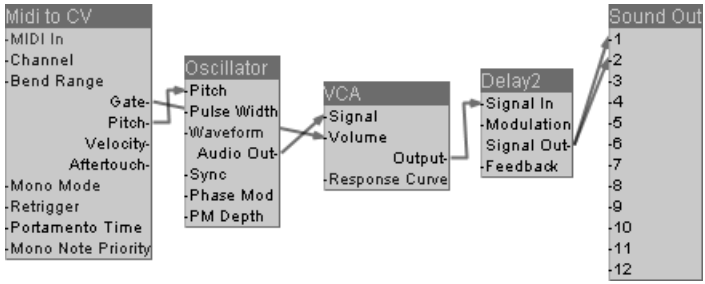


Figure 4.11: Linear vs non-linear modules

Internal definitions designate the oscillator and VCA as non-linear, so they are cloned in this structure. The Delay2 module, in turn, is linear, so its signal is mono. The Delay2 module in figure 4.12 sits between an oscillator and a VCA, which are non-linear modules. This imposes polyphony on Delay2, which may waste CPU. Of course, a structure such as this may be intentional, for example, if you want the oscillator's pitch to modulate a voice's delay time. Note that the CPU graph at the Properties window's top left employs dots to signify polyphony and the actual number of voices in a module.

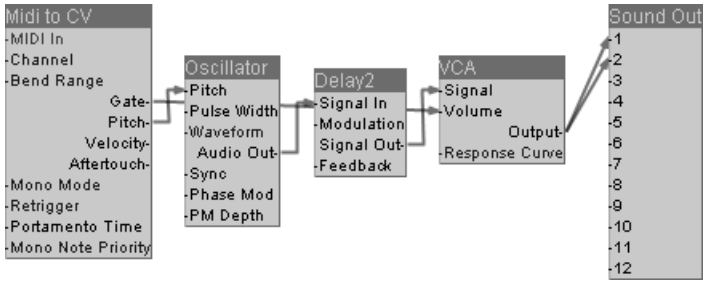


Figure 4.12: A linear module bookended by non-linear modules

Forced Mono

A Special/Voice Combiner module lets you confine cloning to conserve CPU power. It combines incoming voices and shoehorns the composite signal into mono format. Say you drop a Voice Combiner module in front of a distortion module. This economizes on CPU, but yields a different flavor of distortion reminiscent of a guitar stompbox because the effect distorts the composite signal rather than individual voices.

Less Is More, Usually

If two or more solutions yield the same results, the option with fewer modules is usually your better choice. Take Ralph Gonzalez's Quadratic module: It does the math for A + B ∗ X + C ∗ Y ∗ Z, which entails five operands, three multiplications and two additions. This module consumes just some 50 % more CPU performance than a single Multiply module. Calculating this expression with Multiply modules and fixed values takes about twice as much processing power. The reason for this is that every module's discrete function calls, memory management, and so forth want a slice of the CPU pie. The same goes for filters: Filters with internally cascaded stages are usually faster than a cascaded series of discrete filters. Case in point: Routing three single-stage DH_MultiFilter2 modules consumes about twice as much CPU as cascading three filters internally. However, in some instances a solution with more modules is faster. Your best bet is to add up individual modules' CPU use and compare.

Fight the Flab by Cutting Calculations

Do the Math with Waveshaper2

Fewer calculations lighten the processing load. Waveshaper2 beats math expression evaluators and other math modules for calculating mathematical functions with one variable. A remarkably versatile expression evaluator, Waveshaper2 references tables to deliver results faster than all other methods, usually. Note that the Waveshaper2 module's input values range from –5 to 5 volts. If you're dealing with a different range, multiply it and add a constant to rescale it. Use an sc:Rescaler module to do this easily. It calculates the amounts required for a given range, and automatically rescales the signal. Scaling back is easily done in Waveshaper2 by multiplying the result and adding a fixed constant. If you're dealing strictly with positive voltages, use *(x + 5)* in lieu

of *x*. This ensures –5 translates to 0, and 5 to 10. Figure 4.13 shows an example using a Waveshaper2 to convert pitch to Hz in the 0-to-10 volt range. To hide the graph from the user interface, simply drop the module into a container.

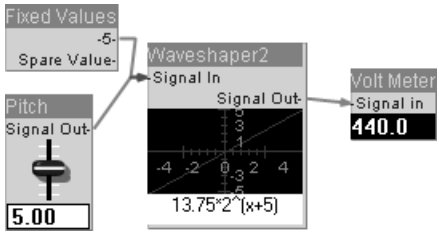


Figure 4.13: Converting pitch to Hz with Waveshaper2

Using Shared Coefficients for Stereo Biquad Filters

Dan Worall's RBJ_Coefficients is the module for you if you wish to reuse the same settings for a stereo biquad filter at fast modulation rates. It calculates the values DH_BiquadFilter's Custom mode requires once, sparing unnecessary repetitions. Figure 4.14 illustrates the structure.

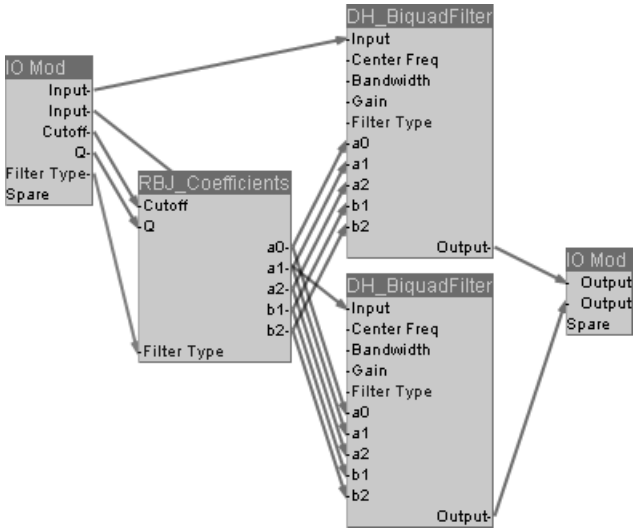


Figure 4.14: A nifty structure module for stereo biquad filters

Go-to prefabs:
Optimization > Stereo Biquad

Stereo Filters with Identical Settings

1 Volt/kHz mode is lighter on the CPU than 1 Volt/Octave mode because the latter requires some heavy math—Freq = 13.75 ∗ 2^{Volts}—to obtain the actual frequency. If you use two filters with the same frequency, the CPU-friendlier approach may be to opt for 1 Volt/kHz processing mode and employ a KDL Volts2Hz to convert the frequency. Divide it by 1,000—better yet, multiply it by 0.001—to arrive at kHz. For even faster processing, try a setup similar to the structure in figure 4.13. It converts pitch to frequency once rather twice as shown in figure 4.15.

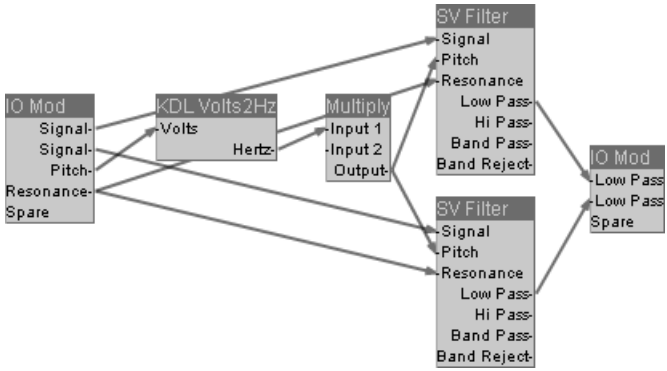


Figure 4.15: Optimized stereo SV filters

Go-to prefabs:
Optimization > Stereo SV Filter LP

Detuned Filters

A similar technique can benefit applications with two or more detuned filters. Say you envision one filter with a cutoff frequency an octave higher than the other's. The straightforward approach is to feed in the same pitch in 1 Volt/Octave mode, and add one volt to one of the two filters. The bad news is that this entails converting the frequency twice. So, let's use the identity:

$2^{a+b} = 2^a * 2^b$

$2^q + 1$ can be written as $2^q \times 2$, meaning it suffices to compute the base frequency once, then multiply it by two to obtain the frequency an octave higher. The rule for obtaining the multiplier for the base frequency is:

$x = 2^D$

D is the detune amount in octaves. If we want to detune by –1 octave, 2^{-1} gives us 0.5, meaning we must multiply the base frequency by 0.5 to obtain the detuned frequency. A Waveshaper2 module gives you the coefficient, as shown in figure 4.16.

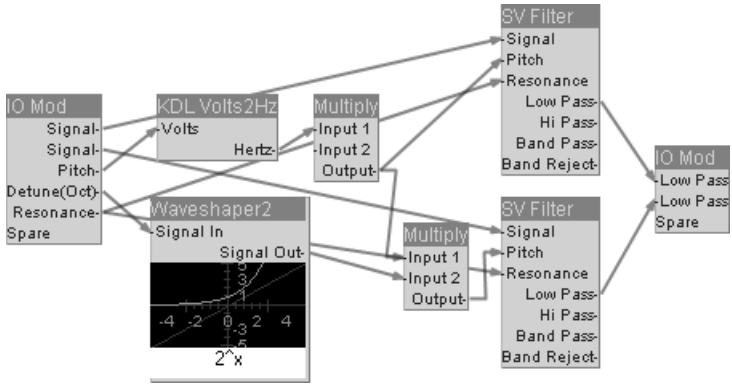


Figure 4.16: Detuned SV filters

Note that the Waveshaper2 module does no processing when the detune amount remains unchanged. Simple multiplication is a far more efficient way of calculating the second filter's pitch than calculating the frequency again with a complex mathematical function. Though this method works with most filters, it's unsuitable for oscillator modules. They do more calculation in 1 Volt/kHz mode, which is due their different internal processing method.

Go-to prefabs: Optimization > Detuned SV Filters

5

All About Sub-controls

What Are Sub-controls?

SynthEdit's sub-control modules are building blocks used to create custom controls for a synth or effect's GUI. This chapter covers the basic concepts you must master to use sub-controls effectively in your SynthEdit projects, and provides plenty of examples to help you get off the ground.

A Traditional SynthEdit Control

Before we set out on our journey into the world of sub-controls, let's recap a native SynthEdit control's structure and functions. We'll use the Slider (Insert menu: Controls) as our example. The module displays a GUI slider control on the panel, and translates the handle's position into a control voltage value at the Signal Out plug (see figure 5.1).

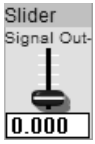


Figure 5.1

Right-click the module, look at its Properties (see figure 5.2), and you will see that the slider offers you several options. You can:

- Assign a MIDI controller for remote control
- Set it to ignore program changes
- Stake out its range using the Lo Value and Hi Value fields
- Change its visuals, turning it into a horizontal slider, knob, or button
- Show and hide its data readout and title

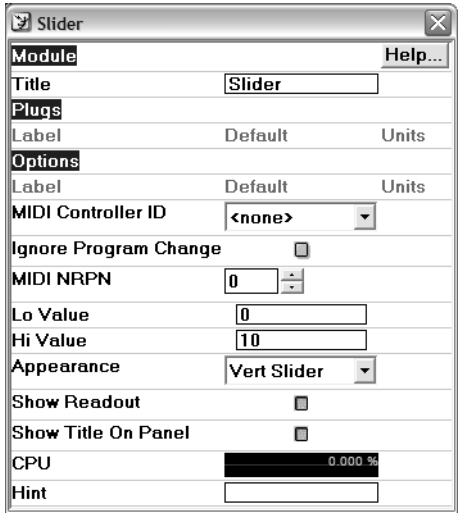


Figure 5.2

Now if you put the module into a container and right-click to view the container's Automation window, you will discover the slider controls an automatable parameter (see figure 5.3). Move the slider with the Automation window open, and watch the parameter value change with the slider's position. Parameters are what ultimately control the processing of a synth or effect, but much more on this later in the chapter.

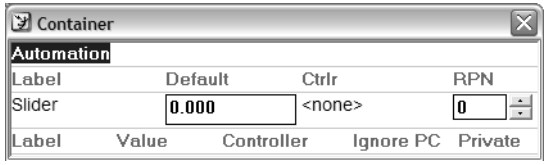


Figure 5.3

A Typical Control Built with Sub-controls

The Knob Sm prefab from SynthEdit's Insert menu, under controls, is an example of a control built with sub-controls. Like the slider control, this prefab displays a GUI control on the panel, and outputs the control's position as a control voltage. For a peek inside the prefab's container, see figure 5.5.

Figure 5.4

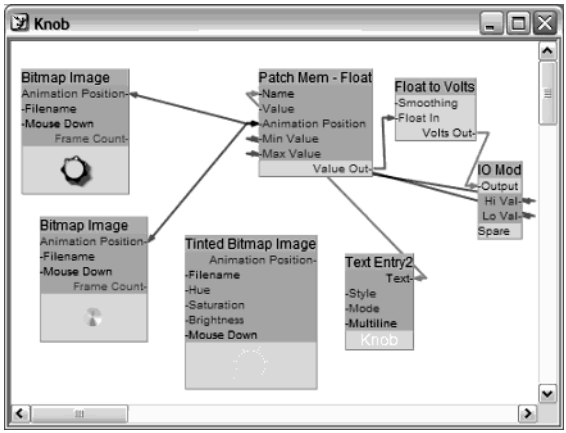


Figure 5.5

We'll examine all these modules' plumbing in greater detail later. For now, a brief overview will do to let us compare sub-control-based and native controls.

The big difference between sub-control-driven and native controls is obvious, yet its importance is easily overlooked. The sub-control-based control is a prefab. Why is this so important? Though you can alter a native SynthEdit control's appearance, your options are limited to a fixed selection of visual styles—button, button_sm, vslider_med, and so forth. While you may substitute different graphic images to re-skin each style, the number of available default controls is finite. Sub-controls impose no such limitations. You can create as many types of custom controls as you want or need. The Bitmap Image module displays any bmp or png image file, and animates multiple-frame images. A Joystick Image sub-control serves to animate single-frame images horizontally or vertically for slider handles and the like, as well as for two-dimensional controls. Once your custom control looks and works the way you want it to, you may simply save it as a prefab, and presto—there you have a new control type. Save it in your SynthEdit\Pre-

fabs\Controls folder, and it will appear in your Insert menu under Controls. If you acquire a taste for creating lots of custom controls, it may behoove you to organize them in sub-folders within your SynthEdit\Prefabs\Controls folder.

At second glance, figures 5.4 and 5.5 reveal that the control's graphics comprises three parts, layered to create the knob. The knob's body and shiny cap are Bitmap Image sub-controls; a Tinted Bitmap Image provides scale markings. While you can create layered images using a good graphics program, doing this on the fly within SynthEdit by combining sub-controls affords you added flexibility. Use the panel view of the control's container to lay out your control's graphic features.

Let's see how the control works. Note that in figure 5.5 the Animation Position plugs of the Bitmap Image modules for the knob and shiny cap connect to the Patch Mem–Float module's Animation Position plug. When a multiple-frame image moves, the Bitmap Image sub-control puts out a number in the range of 0.0 to 1.0, where 0.0 is the first frame and 1.0 is the last frame of the image. Patch Mem–Float then scales the animation position to the range specified by the Min Value and Max Value plugs. The result goes to the Value Out plug, and the Float to Volts module converts it into a control voltage. You'll find this module in the Insert menu, under Conversion.

A control based on sub-controls is easy to hide and show. Set its container's Controls on Parent plug (see figure 5.6) to True, and the graphic sub-controls inside the container appear on the higher level container's panel. Set it to False to hide them.

Figure 5.6

This control does not display a numeric readout. If you wish to add one, add modules to convert the value to text and display the text. Along with the privilege of building custom sub-controls comes the responsibility of adding features.

Looking at the prefab container's Automation window (see figure 5.7), you can see that the knob prefab controls one automatable parameter. If you move the knob with the Automation window open you can see the parameter changing in response to the knob, as was the case with the slider.

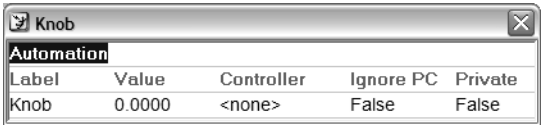


Figure 5.7

The Patch Mem–Float module links the circuit in figure 5.5 to the parameter. The other sub-controls would not affect the parameter if left unconnected to Patch Mem–Float. You can assign the parameter a MIDI controller for remote control in the Automation window. Opt for Ignore PC = True and it will ignore program change messages. Selecting Private = True excludes it from the parameter set available for VST automation.

The Wisdom of Using Sub-Controls

Controls built with sub-controls largely mirror the built-in functions of standard SynthEdit controls, while offering far greater flexibility. Rather than merely letting you re-skin a limited set of stock controls, sub-controls enable you to create an unlimited number of control types. Anything goes, from cosmetic variations on familiar types to new designs offering functionality unattainable with standard controls.

Sub-controls offer even greater benefits:

Easy Animation: Sub-controls that provide GUI features boast built-in animation capability. This lets you create dynamic GUIs that interact with the user. Closely aligned with two-way data flow between sub-control modules, animation will be discussed in that context later in the chapter.

Potential CPU Savings: Sometimes you can use sub-controls to perform conversions and similar calculations on the GUI side in response to user interactions, thereby moving time-consuming computations out of the audio processing stream. The next section discusses the relationship and interaction between the GUI and audio processing.

More on What Sub-Controls Do and How They Do It

GUI Controls, Audio Processing, and Parameters

You'll find sub-controls' purpose and performance easier to grasp if you know a bit about how they fit into the overall structure of a plug-in. This section briefs you on the basics without delving deeply into technical details.

In a VST plug-in, the GUI and audio processing run in separate threads, and are relatively independent of one another. This minimizes CPU contention that could cause audio glitches. Of course, audio processing takes place in real time, so audio data must be processed constantly at the sampling rate (44,100, 48,000, or 96,000 times per second). These time constraints don't apply to the GUI. It operates more or less on its own schedule, driven by events such as mouse moves and clicks. Keeping the GUI and audio processing separate prevents problems. For instance, if repainting a screen were allowed to delay an audio signal, dropouts, clicks, or pops might occur.

A VST plug-in's GUI and audio sides communicate with each other indirectly via the VST host by shuttling parameters to and fro (see figure 5.8).

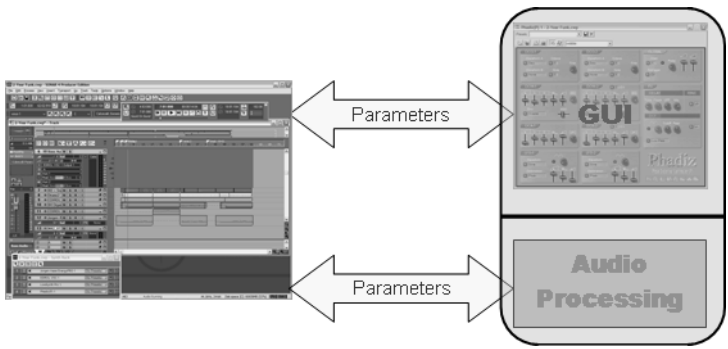


Figure 5.8: Host and plug-in

Each GUI control affecting audio processing must communicate changes in its value to the VST host by updating its assigned parameter. The host then passes the changed parameter value along to the plug-in's audio processing. When audio processing changes a parameter, the host sends the new parameter value to the GUI. For an automated parameter, the host changes the parameter and passes the changes to both the audio processing and the GUI of the plug-in. Most

VST hosts provide a default interface that allows the end user to edit a plug-in's parameters directly, without using the plug-in's custom GUI. In fact, some VST plug-ins lack a custom GUI, relying on the host's default interface instead. A plug-in's custom GUI is really just a fancy graphical editor for the plug-in's parameters.

This division of GUI and audio processing holds true even within a single module for those modules containing both GUI processing and a direct connection to audio processing. SynthEdit's native controls handle this internally. Recall how the standard slider module's parameter changes when the slider's grip moves. If a plug on an audio processing module connects to the slider's Signal Out plug, it also links to the slider's parameter on the audio processing side. When the slider's GUI side sends the parameter change to the host, the slider's audio processing side receives the parameter change from the host and sends it out on the Signal Out plug to the connected audio processing module.

If you're dealing with sub-controls, you must provide a way to connect the GUI features to a parameter. In the case of our example knob, this is the Patch Mem–Float module affording access to the parameter controlled by the knob. Remember, the only way a sub-control can affect (or be affected by) audio processing is to pass parameters through the host.

Of all the SynthEdit sub-controls, only those with Patch Mem in their names afford you access to parameters. Third-party sub-control modules, on the other hand, may or may not access parameters directly. The best way to learn which parameters are in use is to view the Automation window of the container holding the sub-controls. Some third-party modules may use their own parameters for internal purposes. It is usually best to set both Ignore PC and Private to True for parameters that do not correspond to your synth or effect's actual sound parameters (see figure 5.9).

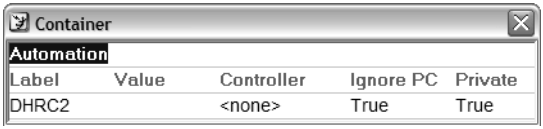


Figure 5.9

GUI Plugs and Data Types

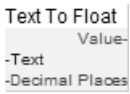


Figure 5.10

One of the first things you may have noticed about sub-controls is that some or all of the plugs appear on blue backgrounds (see figure 5.10) rather than the gray backgrounds commonly found on other SynthEdit modules. The reason for this difference stems directly from the separation of GUI and audio. The blue-background plugs serve to make connections on the GUI side of the plug-in, and will not connect directly to the audio processing plugs with the gray backgrounds. We refer to plugs with blue backgrounds as GUI plugs. Five data types are defined:

Float (blue)	for any kind of numeric value
Int (yellow)	for integers only
Bool (black)	for logical values that can be either True or False
Text (dark red)	for freeform character data
List (green)	for enumerated data

Be sure to use Patch Mem or other conversion sub-controls to transfer GUI plug values to and from the corresponding gray-background plugs (see figure 5.11).

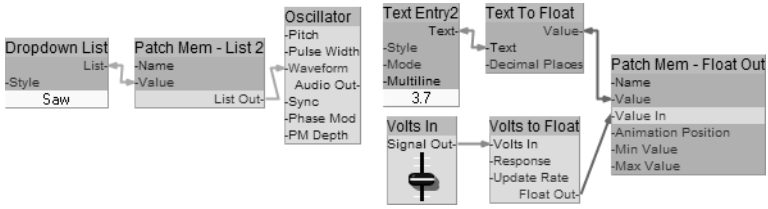


Figure 5.11

It Goes Both Ways—Data Flow and Animation

In contrast to audio connections, where signals flow in one direction, GUI connections are bidirectional. Changes in values move left to right and right to left. In the structure view, the wires that connect GUI plugs have arrow heads at both ends to show this two-way flow (see figure 5.12).

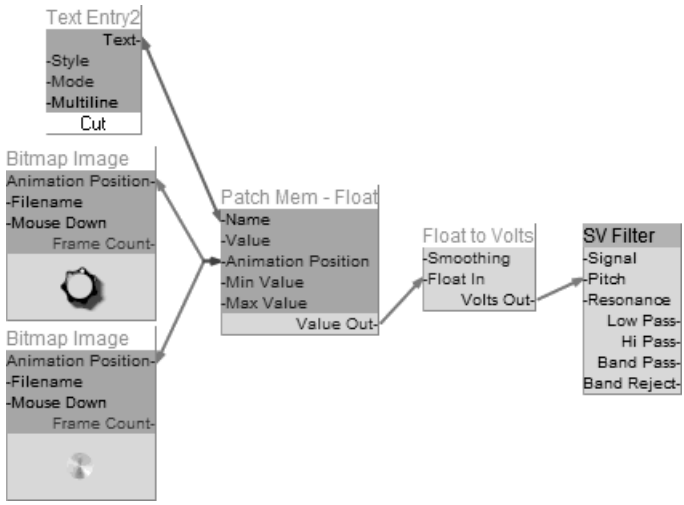


Figure 5.12

The bidirectional nature of these connections may seem confusing after working with the left-to-right flow of other SynthEdit connections. Unfortunately, when you try to make an invalid connection, SynthEdit's error messages don't help: They refer to plugs on a module's left as inputs, and plugs on a module's right as outputs, although a GUI plug can be both.

Most sub-controls that accept graphic or text input also provide graphic or text output. For instance, changing the position of the knob in the Bitmap Image in figure 5.12 changes the value at the Animation Position plug. Conversely, changing the value at the Animation Position plug also changes the knob's position. The knob and its shiny cap in figure 5.12 move in tandem because the motion of one Bitmap Image changes the value of the Animation Position. This change goes to the other Bitmap Image, prompting it to move so it reflects the new posi-

tion. This goes in either direction, so grabbing the knob's outer rim or the shiny cap with the mouse achieves the same effect. The two-way nature of sub-controls makes it very easy to include animation and interactive feedback in your GUI design.

Though GUI plugs on both sides can serve as both inputs and outputs, those at the module's right are not quite identical to those on the left. The latter act as masters, those on the right as slaves. You may connect more than one slave to a master, but only one master to a slave. This means you could not connect a second Patch Mem–Float to the knob Bitmap Image Animation Position plug in figure 5.12. Later you will learn how to solve this problem using splitters.

A Look at Native SynthEdit Sub-controls

This section briefly describes the sub-control modules that come with SynthEdit version 1.0.x. To whip this chapter into shape, we grouped sub-controls by function in six categories:

- Data Manipulation
- Data Type Conversion
- GUI Input/Output
- Parameter Interface
- Routing
- Miscellaneous

Data Manipulation Modules

These modules change the value of a GUI variable. They may or may not also convert the value from one GUI data type to another.

Bools to List and List to Bools

Figure 5.13

Use these modules together to change the composition of a list. Connecting the Value plug of the Bools to List to a green GUI list plug creates a Bool type plug on the left for each item in the list (see figure 5.13). Connecting the Spare plugs of the List to Bools to selected plugs on the Bools to List creates a new list itemizing only items selected at the List to Bools' List plug (see figure 5.14).

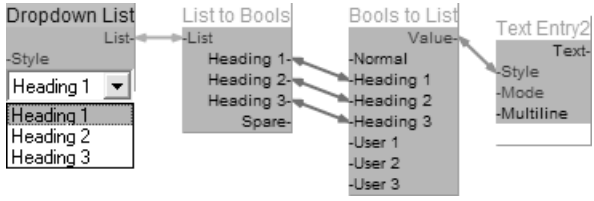


Figure 5.14

dB to Animation

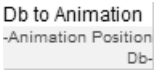


Figure 5.15

The dB to Animation module converts a decibel level in the range of –20 to +3 dB to an animation position in the range 0.0 to 1.0. It uses a non-linear scaling formula whose slope increases along with the input value. This permits the scale of VU meters and the like to be stretched to show more detail at the upper end of the scale. Out-of-range values are clipped to 0.0 and 1.0. The module works in one direction only, from dB to animation position. SynthEdit's VU meter prefab (Insert menu: Controls) is a good example application of this module (see figure 5.16).

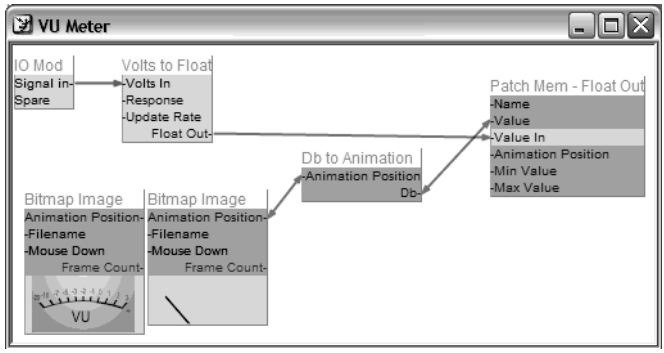


Figure 5.16

Float Scaler

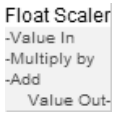


Figure 5.17

The Float Scaler calculates a linear function of its input. Use it to scale a number and/or add an offset. It multiplies the Value In by a specified amount, and then adds a specified amount:

Value Out = y = mx + b

x is the Value In, *m* is the input to the Multiply by plug, and *b* is the input to the add plug. Figure 5.18 illustrates the example 5.0 ∗ 0.50 + 1.0 = 2.50 + 1.0 = 3.50.

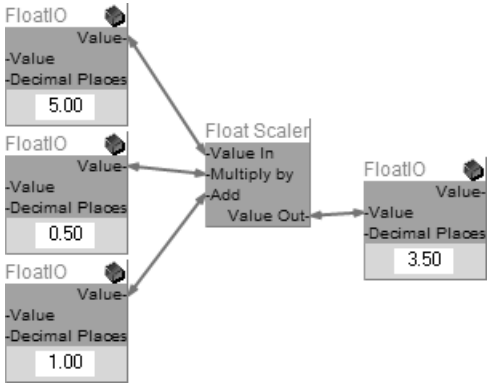


Figure 5.18

The Float Scaler is bidirectional; that is, when the Value Out changes it also calculates the inverse function,

Value In = x = (y – b)/m

The module does *not* recalculate anything when the inputs to the Multiply by or Add plugs change.

Image to Frame

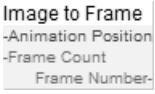


Figure 5.19

Given a frame count, this module converts an animation position to the corresponding frame number. Typically, it connects to a Bitmap Image as shown in figure 5.20. The animation position ranges from 0.0 to 1.0, while the frame number ranges from 0 to 1 less than the frame count.

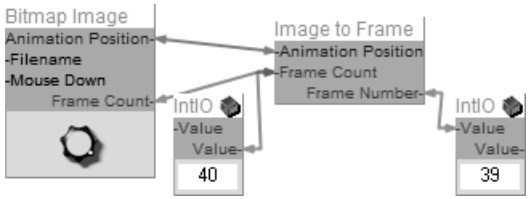


Figure 5.20

Increment2

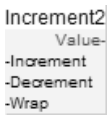


Figure 5.21

The Increment2 module steps through the items of a list, selecting the next item when incremented, or the previous item when decremented. Increment and Decrement plugs respond when their inputs change from True to False, so that a momentary contact button increments or decrements the selection when the mouse button releases.

When the Wrap input is set to True, the selection wraps back around to the beginning of the list when incremented at the end of the list, or to the end when decremented at the beginning. SynthEdit's List Entry2 prefab (Insert menu: Controls) is a good example of this sub-control in a practical application (see figure 5.22).

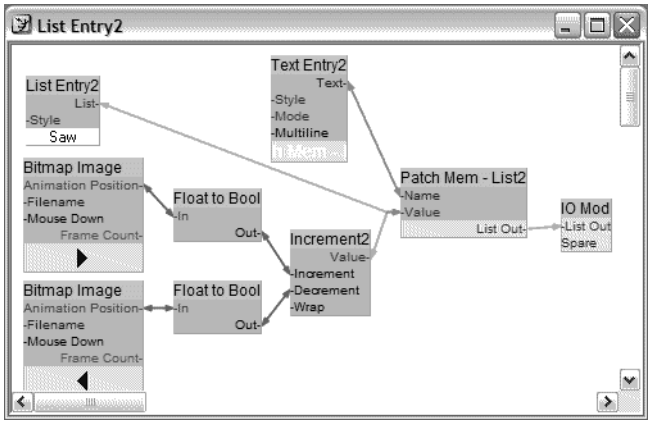


Figure 5.22

Spring

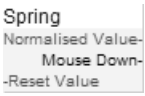


Figure 5.23

This module typically prompts an animated graphic to jump back to a default position when the mouse button releases. It sets the value of its Normalized Value plug to the value of its Reset Value input when the Mouse Down plug's value changes from True to False. The reset value normally lies in the animation position range of 0.0 to 1.0.

In the Pitch Bender prefab (Insert menu: Controls), the reset value is 0.5, which prompts the pitch bend wheel to return to the center position (see figure 5.24).

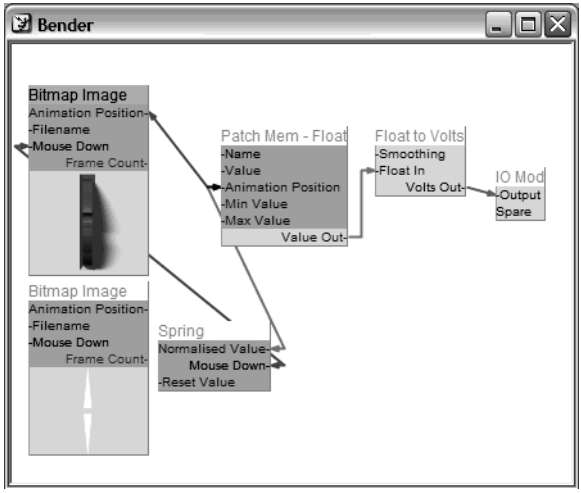


Figure 5.24

Data Type Conversion Modules

These modules convert from one data type to another, without affecting the value.

Float to Bool

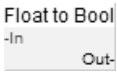


Figure 5.25

This module converts a Float value to a Bool True or False:

If In > 0.0, Out = True
If In ≤ 0.0, Out = False

The module works in both directions:

If Out = True, In = 10.0
If Out = False, In = 0.0

Int To List2

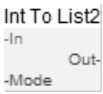


Figure 5.26

The Int to List2 module converts an integer input into a selection in the list connected to its Out plug. It's bidirectional, so it also converts a list selection into an integer. Its two modes of operation are Index and Value.

In Index mode, lists items are indexed starting at 0. Setting the In plug's value to a list item's index selects the item. Conversely, selecting a list item sets the In plug to the item's index. Say the list entries are saw, pulse, triangle, and sine. Then setting the In plug to one selects pulse, and selecting sine sets the In plug to three.

Value mode works like Index mode for simple lists, but not if you're dealing with a list whose items' internal values differ from their ordinal positions. Say the assigned values are

saw = 1, pulse = 2, triangle = 3, sine = 5

In this case, a list item is selected if its assigned value equals the value of the In plug. Conversely, the In plug is set to the assigned value of the selected list item. Setting the In plug to 1 selects *saw*, while selecting *sine* sets the In plug to 5. Setting the In plug to an unassigned value—say, in this case, 4—does not select any list item.

Text To Float

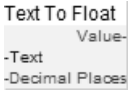


Figure 5.27

This module converts Float values to text strings and vice versa. You decide if you want the plug to determine the number of decimal places displayed in the text string automatically, or define a fixed number in the range of 0 to 10.

GUI Input/Output Modules

This category comprises modules providing GUI features for users to tweak.

Bitmap Image

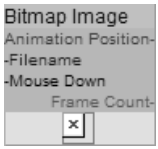


Figure 5.28

The Bitmap Image sub-control displays the bmp or png image named by its Filename plug. Employ it for static images or animated controls.

If you use the module with a multiple-frame image, the Animation Position plug sends the currently displayed frame's position on a normalized scale of 0.0 to 1.0. This goes both ways, so changing the Animation Position plug's value prompts the displayed graphic frame to change.

Pressing the left mouse button while the cursor hovers over the image sets the module's Mouse Down plug to True.

The Frame Count plug outputs the number of frames in the image when an image file is first loaded, and when the window displaying the image first opens. Otherwise, it may not always be accurate.

If you want to use the module to animate multiple-frame images, specify the frame size, type of mouse response, and padding, if any, in a txt file bearing the same name as the graphic file (with txt in place of the bmp or png extension). The txt file format is the same as that used for default SynthEdit skin components.

Earlier in the chapter, Bitmap Image sub-controls in the Knob Sm prefab served to control a numeric control voltage output. The next example demonstrates how a Bitmap Image controls list selections.

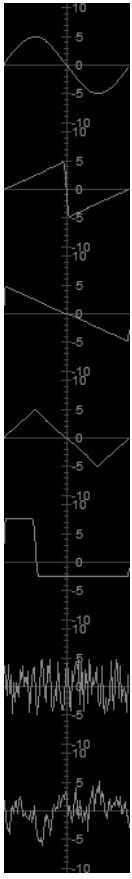


Figure 5.29

Figure 5.29 shows a seven-frame image made from screenshots of the standard SynthEdit oscillator's seven waveforms. Its associated txt file contains the following:

```
type animated
mouse_response horizontal
frame_size 97, 97
```

With mouse response set to horizontal, dragging the image sideways changes the frame, and therefore the selected list item.

In figure 5.30, you can see how the Bitmap Image's Animation Position is converted to an integer frame number. The Int to List uses that integer value to select the corresponding list item. As usual, a Patch Mem module provides the parameter interface between the GUI modules and the audio side.

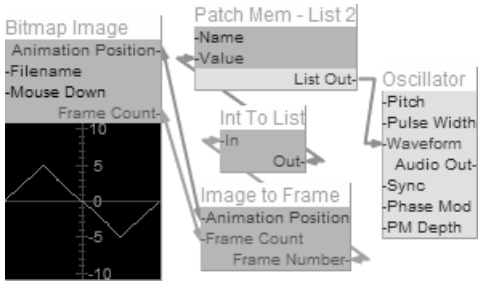


Figure 5.30

Dropdown List

Figure 5.31

The Dropdown List module displays the items of a list connected to its List plug, and allows the user to select an item. Select it in the panel view, and you can resize the module's GUI by dragging the square sizing handles (see figure 5.32). Normally, it looks like a text box with the selected item displayed (see figure 5.33). When the user clicks it (with the panel locked), the list drops down, enabling selection (see figure 5.34).

Figures 5.32, 5.33, and 5.34

Your skin folder's global.txt file contains seven categories offering a selection of font styles for the sub-control—Normal, Heading 1, Heading 2, Heading 3, User 1, User 2, and User 3. Specify the font face, size, color, and background color (including transparent) using the same conventions that define the standard SynthEdit controls' fonts in the global.txt file, for example:

```
FONT_CATEGORY "Normal"
font-family "MS Sans Serif"
font-size 12
font-color #000000
background-color #ffffff
text-align center
```

The Text Entry2 sub-control also accepts these seven font categories.

Joystick Image

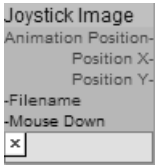
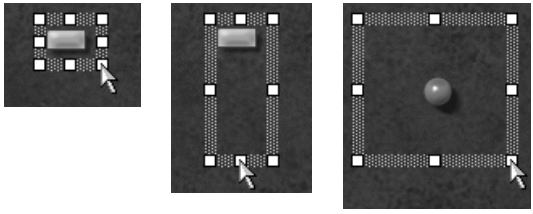


Figure 5.35

The Joystick Image module animates in two dimensions a single-frame bmp or png image named by its Filename plug. It also animates an image in one dimension, either vertically or horizontally. Select it in the panel view, and you can resize the module's GUI by dragging the square sizing handles (see figure 5.36). To confine the motion to one dimension, stretch the rectangle in one direction only (see figure 5.37). To enable free movement from side to side and up and down, extend the rectangle in both directions (see figure 5.38).

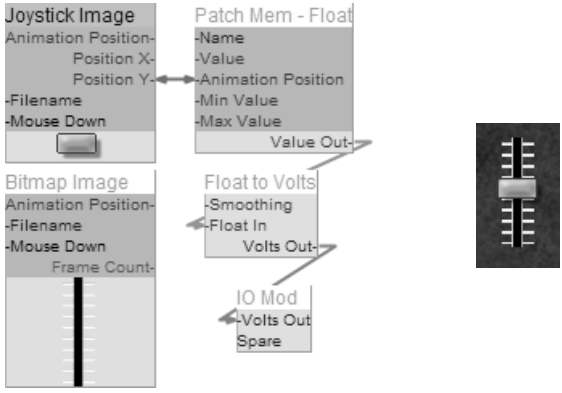


Figures 5.36, 5.37, and 5.38

The Position X and Position Y plugs employ a normalized scale of 0.0 to 1.0 to issue the image's horizontal and vertical position. 0.0 represents the sub-control area's left and bottom edges; 1.0 the sub-control's right and top edges. Interaction is bidirectional, so manipulating the Position X and Position Y plugs' values animates the image's position.

Pressing the left mouse button while the cursor hovers over the image sets the Mouse Down plug to True.

In a typical application, a Joystick Image sub-control combines with one or more Bitmap Image sub-controls to provide a static background (see figures 5.39 and 5.40).



Figures 5.39 and 5.40

Text Entry2

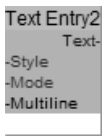


Figure 5.41

The Text Entry2 module accepts input and displays output text. Select it in the panel view, and you can resize the sub-control's GUI by dragging the square sizing handles (see figure 5.42).

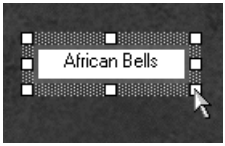


Figure 5.42

Text Entry2 lets you to select from the same seven font categories available for the Dropdown List—Normal, Heading 1, Heading 2, Heading 3, User 1, User 2, and User 3. Again, specify font face, size, color, and background color (including transparent) in the global.txt file using the same conventions that define standard SynthEdit controls' fonts.

The module offers two mode settings, Normal and Read-Only.

If you type text into the sub-control's text box and press the Enter key or click the mouse outside the text box, the Text plug issues this text when the Multiline plug setting is False. If its setting is True, the Enter key advances the insert point to the beginning of the next line. Clicking the mouse outside the text box still prompts the Text plug to put this text out.

The module does not retain typed-in text if the Text plug is unconnected.

Tinted Bitmap Image

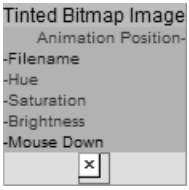


Figure 5.43

Essentially a Bitmap Image module, it colors the image using added plugs for Hue, Saturation, and Brightness—each on a scale of 0.0 to 1.0. In contrast to a regular Bitmap Image module, this one lacks a Frame Count plug.

Parameter Interface Modules

These Patch Mem modules link the other sub-control modules to the parameters they control. Patch memory stores the parameter values for recall when changing patches or loading a song. Again, you can determine if the parameter changes with patch changes by setting Ignore PC to True or False in the parent container's Automation window.

Patch Mem–Float
Patch Mem–Float Out
Patch Mem–Float Out B
Patch Mem–List 2
Patch Mem–List B
Patch Mem–Text

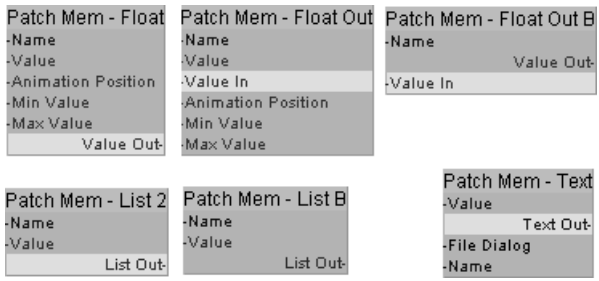
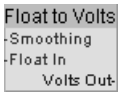
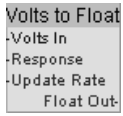


Figure 5.44

We discussed these modules' role in some detail in the sections "GUI Controls, Audio Processing, and Parameters" on page 198. They differ primarily in the type of data (and thus the type of parameter) they handle—Float, List, or Text.

What's more, Patch Mem–Float comes in three variants, with key plugs on the right or left-hand sides to give you more convenient connection options. The gray-background audio processing plugs handle Float data rather than voltages. Voltage must be converted separately because—unlike GUI data—it is processed at audio rates. Float to Volts (figure 5.45) and Volts to Float (figure 5.46) modules (Insert menu: Conversion) do the converting. Float to Volts lets you specify the amount of smoothing applied as changes in Float data are interpolated to voltage changes at the sampling rate. Volts to Float lets you specify the rate at which Float data is updated as voltage changes.





Figures 5.45 and 5.46

It also provides a selection of useful response curves/conversions enabling outgoing Float data to be used to report the input signal in terms of:

- dB VU
- dB PPM
- dB Peak
- dB HeadRoom
- Volts DC (Fast)
- Volts DC (Average)
- Volts RMS

The Patch Mem–List sub-control comes in two guises. Patch Mem–List2 interfaces with modules that use SynthEdit's original List Entry data type (a green plug on a gray background). Patch Mem–Text features a Bool type File Dialog plug that opens a standard Windows File Open dialog when its value changes from True to False (mouse up on a momentary contact switch). Then the file selected from the dialog provides the Value Plug's input.

All Patch Mem modules have a Name plug. This serves several purposes: It gives you a handy place to connect a Text Entry2 you can use to label the control on your GUI. What's more, this label is imposed on the parameter. This name appears at the left of the parameter's value in the Automation window. And the same name appears in the host's VST parameters list when the user opens the host's default interface to your plug-in's parameters, or uses them for VST automation.

Routing Modules

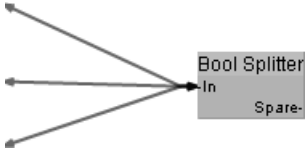
Routing modules help you sidestep some of the constraints on connecting GUI modules. Granted, only one of SynthEdit's native sub-controls falls into this group. However, you can hardly do without several third-party modules in the group if you wish to achieve serviceable results with SynthEdit sub-controls. Jeff McClintock notes that future SynthEdit versions will resolve this issue.

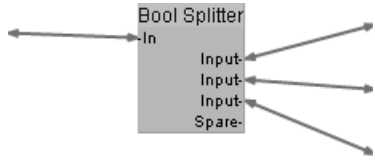
Bool Splitter

Figure 5.47

So, what do you do if you wish to route the same Bool type value to several destinations? Easy enough, if the value feeds a left-side plug: Simply connect it to as many locations as you need (see figure 5.48).

If the value you wish to distribute feeds a right-side plug, you have a problem. The right-side plug is a slave that can have but one master, which means you have just the one connection option. This is where splitters come in. The Bool Splitter duplicates as many right-side plugs as you need, and passes the value of its left-side plug to all (see figure 5.49).





Figures 5.48 and 5.49

Although this problem with right-side plugs affects other GUI data types, SynthEdit provides a splitter for the Bool type only. Third-party module packs provide splitters for Float, Text, and Int.

Miscellaneous Modules

This group contains sundry modules that don't quite fit into other categories. Only one native SynthEdit sub-control falls into this group.

System Command2

Figure 5.50

The System Command2 module lets you call any of the following Windows commands from within your plug-in, using the Filename as the argument to the command:

- Edit
- Explore
- Find
- Open
- Print
- Properties

A command works only if defined as a valid Windows command for the specified argument. This depends on Windows Registry settings, which may vary from one user's machine to another's. Filename need not always be a file. For example, Open can be used with a URL to open a Web browser on the specified Web page. The Explore command expects the argument to be a directory path.

Changing the Trigger plug's Bool value from True to False issues this command.

Putting Your Sub-control Skills into Practice

Now comes the fun part. This section shows you how to do useful things with native SynthEdit sub-controls. Fair enough if you jumped right to it, for learning by doing is a fine way to go. The first few examples recap a few key topics discussed earlier. If you skipped the chapter's earlier sections, you are well-advised to at least thumb through them. You don't have to read all the copy; simply skim the many interspersed examples, and you may discover some to be helpful.

Making Simple Connections

The first two examples merely demonstrate how to replace native SynthEdit controls with sub-controls serving much the same purposes.

As figure 5.51 illustrates, you may use a Patch Mem–List 2 to connect a sub-control such as a Dropdown List to a List Entry type connection. Text Entry2 connects to the Patch Mem–List2's Name plug; it names the parameter and labels the control.

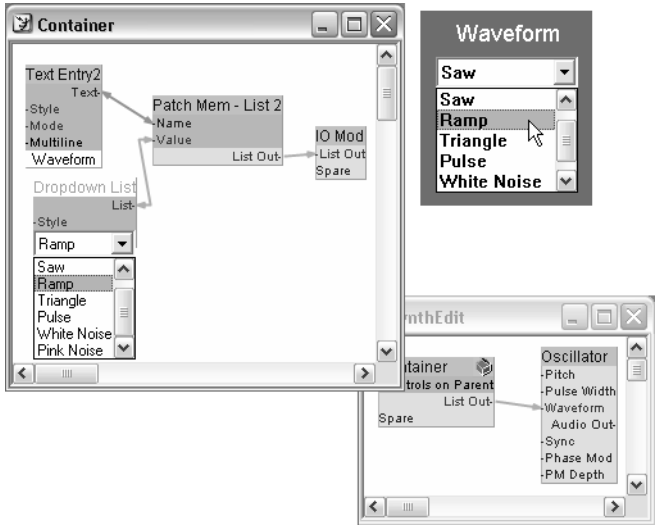


Figure 5.51

With a Patch Mem–Text module, you can use a Text Entry2 sub-control in place of the native Text Entry control, as in figure 5.52.

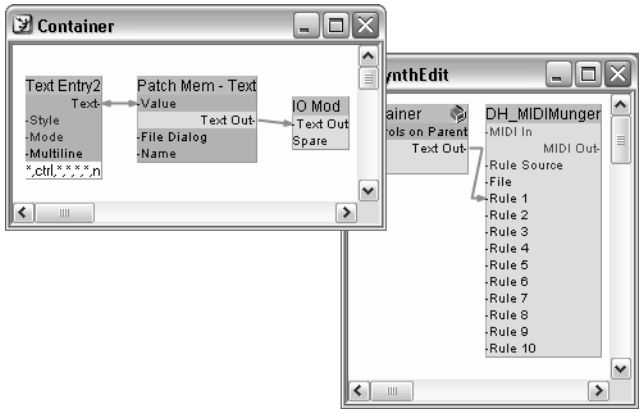


Figure 5.52

This example's Value text is a DH_MIDIMunger rule. You don't want it to appear on the GUI, so hide the Text Entry2 module inside another container, with its Controls on Parent set to Off. In this case, set Private in the container's Automation window to True, as the DH_MIDIMunger rule is unsuitable for VST automation. If you are using different rules for different patches, leave Ignore PC set to False; otherwise you must set it to True so it is available to all patches.

Of course, native controls would work fine for both these examples. Using sub-controls as simple replacements for existing controls has its pros and cons. On the upside, Text Entry2 offers a multi-line entry option unavailable from native Text Entry control. Also, as described in the section "A Look at Native SynthEdit Sub-controls" from page 202 onwards above, both the Dropdown List and Text Entry2 let you select from seven user-defined font styles you can specify in the global.txt file in your skin folder. What's more, sub-controls handle output and deliver input, which lets you change the text or selection dynamically. Native controls can't do this, but they do have other advantages: A Native List Entry control easily morphs from a combo box into an LED stack, a selector, a button stack, a rotary switch, or an up/down selector. All you have to do is choose one of these visuals on its right-click Properties window. Although you can build all these controls using sub-controls, a native List Entry control type that meets your needs requires a lot less elbow grease.

Bitmaps as Controls

The Bitmap Image sub-control lets you use multiple-frame bmp or png images as controls. You've seen this application several times in this chapter, so this example merely rehashes the basics. Figure 5.53 shows the Bitmap Image module's right-click Properties.

Here, we've selected the moog_knob.png file from the SynthEdit default skin folder in the Bitmap Image sub-control's right-click Properties.

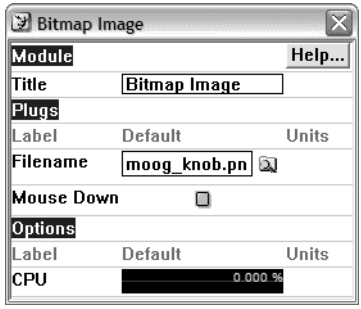


Figure 5.53

This image has 40 frames, so its animation runs very smoothly. Its associated moog_knob.txt file looks like this:

```
type animated
frame_size 48, 45
mouse_response rotary
padding 13, 7, 13, 4
```

In the structure in figure 5.54, a Patch Mem–Float and a Float to Volts converter convert the image's Animation Position to voltage.

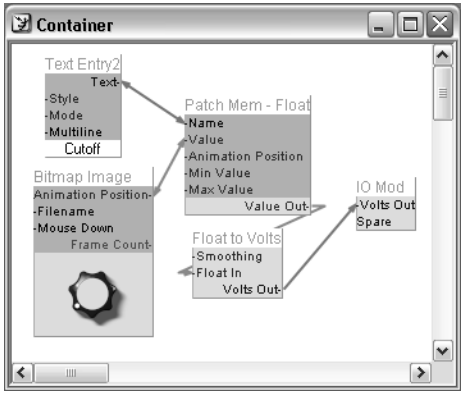


Figure 5.54

Text Entry2 provides a name for the parameter and a label for the GUI.

Patch Mem–Float's Min Value and Max Value demarcate the output voltage range.

You may use *any* multiple-frame bmp or png image in this way. And that speaks volumes about sub-controls' power and flexibility.

Simple Panel Selection

Figure 5.55

You can use sub-controls to provide selectable panels, making it much easier to manage screen space in your GUI design. Containers have a Bool type plug called Controls on Parent (see figure 5.55). When this plug's setting is True, any controls visible in the container appear in the panel view of the container's parent container.

To make selectable panels, build each panel inside a container, lay out each container's panel view as it you want it to appear when selected, and control the process of showing and hiding panels by setting only one of their Controls on Parent plugs to True at a time.

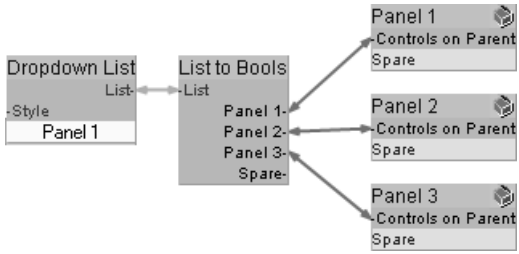


Figure 5.56

A List to Bools module is the perfect choice for this.

It automatically replicates as many Bool plugs as you need, and creates a list offering each container's Controls on Parent plug for selection.

Feel free to change the names of list items in the List to Bools' Properties. Selecting a list item sets the corresponding Bool value to True, and all others to False. This shows the selected container's panel on the parent panel. Of course, a Dropdown List is not your only option for selecting the panel. You could add Increment2 modules to step through panels, or use a Bools to List to connect individual switches, as in figure 5.57.

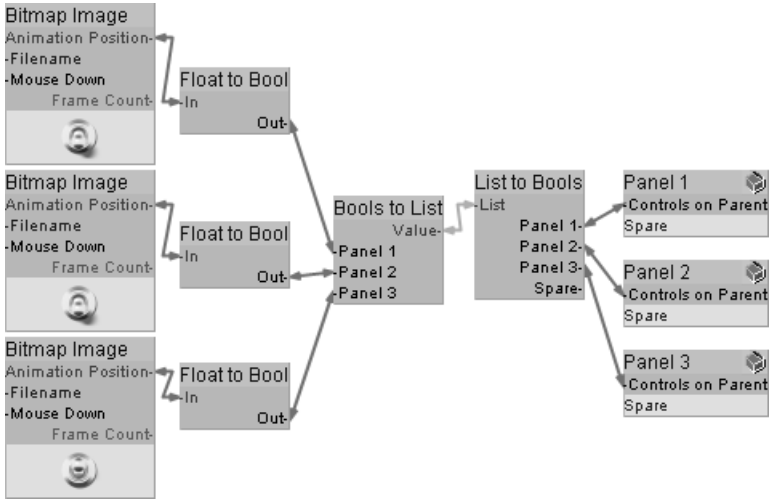


Figure 5.57

Limiting and Ordering List Selection

Sub-controls greatly simplify the tasks of limiting list selection and changing list items' order. When you hook up a Bools to List module to a GUI list type connection linked to a list type input, it automatically sprouts a left-side Bool connection for each item in the list. Connect the items you want in the desired order to a List to Bools as shown in figure 5.58.

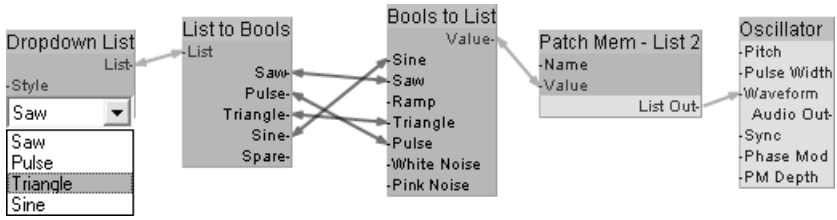


Figure 5.58

Adding a File Open Button

With sub-controls, you can use custom graphics to provide a File Open dialog. Figure 5.59 shows a two-frame button image used in the Bitmap Image module. Here's its txt file:

```
; Load button
type animated
mouse_response click
frame_size 67,30
```

Pressing the button sets the Bitmap Image's Animation Position to 1.0; releasing it to 0.0. Float to Bool issues a True for the former, and a False for the latter. The False-True sequence triggers the File Dialog plug on the Patch Mem–Text, which opens a Windows File Open dialog. The selected file name then passes to the Wave Player.

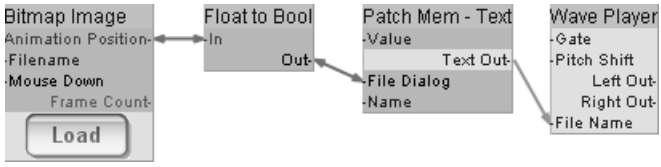


Figure 5.59

Linking to a Website

The System Command2 sub-control performs various functions. Like the file dialog above, a custom graphic can trigger it (see figure 5.60). To link to a website, select *open* as the Command, and enter the complete URL as the Filename as shown in figure 5.61.

Note: Be sure to use the http:// prefix.

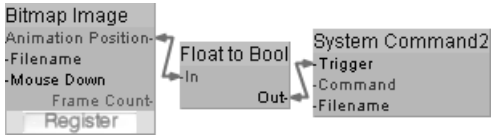


Figure 5.60

No http:// is a *no go!* Use http://www.synthedit.com

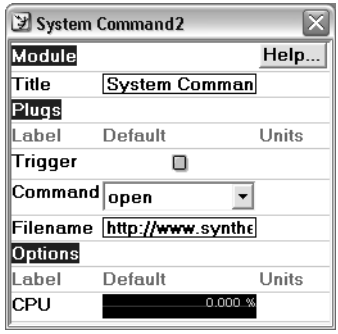
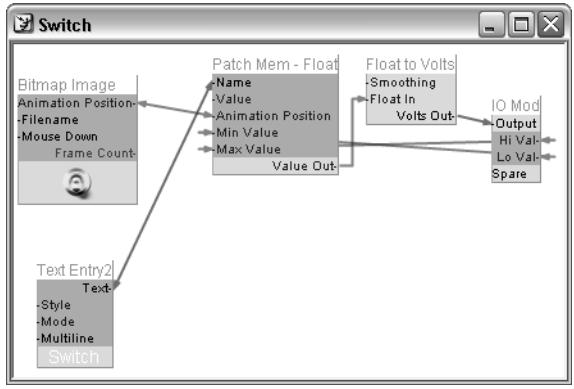


Figure 5.61

Exploring SynthEdit Prefab Controls

Building a custom control using sub-controls is in some ways akin to building a miniature SynthEdit project. You design the functionality in the structure view, and the appearance in the panel view. The prefab controls in SynthEdit's Insert Controls menu offer some choice examples of how all the parts fit together, some of which will be familiar from previous sections. In this section, we'll look at a few more, and examine some prefabs we developed for this chapter to illustrate more techniques for you to apply to your custom control designs.

Let's start with the SE Switch control. Figure 5.62 depicts how it looks in a panel view. Figure 5.63 shows its simple structure, which should be looking fairly familiar to you by now.



Figures 5.62 and 5.63

The Animation Position is 0.0 or 1.0, depending on the switch's position. This value gets scaled up to the Patch Mem–Float's Min—Max Value range of 0.0 or 10.0. First stored as a Float type parameter, this range is then converted to output voltage. Text Entry2 provides the parameter's name and the control's label.

Figure 5.64

Figure 5.64 shows the two-frame image used for the Bitmap Image sub-control. Its text file contains:

```
type animated
orientation vert
frame_size 25, 26
padding 5, 2, 5, 3
mouse_response stepped
```

Note that the switch's panel view in figure 5.65 was unlocked to change Text Entry2's position and text.

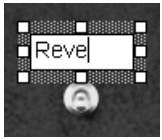
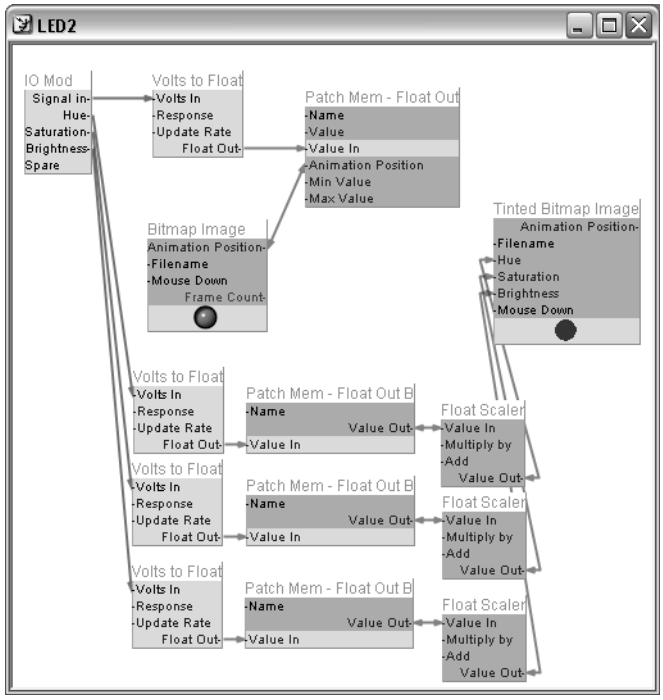


Figure 5.65

Next up is the SE LED2 control. A sub-control-based prefab, it's more flexible than the native LED Indicator control in that its Hue, Saturation, and Brightness plugs let you adjust any color dynamically (see figure 5.66). Figure 5.70 shows its structure. Note that an animated Bitmap Image provides the lens, while a Tinted Bitmap Image supplies the color. Voltages greater than 5 at the Signal In plug prompt the Bitmap image to display the lens image's second frame. It is more transparent than the first, allowing more of the color from the Tinted Bitmap Image to shine through when the LED is on. For this to work, you must position the Tinted Bitmap Image behind the Bitmap Image in the z-order by using the To Back command on the module's right-click context menu.

Figure 5.66

The Tinted Bitmap Image uses a single-frame bmp image file (figure 5.67); the Bitmap Image uses a two-frame bmp image with a mask (figures 5.68 and 5.69).



Figures 5.67, 5.68, 5.69, and 5.70

LED2 works simply enough: Volts to Float and Patch Mem convert each voltage input to a Float value. DC Volts (Fast) is selected for the Volt to Float Responses, with Update Rates set to 20 Hz. Selected Patch Mems' types hinge on whether the left or right side requires Float output. The value originating in the Signal In plug drives the lens Bitmap Image's Animation Position. The Tinted Bitmap Image's Hue, Saturation, and Brightness inputs require values ranging from 0.0 to 1.0, so Float Scalers multiply the voltage values by 0.1 to scale them down from Synth-Edit's 0-to-10 volt range.

Adding Animation

For our next example, we'll start with another prefab from the SE Insert:Controls menu, and then add our own twist to it. Figure 5.71 shows the structure of the Joystick prefab.

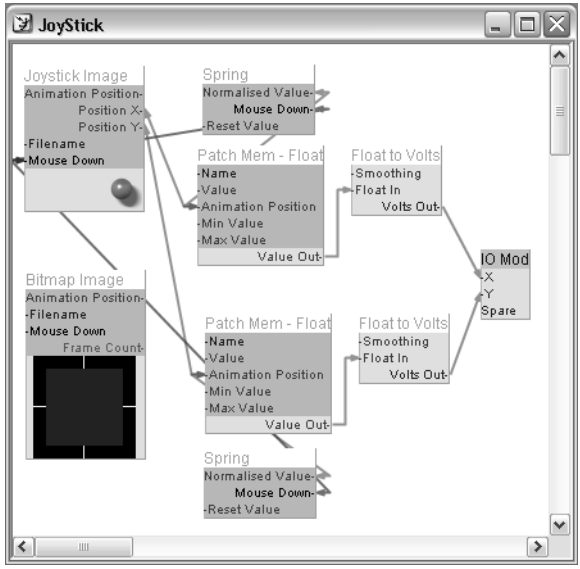


Figure 5.71

Nothing new or revolutionary here: A Joystick Image module reports both of a knob image's X and Y positions, making it perfect for the joystick knob. A static Bitmap Image provides the rest of the joystick. Scaled from 1.0 to 0.0, the X and Y positions provide the Animation Position inputs for the two Patch Mem–Floats. The Patch Mems' Min Value and Max Value are –5 and +5, respectively. They scale Animation Positions to float values within this range, convert to voltages, and feed the signal to the X and Y plugs. When the user releases the left mouse button after clicking or dragging the joystick knob, the Mouse Down value changes from True to False, prompting both Springs to set the Animation Positions to the Normalized Value of 0.5, thereby centering the joystick knob.

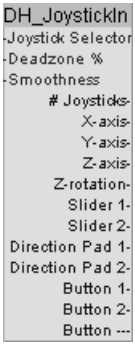


Figure 5.72

Now let's take the Joystick prefab and modify it to create an animated version of the DH_JoystickIn module (Figure 5.72). This lets users employ an external joystick or game pad as an input source for Synth-Edit.

The DH_JoystickIn's X-axis and Y-axis outputs range from –5 to +5 volts, which coincides with the Patch Mem–Floats' Min Value to Max Value range.

Using the X-axis output to vary the input to the Value plug on the "X" Patch Mem–Float changes the Animation Position, and moves the knob sideways. If you want up and down movement, have the DH_JoystickIn's Y-axis output vary the input to the Value plug of the "Y" Patch Mem–float. Figure 5.73 shows you how to do this.

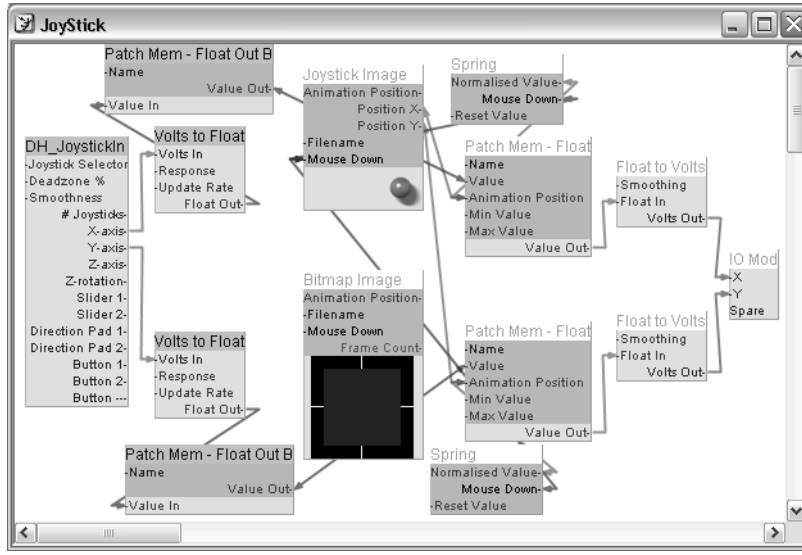


Figure 5.73

Splitting a List

Figure 5.74

Time to build a custom selector button of your very own. This exercise illustrates lots of important techniques, one being how to split a list without having a list splitter module at your fingertips. Our selector will step through a list as it is clicked, returning to the top after arriving at the last item. Its label will change with each click to show the selected item. Figures 5.74 and 5.75 depict the prefab and its panel view. We'll keep it simple by settling for three selections—A, B, and C.

Figure 5.75

For a change of pace, we've used a transparent image with a Tinted Bitmap Image for the button. The tint provides the main color, and the background fills in the shading. Figure 5.76 shows the structure.

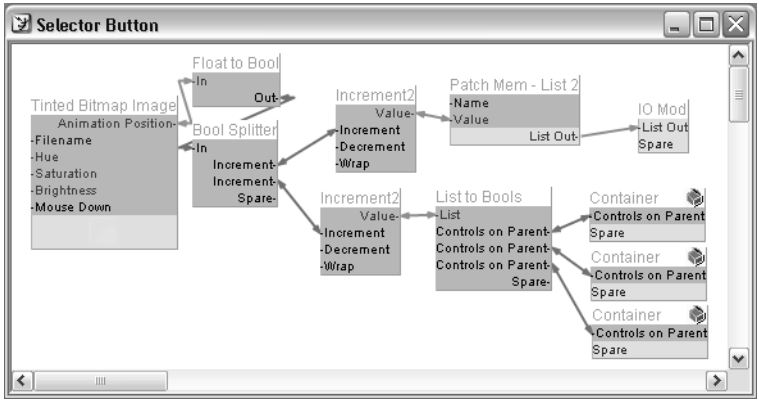


Figure 5.76

Clicking the button moves the Animation Position from 0.0 to 1.0; releasing it resets it to 0.0. The resulting False-True-False output from the Float to Bool converter is then split two ways, prompting each of the Increment2 modules to select the next item in its list. Both Increment2 modules' Wrap plugs are set to True, so the list returns to the first entry after reaching its end.

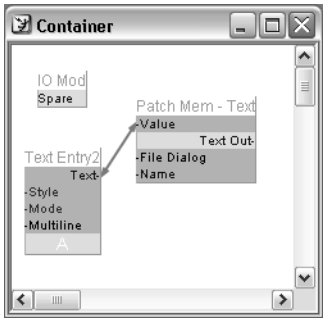


Figure 5.77

The top list connects to the external List Out via a Patch Mem–List2. Created by List to Bools, the bottom list determines which of the three Containers at the lower right Controls on Parent is set to True so its contents appear on the parent panel. The containers hold labels for the list items; in this case, A, B, and C as shown in figure 5.77. Every item in the external list requires a dedicated label container. Though some third-party sub-controls take label information directly from the external list, we're focusing on learning to handle the sub-controls that ship with SynthEdit. We'll look at others later. The point here is that the structure in figure 5.76 splits our list. Though lacking a list splitter module, we had a Bool Splitter split the input, and fed it to two parallel lists, thereby achieving much the same result.

Let's take this a step further to create an LED Stack selector. We'll make it horizontal rather than vertical like the standard SynthEdit control (see figure 5.78). The horizontal LED stack prefab's and our selector button's structures are identical, except that we have split off a third list to control the LEDs (see figure 5.79).

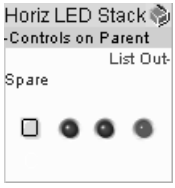


Figure 5.78

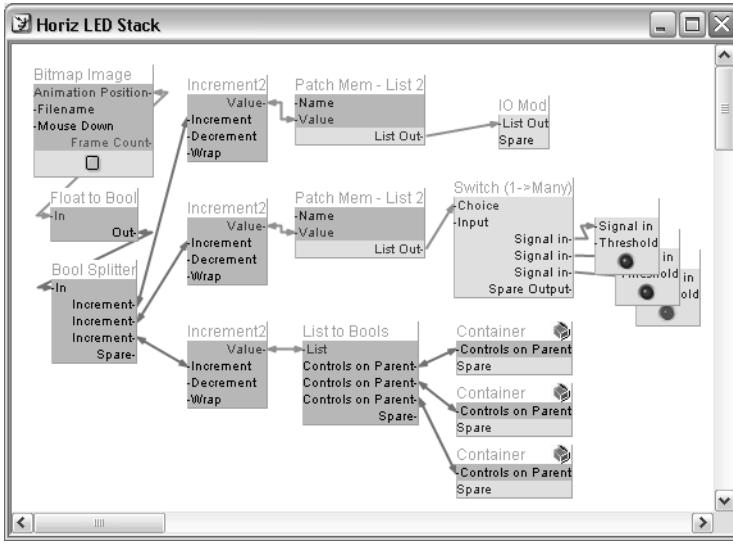


Figure 5.79

Extending the Sub-control Toolkit

SynthEdit's on-board sub-controls only scratch the surface of what sub-controls can do. A varied cast of developers has created oodles of third-party sub-controls. These modules give you mind-boggling options for customizing the your SynthEdit creations' GUIs.

This section surveys the range of third-party sub-controls available at the time of writing. These include modules by this chapter's author, Dave Haupt (DH), as well as Rob Herder (RH), Butch Kratzer (BK), Simonluca Laitempergher (SL), Oli Larkin (OL), and Kelly Lynch (KDL). Rather than providing full documentation of every sub-control, we aim to let you know what modules are out there, and the amazing capabilities they put into your hands. To learn more about a given module, see the help files provided with the modules and the developer's website (more on this on www.wizoobooks.com/synthedit).

Recall that for the purpose of discussion, we broke native SynthEdit sub-controls down into six functional groups. You'll come across the same categories here, with added sub-categories to do the diversity of third-party sub-controls justice. These finer classifications should make it easier to find what you are seeking when you reference this chapter later, as we hope you will. Again, the main categories are:

- Data Manipulation
- Data Type Conversion
- GUI Input/Output
- Parameter Interface
- Routing
- Miscellaneous

Data Manipulation Modules

These modules adjust the value of a GUI variable. They break down into four sub-categories:

- Format Conversion
- Numeric and Logical Operations
- Float Array Processing
- Text/List Manipulation

Format Conversion

These modules convert values from one format or type of unit to another. They are great for expressing GUI control values in terms of familiar or user-friendly units, such as Hz or dB. Having the GUI thread do the converting spares CPU power.

DH Color Format Converters

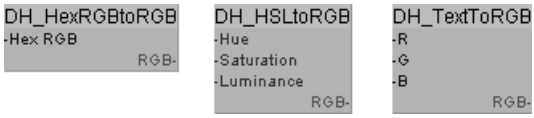


Figure 5.80

Some Input/Output sub-controls have plugs that adjust text and background colors dynamically. They accept an integer value representing a composite of the color's red, green, and blue (RGB). The modules pictured in figure 5.80 convert several common color specifications to this integer RGB format:

- The hexadecimal format for RGB values that you recognize from HTML.
- Hue, Saturation, and Luminance, as used in many paint programs.

❖ Values scaled from 0 to 255 for the red, green, and blue components in text form.

DH_dBToVolts2, DH_FloatExpCurve, DH HzToVolts2, DH_msToVolts2

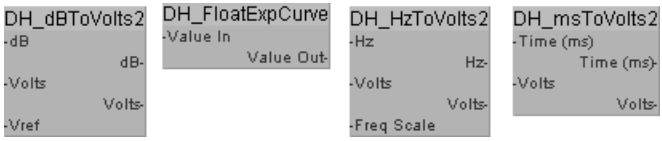


Figure 5.81

SynthEdit parameters generally comprise control voltages ranging from 0 to 10 on a linear scale. These modules let you calibrate GUI controls in decibels, Hertz, and milliseconds, and convert between scaled logarithmic or exponential responses in a range of 0 to 10. All conversions go both ways. See the section "Routing Modules" on page 267 for a technique that reverses the flow through DH_FloatExpCurve to elicit a logarithmic response.

SL Non-linear Scalers

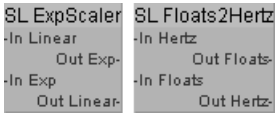


Figure 5.82

SL ExpScaler implements the formula:

$$\text{output} = \text{base}^{((\text{multiply} * \text{input}) + \text{add})}$$

and its inverse for conversions between linear and exponential scales over the same input and output range. Enter the range's minimum and maximum values, and the *base*, *multiply*, and *add* parameters to the module's right-click Properties. SL_Floats2Hertz converts floats to Hertz. It also lets you use reference points for the volt/octave scale other than the 5 volts = 440 Hz option SynthEdit offers.

Numeric and Logical Operations

This group performs basic arithmetic and other numeric or logical operations on GUI values. Again, using modules to calculate control values in the GUI thread saves CPU on the time-critical DSP side. Bear in mind, though, that GUI processing and DSP are separate threads, meaning GUI and audio events are not in sync. Don't use numeric operations sub-controls in an audio processing chain, or to control voltages with critical timing requirements.

BK_ListToBools2

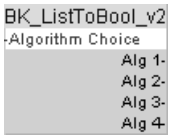


Figure 5.83

There's more to some modules than their appearances would have you think. Though this module's plug names tell you it was designed for algorithm selection, its basic function comes in handy in other situations. The input is a standard (non-GUI) list entry that lets you set the four Boolean outputs' logical states. The output matching the selected list item is True; the others are False. If you want to control panel selection using one of the variants of SynthEdit's List Entry control, this may be just what the good doctor ordered.

DH_FloatIncrement

Figure 5.84

This module adds or subtracts one from a current float value. You can set it up to wrap around when it reaches a specified range's high and low value.

DH_FloatCeil, DH_FloatFloor, DH_FloatQuantizer

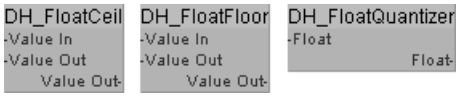


Figure 5.15

All three of these modules cleave the fraction off a floating point number, leaving an integer. DH_Float Ceil converts to the next higher integer, DH_FloatFloor converts to the next lower integer, and DH_Float-Quantizer rounds to the nearest integer.

DH_FloatAbs

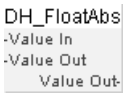


Figure 5.86

DH_FloatAbs renders the absolute value of its input. Positive values remain positive; negative values become positive.

DH Arithmetic Modules

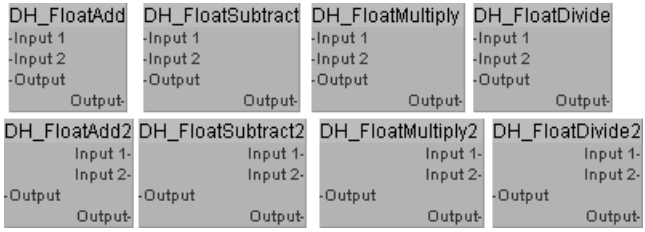


Figure 5.87

The modules in this group do simple arithmetic on float values. Those in figure 5.87's second row provide right-side input, a convenient option for some circuits.

DH_FloatCompare, DH_IntCompare

These modules compare two float or integer values, and depending on the result, set one of three Boolean outputs to True.

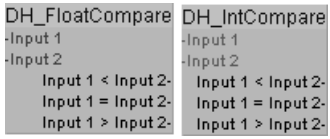


Figure 5.88

DH_FloatMin, DH_FloatMax

You can use DH_FloatMax and DH_FloatMin to find the maximum or minimum of any number of float values. Connecting one input plug automatically creates another.

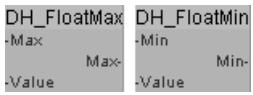


Figure 5.89

DH_FloatToDigits

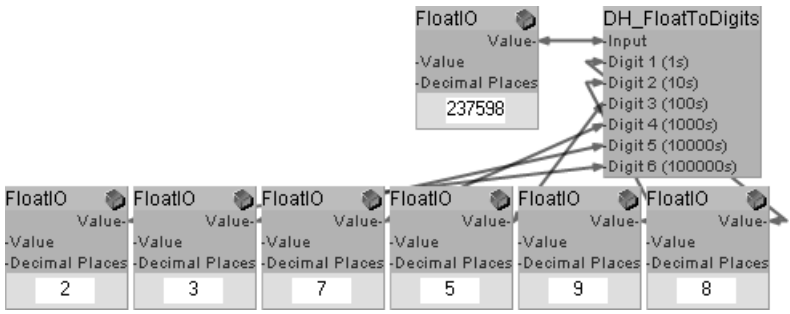


Figure 5.90

DH_FloatToDigits issues the individual digits of a whole number up to six digits, either as float values as shown in figure 5.90, or as animation positions. The latter lets you use a ten-frame graphic image comprising the digits 0 to 9 in any font in Bitmap Image modules to build bespoke dynamic readouts. For more on this, see the DH_Character-BitmapDriver in the section "Text/List Manipulation" on page 247.

DH_ModulusOp

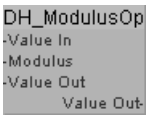


Figure 5.91

The modulus operator divides an integer by a second integer called the modulus, and returns the remainder. It often serves to generate repeating sequences, for instance:

0 mod 3 = 0	3 mod 3 = 0	6 mod 3 = 0
1 mod 3 = 1	4 mod 3 = 1	7 mod 3 = 1
2 mod 3 = 2	5 mod 3 = 2	8 mod 3 = 2

DH_ModulusOp performs this operation on its two inputs.

KDL Animation Controls

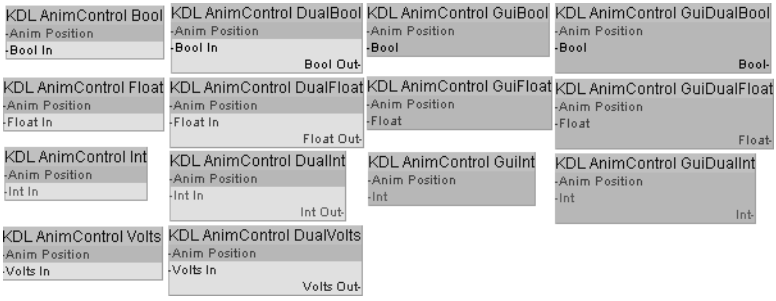


Figure 5.92

These modules take values of any input range specified by entering Min and Max values into right-click Properties fields, and normalize the output to GUI float values ranging from 0.0 to 1.0 as required for animation with Bitmap Image modules.

For example, if an LFO with a peak-to-peak voltage of –5 to +5 volts provides the Volts In input of a KDL AnimControl Volts module, set Min to –5 and Max to 5. The Anim Position's output is then 0.0 at a –5 input value, and 1.0 at +5 input value.

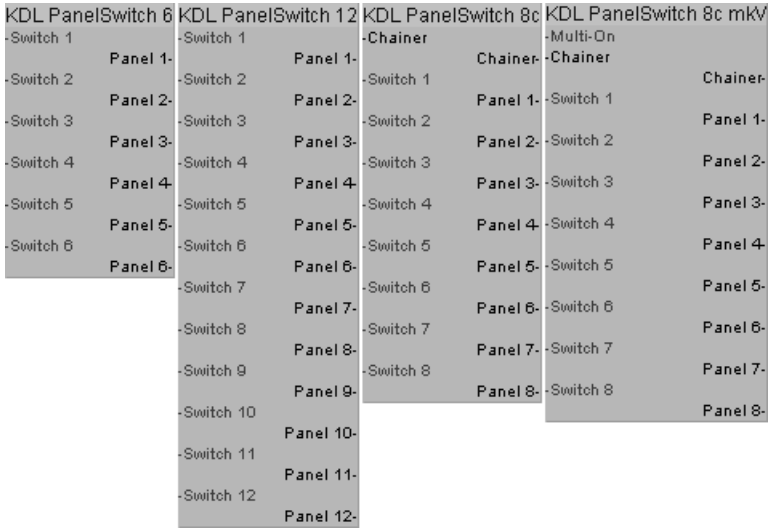


Figure 5.93

KDL Panel Switches

KDL Panel Switches afford users easy pushbutton control for paged panels. When a Switch takes on a value other than 0, the corresponding Bool type Panel output changes to True, and the other inputs and outputs change to 0 and False, respectively. The one exception is that multiple panels may be selected with the KDL PanelSwitch 8c mkV's Multi-On option enabled. You can daisy-chain 8c and 8c mkV versions to support additional panels.

OL Random Float

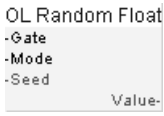


Figure 5.94

OL Random Float is useful for randomizing controls. Its two modes:

- Generate a random number between 0.0 and 1.0
- Randomly choose between a specified High and Low value

RH_2D_3D_Distance

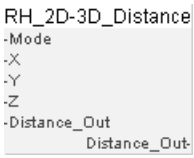


Figure 5.95

This module calculates the Pythagorean distance formula in two or three dimensions, giving the distance from the origin (0,0) to a point (X,Y) in 2D, or from the origin (0,0,0) to a point (X,Y,Z) in 3D.

RH_Bitcount Set

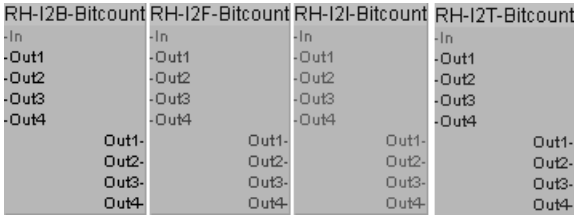


Figure 5.96

These 4-bit binary counters accept integers from 0 to 15, outputting binary representations of the input value. Out1 is the msb; Out 4 is the lsb.

RH_Float_Lcompare

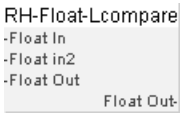


Figure 5.97

This mod compares Float In with Float In2. If they are equal, it sets Float Out to 5; otherwise Float Out is 0.

RH_Int-Simple_Logic

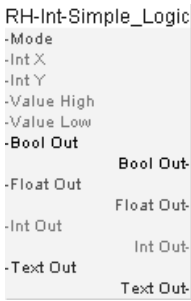


Figure 5.98

This mod compares Int X and Int Y plugs' values based on the relationship determined by the Mode plug, which may be:

X == Y, X != Y, X > Y, X < Y, X ≥ Y, or X ≤ Y

A value of True sets Bool Out and Text Out to True, and Float Out and Int Out to Value High. False sets Bool Out and Text Out to False, and Float Out and Int Out to Value Low.

RH Logic_Gates

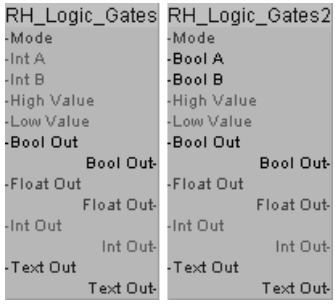


Figure 5.99

These modules implement the basic binary logic gate types for integer (0 or 1) and Boolean (False or True) inputs. The gate types are:

AND, OR, NAND, NOR, XOR, and XNOR

A value of True for the gate output sets Bool Out and Text Out to True, and Float Out and Int Out to High Value. False sets Bool Out and Text Out to False, and Float Out and Int Out to Low Value.

RH_Rescale

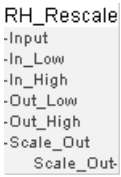


Figure 5.100

RH_Rescale translates and rescales its Input from the input range provided by In_Low and In_High, to the output range provided by Out_Low and Out_High, using a linear transformation.

SL FixedGui Series

Figure 5.101

Use these modules when you need the equivalent of SynthEdit's Fixed Values on the GUI side. Enter the desired constant to the Value right-click Properties field, and it will go to both Out plugs.

SL Gui Limiters

Figure 5.102

Use these to clip GUI float or integer values to a specified range. Enter Min and Max values to the right-click Properties fields.

Float Array Processing

This set of modules interfaces between multi-valued GUI Input/Output sub-controls, such as DH_MultiStepInput (see the section "GUI Input/Output Modules" on page 258), and other modules that work with one floating point value at a time.

DH_ArraySequencer

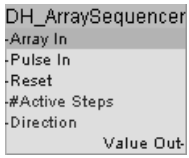


Figure 5.103

DH_ArraySequencer responds to control voltage pulses by taking an array of float values from a multiple-input GUI Input/Output sub-control and spitting them out one by one as control voltages in a repeating sequence.

Though array values are updated on the GUI side in response to the user's mouse movements, the DSP side receives the input pulses, and sequences the output. This means the sequence and audio event processing may be synchronized.

DH_ArrayToFloats

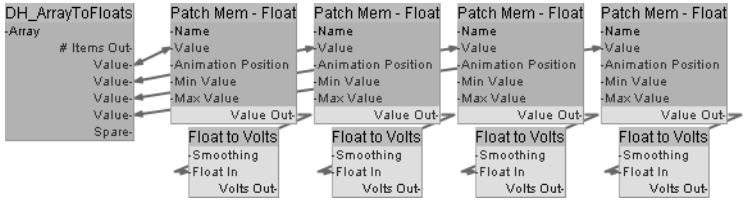


Figure 5.104

DH_ArrayToFloats provides simultaneous access to all array values. Its output plug automatically clones itself on demand.

DH_FloatArray

Figure 5.105

DH_FloatArray provides indexed access to array values. Indexing array values from 0 to 1 less than the number of values in the array, it supports both read and write access.

The module also loads arrays from and saves them to a text file.

If a value changes or a new array loads from a file, the Array In plug issues the new values so the GUI Input/Output sub-control reflects the new values.

Text/List Manipulation

This group's sub-controls let you endow your GUI with sophisticated dynamic lists and other text features. They also enable handy functions like selecting or constructing file names on the fly.

DH_CharacterBitmapDriver

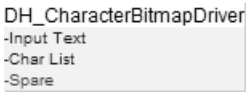
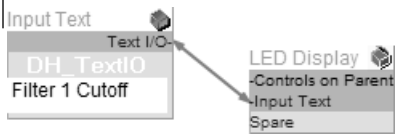


Figure 5.106

Say you wish to build customized dynamic text display controls. Now say you want to use fonts that need not be installed on the end user's machine. This module gets the job done. The result can look like the LED Display prefab in figure 5.107's structure view, and in figure 5.108's panel view.



Figures 5.107 and 5.108

To use the module, you must first create a multi-frame graphic containing all the characters in the font you aim to use, as in figure 5.109. The graphic's text file is surprisingly simple:

```
type animated
frame_size 14, 20
; extra space at: top, bottom, left, right
padding 0, 0, 0, 0
```

Add to the module a character list containing the same characters in the same order as they appear in the graphic. Check out figure 5.110; it shows the LED Display container's plumbing.

Figure 5.109

The display comprises a series of Bitmap Image modules, one for each character in the display, all of which use the character graphic file. Each Bitmap Image's Animation Position plug connects to a Spare plug on the DH_CharacterBitmapDriver. The Spare plugs automatically replicate on demand. Arrange the Bitmap Images horizontally or vertically in the panel view to lay out the display. The display then spells out any text fed into the DH_CharacterBitmapDriver's Input Text plug.

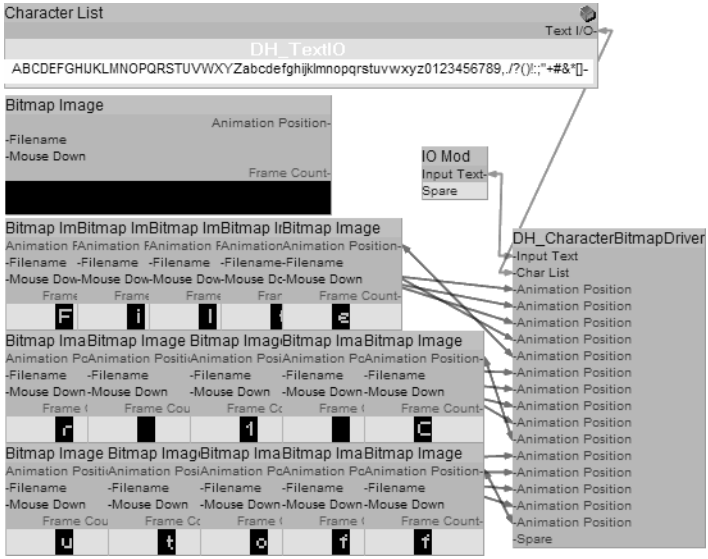


Figure 5.110

DHFormat, DH_FloatFormat

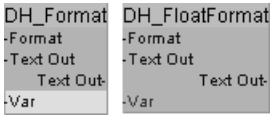


Figure 5.111

DH_Format and DH_FloatFormat let you label numeric values with text characters, or insert numeric characters into a text string. Text feeds into the Format plug; numeric values into one or more Var plugs.

Embed special format specification characters in the text to mark the places where you wish to insert numeric values. You can have as many Var inputs as you have format specifications in the format string. The Var plug replicates automatically on demand. The module assigns numeric input values from the Var plugs top down to the format specifications from left to right in the text.

A format specification starts with a “%” and ends with a letter f. In between the two, specify the minimum field width for the number, and the displayed number of decimal places.

For example, the format specification `%3.2f` displays the number 3.14159 as 3.14. If a Var input is 2300, a format string `%6.0f Hz` displays it as " 2300 Hz". The module's documentation provides lots more insight into formatting capabilities.

DH_LeftString, DH_RightString, DH_SubString

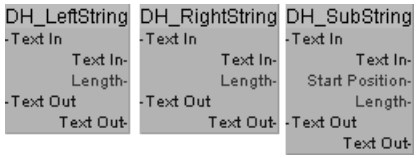


Figure 5.112

These modules extract a substring of a specified length from the left, right, or a specified start position in a text string.

DH_StringCompare

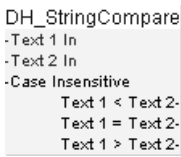


Figure 5.113

DH_StringCompare compares two strings and depending on the result, sets one of its Boolean outputs to True. Performed in alphabetical order, comparisons may or may not consider whether characters are uppercase or lowercase.

DH_StringLength, DH_StringSearch

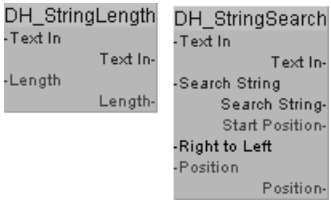


Figure 5.114

DH_StringLength provides the length of a text string: DH_StringSearch a specified search string's position. Commencing at any position in the string, searches move left to right or right to left.

DH_TextAppend

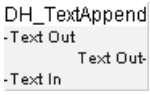


Figure 5.115

This module simply appends any number of text strings. The Text In plug automatically clones itself on demand.

DH_TextArray, DH_TextList, DHTextList2

Figure 5.116

All three of these modules let you load, manipulate, and store lists of text items using simple text files. DH_TextArray uses a zero-based numeric index to access the list items. DH_TextList and DH_TextList2 afford list access at their Selection plugs, which connect to a Dropdown List or other GUI Input/Output list selection sub-control. Use them with lists of patch names, file names, modulation sources, or destinations—anywhere you need a custom list.

DH_TextArray and DH_TextList enable dynamic list updating. You can set up either module to write changes to a file read by several other DH_TextArrays and DH_TextLists elsewhere in the circuit. More than one module should not write to the same file.

DH_ListExtractor

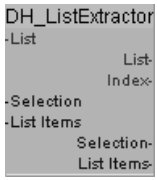


Figure 5.117

DH_ListExtractor issues the index and text value of a list's current selection. It also provides the entire list as a text string, with list items separated by commas.

DH_ListGenerator

Figure 5.118

This module offers a simple method of creating a sequential numeric list dynamically. You can specify minimum and maximum values and the increment, as well as ascending or descending order. The output and list ranges need not necessarily agree. If the high and low values you set for the output and list differ, the module interpolates from one range to the other.

DH_ListSearch

Figure 5.119

DH_ListSearch searches a list for a specified search string. It reports the number of items containing the string, as well as their list indexes. If more than one item contains the search string, the Find Next and Find Prev plugs step forward and back to the nearest strings.

DH_ListStandardizer

Figure 5.120

Custom lists may include special formatting characters serving to control advanced features of DH_PopupMenu (see "GUI Input/Output Modules" on page 258). DH_ListStandardizer removes formatting information so these lists work with standard SynthEdit GUI Input/Output list selection sub-controls such as the Dropdown List.

KDL GuiList2 ... every data type

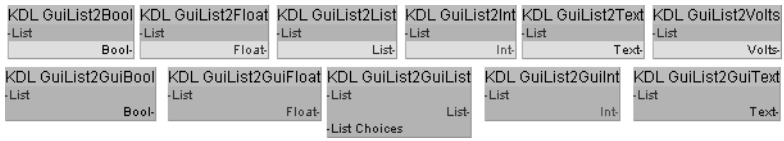


Figure 5.121

You can use KDL conversion modules that convert from GUI lists to other data types to construct custom lists. Use a comma-separated list to enter the list items and their values into the List Choices box in the module's right-click Properties, as shown in figure 5.122.

The format for each item is:

list item = value

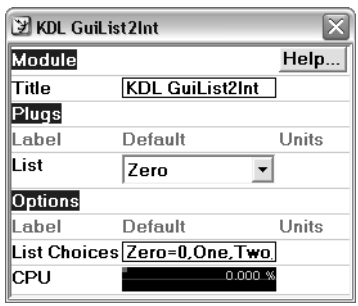


Figure 5.122

The list item is what you want displayed in the list; the value is what you want the module to issue when the item is selected.

Each item's value is True or False when converting to a Boolean value. Say you're dealing with a list of the first eight harmonics. True could signify harmonics that are the same note; False the others:

Fund = True, 2nd = True, 3rd = False, 4th = True, 5th = False, 6th = False, 7th = False, 8th = True

If you don't specify an item's value with an = sign when converting to numeric values, the item automatically adopts a value one higher than the preceding value, starting at 0.

KDL modules that convert GUI lists to text values accept any text string to the right of the = sign. For example, a list's items could be patch names and their values the names of files containing the samples.

KDL GuiText2GuiText

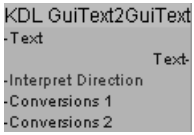


Figure 5.123

This module provides easy-to-use text substitution capabilities. Enter a list of conversions in the format quarter note = crotchet, eighth note = quaver, and so forth, and it will convert from left to right, right to left, or both ways if you wish.

OL Animation Position to List2

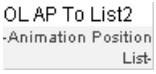


Figure 5.124

This module lets you use a Bitmap Image to cycle through a GUI list's options.

RH_Int2Text-MidiCC

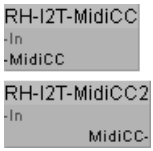


Figure 5.125

Fed the MIDI CC number, these modules issue a text string showing the standard assignment for the CC number, or just the number if the CC number is unassigned.

RH_Int2Text-ascii

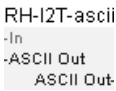


Figure 5.126

Integers go in, text characters come out. Here's the code:

Uppercase: 0 = A to 25 = Z
Lowercase: 32 = a to 57 = z

Note: This module does not convert from standard ASCII codes to ASCII characters. To do this, first subtract 65 from the ASCII code, and then feed the result into the module.

Data Type Conversion Modules

These modules' mission is converting one data type to another, while retaining the same or an equivalent value. Rather than listing them all, suffice it to say that there are sub-controls for converting to and from every GUI data type, and most DSP data types. Kelly Lynch developed a comprehensive set, some of which appear in figure 5.127. If data type conversion is possible, rest assured some KDL module is around to do it.

Some KDL conversion modules do much more than merely convert data types. We discussed these in other categories, emphasizing their added capabilities.

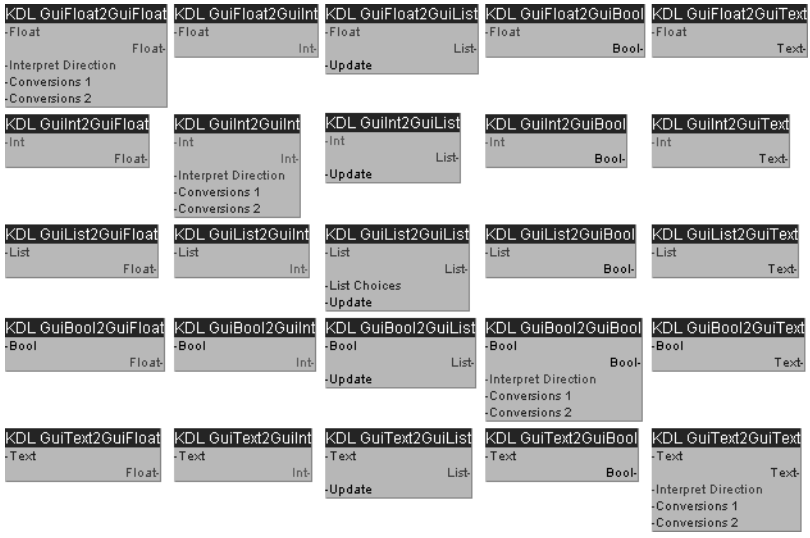


Figure 5.127

KDL modules offer many conversion options, for example:

- ❖ All GUI-to-GUI converters offer a choice of direction, converting from left to right, right to left, in both directions, and with or without changes feeding back to the originating plug. This is good because it lets you compensate for the quirks of data flow among GUI plugs.
- ❖ Conversion sub-controls with voltage inputs let you specify the sampling rate and method.
- ❖ Converters with text inputs let you define word lists for conversion to specific output values.
- ❖ GUI List outputs may be selected either by value or index.

Dave Haupt also developed a few data type conversion modules. These sub-controls are simple, no-frills modules that work from left to right and right to left. Oli Larkin's straightforward OL List2SingleBool and OL SingleBool2List modules convert an On/Off list to a True/False Boolean value, with the option of inverting the relationship.

GUI Input/Output Modules

These sub-controls appear on your GUI for the user to employ when handling your VST/VSTi plug-ins.

Note that their function is confined to feeding in and sending out GUI data. You must combine them with other sub-controls to create full-fledged controls. To this end, you will need a Patch Mem module for each value you wish to store and recall as a parameter.

GUI Input/Output sub-controls come in three basic types:

- List Selection
- Numeric I/O
- Text I/O

List Selection

DH_DropdownList

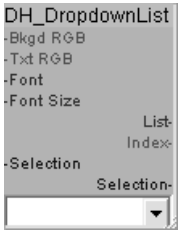


Figure 5.128

An alternative to SynthEdit's Dropdown List sub-control, this module provides a resizable window, and the dynamic color and font selection choices common to DH GUI I/O sub-controls. Figure 5.129 shows an example panel view.

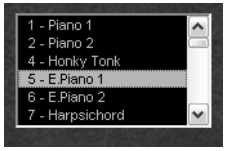
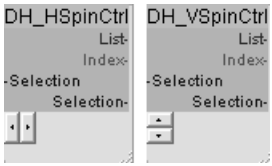
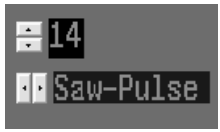


Figure 5.129

DH_HSpinCtrl, DH_VSpinCtrl Horizontal and vertical spin controls serve to step through a list or a specified sequence of numbers in either direction, with optional wrap-around at the end of the list or sequence. Selection plugs issue the currently selected item's text, which Text Entry2 or DH_TextDisplay can display, as shown in figure 5.131.





Figures 5.130 and 5.131

DH_ListBox DH_ListBox offers another GUI list selection option. It features a resizable window, dynamic color and font selection, and automatic vertical scrolling if the list is too long for display in the window.

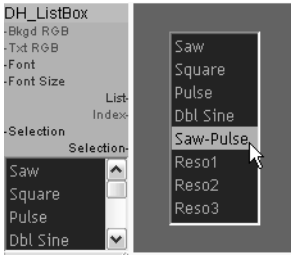


Figure 5.132

DH_PopupListBox DH_PopupListBox provides a list box-type selection sub-control that displays only the current selection until it is clicked (see figure 5.133).

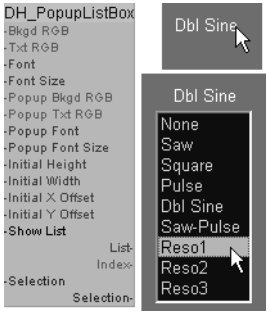
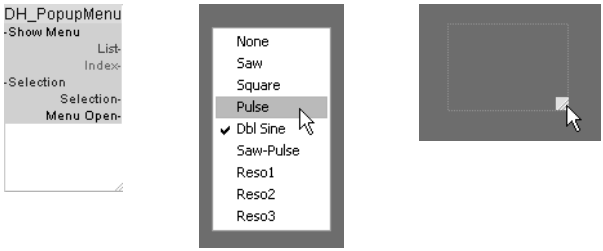


Figure 5.133

It offers dynamic color and font selection for both the text display/click area and the popup list box.

You can opt to fix the pop-up's position, or create a floating window. Use the horizontal and vertical offsets to specify its position relative to the text display/click area.

DH_PopupMenu



Figures 5.134, 5.135 and 5.136

This list selection sub-control pops up a standard Windows menu when its Show Menu plug triggers (see figure 5.135). The module appears on the panel view as a transparent rectangle (see figure 5.136). Clicking the rectangle also displays the menu.

You can create a formatted menu with multiple columns, vertical and horizontal dividers, and non-selectable labels. To do this, build a custom list with DH_TextList, DH_TextList2 or one of the KDL GuiList2<any> modules, and insert special formatting characters into it. Figure 5.137 provides an example. See the DH_PopupMenu documentation in the DH_Sub-ControlPak User's Guide for details on special formatting characters.

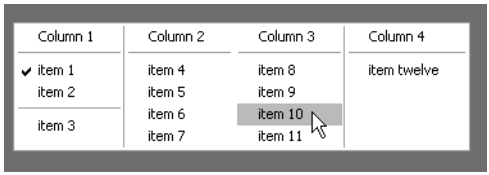


Figure 5.137

Numeric I/O

DH_Breakpoint Input DH_BreakpointInput is a graphic sub-control for entering and displaying two-dimensional data such as an envelope's amplitude and time. It feeds out horizontal (X) values and vertical (Y) values as two parallel arrays accessed by the modules described in the section "Float Array Processing" on page 246.

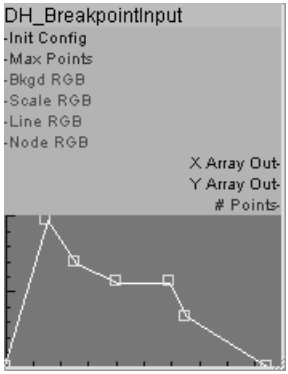


Figure 5.138

Double-click to create more breakpoint nodes; double-click an existing node to delete it. You can lock the number of nodes, as well as the first and last nodes' positions. The breakpoint display handles output as well as input. If the values in the X or Y arrays change, the nodes move accordingly. Node positions may be saved and restored with the patch.

The resizable graphic area scrolls and zooms horizontally. You can select colors for background, lines, nodes, and scale dynamically.

DH_ColorPicker

Figure 5.139

DH_ColorPicker lets you use a standard Windows color chooser dialog (see figure 5.140) to select GUI Input/Output sub-controls' color. It provides the selected color in text form as separate red, green, and blue values with ranges of 0 to 255, and as a composite RGB integer value of the type accepted by most DH GUI Input/Output sub-controls.

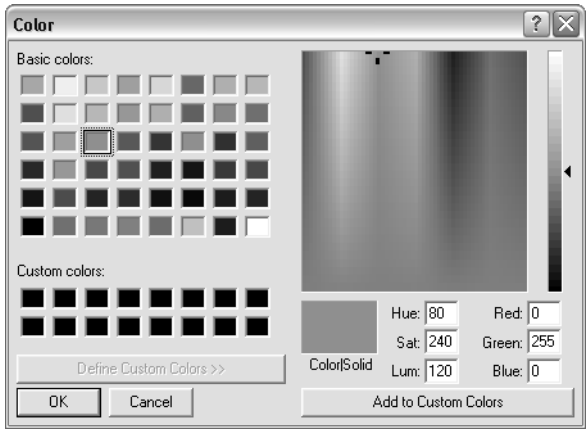


Figure 5.140

DH_ContourInput DH_ContourInput provides a resizable GUI sub-control for entering and displaying a series of float values. The height of a vertical line segment signifies each value. It sends out an array accessed by the modules described in the section "Float Array Processing" on page 246.

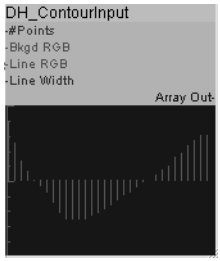
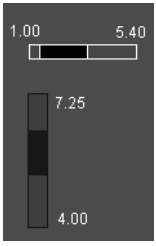


Figure 5.141

Dragging the cursor across the Contour display window adjusts the segments' heights. You can center the baseline as shown in figure 5.141, or place it at the display's bottom or top. The module handles output as well as input. The displayed contour changes to reflect changing array values.

DH_MinMaxBar



Figures 5.142 and 5.143

DH_MinMaxBar lets you enter and display a value range intuitively by sight. Orient the bar vertically or horizontally. You can adjust the high and low values separately, but they may not overlap. Holding the Shift key down while dragging the mouse moves minimum and maximum in tandem. The module handles input and output. The GUI adjusts to reflect Min and Max Value plugs' changing values.

DH_MultiStepInput

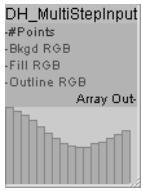


Figure 5.144

DH_MultiStepInput is another GUI I/O sub-control for entering and displaying a series of float values accessed by the modules described in the section "Float Array Processing" on page 246. A bar's height represents the given value. The # Points plug determines the number of bars. Hold the Ctrl key down while dragging a bar's height to make fine adjustments. DH_MultiStepInput works much like DH_ContourInput. Choose the one with graphics better suited for its target parameters. Again, the module handles output and input. The bars move to reflect array value changes.

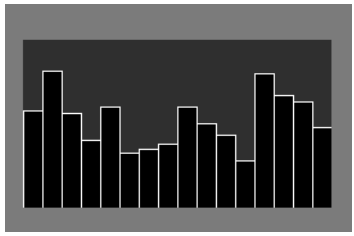


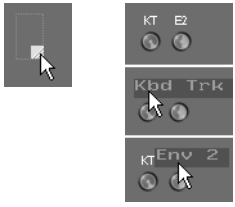
Figure 5.145

Text I/O

DH_PopupTextDisplay DH_PopupTextDisplay initially appears on the panel as a transparent rectangle, as in figure 5.147. Click it to view its Text plug's value in the selected font and colors.

Figure 5.146

The transparent rectangles for the two DH_PopupTextDisplays in figure 5.148 overlay the knob controls' labels, so that clicking KT or E2 pops up the longer labels from the DH_PopupTextDisplays.



Figures 5.147 and 5.148

DH_TextDisplay

Figure 5.149

DH_TextDisplay displays read-only text strings in the selected font and colors. For a transparent background, set the Bkgd RGB plug to –1.

DH_TextIO

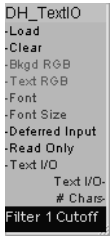


Figure 5.150

DH_TextIO handles text input and output. Simply type to enter text; unlike SE Text Entry2, DH_TextIO does not require you to press the Enter key. It displays text in the selected colors and font. DH_TextIO is always ready to accept typed input, so it remains on top regardless of its and other modules' To Front or To Back settings, and you cannot set its background to transparent.

Parameter Interface Modules

These modules' primary purpose is to store and retrieve sets of values to and from patch storage.

DH_PatchArray Modules

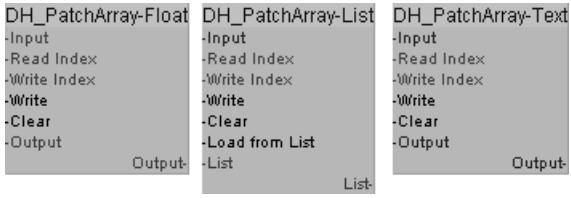


Figure 5.151

DH_PatchArray-Float stores and retrieves numeric values by index. DH_PatchArray-List stores and retrieves text-based values in an indexed list. DH_PatchArray-Text stores and retrieves text-based values by index. All three save values in either of two modes:

- Global mode saves a single float array, list, or text array that does not change when the patch is changed.

❖ Per Patch mode provides a separate float array, list, or text array for each patch.

DH_PatchArray-List has a Load From List plug that lets you populate the list with items from the list of another sub-control connected to the right-side List plug.

Memory for the array or list is allocated automatically as needed. Currently, patch memory stores up to 1,000 characters.

Routing Modules

Routing sub-controls fall into two sub-groups:

❖ Simple Routing Modules

❖ Route Switches

Simple Routing Modules

Simple routing modules let you to connect sub-controls in cases where SynthEdit's rules normally prevent direct connection.

DH Splitters, DH_TextRedirector

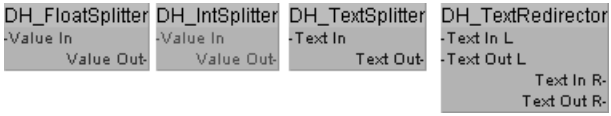
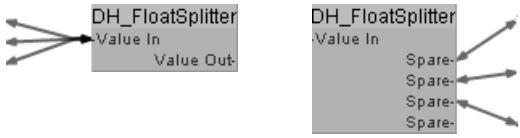


Figure 5.152

SynthEdit defines left-side plugs as inputs, and right-side plugs as outputs, and prohibits you from connecting inputs to inputs and outputs to outputs. As discussed in the section "It Goes Both Ways—Data Flow and Animation" on page 201, GUI data flows both ways, so left-side plugs are not necessarily inputs, nor are right-side plugs necessarily outputs. To further complicate routing, left-side GUI plugs are masters that accept multiple connections (see figure 5.153). Those on the right are slaves that connect to one master only. You need a splitter to route a GUI value from a plug on a module's right to several destinations (see figure 5.154).



Figures 5.153 and 5.154

Splitters can also reverse the direction of flow. For example, figure 5.155 shows how splitters reverse the flow through a DH_FloatExpCurve to obtain the inverse of an exponential function—a logarithmic function.

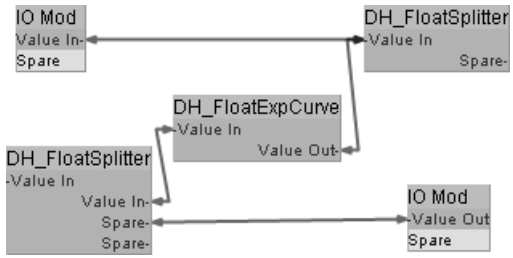


Figure 5.155

RH Redirect Set

Figure 5.156

These modules redirect the input to the output on the same side to enable left-to-left and right-to-right connections.

SL Gui Splitter Series

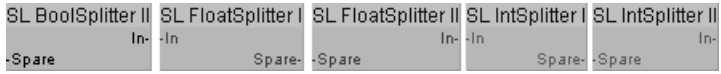


Figure 5.157

Another set of splitters, these include right and left hand versions.

Route Switches

These modules let you route a source to one of several destinations, one of several sources to a destination, or control data flow over a given route.

DH_Route Switches

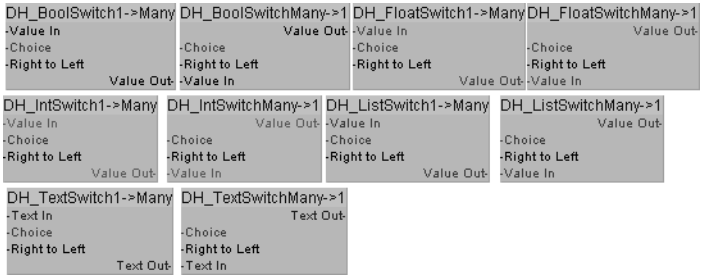


Figure 5.158

DH Route switches come in one-to-many and many-to-one configurations for each of the GUI data types (see figure 5.158). Because of GUI connections' bidirectional nature, each of the one-to-many switches also serves as a right-to-left many-to-one switch, and each many-to-one switch can do double duty as a right-to-left one-to-many switch.

KDL DiscoSwitch bf

Figure 5.159

This module acts as a single-pole, double-throw (SPDT) toggle switch that routes the input Value to the Off output when the Bool plug is False, and to the On output when the Bool plug is True. The unselected output retains its previous value when the switch's status changes.

OL Control Reset2

Figure 5.160

When the Gate triggers, this module sends the Reset Value to the Value output plug. In a typical application, this plug connects to the Value plug of a Patch Mem–Float.

OL Float Gate

Figure 5.161

The OL Float Gate switches a float connection, routing the value of the In plug through to the Out plug while the Gate is True. Switching the Gate on, off, or in either direction can serve to send a Reset Value, depending on the Reset Mode.

SL SliderLinker

Figure 5.162

As the name suggests, this module is handy for linking and unlinking two sub-controls. When the Link plug is True, the Slider 1 plugs connect to the Slider 2 plugs; False disconnects them.

Miscellaneous Modules

These sub-controls either serve special purposes, or don't fall neatly into our main categories. They include three sub-groups:

- MIDI
- File Handling
- Other

MIDI

Kelly Lynch's conversion modules include several sub-control modules providing MIDI processing interfaces. They entail more than merely simple data type conversions, so they merit discussion as a separate group. If you are contemplating using these modules, bear in mind that GUI and audio are separate processes defying perfect synchronization. Though many other KDL MIDI modules convert to and from non-GUI data types, you won't find them discussed here because they are beyond the scope of sub-controls.

KDL GuiFloat2MIDI, KDL GuiInt2MIDI, KDL GuiList2MIDI

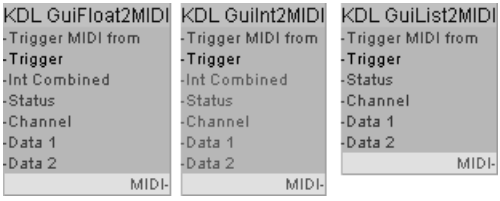


Figure 5.163

This set of modules generates a MIDI message based on the information sub-controls provide for each of a MIDI message's components:

- Status
- Channel
- Data 1
- Data 2

KDL GuiFloat2MIDI and KDL GuiInt2MIDI can also accept all three bytes of the MIDI message. A Boolean trigger or an input value change initiates the message.

KDL MIDI2GuiBool, KDL MIDI2GuiFloat, KDL MIDI2GuiInt, KDL MIDI2GuiText

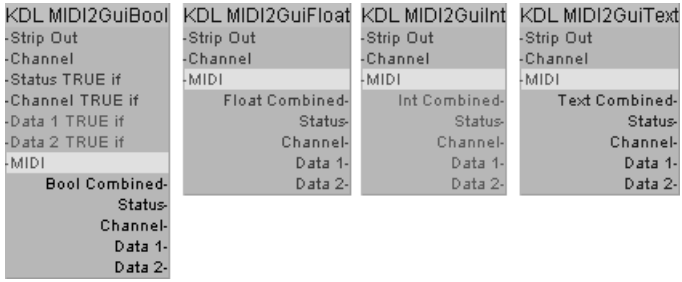


Figure 5.164

These modules deliver data to other sub-controls based on incoming MIDI messages. KDL MIDI2GuiBool's True/False output values depend on whether the components of the MIDI message match the specified values. The other modules' output values comprise the message and its components' float, integer, or text representations.

File Handling

These modules let you add custom file browsing and selection features to your GUI.

DH_FileList, DH_FileList2

Figure 5.165

DH_FileList creates a file selection list in a specified directory. DH_FileList2 also enables browsing in other directories.

Other

If a module didn't fit comfortably into any of our categories, it ended up in this group.

DH_ControlMerger2

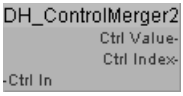


Figure 5.166

DH_ControlMerger2 accepts input from multiple sub-controls, and feeds out the current value and index of the most recently adjusted sub-control. Ctrl In plugs replicate automatically on demand. Starting with 0, it indexes connections from the top down.

DH_ControlTrigger2, DH_CtrlTriggeredTimer

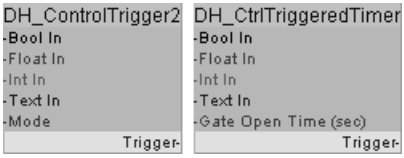


Figure 5.167

DH_ControlTrigger2 sends a one-sample pulse when an input value changes. Output values may be inverted so that it is normally high with a zero pulse. DH_CtrlTriggeredTimer sends a pulse of a specified duration when an input value changes.

SL_FloatAnimator

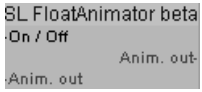


Figure 5.168

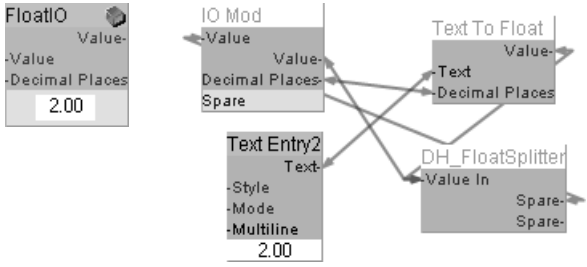
This module issues incrementing and decrementing (rounded) floats at a specified rate. It runs in the GUI thread, so 20 frames per second is the fastest it will go. And it may not be precise, especially at higher frame rates. Enter the number of frames, mode (forward, reverse, alternating), looping (on/off), and rate (frames per second) as parameters in the module's right-click Properties.

More Hands-on Examples

That's it for our tour of third-party sub-controls. Now let's look at some of the ways we can use them. These examples illustrate techniques and familiarize you with sub-controls so you can design prefab controls of your own. Hands-on learning is the best method of acquiring the skills you need, so be sure to try the examples and experiment freely.

FloatIO Prefab

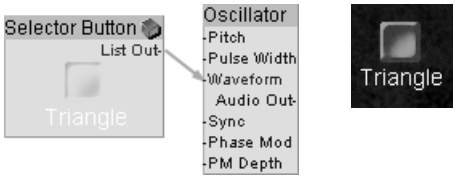
The first example is a simple prefab useful for entering and displaying float values as shown in figure 5.169. Conveniently, it plugs into a float value on the left or right. Figure 5.170 shows the prefab's simple structure comprising a Text Entry2 connected to a Text To Float. Note how the DH_FloatSplitter connects the Text To Float's Value plug to both right- and left-side plugs.



Figures 5.169 and 5.170

Custom Selector Button Redux

Back in "Putting Your Sub-control Skills into Practice" on page 220, we built a selector button prefab using static text labels inside containers, and switched the containers' Controls on Parent inputs on and off to show the selected item (see figures 5.74 to 5.76). There is a better solution—dynamic labeling based on the list to which the selector actually connects. Case in point: Say you connect the selector button to the SE oscillator's Waveform plug, and want the label to display the selected waveform from the list as shown in figures 5.171 and 5.172.



Figures 5.171 and 5.172

To do this, you must find a way to extract the currently selected item's name from the list, which is what a DH_ListExtractor does. Connect it to the Value plug of the Patch Mem–List 2 (see figure 5.173). We used a Text Entry2 to display the selection. You could do the same or opt for a DH_TextDisplay, depending on which offers the colors and fonts you prefer.

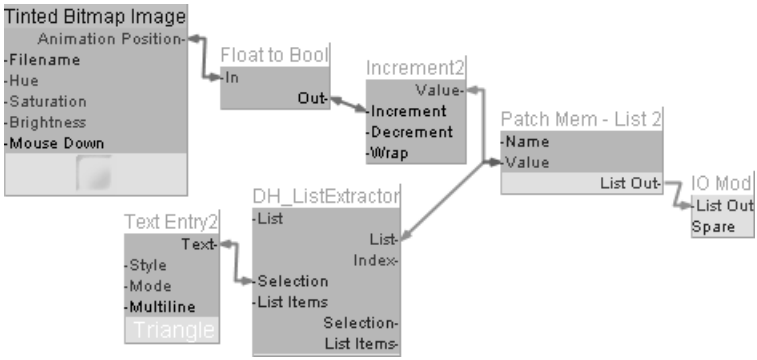


Figure 5.173

You could also connect Bitmap Images to both the Increment and Decrement plugs of Increment2 to step through lists in both directions, as shown in figures 5.174 and 5.175.

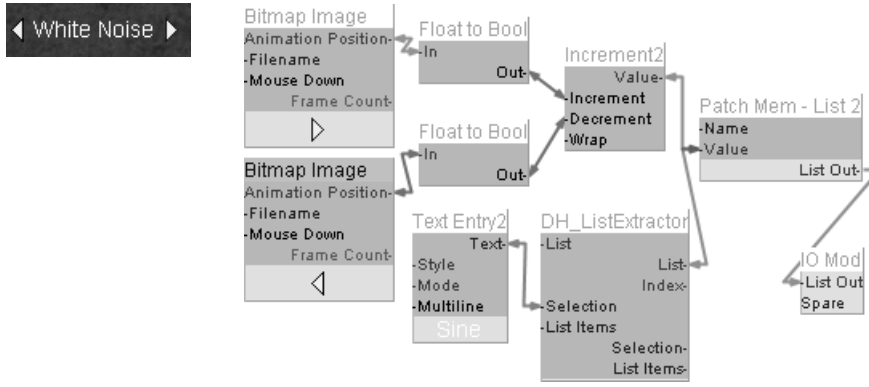


Figure 5.174 and 5.175

Here's another alternative if the standard Windows spinner controls fit into your design scheme: Replace the Bitmap Image and Increment2 sub-controls with a DH_VSpinCtrl or a DH_HSpinCtrl as depicted in figures 5.176 and 5.177.

Figures 5.176 and 5.177

File Name Extractor

This example demonstrates some options for manipulating text. We'll build a prefab to extract just the name of a file for display, without the path or file extension, from a string containing the full file path.

Here's our example text string:

C:\audio files\loops\drum\100bpm*100bpm funk groove 4*.wav

The name we aim to extract is shown in italic in the string above—*100bpm funk groove*. To do this, take the substring beginning one character after the last backslash (\), and set its length so it cuts off characters to the right of the last dot (.). This breaks down into a three-step procedure, or algorithm:

1. Search right to left from the end of the string to find the rightmost backslash's position, and add one to obtain the substring's starting position.
2. Search right to left from the end of the string to find the position of the rightmost dot (.), and subtract one to pinpoint the position of the last character you want to extract. The position's value is also the length of the total string, minus the length of the dot (.) and everything to its right.
3. To obtain only the length of the desired substring, subtract from the result of step 2 the number of characters up to and including the last backslash (\). You calculated the rightmost backslash's position in step one, so all you need to do now is subtract it from step 2's result.

Design the circuit step by step, following the algorithm. Figure 5.178 shows you how to implement step 1.

The full path string flows into DH_StringLength, DH_StringSearch, and DH_SubString. The Search String is a backslash (\). Set the Right to Left value of the DH_StringSearch to True in its right-click Properties. The Start Position is the end of the string because DH_StringLength sets it to the string's length.

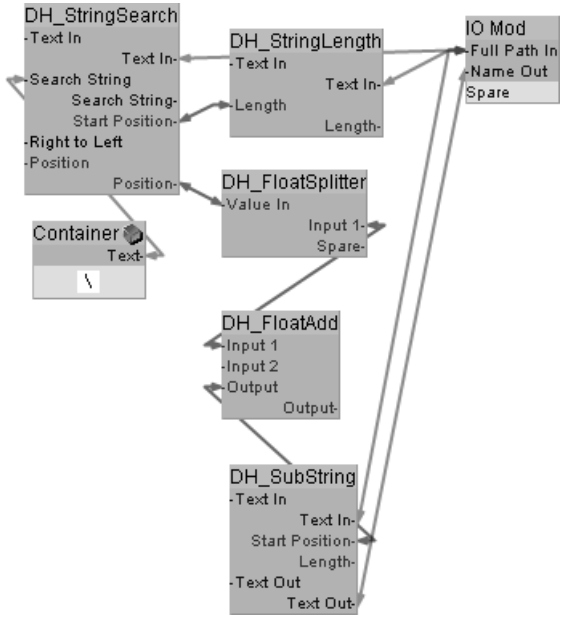


Figure 5.178

DH_SearchString's output goes to a DH_FloatSplitter so that we can use it both here and in step three.

DH_FloatAdd adds one to the position found by DH_SearchString, and the sum provides the Start Position input value for DH_SubString.

Figure 5.179 shows a second DH_StringSearch added to search for the dot (.) from step 2. Again, Right to Left is set to True, and the start position is set to the string's length so the search begins at the string's end.

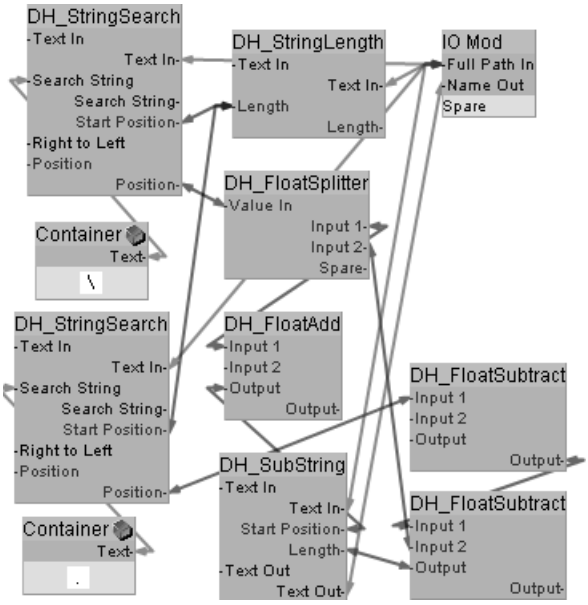


Figure 5.179

DH_FloatSubtract subtracts one from the position of the dot (.) to find the final character's position, completing step 2.

In step 3, the bottom right DH_FloatSubtract subtracts the rightmost backslash's position from the final character's position to obtain the length for the DH_SubString module.

Figure 5.180 shows an example of the prefab in use.

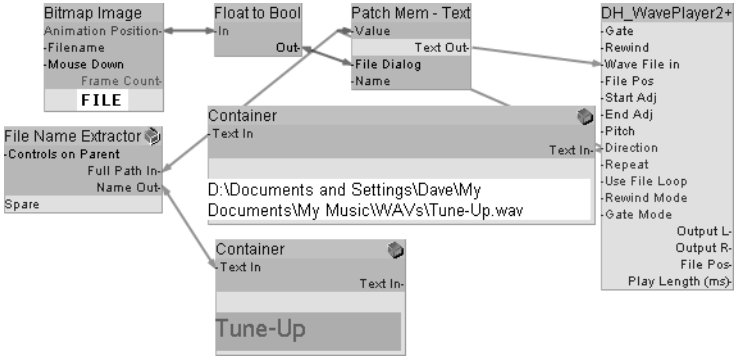


Figure 5.180

Real-time Color Controls

Why not let your users get creative and customize the colors of your GUI's features with controls like these two? The first uses three knobs that dial in the amounts of red, green, and blue in the color (see figure 5.181). Note the DH_IntSplitter; it sends the RGB value to multiple destinations. Figure 5.182 maps the structure.

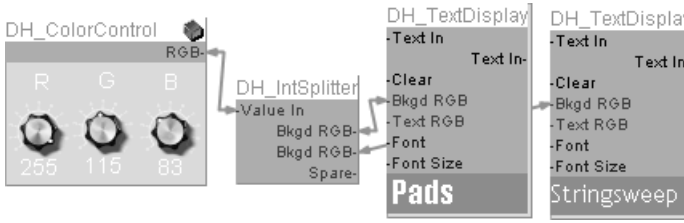


Figure 5.181

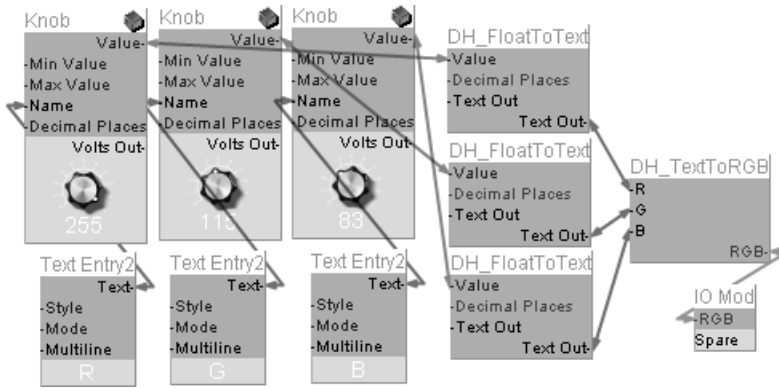


Figure 5.182

For the knobs, we adapted the Knob Sm prefab from the SE Insert:Controls menu to provide float output values. Figure 5.183 shows the revised Knob structure. Connected to plugs on the Patch Mem–Float inside the knob structure, the knob's Min and Max Values range from 0 to 255. Figure 5.182 shows how a DH_FloatToText converts each float value into text, and how DH_TextToRGB combines them to create a composite integer RGB value.

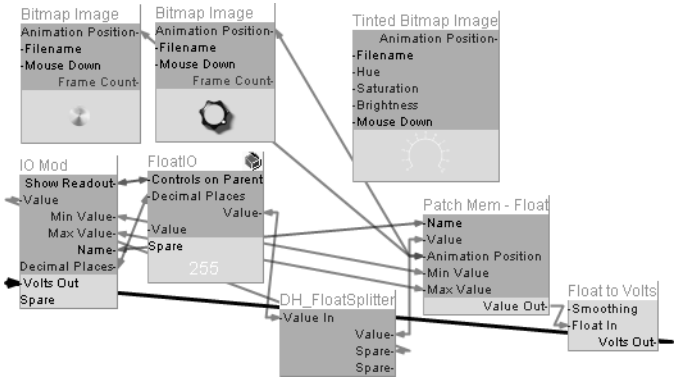


Figure 5.183

The second color control uses a DH_MultiStepInput to create a GUI feature with three bars for hue, saturation, and luminance. The control itself changes color as the bars are adjusted (see figure 5.184). The DH_MultiStepInput's #Points is set to 3 to display three bars, and it feeds out an array of three float values. A DH_ArrayToFloats converts the array to three separate float values.

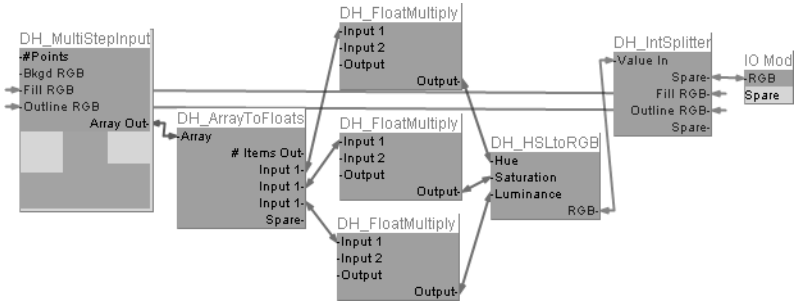


Figure 5.184

Each DH_MultiStepInput bar's output value ranges from 0 to 10, which must be scaled to the correct range for DH_HSLtoRGB. With several choices of input range available, we opted for 0 to 1, which entails dividing the MultiStepInput's output values by 10. Division consumes far more CPU power than multiplication, so let's multiply by 0.1 instead of dividing by 10. The two are mathematically equivalent, but the former is so much more efficient. DH_HSLtoRGB converts the hue, saturation, and luminance values to composite integer RGB for-

mat. A DH_IntSplitter routes this to the output, and back to the DH_MultiStepInput's Fill RGB and Outline RGB plugs so bars will change to reflect the color specified by HSL values. Note that in contrast to audio circuits, GUI circuits permit direct feedback.

Quantized Tuning Knob

Synths typically provide coarse and fine-tuning controls for oscillators, with the coarse control quantized to semitones (half steps), and the fine control covering the range between two half steps. SynthEdit uses a one volt = one octave pitch scale, with 5 volts = A440. The difference between two half steps is always 1/12th, or approximately 0.083333 volts, which makes the math of quantizing to semitones easy. So all we need to do is quantize our coarse knob's output value to the nearest 12th.

Say you want to use this control for an audible oscillator rather than an LFO. This means you need only work with pitch values ranging from 0 to 10. This range comprises 120 half-steps. The knob's Bitmap Image modules' Animation Position ranges from 0 to 1, so multiply it by 120. This quantizes the knob's position to 120 discrete integer values. The fine-tuning knob's and the quantized coarse knob's output values add up, so the floor function—which is the highest integer not greater than the input value—provides the quantization we need. It simply truncates any fractional part of positive numbers. Once quantized, values must be rescaled to the 0-to-10 range, so we'll divide by 12. For any Animation Position in the 0-to-1 range, this yields the corresponding number in the 0-to-10 range, rounded down to the next lower 12th.

Figure 5.185 shows the structure of our Quantized Knob prefab. DH_FloatMultiply multiplies the Animation Position by 120. DH_FloatFloor rounds the product down to the next lower integer. DH_FloatDivide divides by 12 to convert our 120 discrete values from the scale of 0 to 120 to the scale of 0 to 10. Note that the result goes to the Patch Mem–Float's Value plug rather than the Animation Position plug because we have already done the scaling.

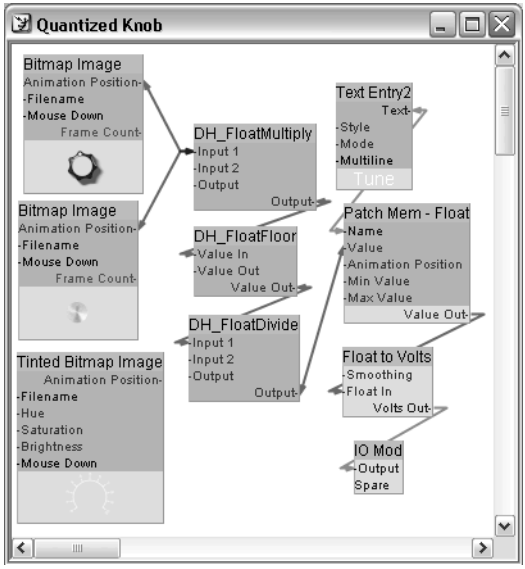


Figure 5.185

4-panel Osc Selector

Figure 5.186

In this example, you'll construct a switchable panel display for controlling four oscillators. It employs the previous example's quantized tuning knob. Your results should look something like figure 5.186. All four oscillators sport the same controls. A skinned version of the standard SE List Entry control with its Appearance set to Button Stack serves to select an oscillator for editing.

First, let's build a prefab providing a set of controls for one oscillator. It comprises our quantized tuning knob, a fine-tuning knob, a DH_PopupListBox to select the waveform, and a DH_TextDisplay to label the oscillator. Figure 5.187 depicts the structure.

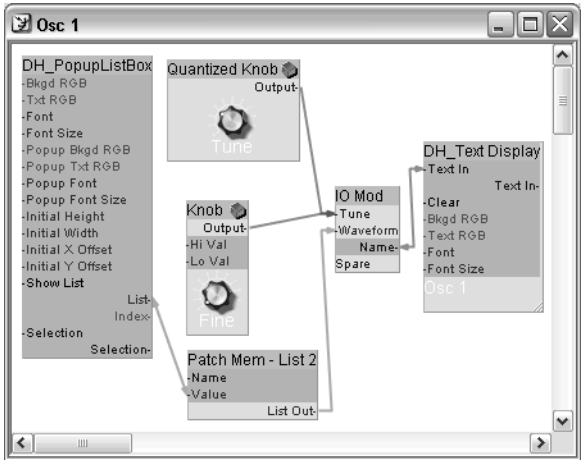
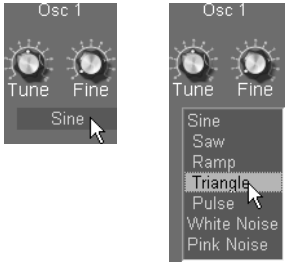


Figure 5.187

A standard knob prefab from the Insert:Controls menu will do for fine-tuning. Set its high value to 0.083333 (1/12th), and its low value to 0 so its full range is one semitone. Its value is added to the quantized knob's output value. We opted for a DH_PopupListBox because it displays the current selection while taking up little space when not in use, and expands to show all of the available options (in this case) when clicked (see figures 5.188 and 5.189). Note that DH_TextDisply and DH_PopupListBox's titles are blanked out, and do not appear on the panel view. The DH_TextDisplay Text In plug connects to the IO Mod so a Name field appears on the prefab's Properties. This makes it easy to change the label without having to unlock the prefab or open its structure view.



Figures 5.188 and 5.189

All that remains to be done now is to replicate the oscillator controls prefab thrice, change their Name fields, and arrange them for switching. We picked BK_ListToBool to handle the switching because it connects direct to a standard SE List Entry, and we wanted to use our skinned version of this control's button stack rendition. Conventional SE controls and sub-controls are not necessarily mutually exclusive. If a conventional control satisfies one of your demands, feel free to mix and match it with sub-controls as you see fit.

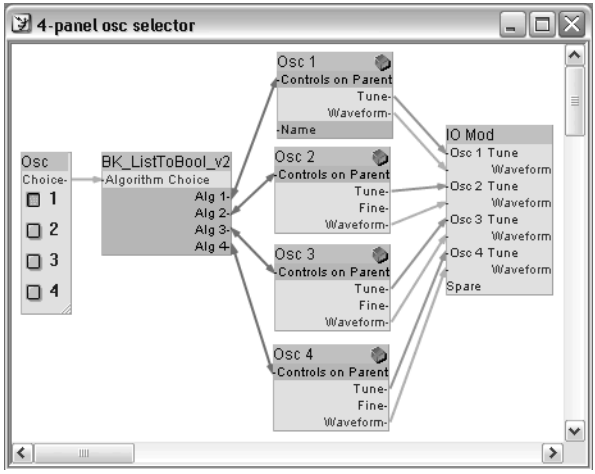
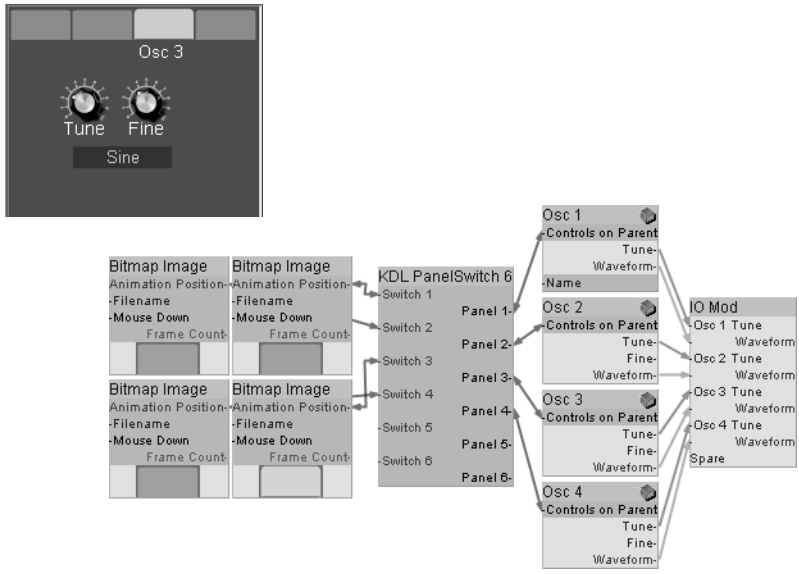


Figure 5.190

Figure 5.190 pictures the 4-panel osc selector's overall structure, yielding the same panel view as in figure 5.186. It takes a little effort to line up the four sets of controls perfectly in the panel view so that they don't shift about when you switch from one oscillator to another. Once in line, they'll serve your users well.

Of course, this osc panel selector could use any selection method that sets one Boolean output to True and all others to False. You could use one of the KDL PanelSwitch modules with Bitmap Images to create a tab-like effect as shown in figure 5.191. Here, the DH_TextDisplays on every oscillator control set prefab's panel view are sited so each appears below its tab when the tab is selected. Figure 5.192 shows this structure.



Figures 5.191 and 5.192

Using One Control Readout

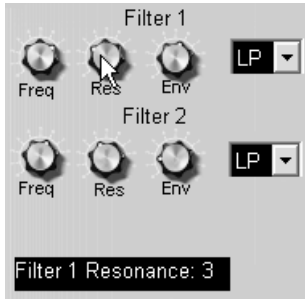


Figure 5.193

Providing a single panel to display the currently adjusted control's name and value conserves screen space and makes it easier for users to focus on this info. Figure 5.193 depicts an example where the currently adjusted control is Filter 1's resonance knob, and the central readout indicates as much.

Figure 5.194 shows one way to set this panel up. DH_ControlMerger2 accepts multiple float inputs, and feeds out the most recently adjusted value and index. Though we wired up just the three controls for this example, this suffices for demonstrating how this works. The index is converted to an integer, and used to access an item in the list managed by DH_TextList2. Pre-loaded from a text file, the list contains entries for the three controls currently connected to DH_ControlMerger2:

```
Filter 1 Cutoff;
Filter 1 Resonance;
Filter 1 Env;
```

List items are indexed 0, 1, and 2. DH_ControlMerger2 indexes inputs from the top down starting at 0, so moving the center knob yields an index of one.

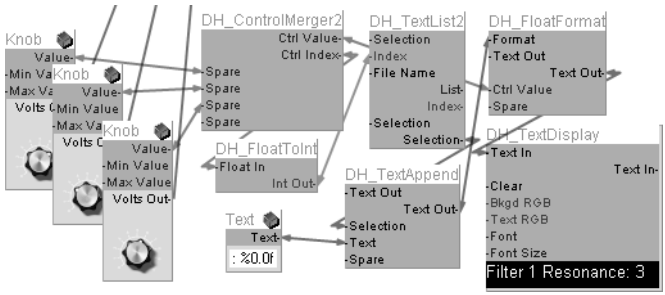


Figure 5.194

Given that index, DH_TextList2 selects Filter 1 Resonance, and sends this text string to its Selection plugs. The string feeds DH_TextAppend, providing the input for the first item. The text in the little box appends the string, yielding Filter 1 Resonance: `%0.0f`. This text then goes to the DH_FloatFormat Format plug.

DH_FloatFormat routes the Format string unchanged, apart from `%0.0f`. This is a format specification that tells DH_FloatFormat to substitute the value at the Ctrl Value plug, at that position in the text string. The 0.0 tells it to use neither a minimum field width, nor any digits after the (implied) decimal point. DH_ControlMerger2's Ctrl Value provides the control value to DH_FloatFormat. This is the filter 1 resonance knob's current value, so the number changes as the knob moves.

The knobs provide values in the standard SynthEdit range of 0 to 10. So, how can you display the cutoff frequency in Hz, and the percentage amount (0 to 100 %) by which the envelope modulates the cutoff? You'll find an answer in figure 5.195. A DH_HzToVolts2 sits between the filter 1 cutoff knob and DH_ControlMerger2. Able to convert both ways, here this module converts the 0-to-10 voltage scale to Hertz. All it takes to convert the filter's Env knob's value to percent is to multiply by 10, so a DH_FloatMultiply is on board to do this job. Note that this affects GUI values only. The controls' Volts Out plugs still send standard 0-to-10 volt signals to the DSP modules.

Converting numbers is not enough, though. You also need to add the unit of measure, Hz, to the cutoff frequency, and a percent sign (%)to the envelope modulation amount. We added another DH_TextList2 to do this. It preloads with a text file containing:

```
 Hz;
;
%%;
```

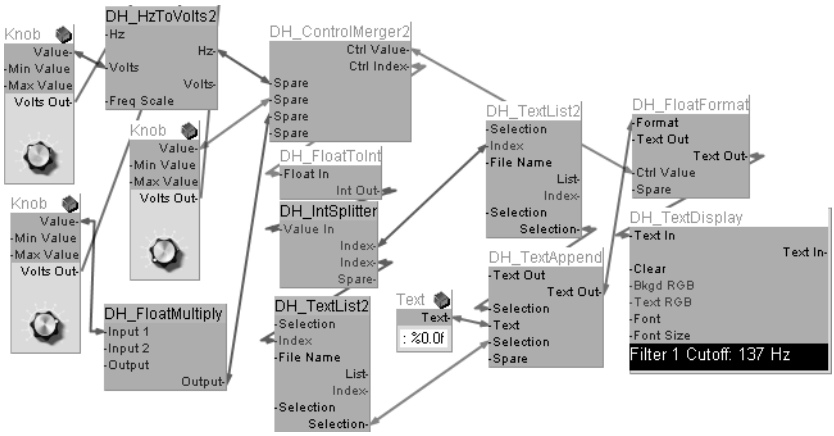


Figure 5.195

This list uses the same index as the first. The first line is " Hz;" with a space as the first character to provide a space between the frequency number and the Hz unit abbreviation. Resonance amount requires no adjunct, so the second item is a line with a blank space followed by a

semicolon. Use two signs (%%) to append percentages because DH_FloatFormat interprets a single % as the beginning of a format specification. Tag the second DH_TextList2's output value onto the end by connecting it to the next available plug on DH_TextAppend.

Note that the tiny text box used for format specification is nothing but a containerized Text Entry2 with Controls on Module set to True, and Controls on Parent set to False to hide it from the panel view.

Graphic MIDI Control Indicator

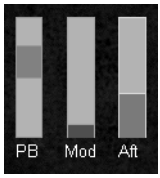


Figure 5.196

For our next little trick, we'll build a graphic indicator to display the status of three MIDI controls—the pitchbend wheel, mod wheel, and channel aftertouch. Its vertical bar's heights indicate mod wheel and aftertouch amounts. A vertical bar indicates the pitch wheel's amount of positive action above the midline, and negative bend below it (see figure 5.196).

Let's start by building the simplest indicator component first—the mod wheel display—and save the most complex for last. We'll use the DH_MinMaxBar for the display. Set the orientation to Vertical and Min Value to 0. Max Value will control the height of the bar.

Our plan of attack is to:

- Isolate mod wheel messages from the MIDI stream
- Extract float values representing the modulation amount
- Scale these values to fit DH_MinMaxBar's 0-to-10 range

First filter everything except mod wheel messages from the MIDI stream using a DH_MIDIFilter+. Mod wheel messages are sent as MIDI CC01 messages, with the controller number in the first data byte, so set the Status to Control Change, and Data1 Min and Data1 Max to one as shown in figure 5.197.

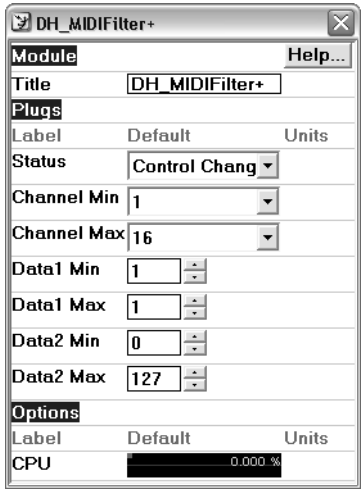


Figure 5.197

A KDL MIDI2GuiFloat extracts the modulation amounts from CC01 messages, providing them as float values. The messages' second data byte indicates the modulation amount as a value ranging from 0 to 127. KDL MIDI2GuiFloat's Data 2 plug issues this as a float value.

DH_FloatDivide then divides the amount by 127/10 = 12.7 to scale it to the 0-to-10 range.

Figure 5.198 shows the full circuit for the mod wheel. Note how the DH_FloatSplitter connects DH_FloatDivide's output to DH_MinMax-Bar's MaxValue plug, both of which are right-side plugs.

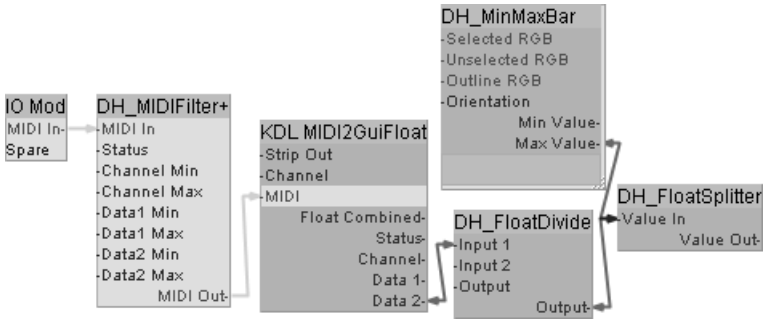


Figure 5.198

Now let's tackle the channel aftertouch indicator. Though very similar to the mod wheel indicator, it requires a few changes:

- Change the status of DH_MIDIFilter+ to Channel Aftertouch.
- Set Data1 Min to 0 and Data1 Max to 127 because the first data byte indicates the aftertouch amount.
- Use the KDL MIDI2GuiFloat's Data 1 output rather than the Data2 output.

The pitch wheel indicator poses a bigger challenge because the bar must clear the midline for an upward bend, and dip below it for a downward bend. This means you have to configure DH_MinMaxBar's inputs so that when the wheel turns past the center position, Max Value ranges from 5 to 10, with Min Value remaining fixed at 5. Likewise, when the wheel turns past the center position in the other direction, Max Value must remain fixed at 5, while Min Value ranges from 5 down to 0.

Also bear in mind that pitch bend messages use both data bytes. The first data byte holds the value's least significant byte (LSB), and the second data byte holds the most significant byte (MSB). MIDI bytes employ 7 bits representing 128 different values from 0 to 127, so a pitch bend message's value is 128 times the MSB value plus the LSB value.

Let's get the ball rolling by formatting the pitch bend value as a float value ranging from 0 to 10. Again, a DH_MIDIFilter+ filters out everything apart from the messages of interest, in this case, pitch bend messages. The defined status is Pitch Bend, and both Data1 and Data2 are set to the full range of 0 to 127.

Here's how to combine the LSB and MSB: Use a DH_FloatMultiply to multiply the KDL MIDI2GuiFloat's Data 1 output value (the MSB) by 128 and a DH_FloatAdd to add Data 2's output value (the LSB). Note that unlike voltages, which add up automatically when connected to the same point, GUI values must be added explicitly using an add module. The result is then divided by $(127 * 128 + 127)/10 = 1638.3$ to scale it down to the 0-to-10 range. Figure 5.199 shows the circuit we have constructed so far.

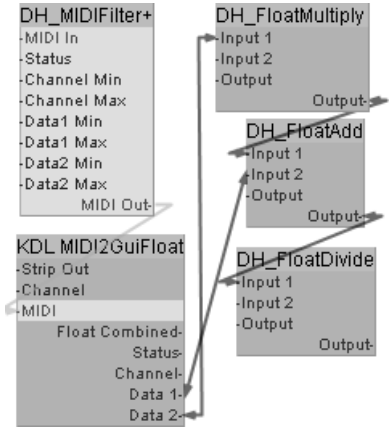


Figure 5.199

Now let's see how we can make DH_MaxMinBar's Min Value range from 0 to 5 when the scaled pitch bend value is less than 5, and equal to 5 when the pitchbend value is greater than 5. In other words, we want the Min Value to be the smaller of the current value or 5. And we want Max Value to range from 5 to 10 for values above 5, and remain at 5 for lower values, so it must equal the larger of the current value or 5.

In figure 5.200, DH_FloatDivide's scaled output splits up, feeding a DH_FloatMin and a DH_FloatMax, each of which receives a constant 5 as another input. DH_FloatMin and DH_FloatMax's outputs provide the desired inputs for DH_MaxMinBar's Min Value and Max Value plugs.

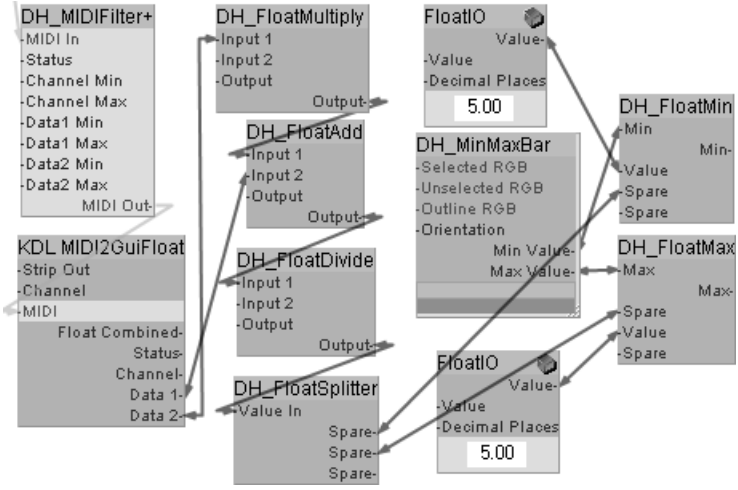


Figure 5.200

The Future of Sub-Controls

Sub-controls afford us so much greater flexibility in building SynthEdit GUIs. And the SynthEdit SDK makes it so easy for third parties to create sub-controls, putting a wealth of powerful tools into the hands of SynthEdit developers. With ever more programmers developing sub-control modules, sub-controls' number and variety will surely continue to grow.

SynthEdit's creator, Jeff McClintock, notes some key changes are in the pipeline that will make sub-controls easier to use in SynthEdit 1.1 and future versions. Redesign efforts aim to alleviate some of native SE sub-controls' annoying little quirks, like having to use a splitter because the plug you want to connect to sits on the wrong side. Lists will be made more flexible, enabling us to work with list text as text and selections as integers. The new version of the SDK will afford programmers even greater leeway to develop new and different kinds of modules. Jeff intends to ensure the application remains compatible with legacy third-party modules, so you can enjoy new goodies without sacrificing the great tried-and-true tools featured on these pages.

Appendix

A Brief History of Synthesizers

A Moog Minimoog (photo by Stefan Hund, www.emc-de.com)

Bob Moog unveiled the *Minimoog* in 1970, marking the first real milestone in the history of synthesizers. Wieldy enough for musicians to play on stage, the Minimoog had the added benefit of bearing the definitive synthesizer pioneer's name. It wasn't the first of the breed. Mammoth modular systems requiring musicians to connect modules with patch cords merely to create a single sound predated it. The hassle of taming these beasts proved too much for all but a few musicians and bands, among them '70s-era sonic adventurers Keith Emerson, Walter nee Wendy Carlos, Tangerine Dream, and Klaus Schulze. The Minimoog wasn't even the first pre-wired, compact synthesizer. The *EMS VCS 3*, embraced by artists such as Hawkwind, Yes, and Brian Eno, preceded it in 1969.

The Minimoog's basic design provides the template for the classic synth. It sported three VCOs with one doing double duty as a modulating LFO, a mixer section, a low-pass filter with resonance, and two envelope generators, one for the filter and one for amplifier). Competitors soon brought to market variations on the theme. *ARP* released the *Odyssey* in 1972, adding oscillator synchronization, sample and hold, and a high-pass filter to the sonic equation. The semi-modular *ARP 2600* followed in 1971. Though portable and pre-wired, it still offered some atavistic external cable connections.

This archetypal monophonic analog synthesizer was produced until the early '80s. Featuring four VCOs, two LFOs, oscillator synchronization, cross-modulation, two envelope generators, and four-voice polyphony, the *Korg MP-4 Mono/Poly* was the most versatile of the bunch, at least in terms of functions.

1978 saw the birth of fully programmable polyphonic analog synthesizers, with the *Prophet 5* by *Sequential Circuits* being the first of the kind, followed by the *Oberheim OB* series in 1979, the *Korg Trident* and the *Roland Jupiter 8* in 1980, and a year later the *Moog Memorymoog*. These held a lot more appeal than their monophonic forebears, particularly for pros. Streamlined versions of programmable polyphonic synthesizers hit the streets in 1982, among them the *Korg Polysix* and *Roland Juno 60*. Though limited to a solitary oscillator, they did ship with a sub-oscillator and an affordable price tag. As engineers added more digital control features on the inside, fewer knobs and sliders remained on the outside, replaced by data entry keys, buttons, knobs, and program selectors. This wave washed to shore the *Korg Poly 61*, debuting in 1983, followed in 1984 by the *Sequential Sixtrak*, the first multi-timbral synthesizer, and the *Korg Poly 800*.

The turn of that decade also saw the rise of a new generation of musical instrument—the sampler. Though it featured some synth functions, samplers enabled users to record waveforms of their very own to the machine. First came the *Fairlight CMI* (Computer Musical Instrument) and the *NED Synclavier* in 1979, and the *EMU Emulator* in 1980. Their price tags put them far out of mere mortals' reach.

Another species of synthesizer emerged in late 1983 bringing to the world a new type of synthesis called frequency modulation, or FM for short. The *Yamaha DX7* birthed a different brand of sound, bringing a jaw-dropping 16 voices to bear, versus typical polyphonic analog synthesizer's paltry six or eight, at best. The DX7 excelled at emulating the chiming sounds of magnetic pianos such as the Fender Rhodes and Wurlitzer electric piano favored by many keyboardists. This upped its marketability measurably, making it an attractive proposition for musicians who otherwise wouldn't have given a second thought to synthesizer. As digital sound generators raced up the sales charts, they crossed paths with analog synthesizers on their way down. This gradual but temporary decline bottomed out in the early '90s.

Another class of synthesizer floated up out of the primordial sound-generating soup in the mid '80s. Given the unflattering nickname *romplers*—a morpheme of ROM and samplers—devices like the *Korg DW-8000* (1985), *Kawai K3* (1986) and *Ensoniq ESQ1* (1987) played wave-

forms stored digitally in ROM chips, yet processed sounds using analog filters and the like. The *Sequential Prophet VS* in 1986 and the *PPG Wave 2* in 1981 brought other forms of synthesis to the world. Those in the know called the Prophet VS a vector synthesizer, for it could toggle among four waveforms. The PPG Wave 2 was dubbed a wavetable synthesizer for its ability to scan through a sequence of waveforms stored in ROM. However imaginative their names, they were in essence subtractive synthesis-driven romplers. The *Waldorf MicroWave* (1989), *Wave* (1995), and later *MicroWave II* (1997)/*XT* (1998) took up where the PPG synths left off.

Two more types of synthesis from this era bear mentioning. *Casio* developed phase modulation for their *CZ* (1985) and *VZ* (1988) models, while the *Kawai K5* (1987) based on additive synthesis. Like FM synthesizers, they lacked the classic subtractive sound-sculpting capability that comes courtesy of filters.

The *Roland D-50* (1987) and the *Korg M1* (1988) went the whole digital hog. Though their filters, effects, and all other sound-shaping components were digital, they continued to take the subtractive path—oscillator → filter → amplifier plus LFO and additional modulation sources, with effects like chorus and reverb slapped on for good measure. Romplers are still in production, though with far richer, many more selectable filters and modulations, and enhanced sound quality, especially when it comes to waves in ROM. Early models sported 8-bit systems, followed by 12 and 16 bits with short single-cycle and attack waveforms. Today's synths ship with very complex waveforms. ROM chips in instruments like the current *Yamaha Motifs ES* contain up to 173 MB of wave data (uncompressed).

Though comparatively short, waveforms stored in ROM can conjure strikingly complex sounds. Listen to a *Korg Wavestation* (1990), which brought the benefits of highly flexible wave sequencing to envelope-controlled oscillators, and you are sure to agree. Its unremarkable commercial success notwithstanding, the *Roland JD-800* (1991) was another highlight of the day. Sliders for touchy-feely instant editing made a comeback on its chassis.

Analog synthesizers' renaissance came in the early '90s, sparked by the 1982 vintage *Roland TB-303*'s contribution to techno music's commercial success. Suddenly vintage synthesizers were back in vogue, and everybody wanted to interact in real time, fondling faders and nudging knobs to tweak sounds. Soon small companies released a new generation of analog synthesizers. The *Novation Bass-Station* (1995), *Doepfer TS-404* (1995), and *MAM MB-33* (1997) all sported dedicated knobs,

switches, and MIDI capability. The analog wave peaked in 2000 with the release of the *Alesis A6 Andromeda*, a fantastically complex 16-voice analog synthesizer. In 2002, even Bob Moog jumped aboard the gravy train and released an enhanced, programmable *Minimoog Voyager*.

In the mid '90s, the guys and gals in white coats let physical modeling synthesis out the lab and into musicians' hands. Offering a complex string model, the *Yamaha VP1* debuted in 1995, followed by the *VL1/VL7* in 1995 with reed and brass models. *Korg's* efforts yielded the monophonic *Prophecy* in 1996 and the polyphonic *Z1* in 1997, both featuring various models, analog oscillator emulations among them. *Technics* choose a different path for the *WSA-1* in 1996, using ROM waveforms as drivers. Modeling algorithms and a subsequent subtractive chain processed the waveforms.

That same year, *Clavia* released the *Nord Lead*, the first DSP-powered virtual analog synthesizer with dedicated knobs and switches for instant editing. The *Roland JP-8000* (1997), *Novation SuperNova* (1998), *Access Virus A* (1998), and *Waldorf Q* (1999) followed in its footsteps.

Though many brands of synthesis set out to rule the world, none would match the tenacity and ubiquity of classic subtractive synthesis harking back to the Minimoog. Its staying power is easily explained: Easy to grasp, handle, and program, subtractive synthesis is a powerful, intuitive way to shape sound. *A good sound does not care how it's been created.* (HGF, 2001)

To learn more about synthesizers and synthesis, visit:

http://www.vintagesynth.org
http://www.synthmuseum.com
http://dictionary.laborlawtalk.com/synthesiser
http://www.cim.mcgill.ca/~clark/nordmodularbook/nm_book_toc.html
http://arts.ucsc.edu/ems/music/equipment/equipment.html

Index

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y