reaktor for burgeoning synthesists

by dave linnenbank

minor revision - 08 march 2005

d.linnenbank@gmail.com
+1.617.515.0342

With gratitude to **TLR** for lunch, et cetera.

table of contents

Native Instruments' Reaktor is a software application for sound generation and processing based around a traditional modular environment where individual units are freely connected to make larger systems.

Approaching any program for the first time requires the user to learn the manufacturer's vocabulary, often times matching ideas you already know with the program's preferred terms. On top of that, each application performs common functions in its own way. Some programs are particularly intuitive. Reaktor is somewhat less so.

Reaktor is a highly capable environment for synthesis which has a steep learning curve. Many tutorials have already been written for new users of Reaktor, but few of these allude to the core ideas of synthesis. Most tutorials rely on Reaktor's library of instruments and macros, telling the user how to connect these pre-made blocks without any explanation of why. For a beginning synthesist, this approach really misses the mark.

This primer intends to acquaint persons who are new to modular programming with Reaktor's characteristics while working within the most basic level of the program. Tutorials here will progress in the direction of building a synthesizer. We are not going to make the greatest synth ever. Instead, our goal is to create simple patches while fully understanding how we did it and how the individual pieces work together.

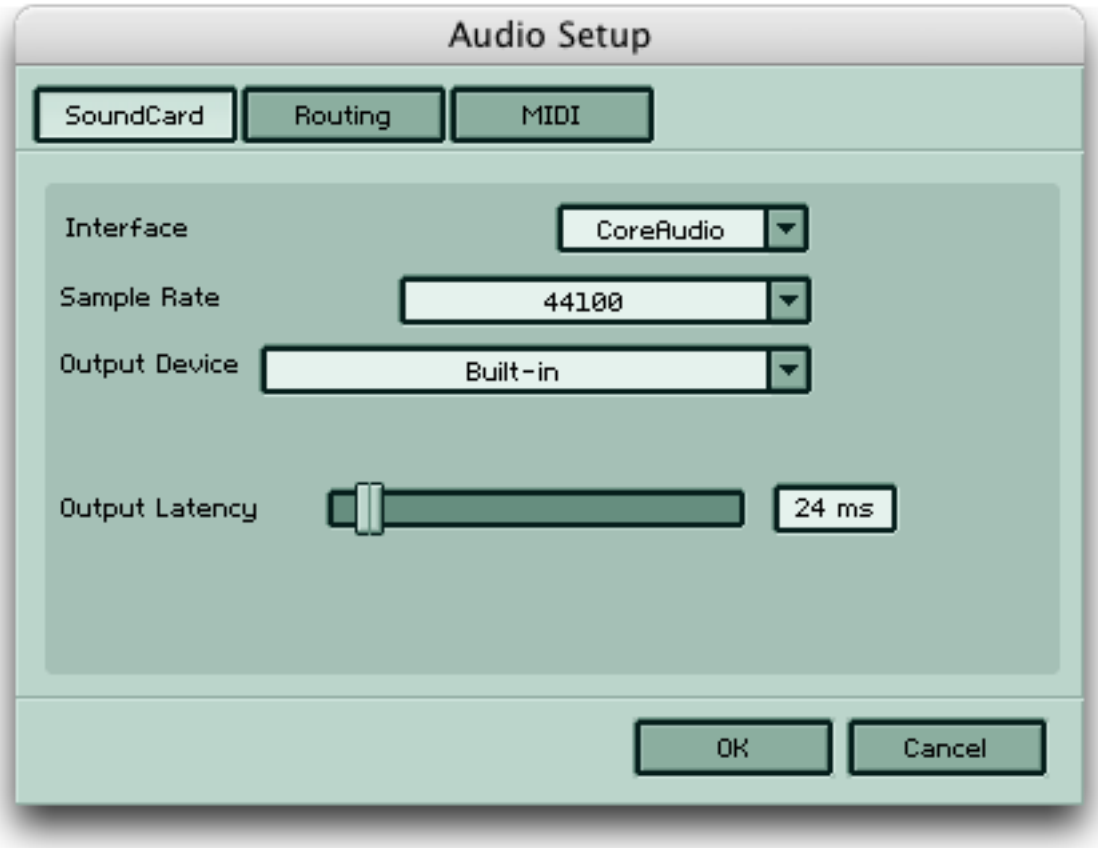
While an understanding of modular functionality and signal flow is essential to using Reaktor successfully, a full explanation of these core concepts is well beyond the scope of this document. However, common synthesis topics — or, "theory" — will be briefly discussed whenever appropriate. Many texts have been written to more thoroughly describe central ideas of synthesis (*The Computer Music Tutorial* by Curtis Roads [MIT Press] immediately comes to mind).

This primer also does not attempt to fully describe each Reaktor module that is discussed. The Reaktor manual's *Module Reference* section (chapter 22 in the current edition) provides adequate descriptions of each module's function and ports.

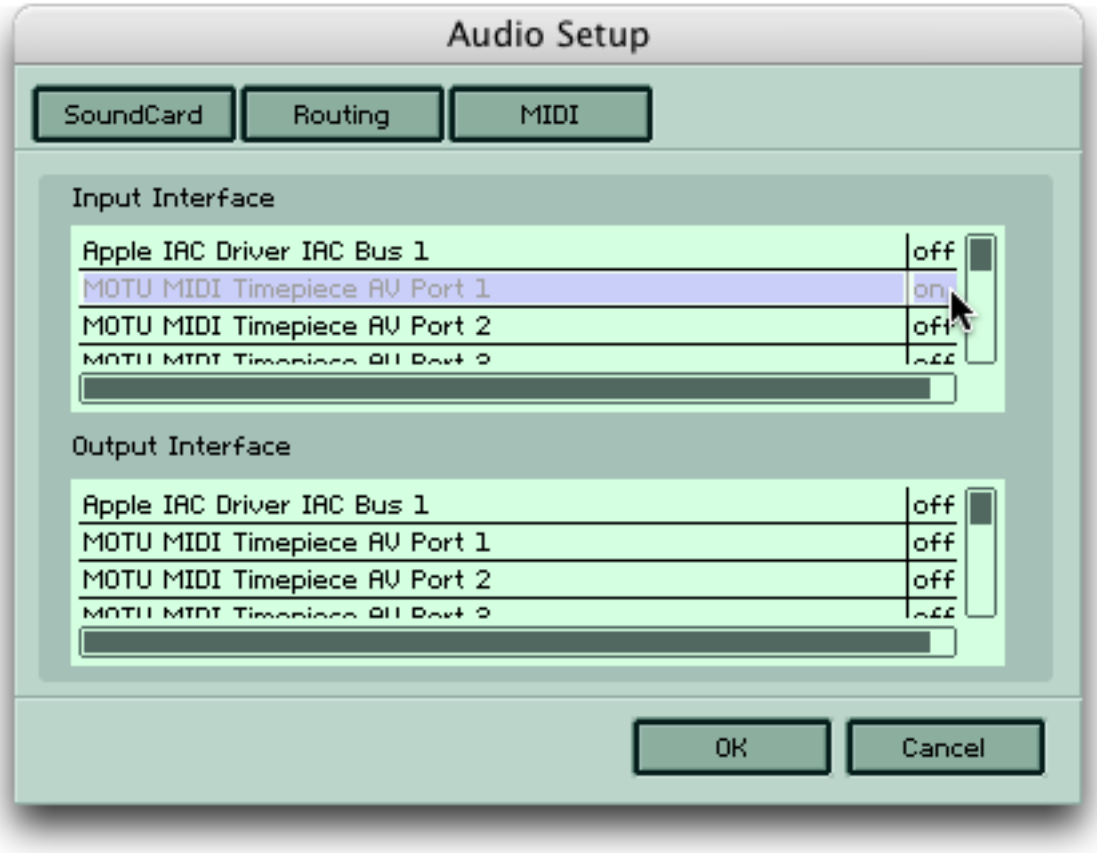
Finally, this document is written with the Mac user in mind. The ideas presented should translate easily, but for additional, PC-specific information, please consult the manual.

-- - --- - --

In order to hear Reaktor, you must select which audio interface to use. Under the **System** menu, select the **Audio + MIDI Settings…** option. In the **SoundCard** tab, select which type of driver you are using under **Interface**. For **Output Device**, you will be given a list of installed audio interfaces. If you are using the computer's internal sound card, select *Built-In*.



In this same window, you can select which MIDI ports Reaktor will respond to. Please click on the **MIDI** tab. All active MIDI devices and ports should show up in this tab. (If something seems to be missing, quit and relaunch Reaktor.) To enable Reaktor receiving (for inputs) or transmitting (for outputs) MIDI data, click the *off* label beside the appropriate item. *off* should now read *on*.



Usually at the top of your screen, you will have a thin window that looks like this.

This floating window is called the **Toolbox**. If you don't see this window while Reaktor is the active program, go to the **View** menu and select the **Show Toolbox** option. The far right side of the **Toolbox** should look like this.

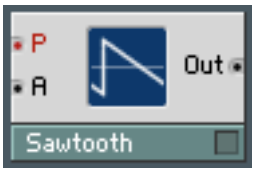
The second icon (with the cursor and letter 'i') sets whether or not Reaktor shows you pop-up hints as you mouse over different components of your patches. You should turn on this feature by clicking on the icon. The button should now appear as seen below.

-- - --- - --

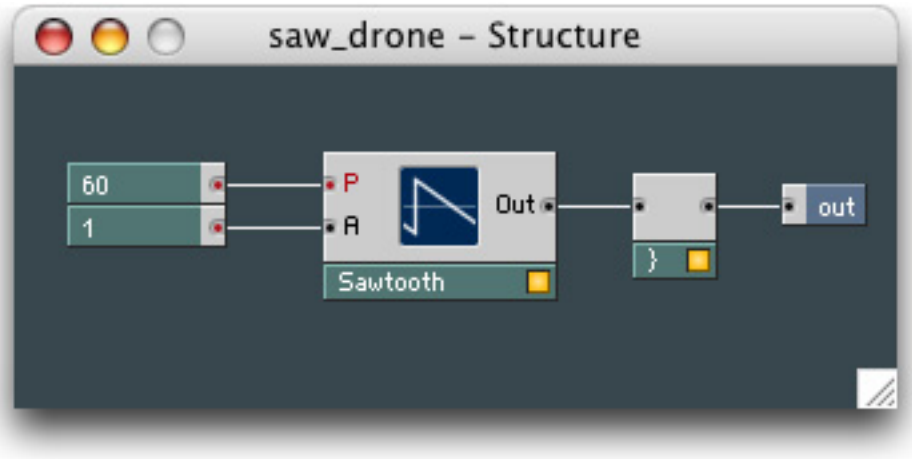
Ensemble Layer
⇓ ⇓
Instrument/Macro Layer
⇓
Module Layer

There are three distinct layers in Reaktor.

• *Modules* make up the lowest level of Reaktor. Each module has a specific function (an oscillator, a filter, a mathematic operation, etc.). They are not files on your hard drive but are internal to the application itself. You cannot make new modules or change the preexisting ones. You simply use Reaktor to connect the modules as you see fit.

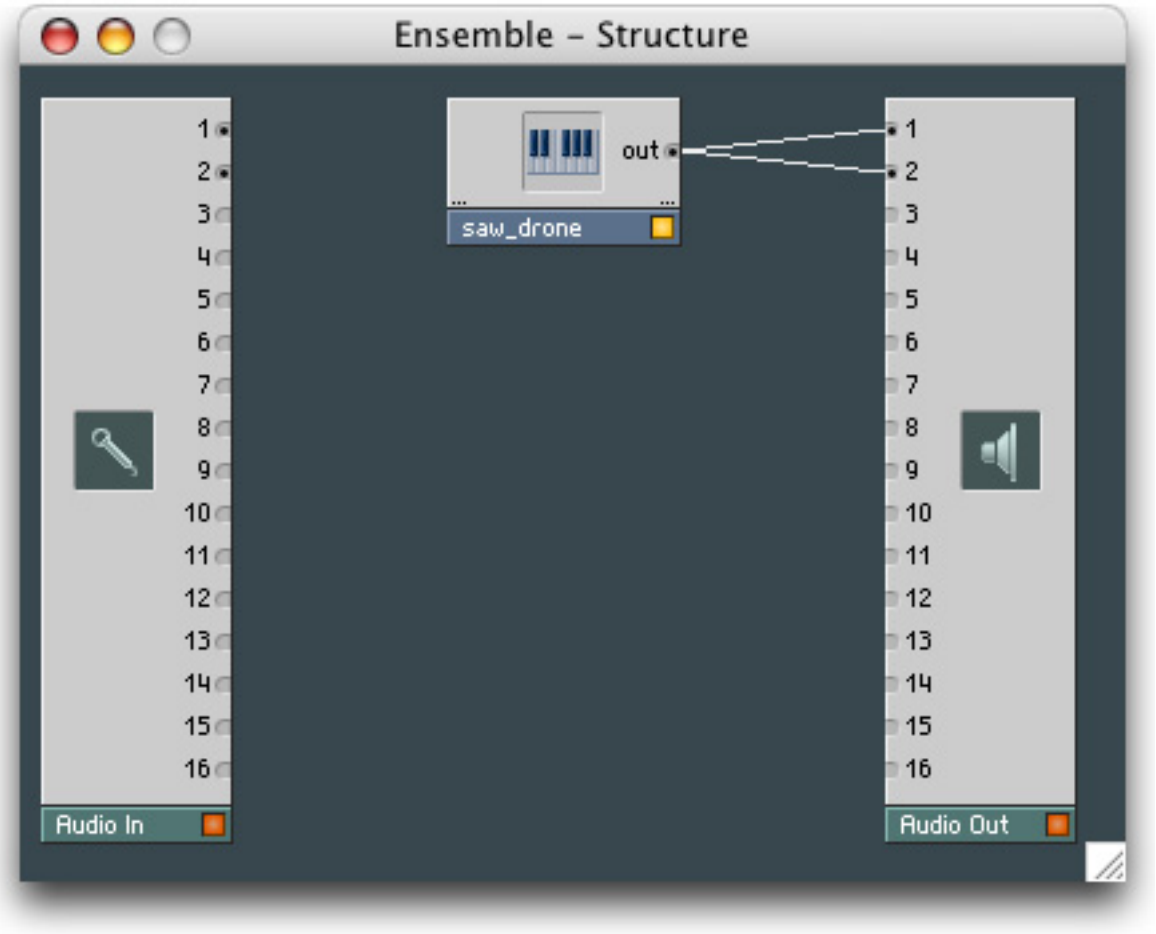


• *Instruments* and *macros* are special modules that make up the middle level. Both are containers for regular modules and patch cords. In short, the heart of your patches goes inside instruments and macros. By default, they are empty. Instruments are independent while macros act as sub-patches within your instruments. Additionally, **In Port** and **Out Port** modules allow instruments/macros to interface with their parent layer.

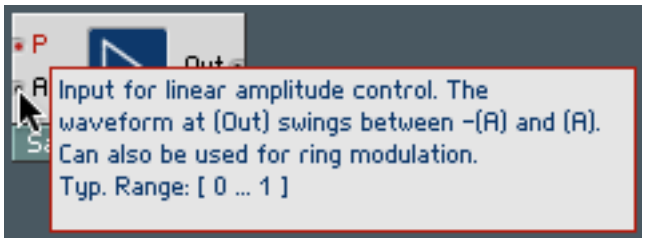


• The top level of Reaktor is called the *Ensemble* layer. Patches you save in Reaktor are referred to as *ensembles*, and only one ensemble file can be loaded at a time. Ensembles house your instruments, allowing you to connect your instruments to each other and with the inputs and outputs of your audio interface.

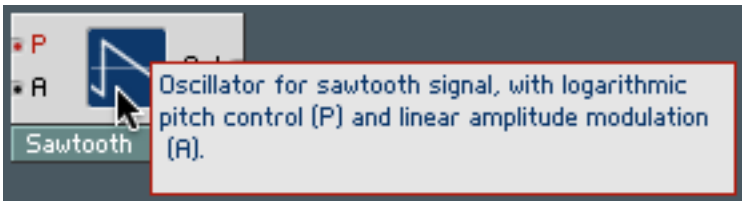
The same instrument shown above appears within the ensemble layer like this.



The little circles on the sides of all devices (including modules, instruments, and macros) represent *ports* for making connections with other devices. Ports on the left side represent *inlets* while those on the right signify *outlets*. These ports are the equivalent of input and output connectors on hardware devices. When you position the mouse over a port, a pop-up window will appear containing a description of the port including the typical signal range associated with it.



Similarly, holding the mouse over the icon in the center of a device will give you a brief description of the device itself.



Reaktor also has two environments for manipulating patches.

• The *structure* environment is where you view actual instruments, macros, modules, and the patch cords that interconnect them. All changes to the framework of your patch — such as creating/deleting modules or rewiring patch cords — are made within this environment.

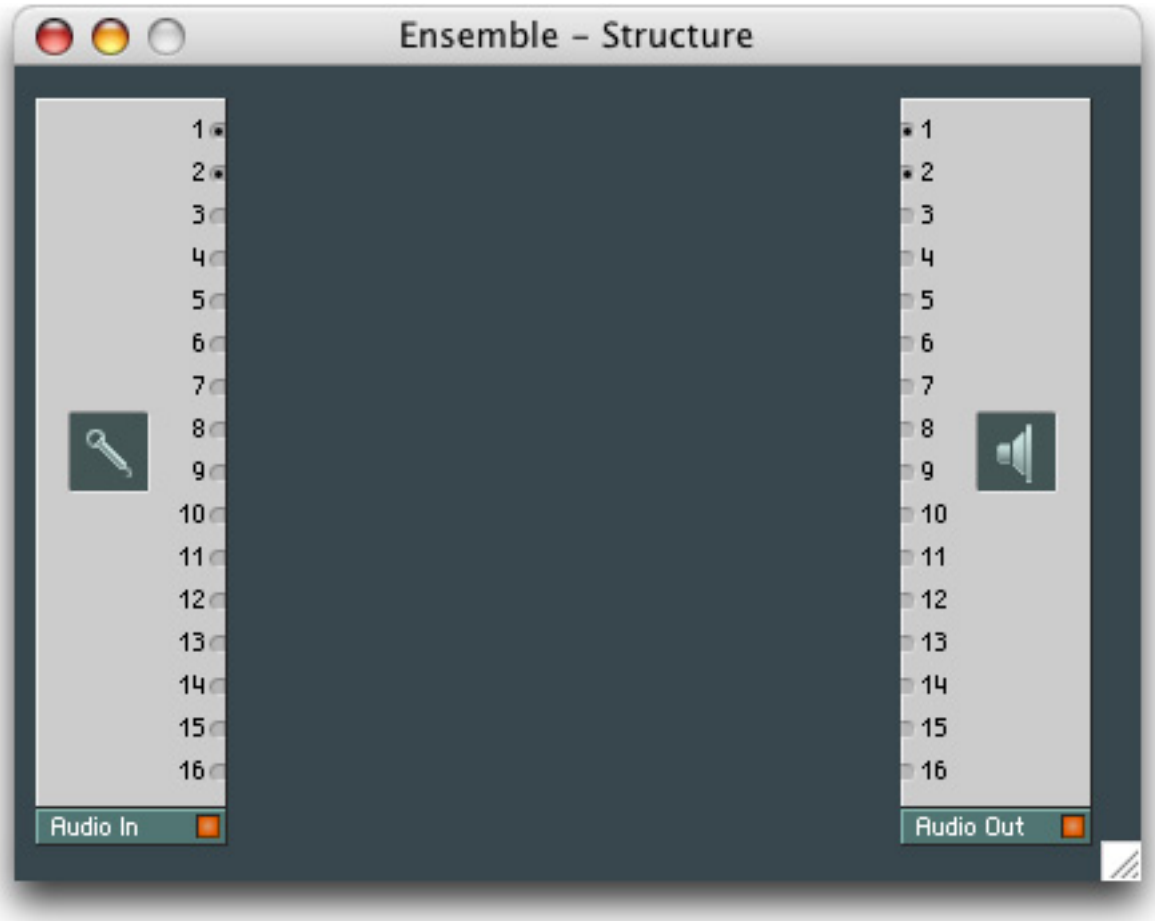
• The *panel* environment (which we have not seen yet) is a symbolic interface for your patch much like the front panel of a hardware synthesizer. Special graphical modules from the **Panel** category (such as faders, knobs, buttons, etc.) are added to the structure environment and typically connected to either give control over various patch parameters or to provide visual feedback of what your patch is doing. Each instrument is given a separate panel window. Every instrument present in your ensemble also has its panel shown within the ensemble panel window.

Each window is named to reflect which layer you are editing and which kind of environment is being shown (*Ensemble - Panel* or *Instrument Name - Structure*, for example).

-- - --- - --

Reaktor comes with a number of pre-made instruments and macros. For this primer however, we are avoiding those units as much as possible. A real understanding of modular concepts comes from creating patches entirely from scratch. We are going to start as simply as possible from a new, blank ensemble.

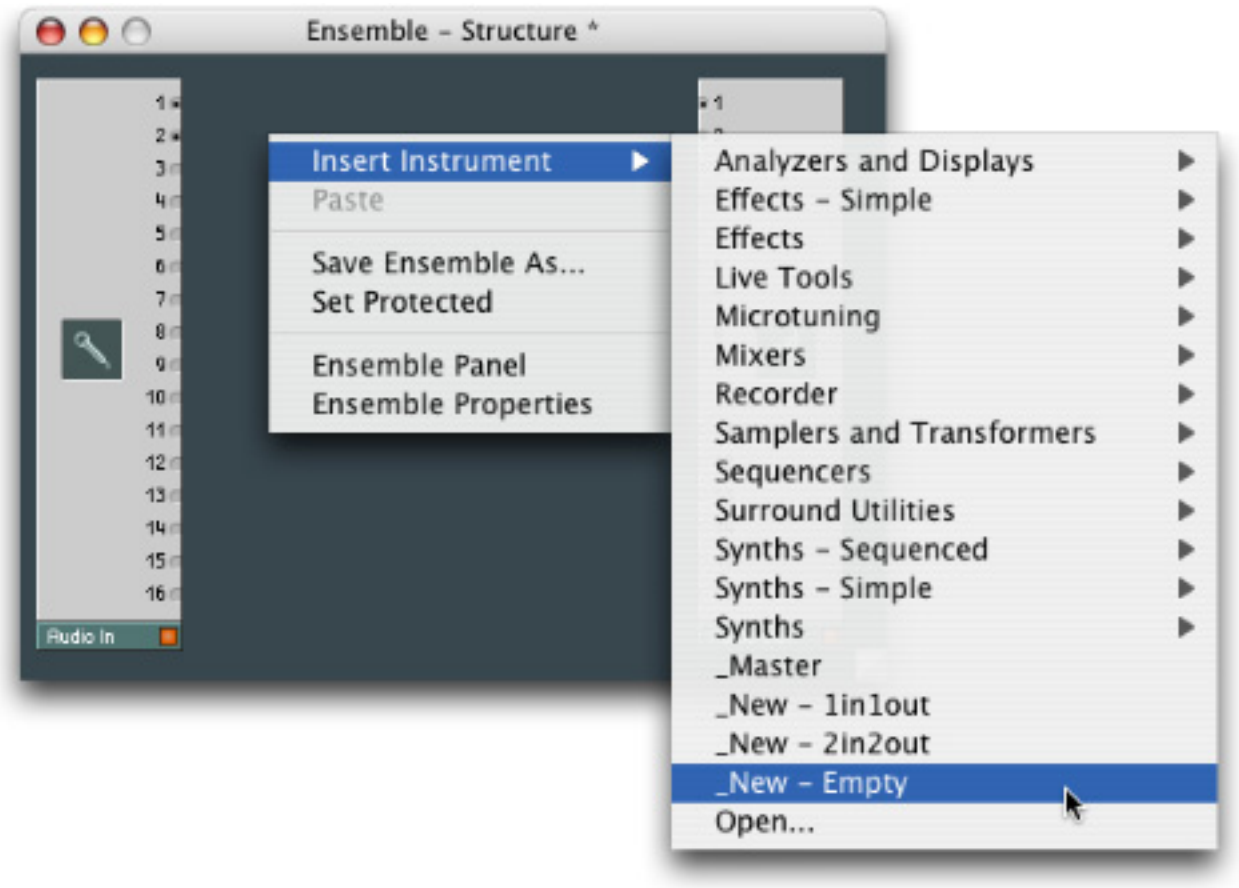
Let's start by going through the first steps of making a patch. We are not aiming to make sound yet but rather to establish a solid starting point for building patches. Under the **File** menu, select **New Ensemble**. An ensemble structure like the one below should appear.



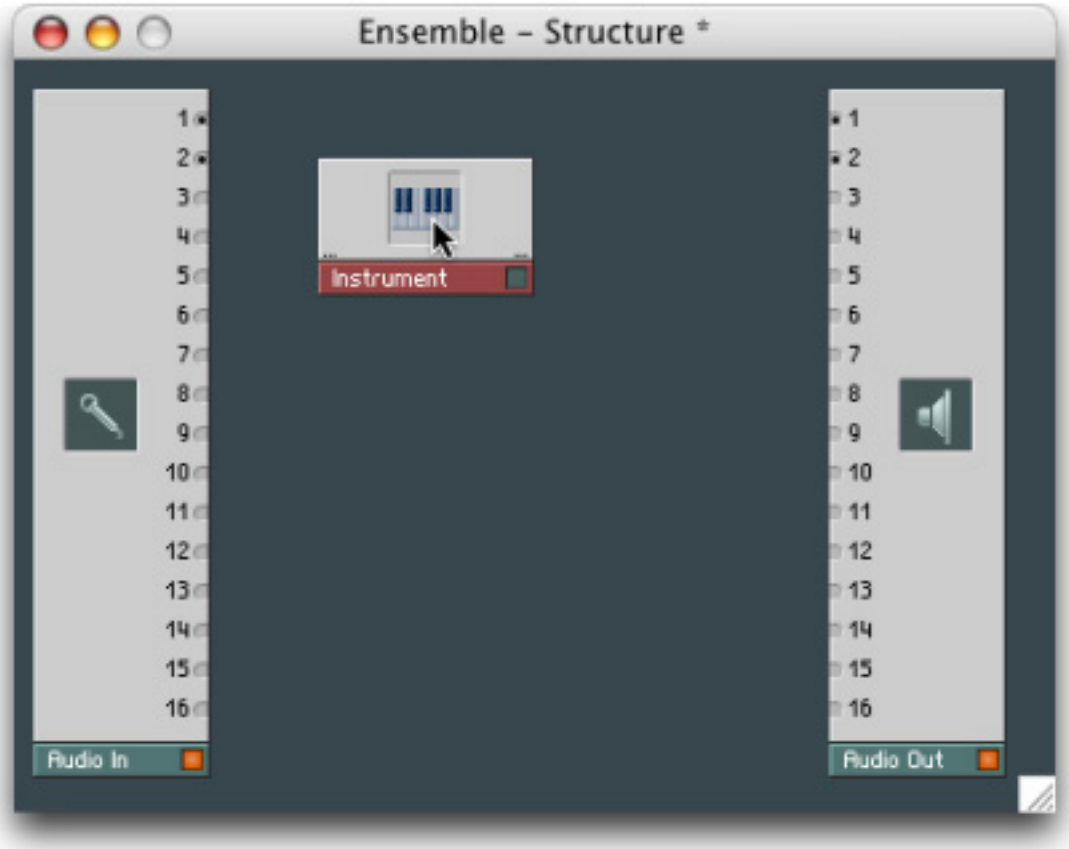
The ensemble layer can contain only three things: the **Audio In** module, the **Audio Out** module, and your instruments.

Audio In and **Audio Out** are special modules that exist permanently on the ensemble layer of any ensemble file and cannot be added or removed. These modules give you a connection to the real world via the inputs and outputs of your audio interface. In short, any instrument you want to hear must be eventually connected to the **Audio Out** module.

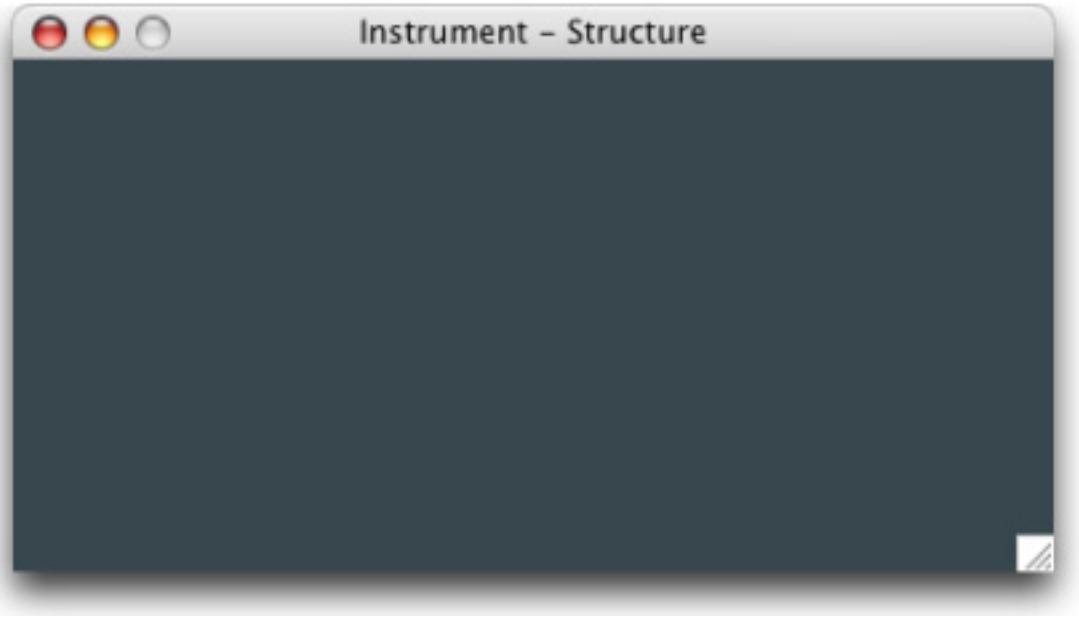
As stated before, *instruments* are shells that contain regular modules and patch cords. Most programming takes place inside of instruments, so let's create a blank instrument. Hold the control key on the computer keyboard and click and hold the mouse in the blank space between the **Audio In** and **Audio Out** modules. A menu will appear.



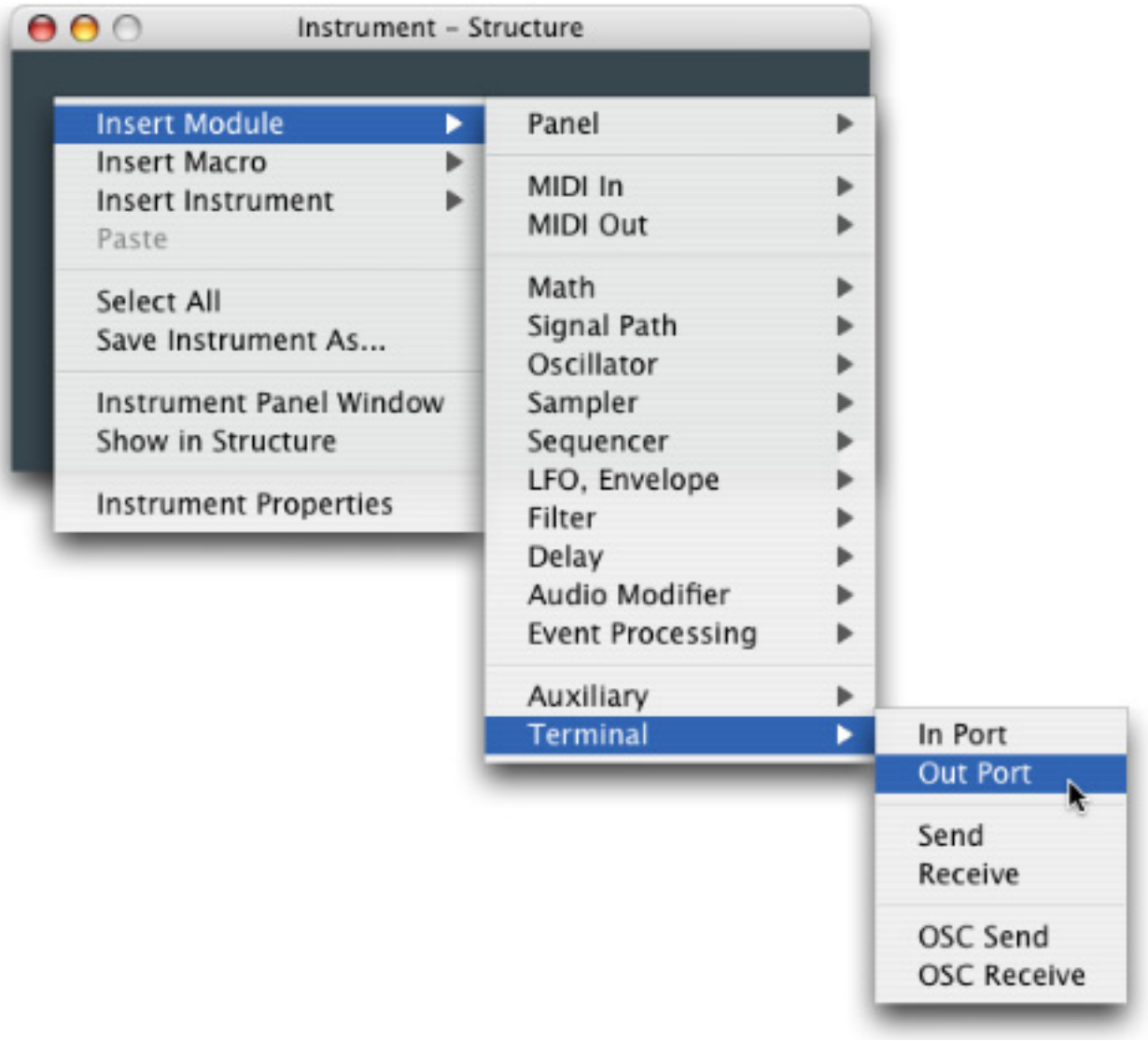
Mouse to the **Insert Instrument** option. Most of these options are the included instruments that we are avoiding. Select the next to last choice, **_New - Empty**. Once you release the mouse, Reaktor inserts a blank instrument with its top left corner positioned wherever you first clicked the mouse.



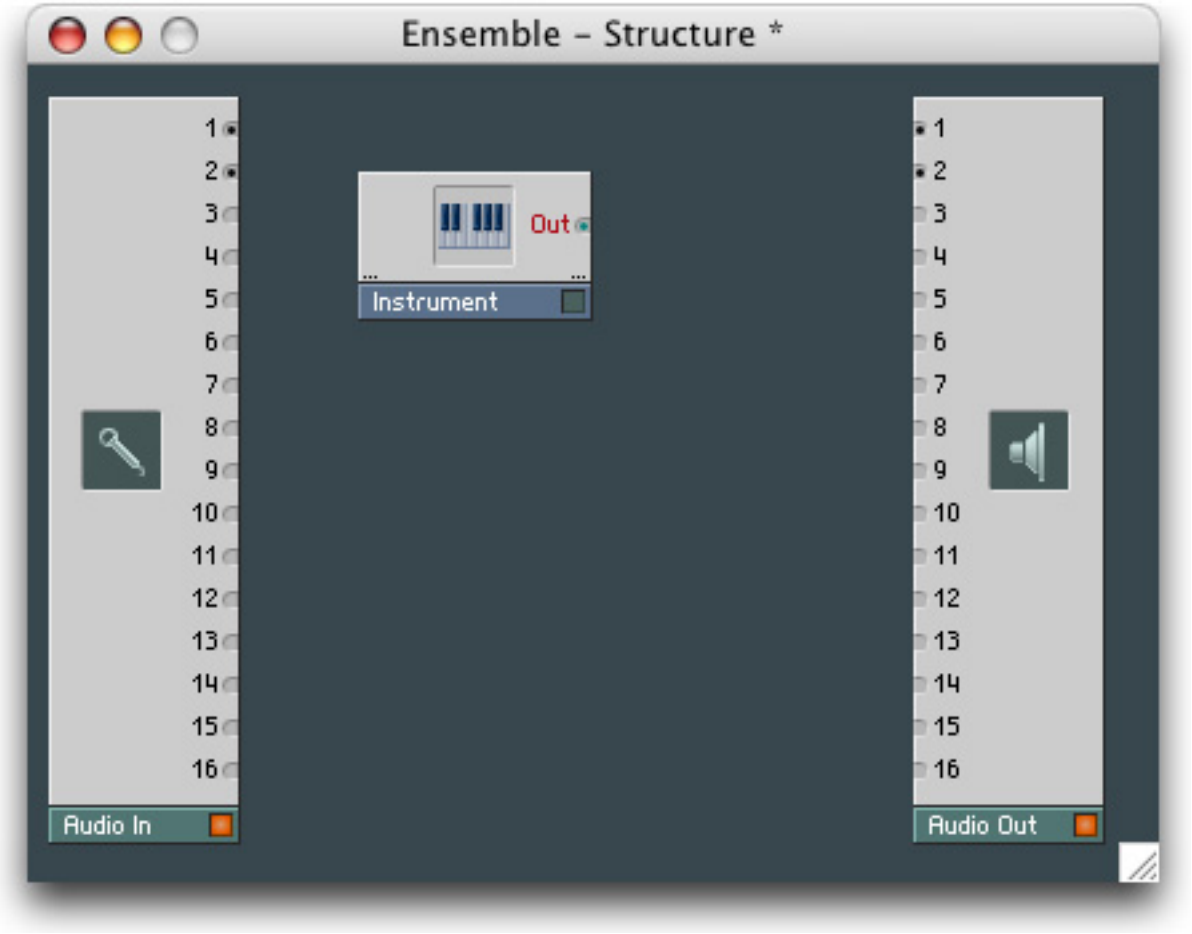
Double-clicking on the instrument's icon will cause the instrument's structure window to appear.



The first thing we need is a way for our instrument to pass signals to the ensemble level. This functionality is provided by the **Out Port** module. If you control-click on the blank space inside the instrument structure window, you will find the **Out Port** module under **Insert Module > Terminal**.

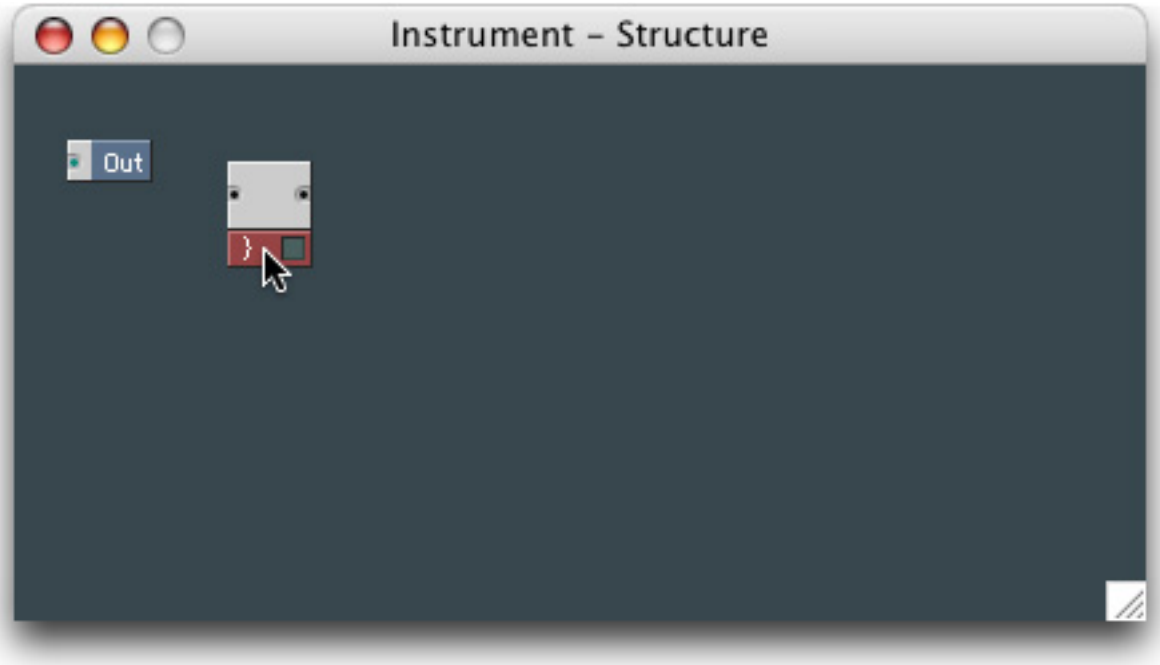


As mentioned earlier, the **In Port** and **Out Port** modules create connectors on the instrument within the ensemble structure environment. Your ensemble should now look like this.

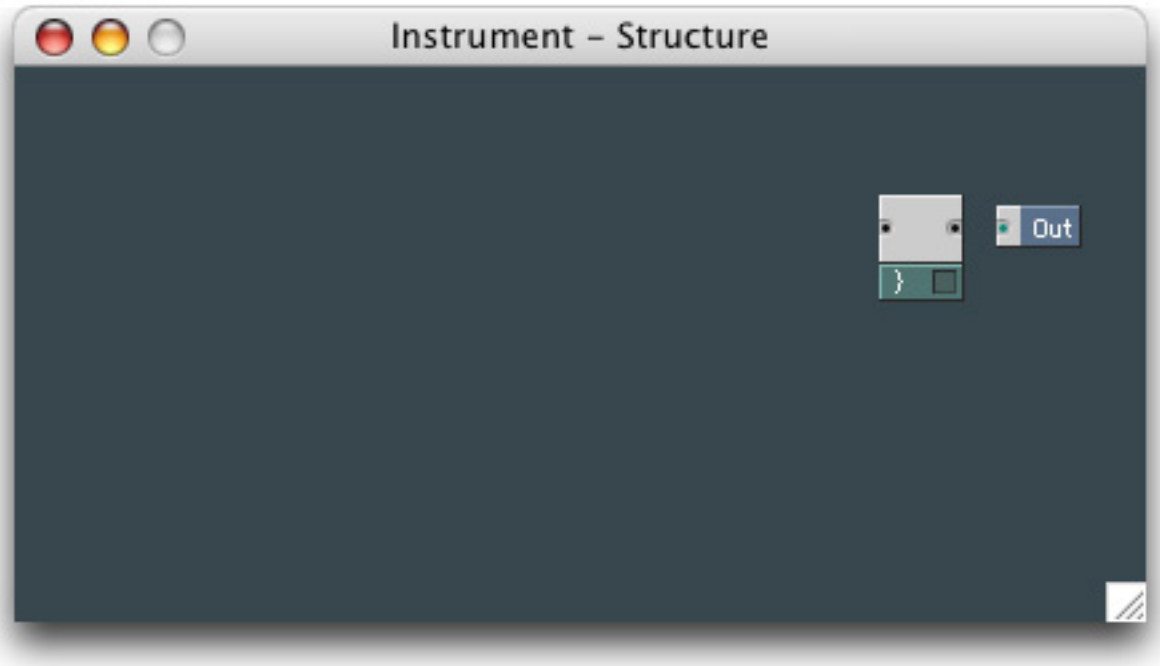


Let's talk about the way Reaktor handles signals for a moment. Often times, we want our patch to be able to play more than one note simultaneously. To achieve this, the computer makes a copy of your patch for each note that you play, and the patch cords — although their appearance does not change — now carry an independent signal for each active voice.

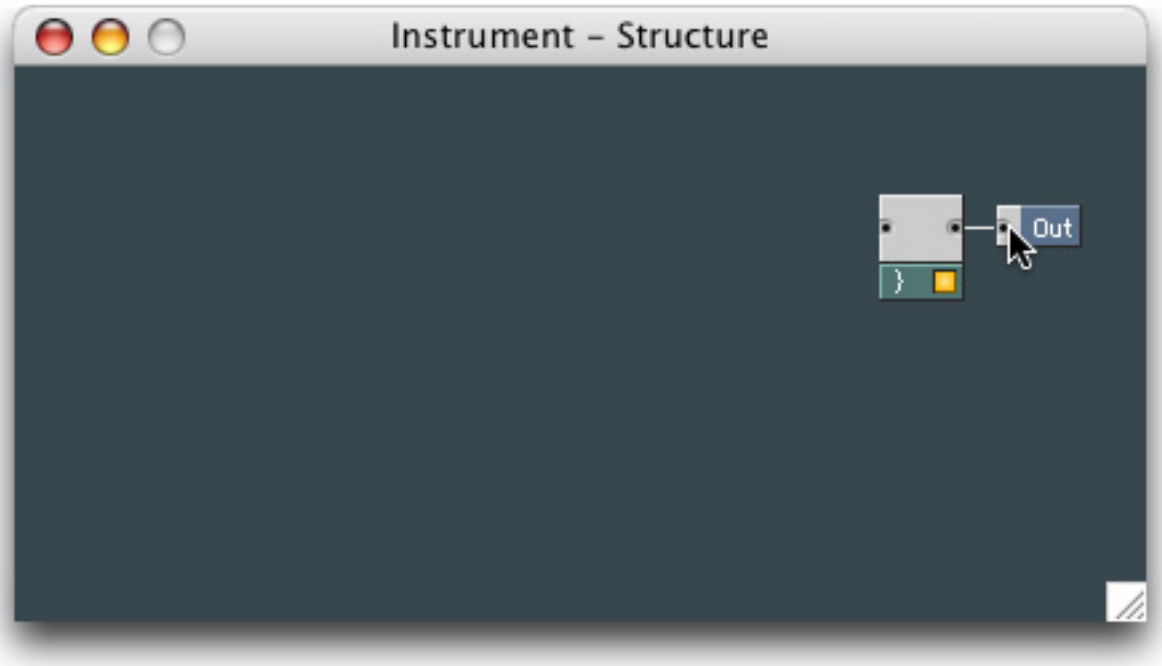
This is how Reaktor internally works. When you want your audio signal to go to the real world (headphones, speakers, etc.), Reaktor's multichannel signals must be mixed down into regular single-channel audio signals. There is a special module that sums a polyphonic Reaktor signal to a singular audio signal. It is called the **Audio Voice Combiner**, and it shows up in your instrument as **}** (a right brace). It is a good habit to use this module in your patch before it is needed. Let's insert one in our instrument now. It can be found under **Insert Module > Auxiliary**.



As mentioned earlier, inlets are on the left side of devices, and outlets are on the right. Let's reorganize our instrument to reflect this left-to-right signal flow by putting the **Out Port** on the far right (since it is the final destination for audio signals within our instrument) and the **Audio Voice Combiner** just to the left of it. You can move modules by clicking and holding the mouse on any part of a module (except the ports) and dragging to the desired location.

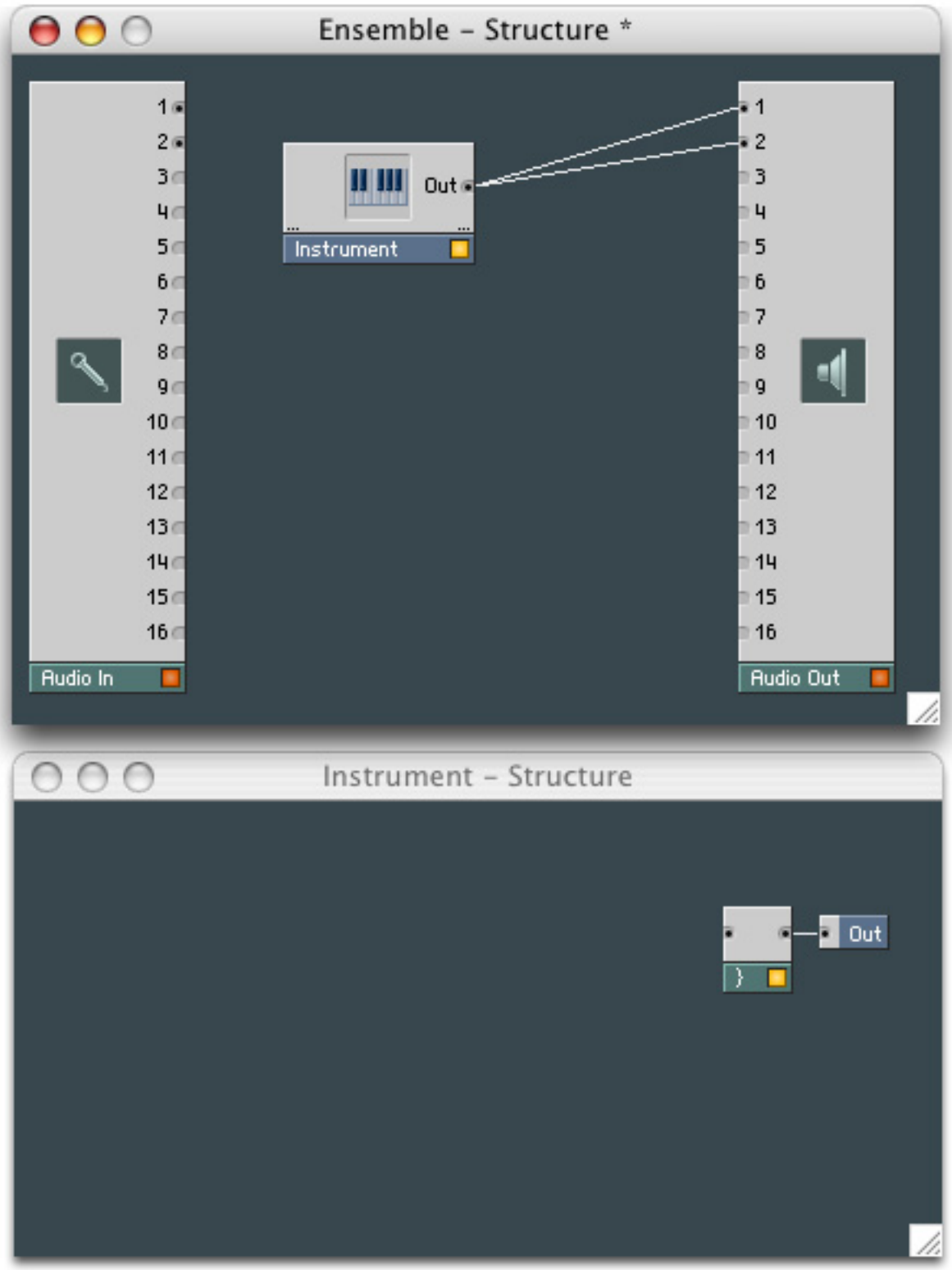


All that remains is creating a few patch cords. Click and hold the mouse button over the out port of the **Audio Voice Combiner** module. Drag the mouse to the inlet of the **Out Port** and release the mouse button.



If a cord is not visible between the ports when you release the mouse, please try again. Make sure you click and release at the center of each connector.

On the ensemble level, connect the instrument we created to the inputs labeled *1* and *2* of the **Audio Out** module. Your patch should now look something like the following.



As you may have noticed, our instrument's audio output is currently monophonic because both ports 1 and 2 of the **Audio Out** module (left and right on a stereo system) are receiving the same signal. Another thing you will notice is that the little lights in the bottom right corner of the **Audio Voice Combiner** and of the instrument are now glowing. This status light tells you that a module or instrument is connected to a signal chain that ends outside of the computer (by way of the **Audio Out** module). If this light is not active, Reaktor has automatically shut off the device for the time being to save your processor from doing unnecessary work. As limitless as the possibilities of a digital computer may seem, the real-time performance of any system is mercilessly finite.

The idea is that we can now create the guts of our patch inside the instrument we have started here and route the final audio signal to the **Audio Voice Combiner**. At that point, the signal will be mixed down (as necessary) and passed to the **Out Port** before proceeding to the **Audio Out** module.

You should probably save this ensemble as something like *starter.ens*. This way you can use it as a template for your own patches if you like, but it is recommended that you rebuild this patch from scratch to start the next few tutorials to review the procedures and reinforce the concepts.

-- - --- - --

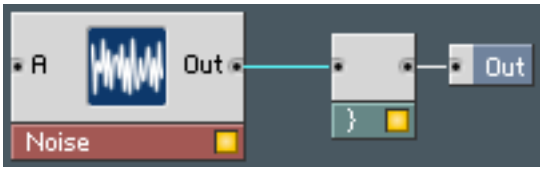
In the **Toolbox** window, you will see this segment on the left side.

This control allows you to toggle whether Reaktor's audio is enabled or shut off by clicking the icon on the left. A dimmed icon (as shown above) indicates that audio is turned off. The next picture signifies that audio is enabled.

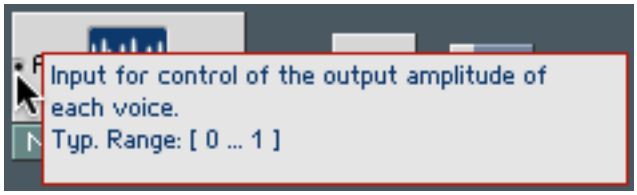
As you can see, the icon is lit up. The display on the right indicates what percentage of your system's processing capability is being used to realize the current ensemble. When constructing patches that actively output audio, it is less dangerous to your ears if you turn off Reaktor's audio while editing. Please turn off the audio for now. (*Note*: Every time you open Reaktor, the audio is automatically turned on. It is highly recommended that you turn your audio level all or most of the way down before opening the program. Not all surprises are good.)

Please recreate or open your copy of the ensemble patch described in Tutorial 01.

Go to the instrument's structure window and insert a **Noise** module. You will find it under the **Oscillator** module group of the **Insert Module** option. Connect the out port of the **Noise** module to the in port of the **Audio Voice Combiner**.



You can see that both modules are lit up now, but if you turn Reaktor's audio on briefly (please do), you will notice that we are still not getting any sound. First, let's figure out what the *A* inlet wants. Hold the mouse over the inlet, and a pop-up window should appear.

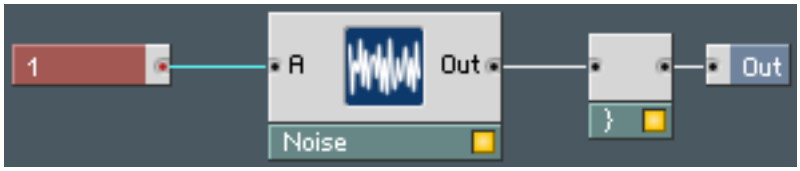


A signal's amplitude could be described as its momentary strength. While most oscillators simply generate a bipolar signal that swings around 0 (zero) between +1 and -1, oscillators in Reaktor have an amplitude parameter for scaling the signal's strength. A setting of 0.5 represents a half-strength signal (+0.5 to -0.5) whereas 0 represents no signal. So how does all this affect the sound?

The function of any signal is determined solely by its destination. In our current patch, the **Noise** generator is connected directly to our audio output. In this case, the amplitude of our **Noise** oscillator directly controls the audio's volume. Later on, we will investigate how different routings bring different results.

Reaktor's modules — to allow greater opportunities for modulation — provide an inlet for any non-fixed parameter. To set any parameter of a module, you must feed a signal to the module's corresponding inlet. An unconnected inlet uses a value of 0. Accordingly in our current patch, the unconnected amplitude inlet of the **Noise** generator assumes a value of zero. To get sound from this module, we must provide a signal to set the amplitude.

Be sure that audio is turned off. Then control-click on the inlet and choose **Create Constant** from the menu.

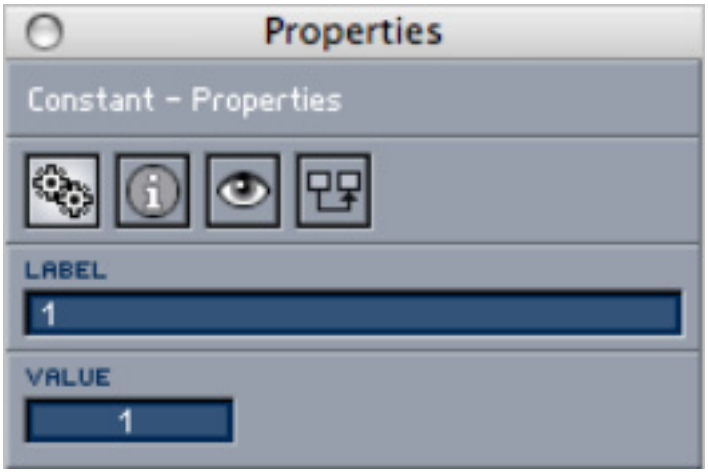


The **Constant** module (found under the **Math** category) outputs a continuous signal at a specified value. When you use the **Create Constant** function from an inlet's contextual menu, Reaktor creates a **Constant** module, connects it to the inlet you selected, and sets the value of the constant to a level Reaktor feels is appropriate for that type of inlet.

The value of a **Constant** can be changed by using the **Properties** window. To access this window, control-click on the module (labeled *1* in our patch, representing the current value of the constant) and select the **Properties** option.

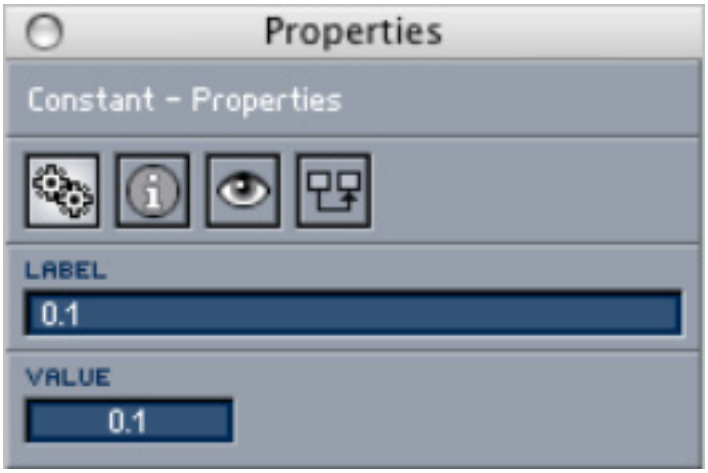
As you can see, the top of the window indicates which unit's properties are being shown (*Constant* in this case). The four icons towards the top of the window represent the four different pages of the **Properties** window.

They are referred to by Native Instruments as the *Function*, *Info*, *Appearance*, and *Connection* pages, respectively. The contents of each window change depending on what type of unit is active (module, instrument/macro, or ensemble). When dealing with modules, the *Function* page is by far the most useful. Please click on the leftmost icon.

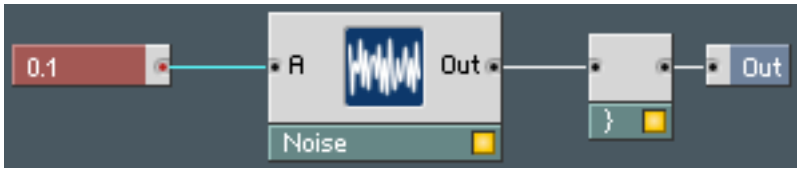


The *Function* page of the **Properties** window shows settings relevant to the active module. *LABEL* is a purely cosmetic setting present in every unit for choosing the text that appears in the structure. The only particular setting for a **Constant** module is the *VALUE* of the signal being output by the module.

Since this **Constant** determines the amplitude of the **Noise** oscillator, let's set the value to something less than full volume. Click in the *VALUE* field so that the cursor appears, change the current value of 1 to 0.1, and hit return. The Properties window should now appear as seen below.



As you probably noticed, the **Constant** module has updated its label to match the new value. This is a nice practice because it makes the structure much easier to read.



We are finally ready to turn on the audio, so make sure your speaker volume is low or completely down. Enable Reaktor's audio. The *AUDIO OUT* meter on the **Toolbox** should display the level of audio going to the speakers.

Slowly turn your speaker volume up to a comfortable level. You should be hearing white noise. If you edit the value of the **Constant**, you will notice the volume and the *AUDIO OUT* meter fluctuate accordingly.

As you can see now, Reaktor uses an amplitude-driven model where no signal is produced until you set an amplitude at the beginning of your signal chain (often with an oscillator).

You may be wondering why it took 24 pages just to get some noise. Fair question.

Reaktor's architecture is one that has to be fully appreciated before any level of it can be approached. It is a complicated program intended to do complicated things in a complicated way. This is not to discourage you from exploring the full potential of the program. As stated before, this document is intended to get you working with Reaktor so you can explore central synthesis concepts. The program's biggest conceptual blocks are now over. Don't worry; no one expects you to fully understand this complex program after reading only 24 pages of some guy's tutorial. We are taking a progressive and thorough approach. Our tangible results may not be great yet. An understanding of the basics is what we are after.

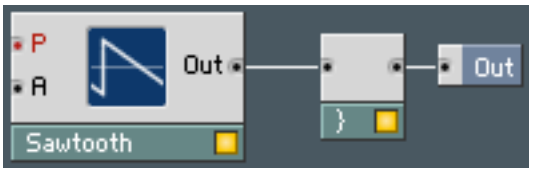
A word of advice. When experimenting with new ideas, start as simply as possible. By the same token when things get confusing, reduce what you are working with to the bare essentials, and then insert additional components, one at a time. We are starting with the simplest system possible within Reaktor (the **Noise** oscillator has only one parameter). From here, we will start adding more variables.

-- - --- - --

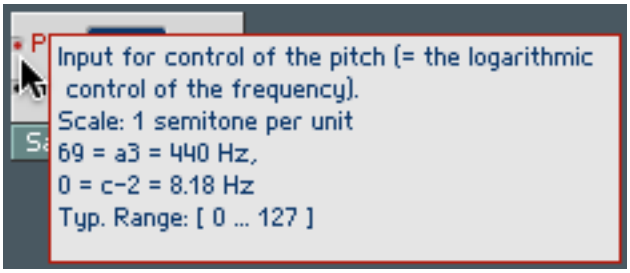
We used a fixed oscillator in the last tutorial that had only an amplitude parameter. (In fact, amplitude is the one parameter common to virtually all of the **Oscillator** modules in Reaktor.) This time, we will use an oscillator that also has a frequency parameter. The presence of a frequency parameter indicates that our oscillator is periodic and, accordingly, pitched.

Please recreate or open your copy of the ensemble patch described in Tutorial 01.

Create a **Sawtooth** module (found in the **Oscillator** group) and connect it to the **Audio Voice Combiner** module.



There are a couple things you will probably notice at first glance. First, our amplitude inlet is there as we expect, but the frequency inlet you were promised is showing up as *P*. Please mouse over the inlet to investigate.



When an inlet is labeled *frequency* (*F* in Reaktor's structure window), this indicates a linear control of frequency in Hertz (cycles per second). An inlet labeled *pitch* (*P*) is a logarithmic control of frequency using MIDI note values. Be sure you know which type of inlet you are using.

Second, our pitch inlet and its lettering are red while most of our ports thus far have been black. In Reaktor, black connectors represent ports for *audio signals*. These are signals that are continuously being updated and outputted. As such, they constantly use the processor. Red connectors represent ports for *event signals*. These are impermanent signals that occur at irregular intervals. Since these signals are temporary, they only impact the processor when a new value is received. As the name implies, think of them as occasional events (like a series of MIDI notes) instead of a constant stream (like a continuous audio signal).

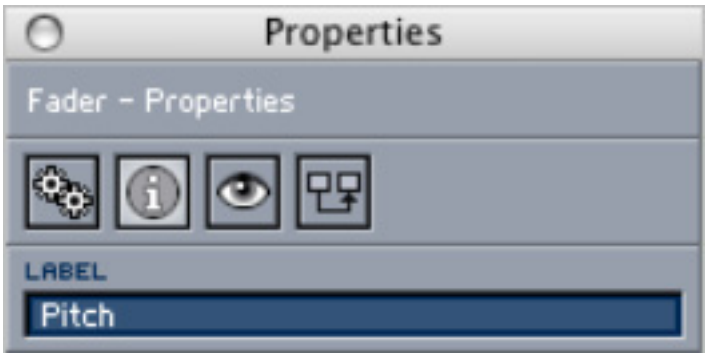
Let's go back to our patch and use the **Create Constant** option on both the pitch and amplitude inlets.

As you can see, the amplitude inlet is given a **Constant** with a value of 1, the same as was given to the **Noise** oscillator earlier. Again, please change the value of the constant to 0.1 in order to reduce the signal size.

The pitch inlet is being fed a value of 60 by its **Constant** module. Since the pitch inlet is using MIDI note values, 60 in this case is equivalent to middle C. This note is sometimes called *C3* and has a frequency of roughly 261.63 Hz. Please turn on the audio briefly. You should hear a sawtooth wave tuned to middle C.

So far, we have succeeded in getting our oscillators to drone endlessly. We will now start considering ways to make our patches more flexible and dynamic. Let's start by deleting the **Constant** module that is driving the pitch value. Select the **Constant** module and then press the delete key, or control-click the module and select **Delete** from the contextual menu. Now control-click on the pitch inlet and select **Create Control**.

A new module we have never seen before — labeled *Pitch* — is now connected to the pitch inlet. Let's look at the **Properties** window for this module by either control-clicking on the module and selecting the **Properties** option or simply double-clicking on the module itself.



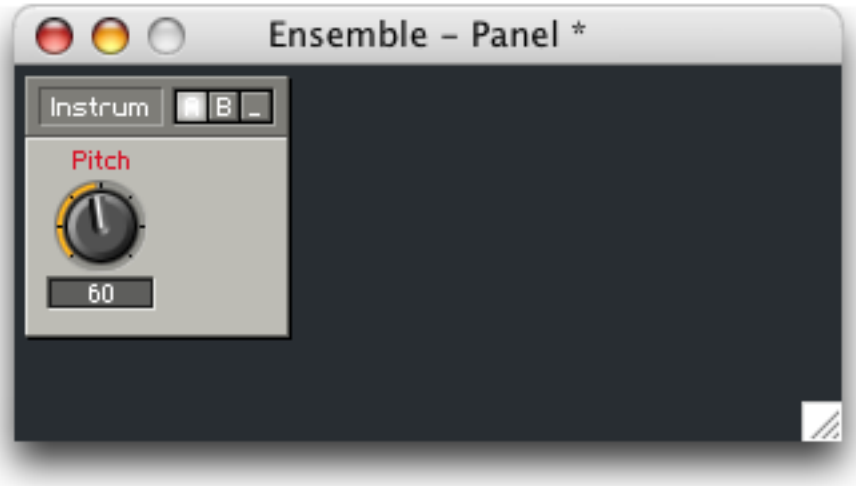
Apparently, this is really a **Fader** module (which can be found in the **Panel** category).

When you use the **Create Control** function from an inlet's contextual menu, Reaktor creates one of the **Panel** category modules, labels it, sets a range and current value for it — all based on what type of inlet you are dealing with — and finally connects the **Panel** module to its inlet.

As briefly mentioned earlier, modules from the **Panel** category appear in both the structure and panel environments. Let's visit the panel environment for the first time.

Take a look at the far right side of the **Toolbox**.

The middle icon that looks like buttons and faders is a shortcut to the current ensemble's panel window. Go ahead and click on the icon.



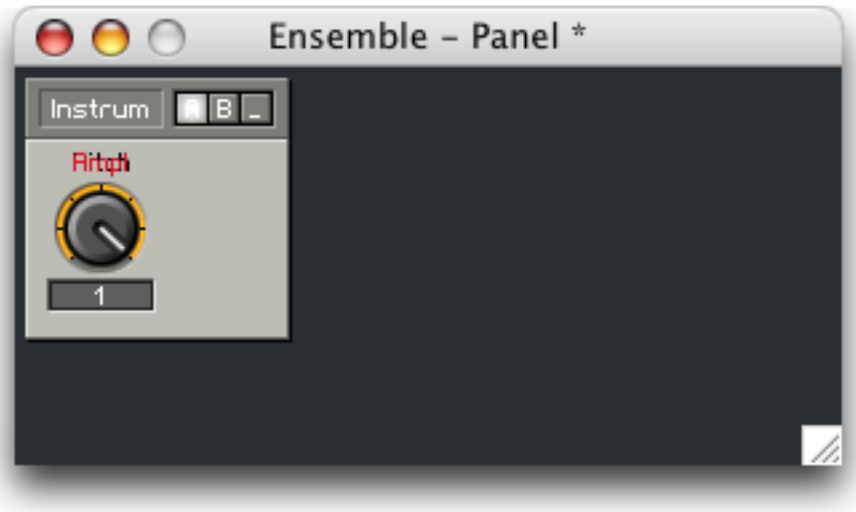
A window like the one above should appear. As the title explains, this is the panel window for our ensemble. Click on the knob and drag up or down to adjust the pitch value. Turn the audio on for a moment and listen to the frequency change as you manipulate the knob. Since we are using MIDI pitches, each whole number change is equal to one half step.

This knob is the panel representation of the **Fader** module we created in the structure environment (where it is still labeled *Pitch)* and not a separate object. Changes made in either environment (panel or structure) affect both versions of the module.

Just as the **Constant** module allows us to set its value, the **Fader** module allows us to set its many parameters. This is also done through the **Properties** window. Control-click on the knob image and select **"Pitch" Properties** or simply double-click on the knob. Select the *Function* page of the **Properties** window (the icon on the far left).

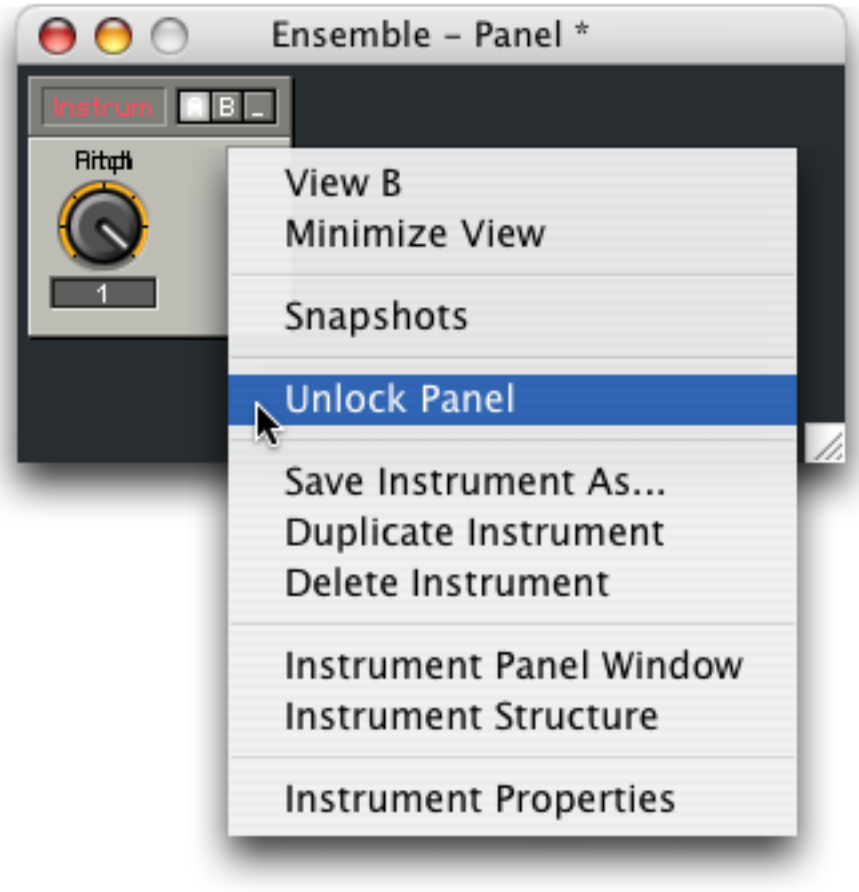
Of these settings, the key ones are *MAX* (the maximum level of the fader), *MIN* (the minimum level), and *STEPSIZE* (the distance between neighboring values). Since MIDI pitch is expressed from 0-127, the provided settings cover the full range of typical note values. All of these values can be edited, but we will leave them alone for now.

We might as well make the amplitude parameter dynamic, too. In the structure window, delete the **Constant** module that is feeding the amplitude inlet of the **Sawtooth** oscillator. Control-click on the amplitude inlet and select **Create Control**. Now take a look at the panel window.



As you can see, the new **Fader** module for controlling amplitude has been placed directly on top of our *Pitch* fader. Unfortunately, this is the expected behavior.

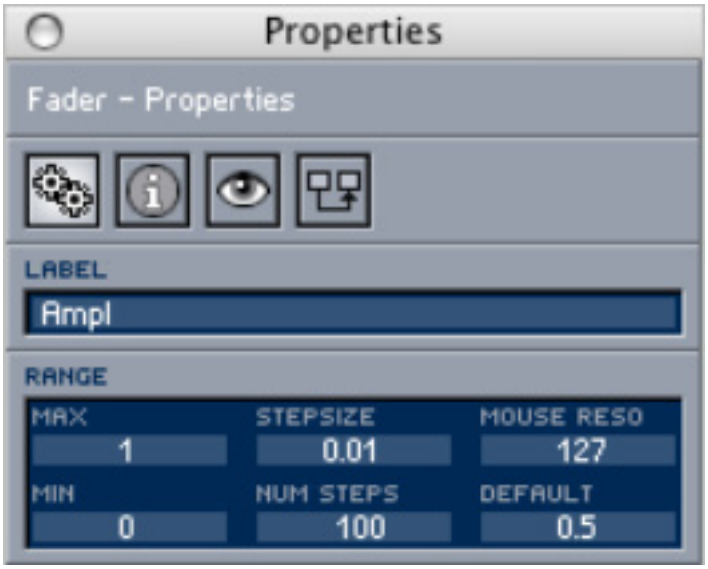
Reaktor behaves somewhat strangely within the panel environment. Any **Panel** module created within an instrument is positioned at the top left corner of the instrument's panel. This environment is locked by default. To unlock it so we can move things around, control-click in the blank, gray area and select the **Unlock Panel** function.



The panel background appears as a grid when the window is unlocked. To move a module in the panel window, click and hold the mouse on any part of the module and drag it to the desired location. Go ahead and move the **Fader** module controlling amplitude — currently labeled *Ampl* — to the right of the *Pitch* **Fader**.

Another odd thing about the panel environment is that you cannot set the size of the panel. Reaktor automatically uses the minimum possible size based on the position of the modules in the panel. This explains why moving a module out of the top left position (the *Pitch* **Fader** in this case) makes the whole window seem like it's moving.

Double-click the *Ampl* **Fader** (or control-click and select **"Ampl" Properties**).

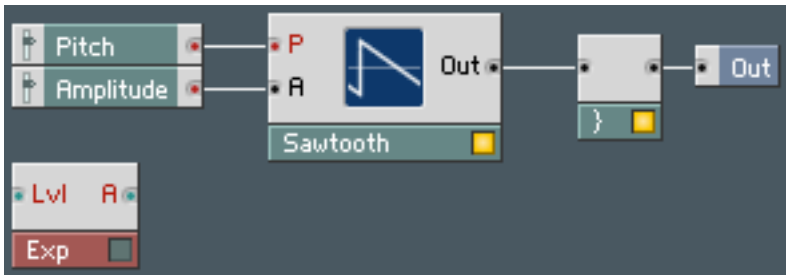


Reaktor has logically set this module's range for amplitudes from 0-1 for silence to full signal. Since we have plenty of space, go ahead and relabel this **Fader** as *Amplitude*.

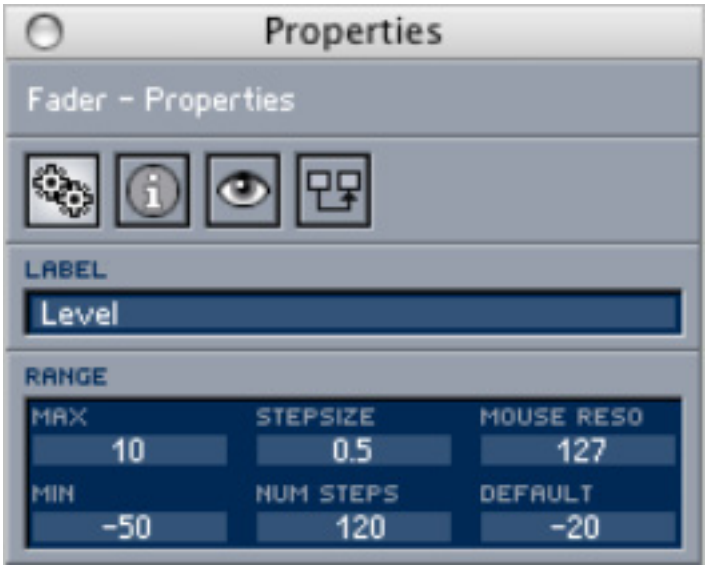
Please click on the wrench icon beside the label *Instrument*. This is another way to lock (and unlock) the panel.

Amplitude is more often measured in logarithmic decibels (dB) than the linear amplitude values the computer uses. This is because we do not perceive loudness in a linear fashion. As an example, set the *Amplitude* knob to zero and enable Reaktor's audio. Turn the knob up in very small steps. Listen to the volume and watch the *AUDIO OUT* meter. You will notice that steps are not evenly distributed across the range. In particular, values from 0.0 - 0.1 cover about 60% of the total dynamic range.

It may be better in situations like this to control the amplitude logarithmically, providing a more linear perception across the entire range. Reaktor has a module that converts decibels to linear amplitude. It is named **Expon. (A)** and can be found under the **Math** category. Go ahead and add one to the patch.



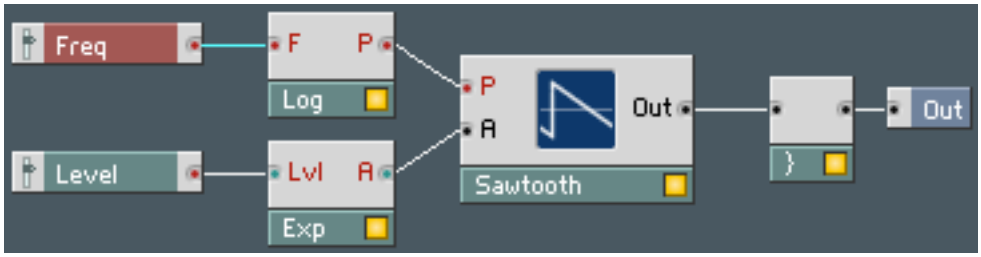
In the **Structure** window, delete the **Fader** labeled *Amplitude* and connect the **Expon. (A)** module to the amplitude input of the **Sawtooth**. Control-click on the inlet of **Expon. (A)** and select **Create Control**. We could have reused the old **Fader**, but using the **Create Control** function simplifies our job by having Reaktor set a reasonable range for us. Open the **Properties** window for our new **Fader** (which is appropriately labeled *Level*). Notice how different these range settings are from the previous **Fader** which controlled linear amplitude.



If you look at the panel window, you will notice that Reaktor has put the new **Fader** module on top our *Pitch* knob again. Now is a good time to move the *Level* knob to the right and to

accept that Reaktor will always behave this way. Please lock the panel.
Again, set the *Level* **Fader** to zero and enable Reaktor's audio. As you did with the linear amplitude controller, turn the knob up in very small steps, listen to the sound, and watch the *AUDIO OUT* meter. Notice how the changes are much more uniform over the entire range of the **Fader**. Also take note of how the tone of the sound changes as the level meters approaches zero. This is only the result of amplifying the signal too much so that the signal becomes distorted.

For our pitch inlet, the opposite sort of conversion is possible. The inlet's use of musical semitones sounds linear, but for some applications, it is easier to control frequency in Hertz. The conversion module that makes this possible is called **Log (F)** (also found in the **Math** category). Please add one of these modules to the instrument. As we did with the amplitude inlet, delete the already connected *Pitch* knob, connect the **Log (F)** module to the pitch inlet of the **Sawtooth**, and use the **Create Control** feature on the input of **Log (F)**.



Rearrange the panel window so that both controllers are in usable positions. Adjust the value of the new *Freq* knob and observe how this type of control differs from the *Pitch* method.

A couple brief notes regarding Reaktor's signal architecture. Both the **Expon. (A)** and **Log (F)** are actually *hybrid* modules, meaning they can process and output either event or audio signals depending on how the module is connected. Feeding an audio signal to the input forces the module to operate as an audio module while connecting the output to an event inlet makes it work exclusively as an event module. There are many hybrid modules in Reaktor. They can be identified by their green ports when no cables are attached.

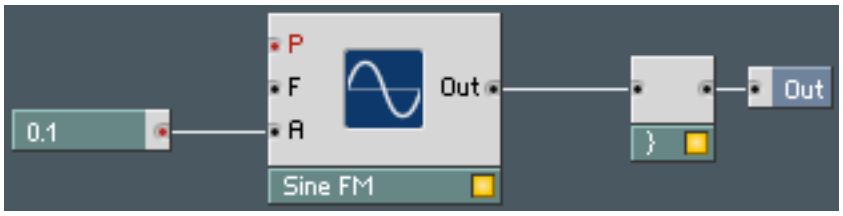
Additionally, event signals can be connected to audio inlets, but operations will still occur at the more processor-intensive audio rate. Event inlets, however, are unable to directly handle audio signals. We will demonstrate a workaround in the next tutorial.

-- - --- - --

While on-screen controllers like **Fader** are more dynamic than the **Constant** modules we were using at first, the most interesting sounds involve automatic controls — or *modulations* — as well. Let's explore some of those possibilities.

Please recreate or open your copy of the ensemble patch described in Tutorial 01.

Create a **Sine FM** module from the **Oscillator** category. Connect the oscillator's output to the **Audio Voice Combiner** module, and create and connect a **Constant** module with a value of 0.1 to the amplitude input.

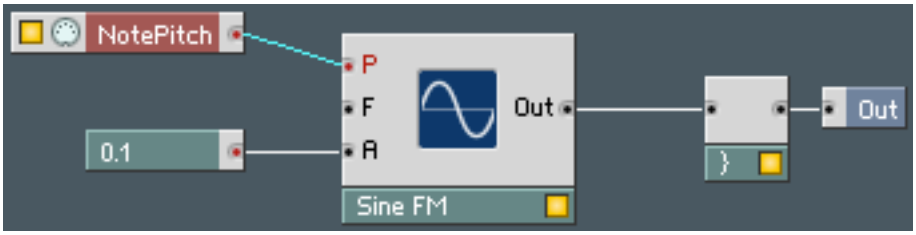


The *FM* of **Sine FM** signifies that this module has a *frequency modulation* input in addition to the pitch inlet we have already worked with. The sum of the pitch and frequency inlets sets the module's actual frequency. We will start with the pitch inlet.

As with the **Saw** module previously, the pitch inlet of the **Sine FM** module (and most other modules in Reaktor) is intended for event signals. Event signals are generated by on-screen controllers (such as **Fader** and other **Panel** modules) or received from MIDI controllers, for example.

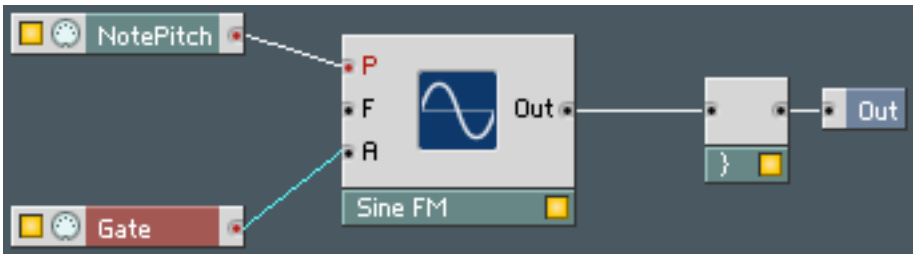
We will now start making our patches sensitive to MIDI note messages. This way, we can begin using a MIDI keyboard to control our patches more intuitively. (If you don't have a MIDI keyboard hooked up to your computer, Reaktor uses the computer keyboard to send MIDI note messages. Chapter 21 of the Reaktor manual contains a useful chart of the MIDI notes triggered by each computer key.)

The modules of the **MIDI In** category each receive various types of incoming MIDI messages. Add a **Note Pitch** module to your patch and connect it to the pitch inlet of the **Sine FM** oscillator.



The **Note Pitch** module outputs the MIDI pitch of every note-on message received by Reaktor. Since the pitch inlet of the oscillator expects a MIDI note number to set the oscillator's frequency, this is a wise pairing. Turn on Reaktor's audio for a moment and play some notes on the MIDI keyboard (or the computer keyboard). Assuming everything is configured correctly, you will hear a sine wave with the frequency of whichever key you pressed last.

Our first patches were made to endlessly drone, but musical sounds usually have an articulated amplitude. Let's use the keyboard to gate the oscillator's amplitude. Insert a **Gate** module from the **MIDI In** category. Delete the **Constant** and connect the **Gate** module to the oscillator's amplitude input.



The **Gate** module takes the velocity of any note message and transforms it to a set range. Double-click this module to call up its **Properties** window and go to the *Function* page.

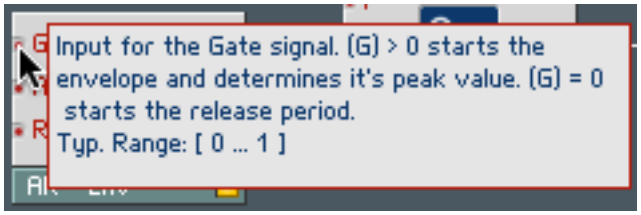
The *MIDI* input range is set to MIDI's standard, 7-bit limit of 0-127, and the *RANGE* for output is set from 0-1. This is ideal for transforming incoming velocity data into usable amplitude values. The ranges could be edited but seem perfect for our purposes. (Many **MIDI In** modules have these same options, including the **Note Pitch** module.)

Once again, turn audio on and play the keyboard. Sound will only be produced when you hold a key down. Once you stop playing, **Gate** receives a note-off message from the keyboard and correspondingly sends out a value of 0, causing sound to stop. Play the keyboard both gently and strongly to observe the different volumes produced. (Keep in mind that only if your keyboard is velocity sensitive will the notes you play vary in volume. Many keyboards have this feature, but some — including the computer keyboard — do not.)

The problem is that most musical sounds do not jump immediately from silence to full volume as the **Gate** signal does. Please delete the cord connecting the **Gate** and **Sine FM** modules. (Before selecting the patch cord, click on the background to be sure nothing else is selected.) There is a type of module called an *envelope generator* that transforms a gate signal into a sloping, time-varying signal. Please add the **AR** module to our patch from the **LFO, Envelope** category and connect its output to the amplitude inlet of **Sine FM**.

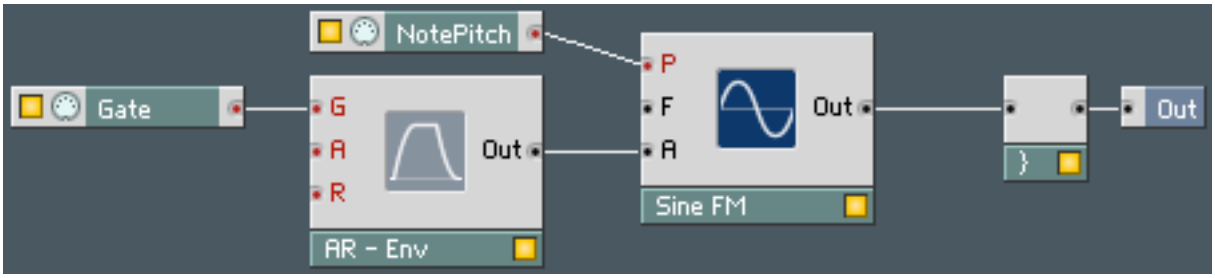
(Apparently this module is really named **AR - Env**. One of Reaktor's inconsistencies is that some modules show different names in different parts of the program.)

Envelope generators are named for the parameters they possess. *AR* stands for *attack-release* because these are the two adjustable segments for this type of envelope. Both are time settings which determine how long the signal glides up to its full strength (attack) and the time it takes to return to zero (release). Both of these stages are triggered by the so-called *gate* inlet. Mouse over the gate inlet (*G*) to figure out how this inlet works.



By "gate signal", Reaktor means a signal with two states: either on or off. Any incoming value greater than 0 starts the envelope and is also used as the maximum value of the

envelope signal. An incoming value of 0 triggers the release stage of the envelope. The values provided by the **Gate** module are a good match for the gate inlet of **AR - Env**. Go ahead and connect **Gate** to this inlet.



The remaining two inlets are for the attack and release parameters. Use the **Create Control** function for each of these inlets. Also, organize the knobs however you like in the ensemble's panel window and then lock the panel.

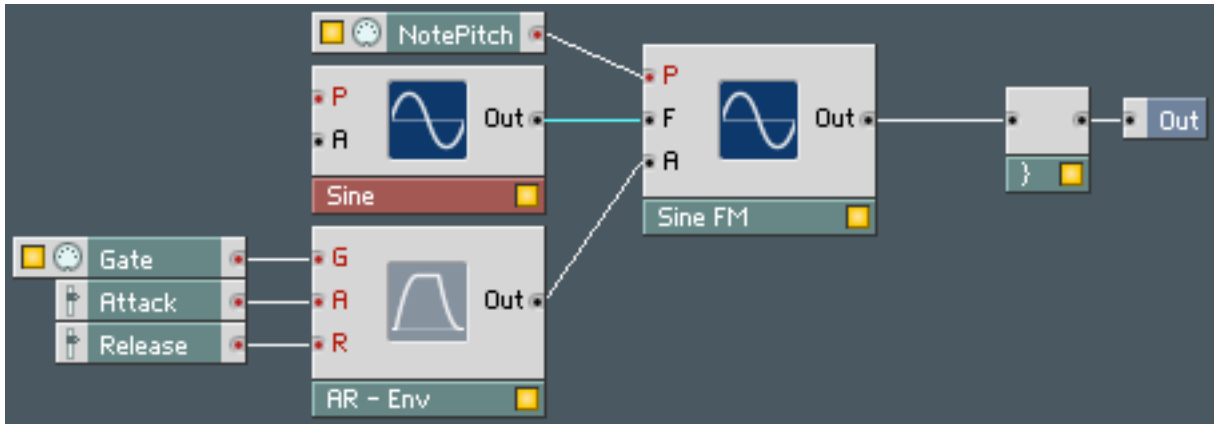
Adjust both the attack and release times as you play notes on the keyboard. Smaller values represent shorter times.

In summary, envelope signals are time-varying, segmented, aperiodic signals. In our patch, the envelope signal is not directly connected to the *audio signal path* because it does not lead directly to the speakers. We don't hear the envelope itself, but we do hear its control over the sine oscillator, resulting in loudness changes. Therefore, it is part of the *control signal path* which manipulates the audio signal path but is not heard by itself.

Using an envelope to modulate our oscillator's amplitude provides a sense of musical articulation. Let's add a sense of musical vibrato to our patch using the frequency inlet of the **Sine FM** module. Vibrato could be defined as a slight, periodic modulation of an oscillator's frequency. A *periodic* signal is one that repeats its waveform over and over again at a specified frequency. We cannot use an envelope for this effect since an envelope signal is *aperiodic*, meaning it goes through its cycle only once and therefore has no frequency.

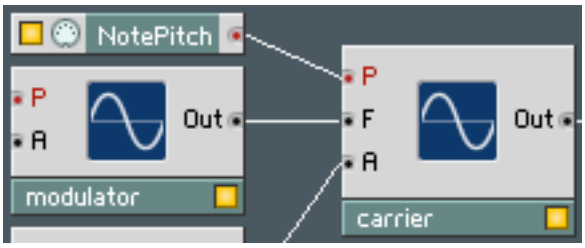
A sine oscillator is a good example of a periodic signal generator. Add a **Sine** module to our patch (we don't need the extra frequency inlet of a **Sine FM** module) and connect its

output to the frequency input of the **Sine FM** module.

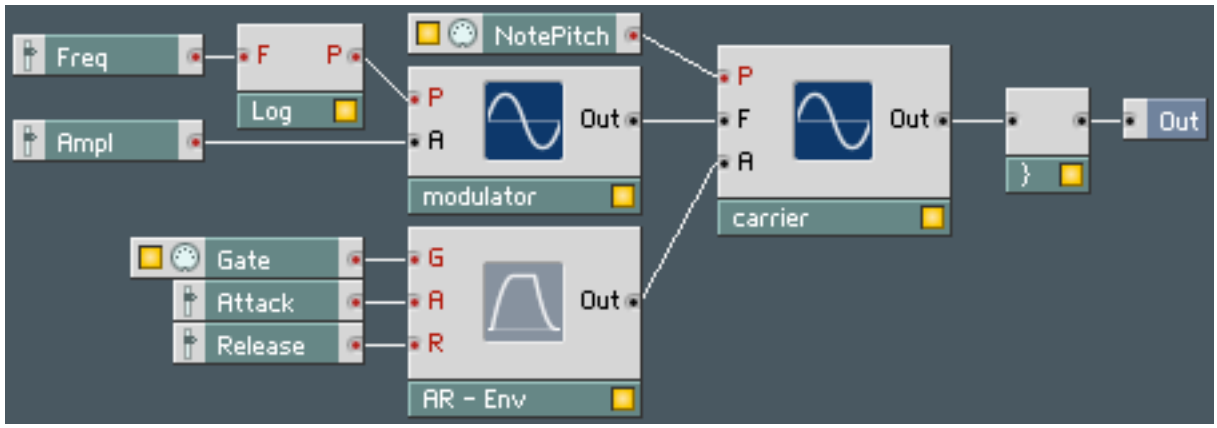


Now that we have two sine oscillators, let's label them by function to make editing easier. When two oscillators are connected like this, the oscillator that we can hear is called the *carrier* while the one modulating the carrier is called the *modulator*.

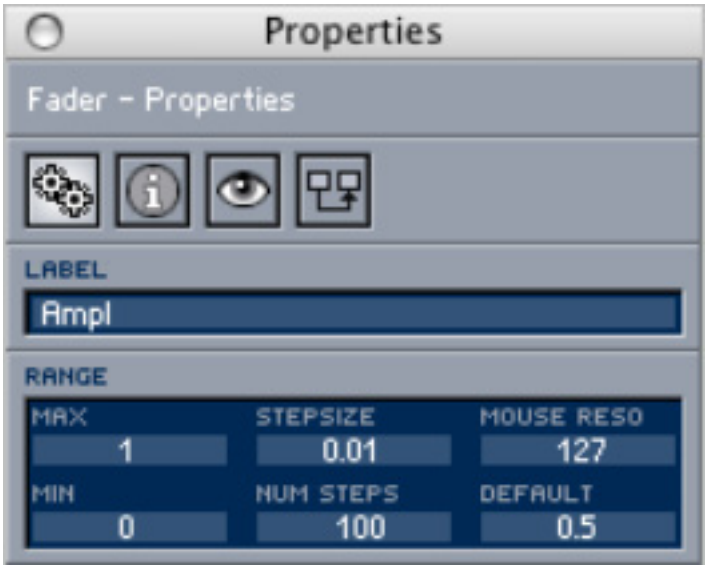
Call up the **Properties** window for the **Sine FM** module. In the *Label* field, change *Sine FM* to *carrier* and press the return key. Keep the **Properties** window open and click on the **Sine** module. (The **Properties** window is actually a floating window that displays the information of whatever object is currently active within Reaktor; it is not tied to one module.) The **Properties** window now focuses on the **Sine** module. Change its label to *modulator*. Leave the **Properties** window open.



It makes more sense to control vibrato in Hertz rather than MIDI note values. Insert a **Log. (F)** module from the **Math** category and connect it to the pitch inlet of the *modulator*. Use the **Create Control** feature on both the frequency inlet of the **Log** module and the amplitude inlet of our *modulator*.



Click on the new amplitude control (*Ampl*). Please go the the *Function* page of the **Properties** window.



This standard amplitude range works well for audio signals, which ideally vary from +1 to -1 as mentioned earlier. This oscillator, however, is acting as a *modulator* whose total amplitude modulates the carrier in Hertz (Hz). We need a range wider than +/- 1 Hz to achieve effective vibrato. Change the *MAX* setting to 10. Now the output of this modulator goes between +10 and -10 at maximum, effectively adjusting the carrier's frequency by +/- 10 Hz. Please click on the frequency control (*Freq*).

Reaktor has used the default frequency range for this knob, from 0 - 5000 Hertz with a *STEPSIZE* of 50. Vibrato is a relatively slow modulation, but keep these settings the way they are for a moment as an experiment. Organize the knobs in the panel window and adjust the frequency and amplitude of the *modulator* as you play some notes. Frequency modulation in the pitched frequency range (20+ Hz) produces complex timbre changes. This topic is worth researching on its own beyond this document. Let's go back to making vibrato now.

Vibrato occurs below the pitched frequency range. For the *Freq* knob, let's change the *MAX* of the range to 10. Now our *modulator* oscillates between 0 and 10 cycles per second, and our *STEPSIZE* is a fine 0.1 Hz. Again, adjust the *modulator* settings as you play some notes.

Notice how the frequency of the *modulator* controls the speed of vibration and the amplitude of the *modulator* controls the depth of vibration. Relabel these **Fader** modules accordingly.

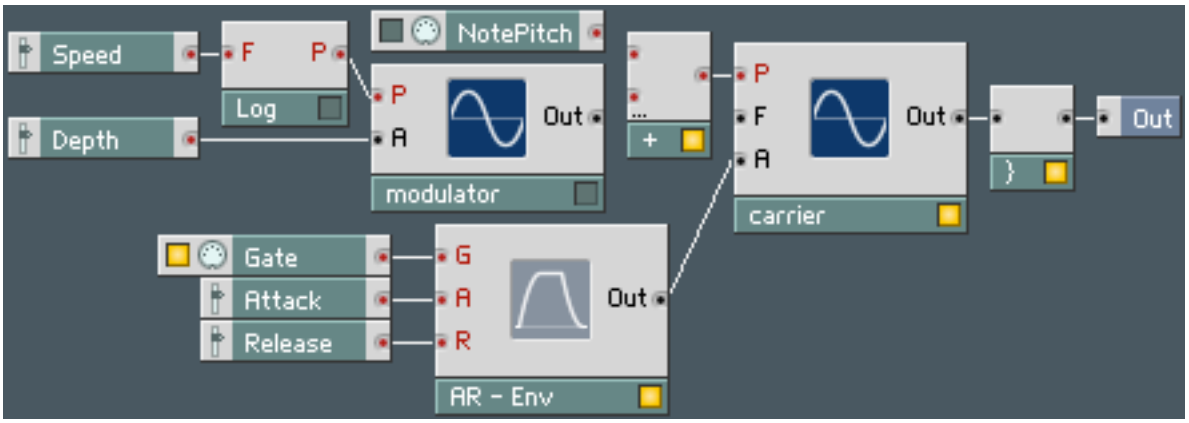
While our *modulator* and *carrier* oscillators are basically the same module doing the same thing, the way they are being used in the patch causes enormous differences in their behavior. With our *carrier* (or any oscillator in the audio signal path to the speakers),

frequency determines pitch and amplitude determines loudness. But with our *modulator* (in the control signal path, directly modulating the carrier), frequency determines the speed of modulation while amplitude determines the modulation's depth.

As you play up and down the keyboard, you will notice that the vibrato is much deeper for low notes than high notes. This is because we have set the vibrato's depth in absolute Hertz, and lower pitches have fewer Hertz between them. To make all notes vibrate equally, let's switch our vibrato to modulate by pitch instead of frequency.

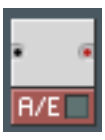
To get both our MIDI keyboard (via the **Note Pitch** module) and our *carrier* oscillator controlling the pitch parameter, we must combine their signals. From the **Math** category, insert an **Add** module.

The **Add** module outputs the sum of the inlets. Please delete the wire connecting the *modulator* to the frequency inlet of the *carrier* and connect the **Add** module to the pitch inlet.



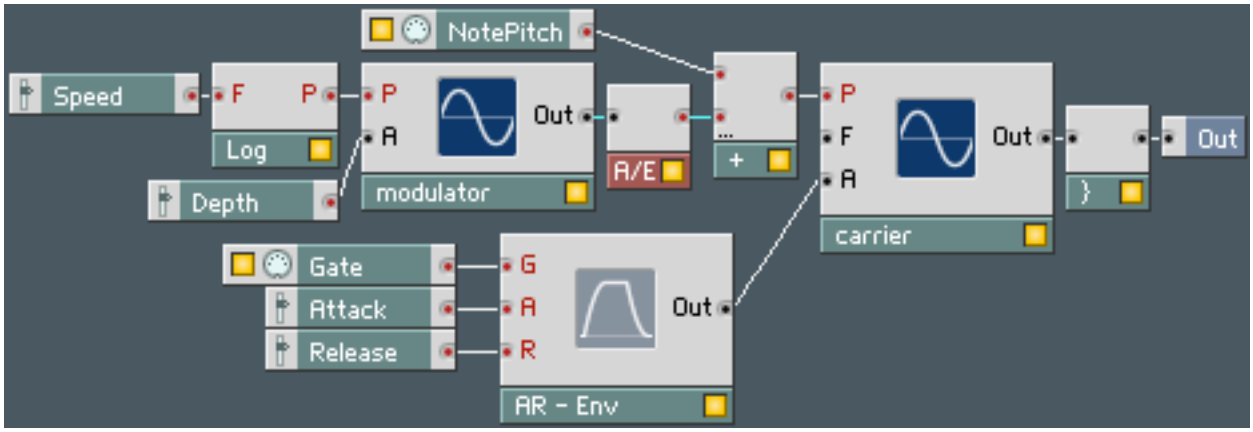
(Since only one cord can be connected to an inlet, making the new connection from the **Add** module automatically breaks the old connection between the *modulator* and *carrier* modules.)

Once we connect the hybrid **Add** module to an event signal port, you will notice both inlets become event signal inlets. As mentioned earlier, event signal inlets cannot directly handle audio rate signals (like the one coming from our *modulator*). Fortunately, there are modules which allow us to downsample audio signals into event signals. From the **Auxiliary** category, insert an **A to E** module to our patch.



This module (called **A/E** within the structure environment) samples an incoming audio signal

at a specified control rate (400 Hz by default) to create an event signal that is sent to the outlet. Connect *modulator* to the **Add** module via the **A/E** module. Connect the **Note Pitch** module to the other **Add** inlet.



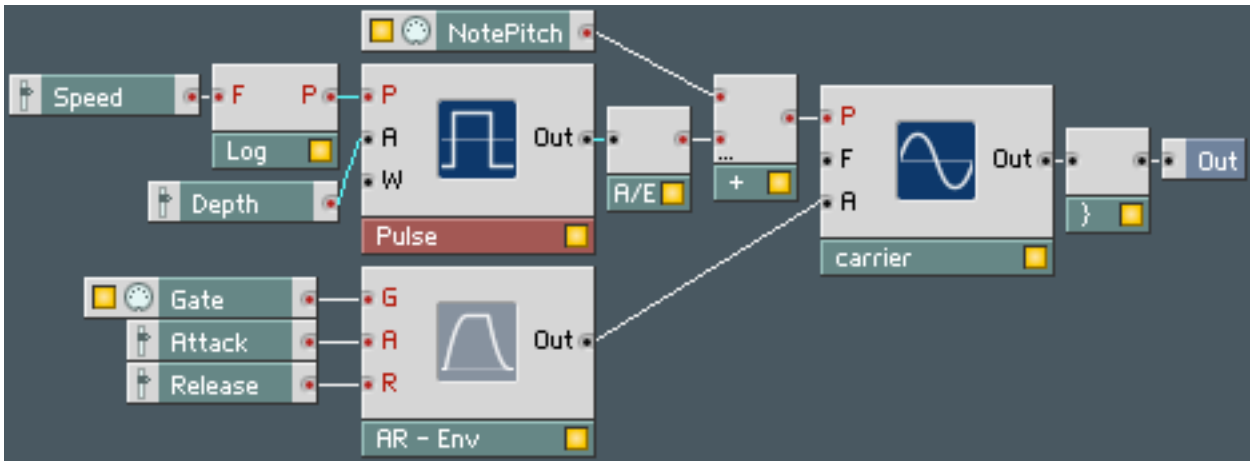
As you play the keyboard and readjust the knobs, remember that our *depth* control still has a range intended for Hertz instead of semitones. Change the *MAX* setting of this **Fader** module to 1 for a more reasonable range of vibrato.

The lesson from before is still highly important; a signal's function is defined solely by its destination. In other words, modules don't have functions until they are connected. They act in particular ways but can be used however you see fit. That is the true strength of modularity.

We have now seen examples of aperiodic and periodic signals modulating various parameters, but our patch did not have to look this way at all. We could just as easily swap the destinations of the modulator and envelope generator. (Go ahead and try this.)

There is no limit to the number of possibilities. Through studying types of modules and their typical functions, however, you will best equip yourself to more accurately predict what a certain configuration of modules will achieve in function and sound.

As a final thought about this patch, predict how the sound would change if we used a pulse oscillator (such as **Pulse**, which has a square shape by default) as our modulator.



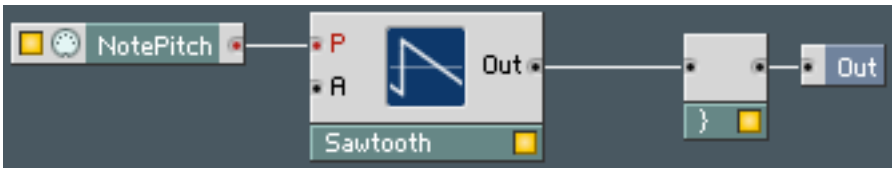
The waveshape of our carrier helps determine tone color, but how does the waveshape of our modulator affect the sound?

-- - --- - --

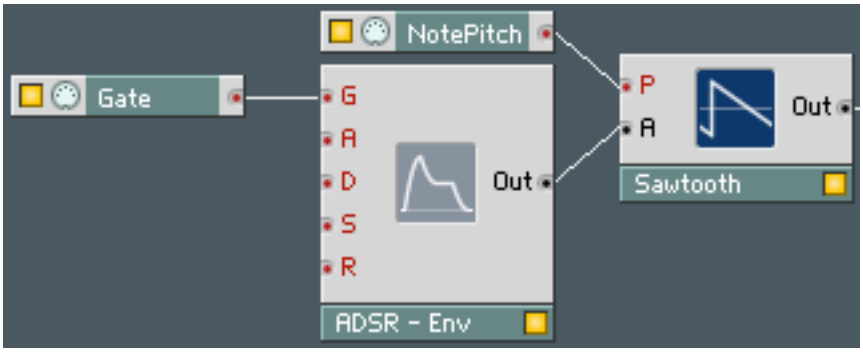
We have covered a good deal of the nuts and bolts of Reaktor. While there are many complicated modules that have not been mentioned, those will not be discussed here. Instead, we will continue on the same path of adding basic modules to extend our patches' functionality and our own fluency.

Please recreate or open your copy of the ensemble patch described in Tutorial 01.

Let's start this lesson's patch by adding a **Sawtooth** oscillator (the same as we used in Tutorial 03) and connecting its output to the **Audio Voice Combiner**. Use the **Note Pitch** module (from **MIDI In**) to control the oscillator's pitch.



For this tutorial, we will use the most common type of envelope generator. From the **LFO, Envelope** category, add an **ADSR** module to your patch and connect it to the amplitude inlet of the **Sawtooth** oscillator. Also add a **Gate** module from the **MIDI In** category and connect it to the envelope's gate inlet.



First of all, this is called an **ADSR - Env** module inside the structure environment. This type of envelope — like the **AR - Env** — sustains until it receives a gate signal of 0, but you also set the level at which this envelope sustains. The *S* parameter (sustain level, typically from 0-1) represents the strength of the sustain portion relative to the gate signal's strength.

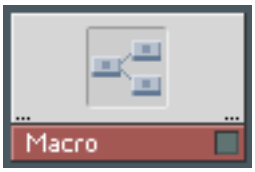
While using the **Gate** module for the envelope's gate input, the parameters work in the following way: *A* (attack) represents the time the envelope takes to rise to the gate level once a note-on message is received. *D* (decay) represents the time to reach the sustain level once the attack stage is finished. *S* is the relative level (multiplied by the gate signal) of the envelope after the attack and decay segments are finished. *R* is the time to return to zero once the note is released.

Use **Create Control** on each of these parameters and experiment with different settings.

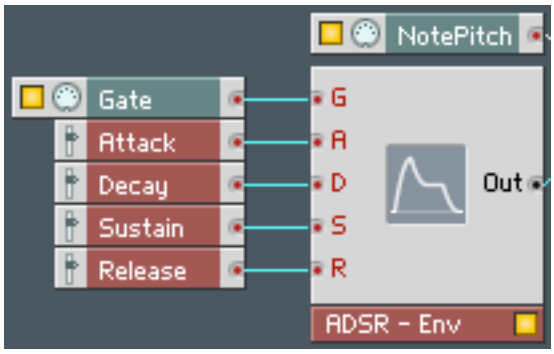
An *attack-decay-sustain-release* envelope offers a range of flexibility much greater than the attack-release model since we now have a level control (sustain) as well as three timed segments.

As mentioned previously, a great strength of modular environments is that each module is reusable and can be applied in many different contexts. Often times, it is desirable to take a group of modules that achieve a single function together and make them into one reusable unit. This is made possible in Reaktor by using *macros*.

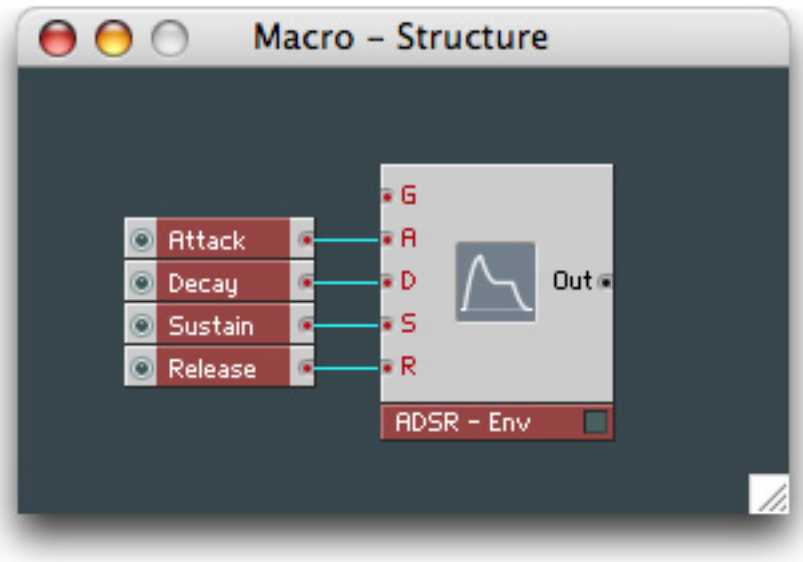
Let's use the **ADSR - Env** module and its associated controllers as an example. First, control-click on a blank area of the instrument's structure window and choose **Insert Macro > _New - Empty**. This creates an empty macro.



As you can see, the icon looks like several modules that are interconnected. This is exactly the kind of thing that will go inside a macro. Double-click the icon, and a blank structure window will appear, showing you the inside of this unit. Back in the instrument's structure window, click and drag to select only the **ADSR - Env** and the four **Fader** modules connected to it.

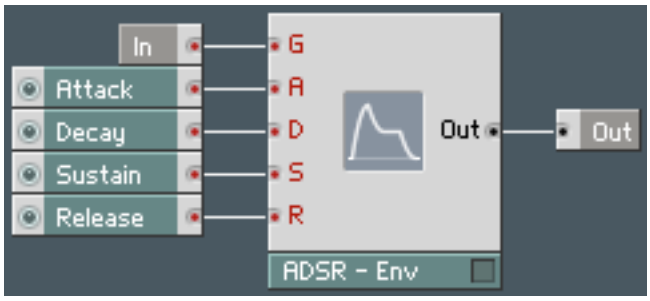


Cut this selection using the **Edit** menu. Bring the macro's structure window to the front and select **Paste** from the **Edit** menu.



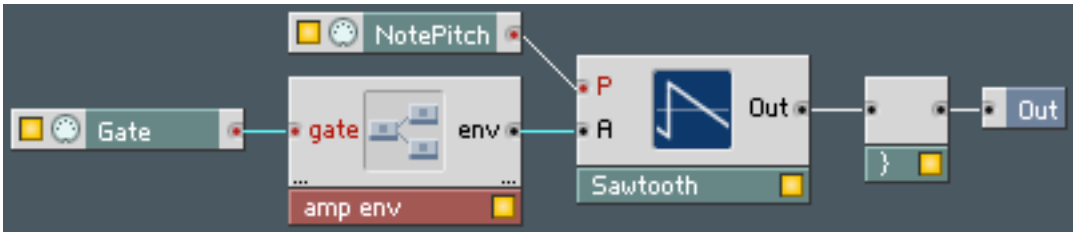
One thing to notice is that all of our **Fader** modules now have a knob icon instead of the fader icon from before. Reaktor meant to show us this from the beginning. Since the **Fader** modules are all displayed as knobs in the panel window, the knob icon we now see is correct. (With a **Fader** module, the *TYPE* of controller can be changed under the *Appearance* page of the **Properties** window.) These icons are also updated when you save and reopen a patch or simply close and reopen the appropriate structure window.

We need to connect the **Gate** module to the **ADSR - Env** as well as send the envelope signal out to the **Sawtooth**. **In Port** and **Out Port** modules are used to connect our macro with other units in our patch, just as instruments do. Add one of each to our macro from the **Terminal** category. Connect the **In Port** to the envelope's gate inlet and the **Out Port** to the envelope's outlet.



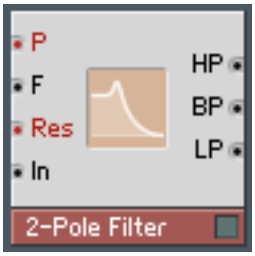
Connect the macro appropriately within the instrument. The ensemble will now work exactly as it did before. Play the keyboard to make sure everything is working.

To make things more intuitive, let's relabel the *In* as *gate* and the *Out* as *env*. Also, pull up the macro's **Properties** window and label it *amp env* to represent the function it serves.



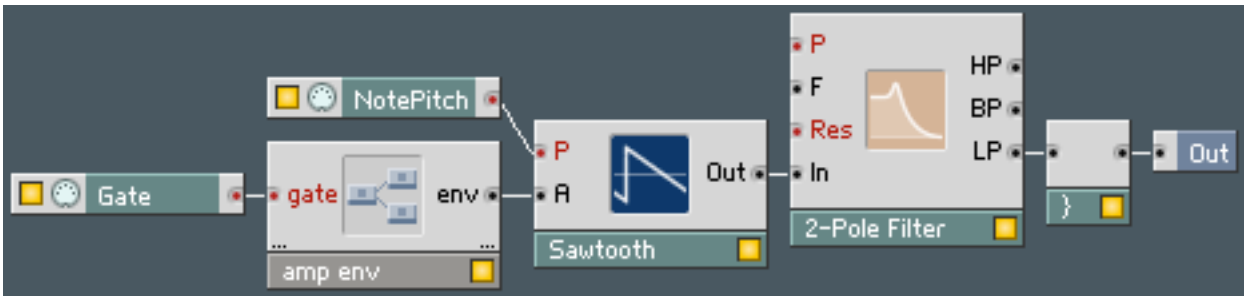
Using macros is an excellent way to encapsulate different elements of your patch while making the structure window easier to read. We could have placed the **Gate** module inside the *amp env* macro, but by leaving it on the instrument level, it is still accessible for additional connections. Also notice how the panel window has been organized so that the contents of *amp env* are visually grouped together and can be moved as a unit.

One of the basic synthesis modules we have not discussed yet is the *filter*. A filter is a signal processor that gradually removes certain frequencies by sloping down at a set *cutoff frequency*. From the **Filter** group, add a **Multi 2-Pole FM** module to the patch.



Filters have different behaviors — called *modes* — for selecting which frequencies to keep and which to remove. The three outlets of this object represent three different filter modes, each named for the part of the spectrum you want to keep. The first outlet (*HP*) signifies *high pass* mode which keeps the sound above the cutoff frequency and gradually removes the part below. The third outlet (*LP*) represents a *low pass* mode which passes the lows, and the middle (*BP*) is a *band pass* mode which combines the other two modes, leaving only a center 'band' of sound around the cutoff frequency.

The bottom inlet — labeled *In* — is for the signal we want to filter. Connect the output of our **Sawtooth** module to this inlet of the filter. Connect the low pass outlet of the filter to the **Audio Voice Combiner** module.



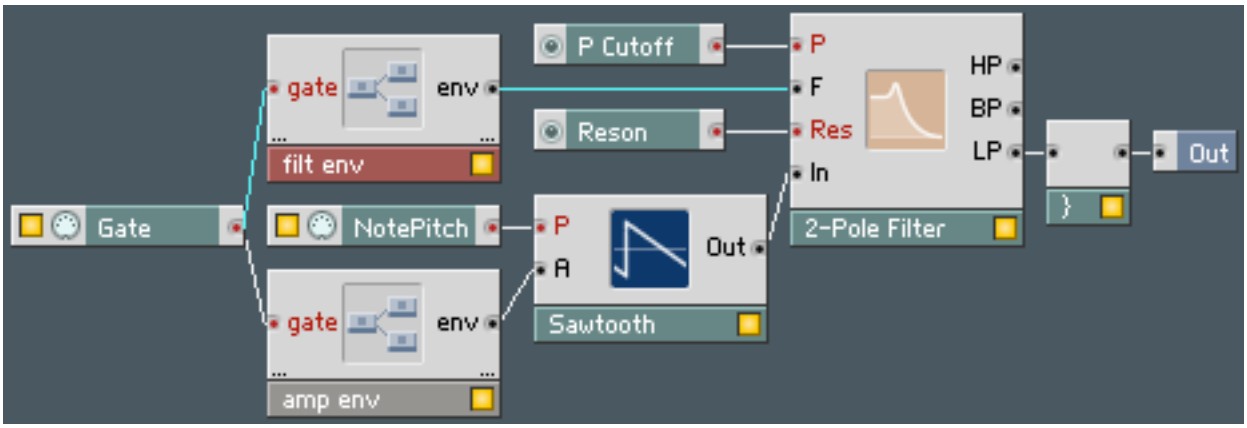
Technically speaking, the cutoff frequency is the point at which the signal is reduced by about 70% (-3 dB). In our **Multi 2-Pole FM** module (which, yes, is called **2-Pole Filter** inside the structure environment), the cutoff frequency is the sum of the *P* and *F* inlets (representing pitch and frequency, as before). Use the **Create Control** function for the pitch inlet.

The *Res* inlet represents a feedback parameter called *resonance*. This parameter sets the amount of output signal being sent back to the processor's input and is usually in the range of 0-1. As resonance increases, the cutoff frequency is amplified and the pass band is attenuated. Use the **Create Control** function for the resonance inlet.

Play some notes and adjust the *P Cutoff* knob. As the cutoff pitch goes lower, the sound becomes darker, and as it rises, the sound opens up by allowing more high frequencies. Adjust the *Res* knob as well. As you turn this knob up, the cutoff frequency will get louder, and the sound will become thinner.

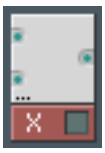
Let's apply what we learned last tutorial and add another envelope generator to modulate our filter's cutoff frequency. This is much easier since we turned our amplitude envelope into a macro.

Select our *amp env* in the instrument structure window and choose **Duplicate** from the **Edit** menu. Relabel the new macro as *filt env* and connect its output to the frequency inlet of the **2-Pole Filter**. Adjust the parameters and play some notes.

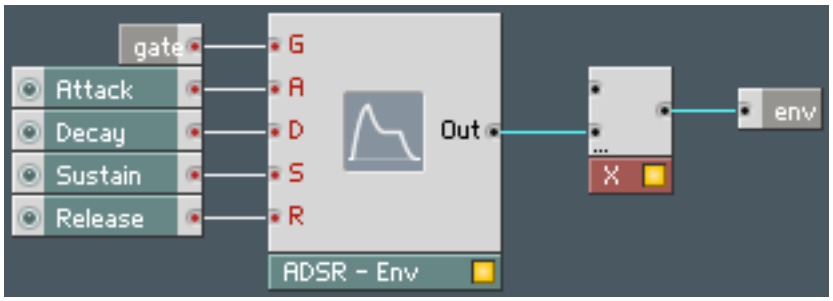


You will notice that the filter envelope currently has little to no effect. Remember the signal flow driving the filter envelope. The MIDI **Gate** module outputs values from 0-1 which we are using as the gate signal for the **ADSR - Env**. Since this envelope signal is controlling the frequency parameter of the **2-Pole Filter**, our cutoff frequency is being modulated by 1 Hertz at most.

To hear an audible difference, we need to scale the filter envelope to a higher range. Scaling is easily done with math by multiplying. In the *filt env*'s structure window, insert a **Multiply** module from the **Math** category.



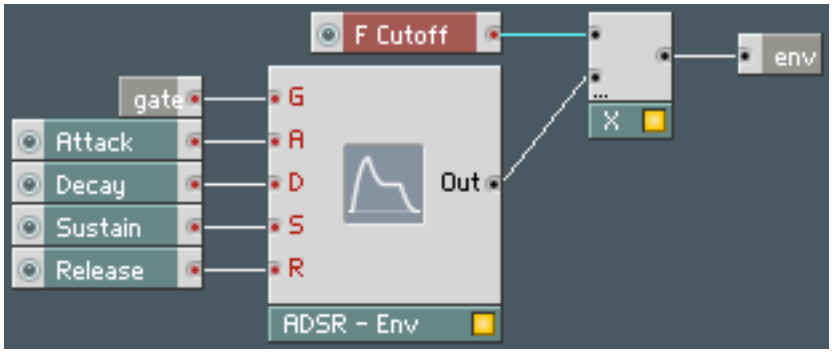
The signals connected to the module's inlets are multiplied, and the product is sent to the outlet. Connect the output of the **Multiply** to the **Out** module (labeled *env)* and the outlet of the **ADSR - Env** to one of the **Multiply** module's inlets.



As you can see, the ports on the **Multiply** module turn black once you feed it an audio signal.

To successfully scale the envelope's output to a new range, we need to send the maximum value we want to the other inlet of **Multiply**. Instead of guessing a range for the frequency inlet of the **2-Pole Filter**, let's cheat a little.

In the instrument's structure window, use the **Create Control** feature on the frequency inlet of the **2-Pole Filter**. **Cut** (using the **Edit** menu) the new **Fader** module which is labeled *F Cutoff*. Reconnect the *filt env* macro to the filter's frequency inlet. In the *filt env's* structure window, **Paste** the *F Cutoff* module and connect it to the other inlet of **Multiply**.



As you play notes on the keyboard, adjust the *F Cutoff* knob in the panel window. The effect of the envelope should now be distinctive.

If you wanted, you could switch the filter envelope to manipulate pitch instead of frequency as we did with the modulator in the last tutorial. [Hint: you might want to use the **Note Pitch** output to help control the cutoff in that situation.] Try that idea sometime if you like.

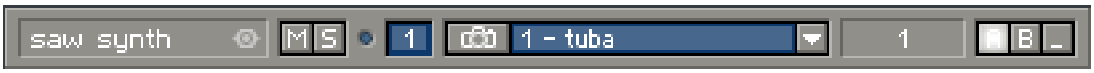
Since the **Fader** module does not control the actual cutoff frequency but rather scales the envelope signal which modulates the cutoff, let's relabel the **Fader** to something more appropriate.

Since we have come this far, we might as well label our instrument, too. In the ensemble's structure window, double-click the name of our instrument (currently called *Instrument*) so the **Properties** window appears. Relabel the instrument as *saw synth*. The change will be reflected in the panel window as well.

So far, we have spent all of our time in Reaktor constructing ensembles and adjusting their parameters. It would be wasteful, however, to save a new ensemble file every time we find a combination of knob settings we like. Reaktor has the capability to embed *snapshots* of all the panel settings within the ensemble file.

First, adjust the settings of the **Fader** modules in the panel window until you find a sound you like. Then in the instrument's panel window, click the camera icon beside *1 - <empty>*. This will open the **Snapshot** window.

There are many ways in this window to save a snapshot. The easiest method is to click the *APPEND* button which selects the first blank memory slot and lets you name your new snapshot. Click the *APPEND* button, type a name for the snapshot, and hit return. The panel window will now display the snapshot's name.



(Please name your snapshot something appropriate. If your settings sound like a tuba, feel free to name your snapshot *tuba* as well.)

Now adjust the knobs again to create a sound very different from your first snapshot. Once you are happy with the sound, click *APPEND* and name the new snapshot appropriately.

Every snapshot that is created appears in the drop-down menu beside the snapshot name. Go ahead and click the little triangle to the right of the current snapshot name and select snapshot *1* from the menu that appears.

You will notice that all the **Fader** modules have returned to the setting of the first snapshot. Play the keyboard to make sure this is the proper sound. Try switching between the snapshots as you play.

We are now at the end of our tour. Hopefully Reaktor makes a bit more sense now.

We have discussed only a few of the many ways to achieve things in Reaktor. Find the work flow that is most comfortable for you (key commands versus mouse, for example).

Please remember that many of Reaktor's functions have been completely ignored for this primer. The program's manual should be consulted for all additional topics.

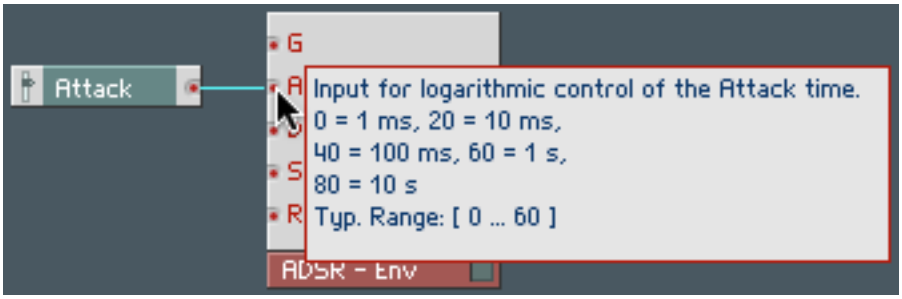
-- - --- - --

This entire document has been written to assist in learning the basics of Reaktor, but Reaktor is only one of many available software applications in the field of sound synthesis. It would serve us well to briefly observe some of these other platforms.

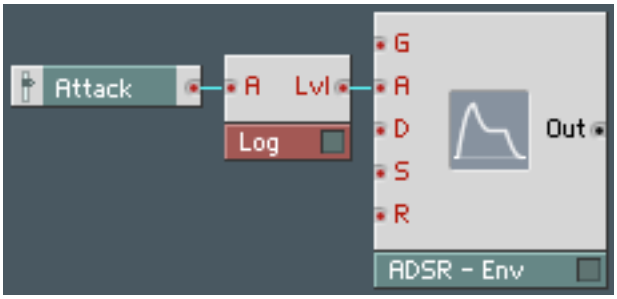
Here are a couple details you should know about Reaktor before looking at other environments.

While Reaktor uses an amplitude-based model (giving you an amplitude parameter for all oscillators), most systems have droning oscillators and require the use of an separate amplifier (also called multiplier or scaler) to adjust the signal's amplitude using multiplication.

Also, Reaktor's envelopes set their time segments on a logarithmic scale.



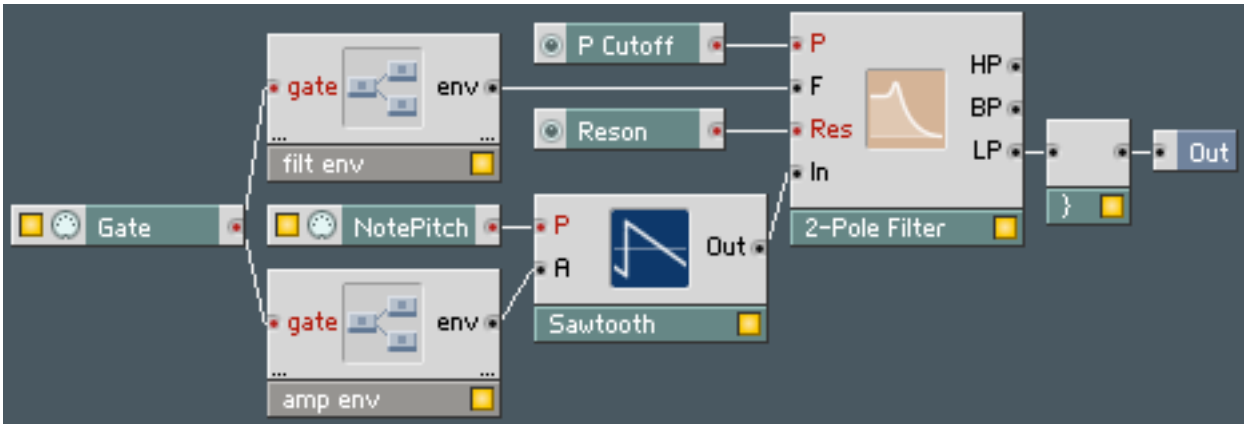
If you wanted to set envelope times in actual milliseconds, the **Log (A)** module (from the **Math** category) handles the conversion properly.



The linear amplitude to logarithmic decibel conversion only works in this situation because Reaktor uses the exact same logarithmic scale for time units . The port labels are completely wrong for this usage, but any number you put in will be treated as a millisecond value.

Likewise, be sure to remember that our patch defines cutoff frequency in MIDI pitch value. If you wanted to use Hertz, the **Expon (F)** module would work.

Here was our final patch from the tutorials.



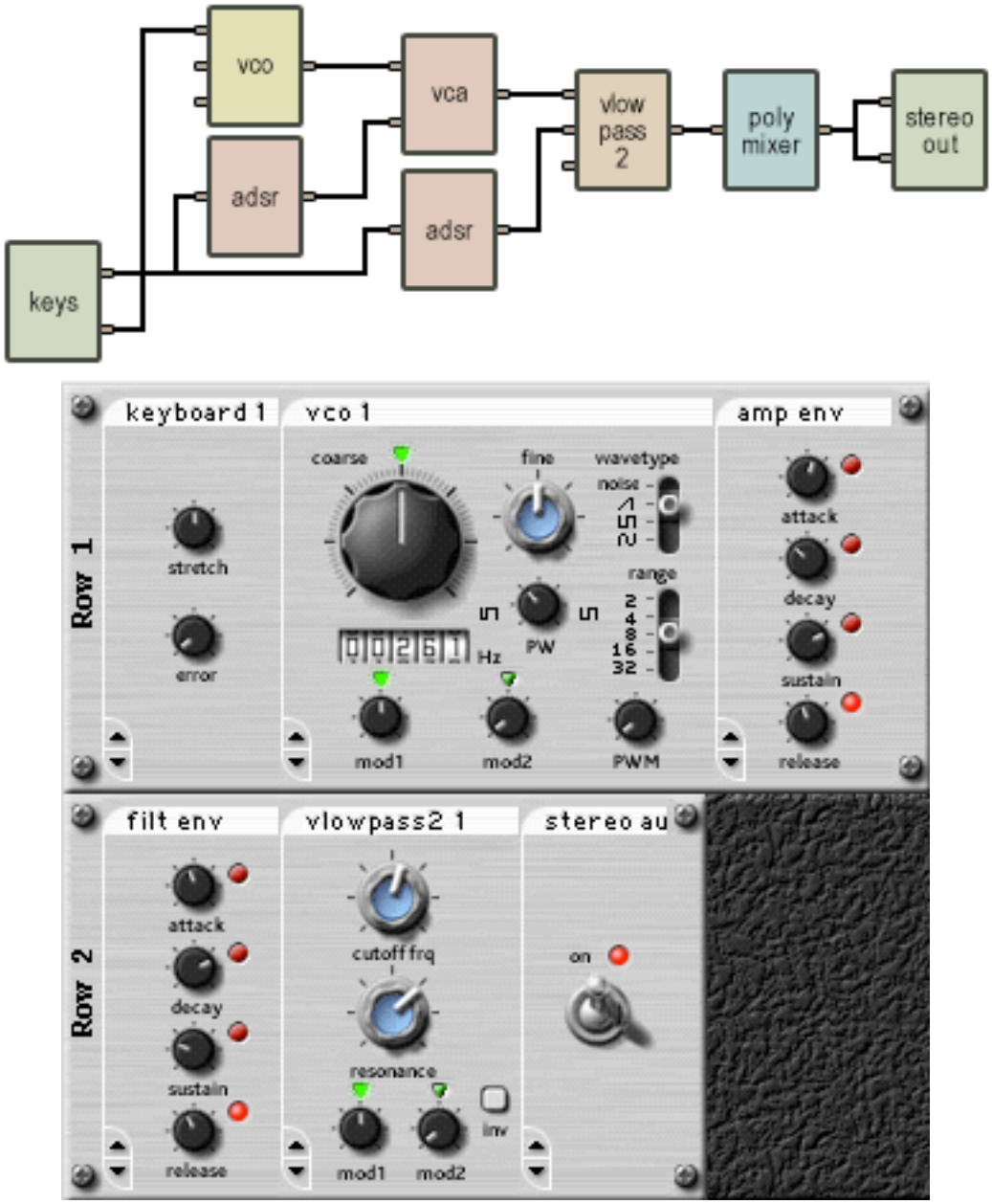
A fair description of this system would be a sawtooth oscillator passing through a 2-pole, low pass filter — which has controls for cutoff frequency and resonance — and two envelope generators, one for amplitude and one for filter cutoff. With these general terms, we can reconstruct our system on other platforms. As with anything, observing the same idea in different contexts will help to strengthen our understanding of the underlying concepts.

Recreating the system is the larger goal, but here is the patch we will recreate.

As said before, the biggest part of learning a new platform is understanding its vocabulary of objects and its quirky behaviors. The theory part largely remains the same. The following pages show this same system and patch realized on other platforms. These are all software applications (with one exception). Each program has its own particular strengths and weaknesses. There is no exclusive winner in this field of software.

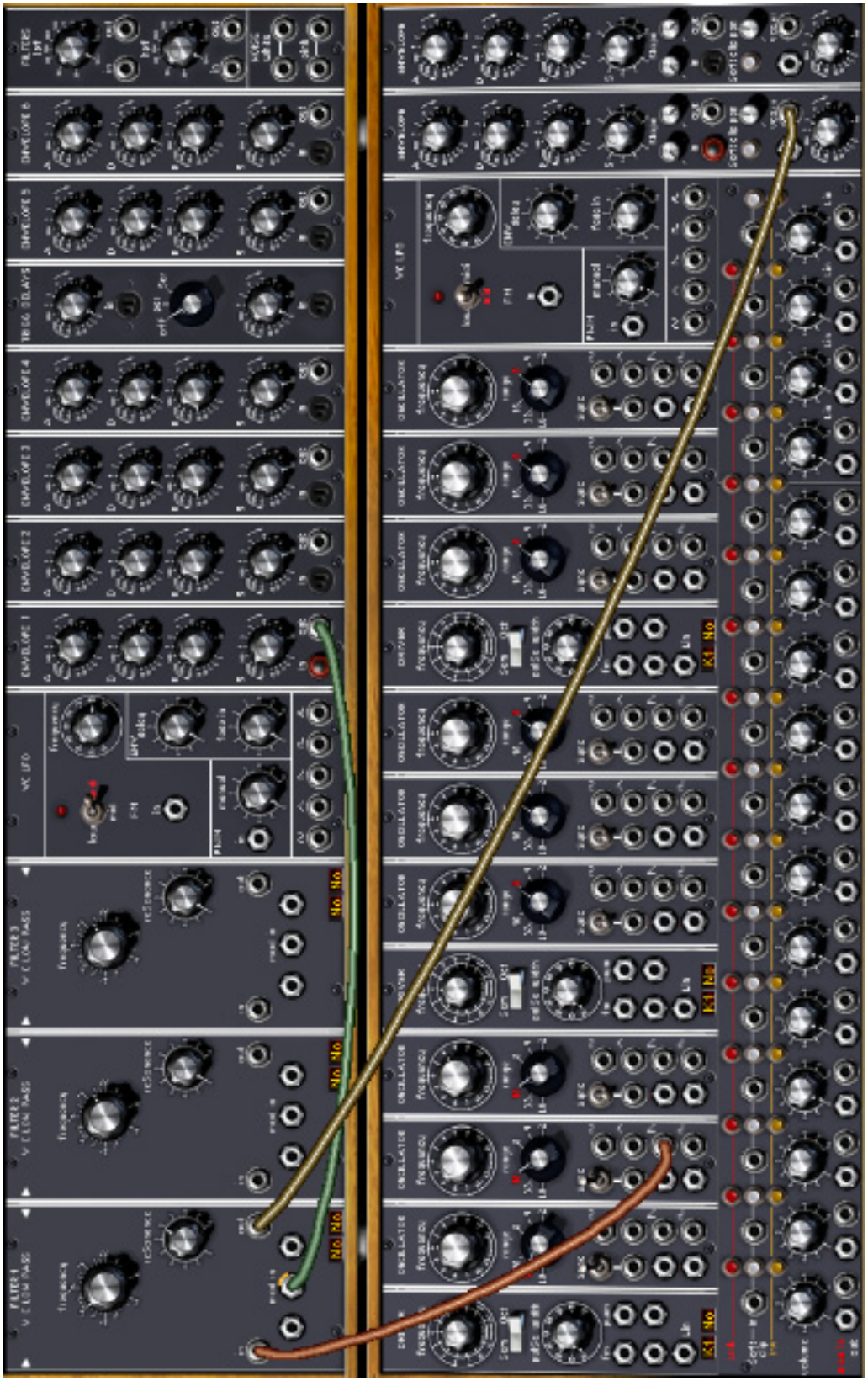
After all, it is all digital.

Applied Acoustics Systems' **Tassman**



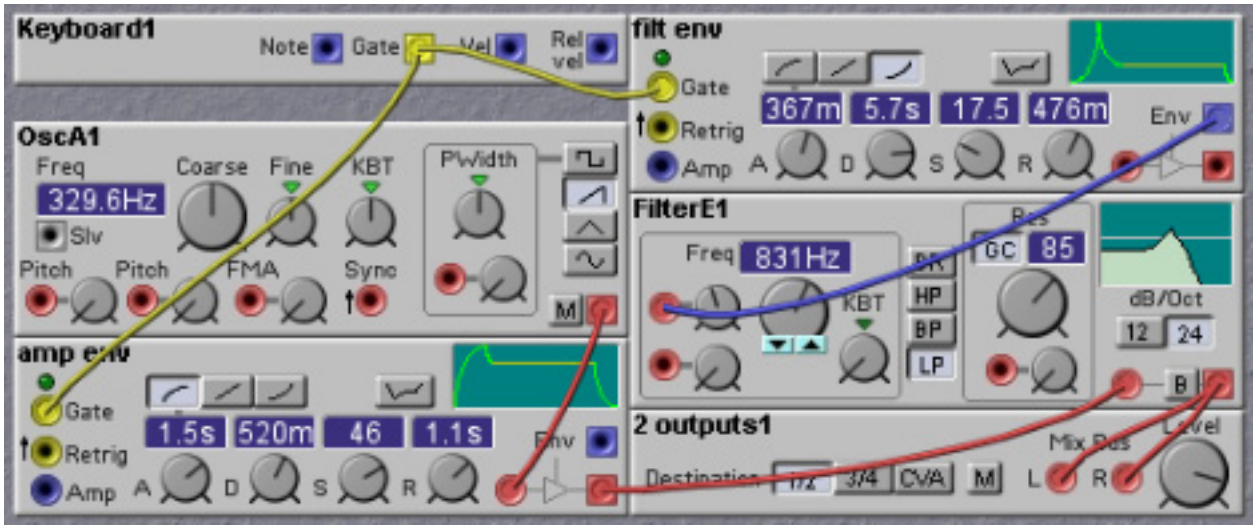
Like Reaktor, Tassman is a modular software synthesizer that has a structural window and a separate graphical front end. One major difference is that the graphical panels are predefined for each module. Also, each modulation input provides an attenuator on the corresponding graphic panel, making additional scalers unnecessary.

Arturia's **Moog Modular V**



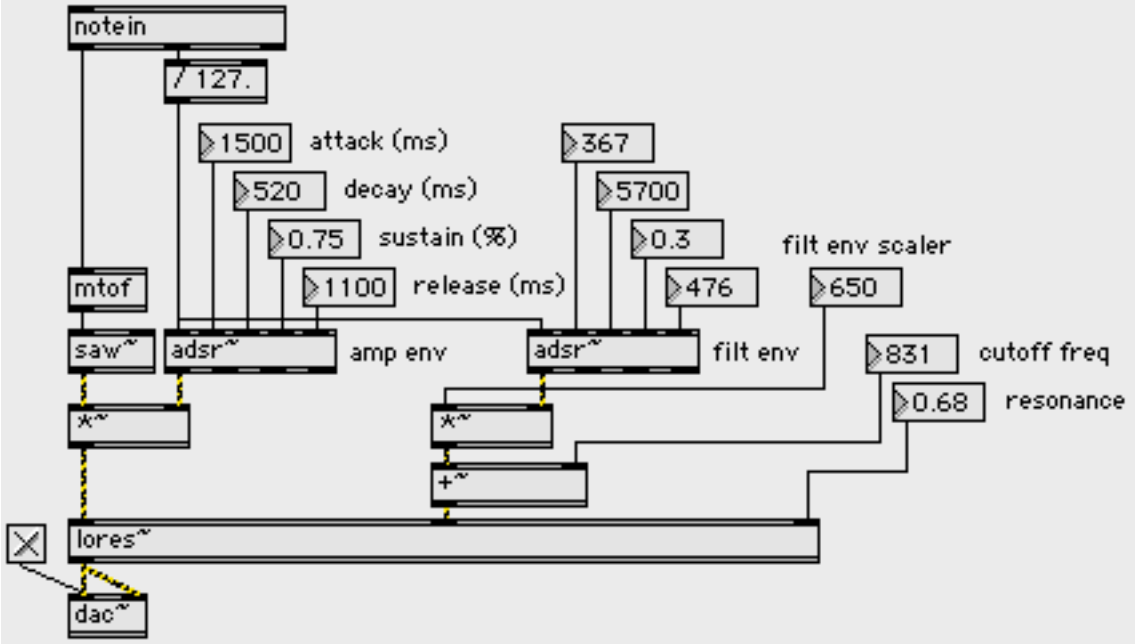
Moog Modular V is a digital software instrument built to emulate the analog modular synthesizer originally made by Moog Music. Even though the number of modules and their type is limited, it is a completely modular system in that no connections are hardwired. The only thing that might seem peculiar is the way the audio outputs have a stereo amplifier controlled by an envelope generator.

Clavia's **Nord Modular**



Nord Modular is a hardware synth that contains dedicated DSP chips and is programmed with its companion software editor. Some modules on the Nord have a *KBT* (keyboard tracking) parameter for connecting the keyboard's pitch messages to certain parameters (as is enabled with our oscillator, *OscA1*, to manipulate the oscillator's pitch). Also, the envelope generators have a control signal outlet (blue) and an audio inlet and outlet (red). The implication here is that the audio input is sent to an amplifier which is controlled by the envelope.

Cycling ‘74’s **Max/MSP**



Max/MSP is a specific application but is somewhat closer to a programming language because of its open-ended nature (you can program your own modules) and purpose (it can be used for MIDI manipulation/generation, audio systems, building audio plug-ins, video processing, et cetera). *notein* provides the pitch, velocity, and channel of each note message received by the selected MIDI port. Velocities are being divided by 127 to move them from the MIDI range (0-127) to a logical audio scaler range (0-1). The *mtof* object translates MIDI pitches to frequencies (Hz), the same as the **Expon (F)** module in Reaktor. The *dac~* object provides access your audio output device, much like the **Audio Output** module.

Emagic's **ES2**

ES2 is a software instrument that operates within Logic. It has a fixed — as opposed to modular — architecture. While the audio connections are unalterable, modulations are freely configurable using the modulation matrix in the middle of the window (with the dark blue background). Here you can see how we assigned Filter 2's cutoff frequency to be modulated by Envelope 2. The fader to the right of each matrix unit is an amplitude scaler for the modulator signal.

The one audio routing option involves whether the filters operate in series or parallel. In our patch, we have put them in parallel mode and set the *Blend* parameter all the way down so that we only hear Filter 2.

All three envelope generators allow you to set a range for the attack time. In this case, each note's velocity directly determines the attack time for the voice being triggered within the set range.

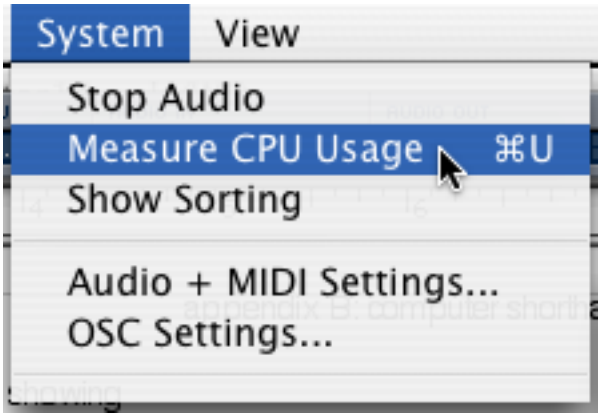
```
sr        =     44100
kr        =     2205
ksmps     =     20
nchnls    =     2

     instr 05
ilength   =         p3              ; note length in seconds
ifreq     =         p4              ; pitch (Hz)
iamp      =         ampdB(p5)       ; amplitude (dB)
iampA     =         p6              ; amp env attack time
iampR     =         p7              ; amp env release time
ifiltA    =         p8              ; filt env attack time
ifiltR    =         p9              ; filt env release time
imult     =         p10             ; filt env amplitude
icutoff   =         p11             ; filt cutoff frequency
ires      =         p12             ; filt resonance
kAenv     linen     iamp, iampA, ilength, iampR
kFenv     linen     imult, ifiltA, ilength*0.8, ifiltR
asaw      oscil     kAenv, ifreq, 1
aout      lowres    asaw, icutoff+kFenv, ires
          outs      aout, aout
     endin
```

Csound is actually a programming language (one of the so-called Music V languages) designed for composition as opposed to realtime performance. Unlike our other examples, it is not one specific application. Also worth noting is that Csound uses an amplitude driven model like Reaktor. This above Csound orchestra file emulates our ensemble patch as instrument 05. (A companion score file would contain the note list to be played and other performance data.) The i-rate variables (*ilength*, *ifiltR*, etc.) are the equivalent of our knobs in Reaktor. Since these variables are assigned to parameters (*p3-p12*), their value can be changed with each note event.

Moog's **Micromoog**

Micromoog is actually a hardware synthesizer. It also has a fixed architecture but fits our requirements well with its one oscillator, one low pass filter, and two envelopes (amplitude and filter). The main differences are that the envelopes here are called *contour generators* and that they are based on an attack-release model.

-- - --- - --



Computer menus usually mention a function's key command when one is available. The following list explains some of the more cryptic symbols used for keyboard keys.

⌘	command (apple)	⇥	tab
⌥	option	↩	return
⇧	shift	⌅	enter
^	control	⌫	delete ("backspace")
⇪	caps lock	⎋	escape
←, ↑, →, ↓	arrow keys		

Another common source of confusion is moving between Mac and PC platforms. Here is a list of the most common key equivalents.

_MAC	**_PC**
Option key	Alt key
Return key	Enter key
Command (Apple) key	Control key
Control key	Windows ("Start") key
Delete key	Backspace key
Control - Mouse click	Right Mouse click

-- - --- - --