Event Tables

A tutorial focusing on the basic uses of the event table

by Theologiae
There has been a request for tutorials on the event table for at least 6 years in the Reaktor forum. I am in no way a master of Reaktor, but having a hard time understanding event tables myself, I feel that it is time for someone to write one.

The pace of this tutorial will be slow. I want to make this as comprehensive as possible for users that have started learning but are looking for the missing link from beginner to builder. I want to cover as much information as I can so that newcomers can take the most from this. For people that have a few months of building in Reaktor under their belt, this tutorial may seem simple or obvious at times. I want this to be accessible, but as a standard I will take for granted that one using this text has gone through the “First Steps in Reaktor” tutorial found on pages 61-89 in the Reaktor 5 printed manual (and the same page in the pdf that is available in the documentation folder in your Reaktor applications file). It would also be a good idea to have finished the other tutorials at the NI site (http://www.nativeinstruments.de/index.php?id=reaktortut_us)

You should also have some good standing with electronic music. If you don’t know what an LFO is, you might not get the most out of this tutorial. I have been doing this for about 2 years and I learn more everyday. My advice is to read everything you can. SoundOnSound (http://www.soundonsound.com) has a great group of papers about the basics (terminology, math, tools, etc.) and I have gone back to them many times to learn new things. I also suggest searching the Reaktor user forum. When I begin working with something new, be it a module or an instrument, I do a search for it by name and read a few dozen posts. Most times I find a solution to my problem and/or answer to my question. And if all else fails, the user forum has a lot of great people there that really go out of their way to help others.

I am, and this tutorial is being written, with no affiliation with Native Instruments, so there is a possibility of errors. If you want to point a finger at someone, it would be me. As Reaktor is updated, portions of this tutorial may fall out of date. So if you are on Reaktor 8 and you find a lot of problems with the text, I’m sorry.

It’s confession time. I have a Mac. The key commands and the look of the windows may be different for a PC. The main thing to keep in mind is when I say ctrl click, it means right click for a PC. I have noticed from other screen shots that the windows are a little different. So if you get a little lost while working, you might need to get the manual and look up whatever might look different.

Oh, and when I’m referencing another builder, their work or something they have said, I will use their forum names. I have learned a few of the real names of the many helpful members of the forum, but since I really don’t “know” any of them, I will use their screen name.

And lastly, why did I spend the time to do this? I did it because there are a lot of people that have helped me in the past. I owe it to everyone else to work in the same spirit. The more builders Reaktor has, the more creativity there will be in the community and the better tools we will all have to make music.
On page 240 in the Reaktor manual, there is a good explanation of what can be done with an event table: “The Table modules allow for very flexible handling of events and audio data [audio cannot be done with the event table, for audio use the audio table which will not be covered here]. The Table modules can be used to design oscillators, LFOs or wave shapers by drawing your own waveforms with the mouse. Or you can cross fade between wave tables and create envelopes with curve shapes drawn by hand or with countless breakpoints. The Event Table can be used as a sequencer for outputting gate and pitch values or for controlling any parameter in a Reaktor Ensemble.”

So what is it? And more importantly, how do you work with it. You can think of an Event Tables as a grid of data. Look at the tables icon, what a Table does is hold a two dimensional group of information. That information can be imported into a Table or drawn onto it and read back. But to be honest, the very look of the event table is hard to chew on. There is a lot of control here, and understanding how to use it isn’t exactly easy or intuitive for people without a history of programing. For this project, we will be building a simple drum machine. I will talk about all the ports on a Event Table, but I won’t cover how to use them all within the next few parts. We will limit our use to the poets we need and the others will be covered in future parts of this tutorials.

Step 1. Open Reaktor with a new ensemble and place a “new2in2out” Instrument in the structure window. Hook the instrument up to the audio output of your audio card (fig. 2). I make a habit of hooking this up every time I start building. I’ve had a few moments of frustration when I hooked modules together and not gotten the right results, only to find I hadn’t hooked the Instrument up in the first place. Double click on the Instrument to get a view of the all too familiar home of your musical children. Go ahead and place in the Event Table (Built-in module⇒Event Processing⇒Event Table). (fig. 3)
A little building tip! We won’t be hooking up the Event Table to the audio outs here, but we need the Table to be active (looking at the little light on the bottom right, you’ll notice that it isn’t lit like the two Audio Voice Combiners). A little trick is to hook up the Event Table to a Lamp. The Lamp will make the Table active without having it hooked to an output.

Place a Lamp in the structure (Built-in module⇒Panel⇒Lamp) and draw a wire (fig. 4). Wait, it didn’t do anything, right? You need to open the Properties window (I double click on the Lamp to do this fast) and select “Always Active”. But now we have another problem, there’s a little red X over the input of the Lamp (fig 5).

This little red X lets you know that you have a polyphonic module going into a monophonic one. The orange light coming out the Lamp tells us that it is mono. The yellow one on the Table is poly. The use of the words “mono” and “poly” can get confusing in Reaktor, but I will try and clear it up through our building example.

The Lamp won’t be telling us anything we need to know right now. There is a function for the Lamp, but for what we doing, it’s not important. To clear up space in the Panel window, go into the Lamps Properties Appearance page and un-check the Visible so the Lamp disappears.
To get both the Table and the Lamp to work together, we can do one of two things. The first is to change the Table to mono. That can be done by ctrl clicking on the Table and selecting mono (fig. 6). The other way, and what we are going to do, is change the number of voices in the Instrument to 1 (fig. 7 and 8). You can think of voices in an Instrument as the number of calculations in a module. Because a Lamp is designed to signal just one event, it’s mono by default.

Go into the Instrument’s Properties and halfway down you will see the value of the Voices. Change the number you see (the default is set to 4) to 1 voice.

A fast way to get the Instrument’s Properties is to single click on the background with the Properties window already open.

I have found no clear rule of thumb to explain Voices. Every Ensemble can call for different uses and settings for voice information. To get an idea of the importance of voice allocation, open up any work by Lazy Fish and try to get it to work after changing the Voices around. The issue of voices is a bit too much to cover here. (For more information see page 141 and 144 of the manual.)

Just keep in mind that a poly module can’t go into a mono module. You get that little red X. Mono modules can go into poly ones though. And that make intuitive sense. Think about the idea of processing different calculations for different voices. In a multi-voiced instrument you can Tool Tips over a wire and see how many voice are running through it (fig. 9 and 10). This poly module has 4 voices, with 4 different calculations. So the wire from the module must be received by another module that can handle those voices. A mono module cannot. This is the reason for Audio Voice Combiners (fig. 11), which combine all the voices in an instrument into one mono voice. Because each output into your sound card is mono, you have to combine all the voices to have them pass through. By setting the voices to 1, there will never be more then 1 voice in the instrument. With this setting, poly modules (yellow) can run into mono ones (orange). This will speed up the building here, but I don’t want to give the wrong idea. This is only the case when you have the voices set to 1. After hooking everything together, if you want a better understanding, change the voice back to 4 and see what happens.
Now that we have our Table active with a Lamp, we are ready to take a closer look. I will first talk about the module itself and then move into the Properties.

First, make sure that your Tool Tips are on. With this on, you will get short hints about parts of the module by placing your cursor over each port or wire (fig. 12 and 13). This is a basic tool which is helpful when learning to build. I keep it on all the time to check the values of wires in a structure as I work. There are limitations to using the Tool Tips though. It doesn’t tell how many events are passing by, or the order of the events in real time (we will deal with this in a little bit). The other useful side of this function is the notations that other builders can make in their macros. As you study more structure you will run across this and it’s a great help. I make a point to note my macros to help others use my work.

I want to go over what the manual has written about each of the ports.

Starting at the top left of the table the manual says:

1. RX audio/event input “the X-position of the table from which the data is read”
2. RY audio/event input “the Y-position of a table cell from which the data is read. This is used in 2D-mode or for addressing the row number if more than one row exists.”
3. R event input “for triggering table read operation at the position given by RX and RY. Each event here produces an event at the Out output.”
4. WX audio/event input “for the X-position of a table cell in which the data is written.”
5. WY audio/event input “for the Y-position of a table cell in which the data is written.”
6. In event input “for writing into the table at the position given by WX and WY. The value of the data written into the table cell is the value of the event arriving at In.”
7. XO event input “for the horizontal offset of the displayed data region. XO controls the data position that appears in the display (according to View Alignment). The value of XO is in the units specified in properties.
8. XR event input “for the horizontal range of the displayed data region. XR controls how many units of data fit in the display, i.e. it lets you zoom into the data.
9. YO event input “for the vertical offset of the displayed data region. YO controls the data position that appears in the display (according to View Alignment). The value of YO is in the units specified in properties.”
10. YR event input “for the vertical range of the displayed data region in 2D-mode. YR controls how many units of data fit in the display, i.e. it lets you zoom into the data.”
11. Out event output “for data from the cell controlled by the RX, RY and R inputs.”
12. DX event output “for the size of the horizontal table rows in units.”
13. DY event output “for the size of the vertical table columns in units.”

Now, if you are like me, reading this doesn’t clear up how to use the Table one bit. The ideas are easy to wrap your head around. The rest of this tutorial will focus on clearing up this confusion.
Before we start working, it’s a good idea to set up your screen so that you can see what is going on as you work. Here’s a clip of how I’ll have it (fig. 14). From here you can turn on and off Reaktor in the toolbox, see the CPU load, see the structure window, see the influence on the panel, the properties you have to work with, and a shameless plug for apple and itune’s...you will buy apple, oh yes, buy apple...it’s so much better in here, come to the dark.....um....sorry.....

To start, we need to set the size of the Table. That is done in the Function page of the Table’s Properties (fig. 16). The size of the table will dictate how many rows you will have for saving data and must be set before any data can be placed in the table.

1. Click the Set button and this window will appear (fig 15). For our demo we will set the X to 32 and the Y to 32. After entering the values, click Apply. The X value determines how many rows are in each table position (the right and left of the table). The Y value describes how many different positions you can have (the up and down of the table).

There are a few different ways to place data into a table. For this part of the tutorial we will only use one. We will manually draw into the table with our curser but there are a few more settings to make before we can do this.
2. In the Panel view, ctrl click on the Table and select “draw mode” from the menu (fig. 17). This will set the Table to receive cursor events.

3. Go into the Panel view and draw (fig. 20). You might have done this before in other ensembles. If not, take a second to have a little fun, for the rest of you, go get a drink, something stiff.

To understand the Table better, go back into the Properties window and change the value of the X size to something like 10. Click Apply. What use to be your stepped mountain range of values are now just 10 steps. This is the best way to understand both what the X value is doing in a Table and what the Table size setting does. To have a Table work, you have to set the size. The values you use will depend heavily on what you are using it for. After this demo, you will have a better idea where to go with it. Make sure to reset the X size to 0.

4. In the structure window, create a Knob (Built-in Module⇒Panel⇒Knob) and attach it to the RY and YO of the Table (fig. 18).

5. In the properties of the Knob, set the max. value to 32, min. to 0 and the num. steps to 32. Change the label to “Seq.” (fig 19)

6. Make sure you have something drawn into the table. Now move the Knob (fig. 21 and 22) but remember the value the Knob started on. Blank screen right. Now draw on the table and shift back to the old value. There’s the old drawing.

The Knob you connected to the Table went into the RY and YO inputs. The RY input tells the Table what row you want data read from (which is another way to say the data that will be sent out of the Table), and the YO input tells the Table the offset of the displayed data. These inputs are linked here so the Table will be able to read from the same data you are seeing. The data that you wrote into the other row of the Table is still there. Go ahead and move the Knob back to the other row you were on. Nothing has changed because it wasn’t being read or shown. You could think about it as being in storage. (or you could think of it as a bunny, either way....)
1. Under Properties (fig. 23), change the Interpolation from none to X. The steps have now turned to a smooth line.

That is what Interpolation does (fig. 24), it smooths out the values so that the steps can be read as a consistently changing values. The importance of this will become clear later in the tutorial, but for now think of Interpolation as a change in the values being read. With no Interpolation, the values might be read back as 2,6,3,7,etc., with it the values will read like 2, 2.1,2.2,2.3,etc... so you end up with......you guessed it.....a smoother reading.

2. Change the interpolation back to “None”.

The Table value determines the scale of the value used.

1. Under the Value section of the Properties (fig 25), change the max value to 127. Your drawing looks gone, but it has only been scaled down.

2. In the Panel window, change one of the bars in the Event Table and take note of the top listing under the “Event Table” heading. The value will read to a scale of 0-127.

3. Now change it to max. 1, min. 0 and 100 steps.

There are a few more things in the Function page that I am skipping over for right now. I will get into those parameters as we begin building, so they can be clearly understood within their context.

There isn’t a whole lot that is critical to know about the Appearance of the Table. Though it does set up the GUI for better interaction, there are only two things we need to bother with right now.

The issue of size is straightforward. For this demo, set the X size to 300 and the Y to 100. The Graph sets the way the Table information is shown. Change this around to get an idea of what it’s doing, then set it to line and fill.

The grid is best understood by setting it. Click on the “Enable Grid” boxes X and Value. You will see a grid pop up. Change the values like shown and see how the grid changes.

The Grid gives an understanding of each cell in the table. It will come in handy later. By breaking down the cells in the X grid, you will be able to clearly place data in time with the beat and clock.
Part 3, Building

With this tutorial there came an ensemble, we are now going to build that step by step. It would be better to build this guy as we go, one of the best methods for learning Reaktor is to just get lost inside it and try and find your way out. Along the way, I’m going to take asides to point out how the modules in the structure work. If you know what a “Song Position” module does, just skip by those sections. Once again, I will be taking the long road so that more ground can be covered.

The best thing for a Builder, the Event Watcher

In the beginning, Reaktor was a pandora’s box, that seemed to let out the chaos of music only when others opened it and built instruments. The structure window was distant and cold, like the surface of the moon. Then, on a fateful day, cList created the Event Watcher (fig. 29) and a sun of clarity fell upon the world.

This tool will show you the order and real time values of events in a structure (fig. 30), (the Tool Tips do not) which might not sound like such a big deal, but for reverse engineering, it is a must have. You can find all of Chris List’s test equipment here: http://www.native-instruments.com/index.php?id=userlibrary_us&type=0&ulbr=1&plview=detail&patchid=2785. (I don’t know if cList was the first one to build this macro, he may have gotten the idea from elsewhere, but I’m going off of what I know, and he is good about giving credit to those who earned it, so there you go.)

I have saved his Event Watcher to my macro file (fig. 31), it has become such a core of my study that I now use it every time I look into a structure. By doing this, you will have it under the macro list so you don’t have to look around for it every time you want to place it in a structure.

The macro itself is easy to use. Place it in your structure and wire it up to the module you would like to know the output of. cList has made it very easy to understand with his Tool Tips notations, so after placing it in the structure, read it. Also, the Event Watcher is mono, so it doesn’t process poly signals. That won’t be an issue for us now, but might be important for you to note in the future.
What we’re going to build

What we are going to build is a very basic sample trigger (fig. 32). There is a clock that drives the Event Table, which holds the trigger patterns that drive the sampler. This kind of design works well as a basic drum machine.

Building

The first thing we need for this (other than the Table itself) is a clock. There are many different ways to build a clock in Reaktor, and almost all of them will work. We will build the one I like, but by all means it is not the best.

The clock will drive the Table. It’s paced ticks will trigger the Table in a linear way that will make sure event coming out of the Table are doing just what we want them to.

1. Place a Song Position (fig. 33) in the structure (Built-in modules ⇒ MIDI in ⇒ Song Position).

   The Song Position is a clock source, it puts out an Event out every 96th of a bar. Those event can be divide into beats with a Modulo module. We’ll see that later, but for now the important part to understand is that when the master clock is running, the Song Position sends out events.

2. Hook the Song Position to the Event Watcher (fig 34).

   In the Panel window place the Event Watcher somewhere you can see it. If the master clock isn’t running, click it on or press the space bar. You will see the meters scrolling through numbers fast (fig. 35).

3. In the Panel window place the Event Watcher somewhere you can see it. If the master clock isn’t running, click it on or press the space bar. You will see the meters scrolling through numbers fast (fig. 35).

4. Stop the clock and double click on the stop button in the Panel Tool Bar (fig. 36) to reset the Song Position back to 0.
If you don’t see any values moving with the clock running there could be a few things you might need to check. The first is always in the Toolbox, make sure that Reaktor is running. Also, make sure that both the Event Watcher and the Song Position are active, you can see that by the little lights in the bottom right corner. If you still don’t have it going, you need to go through the manual again to see how your system is hooked together. There are many steps to getting Reaktor to run right, don’t be discouraged, it might take a little time to get things running if you are new to this. And it is worth it in the end.

5. Place a Modulo module (fig. 37) in the structure (Built-in modules⇒Math⇒Modulo)

6. Hook the 96 event out of the Song Position to the A input of the Modulo.


8. Create a Constant on the B input of the Modulo with a value of 6. A fast way to do this is by ctrl clicking on the port itself (fig. 38), you can then create a Control or Constant. You should end up with a structure like this (fig. 39).

The Modulo module is a part of a lot of clocks in Reaktor. It does a basic, but very useful division process. The input of A is divided by the input of B. The module separates the integer value and sends that out of the Div. The remainder is sent out the Mod.

9. Go into the panel view and start the clock. After a short time stop the clock. You will notice that the numbers are paced the same but they are building slower. Where you had the value rise with every tick of the Song Position, now the values stay the same for 6 ticks before it rises 1.

10. Change the constant to 12. Start the clock again and stop it shortly after. You will now see that the value has 12 clicks before rising. Delete the Constant.

11. Place a Selector in the structure (Built-in modules⇒Signal path⇒Selector/Scanner.

12. In the Selectors properties, give it a 4 in the “min num port group” so it looks like the picture.

13. Hook them all together like so (fig. 40).

The Selector is going to function like a switch, choosing the different values we will use to change the value rate of the clock. We use a Selector here instead of a normal Switch for a good reason that will be of use to you in the future. When you change a Switch in any
structure, that Switch will fire all the Constant and some other things in a structure, meaning that events might be thrown out of whack. In event processing, there may be time when event firing at the wrong time will mess up the structure. So using a Selector in the structure will keep things in order if you change a value at any time.  

14. Create Constants of the shown values and hook them up like so (fig. 41).

15. Create a List (Built-in module⇒Panel⇒List) and open its Properties. You will find a blank page like shown (fig. 42). By clicking the Append button you can fill in the list (fig. 43). In the List’s appearance page, set the style to menu.

This List will choose what Constant will be sent to the output of the Selector. By looking at the two you can see the correlation of the Constant 24 and the List label 1/4. As we have seen, when a value of 24 is sent to the B input of the Modulo, there are 24 repeats before the value changes. Your next question might then be, “well, they are repeating, but the pace is still the same. So how will that change that pace?” and if that isn’t the question your asking.....sorry, I’m not a mind reader.

What we need is something to only let a values through when there is a numerical change. And aren’t we so lucky, there is a Step Filter.

The Step Filter does just what we need, events “only pass if the input value is larger/smaller then the previous input.”

16. Place a Step Filter in the structure (Built-in Modules⇒Event processing⇒Step Filter). Hook up the structure like so (fig. 44) and attach the Event Watch to see how the events out of the system has changed.
Now the values are rising one after the other (fig 45). From the Panel view, change the List you attached to the Selector and see that now we have the ability to control the pace.

This leaves us with only one more thing to control. We need to control the X value loop, because it will be the values themselves that tell the Event Table how many X values to read before looping back to the beginning.

17. Place another Modulo (Built-in modules⇒Math⇒Modulo) in the structure, attach the Step Filter to the A input of the Modulo like so (fig. 46). Hook the Mod out of the Modulo to the Event Watcher.

18. Create a Knob and hook it to the B input of the Modulo. Give it a range of 0 min, 32 max with 32 steps. Name the knob Steps or the such. This is the Knob we will use to control how many X rows you will have.

The value that you give this Knob should click in your head. We give it a max range of 32, and since the steps Knob is controlling the range that we will travel on the X row, (which will show a value of 31 since the table counts from 0-x and not 1-x) our Knob is scaled to the value of the size of our Table.

You should now have a panel like this (fig 47). With the Event Watcher set up, you will see we now have a loop of values. Now lets hook it all together so the Event Table is in on the action.
Hooking Everything Up

So far, you should have the Seq. Knob hooked up. Now to hook up the everything else.

9. Hook the Mod output of the Modulo to the RX and R inputs of the Event Table.

Remember from the list of Event Table ports above the RX is the input that determines the read position of the X data. And the R input triggers a read operation (that goes for both the X and the Y of the table). That also means that every event at the R input will trigger an event at the output of the Table. This is important, because the Table will only put out values when there is an event at the R input.

20. Hook up the output of the Steps Knob to the XR input of the Table.

The XR sets the range of the X data. This all adds up, because the Modulo will trigger the read operation, while the steps Knob will set the how far that reading will go.

You should now have a structure like this (fig. 49).

Now let's see what all this is doing in the Table itself (fig. 50).

Look at the Panel view and start the clock. You now have a bar that will travel, at the rate you set it to, to the range you set it to. By hooking up the Watcher, you will see what values the drawn pattern is sending out.

fig. 49

fig. 50
We are going to finish the trigger for a drum machine. But before we plug in the sampler, we need to change around the look and function of the Table. For the drum sample to be triggered we only need to send out a 1 or a 0. Although we could control the Pitch and Amp of the Sampler with their own tables, they will be controlled by their own Knobs for now.

1. Go into the Properties of the Table and take off the grid in the Appearance page (fig. 51).

2. Clear both the H scroll bar and the V scroll bar. Then change the Y size to 10 (fig. 52).

3. Make it so that the label and the value are no longer visible (fig. 53).

4. Make sure that “Auto Tit” (fig 54) is on.

5. Now under the Function page of the Properties, change the max to 1, min to 0 and the number of steps and step size to 1 (fig. 55).

6. Go into the panel view and arrange it to your liking. Your Table should now look something like this (fig. 56).
Time to add the Sampler (fig. 57). For this demo I’m using the basic sampler for simplicity. I’ll leave working with the other Samplers to you. Or you. Or that guy over there.

7. Place a Sampler in the structure. In the Samplers Properties, take off the Label and the Value (fig. 58). Change the size X to 300 and the size Y to 20. These are just cosmetic changes, they won’t change the sound, but I like to keep a clean ship.

Now, we need to now load the Sampler. There are two easy ways to open the Sample Map. One is to click on the icon that looks like a waveform in the section called Map (fig. 59). The other is to double-click on the sample window in the Panel view.

But what is a Sample Map? The manual gives a very good break down of what the Sample Map is and how you use it in Reaktor on page 225. Please take time to read it now if you haven’t already. It’s a few pages but you need to know it.

8. In the file that this tutorial came in there is a wav. file, load that file now by opening the Sample Map, open the “edit sample list” and select Add (fig. 60). Look for the file on your system and select it.

The heading you see below can be a cause of confusion. If it’s not for you, lucky. There is a lot to explain, and I’m not going to get to it. Mostly because we won’t need to mess with this for our demo. With the sample selected like this, it is fine for what we need to do. If you have the same set up for your file, just leave it, you’re done. If you don’t, now is the time to change it.
Before we move on, make sure that the loop is off in the Sample Map. You will find it at the bottom, under the waveform display. To know that it’s off, just make sure it’s not lit. If this were to stay on, the sample would just loop over and over after being triggered once. We need it to only trigger when we want it to.

9. Back in the structure window, hook the output of the Table to the Trig of the Sampler (fig. 62). Then hook the out of the Sampler to the Audio Voice Combiner. These guys should still be there from when we created the Instrument. If the Lamp and the Event Watcher are still there, you can delete them.

10. Now create controls for both the P input and the A input of the Sampler (fig. 63). One of the advantages of ctrl-clicking on the port to create the Knob is most times, the Knob will be scaled for you when you are dealing with a module like the Sampler. Checking the properties for the Pitch Knob, you will see that it’s ready to use (fig. 64).

11. Go into the panel view and arrange it to your liking. Something to make it easy to use (fig. 65).

12. Set up your Knobs so that you have a blank sequence, 16 steps, amp of 1 on the Sampler and a Pitch of 60. Rate isn’t important right now.

Set up some values in the Table and hit the Start. Now you have a trigger for this sample. Get a feel for the controls. Change the Pitch or the rate. Cool right? But also very basic. About as basic as it can get. A note before you begin creating beats and really getting into this set up. One of the biggest concerns with using Event Tables are their inability to save Table data with snapshots. To save data, you have to change the Seq knob for every pattern. Look at the ensemble I included to get the feel for this. This part of the Table’s nature will come in handy in the future, but it might seem a pain at times. When we get to part 8 this will clear up.
Part 5, Finishing up

There are a few more things I’d like to wrap up before we move onto the next part. Details and some basic Reaktor stuff. We have some more building to do so you might want to take a cookie break (below).

To poly or not to poly

One of the things I talked about was the use of voices and mono/poly modules. I think it’s worth a few more words because it can cause problems down the road for you. In the structure you should switch all the modules other than the Sampler to mono. I have already shown how to do it, but in case it was too long ago, just ctrl click on the module itself and select mono, the first option (fig. 66).

The reason for this is easy of future use. The clock structure that we built could be used in a lot of structure and there isn’t really a need to have this be polyphonic. If we were to leave the structure as it is right now and you were to place it in another structure that is using multiple voices, your structure could fill up with red X’s. So it is always a good idea to think about and fix your structures at that time you make them. It will save you a lot of time in the future.

The right kind of reset

You may have noticed something while playing around with this structure, and if not you will now. Make sure you have a value in the first row of the table (fig. 67). Now double click on the Stop Button. Did you hear the sample trigger? Now start the clock, that first bar doesn’t trigger the sample again. This is because the 96 position clock is reset when you double click on the stop button. That reset sends a 0 through the structure, and if you follow that 0 with the Event Watcher you will see that it creates an event for that Table data at the first position. What we need is something that resets that clock but doesn’t trigger an event so our drum machine will start from the start.

Lucky, Kid Sputnik made a structure to do just that.
In this structure we have 3 new modules that I have not covered. I’ll discuss each as we build and why they are there.

1. Place a Start/Stop module (Built-in modules ⇒MIDI in⇒Start/Stop) in with the Event Table.

This module you will find in a lot of clocks, it has two event outs. The first, G, sends out a 1 when the clock starts (from clicking your space bar) and a 0 when you stop it. The other out, and the one we’ll use, called Rst, does something a bit different. It sends out a 1 after the clock has been reset and started again. If you haven’t tried that out on the Event Watcher, do so. It will give you a understanding of the idea behind putting this in the system. The idea is to have a reset value be sent only when the clock has been restarted.

2. Place a Separator in the structure. Hook the Rst of the start/stop to the In of the Separator, then the Hi of the Separator to the Event Watcher. Reset the clock like you did while testing the Start/Stop and start it again. Did you see any difference in what it was doing? You shouldn’t have.

This brings us to one of the hardest part about working with Reaktor. Sometime you will be studying a structure and there will be modules that seem like they are doing nothing. I thought this about the Separators at first. But after a few question in the user forum, Kid Sputnik explained this trick to me.

The Separator is a very straight forward module, it takes events at the In and sends them to one of two channels, Hi or Lo, based on the amount at the Thld. So if you have a Compare module that is spitting out 1’s and 0’s, and you only want the 1’s or the 0’s or you need both but you want them going to different places, you put in a Separator with a value of .5 at the Thld and you have what you need.

So what is it doing here with nothing in the Thld? When you have a global reset in you ensemble, events can fire. What the Separator does here is make sure that a 0 will not pass through. Here’s the man in his own words. “this is one of the biggest things you must learn as a Reaktor event programmer. The reason alot of people use a Separator after a button, esp in trigger mode, is that the button sends out a 0 event at startup/init, if it is going to some thing like an iterator [or a value like we have here] (which will trigger with ANY event, even a zero), the Separator ensures that only button-press events will leave the button, and not init values. Also, if the iterator doesn’t have anything attached to it’s Trigger input, it will trigger on startup as well, since an unconnected input is the same as a zero constant, and constants fire at initialization.”

We aren’t doing this because of a global reset, here we are doing this so that resetting the clock that would send any 0 is stopped. And, it’s good measure to always have a fail safe.
3. Place another Separator in the structure and wire the 96 out of the Song Position from the original clock into the In of the second Separator.

4. Place a Value module in (Built-in modules⇒Event processing⇒Value). Wire the Hi of the first Separator into the Trig of the Value. Now the Lo of the second Separator into the audio in of the Value.

The Value module is very common in event processing. And is as straightforward and useful as the Separator. It takes the value in the audio in and sends it out when the Trig receives an event. In this structure we are using the Song Position at the In of the Separator because the Song Position sends out a 0 when you reset the clock. Because the value module will hold it’s last received value, when there is a clock reset, the Lo of the Separator will make sure the Value module will always send out a 0. And if you think back (before I started taking us to the center of the earth with all this stuff) to the problem we were having, it was a 0 we were chasing around right. Since the Start/Stop sends out a 1, we need that to be converted to a 0. That’s what the Value module is great at doing.

5. Place a Merge in (Built-in modules⇒Event processing⇒Merge) You will need to give the Merge two inputs, that is done in the Properties window. Wire the Hi of the second Separator into the top input (the order you wire a Merge doesn’t matter though) Wire the out of the Value module into the other Merge input. Wire the Merge out into the Modulo that use to have the Song Position wired in.

The Merge takes whatever input arrives first at the input and sends it to the output. In this structure, when you start the clock, the 0 that the Value is holding will get to the Merge before the rising values of the Song Position. That makes sure that the 0 we were chasing gets sent only when the clock is started and gets to the rest of the structure first.

For a little more information about this structure you can read the post by Kid Sputnik here. http://www.native-instruments.com/forum_us/showthread.php?t=33212&highlight=song +position

And with that, we have the structure we need. Everything is in place for you to start playing around and trying things out.
The first way to go should be clear. Duplicate what you have just built, and load different samples (bass, snare, etc...) and start to create patterns to make beats. It might be in your best interest to put the table grid back in at this point so you can have a visual idea of where everything is at. What we just built is a classic set up for a drum machine. Dig into the R3 library that you can download from NI’s web site and you will see structures that now make a little more sense. One of the other ways to learn Reaktor is look at these older structures and ask yourself how the build got to where they did. Getting inside the mind of the builder will show different ways of building. It will also clue you onto other things that you can build into the structure that makes it more usable.

And after that, hook up some FX and start to play around. Under the macro section of the user library there is a host of pre-built FX that are ready to go. There are the classic distortions, delays, reverbs and such. Try out the pitch shifter under macros>building blocks>pitch shifter. Sound cool right. As you begin to add more and more to the structure you will start to seed the creativity that leads to building better and better instruments. This is the fundamental way that people have become builders.

So go and make some music already!

Where to go from here
Part 6, Pitch Your Gate

The greatest aspect of Reaktor is the ability for total control over the structure. This is why simple ideas grow into vastly complex tools over time. In this part we will create Event Tables to control the Pitch and the Gate of the Sampler, giving us more sound sculpting ability.

For this part we will mostly be copying the first table, changing a few values and hooking them up to the Sampler. So why make a whole part for this? Well, I want to build with the Stacked Macro (fig. 70) module. This module helps to order the Panel window so it’s easier to work with. While some may say this is cosmetic and not “really” important, I think that the way in which a user interacts with an Instrument greatly influences the sounds they make. I think Metaphysical Functions is a great example of that.

To make building more efficient, we will use the same clock for all the Tables.

1. The first step is to create 1 empty macro.
2. Select everything but the reset clock we built earlier and the Sampler (fig. 71). Cut that out, go into the macro and paste the Table and such into the macro.
3. Create one In and one Out. Rename the In to Clk (clock) and the Out to Trig. Hook the out of the Table to the Out port. Hook the Clk port to the A of the first Modulo.
4. Go outside the macro and duplicate it two times (fig. 72). For the first macro give it the name Trig and Gate, the second name Gate and the last name Pitch. We now have copies of our first Table in all three macros. We name the first “Trig and Gate” because we will be placing the Gate and Trigger together in one view and the Pitch in the other.

Now we need to change around the Tables parameters so they work they way we need them to.

5. Go into the Gate macro and change the Num Steps in the Value section of the Properties to 00 from . Since the Amp of the Sampler has a scale of 0-, we want a range of values to control the amp with. Now for looks, change the Pixel Size of the Y to 40 from 10 (fig. 73).

The Pitch we’re going to need to change around a little it more.
6. Like the Gate Table, change the Pitch Pixel Size of the Y to a higher value, this time 80.
7. For the Values, we need to change the Min to -12 and the Max to 12 with 24 steps. We do this to scale the Pitch so we can get positive pitch values and negative pitch values.

We need to also change the output of the Table. Because Reaktor samples have a root pitch of 60 in the Sample Map, we have to add 60 to the output of the Pitch Table, so the positive and negative Table output values will scale right to the original pitch of the sample.

8. Put an Add in the structure and run the Out of the Table with a Constant of 60 to the Out of the macro (fig. 74).

Now the fruit of out labor. There isn’t a lot to the Stacked Macro. The only thing you need to remember is that to have the Stacked Macro work right, each object in the Stack Macro has to be in its own macro.

8. Place a Stacked Macro (Built-in modules⇒Panel⇒Stacked Macro) in the structure, outside of the macros that we made for the Pitch, Gate and Trig. Place all three in the Stacked Macro by cutting and pasting them.

9. Cut the Gate macro and paste it inside the Trig and Gate. While still inside the Trig and Gate macro, create an Out for the Gate and hook that up. Also, connect the Clk In to the macros.

10. Place a Panel Index (Built-in modules⇒Panel⇒Panel Index) in the structure. This is the module that will tell the Stack Macro which macro to show. Place a In port next to the Panel Index and connect them. In the Panel Index Properties, activate Always Active. Create another In and label it Clk. Create 3 Outs and give them the respective names for the three (fig. 75).

11. Outside the Stacked Macro, place a Button in the structure. Hook this to the In that goes into the Panel Index. Hook the out of the reset clock’s Merge to the Clk of the Stacked Macro.

Right now if you go into the Panel window you will see that the Stacked Macro is cutting off the Tables and such. This is because the size of the Macro is set when you place it. That needs to be changed before it can be useful.

12. In the Properties of the Stacked Macro, change the Size to 400 X pixels and 115 Y pixels.

Now use the button to toggle between that two views. It will take some arranging before you find a setup you like, but you’ll get the feel of it. In the pic below there are some Knobs and Buttons that I haven’t talked about yet, but don’t worry, I’ll get to them in the next part.
Part 7, Randomize

With the structure that we have built, you could develop a really nice drum machine that could generate a host of great beats. But a personal interest of mine is the use of random to mix up the pot and create beats that one could never program by hand. In this part we will build a macro that can randomize the Y position of the table, so as the clock scrolls through the patterns on the X row, the patterns themselves will be changing.

![](image)

Randomizing the Y

1. Open up any of the Table macros that we just built. In the structure, add an empty macro.

2. Add a Randomizer (fig. 77) to the structure (Built-in module⇒Event Processing⇒Randomizer). Create a Control on the Rng or the Randomizer. Give it a range of Max 32, Min 0 and Steps of 32.

The Randomizer is one of the three random modules in the built-in library. The other two, Random Oscillator and Slow Random, are of a very different breed. They have their own benefits, but for event processing, the Randomizer works the best. It has two input, Rng which controls the range of values the random can go and the event In. The In take whatever event is put into it and creates a random output based on that event and limited by the range. As an example, if you were to send a 1 to the In with a Rng of .5 you could expect a random number like 1.3 or 0.8 to come out. This simple scaling of values make the Randomizer useful where the other two aren’t. Because event structures can rely so heavy on what values can be sent or received, the Randomizer’s Rng takes a lot of work out of the process.

One of the problems I had when first started working with the Randomizer was the misunderstanding of the In. I thought that putting a Knob on the In and hooking the Out to, say, the pitch of a sampler, I could get a host of random values to create interesting pitch behavior. But the Randomizer will only randomize the events that are sent to it. One twist of the Knob is only one value sent out. The usefulness of the Randomizer is found when you drive the module with events, from a clock or the such, and wire those events into a input you want to automate.

Much like the clock we built for the Event Table we are going to build a driver for the Randomizer.
2. Add a Sync Clock (Built-in modules⇒MIDI IN⇒Sync Clock).

The Sync Clock (fig. 78) sends out a pulse of 1’s and 0’s, one after the other. The rate of which is controlled in the Properties of the module (fig. 79). Like the Song Position that we used earlier, this module is used to clock events at a controllable rate. For this structure we could use the Song Position, but we can’t use rising clock values, we need constant values. And since the Sync Clock is easier to set and control, we’ll use it here.

3. Add a A to E Trig (Built-in modules⇒Auxiliary⇒A to E Trig). Hook the Out of the Sync Clock to the Trig of the A to E Trig. Create a Constant on the A input and make the value 0 (fig. 80).

The A to E Trig (fig. 81) is apart of the Audio to Event family. They sample from the audio input and output an event based on that sample. This module puts out an event when a positive value hits the Trig.

That value is the input of A.

In this structure we use it as the driver for the Randomizer. The Sync Clock will trigger the A to E Trig with every other pulse sending out a 0 to the Randomizer. Because we have a Knob on the Rng of the Randomizer, we can set the limit to the random values that will travel with a maximum of 32 and -32, the Randomizer will always generate positive and negative numbers.

At this point you might be wondering how we will use it to randomly change the Y of the Event Table. From the Knob we created for the Y earlier we know that whatever values we send should be scaled to integer values only. Right now the Randomizer will send out any number of values but very few of them will be integers. The rest of this structure will be dedicated to controlling those values.

4. Add a Quantize (fig. 82) to the structure (Built-in modules⇒Math⇒Quantize).

Create a Constant on the St input of the Quantize and give it a value of 1.

The Quantize will take the values from the Randomizer and round them to the value at the St. Since that value is 1, all the events from the Randomizer will come out as integers. Time to get the Randomizer in on the action.

5. Hook the E out of the Ato E Trig to the In of the Randomizer and the Out of the Randomizer to the in of the Quantize. You should have a structure like such (fig. 83)

At this point if you haven’t been watching the event system, you should. I haven’t been showing screen shot of it because I will assume that part to save time and space on the page.
There are only a few more steps to take.

6. Add a Rectify (Built-in modules ⇒ Math ⇒ Rectify) to the structure. Wire the Out of the Quantize to the Rectify. The Rectify module (fig. 84) inverts a negative value to a positive value. Watching the events from the structure, the Randomizer is sending out both positive and negative values. Because the Y position of the Table doesn’t go negative, those values would just be ignored by the Table. We could use a Separator here to filter out negative values (with it’s Thrd set to 0, the Separator would stop any negative values from coming through the Hi out), but then those negative values would be lost in that system. With the Rectify, we keep all the values sent by the Randomizer.

7. Add a Merge modules to the set up here. Create a Knob on one of its ports. This Knob will control the Y range if you choose not to use the random system. It is scaled the same as the Y Knob we used earlier (Max 32, Min 0, and 32 Steps) and does the exact same thing. Take the out of the Rectify and hook it up to the other port of the Merge (fig. 85).

The Merge will send out values to the system without preforming any addition to the structure. This highlights the usefulness of the Merge. If we were to use a Add module here (which is might seem like a logical idea), the value that the Knob is left at would be added to the system and the random values would never drop below that value. The user could just make sure to leave the Knob at 0 before playing the Instrument, but as your working, having to double check every Knob can become a waste of time. And forgetting one Knob could through your set up off. It also makes it easier to understand for other users of your Instruments. I may seem to be beating the point dead here, but remember that in building you are making a tool, and the ease of use need to be built into the structure.

One more thing about the Merge. When you place it into a structure, it has only one in and one out. This confused me for a long time, what use does it have with only one input? Well, this is a hold over from an older version of Reaktor. Before R5, the Merge would act like a Step Filter, filtering out any repeating events. But when NI change this aspect of the Merge, they kept the old placing structure. A little bit of history for you.

8. Hook that Merge to an Out (fig. 86).

9. Hook up the macro the same way we hooked up the Knob (fig. 87).

Now give it a try. Start the clock and watch as the Y sequence pattern changes. If you hadn’t created patterned before now go ahead and do as much. Other wise you will see black screen after blank screen pass by. By adjusting the Randomizers range Knob, you will limit the scope of the Y. With the Knob all the way to 0 you will stop the random all together. Also the Seq Knob will work to change the Y value just like it did before we stated all this craziness. We now have a nice little pack of spice to mix in the pot.

Oh but wait, there are something that might be bugging you. I’ll spell it out.
The biggest problem with this random structure as it is, is not having the ability to control the pace of the random from the panel area. The reason we used the Sync Clock was to have that kind of control (available in the modules properties) but it isn’t easy to get to and not clear how to do it. For making beats, the pace of the random is important, so we will now build that.

For the most part we will be copying the structure from the other clocks (fig. 88). We only need the Modulo and the Selector here because we are only controlling the rate, so for this macro we will use the other parts of the clock by creating an In to hook up to the Modulo.

1. Outside the random macro, select the Selector, Modulo and Step Filter modules (including the list and the constants. Ctrl click on the modules and select copy (fig. 89).

11. Go into the random macro and paste the modules there. We will be hooking the modules to the In of the Randomizer, but we need to add on more module before we do. If you think back to page 12, you will remember that the Modulo divides the clock ticks of the Song Position into rising values that change giving the length put out by the Selector. But for this to work in our structure we can’t have rising values like the Song Position puts out. With a Rng of 32, the Randomizer is scaled to the given input, so with the rising clock values, a value of 200 will send out random values between 232 and 168. Which would be useless to us. So we will bring in a Value module with nothing in it’s input so that ever tick of the clock will generate a 0 to the In of the Randomizer, much like the Sync Clock.

12. Connect the Out of the Value to the In of the Randomizer (fig. 90). You now have control over the random pace in the panel.
If you have made it this far, I would guess that you have played around with the Instrument we have built. One thing you might have noticed is the Table data does not save with snapshots. This might have even pissed you off, and if it hasn’t, well, lucky you. I spoke earlier about this aspect of the Table (some would call it a problem, but the random structure we just built shows that not to be true. But to be sure there has been an effort by many to get around this). Because the data does not store with snapshots, the tables can be a bit counter intuitive to new users and might even get in the way of creativity. I’m now going to show one way around this. In the process we will lose the use of the random macro we built, but at the same time we will gain another randomizing ability not easily possible with the past Table structure.

Save a copy of the last Instrument we’ve built so you can work on this one without losing anything. Before we get going, delete the random macro from part 7. It wont function the same under this structure.

The core of this new structure will be the Snap Value Array (fig. 91). This module came along with R5, and will really be our second step into thinking about Reaktor with a R5 mind set (I won’t even get to Core in this tutorial, which is testament to the depth of this program). The Snap Value Array is much like the Event Table without GUI that saves it’s data with snapshots. It will store an array of given values sent to it. The number of arrays is determined in the Properties and the number of elements is only limited by the amount of memory you have (this is not a plug to upgrade, I know you want to, but there is no “need” at this point). So in this context the Array Size is equal to our X Size, 32 (fig. 92). It you change the X size value in the Table, that value has to be reflected in the Array. The Array comes with one Element, but you can add more by clicking the Append button. For this tutorial we will only need one. I will get to some of the other parameters in just a minute.

What we are going to build
By looking at the picture above you will see that we are also going to use another module that we have not used before. The Mouse Area (fig. 93) is also new to Reaktor and is apart of the suite of modules for use in making custom GUI. There is a lot you can do with this guy. Open any of the factory library that came with R5 and you will see that it is a major part of most, if not all the GUI. We are going to use it to write values into the Snap Value Array. Remember when we switch the Table to draw mode on page 8, the Mouse Area is going to basically be the draw area. We will be building onto the Trig Table (but you can do this with any of the three Tables we have, the structure will be the same) , so go and get into that macro.

1. Place a Mouse Area into the structure (Built-in module⇒Panel⇒Mouse Area) and fig. 93 open up it’s Properties. Go into the Panel view and look for it. If you don’t have the Panel unlocked, you might not see it. Unlock the Panel and you will see a grey square that look like a transparency.

The Mouse Area is clear and will transmit mouse information but you have to have some kind of object under it to see what it is doing. In our case we will use the Table to show the values we are writing in, but in many cases the Multi or Poly-Display are used. One of the advantages of using an Event Table in this way is the speed in which you can build. Using the Poly-Display gives you a lot more options, but they are harder to work with if you don’t have a history with it. I’m not trying to scare you away, everyone should get the feel for them, but you will need to give it a few days of work to get where you need to be. For making what we are after, and with the skill you have after all these pages, you can get a Table up and running in no time.

2. Change the size of the Mouse Area in the Properties (fig. 94) to the same size of the Table to match them but add 2 to both the X and Y. I don’t know why the Mouse Area is like this, it doesn’t scale to the same size as everything else. But that’s the breaks......er.........Reaktor. Place the mouse area over the Table and lock the Panel.

3. Set up the X and Y range like shown (fig. 95). The only thing you will have to change is the Num Steps under the Range X. The change is to better the scaling of the Mouse Area. It might not be clear right now, but when you start drawing on the area, you will want a lot of resolution in play so that you can freely draw in the values you want. Run the Out of the X or Y into the Event Watcher and in the Panel, draw on the Table. You wont see any change in the Table but the Watcher will show that things are happening.

4. Place a Merge in the structure and hook the X and the Y of the Mouse Area up to it.

5. Place a Value module in there too. Hook together the Merge to the Trig and the Y of the Mouse Area to the In of the Value (fig. 96). The use of the
Value is apart of the smoothing of drawing resolution. For the Snap Value Array to work the way we want it too, we need its In (you may see the In listed as “W” right now, but you can change that by double clicking on it) to be triggered if we are changing the X or the Y of the Mouse Area. But we are only writing in the Y value. With the Value triggered by both, we get all the right functions without losing our values.

6. Hook the Out of the Value to the In (W) of the Snap Value Array.

7. Place a Multiply in the structure (fig. 97) and hook the X of the Mouse Area to one of the inputs. The other input is going to be connected to the Steps knob we already have in the structure. The knob that connects to the XR in of the Table. This place here is critical. Because we want to be able to adjust the how many X rows on the Table will be played back, we have to scale the X values of the Mouse Area to the Steps of the Table. The multiply performs this function.

8. Place a Add module in the structure (fig. 98). Hook the out of the Multiply to it with a 1 Constant in the other input. We do this because the Snap Value Array is scaled 1-X, and both the Mouse Area and the Table are scaled 0-X. Don’t ask me why, I’m not that far into Reaktor to understand that, but it’s just in it’s nature.

9. Place the Snap Value Array in (Built-In modules ⇒ Auxiliary ⇒ Snap Value Array) Hook it up like shown (fig. 98). You notice the Subtraction module after the Idx output of the Array? Just like the Add in the Idx input, we need to now subtract that amount so it again scales to the Table.

I’ve shown the properties of the Snap Value Array but here is another pic (fig. 99) of how this one will be set up.

The one thing I need to address is the Self-Iteration properties. As you change values in the Mouse area, having the Self-Iteration on will make sure that the Array is updated when you do this. If you save a snap and this isn’t on, the snap will not hold your settings.

For hooking up to the Table, hook up the out of the Array to the In of the Table and the Out of the Subtraction to the WX (fig. 100).
Before you can start rocking this baby, you need to change a few other things about the Table. Since it is now going to work as nothing more then a glorified display, we will make it as such.

10. In the Table’s properties, change the Table Size to X 32 and Y 1. Since we will no longer scroll through the Y range (we will save data with the snap) we only need one for the Y range.

11. In the appearance page of the properties, un-check the X auto Fit. We haven’t covered this before, but this parameter makes the view of the table the size of the X Table size. With is off, the view of the Table will be the steps we set with the Steps Knob. Also, pull the alignment sliders all the way to the right. You might also want to click the grid back on, this does make it easier to see where you are at in the Table.

One more thing I need to mention it the use of a Mouse Area and Macros. For this structure to work, you need to have the Table and the Mouse Area on the same level in the macro build. If the Mouse Area is inside a macro and the Table isn’t, the Mouse Area will always sit behind the table. Instead of writing into the Array like we want to, you will be drawing onto the Table, which will not store any of the written data. So, now you are set.

This is only one way to do this. In Lurker (in the R5 factory library) there is another way of doing this, much like what we built, but with some nice differences that you will certainly learn from. The one thing about Reaktor that will always be true is there in never just one way to do something.

I got this structure from “Dubby Red Planet” by Peter Dines and a discussion in the user forum, built by Weeklyhaircut from an idea by Toxonic. The full discussion is found here: http://www.native-instruments.com/forum_us/showthread.php?t=33116
Part 9, Randomize it!

Randomizing the X values in the Event Table is a bit more tricky than the Y. Where as the other structure we made is driven by a clock and will randomly select patterns, to randomize the X values it is necessary to create a set of new values across all the rows. So we wont be able to send a pulse of random values into the X position, we have to build a structure that will scroll through the X row and write in new values as we go (and by we I mean my cats and myself, since we are the only ones here right now). This structure (fig. 101) is going to wildly different from what we have built and introduce two other module, the Iteration and the Order.

1. Create a new macro. We will use this structure again, so it’s a good idea to keep it in order.

2. Place a Button (Built-in Module⇒Panel⇒Button) inside the macro. The operation that this structure performs scrolls through the X row and place in new values, so we need to start with something to trigger the thing. A Button works great for this. With R5 there are a lot of really cool buttons you can use (I think Herw’s Mini Modular has really nice ones, as well as LazyFish’s stuff) but for right now just use the stock Button.

3. Use a Separator after the Button with nothing in the Thld like before to make sure that only a positive value will trigger the structure. Place a Value after that to make that triggering value a 0 since the separator will only let through 1 and higher. That leads up to the Iteration.

   Place one in there (Build-in Modules⇒Event Processing⇒Iteration) The Iteration (fig. 102) will send out a series of values when there is an event at the event In. Any event will trigger the Iteration, positive-negitive-whatever. That is why the Separator is so important in this structure. (In the future, if you are de-bugging a Instrument and there are Iterations in the structure, not having this set up could be the problem. This is true for the buggy nature of Vectory, as found by Kid Sputnik.) The number of values the Iteration sends out is determined by the N input. The increment that the Values are determined by the Inc input. With a value of 1, each event will rise by 1 i.e. 1, 2, 3, 4, etc. If it was 0.5 it would rise 0.5, 1, 1.5, 2, 2.5 etc. This module can be used for a lot of things but in this structure it will simply scroll through the X row and place the new events in.
For a better understanding of the Iteration module, make a step up like the pictured (fig. 103, next page) and watch what the Event Watcher does when it is triggered. You will see 32 events roll by. This simple example show the nature of the Iteration module.

4. Hook the Out of the Value to the In of the Iteration. Give the Inc a value of 1 but don’t worry about the N input and the Subtraction for right now.

5. Place an Order Module in (fig. 104). To be honest with you, I get a little head ache when I see an Order module in a structure. I’m sure that over time the feeling will pass, but right now they still make me cringe a little. The module is very straightforward. The event at the input is sent to the 1 output and travels all the way through all the other modules in the chain before the exact same event is sent out of the 2 output (the same for the 3 output). I get a little twitch from them because you really have to think about event processing when you see them. You need to understand what all the modules in the chain are doing, and how they are working in that particular structure. That can be a bit of a challenge when you are just learning. But luckily for us, in this structure the Order module is doing simple things.

6. Connect the 1 output of the Order (fig. 105) to the WX of the Table. That is the X write position. This will trigger the X row at each step, starting at the beginning and running all the way to the end. But at each step of the write position, the 2 output of the Order module will trigger the Value and that will trigger the Randomize which is going into the In of the Table. And because the Order will only send the next event out the 1’s output when the 2nd is finished, you have a tit for tat process. The WX will trigger with the step value and the random value at the In will set that rows value.

We are going to make this structure for each of the three Tables we have going right now, so the Rng of the Randomize will be different for each.

7. For the Trig Table, set it to the shown value, 0.5.

8. This next set of modules (fig. 106) will be in the Trig Table but not the other two. With the Trig Table we need only two values, 1’s for a trigger and 0’s for a pause. The structure without this set would send out all values of events in the given Rng. So to limit it to only the two values we want, we will separate the positive from the negative (oh, how progressive of us) and have the positive values trigger a 1 and a 0 for the negative. The two events are then merged (keeping in mind that the merge just sends the first event to the output, and since the events are ordered, all events get sent from the Randomize) and sent to the In of the Table.
9. The last part of the structure to describe is the DX input and the subtraction going into the N input of the Iteration (fig. 90). The DX is a part of the Event Table that I haven’t talked about yet. It’s not too complex, it puts out the event value “for the size of the horizontal table rows in units” which in this case is 32. The DX doesn’t put out a stream of events (you can check with the Event Watcher) so you might ask yourself, why not just put a constant of 32 at the input of the Iteration? Well, this comes back to the idea of putting this structure in a macro. I set this up so that this structure could be used for all three of the Tables in the Instrument, and while they all have the same X size (and it is very unlikely that will changed while working with the Instrument) you might drop this into another structure in the future where the DX might be different. This way you can work faster by just hooking up the DX to the in and forgetting about it. It also get to the mind set of creating macros that you will use again and again. Instead of reinventing the wheel every time your building, you can rely on building blocks to get you where you want to go.

10. Hooking this bad boy up to the Table is going to need some Merges. Because the Snap Value Array is going into the In and the WX of the Table, we merge the two so that both can pass through when we need them too.

So go and try it. Randomize the hell out of this bad boy.

11. Now, for the other two Tables, we only have to change a few things. First, copy the macros and label them. Cut them out of the Trig macro and paste them into the Pitch and Gate. For the Gate, you can erase everything coming off the Randomizer. Since we will need a dynamic randomizing, we just have to make all values positive (a Rectify) and send it out. We need random scale of 1, so we have a Constant of 1 going into the Randomizers Rng. For the Pitch Table, I put a 5 for the Randomizers Rng and skipped the Rectify all together. Our Pitch Table goes both positive and negative. The 5 isn’t the full range of the Table, but I find I have a better sound when randomizing in lower amounts and drawing in the more extreme values. But that’s just me.

So, with a little sorting in the Panel window, you have a whole new set up to play with and a new way of working with it. This structure was designed by Kid Sputnik and cList per a question I had in the user forum. You can find that here: http://www.nativeinstruments.de/forum_us/showthread.php?t=34926