# Some thoughts on Elektron' s Octatrack

Merlin

April 6, 2013

# Contents

# 1 Introduction

This article provides an analysis of the Elektron Octatrack from a users point of view. The aim is to look at the ot from the ground up, to explain how things work and to develop an easy to understand overview of the ot' s structure. Further on, several scenarios will be described in which I show/suggest how to approach the ot for common tasks.

As we will see, understanding just a couple of key concepts greatly reduces the fog and confusion which seems to haunt many owners of the ot.

A couple of notes:

- This article has been written in a cumulative style. If you read something on page x, I implicitly assume that everything I covered before that is known. You have been warned.
- Some information presented in this article will be exprienced as "obvious" or "nothing new". This is inevitable.
- Writing this article is not exactly hard: The ot is what it currently is and it is fairly easy to write a story around things that already exist.
- I do not work for Elektron and have no insight in their technology. This article is written from a user' s point of view. Anyone on the Elektron r&d team who reads this is probably in for a good laugh;)
- This document is an ongoing effort. Feel free to suggest and correct.
- Why did I write this document? Well... Having obtained the ot a few months ago, I experienced a vast overkill of it' s possibilities. Merging all these posibilities into a productive workflow was difficult. I therefore decided to write down my thoughts as this forced me to express them in an understandable way. By doing so, this document grew and grew as time passed by while my abilities of integrating the ot' s functionality into a good workflow grew with it. I therefore have to admit that I wrote this document for selfish reasons. Adapting it to a format that can be given away to the elektron users forum, was little extra work though... So if this helps people in unleashing the ot, I don' t see any reasons for not sharing it...

# 2 Samples

Before diving in the ot itself, it' s wise to spend a couple of words on samples first:
Basically, we can distinguish between two types of samples:

- short samples (single cycle waveforms, single shot samples and reasonbly small loops)
- long samples that may take many minutes

Although the above distinction is obvious, it is important to mention since it has consequences for the way in which the ot handles them.

## 2.1 Streaming

Ideally, the ot should be able to handle all kinds of samples, both small and long, but... If you allow people to use extremely long samples, you have to equip the ot with an enormous amount of memory. There is a solution for this: streaming long files directly from the cf-card. Although streaming is a reliable technique, it has it' s disadvantages. The ot has the ability of playing 8 samples simultaneously. So if you load 8 static machines to a pattern and let them play at the same time, you are effectively sucking 8 streams of cd-quality audio from the cf-card. This puts a heavy burden on the cf-card and by doing so, you can run into timing problems when the card is cheap and not fast enough. Therefore, although streaming can be used, my personal attitude is that I avoid it as much as possible, just to be on the safe side.

On the other hand: Elektron offers streaming as a valid technique, so if you use it, it should work as advertised, provided that your cf-card can handle the speed.

## 2.2 Ram memory

An alternative to streaming is this: Equip the ot with a fair amount of internal memory. Once the ot boots up, a project is automatically loaded and a small collection of samples is read *once* from the cf-card and put in ram memory.

By doing so several advantages are obtained: the samples are read once so no further streaming is needed. Since the samples are loaded in ram memory, the further handling of those samples is independent of the cf-card and demands for speed and processing can be increased: Reading from and writing to ram is simply faster than streaming from a cf-card.

The disadvantage is of course that the amount of ram memory ultimately limits the amount of audio that can be made available to the rest of the system. Elektron equipped the ot with 80Mb of memory, which corresponds roughly to 8 minutes of cd-quality audio. 80Mb. . . hmm. . . doesn' t seem much! Conclusion: whatever drives Elektron to use only 80Mb, the emphasis is clearly not on *quantity* (that is: layering and excessive polyphony), but on *quality* and doing some serious stuff with a minimum of samples and memory. . .

# 3 The sample slot lists

In the previous section we saw that Elektron expects you to select a relatively small amount of samples and squeeze the most out of those.

Selecting a small amount of samples naturaly leads to the need for some kind of list in which we put the samples we want to use. Elektron provides two lists for doing so: The flex list and the static list. Since both work the same way I will only describe the flex list, but keep in mind that the static list is used for long samples which are streamed, while the flex list is used for shorter samples which are put in ram memory.

The flex list can be filled with up to 128 samples. When filling the flex list with samples, we therefore face two conditions:

- We cannot use more than 128 samples in our project.
- Whether we use 128 samples or less: the total sum of those samples should not exceed 80Mb.

## 3.1 Sample editing

Selecting samples from the cf-card and putting them in a list is all fine, but we want more than that. We want very precise control over the way the sample is played. We want to "tailor" our samples so they behave the way we want. Thus, several features should be offered:

- defining start and end points of the sample
- defining loop points of the sample
- defining whether the sample should loop and *how* it should loop: repetitive start to end looping or ping-pong-ing from start to end and back, etc.
- slicing: the ability of cutting up a sample in several slices, allowing for ridiculous ways of playing back those inidividual slices.

All the above operations are carried out in the sample editing menu. Once you sliced a sample or defined loop points of whatever, the decisions you made are stored *in the sample slot list*. This is an important concept: the above list of operations is stored with the sample in the sample slot list and has nothing to do with the flex machines that eventually play the sample. This means that you can add a sample to the sample slot list, slice it, trim it, etc. without assigning it to a flex machine. You could even do it without using the sample anywhere in your project. This would obviously be pointless, but it clearly illustrates that the sample slot list lives on it' s own and exists throughout the whole project, whether flex machines use anything from that list or not.

The most important train of thought is therefore easy: you select samples from the cf-card and put them in a list. Once a sample is in the list, you define the basic properties of that sample and thereby determine how the rest of the system should handle it.

## 3.2 Advantages of the sample slot list

Putting samples in a slot list and defining some basic ways the rest of the system should handle the sample has several advantages:

**Speed** You prepare a sample by defining and slicing it the way you want and once you have done that, the sample and it' s settings are available in any track holding a flex machine in any pattern in any bank throughout the project. Basic train of thought: you do the hard work once and re-use it wherever you need it.

**Flexibility** The same sample can be put multiple times in the list. This allows for handling the same sample in completely different ways. For example: put some kind of loop in slot1 and slice it to 16 slices. Now select the same sample and put it in slot2. You can now slice it up to 64 slices, choose different start and end points, etc. Load the same sample in slot 3 and you can again make different choices. You have now defined 3 different ways of handling the same sample.

**Adaptivity** Since creating music is a highly emotional and artistic process, the sample slot list does not need to be filled straight from the start: as your project progresses, more and more samples can be injected into the list and unused samples can be removed as time passes. Just pay attention to the conditions described earlier: no more than 128 samples, eating no more than 80Mb of memory.

# 4 The flex machine

So far we have seen that samples are drawn from the cf-card and stashed into a list. Each of these samples can be prepared for further use by using the operations in the sample editing menu.

For playing samples from the flex list, the flex machine is needed. Once you place a flex machine on a track, the ot asks you to select a sample from the list. At this point you can select an already exisiting sample or you can select a free sample slot after which the ot allows you to select a sample from the cf-card. The selected sample is then added to the list and assigned to the flex machine.

An important thing to understand here is that Elektron provides you a user interface in which adding samples to the sample list and assigning samples to a flex machine go hand in hand. Keep in mind, however, that these are two seperate procedures.

Now that a sample is assigned to a flex machine, you can trigger it by programming triggers on the sequencer of that track or by live recording those triggers.

## 4.1 The flex machine and slices

Assigning a sample to a flex machine seems easy and straightfroward, but there are a couple of details to keep in mind: The samples that live in the sample list can be prepared as described earlier. This means that some samples will be sliced, while others will not. This has consequences for the way the flex machines handle them.

If a sample is sliced, you must instruct the flex machine to use those slices. Dive in the playback setup menu of the flex machine and set the slice option to "on". With this setting the start parameter on the playback page can now be used to select which slice should be played.

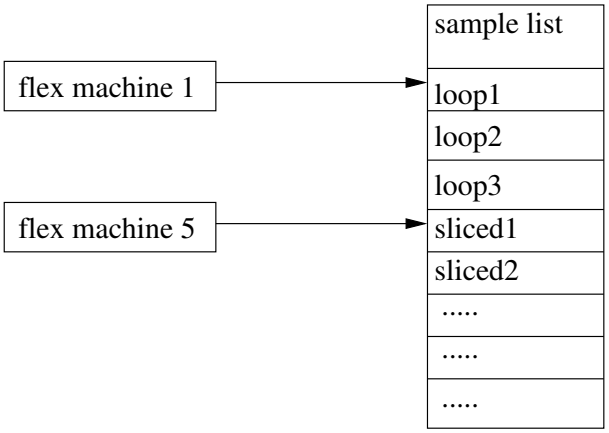
Bottom line: since samples can be prepared in different ways, the flex machines have to be adapted/tailored to make use of what was prepared. The features you have in the sample edit menu form the basis on which flex machines operate.

## 4.2 The flex machine and sample locks

Before reading further: I assume that you understand the concept of parameter locks. Parameter locks form a very powerful feature when used on the flex machines. To explain their power, I will turn to a simple example. Assume:

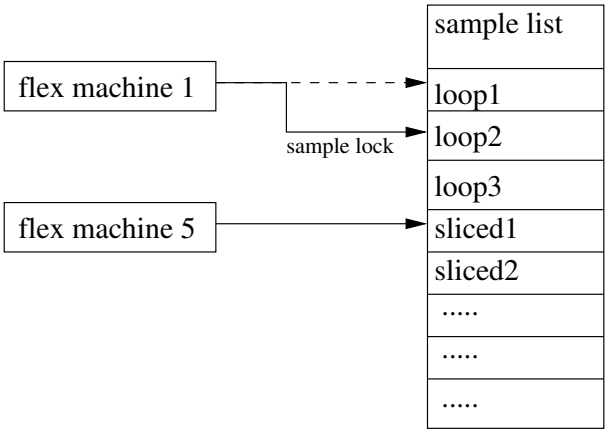
- The sample list contains three loop samples, simply called loop1, loop2 and loop3.
- The sample list also contains two loop samples which have been sliced. Let's call them sliced1 and sliced2.
- A flex machine lives on track one and has been assigned to loop1.
- Another flex machine lives on track 5 and was assigned to sliced2.

Schematically speaking, the pattern was setup like this:




This is a pretty straightforward setup. Let' s look at flex machine 1...

Flex machine 1 plays a simple loop. Therefore the sequencer only needs one trigger to play it and that trigger is placed on the first step. Nothing wrong here, but suppose I like to play a different sample. You can now hold the trigger and turn the level knob. The ot presents you the sample slot list and you can select any of the samples in that list. Suppose I select loop2. The result is that, although loop1 was (and is) assigned to the flex machine, I have overridden this by using a sample lock on that step. Schematically speaking:

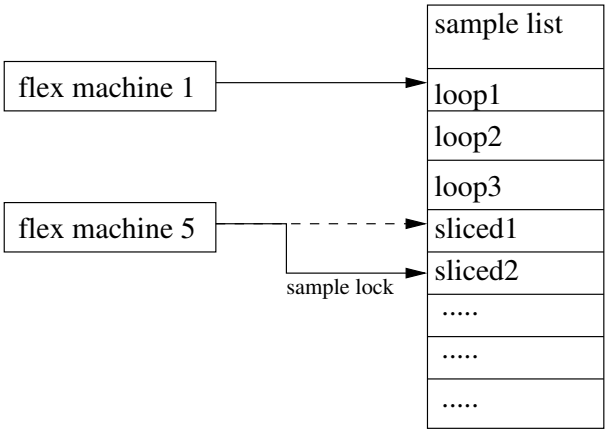



And there it is: By using sample locks, I can play *any* sample from the sample list at *any* step on *any* track that contains a flex machine in *any* pattern in *any* bank. That is a pretty powerful way of dealing with samples. This approach shows how different the ot works when compared to other samplers: Instead of coming up with excessive layering and polyphony, Elektron put the power in the *accessibility* of samples.

### 4.3 Slice locks

Again look at the example as used in the previous section. Flex machine 5 has been assigned to a sliced sample. For using the slices, the flex machine was setup as described earlier. In the previous section we have seen that by using sample locks, you can select whatever sample is in your sample list. So far so good, but just like the sample can be locked, so can the start parameter on the playback page. The start parameter determines which slice is played. Thus, you can not only determine which sample you want to play, but also which slice *in that specific sample* by locking the start parameter.

As an example, assume that we lock flex machine 5 to sliced2. Schematically:




Since the sliced2 sample also contains slices, I can now lock the start parameter to select which slice I need.

Bottom line: *Any* flex machine configured for dealing with slices can play *any* slice of *any* sample that contains slices. You select the sample by using sample locks and you select the slice by locking the start parameter on the playback page.

Let' s put this to the extreme and do a little math: The sample slot list can contain up to 128 samples. Each of those samples can contain 64 slices. We therefore have $128 \times 64 = 8192$ slices available.

Add to this that that we can do the same thing with the static sample list and static machines and this number doubles to a massive 16384 available slices.

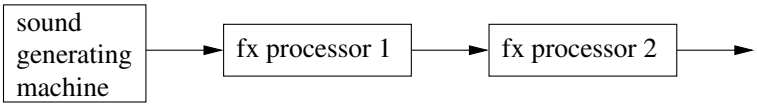
### 4.4 Summary and overview

Up till this point the overall structure of the ot is, in my view, still relatively easy to grasp:

- Samples are selected from the cf-card and put in a sample list which is available project wide.
- Samples that are in the list can be prepared for further mangling. This is done in the sample edit menus.
- Flex machines play samples from the sample list. Sample- and slice locks can be used to access any slice of any sample in the list wherever and whenever you need.

# 5 Effects

In the previous section we have seen how samples can be prepared and played back in a zillion different ways. Although this creates a lot of welcome possibilities already, it would be nice if we could wrench the samples through some effects as well. In contrast to previous models like the md and mnm, Elektron provides us with two effect slots in which we can *insert* an effect. In other words: the sound that is poduced by a machine is fed into two fx processors of our choice. Schematically speaking:




An advantage of this approach is that we can *select* which effects we want to apply to a specific sample. That' s a lot more flexible than being forced to use a fixed array of effects.

# 6 Scenes

In this section I'll describe scenes, but *not* by wasting a lot of pages on how to set them up as I would end up rewriting Elektron' s manual. Instead, I' ll look at scenes from a slightly more conceptual point of view and provide you with a simple model of how to look at them.

Basically a scene allows you to use the crossfader for controlling several parameters on several pages on several tracks at once. The idea of attaching several parameters to a crossfader is not exactly new. For example, if we use Ableton Live with Akai' s apc40 controller, we obtain the same functionality. The crossfader on the apc40 can be attached to several parameters scattered all over the place and a simple fade changes them all.

On the ot Elektron extended this functionality to a higher level: Complete collections of parameters can be stored in a scene and no less than 16 scenes can be loaded to both scene slots living on the left and right side of the crossfader. At his point it is important to look at a couple of scenarios:

**Fade from default to a scene** Assume that the crossfader is on the left and that scene slot A is muted. This effectively disables the scene and the pattern plays according to it' s default parameter settings. A scene has been loaded to scene slot B. Now, when I move the crossfader, I am fading from the default settings to the scene settings. Fading back brings me back to the default settings of the pattern.

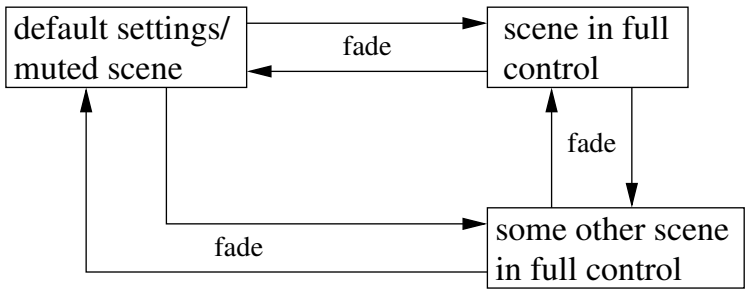
**Fade from scene to scene** Another scenario: while the crossfader is at scene slot B, I can load another scene to scene slot A. Fading back to the left now moves the parameters from scene B to scene A.

**Muting a scene** Finally, when I mute a scene, the ot immediately reverts back to it' s default settings or gradually moves back if I fade to it.

To summarize, there are three key things you can do with scenes:

- fade from default to scene
- fade from scene to scene
- fade from scene to default

Schematically speaking:




Metaphorically speaking:

A scene allows you to undertake an expedition from the base camp (default settings) into new territory (scene) and move further and further way as you keep loading new scenes and fade to them. Returning back to base camp is alway possible by muting a scene and fading to it. Of course you can also skip the fading and mute/unmute a scene while the crossfader is there. In that case you cause a sudden jump from default to scene or vice versa.

## 6.1 Summary and overview

So far the overal structure of the ot is still fairly straightforward:

- We draw samples from the cf-card, place them in a list and prepare them for further use.
- We load machines to tracks an assign (some of) our prepared samples to them.

- We set up effects to mangle sounds coming from the machines.
- We set up scenes to allow for changing several parameters at once, thus allowing us to drift away from the default settings (and back again) in a convenient way.

A crucial thing to understand here is how things depend upon each other: You first select samples, then setup the effects as you need them and once this foundation stands, you build scenes for performing on that platform.

# 7 Parts

The part system is probably the most complicated aspect of the ot. Although the part system is very powerful and allows for very intriguing setups and possibilities, it is also a huge source of confusion for both newcomers and more experienced users. Describing what parts are is difficult since they can be used in so many different ways. Therefore simply describing what a part *is* will not help since this results in a rather abstract and formal description which will keep many people puzzeled about what parts are and how to use them.

I therefore decided to take a different route. Instead of coming up with a formal definition, I will explain what parts are and what they do by describing a couple of simple experiments. By simply following this step-by-step approach our understanding of parts increases and (hopefully) evolves up to a level where all the fog has disapeared... First though, we have to dive into a little dry theory concerning a simple question: which information is stored where.

## 7.1 Parts and what they store

Question: What information does the ot need for playing back a pattern as we intended? Let' s see... First of all, it needs to know which samples we want to use. The samples are stored in the sample list as described earlier. That list, however, is available project wide. It is therefore not *specific* to our pattern. Thus, I ignore the sample list.

Next up: The ot wants to know which machines have been loaded to which tracks and how we configured them. So here we have the first valid piece of information the ot needs: machines and their settings.

Next up: the ot also wants to know what effects we use on those machines and how we configured them. So here is number two: effects and their settings.

Next up: We have machines and their effects... and we have set up several scenes that we use for performing. Scenes are therefore number three.

Next up: All tracks play their audio on their own volume. Volume settings of the tracks are therefore number four.

Finally: The ot wants to know when to trigger the samples and whether parameter locks were used to add some bells and whistles, so here is number five.

To summarize: the ot needs five chunks of information to play a pattern as we want:

- triggers and parameter locks
- scenes
- effects and their settings
- machines and their settings
- track volume settings

Next question: Where should all this stuff be stored? The most straightforward way would be to store all of this stuff into a pattern. . . and this is where it gets interesting. . . Elektron decided to do something different. Instead of storing all this information inside a pattern, Elektron has spread this information over patterns and what Elektron calls *parts*. So what is stored where? Well:

**Patterns** store triggers and parameter locks

**Parts** store machines (+settings), effects (+settings), scenes and track volumes.

So why this distinction? At first glance this seems to complicate things unnecessarily. Well, time for some action. . .

## 7.2 Part reloading

First things first: Let's build a simple pattern to work with. The pattern I built is set up as follows:

**Track 1** holds a bassdrum. To give it a bite I put a compressor on the track, followed by a delay which will be used when constructing a buildup.

**Track 2** holds a clap. To sculpt it, I put a filter and a reverb behind it.

**Track 3** holds a loop with some hi-hat stuff. Effects: filter and delay.

**Track 4** holds a bassline loop, followed by filter and lo-fi.

**Track 5** holds a pad with filter and reverb.

Further on, I set up a couple of scenes for creating some movement.
Now then, without explaining why, try the following:

- Press [FUNCTION] + [PART] to enter the parts menu. You will see four parts. The top one is highlighted and shows a small star. That star is important, more on that later.
- Press [FUNCTION] + [EDIT]. A menu appears which allows you to save the part. Save and leave the menu.
- Play the pattern. If all went well, nothing has changed and the pattern plays as usual.

Next up:

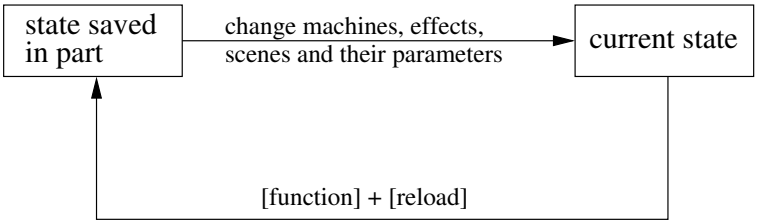
- While playing, tweak your pattern by randomly changing parameters on whatever track. Also replace effects by different ones and remove some machines or assign different samples to them. Be brutal here: change the pattern so heavily that the original pattern is completely destroyed. Important note: *Do not record parameter locks and do not add or remove triggers*. To sum it up: Change whatever *is stored in a part* but leave intact what *is stored in a pattern (trigs and parameter locks).*
- Congratulations: You just turned a nice sounding pattern into a complete mess.

Next up:

- While playing your destroyed pattern, hit [FUNCTION] + [RELOAD]. Once you do that, all destruction you caused is undone and the pattern is instantly reverted to the state it was in when you saved the part.
- Repeat the above: destroy your pattern, change parameters, replace machines, whatever, and again hit [FUNCTION] + [RELOAD] to return home.

Metaphorically speaking: By saving a part, you have set up a base camp. Once set up, you can go out for a walk and see what is out there by experimenting with different machines, effects and scenes. Delete, change and replace whatever you want. Once you get lost, hit [FUNCTION] + [RELOAD] and you are instantly zapped back home.

Schematically:




As you can see, reloading a part has some similarity with using scenes: Both can be used to drift away from the standard settings and both allow you to return home safely. Scenes, however, are only used for changing parameters and allow *fading* between parameter values.

Parts, on the other hand, are far more radical. They don' t allow fading, but they do allow replacement of machines, effects and scenes while keeping the possibility of reverting everything back to the last saved state.

So what was that little star about? Easy: Once you saved a part, the current state of the pattern is *exactly* the same as what was saved in the part. Once you make a change (by turning a parameter knob or replacing an entire machine or whatever) the current state has changed from what was saved. The star indicates that modifications have been done to the originally saved state. In other words: When you see that star, what you are currently hearing is in some way different from what you saved to the part.

### 7.3 Part reloading: usage

There are several situations in which reloading a part can be handy. Some suggestions:

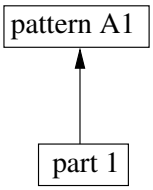
**Painless experimentation** This was described in the last section: After you saved the part you can replace, throw away and sculpt whatever you want, but the base camp is always in reach.

**Incremental development** Assume you have a pattern running with a saved part. As you listen critically to it, you find that the hi-hat is a little too loud. You correct it by reducing it' s volume. You also find that the bassdrum you used is completely rubbish and should be replaced. So you find a suitable bassdrum and dial it in. As time passes, you keep on fine tuning the pattern until it sounds profoundly better than what you started with. Once you' re satisfied: save the part! That is: move your base camp to this better sounding location.

## 7.4 Parts and patterns

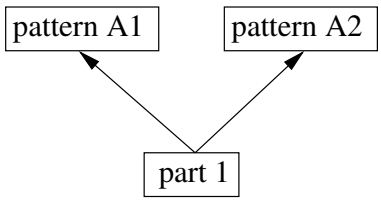
In the last section we have seen some basic usage of parts and gained some understanding about how they work. This is not the big picture though. In this section I'll focus on the relation between a part and a *collection* of patterns.

I' ll start with the same example I used earlier. Assume that the pattern lives on a1, that it is using part1 and that the part has been saved. This gives me a clean point to start from. Before going further it is important to understand what the memory of the ot looks like at this point. The following diagram shows the current situation:




The arrow illustrates that the pattern is using the part. Remember: Patterns store triggers and locks, while the part stores the underlying infrastructure: machines, effects, scenes and track volumes.

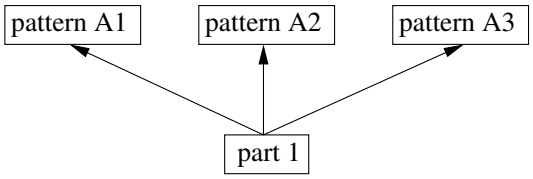
What happens when we copy pattern a1 to a2? Well... It' s obvious that all triggers and parameter locks that live on pattern a1 are now copied to pattern a2. But what happens to the part? Is a hard copy of the part made as well? Nope... The part is left unchanged. Instead, pattern a2 now *also* uses part 1. Schematically:




Think a moment about the implications of this...

If I change something to the part while pattern a2 is running, that change will be reflected in pattern a1 as well, since it uses the same part. This is a huge source of confusion to many users as they, wrongly, think that a pattern stores all of it' s information locally. This is not so as I described earlier. We therefore conclude something very important: *Any changes you make to a part will be reflected in all patterns that use that part.*

Let' s expand a little further on this. I move back to pattern a1 and copy it to Pattern a3.
Schematically:




At this point you should be able to figure out where this is going. I can create various variants of a pattern by copying and pasting at will and all these patterns will use the same underlying machines, effects and scenes. Changing something to the machines, effects or scenes in any of those patterns will be reflected in all other patterns that use this part.

So what is the purpose of this? Easy: Convenience and coherency. For example: Suppose that the resonance of my filter in track 5 is a little too aggressive. So I tame it a bit. If every pattern stored it' s information locally, I would have to change this on pattern a1, then move to pattern a2 and change it again, then move to pattern a3 and change it for the third time. . . which profoundly annoys me.

With multiple patterns using the same part, I only need to make changes once and all patterns using that part follow with it. This saves me a lot of dull work and guarantees that all patterns form a coherent collection of variations, all using machines that sound the same, effects that behave the same and scenes that work the same over all these patterns.

This expands to *performing* with multiple patterns: Suppose a1 is running and that I make changes to various parameters, track volumes, etc. When I switch to a2 (or any pattern using this part), the parameters I changed will *keep* being changed (as long as the part is not reloaded), allowing for a natural switch from a1 to a2. If every pattern stored these settings locally, switching a pattern would cause all kinds of parameter jumps.

The only real difference between pattern a1 and a2 lies in the triggers and the parameter locks. Although this may look a little silly at his point, remember that parameter locks can be used to play *any* slice of *any* sample in the sample list so triggers and locks alone already create enough mayhem to turn pattern a2 into something that is very different from a1 and a3.
To top it off, consider a simple real world example:

- Pattern a1 contains my basic pattern. I can use scenes to perform some tricks on this pattern and also use the part reload function I described earlier.

- Pattern a2 contains a copy of pattern a1, but here I added some extra triggers on my bassdrum at the end of the pattern. I also used some

parameter locks to play the bassdrum in reverse. I also used some locks to retrigger a couple of hi-hats and, finally, I used locks to add dirt to my bassline on track 4.

- As pattern a1 ends I instruct the ot to move on to a2. A2 contains the buildup and I patiently wait for it to develop. As a2 ends I direct the ot to pattern a3...
- Pattern a3 serves as a break: All triggers on the bassdrum were deleted. Instead I put one trigger on step 1 and locked it to a sample in the list which contains a massive noise sound that slowly increases in volume and washes away everything. As this pattern approaches it end, I direct the ot to pattern a1 and everything starts all over again...

## 7.5 Scene stacking

In the last section we saw how multiple patterns can use the same part. Scenes are stored in the part. This means that once a scene is active, it will *remain* active even if I move on to the next pattern. In other words: 16 scenes can be used freely over all patterns that use the same part. This can be used to move from scene to scene and from pattern to pattern in a natural way. A slight problem emerges, however: I can prepare 16 scenes for use with my patterns, but how do I remember which scene does what? Ok...I could write it down, but there is a simple way of letting the movement through patterns go hand in hand with the movement through scenes. I call it *scene stacking.*
The idea is easy:

- When the first pattern runs, it runs without being affected by any scene.
- I move the crossfader to the right and load scene 1 to the left. Fading to the left now puts scene 1 in full control.
- I copy scene 1 to scene 9 and load scene 9 to the right. Crossfading to the right now has no effect, since scene 9 holds the same parameter values.
- I make all changes I want to scene 9. Since the parameters of scene 1 are still stored in scene 9, I can move naturally from 1 to 9. The parameters that changed in scene 9 are stacked *on top of* what already was changed in scene 1.
- I copy scene 9 to scene 2 which is loaded to the left. Again I make changes *on top of* what was changed in scene 9.

- copy 2 to 10, modify and fade
- copy 10 to 3, modify and fade
- copy 3 to 11, modify and fade
- ...

By putting up scenes in this way I can move through them in a natural way: Scenes 1 to 8 are always on the left while scenes 9 to 16 are always on the right. Fading from scene to scene is easy:

- If the crossfader is on the left, I press the right scene button, look at the leds (9-16) and I load the scene that comes after the one currently loaded.
- If the crossfader is on the right, I press the left scene button, look at the leds (1-9) and I load the scene that comes after the one currently loaded.

Since moving through scenes has now become easy, moving from one pattern to the next one has become easy as well. As I move through the patterns, I fade from scene to scene. Those scenes have been setup in such a way that they create a fluent progression through the patterns...

## 7.6 Using multiple parts

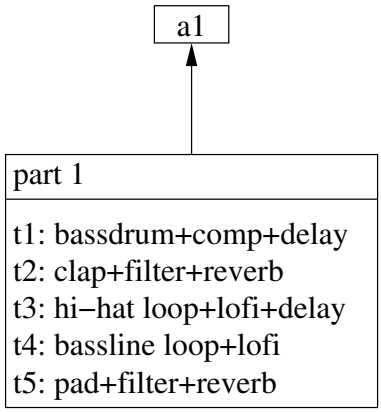
As we have seen in the previous section, parts store machines, effects, scenes and volume. Several patterns can use that part and sequence their machines in various ways by playing around with triggers and parameter locks.

So far, however, we have only used one part and as you have probably figured out already, the ot can hold up to four parts per bank. Four parts, means we have four individual "platforms" (machines, effects, scenes and volume) we can feed to our patterns.

In this section I will extensively cover an example which shows how using multiple patterns, scenes and parts can be used for constructing a transition between two patterns.
To get started, assume the following, very basic, setup:

- A pattern lives on track one.
- The pattern uses part one.
- The part is saved: basecamp is secured.

Instead of summing up what kind of sounds were loaded to which track, I put this in the following diagram:




I can play around a little with this pattern by changing some parameters, do some mutes and unmutes, manually triggering samples, etc. Remember I can always return to the basecamp by using the part reload function as described earlier.

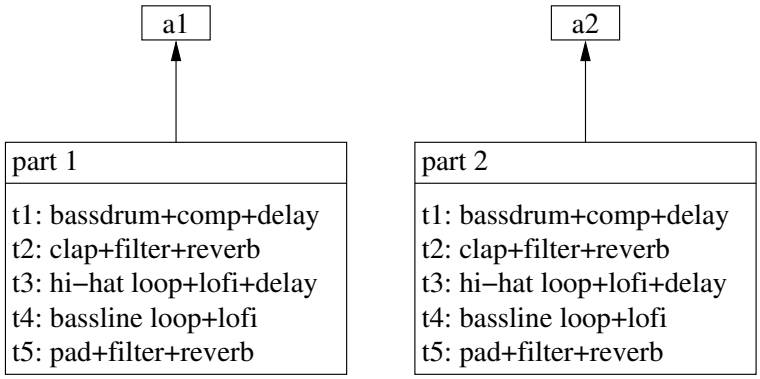
Time for a change: I want to construct a buildup. I do so by setting up a scene on scene slot b. The scene is constructed easily:

- The bassdrum feeds a delay, which results in a bassdrum bouncing all over the place.
- The filter on the clap is opened, resonance is increased a little and reverb is added.
- The rate reduce and distortion parameters on the hi-hat are used for creating artifacts and destruction.
- The distortion parameter is increased on the bassline to let it go nuts.

With this scene loaded, a simple and slow fade of the crossfader from left to right pumps more and more madness into the pattern. As I keep moving the fader, the buildup grows more and more... to what?! Well... ehh... Injecting a buildup is all fine, but sooner or later, a buildup has to be followed by a move into a new direction. I can move to a new pattern or fade to a new scene as was described in the last section, but the problem is that whatever I do, the machines and effects will remain the same. For really making a move into a new direction, some more radical things have to be done:

- I copy pattern a1 to a2. As described earlier, we now have two patterns both using the same part.
- While in pattern a2, I turn to the part menu and copy part 1 to part 2.
- Copying from part 1 to part 2 still doesn' t mean that a2 is *using* part 2. So I select it and hit [YES].
- I save all parts.
- I look ot the ot' s screen and verify that a2 is indeed using part 2.

After the above operations, the memory of the ot looks as follows:

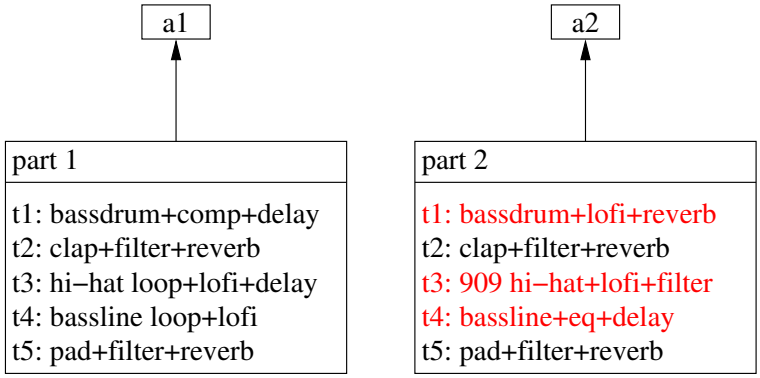



As you can see, I now have two patterns, *both using their own part*. Since a2 is a copy of a1 and part 2 is a copy of part 1, both patterns will sound exactly the same. Yet, modifying part 2 has now become fully independent of part 1.

Time for some changes: I want a2 to sound far more aggressive than a1. In order to achieve this I modify several elements in part 2:

- The bassdrum is replaced by a more aggressive one. I also replace the effects: lofi and reverb are now in place.
- The hi-hat loop is replaced with an aggressive 909 hi-hat+lofi+filter.
- The bassline is replaced by a really deep and pumping one+eq+delay.
- Clap and pad are left unchanged.

I also add some triggers and parameter locks on various tracks and tweak and finetune until I have the loop I want. To make these changes permanent, I save all parts. After the above operations, the ot memory looks like this:




The differences between part 1 and part 2 have been coloured red. Right: all has been set. . . let' s perform:

- I switch to a1 and hit play.
- I load my buildup scene to scene slot b.
- Time for the buildup: I slowly move the crossfader and the pattern starts building and building. . .
- At a moment of my choice, I hit [PATTERN] + [A2]. At the moment I do that, I already know that when a1 ends, it will be followed by the more aggressive sounding a2.
- I wait until a1 ends and on the moment a2 kicks in, I mute scene slot b. By muting scene b, a2 plays without being affected by the scene.
- A2 now plays cleanly: The new bassdrum and bassline really punch away and the 909 hi-hat cuts through the air. The original pad and clap from part 1 are still present as they were: This new pattern still has some remnants of it' s predecessor but the newly added elements pushed this pattern into a new direction.

I am now left with a2 which uses it' s own part. I can therefore setup scenes that are tailored for that pattern, goof around with parameters and the part reload function, mute, unmute, etc. I can also copy a2 to other patterns and build variations, knowing that all these variations use the same part. Effectively the game starts all over again.

## 7.7 A small but convenient enhancement...

In the previous section I constructed a pattern, used a scene to create a buildup and moved to the next pattern, which uses it' s own part. Once a2 kicks in, I mute the scene and a2 runs completely clean.

Running clean in this case means that the aggressive bassline I installed on part 2 is *immediately* heard on full volume at the moment a2 kicks in. Nothing wrong with this, but suppose that I don' t want that. Suppose I *don' t* want to hear the bassline at the moment a2 kicks in and that I want to fade it in myself as a2 plays on... How to do that?

At first glance, a mute of the bassline track could be set on the moment a2 kicks in. Although this is possible, it requires me to be very precise in my timing. To make things worse, I already had to care about timing since I had to mute scene B at the exact moment a2 kicks in. This is obviously not convenient. Let' s explore some solutions...

### 7.7.1 Parts store track volumes

A part stores machines, effects, scenes and the volume at which the tracks feed their audio to the main outs. Therefore setting the track volume on the bassline' s track to 0 and saving the part, wil "mute" the track once a2 kicks in. I can then select the track and increase the volume as the pattern plays on.

There is, however, a downside to this: Every time the part loads, the volume of that track is set to 0. Every time I use the part reload function, the track drops to 0. So as long as the part doesn' t reload, I am safe, but this is not ideal.

### 7.7.2 Parts store scenes

Remember that a part stores machines, effects, scenes and volume. The fact that a part stores scenes is very important here. While a1 is playing, the scenes in part 1 can be used to do whatever you want. For creating the buildup towards a2, I loaded scene 9 in scene slot b and slowly faded to it. So... Once a1 ends, the fader is pushed full to the right and scene 9 is in full control.

Then a2 kicks in and loads it' s own part. The fact that a2 loads it' s own part, implies that it *also loads it' s own scenes!* Elektron went one step further though: A part also stores which scenes are loaded to scene slots a and b and also whether a scene is active or muted. Let' s use that to our advantage:

- I move to a2 and *delete* scene 9.

- I load a clean scene 9 to scene slot b. By starting with a clean scene, I make sure that no unexpected parameters are affected by that scene.
- I configure scene 9 in such a way that the volume of the aggressive bassline is set to 0.
- I save the part to make this permanent.

And there it is: Once a2 kicks in, I don' t need to mute scene b at that exact moment anymore. In fact, I don' t need to do anything: At the moment the ot switches from a1 to a2, part 2 is loaded and thus scene 9 of part 2 is loaded as well. This new scene kills the volume of the bassline. Moving back the fader to the left (where no scene lives at all) fades in the bassline to it' s normal level. The pattern now runs cleanly.
Metaphorically speaking: I have left my previous basecamp for good and arrived at my new home. Since I really liked the clap and pad, I took them with me, but someday when I move to the next basecamp, I might leave them behind...

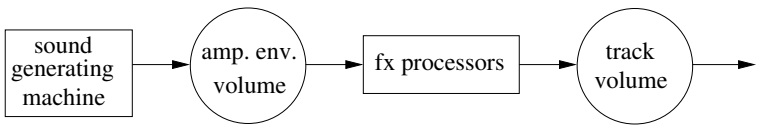
# 8 Volume handling

Once a machine plays a sample, several parameters affect it' s volume. Understanding which parameters live where and how they interact with scenes will avoid several volume handling pitfalls.

## 8.1 Signal flow

The signal that a machine generates, is first wrenched through the amplifier envelope. On the amp page, you will find the vol parameter which is a bipolar parameter, meaning that it' s range goes from -64 to +63. The parameter defaults to 0. Generating tremolo effects can be done by letting an lfo modulate this parameter. Mimicking side chain compression is done by setting up an lfo in such a way that it decreases volume on those moments a bass drum kicks in. Parameter locks can be used for that as well.

After the amp envelope, the signal moves through two fx processors. Finally the track volume parameter determines the overall volume of the whole track (sample+fx). Schematically:



As can be seen in the diagram, the ot offers both pre and post fx volume handling on each track.

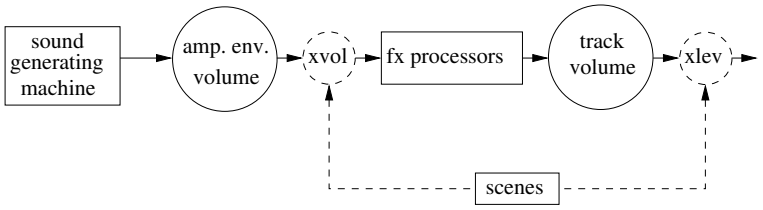
## 8.2 Volume and scenes

On top of the regular volume parameters, the ot offers two additional parameters which are exclusively available to the scenes:

**Track volume: xlev** can be set to min or max. The parameter is shown once you hold one of the scene buttons. Min equates simply to 0 and max equates to the volume that is currently set (that is: if the scene is muted). It thus allows you to "mute" a track when the scene is active or to fade in a track to the "usual" track volume.

**Amplifier volume: xvol** is found in the amp envelope page when holding a scene button. It works just as the xlev parameter, but this time it affects the amp volume parameter.

Schematically:




The xvol and xlev parameters are very useful as they allow the scenes to fade in and out tracks both pre and post fx. If, for example, some signal is processed by a delay with lots of feedback or a reverb with a long reverb tail, fading out a track using the xvol parameter allows the delay/reverb to die out naturally. From this it follows that during a performance most track mixing is ideally done by programming the xlev and xvol parameters in the scenes.

## 8.3 Correcting sample volume

Let's assume you have a bass drum running on pattern a1. The bass drum sample fits the rest of the pattern and both it' s amp and track volumes have the level you want. The settings are saved to the part.

Pattern a2 is a copy of pattern a1 with one exception: the bass drum was replaced by a different one using sample locks. Although that new bass drum has the character you want, it unfortunately is way too loud. How to correct that?

At first glance it is tempting to decrease the track volume, but this creates a problem: once you decrease the track volume, the bass drum living on a1 becomes too soft. Apparently we bumped into a situation of mutually exclusiveness: either the a1 bass drum is too soft or the a2 bass drum is too loud. Changing track volume is obviously not a solution so I leave the track volume knob untouched.

Perhaps using parameter locks on the amp volume parameter does the job. It indeed does, but now I have to remember that the bass drum should be locked on every step I program in patterns that use that bass drum. Although this is a solution, it is still not very convenient.

Effectively the problem is this: everything on your patterns was ok, until that misbehaving bass drum was dialed in. The problem is caused by the sample itself and, thus, should be corrected on the sample level and not by disrupting exisiting settings which also affect other elements. Basic train of thought: the sample should adapt to the system and not the other way around.

Fortunately the ot offers a simple way to correct "misbehaving" samples: On the attributes page in the sample editor a gain parameter is found. This parameter allows you to boost or decrease the sample volume with a massive 24dB in both directions. Simply start the pattern and set the volume to taste in the attributes page: Many problems with inconsistent track volumes or parameter locks are now history since that parameter is considered an attibute of the sample and stored in the sample slot list.

# 9 Sampling

So far I have completely ignored the ot' s abilities for sampling. I had a simple reason for doing so: Explaining the overall structure of the ot is hard enough and talking about sampling would not have changed the overview. In order to avoid making things more complicated than necessary, I therefore avoided the sampling issue as a whole.

## 9.1 The ot' s approach to sampling

When I look at samplers of the past, I can see that most samplers have the ability of recording new material from the outside world and playing it back. Indeed this is obvious, but the point is that on most samplers these two features have been completely seperated. Most samplers offer the user a "sampling mode" and a "playback mode" and for good reasons: When recording new material, you enter sampling mode, hit start and make the recording. Once the recording has taken

place, some facilities are offered for trimming and preparing the sample before it, finally, can be used for playback. Throughout the years several enhancements have been added. For example: midi can be used to synchronize the sampling and playback to external gear, etc.

Then, in 2004, Ableton Live 4 hit the markets. Ableton took a different view on sampling: It offers a system in which sampling and playing back samples can be done without stopping the sequencer or switching between sampling and playback modes. In other words: Ableton integrated both in a system which is under direct control of a sequencer and a user manipulating it.

The biggest difference between Live 4 and Live 3 lay in the fact that Live 4 was the first version that allowed the use of vst synths and effects, thereby connecting the world of sampling with the world of producing and blurring the distinction between both.

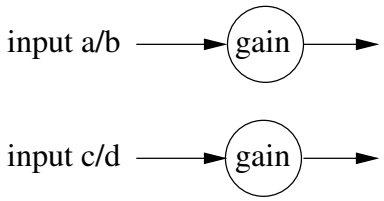
At that time, Elektron must have been thinking about integrating samplers in their products already. In 2005 the mduw was introduced and this unit takes more or less the same approach to sampling as Ableton: Both sampling and playback can be done without stopping the sequencer. By doing so, sampling new material has become just as trivial as playing back samples. This approach allows for a seamless integration of sampling in a performance and that is a big step forward from what already existed at that time.

## 9.2 Signal flow

Before diving into the sampling process itself, a few words should be spent on signal flow. Perhaps a little dull, but it is important as understanding the signal flow will avoid problems with volume.
First things first: The ot offers two stereo line inputs, simply named input a/b and input c/d. When external equipment is connected to the ot, a small problem appears: Some synthesizers deliver a very hot signal while others deliver a signal that is too soft. The ot, therefore, should offer a facility to boost or suppress an incoming signal *before* that signal hits the rest of the system.

The ot offers simple gain parameters for both a/b and c/d in the mixer page. The gain allows for boosting an incoming signal with 12dB or suppressing it to 0. The role of these parameters should be clear: you use the gain for correcting the volume of incoming signals so the rest of the ot receives a healthy signal. Ideally, the gain should be configured once and be left untouched throughout the rest of the project. Set and forget. The gain parameters can therefore not be automated, sequenced, or whatever. They exist for providing the ot with healthy signals from the outside world and that' s it. Any tricks on volume, fading in or out during performance etc. should be made elsewere. . .

Schematically:




Now that we have a healthy signal, we can do two things with it:

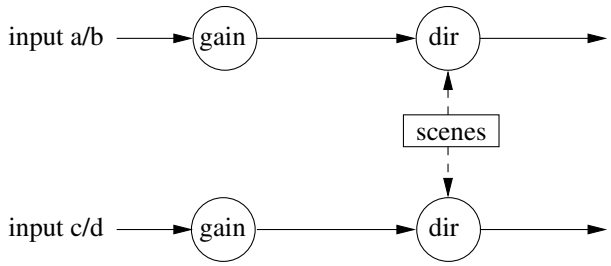
- send it straight to the main outs
- feed it to various track recorders

### 9.2.1 Main outs

Sending the signals to the main outs is probably the easiest part: The dir parameters in the mixer page are used for that. By increasing the dir parameter you determine how loud the incoming signal is fed *directly* to the main outs. Hence the name "dir". All other things that happen inside the ot are completely separated from this, with one important exception:

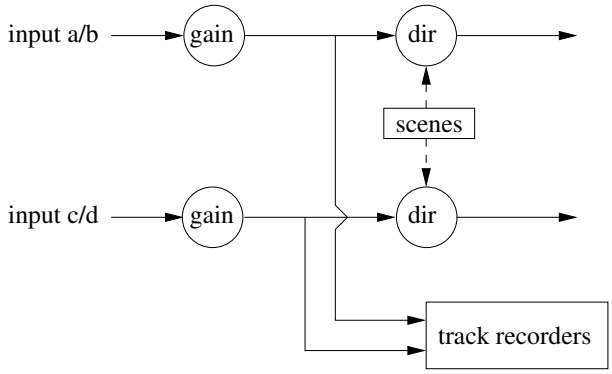
Feeding an incoming signal directly to the main outs is all fine, but during a performance I' d like to fade the live inputs in and out, when I need them. During a performance, however, I am probably already goofing around with various track parameters, scenes, etc. Every time I want to fade in or out the live inputs, I have to turn to the mixer screen to control them. Not very convenient. . .

Fortunately, the volume of the live inputs can be controlled with scenes. While in the mixer page, hold the scene button and you will see that the dir parameters turn to "xdir". I can now set the volume of both inputs to maxium or minimum. This allows me to fade in or out the live inputs with the crossfader whithout opening the mixer page. Live inputs are now fully integrated in a performance. I can use the scenes to control parameters on all tracks and mix in or out the live inputs as I need them. Effectively I can use the scenes for controlling ten "channels" of audio: eight tracks from the ot itself and another two live inputs.
Schematically:




### 9.2.2 Track recorders

Live inputs can be sent directly to the main out, but we can also feed them to the track recorders for sampling.
Schematically:




Designing the signal flow in this way is pretty clever as it allows for fading in or out the live inputs, irrespective of the sampling process. I can thus sample without hearing the inputs, or vice versa or any combination.

## 9.3 Sampling guidelines

The ot features eight fully independent recorders on each of it' s tracks. The recorders are built "on top" of a track and are always available, regardless of what happens on a track. Each of these recorders can record from any combination of three sources: The live inputs a/b and/or c/d and/or a third source, which can be set to any track, the main output or the cue output.

Recording from any of these sources can be triggered by the sequencer or manually and I will not dive into the details of that here, since I would end up rewriting Elektron' s manual. I will, however, present you a couple of brief guidelines:

**Sequenced sampling** By setting up recorder triggers, the recording process is under full control of the sequencer. Triggering recorders from the sequencer has an advantage over manual sampling: You guarantee that sampling is always synchronized to the sequencer. Thus trimming or further preparation of the sample is usually not needed. It can be used directly in the rest of the system without stopping the sequencer.

**One shot triggers** If you want to sample *once* without overwriting the sample the next time the pattern loops, you can use one shot triggers. They trigger the recorder once but no more. The one shot triggers can be re-armed at will, so this is a nice technique if you want sequenced sampling, but only when you need it. This more or less equals to triggering a recorder in Ableton Live.

**Manual sampling** By triggering the samplers manually, you throw away the timing advantages of sequenced sampling, but the inadequacies of your own timing can lead to happy accidents.

**Quantized sampling** This is a form of manually triggering the recorders, but by setting the qrec and qpl parameters in the record setup page the recorders are not triggered instantly, but quantized to the ot' s sequencer. This more or less equals to triggering a recorder in Ableton Live.

## 9.4 Track recorders and the sample slot list

So where do these recorders write their audio? Well. . . Every track recorder has it' s own dedicated recordbuffer. This means that a recorder on track one always writes it' s audio to recordbuffer one. This is hardwired into the system.

So where live the recordbuffers? Answer: In the sample slot list. Think a moment about the implications of this.

As described earlier, the sample slot list is available *project wide*. In other words: once a recorder writes something to it' s recordbuffer, that sample is instantly available on any track holding a flex machine in any pattern in any bank as long as the track recorder doesn' t overwrite the sample. . . and even if it does, that new sample is, again, immediately available everywhere.

Finally, a couple of handy details about the recordbuffers:

**Editing** There is no difference between a recordbuffer and a regular sample slot. So anything that is put into a recordbuffer can be finetuned with the sample editor. This includes slicing. . .

**Saving to cf-card** If you want to secure a sample that is currently in the recordbuffer, you can dive in the sample editor and move to the file page. There you find the option "save and assign sample". By selecting that, the contents of the recordbuffer are written to the cf-card. You decide how it should be named and afterwards, the ot automatically assigns that sample to a free sample slot. This is handy: If you want to save a recorded sample from destruction and immediately make it available in your project, this is the way to go.

## 9.5 Sampling techniques

I will now describe a couple of sampling strategies.

### 9.5.1 Dedicating patterns

The fact that a recorded sample is instantly available anywhere, creates many possibilities. One could, for example, set a couple of recorder triggers in a pattern and immediately move on to the next pattern which doesn' t have these triggers. In this way a pattern could be regarded as a recorder fly-by: move to the pattern for recording what you need and immediately leave to the next pattern where (some of) these samples are used, amongst others.

Although sacrificing a full pattern to hold just a couple of recorder triggers, may sound like a waste of patterns, we have enough of them, so this is not a big deal. Of course, one could also use one-shot triggers for setting up recorder fly-bys.

### 9.5.2 Abusing the cue outputs

Every track in the ot can be sent to the master output, but also to the cue outputs. The cue outputs are independent of the master output. As described earlier, a track recorder can record the live inputs and a third source. This third source can be any track, the master outs *and the cue ouput*.

So. . . Setup a track recorder to record the cue output. Whatever combination of live inputs and tracks you need, can be sent to it by cueing them, irrespective of what happens on the master outs. If, for example you need a recording of the bassdrum, hihats, claps and input c/d, just cue these four, trigger the recorder the way you want and you obtain a sample of what you selected.

If you need some form of pre-listening, plug in some headphones and assign the headphones to the cue output in the mixer page.

### 9.5.3 Abusing a track for live input

This sample technique is nasty and not described in the manual as a seperate technique, but it works and is, I.m.h.o. very usable so I present it here.

As described earlier, track recorders and tracks are independent. A flex machine running on a track will not bite the track recorder on the same track and vice versa... There is, however, a way of setting up a track recorder and a flex machine in such a way that they *do* bite each other in a useful way. I set it up as follows:

- Set up a track recorder and let it record the live input. Record length should match the pattern length.
- Place a full trigger on step 1. By doing so, the recorder keeps on recording, boldly overwriting the last recording every time the pattern reloops. Although the recorder does it' s job, I cannot hear the live input.
- Set up a flex machine on the same track.
- Assign the flex machine to it' s own record buffer. In other words: if the recorder of track 4 was used, we now put a flex machine on track 4 and assign it to recordbuffer 4.
- Trigger the flex machine on step 1. Indeed this is the same step on which the recorder was triggered.

We have now set up a strange situation: the track recorder is writing to the buffer while the flexmachine is reading from it *at the same time...*

So what happens? Well, and this *is* documented in the manual, in such a situation, you will hear the live inputs. This yields a couple of advantages:

- Removing the recorder trigger breaks the read/write situation. Result: the flexplayer simply plays the contents of the track recorder. Live input has gone.
- Whether you hear the live input or the sample, they are both affected by the track effects.
- The track volume can be used to fade in or out the track as usual, whether it plays the buffer or passes the live input. Oh, and did I mention scenes?