Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

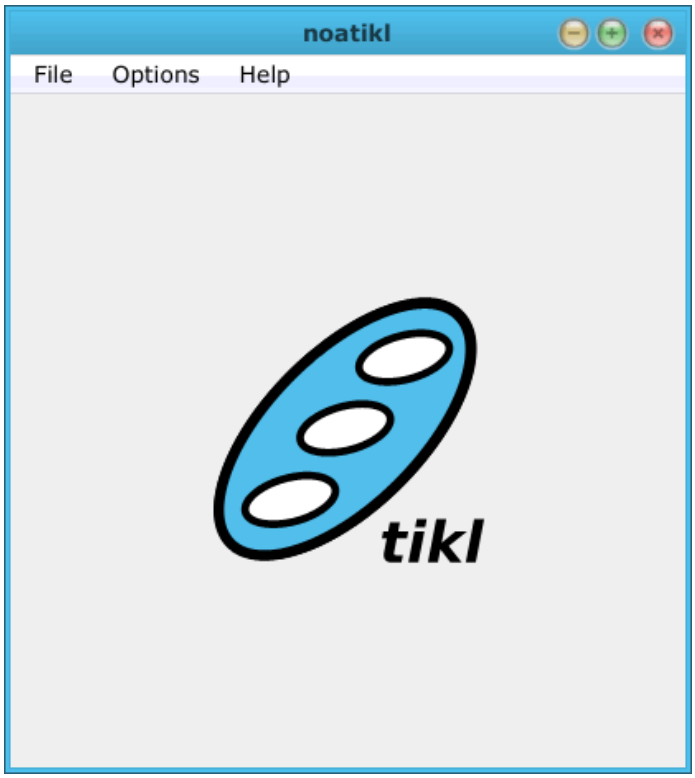
Noatikl User Guide

next

Contents

General

Introduction
The initial idea
How it works
Learning to delegate
Koan & Koan Pro
Brian Eno
Noatikl variants
Standalone or Plug-in?
Installation
Standalone & Logic/Mac
I get silence, help!
Now what?
Registration
The interface
Sync?
Object types
Tutorials
Tip: Auto-control Logic tempo



Noatikl UI

Views - Voice

Basics
Listening Voices
Ambient
Following
Repeat
Patterns
Chords
Rules
Scripts
Notes

Views - Controllers

Controllers
Micro Controller 1
Micro Controller 2
Micro Note Delay
Micro Pitch
Note to MIDI CC Mapping

Views - Envelopes

- Velocity
- Velocity Range
- Velocity Change
- Velocity Change Range
- User Envelope 1 (Volume)
- Micro User Envelope 1
- User Envelope 2 (Pan)

Views - Rules

- Scale Rule
- Harmony Rule
- Next Note Rule
- Rhythm Rule

Views - Piece

- Basics
- Tempo
- Rules
- Roots
- Scripts
- File

Patterns

- Examples
- General syntax
- Sequence sub-patterns

Scripting

- Overview
- Hyperinstrument
- Reference

Credits

- Credits

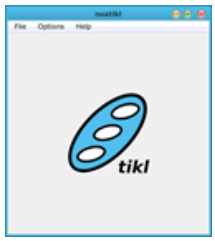
Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Introduction to Noatikl



Noatikl is a powerful MIDI-based generative music engine that can help you come up with new musical ideas. It can also be used as a generative MIDI event controller allowing you to bring generative variety to sound shaping and FX where your synth, sampler or FX unit allows parameters to be controlled via MIDI controllers. Noatikl has extensive built-in scripting features. Noatikl can also be used as a customisable hyperinstrument!

It is available for both Windows and Mac OS X and comes in a number of different variants so that it can fit it with the way you prefer to use your favourite audio tools!

Noatikl draws on the Zen koan tradition in that there seems no obvious way it generates its ideas, and it is pronounced "noh - tickle". To quote a haiku by Mark Harrop:

Music, recorded,
A meal, frozen.
noatikl: Real,
home, fresh cooking.

Noatikl is created by Pete Cole and Tim Cole, creators of the 2001 BAFTA-award winning SSEYO Koan Interactive Audio Platform, and the 2005 BAFTA-award winning SSEYO miniMIXA.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

The Initial Idea

Noatikl is a new generative music system for a world where the Koan Music Engine (created by Tim and Pete Cole in a previous company called SSEYO) is no longer available. With Noatikl, we've started again from a completely clean slate, focused on creating a range of Noatikl variants in plug-in form that can be used the desktop tool sequencers that everybody in music uses these days. We've also created a standalone version of Noatikl that can sync where required with your favourite sequencer. Generative music is here to stay!

Our original inspiration for our approach to music engines, came from the classic Foundation series by Isaac Asimov. In this series, Asimov introduced an imaginary device called a Visi-Sonor, which could be manipulated by a skilled operator to create amazing multimedia entertainments combining music and visuals. You can think of this device as a kind of hyper-instrument. Tim was really excited about this idea, and it kept coming back to him, until eventually we started to create software that went some way to fulfilling Asimov's wonderful vision.

The imagery for Noatikl is meant to symbolise empty note placeholders on a surfboard, that can be filled by an engine. With Noatikl, you can surf the wave of sound and enjoy the fluidity of its music. Also, Noatikl is primarily a MIDI note generator, so we thought “note tikl” was appropriate. This concept moved along similar lines to those for Liptikl, in that the product would be tickling notes into existence, and hopefully resulting in cool new ideas.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

How it Works

Noatikl allows you to tell your computer automatically to create both music and MIDI controller events, based on properties that you define through the user interface.

Yes, that is right: you let Noatikl create the composition for you, in real time, in all its detail! Your job is to use the Noatikl user interface to tell Noatikl which rules to follow, to make the music you want to hear.

The Noatikl user interface allows you to define various musical objects which go together to make your Noatikl composition. When you start your piece playing, the Noatikl music engine analyses the objects, and according to the rules you have defined, chooses for itself the exact notes and MIDI controllers to generate in real time!

The reason this all sounds great, is that at its heart, the Noatikl music engine uses random events in combination with a a powerful set of musical rules. How you interpret what you hear is filtered through your own internal knowledge of music. This combination of chance and logic is what allows Noatikl to keep the music fresh and interesting, and unpredictable.

One of the wonderful features of Noatikl is that you are allowed to change all the properties of the piece in real-time while Noatikl is playing, so you can hear the effect of your rule changes as you apply them. As a result, the process of using Noatikl to create music is very interactive and rewarding.

The process of using Noatikl to create music combines various skills, encouraging you sometimes to simply listen and enjoy or analyse what the music engine is doing.

One of Tim's favourite analogies, is that Noatikl music can be thought of as being comparable to a set of ball bearings traveling down a Bagatelle or Pachinko game. Each time the ball bearings makes their journey, they will combine to travel a different set of paths, bouncing off each other as they go, but the available paths are constrained by the internal physical design and boundaries of the game. In a similar way, music created by Noatikl is governed by an envelope of possibilities. Each time a Noatikl piece plays it will have certain boundaries set by the author, outside which the music will not go. The result is that the music created by Noatikl can be different every time you hear it. That keeps the music fresh and interesting. This is the difference between looking at a photograph of a river, and watching a real live river flowing in nature.

And in case you were wondering, you don't have to use Noatikl as a composer. It is a very, very powerful MIDI event generator with enormous depth; use it to make your mixes hugely more dynamic!

Noatikl also works as a hyperinstrument, accomodating incoming MIDI note events in a customisable way.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Learning to Delegate

There are many rules used to create music within noatikl. When the Noatikl music engine is figuring-out what to do, it combines all the rules you have told it to use, and respects them as best it can, but ultimately Noatikl makes its own choice as to exactly what to do. In other words, you can give the engine brain lots of guidance, but ultimately (like a child) you let Noatikl make the final, detailed decisions as to what to do.

One of the things that a first user of Noatikl might find, is that delegating responsibility to a music engine feels like a weird thing to do!

However, give this a little time. Many, many users of our generative software have found that this process of learning to sit-back and delegate responsibility for small details to a generative engine is extremely liberating. You instead start to focus on other things, like tweaking the sounds used to render the music, or just sitting back or wandering around enjoying what you are hearing in the background, waiting for nuggets of musical beauty to appear when you least expect them. Yes, you can focus on details in Noatikl, but you can also take more of a view of a gardener's view to creating music: casting musical idea seeds on the ground, and selecting those ideas which blossom and discarding those that don't appeal.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

Koan and Koan Pro

Koan is a generative music system that was initially conceived in 1990 by SSEYO (a company founded by Tim and Pete Cole a long time ago). Koan and the Koan Pro authoring tool are no longer available, but Intermorphic has acquired the rights to them.

Noatikl is a brand new generative music engine, and what Intermorphic has done is to try very hard to make Noatikl a product that users of the old Koan system will come to love. We have also figured-out an easy way to get a lot of your old Koan data into Noatikl, to help you get started with Noatikl.

How do I get data from Koan Pro into Noatikl??

This uses one of the very earliest features of Koan Pro, where you can export a Koan file in as a text file listing that you could inspect at your leisure. This comes from the early days (before Koan really had a user interface!) when Koan files used to be plain text files that you edited by hand.

To get your old files from Koan Pro into Noatikl:

1. Using Koan Pro, open-up the file you want to use in Noatikl (e.g. MYTHOUS.SKD)
2. Save the file as a text file (e.g. mythous.txt) using the File -> Create Text File Listing menu item.
3. Using Noatikl, open-up the text file (e.g. mythous.txt) that you saved in the previous step.
4. Try playing the file using noatikl. It should sound pretty close to the original (but not identical!).
5. Save your file from Noatikl, as mythous.noatikl. You can now re-edit the file using Noatikl, at your leisure!

How does this work?

The Text File Listings that can be exported by Koan Pro were designed to be used by anybody who wanted to get their head around the many parameters used by the old Koan music engine. This made it very easy for us to write code that allowed Noatikl to scan the text fields in the text file listing, and import that data into the Noatikl engine.

Of course, the Noatikl engine is totally unrelated to the old Koan engine, so while we do our best to render old Koan data through Noatikl, what you hear won't sound absolutely identical to how it used to sound. For example, Noatikl doesn't support various features that were in Koan Pro, such as software synthesis. Also, the Noatikl music engine is based on completely different code to that used in Koan, and that inevitably makes things sound different.

There is also no way to get data from Noatikl back into Koan Pro. Though we're not sure why you'd want to try that anyhow!

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

Brian Eno

No less a luminary than Brian Eno has used music engines to create fascinating (and fabulous) music. In fact, he used our earlier generative music engine (Koan) to create his extraordinary hybrid album Generative Music 1.

Brian Eno has had a lot of interesting things to say on the subject of generative music. You can find out more about this by searching on the internet. His early relationship with Koan Pro was captured in his 1996 diary "A Year with Swollen Appendices".

Logic, Mac and Koan (or not)

Having worked with Eno we know that he uses Logic on a Mac. Koan was never designed to work with Logic/Mac, so all those years ago he used to have to use Koan on a specially purchased Windows PC (much to his chagrin). We knew that one of the *big goals* we needed to have for Noatikl was to make sure that this time we properly looked after Logic users – and we are pleased to say we have with both a Noatikl AU plugin and a standalone Mac version.

Brian Eno, 1996:

"Generative Music 1" with SSEYO Koan software

"Some very basic forms of generative music have existed for a long time, but as marginal curiosities. Wind chimes are an example, but the only compositional control you have over the music they produce is in the original choice of notes that the chimes will sound. Recently, however, out of the union of synthesisers and computers, some much finer tools have evolved. Koan Software is probably the best of these systems, allowing a composer to control not one but one hundred and fifty musical and sonic parameters within which the computer then improvises (as wind improvises the wind chimes)."

"The works I have made with this system symbolise to me the beginning of a new era of music. Until 100 years ago, every musical event was unique: music was ephemeral and unrepeatable and even classical scoring couldn't guarantee precise duplication. Then came the gramophone record, which captured particular performances and made it possible to hear them identically over and over again.

But now there are three alternatives: live music, recorded music and generative music. Generative music enjoys some of the benefits of both its ancestors. Like live music it is always different. Like recorded music it is free of time-and-place limitations - you can hear it when and where you want.

I really think it is possible that our grandchildren will look at us in wonder and say: "you mean you used to listen to exactly the same thing over and over again?" © 1996 Brian Eno.

Using the pseudonym CSJ Bofop, 1996:

"Each of the twelve pieces on Generative Music 1 has a distinctive character. There are, of course, the ambient works ranging from the dark, almost mournful Densities III (complete with distant bells), to translucent Lysis (Tungsten). These are contrasted with pieces in dramatically different styles, such as Komarek with its hard edged, angular melodies, reminiscent of Schoenberg's early serial experiments, and Klee 42 whose simple polyphony is similar to that of the early Renaissance. But of course, the great beauty of Generative Music is that those pieces will never sound quite that way again." © 1996 Brian Eno.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl Variants

Depending on your platform (Windows or Mac), Noatikl comes in a variety of different variants to help you work the way you like to work.

Noatikl standalone is an application that comes in the following variants:

- Windows: standalone application
- Mac: standalone application (the recommended approach for Pro Tools!)

Noatikl plug-ins are audio plug-ins that work with your Digital Audio Workstation (*DAW*) tools; and come in the following variants:

- Windows: VSTi (for Cubase, Reaper, Ableton Live etc.)
- Windows: DXi/MFX MIDI Effect plug-in (for Sonar/Cakewalk)
- Windows: VST Module Architecture SDK MIDI Effect (MIDI Effect for Cubase etc.)
- Mac: VSTi (for Cubase, Ableton Live etc.)
- Mac: AU (Audio Unit - for Logic, GarageBand etc.)

What is the best approach to use - Standalone or Plug-in?

This is such an important question that it is answered in the next section of the user guide; click here for the full run-down!

MIDI Plug-in: MIDI Port Routing

Note that if you want to use the Audio Unit on Mac, or the standalone Noatikl application on Win/Mac, or possibly with the Noatikl VSTi (depending on your sequencer); you will need to redirect the output from Noatikl to:

- Windows: MIDI Yoke port; you can get MIDI Yoke from http://www.midiox.com/index.htm? http://www.midiox.com/myoke.htm
- Mac: IAC MIDI port; this is built-in to every Mac!

You must also assign one or more MIDI tracks to listen-out to the port to which you are directing your Noatikl output! Note that for quick testing on Windows, you might configure Noatikl to target directly your built-in Windows wavetable synth.

You'll find a lot of information on how to configure the various variants of Noatikl in the Noatikl FAQ

What Sequencers support interactive MIDI Plug-ins?

Steinberg (e.g. Cubase SE):

- supports pass-through of MIDI events generated from VSTis - see here for how to use Noatikl VSTi with Cubase!
- the Windows versions supports VST MIDI Effects, but not DXi/MFX
- see the tutorials for hints on how to get started!
- relevant Noatikl versions:
  - `noatikl.exe (standalone Windows)`
  - `noatikl.app (standalone Mac)`
  - `noatikl_VSTi_win.dll` and `noatikl_VSTi_mac.dll`
  - `noatikl_VST_MIDI_Effect_win.dll`

Cakewalk (e.g. Sonar, HomeStudio):

- supports pass-through of MIDI events generated from VSTis - see here for how to use Noatikl VSTi with Sonar!
- supports DXi/MFX, but not VST MIDI Effects

- there are some keyboard issues with Sonar 6...
- see the tutorials for hints on how to get started!
  - `noatikl.exe (standalone)`
  - `noatikl_VSTi_win.dll`
  - `noatikl_DXi_MFX_MIDI_Effect_win.dll`

Ableton Live:

- does not supports pass-through of MIDI events generated from VSTis - see here for how to use Noatikl VSTi with Ableton Live
- does not support MIDI effects
- relevant Noatikl versions:
  - `noatikl.exe (standalone Windows)`
  - `noatikl.app (standalone Mac)`
  - `noatikl_VSTi_win.dll` and `noatikl_VSTi_mac.dll`

Reaper:

- supports pass-through of MIDI events generated from VSTis
- does not support MIDI effects
- relevant Noatikl versions:
  - `noatikl_VSTi_win.dll`
  - `noatikl.exe (standalone Windows)`

Logic / Logic Express / GarageBand

- supports Audio Units
- you can also use Noatikl standalone with Logic
- see the tutorials for hints on how to get started!
- relevant Noatikl versions:
  - `noatikl_AU_mac.component`
  - `noatikl.app (standalone)`

Pro Tools

- supports RTAS
- we are not creating an RTAS plugin at this time, so recommend using Noatikl standalone with Pro Tools
- relevant Noatikl versions:
  - `noatikl.app (standalone)`

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Standalone or Plug-in?

What should I use - Noatikl Standalone or Noatikl Plug-in?

Noatikl standalone version

Use this where:

- you have a sequencer which does not support per-channel routing of MIDI data output by a Noatikl Plug-in variant.
- there is no Noatikl plugin variant for your sequencer (e.g. ProTools.)

Noatikl plugin

Use this where:

- you have a sequencer (such as Sonar 6 or Cubase) which supports per-channel routing of MIDI data output by a Noatikl Plug-in variant.
- you can set up your sequencer to use a Noatikl VSTi to control multiple synths, all from one instance of Noatikl (depending on how your sequencer works note timing accuracy might be dependent of the setting for the Noatikl timer resolutions). See Cubase VSTi tutorial or Sonar VSTi tutorial for examples.

Standalone

The standalone version of Noatikl runs as a tool on its own, but you can use it to interact with other tools in various ways. To use this version of Noatikl, simply run it up, attach it to a MIDI output device, and get started!

You can use it in this manner with any Windows (e.g. Sonar, Reaper, Cubase etc.) or Mac (Logic, Cubase, Pro Tools etc.) MIDI sequencer (also referred to as a DAW or *Digital Audio Workstation*) or standalone software synth or sampler.

We have created tutorials and template projects to "get you started", for the following popular sequencers:

- Logic 8 - template project zip
- Sonar 6 - template project zip
- Cubase SE 3 - template project zip

To use these files, simply download the file, extract the contents to the appropriate place on your computer, and follow the appropriate tutorial to see how the template project works!

Plug-ins

Challenge number one: Noatikl emits data for more than one MIDI channel

The most fundamental challenge with the Plug-in variants of Noatikl, is that Noatikl is quite unique in the music world; in that it *generates* MIDI data *across more than one MIDI channel*. Why does this make things difficult? Well, most MIDI Effect Plug-ins are designed to apply simple effects to single channels of MIDI data, such as echo or delay effects; or even to generate simple sequences that target a single MIDI channel (such as standalone bass lines or drum riffs).

However, if you were to assign a Noatikl Plug-in variant to a MIDI track in your sequencer, you would find that you could attach that MIDI track to render through only one softh synth (such as a VSTi synth). This is not good ... as what you *really* want to do is render each channel of data generated by Noatikl through a *different* synth! Some sequencers are not able to split output from MIDI Plug-ins attached to a MIDI track, on a per-MIDI channel basis, to target different synthesizers with data from different MIDI channels generated by noatikl!

That said, many sequencers do work well with the Noatikl VSTi, Sonar and Cubase being examples!

Challenge number two: the wide variety of plug-in formats

There is a huge range of plug-in formats, each of which behaves differently on different platforms. Noatikl is a MIDI-based generative music engine, and therefore needs to emit (and respond to) MIDI events. However, each sequencer requires a slightly different variant of plug-in!

If you wish to use Noatikl as a plug-in, then which plug-in variant to use, depends on what tool you are using,

and which platform you are running on (e.g. Mac OS X or Windows).

- **Cubase SE** – Noatikl can be deployed in Cubase (Mac/Windows) either as the Noatikl VSTi, or as a MIDI Effect written using the VST Module Architecture SDK (Windows only). The VSTi is the probably the best approach, click here for details! The output from this is fed-in to your favourite VSTi to render the MIDI events using the sounds you want. There might be some circumstances when you might instead consider using the MIDI Effect (Windows only).
- **Cakewalk Sonar/Home Studio** – Noatikl can be deployed in Sonar and Cakewalk either as the Noatikl VSTi, or the Noatikl DXi/MFX MIDI Effect. The VSTi is the best approach for Sonar 6 or later, click here for details! For earlier versions of Sonar or Cakewalk, you might consider using the MIDI Effect. In both cases, the output from the Noatikl plug-in this is fed-in to your favourite synthesizer to render the MIDI events generated by Noatikl using the exact sounds you want. Note that there can be some timing problems inherent in using the Noatikl VSTi with Sonar; see here for more details about how to work-around these issues.
- **Ableton Live** – Noatikl is deployed as a VST plug-in, click here for details!.
- **Reaper** – Noatikl is deployed as a VST plug-in, as Reaper does not support MIDI effects, but can instead route MIDI events generated from a VST plug-in into other VSTi synthesizers.
- **Logic** (for Mac) – Logic (which uses the Audio Unit Format, but which does not support VST plug-ins), simply doesn't support MIDI event generation by any Audio Unit plug-in... but we have documented a way to work-around this this. So you have two options to consider, depending on how you like to work: either use the Noatikl standalone version or the Noatikl AU version; follow the appropriate link to view the corresponding tutorial.

Challenge number three: not all hosts are equal!

Just to add to the challenge, not all hosts handle VSTi plug-ins correctly, and they all have their own ideosyncratic behaviours. So you'll probably need to experiment to find the Noatikl variant that works best with your favourite host. If in doubt, the standalone Noatikl version can usually be configured to work well with your sequencer; though the flexibility of the plug-in approach makes it well worth the effort in giving a plug-in variant of Noatikl a try too!

Cubase (Windows/Mac) and the Noatikl VSTi

Cubase works well with the Noatikl VSTi! The steps for using it with Cubase SE 3 are as follows (see also the tutorial!). These steps might also work (in slightly different form) with other Steinberg authoring tools; let us know if you find out!

Configuring the Noatikl VSTi the first time, for use with Cubase SE 3

When you've first copied the Noatikl VSTi to your computer, be sure to scan for the VSTi!

View larger image (51Kb)

Using the Noatikl VSTI within your Cubase SE project

- Step 1: Add the Noatikl VSTi to your project
  - To add the Noatikl VSTi to your project, launch the *VST Instruments* window by selecting *Devices -> VST Instruments* from the menu
  - Insert the Noatikl VSTi by clicking the top area in the first VST instrument slot; select the Noatikl VSTi plug-in (*noatikl_VSTi_win* or *noatikl_VSTi_mac*).
  - Load-up a Noatikl file (or create a new one) within noatikl. That will give you something to hear in a moment. Make sure you set the MIDI Channel parameter for each voice, so you can be sure which software synth you want to direct it to.

- Step 2: Add some Software Synths!

- Return to the *VST Instruments* window.

- Add as many software synths as you want to your Project; one for every distinct sound you want rendered by your Noatikl piece. In the above screen shot, you'll see three soft synths that have been set-up in the *VST Instruments* window, listed after the Noatikl VSTi plug-in.

- Step 3. Prepare your MIDI tracks.
  - Create one MIDI track per Noatikl MIDI channel that you're interested in. For **every** such track:
    - Select the MIDI Input ("*in")* as the appropriate Noatikl MIDI output (for example, *noatikl_VSTi_win* or *noatikl_VSTi_mac*).
    - To ensure that your MIDI track listens-out to only one of the MIDI channels output by Noatikl, select the *Input Transformer* button, and select the *Local* entry in the pop-up list that appears. In the presets drop box, select *Channel Filtering* followed by the specific channel that you want to listen to. Finally, set the *Active Module 1* checkbox, and close the window. In case you find these instructions confusing, you should find that they are made clear in the video tutorial!
    - Set the MIDI Output ("*out*") to be the appropriate software synth that you added earlier.
    - You **must** make sure that *Record Enable* is turned-on (the red cicular buttons in the above screen shot)!
  - Save your project before you forget!

- Step 4: play and enjoy!
  - Start playback - enjoy what you hear! Note that of course if you have too many software synths in your project, or are running on a slow computer, you might hear some audio break-up.

Sonar and the Noatikl VSTi

Good news: Sonar 6 (and we expect Sonar 7) works well with the Noatikl VSTi (albeit with some keyboard issues due to a bug in Sonar...!)! There can also be some timing problems inherent in using the Noatikl VSTi with Sonar; see here for more details about how to work-around these issues.

The steps for using Noatikl VSTi under Sonar Home Studio (which is similar to Sonar Studio/Producer) are as follows (see also the tutorial!). These steps might also work (in slightly different form) with earlier versions of Sonar; let us know if you find out!

Configuring the Noatikl VSTi the first time, for use with Sonar

When you've first copied the Noatikl VSTi to your computer, be sure to scan for the VSTi. On my machine, this involves the following steps:

- Select Tools -> Cakewalk Plug-in Manager from the Menu
- In the VST Configuration section, click on the Options button. Add to the list of scan paths, the folder to which you copied the `noatikl_VSTi_win.dll` file (on my machine, this is `C:\Program Files\Steinberg\Cubase SE 3\Vstplugins`, but could also be something like `C:\Program Files\Vstplugins`) .
- Press the Scan VST Plug-ins button.
- Noatikl VSTi_win should now appear in the VST Instruments (VSTi) Plug-in Category.

View larger image (91Kb)

Using the Noatikl VSTI within your Sonar Project

- Step 1: Add the Noatikl VSTi to your project
  - To add the Noatikl VSTi to your project, do not start by using the synth rack; it will crash Sonar!
  - Insert the Noatikl VSTi by right-clicking the FX bin of an **audio** track; select the Noatikl VSTi plug-in.
  - Load-up a Noatikl file (or create a new one) within noatikl. That will give you something to hear in a moment. Make sure you set the MIDI Channel parameter for each voice, so you can be sure which software synth you want to direct it to.
  - Next, open-up the Synth Rack (*Views -> Synth Rack*)
  - Right-click on the far-right pane of the Noatikl VSTi, and click on "Enable MIDI Output". This lets Sonar use the MIDI output data emitted from the VSTi by noatikl. If you forget this step, Noatikl will remain silent!
  - Close the Synth Rack.

- Step 2: Add some Software Synths!
  - Add as many software synths as you want to your Sonar Project; one for every distinct sound you want rendered by your Noatikl piece. Each software synth is on a separate audio track, of course.

- Step 3. Prepare your MIDI tracks.
  - Create one MIDI track per Noatikl MIDI channel that you're interested in. For **every** such track:
    - You **must** make sure that *Input Echo* is turned-on! Note: you can see weird behaviour with Sonar, where changing one MIDI track's Input Echo status can reset the Input Echo status of another track (!); watch-out for this! The way to avoid this is first to select the *Options -> Global -> General* menu item, and ensure that the *"Always Echo Current MIDI Track"* item is **not checked** (the default behaviour is that this *is* checked, which is not what you want!).
    - Select the MIDI Input as the appropriate Noatikl MIDI output (for example, noatikl_VSTi_in 1-> MIDI Ch.1). Set a separate Noatikl MIDI output channel for each MIDI track.
    - Set the output device as the appropriate software synth that you added earlier.
  - We have found that if you should set-up at least some MIDI data in at least one track (such as dummy MIDI controller events), otherwise the Noatikl data will be ignored! You probably also want to set a loop start and end point (presumably, choosing a loop size equal to or greater than your Noatikl Piece Length).
  - Save your project before you forget!

- Step 4: play and enjoy!
  - Start playback - enjoy what you hear! Note that of course if you have too many software synths in your project, or are running on a slow computer, you might hear some audio break-up.

How to use the Noatikl DXi/MFX MIDI Effect Plug-in with Sonar/Cakewalk

Use the following steps to get the Noatikl DXI/MFX MIDI effect working in Sonar/Cakewalk:

1. Start Sonar
2. Set-up a MIDI track
3. View in "Track" View
4. Attach the Noatikl DXi/MFX MIDI Effect to the effect bin, by right clicking, and selecting MIDI Effects -> *Noatikl MIDI Effect*
5. Click on the *Noatikl MIDI Effect* item you'll now see in the effect bin!

How to use the Noatikl AU (Audio Unit) Plug-in with Logic

To use Noatikl AU with Logic is very easy; simply download the template project zip and follow the tutorial!

How to use the Noatikl AU (Audio Unit) Plug-in with GarageBand / AU Lab

GarageBand

- Before starting GarageBand on your Mac, you *must* first start MIDI Pipe and "*hijack*" the IAC port that Noatikl will use. Check out the GarageBand tutorial to see exacly how to use Noatikl with GarageBand do this!
- You first need to show details in the track (the right pointing arrow below the instrument icon at the bottom of the right hand side track info panel).
- That expands and you can click on one of the "effects" comb boxes showing "None" to select "*Noatikl AU* " (this is at the bottom of list that pops-up).
- Click the "pencil" button to invoke the Noatikl user interface.

AU Lab (find this app using the spotlight search tool on your mac!)

- Click in the "Master Out" panel on Effects then "intermorphic" then "*Noatikl AU*".
- To play back a file from AU Lab:
  - Choose Edit > Add Audio Unit Generator. You must then choose e.g. "AUAudioFilePlayer" as the generator.
  - Add an audio file
  - Press play and hear Noatikl do its thing (assuming you have specified the MIDI Output device targeted by Noatikl, and assuming that you are using e.g. Midi Pipe to render the audio!).

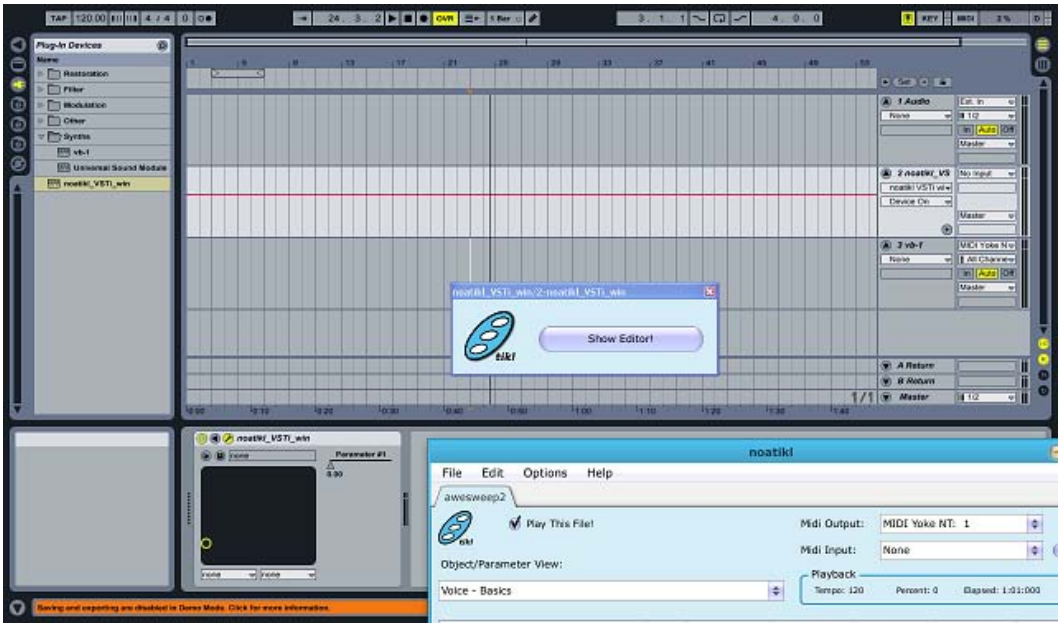
Ableton Live and the Noatikl VSTi

The steps for using Noatikl VSTi under Ableton Live are as follows; these steps apply to both the Windows and Mac versions of Ableton Live.

Configuring the Noatikl VSTi the first time, for use with Ableton Live

When you've first copied the Noatikl VSTi to your computer, be sure to scan for the VSTi from Ableton. Once done, you should see *noatikl_VSTi_win* (Windows) or *noatikl_VSTi_mac* (Mac) in Ableton's list of "Plug-In Devices".

Using the Noatikl VSTI within your Ableton Live Project



View larger image (113Kb)

- Step 1: Add the Noatikl VSTi to your project
  - Add the Noatikl VSTi by selecting the plug-ins view, and dragging *noatikl_VSTi_win* (Windows) or *noatikl_VSTi_mac* (Mac) from Ableton's list of "Plug-In Devices" on to a MIDI track.
  - Show the input/output options for your tracks; set the track with Noatikl on it, such that input is set to "No Input".
  - Show the Noatikl editor window, and set-up a file to play (e.g. create a couple of voices).
  - Set the Noatikl MIDI Output to be a spare MIDI port; such as MIDI Yoke NT 1 (Windows) or IAC Port 1 (Mac); you can then hide Noatikl if you want to clear some screen space.
- Step 2: Add some Software Synths!
  - Create another MIDI track.
  - Drag a VSTi synth onto that track.
  - Make that track's MIDI Input to be the MIDI Port to which Noatikl is sending its data; such as MIDI Yoke NT 1 (Windows) or IAC Port 1 (Mac) in this example.
- Step 3: play and enjoy!
  - Start it all playing!
  - If you can't hear anything, make sure that the green "on" button was set in the bottom panel in Ableton where Noatikl is shown. If this isn't enabled, you won't see Noatikl flash away when playing notes; so that is a clue to if you need to press this button or not!

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Installation

Now that you have all that background behind you, the next thing to do is to install Noatikl!

Windows

What's included in the downloaded zip file (noatikl_win32.zip):

`noatikl_win32_setup.exe` that installs:

- **noatikl.exe (standalone)**
- **noatikl_VSTi_win.dll**
- **noatikl_VST_MIDI_Effect_win.dll**

Installing Noatikl on Windows

1. Save the noatikl_win32.zip file to your Windows desktop.
2. Use Windows Explorer to look inside the file (or use a tool like WinZip).
3. Windows: From zip file above, extract and run `noatikl_win32_setup.exe` to install the files listed above.
4. You may be notified that Setup/Uninstall is trying to launch e.g. C:\Windows\system32\regsvr32.exe. The installer needs to do this to register the DXi plugin.
5. If you want to use the Noatikl VSTi, you might need to tell your Sequencer program to scan the folder *c:/Program Files/VstPlugins* (or equivalent on your machine); this is where the Noatikl installer puts the Noatikl VSTi file, which is called *noatikl_VSTi_win.dll*).

Configuring your Windows computer so that Noatikl can route its MIDI data

1. Install and enable MIDI Yoke (or other similar software MIDI router such as Maple)
   - Download MIDI Yoke from http://www.midiox.com/myoke.htm
   - Install the software!
2. Prove it all works (e.g. in Cubase)
   - Create a MIDI project
   - On MIDI track 1 enter some notes with piano roll editor
   - Set "in:" to "Not Connected"
   - Set "out:" to one of the virtual MIDI Yoke ports.
   - On MIDI track 2 set "in:" to the same MIDI Yoke port.
   - Set "out:" to be your favourite VST Synth or sampler, e.g. "The Grand SE"
   - Press transport Play, and you should hear your MIDI events being rendered through a soft synth where your MIDI events have been routed from one track to the other through the MIDI Yoke driver.

Mac OS X

What's included in the downloaded zip file (noatikl_mac_dmg.zip):

`Intermorphic - noatikl.dmg` that installs:

- **noatikl.app (standalone)**
- **noatikl_AU_mac.component**
- **noatikl_VSTi_mac.vst**

Installing Noatikl on Mac OS X

1. Note: If you have any old copies of the Noatikl plugin file, perhaps from early days of Noatikl before we had an installer, do please remember to remove those by hand first!
2. Save the noatikl_mac_dmg.zip file to your Mac desktop

3. Double click on it, creating a `Intermorphic - noatikl.dmg` icon.
4. Double click on the icon; this opens a folder containing an icon for the Noatikl installer – run that file to install the files listed above.
5. You'll find the Noatikl application in your Applications folder under Finder; if you want to make it as easy to find as possible, you can drag it direct to your Dock for future easy access!

Configuring your Mac so that Noatikl can route its MIDI data

Use the following steps, which are also shown in one of the video tutorials:

1. Enable your IAC driver
   - Click on the blue magnifying class (spotlight) icon on the top right of your Mac screen
   - Type "audio midi setup"
   - Click on the Audio MIDI Setup entry you will see listed under applications. This will run the audio midi setup application. Note: more advanced Mac users could use Finder to run this application directly, you will find it under Applications / Utilities Audio MIDI Setup
   - Click on MIDI Devices tab
   - Click on IAC Driver
   - Click on Show Info icon
   - Ensure that "Device is online" is ticked, and press Apply
2. We **strongly** recommend that you download and use the excellent MidiPipe by SubtleSoft, which you can get from http://homepage.mac.com/nicowald/SubtleSoft/ ... a very useful piece of software, which most of our Mac tutorials rely on!

Sequencer-specific Plug-in notes

How to use the Noatikl VSTi in Sonar

Sonar users should see here for information on how to use the Noatikl VSTi in Sonar!

How to use the Noatikl Dxi/MFX MIDI Effect in Sonar/Cakewalk

See here for information on how to use the Noatikl DXi/MFX MIDI Effect in Sonar/Cakewalk!

How to use the Noatikl Audio Unit (AU) plug-in in Logic

Mac Logic users should see here for information on how to use Noatikl AU in Logic!

How to use the Noatikl Audio Unit (AU) plug-in in GarageBand

Mac GarageBand users should see here for information on how to use Noatikl AU in GarageBand!

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Standalone with Logic on the Mac

Logic sums all incoming midi data to the current active track in the arrange window because this is normally what you might want. However, it isn't what you want with Noatikl and the Logic manual isn't clear as to how to change this.

Using Noatikl stand-alone application with Logic Pro:

1. Enable IAC bus and assign this as the Noatikl output
2. Launch Logic
3. Select first track in the Arrange window and create an Audio Instrument track.
4. Insert instrument plug-in of choice.
5. Set Midi channel of this instrument to 1
6. Arm for recording
7. Repeat for up to fifteen other tracks - assigning each one to a different midi channel
8. Go to File -> Song Settings -> Recordings
9. Check the box marked "Auto demix by channel if multitrack recording"

This now works correctly i.e. setting Noatikl to transmit on c1 directs the data to the first instrument in Logic, transmitting on c2 goes to the second etc.

You are now ready to make amazing music with Noatikl and Logic on your Mac!

Noatikl, Logic and MIDI feedback

From v 1.5 Noatikl now listens to incoming midi data to allow you to create dynamic hyperinstruments.

In order to use this feature in the standalone version of Noatikl, Logic users need to take certain steps to eliminate the risk of MIDI feedback.

Due to the way Logic implements it's own midi through system, it is essential for Noatikl to use different midi ports for midi in and midi out. Port sharing will cause midi feedback, lockups and the risk of system crashes.

Instructions for using the IAC driver.

Open Audio/Midi setup in the Applications/Utilities folder. Select the midi tab

Double click on the IAC icon and select *Ports* Click on the *Add port* button underneath the ports window. Accept the default setting by clicking *Apply*.

In Noatikl select IAC bus 1 as the Midi output bus and IAC bus 2 as the Midi input bus

Launch Logic.

Open the Environment window (*Command 8*) and select the Clicks and Ports layer.

Click on *New -> Monitor* to create a new Monitor object. Draw a cable from the Midi input object from IAC Bus 2 and connect it to this monitor. Do not connect the output of this monitor to any other object in the environment.

In the arrange page, select *"Settings -> Synchronisation"*. Click on the midi tab. Set the midi clock destination tab to be IAC Bus 2

And you are good to go.

Instructions for MidiPipe

To create a second midipipe port, drag another midi in and midi out into your current pipe.

Select the second midi input and click on the blue menu display tab to show all the midi inputs. Click on *Edit virtual Inputs*

Click on *New Virtual Input* to add the second input, repeat this in the Virtual Outputs view to create a new virtual output. Click *Done*.

Highlight the first Midi output in the pipe. Ensure the box marked *Pass Through* is unchecked (this box is checked by default).

Save your Pipe.

In Noatikl select Midi Pipe input 1 as the output and Midipipe Output 2 as the input.

Instructions for Logic

Launch Logic.

Open the Environment window (*Command 8*) and select the Clicks and Ports layer.

Click on *New -> Monitor* to create a new Monitor object. Draw a cable from the Midi input object from MidiPipe Output 2 and connect it to this monitor. Do not connect the output of this monitor to any other object in the environment.

In the arrange page, select *Settings -> Synchronisation*. Click on the midi tab. Set the midi clock destination tab to be MidiPipe Input 2

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

I Get Silence, Help!

Windows users:

Only one component at a time can render through the Windows Software Synth. If your Noatikl component is specified to render through this and you hear silence, make sure there is not something else trying to render through it.

MAC users:

If you selected an IAC driver bus as your output device, make sure you have an active rendered out for that device e.g. MIDI Pipe or once of your sequencer tracks with an attached synthesiser or sampler plug-in.

All platforms:

Don't forget to press play! Make sure you haven't muted all voices, or set their volumes or velocities all to zero !If you are using fixed-pattern voices, make sure they don't all have empty patterns!

Sonar

You may hear silence from Noatikl when using the *Noatikl VSTi* or *Noatikl MIDI Effect* with Sonar.

- The solution to this problem is to modify your MIDI track to have a continuous stream of dummy MIDI controller events! This is required for Sonar to keep “pumping” Noatikl for more MIDI event data!

If using the Noatikl VSTi and your hear silence, make sure you have added Noatikl VSTi to your project properly!

- To add the Noatikl VSTi to your project, do not start by using the synth rack; it will crash Sonar
- Insert the Noatikl VSTi by right-clicking the FX bin of an audio track; select the Noatikl VSTi plug-in.
- Load-up a Noatikl file (or create a new one) within noatikl. That will give you something to hear in a moment. Make sure you set the MIDI Channel parameter for each voice, so you can be sure which software synth you want to direct it to.
- Next, open-up the Synth Rack (Views -> Synth Rack)
- Right-click on the far-right pane of the Noatikl VSTi, and click on "Enable MIDI Output". This lets Sonar use the MIDI output data emitted from the VSTi by noatikl. If you forget this step, Noatikl will remain silent!
- Close the Synth Rack.

Cubase

If *Noatikl VSTi* is silent, make sure you are not running with the “Sequencer MIDI Pipeline” selected. Not all sequencers support this feature!

- The solution is to select one of the other MIDI output ports instead (and make sure you have a synthesiser listening out for that port!).

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Now What?

Once you have installed Noatikl, you can choose one of the product variants and get creating some music. The easiest variant to get started with is the standalone version.

You should now get on with following the tutorials!.

Want some more ideas? Try-out the Noatikl on-line forum at:

http://forum.intermorphic.com/index.php?c=6

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Registration

When you first run noatikl you'll get reminders to enter your product key. After using Noatikl for a number of days, the evaluation period will expire. From that point onwards, you'll have to buy a key from intermorphic in order to continue using noatikl. Once you've received your Noatikl product key, you must select Help -> Noatikl Product Key, and enter your details into the three fields. Press the appropriate key to check and confirm your details! When you buy your key, you might be sent a “keep you going” key that you can enter immediately while we check your details. This key will keep Noatikl running for a few more days. Be sure to enter the full product key when you receive it, in order to keep using noatikl.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

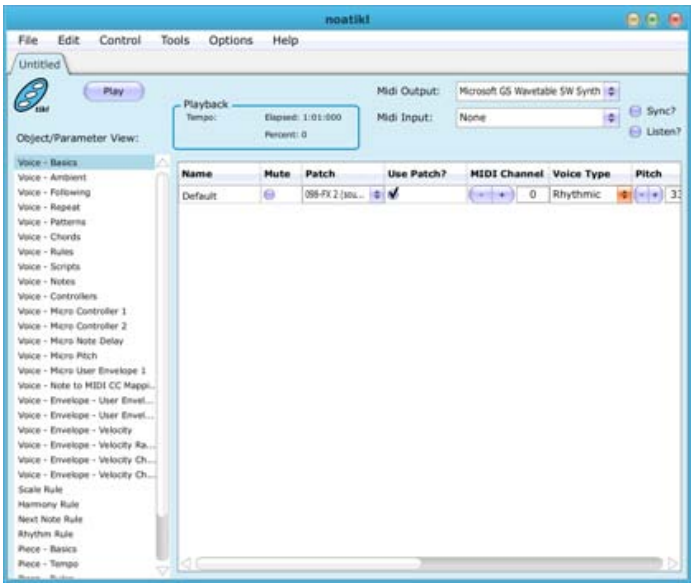
back | next

The User Interface

Creating your first file

▶From the Noatikl menu, select File -> New.

This presents you with a window, containing a table showing a single row which shows you a default Voice in your new Noatikl composition. It will look something like this!



Objects and Parameters

There are various object types that you can edit with Noatikl, including Voices. You can view these by selecting an item from the list down the left-hand side of the window.

You can add Voices (or other musical objects) within the table, so that each row represents a different Voice, and each column represents a different attribute of that Voice.

There are many parameters (attributes) for all objects, including Voices, and you will see that for some objects there are many available views of data!

▶You can press the Play button if you want to hear the default piece play! Note: you might first need to set-up a MIDI device through which to render your music. See the Noatikl FAQ for more info...

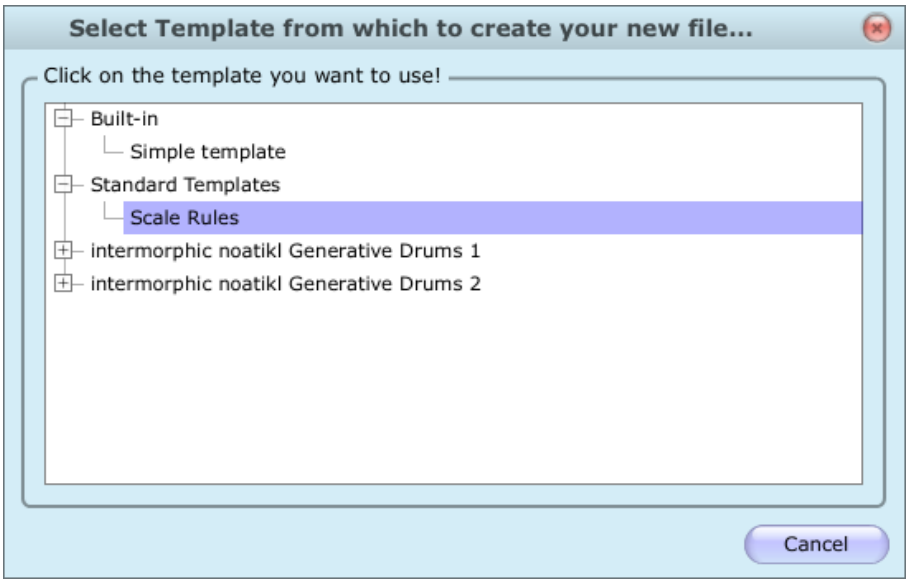
File Window

The File Window contains Play, Stop and other buttons and combo boxes. It also contains a table where you can view and edit the objects that make-up your Noatikl composition.

You can add Voices (or other musical objects) within the table, so that each row represents a different Voice, and each column represents a different attribute of that Voice. Use the menu items to add, delete and copy voices and other objects.

There are many attributes for Voices and other objects, and you can choose to display related attributes/parameters of any object type. You choose the set of Objects and Parameter attributes to look at via the Object/Parameter View list box.

- *New* - creates a new file for editing; select the template file that you want to use as the basis for your new file, via the dialog that is presented to you. Which templates get listed, depends on which template packs you have got installed on your computer. You can get new template packs from the Noatikl store!



- *Open...* – browse your computer for a file to open.
- *Close* – close the currently open file .
- *Save* – save your file with its current name in its current folder location.
- *Save As...* – save your file with a new name or folder location.
- *Recent Files* – open from a list of your recently accessed Noatikl files
- *Quit* – quit noatikl.

Tip: Making/saving your own templates

Create and save the noatikl file(s) you want to be template(s); then zip those files up and put the zip file here (where <drive> is the disk on which you have installed Noatikl):

- Windows
  - <drive>:\Documents and Settings\All Users\Application Data\intermorphic\noatikl\templates
- Mac
  - <drive>:Library\Application Support\intermorphic\noatikl\templates

You can also get new template packs from the Noatikl store!

Edit menu

Use the *Edit* menu items to copy and paste object parameters. The menu items available under *Edit* are as follows:

- *Add Voice* – adds-in a new default object of the currently viewed type
- *Cut Voice* – cuts the currently focused object
- *Copy Voice* – copies the currently focused object
- *Copy Parameter* – copies the currenly focused parameter
- *Paste Voice* – pastes in the most recently copied or cut object or parameter
- *Delete Voice* – deletes the currently focused object
- *Default Parameter* – restores the currently focused parameter to its default value
- *Default Column* – restores all parameters in the currently focused column to their defaults
- *Default Row* – restores all parameters in the currently focused row to their defaults
- *Merge-In Template*– from any installed templates, browse for a template that you want to merge-in to your current piece; Noatikl does the "right thing", pulling in all voices and rules from that template (it does *not* replace any rules, but will bring-in rules that do not currently exist in your piece); copies-in all voices and renames each voice if required to keep them unique. Try merging in one or more drum templates!
- *Merge-In File*– as for Merge-In Template, but browse for a Noatikl file that you want to merge-in to your current piece. Try taking any piece, merging-in any other piece, and amaze yourself at what you hear!

Control menu

Use the *Control* menu items to play tracks and solo and mute voices. The menu items available under *Control* are

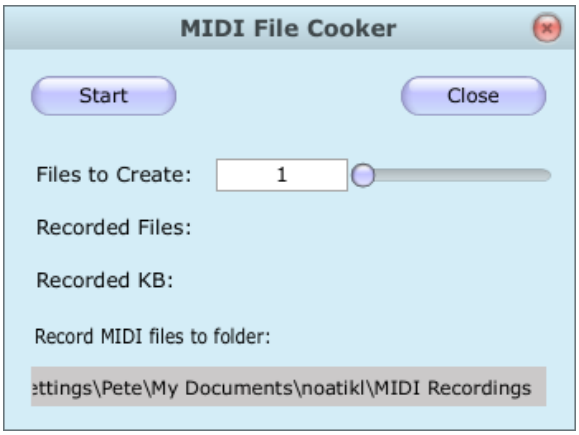
as follows:

- *Play / Stop* – starts or stops the piece playing (you can also use the Ctrl+Spacebar shortcut)
- *Solo Voice* – solos the selected voice
- *Unmute All Voices* – unmutes all voices
- *Mute All Voices* – mutes all voices

Tools menu

Use the *Tools* menu items to access integrated helper utilities. The menu items currently available under *Tools* are as follows:

- *MIDI File Cooker* – launches a utility to capture (as fast they can be generated!) a user configurable number of takes to sequentially numbered MIDI files.



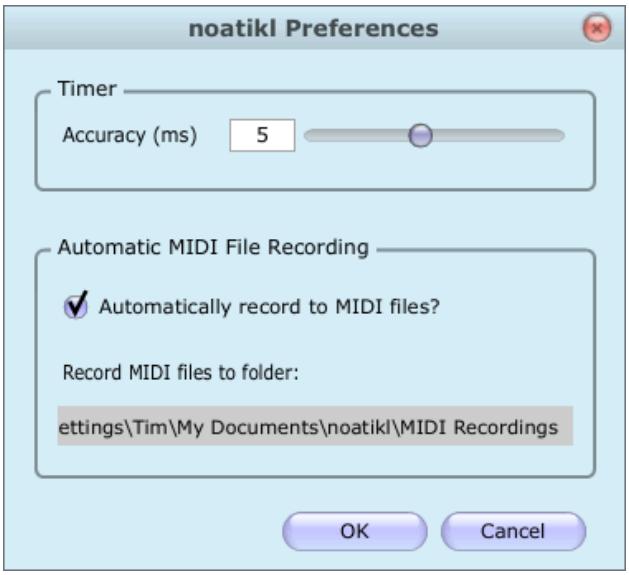
  - Start/Stop – starts off/stops the "cooking" process (make sure you have a Noatikl file loaded, and have set the number of "Files to Create" and "Record MIDI files to folder" below).
  - Close – closes the utility.
  - Files to Create – use the slider to select a number from 1–100 (or double click the number field to enter a number).
  - Double click the field "Record MIDI files to folder" and enter the path you require.
  - Recorded Files – tells you how many MIDI files have just been recorded.
  - Recorded – tells you the total size of the MIDI files that have just been cooked.

  Important note: when cooking a MIDI file, the tempo comes from the Piece Tempo parameters; not from any host sequencer!
- *Noatikl Script Window* – displays the Noatikl script window, the use of which will be extended and documented in further Noatikl updates (as appropriate).

Options menu

Use the *Options* menu to select various options which govern the way that Noatikl works. These include the following:

- *Noatikl Preferences...* –



- adjust the timer resolution used by Noatikl when Noatikl interacts with MIDI output devices (such as IAC ports or MIDI Yoke ports). The default timer resolution is 10 milliseconds, and you can reduce this all the way down to 1 millisecond if your machine is fast enough and if you need that sort of accuracy.
- enable automatic recording (when played) of your Noatikl piece to a MIDI file - select the tick box to enable this, and double click the field "Record MIDI files to folder" to enter the path you require.
  - Note: You wouldn't want to set the auto-record toggle button if you were using Noatikl for an installation!
  - Note: when auto-capturing a recording through this preferences setting, where the tempo comes from depends on what configuration you are currently using:
    - Noatikl Plug-in: the tempo comes from the host
    - Noatikl Standalone synced to host:, the tempo comes from the host
    - Noatikl Standalone unsynced: Noatikl provides its own tempo from the from the Piece Tempo parameters

- *Reset Column Widths* - resets the column widths for every parameter, to their default values.
- *Use Tabbed Windows* - use tabbed windows for each Noatikl piece.
- *Use Floating Windows* - use a floating window for each Noatikl piece.
- *Use Native Window Title Bar* - changes the look of the title bar between a more Mac or Windows feel.

Play

The Play button starts you playing. You only see this button when the file is stopped! You will only see this button for Noatikl standalone; for the plug-in variants of Noatikl, play/stop is controlled directly by your DAW's transport controls.

Note: you might first need to set-up a MIDI device through which to render your music. See the Noatikl FAQ for more info...

Note: if you have *Sync?* selected, then playback won't start making any sounds until the sequencer that you are listening to has the play button pressed on its own transport control. In this case, you must first press *Play* before Noatikl can respond to events from your sequencer's transport control!

When the piece is playing, you will see various interesting properties get displayed in the Playback group:

- *Tempo* - this shows the current Piece tempo. If you are using one of the Noatikl Plug-in variants, then this value will be the value that is forced by the sequencer host in which Noatikl is running.
- *Percent* - the approximate elapsed Noatikl Piece position, relative to the Piece Duration.
- *Elapsed* - the elapsed time in Measures:Beats:Ticks format (MBT).Note that if you are using of of the Noatikl Plug-in variants, then this value might be up to a bar ahead of the value displayed by your sequencer's transport control; this is because Noatikl composes ahead according to a latency dictated by the behaviour of your host sequencer.

Stop

The Stop button stops you playing. You only see this button when the file is playing! You will only see this button for Noatikl standalone; for the plug-in variants of Noatikl, play/stop is controlled directly by your DAW's transport controls.

MIDI Output

- Whichever variant of Noatikl you use, you will need to specify the MIDI output device to which Noatikl sends its MIDI data.
- Standalone:
  - Windows - use a MIDI output device; use MIDI Yoke if you want to route data from Noatikl to your DAW!

- Mac: select an IAC port, and maybe use MIDI Pipe by SubtleSoft to render your audio data!

- Plugin: In general, use the "Sequencer MIDI Pipeline" option.
  - Audio Unit (AU): See listed IAC Driver Bus 1 (this assumes you have configured the IAC as outlined above) and perhaps some other devices.
  - VST:
    - Mac: See "Sequencer MIDI Pipeline" followed by Mac - IAC Driver Bus 1
    - Windows: Use Microsoft GS Wavetable SW Synth or perhaps some other MIDI output devices such as MIDI Yoke channels.
  - MIDI Plug-in: both MIDI effect plug-ins list the same options as you get for the VST. You probably want to select the Sequencer MIDI pipeline option though you can use one of the other options should you so wish!

If you are running within Cubase SE 3 or other sequencers which do not support MIDI routing from one VST to another then you **must** select any value **other** than "Sequencer MIDI Pipeline" (otherwise you will just hear silence!).

If you are using Reaper, or some other VST Host which allows MIDI data to be routed from one VST into another one, then you may select any of the available options though you are most likely to want to select the "Sequencer MIDI Pipeline" option.

MIDI Input

You can tell Noatikl to listen-out for incoming MIDI data on the specified port. Noatikl can use incoming MIDI information for various purposes, including synchronisation and we even hope that one day you'll be able to operate Noatikl as a hyper-instrument through Listening Voices!

- *Sync?* - if this is selected, then playback won't start making any sounds until the sequencer that you are listening to has the play button pressed on its own transport control. In this case, you must first press *Play* before Noatikl can respond to events from your sequencer's transport control!

- *Listen?* - This is provided to protect against MIDI feedback, which can occur with some sequencers, noatably Logic (which will by default present all data output to a port, as an input to all other ports!)! If *Listen?* is ticked *or* if Noatikl is using the Plug-in internal sequencer pipeline, then:
  - MIDI events from the MIDI input will be passed-through to noatikl
  - the MIDI Input trigger scripts will respond to MIDI input
  - listening voices will compose where appropriate
  - all note on/off events presented at the input will reach the output.
  - Noatikl will try to harmonize with incoming MIDI note data.

  If this is *not* selected and if Noatikl is not using the internal sequencer pipeline, then:
  - MIDI events from the MIDI input will *not* be passed-through to noatikl
  - the MIDI Input trigger scripts will not respond
  - listening voices will not compose
  - the **only** events that will be passed through to Noatikl will be *MIDI Sync* events
  - Noatikl will **not** try to harmonize with incoming MIDI note data.

Noatikl 1.5, Logic and midi feedback

From v 1.5 Noatikl now listens to incoming midi data to allow you to create dynamic hyperinstruments.

In order to use this feature in the standalone version of Noatikl, Logic users are advised to take the following steps to eliminate the risk of midi feedback!

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Sync

You can use this feature to allow Noatikl (when playing through an external MIDI device or port) to synchronise with audio playback in your favourite sequencer.

In this case, set your Noatikl MIDI input device to listen-out to the MIDI port to which you have told your sequencer to send MIDI sync information.

Then, for:

- Noatikl standalone: press the Noatikl Play button
- Noatikl plug-in: press the play button in your transport control.

Noatikl will then start and stop playing automatically in response to your pressing the play and stop buttons in the sequencer which emits the MIDI sync information!

Note: you will experience a degree of MIDI latency (i.e. all notes offset by a certain amount) and some small variation in the note playback timing (typically +/- 5 or 10ms). We are looking at ways to minimise this in later updates. If you record the Noatikl output to a MIDI file, then it is fairly easy to adjust for the latency and also quantise (if you want) the note timing.

MIDI Sync using Logic

Using Logic Pro on the Mac, you can use MidiPipe to provide the "cabling" between Noatikl and Logic.

Set-up couldn't be easier.

Set noatikl's in and out ports to MidiPipe, engage the Sync? option and hit play in noatikl.

Open Logic, set the sync option to send midi clock to the MidiPipe port and that's it really. Once you've done that, the Logic transport controls work both Noatikl and Logic - hit play in Logic and Noatikl starts composing and feeding the tracks in Logic with midi data. What you do with it then is up to you. Control the tempo of the piece live in real-time from Logic's transport. Or, if you have them, you can use a hardware transport controller!

Note for Logic 8 Users

To use MIDI sync in Noatikl AU for Logic 8:

1. Just connect-up IAC1 in the click and port layer...! i.e. remove the SUM option... if you don't use the SUM input you need to connect input ports on a port by port basis. Needs good eyesight and a steady hand if you have a lot of ports :-)
2. project MIDI sync settings to send to IAC port 2.
3. noatil file: set output IAC1, output IAC 2, sync = checked.
4. Should then work OK!

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

Object Types

The object types supported by Noatikl, and how they are used, is as follows.

Voice: The Voice is probably the most important Object Type in noatikl. A Voice is a Noatikl object that represents a generator of Music data. For example, a Voice can be used to play a fixed musical pattern. Or, a voice can be used to create notes according to various musical rules. A voice can also be used to emit MIDI controller information that is used to control your favourite sequencer. You must have at least one Voice in your Noatikl composition. You may have many more if you want! The basic behaviour of a Voice is defined by its Voice Type property. You may tell Noatikl that individual Voices use specific Scale Rules (other any other of the supported Rules).

Piece: There is always exactly one Piece object in your Noatikl composition. The Piece object manages various high-level properties of your composition. These include things like what musical rules to follow by default (where not overridden by individual Voices), the Tempo to follow, and various other global properties.

Scale Rule: The Scale Rule defines the musical scale that may be used when composing a voice. You may create as many Scale Rules as you want. You typically give a Scale Rule a name that tells you what it does (such as Major or Minor). A Scale Rule is defined graphically, using a system where you show what notes are available within a scale, and their relative probabilities of being chosen when all other things are equal.

Harmony Rule: The Harmony Rule defines the musical scale that may be used when composing a voice. You may create as many Harmony Rules as you want. A Harmony Rule is defined graphically, using a system where you show how likely a note for that Voice is to be chosen, if it has to harmonize with another note at a given musical interval in semitones.

Next Note Rule: The Next Note Rule defines the distances that are to be used between each note composed for a Voice. You may create as many Next Note Rules as you want. A Next Note Rule is defined graphically, using a system where you show how likely a note for that Voice is to be chosen, given that it might be a specific musical interval away from the last note in semitones.

Rhythm Rule: The Rhythm Rule defines the musical scale that may be used when composing a voice. You may create as many Rhythm Rules as you want. A Rhythm Rule is defined graphically, using a system where you show relatively how likely the duration of a note for that Voice is likely to be. This rule is combined with other factors, including the remaining length of time a Voice has left in the current bar (Noatikl tries to avoid having notes from non-Ambient voices drifting across bar boundaries!).

File: You can store certain basic bookkeeping information related to your file, such as the title of the composition and the author. You do this with the File object. The File object doesn't define any data that affects what you hear in your composition.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Tutorials

The tutorials for Noatikl are mainly in video form. We will add specific links to videos in this page as they become available. You will also find links to tutorial information in the Noatikl forum, which is at http://forum.intermorphic.com/index.php?c=6

There are various tutorials and videos demonstrating Noatikl in the following locations:

- Intermorphic high-definition video channel http://www.vimeo.com/intermorphic
- Mark Harrop (also featuring some great audio content!)
  - High Definition: http://www.vimeo.com/umcorps/
  - Lower Definition: http://uk.youtube.com/user/UncertainMusicCorps

Pete Cole:

- High Definition: http://www.vimeo.com/kantudok/
- Lower Definition: http://www.youtube.com/kantudok

Using the video tutorials

Simply click on the link (high or lower definition) to hear the video play!

Where possible, we have provided both High Definition (HD) videos via Vimeo, and lower-quality videos via YouTube for those of you with older machines or slower connection speeds.

- If you have a modern, fast computer, with a fast internet connection, then we highly recommend you view the high definition (HD) videos from Vimeo! **Tip**: you can click on the "FULL" button on the HD videos to have them run at full screen!
- if you have a slower machine, or a slow internet connection, you might want to use the lower definition videos from YouTube.

Note! The videos show an older evolution of the Noatikl user interface, where the parameter view selection was via a small combo box rather than what we now use (which is a the list of all available views on the left-hand side of the file window)!

Noatikl tutorial - creating your first piece on Windows

by Pete Cole

Creating your first simple MIDI piece with Noatikl standalone for Windows.

Play video (High Definition)

Play video (lower definition)

Noatikl tutorial - creating your first piece on Mac

by Pete Cole

Creating your first simple MIDI piece with Noatikl standalone for Mac.

Play video (High Definition)

Play video (lower definition)

Written Tutorial - getting started with Noatikl Standalone

In this tutorial will create a simple Noatikl piece in an ambient style. This is quite a good musical style to begin

with, because you can concentrate on building an atmosphere rather than getting too worried about the precise placement of notes.

This example mostly uses General Midi voices.

This example assumes you are using Noatikl Standalone!

Start and configure Noatikl

First things first: you must start Noatikl standalone, and configure Noatikl and your system so that you can get sounds out!

Windows: Follow this video tutorial!

Mac: Follow this video tutorial!

Creating your first piece

To create this example piece from scratch, you first need to open a new file; select *File -> New* and select *Standard Template -> Scale Rules*.

Change view: "Piece - Tempo" (use the Object/Parameter View list)

In the Tempo cell, drop the tempo down to about 45 beats per minute.

Change view: "Piece - Rules"

Click on the Scale Rule cell. This will show you a drop down list of available scales. Select Phyrgian from this list. This is now the scale rule used by every voice in the piece unless we override it at voice level.

That's as much as we need to do in this view, so lets get on with building the piece, bit by bit.

Element 1: the drone

Where would ambient music be without a drone somewhere in the background?

Change view: "Voice - Basics"

The first voice has already been defined for us by the template. It needs changing!

You can call the Voice whatever you like by just entering the text you want in the Name Cell. However, we want to leave the name as it is for now.

Set the Patch that this voice will use (for example, you could use *90-Pad 2 (Warm Pad)*, which is the default value.

Set the Pitch to 33 and the Pitch Range to 11. This will force this voice to stay within a single, low octave (note 33 is equivalent to A1 on most keyboards). As this voice will be a continuous drone; set Phrase Gaps to 0 and Phrase Gaps Range to 0. There isn't going to be a phrase gap for this voice, so the Phrase Length parameters are irrelevant.

Change the Voice type to *Ambient* (a click on the voice type box lists the available options!).

Change view: "Voice - Ambient"

You should now set-up the Ambient properties for this voice.

Set the Units to "Full seconds". Set the Duration somewhere around 15 seconds and set the following to 0: Duration Range, Gap Minimum, Gap Range.

After all that, we now have a voice that will play a sustained note, selected from the scale rule at 15 second intervals for the duration of the piece. Press play and see how it sounds!

Not very interesting, but we'll now look to improve on that!

Adding movement to this basic sound

With the focus on this first voice, select *Edit -> Copy*. Now select *Edit -> Paste*. This will create a copy of the first voice, which will save a lot of effort.

Change view: "Voice - Basics"

Rename this voice to "Voice 2", and change it's type to *Follows*.

Change view: "Voice - Following"

Set "Follow Voice" to the name of the first voice, and set the Strategy to "*Semitone Shift*". The other parameters are OK as they are.

All we have now is a second voice that will play identical notes to the first.

Press the Play button, so you can hear what you are doing!

Change view: "Voice - Micro Pitch"

The parameters in this view make very small, subtle shifts to various aspects of the voice in question. This is a useful place to tweak parameters to get a "humanised" feel to your pieces.

In this piece we are only interested in one parameter in this view and that is the Pitch Bend Offset. This can raise or lower the pitch of the voice by small amounts, it's a kind of fine tuning control. While the piece is playing, adjust the Pitch Bend Offset slightly down. As the voice is tuned away from the pitch of the one it is following you will hear the sound thicken up considerably and it will start to gain a sense of motion. Adjust this value to suit your taste.

Change view: "Voice - Envelope - User Envelope 2 (Pan)"

Shift the level up a bit on the first voice, and down a similar amount on the second you will enhance the motion effect. Try it!

Adding another follows voice

This is where we really start to make the piece sound interesting!

Change view: "Voice - Basics"

Paste a copy of Voice 2 into the piece, and rename it to "Voice 3".

Set the Voice Type for this voice to be "*Following*"

Change view: "Voice - Following"

You"ll need to ask it to follow the first voice again. Change the Units to "*Full seconds*", set both the Delay and Delay Range to 5 seconds each, and set the Semitone Shift value to +5.

What we have done here is set up a third element of the drone that means you will have a new note, coming in a perfect fifth above the base drone note some time between 5 and 10 seconds after the drone selects a note to play. Again, it adds interest to the sound and the choice of patch for these voices means that you get a little, slightly randomised chime effect when this third voice kicks in. Play it back and check it out!

So in three simple stages we have gone from a rather boring sound to something with some movement and a bit of surprise about it. So far so good.

It's probably a bit too loud at this point so it might be a good idea to lower the volume envelopes on all three voices now. We can fine tune them later on. Note that, because Noatikl is managing the midi channels for us in this example we don't have to worry about these envelopes being channel specific.

Element 2: What about a melody?

Now we have this nice ambient drone as a bed we need something melodic to set it off. This element demonstrates that you don't have to combine multiple voices to get impressive results. For this element we will use just one voice and get Noatikl to do the donkey work.

Change view: "Voice - Basics"

Select "*Edit -> Add*" to get a new default voice. Rename this voice to "*Voice 4*". This element will be based on a rhythmic voice type.

Set the Patch to be the Acoustic Guitar patch (025). Set the Pitch to 40 (which is the lowest note a guitar normally plays) and set the Pitch Range to 24. Leave the other parameters at their defaults.

Change view: "Voice - Rules"

Just set the Rhythm Rule for this voice to "*Very Slow*". Start the playback and see how it sounds.

It's OK but it's not very interesting as single notes. It sounds a bit mechanical and computerish. We can do better!

Change view: "Voice - Chords"

This show another really powerful area of Noatikl where a couple of tweaks here and there can completely transform a voice. You can do all kinds of special effects in this view, from creating basic chordal shapes, or generating all manner of broken chords right through to creating some very convincing delay effects. On this occasion we"re going to break some chords and stretch them to add some extra improvisations to the very basic melodic themes this voice generates on it's own.

Try these settings. Set the Depth Range to 3. This means that our chords can have between 1 and 4 notes in them. I know a one note chord isn't a chord. It's just another way of saying that there won't be a chord sometimes! If you always want a chord of some sort, you need to set the chord depth to be greater than 1.

Set the Pitch Offset to +7. This is a really fuzzy parameter and it's always hard to predict exactly what it's going to do. In this case it forces the notes of a chord to be spaced up and out from the root note. The space between each note of the chord will be about 7 semitones. So, with a 4 note chord we could be looking a the highest note being somewhere around 28 semitones higher than the lowest. Like a lot of things in Noatikl you just need to play around with this parameter until you get a result you like.

Set the Delay and Delay Range to be 250 each and set the Delay Unit to be'seconds (thousandths of a)". Set the Depth % to around 80% (which means that you won't get any chords at all for 20% of the time) and set the Strategy to be "Chordal Harmony" which means, in this case, that the chords are calculated according to the Piece's Harmony rule.

After all that hit play and see what you have done. Not bad eh? Quite an improvement over the basic, unadorned rhythmic voice and not exactly hard to achieve.

Saving your work

Finally, having composed this piece we need to think about saving it for future use. Saving the file is as easy as any other application you might be familiar with: "File -> Save"

And that's it for this tutorial!

Setting-up Noatikl VSTi to work with Cubase under Mac and Windows

by Pete Cole

This tutorial shows you how to get data out from Noatikl VSTi, into Cubase SE 3 for Mac and Windows, and targeting your favourite software synths! We have also documented the steps required in written form here.

Don't forget to download the template project zip!

Play video (High Definition)

Play video (lower definition)

Setting-up Noatikl VSTi to work with Sonar 6 under Windows

by Pete Cole

This tutorial shows you how to get data out from Noatikl VSTi, into Sonar and targeting your favourite software synths! We have also documented the steps required in written form here.

Don't forget to download the template project zip!

Play video (High Definition)

Play video (lower definition)

Setting-up Noatikl standalone to work with Sonar under Windows

by Pete Cole

This tutorial shows you how to get data out from Noatikl standalone, into Sonar and targeting your favourite software synths!

Don't forget to download the template project zip !

Play video (High Definition)

Play video (lower definition)

Setting-up Noatikl standalone to work with Logic on your Mac

by Mark Harrop

This video is a walkthrough of how to setup Noatikl standalone on the Mac under OS X using either the IAC driver or using MidiPipe and how to set up Logic 7/8 to work with Noatikl using midi sync.

Don't forget to download the template project zip !

Play video (High Definition)

Play video (lower definition)

Setting-up Noatikl AU (Audio Unit) Plug-in to work with Logic on your Mac

by Pete Cole

This video tutorial follows-on from the Noatikl standalone with Logic tutorial, and shows you how to use Noatikl AU with Logic as an alternative!

Don't forget to download the template project zip!

Play video (High Definition)

Play video (lower definition)

Setting-up Noatikl standalone to work with Cubase SE

by Pete Cole

This tutorial shows you how to get data out from Noatikl, into Cubase SE and targeting your favourite software synths!

Don't forget to download the template project zip!

Play video (High Definition)

Play video (lower definition)

Setting-up Noatikl AU (Audio Unit) Plug-in to work with GarageBand on your Mac

by Pete Cole

The trick with GarageBand is to first configure MIDI Pipe to *hijack* the IAC MIDI Port used by Noatikl, *before* starting GarageBand. This video tutorial shows you exactly how to do this.

Play video (High Definition)

Play video (lower definition)

---

note to MIDI CC conversion in noatikl

by Mark Harrop

How to use note to MIDI controller conversion to create a system of MIDI control that conforms to the musical compositional rules in noatikl.

Play video (High Definition)

Play video (lower definition)

Noatikl and mix automation in Logic 8

by Mark Harrop

A detailed walk-through of how to set up Logic 8 to enable midi continuous controllers sent from Noatikl to hook into Logic's internal fader message system. This powerful technique places virtually all synth, plug-in and channel strip parameters under direct generative control.

Play video (High Definition)

Play video (lower definition)

Generative tempo control in Logic Pro using Noatikl standalone

by Mark Harrop

A demo of how to set up Logic to allow Noatikl to generate tempo variations whilst maintaining full midi sync.

Play video (High Definition)

Play video (lower definition)

Tutorial - St Ive's song 07 dissected

by Mark Harrop

A detailed breakdown of the St Ives Song 07, exploring the techniques used in Noatikl to create the piece, including the use of rhythmic, following and ambient voice types, rule definition and chord creation. Also looks at one way to exploit velocity controlled sample switching.

Play video (High Definition)

Play video (lower definition)

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Tip - Autocontrol of Logic Tempo

Set up a pattern Voice in Noatikl on channel 1.

Set another Voice on channel 16. Set this Voice's Micro Controller 1 to send CC22 with the mode set to 1-LFO (min, max, min), beat frequency 2000.

Set Noatikl to sync to the IAC port for which you are generating MIDI clock from Logic!

Open the Logic environment. Cable a tempo fader between midi ports and sequencer input - set it to listen for CC22 on channel 16

Set the piece tempo in Logic to <50 and hit play.

The Tempo of the ensuing Piece ranges between 50 and 172 BPM according to the cycle output of the controller LFO in noatikl.

Sit back and feel clever!

Note that the choice of midi channels and CC22 is entirely arbitrary. You can use any channel or CC you want. As long as the settings in Noatikl are matched to the settings in Logic, you are in business!

You can, of course, set up multiple "voices" in Noatikl simultaneously generating CC 22 on channel 16 to different rules and specifications to create really strange and unpredictable tempo effects. There is a lot more generative potential here than you can get from a simple tempo envelope.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

View: Voice - Basics

This view allows you to define some of the key properties that govern how your Voice work.

Name

Every voice in a Noatikl file has a unique name. You define this in the Name column. You can use any name you want, provided it is not empty, and provided it is not a single question mark (which has a reserved meaning for use with rules, which you will find out about later!).

The name field is available in every view, so we won't go into any more detail with it when we refer to this column later on.

Mute

If you want to mute your voice, click on the cell to set the checkbox to a "tick". To unmute the Voice, click on the checkbox again. Note that certain Voice types might take some time to respond, depending on how far in advance their notes are composed.

When the keyboard focus is on the Mute cell, you have various extra menu options available to you in the "Control" Menu. These are as follows:

- Solo Voice
- Unmute All Voices
- Mute All Voices

If you hold down the *ctrl* key when you click on the mute cell, you will toggle all other voice's mute states, without changing the mute state of the voice that you *ctrl-click* on. This can be very handy!

The mute field is available in every voice parameter view, so we won't go into any more detail with it when we refer to this column later on.

Patch

Every voice is assigned a given Patch. This specifies the sound that you will hear whenever a Noatikl file plays a note. The exact sound you hear depends on how the destination synthesizer works. You might specify piano, then a General MIDI wavetable synthesizer will play you a piano. However, if you are playing through some sort of VSTi or other software synthesizer, you will likely hear something completely different!

In general, Noatikl does not emit any MIDI bank select CC information for a voice before it emits the Patch Change MIDI event. However, you *can* force Noatikl to emit such information, by typing-in a special format patch value; where you type-in the patch in the format: *patch.msb.lsb*, for example:

```
98.53.4
```

In this example, Noatikl will emit bank select CCs for both MSB and LSB according to the settings you supply (53 and 4 respectively, in this case).. If you don't specify a value for the lsb, then Noatikl will only emit a Bank Select MSB CC (CC number 0). If you supply the lsb, then Noatikl will also emit a Bank Select MSB CC (CC number 32).

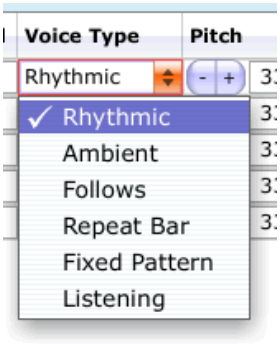
Use Patch?

Not all software synthesizers for your favourite sequencer like having Patch data supplied to them via a Patch Change MIDI event. If this is the case, simply change the *Use Patch?* parameter to *No*, and Noatikl won't generate any MIDI patch change events.

MIDI Channel

A Voice emits data on a MIDI Channel. MIDI channels are numbered from 1 to 16. The default MIDI channel for a voice is actually MIDI channel 0 - which tells Noatikl to assign a free channel from 1 to 16 automatically, as best it can. MIDI channel 10 is always reserved for percussion sounds, such as drum sounds or other untuned sound.

Voice Type



Every Voice composes according to its Voice Type.

Learning how to use the Voice Type is critical to your use of noatikl!

The various Voice Types are as follows:

- **Rhythmic**: Rhythmic Voices are the default voice type. Notes have their durations composed according to the rhythm rules you define for your voice, to fit as well as possible within the bar structure of your piece.
- **Ambient**: Ambient Voices have notes composed such that their durations are defined by the properties in the Ambient property set. Rhythm rules are not used for Ambient Voices. An ambient voice plays its notes without any respect for tempo or bar timings! Ambient Voices are wonderful for creating drifting, floating sounds for either background or foreground use as drones or for musical texture.
- **Follows**: Following Voices are fascinating. They are used to create Voices which work in a call-response manner, following the behaviour of other Voices in a variety of exciting ways according to properties in the Following property set.
- **Repeat Bar**: Repeat Bar Voices are like Rhythmic Voices, with the added feature that they can be defined to repeat work that they have composed in previous bars, according to rules defined in the Repeat property set. When not repeating previous bars, Repeat Bar Voices work in exactly the same way as Rhythmic Voices.
- **Fixed Pattern**: Fixed Pattern Voices are Voices that play in accordance with various fixed MIDI patterns that you import into noatikl. These patterns are able to follow generative sequencing rules, and can adapt automatically to changes in Scale rules. They are great for bringing some structure to your Noatikl composition. Fixed Pattern Voices can also be capable of mutating their patterns while playing, according to properties defined in the Patterns property set.
- **Listening**: Listening Voices are Voices that behave in response to incoming MIDI note events in special ways. Find out more about how to use them to create your own fascinating hyperinstruments here!

When Noatikl plays a voice, you will see the corresponding voice name cell flash, and will also see the a rule element flash if that rule cell was the one that caused the played note to be selected by noatikl. The colour of the flash represents the voice type that is responsible for the flash; you will see that the voice type is colour-coded! This helps you keep track of which rules and voices are working together to create the music you hear.

Pitch

Set the Pitch to be the minimum pitch for which you want your Voice to compose. Noatikl will ensure that it composes no notes less than this pitch value.

Phrase Length

Set this to define the shortest possible sequence of notes that Noatikl will compose in sequence. Noatikl composes a sequence of notes, followed by a sequence of rests. The length of each sequence of notes is governed by this and the Phrase Length Range parameter.

Phrase Length Range

This value defines the upper limit to the number of notes that Noatikl will compose in sequence. For example, if the Phrase Length is 3, and the Phrase Length Range is 25, then the minimum phrase will be 3 notes, and the maximum phrase length will be (3+25) = 28 notes.

Phrase Gaps

Set this to define the shortest possible sequence of rests that Noatikl will compose. Noatikl composes a sequence of notes, followed by a sequence of rests. The length of each sequence of rests is governed by this and the Phrase Gaps Range parameter.

Phrase Gaps Range

This value defines the upper limit to the number of rests that Noatikl will compose in sequence. For example, if the Phrase Gaps is 3, and the Phrase Gaps Range is 25, then the minimum phrase will be 3 rests, and the maximum phrase length will be (3+25) = 30 rests.

Note Rest %

This value defaults to zero. If not zero, then Noatikl will cause the defined percentage of notes that would

otherwise be played by Noatikl, to instead be treated as a rest. This is very useful for making any voice sound more sparsely. Give it a go: this parameter is very powerful, and applies to **all** voice types!

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

View: Voice - Ambient

This view allows you to define some of the key properties that govern how Ambient Voices work.

Name

The name of the Voice.

Mute

The Mute parameter.

Units

You define the Unit of Measure for which the other Ambient Voice parameters are interpreted. This may be one of the following values:

Seconds (thousandths of a)

The properties including Duration are all interpreted as being in thousandths of a second (i.e. Milliseconds). So, a Duration value of 1000 means one second.

Beats (60ths of a)

The properties including Duration are all interpreted as being in 60ths of a beat. In Noatikl a Beat is defined as being one crotchet; you get 4 beats in a bar of 4:4 music. So, a Duration value of 60 means one beat. A Duration value of 30 means a quaver. A Duration value of 20 means a triplet. A Duration value of 15 means a semi-quaver. A Duration value of 240 means 4 beats (which is a full bar if the Piece Meter is 4:4).

Full seconds

The properties including Duration are all interpreted as being in seconds. So, a Duration value of 10 means ten seconds.

Duration

This defines the minimum duration for which the Ambient Voice will play when it composes a note. The actual value chosen for each note is a value between Duration, and Duration plus the Duration Range. Each and every note composed for this Ambient Voice will have a note whose duration is separately calculated.

Duration Range

This is combined with the Duration parameter, to determine the duration for which the Ambient Voice will play when it composes a note. The actual value chosen for each note is a value between Duration, and Duration plus the Duration Range. Each and every note composed for this Ambient Voice will have a note whose duration is separately calculated.

Gap Minimum

This defines the minimum duration for which the Ambient Voice will play when it composes a rest. The actual value chosen for each rest is a value between *Gap Minimum*, and *Gap Minimum* plus the *Gap Range*. Each and every rest composed for this Ambient Voice will have a rest whose duration is separately calculated.

Gap Range

This is combined with the *Gap Minimum* parameter, to determine the duration for which the Ambient Voice will play when it composes a rest. The actual value chosen for each rest is a value between *Gap Minimum*, and *Gap Minimum* plus the *Gap Range*. Each and every note composed for this Ambient Voice will have a note whose duration is separately calculated.

Phrase Length

Set this to define the shortest possible sequence of notes that Noatikl will compose in sequence. Noatikl composes a sequence of notes, followed by a sequence of rests. The length of each sequence of notes is governed by this and the Phrase Length Range parameter.

Phrase Length Range

This value defines the upper limit to the number of notes that Noatikl will compose in sequence. For example, if the Phrase Length is 3, and the Phrase Length Range is 25, then the minimum phrase will be 3 notes, and the

maximum phrase length will be (3+25) = 30 notes.

Phrase Gaps

Set this to define the shortest possible sequence of rests that Noatikl will compose. Noatikl composes a sequence of notes, followed by a sequence of rests. The length of each sequence of rests is governed by this and the Phrase Gaps Range parameter.

Phrase Gaps Range

This value defines the upper limit to the number of rests that Noatikl will compose in sequence. For example, if the Phrase Gaps is 3, and the Phrase Gaps Range is 25, then the minimum phrase will be 3 rests, and the maximum phrase length will be (3+25) = 30 rests.

Note Rest %

This value defaults to zero. If not zero, then Noatikl will cause the defined percentage of notes that would otherwise be played by Noatikl, to instead be treated as a rest. This is very useful for making any voice sound more sparsely. Give it a go: this parameter is very powerful, and applies to **all** voice types!

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

View: Voice - Following

This view allows you to define some of the key properties that govern how Following Voices work.

Name

The name of the Voice.

Mute

The Mute parameter.

Follow Voice

Select the Voice which you want your voice to follow. If you don't specify a Voice to follow, then the following Voice won't play. You may follow any Voice you want, of any type. You may even follow a voice that is following another Voice that is following another Voice... provided that you don't try to define a cyclic dependency which loops back to the current Voice!

Percent

This property tells Noatikl for which percentage of notes that the Followed Voice responds to, the Following Voice actually emits a note. Set to 100 if you want the Following Voice to emit a note for every note played by the Followed Voice. Set to a smaller value if you want to thin-out the notes played by the following voice. This is also useful for building networks of chords, where if you have a number of following voices all following either each other or one main voice, and if those Following Voices have the Percent Parameter to less than 100, then sometimes you will hear dense chords, and sometimes you will hear thinner chords.

Strategy

This property tells Noatikl what it should do when it decides the pitch to use for a note generated by the Following Voice. The available values are:

Chordal Harmony

This causes the Following Voice to choose notes which respect the currently defined Scale, Harmony and Next Note Rules.

Interval Within Scale Rule

This causes the Following Voice to choose notes which are offset from the followed note, such they are at an interval within the Scale Rule, defined as a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values.

For example, if these values are 1 and 2 respectively, then each time a note is chosen, it will be between 1 and (1+2)=3 Scale Rule intervals up from the Followed Voice's note. It is important to understand that this refers to the non-zeroed elements in the current Scale Rule, in other words only those notes that are available within the Scale Rule.

So, in our example, if we were using a Major Scale Rule, and if the followed note were C4 (Middle C), and if Noatikl chose a value of 2 as its random value; then the played note would be E4 (Middle C), which is the second note up from Middle C within the Major Scale Rule.

Semitone Shift

This causes the Following Voice to choose notes which are offset up from the followed note, such they offset from the followed note by a number of semitones which is a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values. A note chosen in this way ignores the current Scale Rule.

For example, if these values are 1 and 2 respectively, then each time a note is chosen, it will be between 1 and (1+2)=3 semitones up from the Followed Voice's note.

So, in our example, if we were using a Major Scale Rule, and if the followed note were C4 (Middle C), and if Noatikl chose a value of 3 as its random value; then the played note would be D#4 (Middle D#), which is the third semitone note up from Middle C. This value is used even though it is not in the current scale rule!

Units

You define the Unit of Measure by which the Delay and Delay Range parameters are interpreted. This may be one of the following values:

Seconds (thousandths of a)

The properties including Duration are all interpreted as being in thousandths of a second (i.e. Milliseconds). So, a Duration value of 1000 means one second.

Beats (60ths of a)

The properties including Duration are all interpreted as being in 60ths of a beat. In Noatikl a Beat is defined as being one crotchet; you get 3 beats in a bar of 4:4 music. So, a Duration value of 60 means one beat. A Duration value of 30 means a quaver. A Duration value of 20 means a triplet. A Duration value of 15 means a semi-quaver. A Duration value of 240 means 4 beats (which is a full bar if the Piece Meter is 4:4).

Full seconds

The properties including Duration are all interpreted as being in seconds. So, a Duration value of 10 means ten seconds.

Delay

This defines the minimum delay after which the Following Voice will play a followed note. The actual value chosen for each note is a value between Delay, and Delay plus the Delay Range. Each and every note composed for this Following Voice will have a note whose delay is separately calculated.

Delay Range

This is combined with the Delay parameter, to determine the delay after which the Following Voice will play a followed note. The actual value chosen for each note is a value between Delay, and Delay plus the Delay Range. Each and every note composed for this Following Voice will have a note whose delay is separately calculated.

Shift / Interval

Used when the Strategy is either Interval Within Scale Rule or Semitone Shift.

This causes the Following Voice to choose notes which are offset in some way from the followed note, according to the Strategy; where the offset is defined as a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values.

S/I Range

This represents the "*Shift/Interval Range*", and is used when the Strategy is either Interval Within Scale Rule or Semitone Shift.

This causes the Following Voice to choose notes which are offset in some way from the followed note, according to the Strategy; where the offset is defined as a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Repeat

This view allows you to define some of the key properties that govern how Repeat Bar voices work.

Name

The name of the Voice.

Mute

The Mute parameter.

Voice

You define the name of the Voice from which you wish to repeat past bars of music from time-to-time. If you simply want to repeat bars played in the past for the current voice, simply select the magic value of '?', which is also the default value.

Percent

When Noatikl starts composing a new bar for this voice, it takes a look at this parameter value; which defines for what percent of the time the Voice should repeat previously-composed music. Set this parameter to 100 if you always want past composed music to be repeated (where available!); set to 0 if you never want past music repeated by this Voice. When the Voice doesn't choose to repeat past data; it composes a new bar of music using the normal rules.

Bars

Defines the number of bars for which the voice should repeat a past-composed chunk of music. The actual value chosen is somewhere between Bars and Bars + Bars Range.

Bars Range

Defines the upper limit of the number of bars for which the voice should repeat a past-composed chunk of music. The actual value chosen is somewhere between Bars and Bars + Bars Range.

History

Defines the number of bars in the past, from which Noatikl will choose the past-composed music to repeat. The actual value chosen is somewhere between History and History + History Range.

History Range

Defines the upper limit of the number of bars in the past, from which Noatikl will choose the past-composed music to repeat. The actual value chosen is somewhere between History and History + History Range.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

View: Voice - Patterns

This view allows you to define some of the key properties that govern how Fixed Pattern Voices work.

Name

The name of the Voice.

Mute

The Mute parameter.

Use Percent

When Noatikl starts a new sub-pattern at the start of a bar, noatikl consults the value you have defined for *Use Percent*. This property determines the probably of Noatikl using the Pattern for the bar; or alternatively, compose a *completely* new bar (that you will hear only once!) according to the normal rules for the voice.

If *Use Percent* is 100, then Noatikl will always use the pattern. If *Use Percent* is 50, then Noatikl will instead compose a new bar every other bar or so. Note that Noatikl will never interrupt a sub-pattern that it is playing; the *Use Percent* property is considered only on a sub-pattern boundary, at the start of a new bar.

Mutation Factor

The mutation factor is used when a bar is considered for mutation (which can happen only if *Bars Between* is not zero! The *Mutation Factor* determines the level of mutation to apply. If set to 10.0%, then when playing from a sub-pattern, this means that each note that would be played from the pattern, has a 10% chance of having a different one composed, with subsequence pattern playbacks keeping that mutation! Note that if *Mutate Rhythm?* is set to *Yes*, then if the composed note is longer than the composed-over pattern note, this might overlap and cancel-out some other notes in the sub-pattern!

Bars Between

This property defines the number of bars that Noatikl waits between, before trying to mutate a bar in a pattern according to the *Mutation Factor*. If *Bars Between* is set to zero, the voice can never mutate. Set to 1 if you want mutation every bar, 2 if you want mutation every other bar, etc. ...

The actual number of bars used is selected randomly each time, somewhere in the range from *Bars Between*, to *Bars Between* plus *Bars Range*.

Bars Range

This property is used to help define the number of bars between attempts by Noatikl to mutate the current pattern. The actual number of bars used is selected randomly each time, somewhere in the range from *Bars Between*, to *Bars Between* plus *Bars Range*.

Mutate Rhythm?

If set to *No*, then the timing of the sub-pattern is preserved perfectly; only the frequency of the pattern notes will be changed when the pattern is mutated. Otherwise, the duration of each note is chosen from the rhythm rules and phrase/phrase gap rules for the voice.

Meter

Defines the Meter to be used by this Voice, such 4:4 or 3:4 or 6:8. A value of ?, which is the default, means to use the Meter defined for the Piece. A different value allows the Voice to work with a completely different meter, which can be used for interesting polyphonic effects!

Patterns

Defines the Pattern to be used by a Fixed Pattern Voice.

Tip: The pattern syntax is very complicated, and existing Koan Pro users might wish to use that tool for now to perform pattern editing, until such time as we have created a pattern editor for noatikl!

Until such time, you will be interested to hear that patterns are defined using strings, where the strings have quite a complicated syntax! The string can contain a list of sub-patterns. I'll repeat that: each Pattern is made-up of a number of Sub-Patterns.

There are 4 types of Sub-Pattern:

Note sub-pattern types:

- R - Rhythm only. Defines note durations to use, but leaves selection of the note pitches to use up to noatikl!
- F - Frequency. Defines the pitch relative to the current scale, whatever that might be but not the duration. Useful for drum riffs!
- B - Melodic, i.e. "Both". Both the above - the most common type.

Sequence sub-pattern types:

- S - sequence. Sequenced patterns allow Noatikl to use generative rules to select which sub patterns to use while playing a pattern as a generative sequence of sub-patterns.

Which sub-pattern is chosen by Noatikl, depends on a few things:

If there is at least one sequenced sub pattern, then a sequence is used to drive the sub-pattern. Which sequence to use, is based on Noatikl making a weighted random selection from the available sub-patterns. When (if!) the sequenced sub-pattern end is reached, Noatikl will make another selection as to which sequenced sub-pattern to use.

Otherwise, a sub-pattern is chosen, based on Noatikl making a weighted random selection from the available sub-patterns. This sub-pattern is played through to the end, at which point Noatikl will make another selection as to which sub-pattern to play.

A Note Sub-Pattern that is less than a whole number of bars at the Voice's current Meter, will be padded automatically with silence to ensure that it remains bar synchronised.

See Also

- Patterns: Examples
- Patterns: General syntax
- Patterns: Sequence sub-patterns

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

View: Voice - Chords

This view allows you to define some of the key properties that allow you to configure *any* Voice Type to generate chords automatically.

The properties in this view are incredibly powerful and very easy to use!

In outline, use the *Depth* and *Depth Range* values to define the "Chord Depth"; which is the number of notes that Noatikl will play at any one time for a given voice. The first note in any chord is composed according to the normal mechanism for a voice type; additional notes that cause a chord to be built-up may be generated automatically by Noatikl according to the properties in this view.

Name

The name of the Voice.

Mute

The Mute parameter.

Depth

Specify the minimum *Depth* of chord that you want your Voice to play with. A value of 1 will mean that the Voice will not chord (unless the *Depth Range* property is greater than zero)!

The *Depth* defines the number of notes that are played by the voice at any one time.

Depth Range

Specify the relative maximum *Depth* of chord that you want your Voice to play with. A value of 0 means that whenever the Voice is played, it will play a number of notes equal to the *Depth*. A value of one or more means that whenever the Voice is played, it will play a number of notes equal to a randomly selected value between the *Depth* and the *Depth* plus the *Depth Range*.

Percent

This property tells Noatikl the percentage chance that it should actually emit any given note in the chord (after the first note, of course!). Set to 100 if you want the Chording Voice to always emit a note for every note played by the Followed Voice. Set to a smaller value if you want to thin-out the notes played within the chord.This allows you to create chords of varying depth; sometimes dense, sometimes thin.

Strategy

This property tells Noatikl what it should do when it decides the pitch to use for a note generated within a chord. The available values are:

Chordal Harmony

This causes the Voice's chord notes to be selected according to the currently defined Scale, Harmony and Next Note Rules.

Interval Within Scale Rule

This causes the Voice's chord note to be selected offset from the followed note, such they it is at an interval within the Scale Rule beyond the previous note in the chord, defined as a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values.

For example, if these values are 1 and 2 respectively, then each time a note is chosen within the chord, it will be between 1 and (1+2)=3 Scale Rule intervals up from the previous note in the chord. It is important to understand that this refers to the non-zeroed elements in the current Scale Rule, in other words only those notes that are available within the Scale Rule.

So, in our example, if we were using a Major Scale Rule, and if the first note in the chord were C4 (Middle C), and if Noatikl chose a value of 2 as its random value; then the played note would be E4 (Middle C), which is the second note up from Middle C within the Major Scale Rule.

Semitone Shift

This causes the the Voice's chord note to be selected offset up from the previous note in the chord, such it is offset from the previous chord note by a number of semitones which is a value randomly selected between the

Shift / Interval and Shift / Interval plus Shift / Interval Range values. A note chosen in this way ignores the current Scale Rule.

For example, if these values are 1 and 2 respectively, then each time a note is chosen, it will be between 1 and (1+2)=3 semitones up from the previous note in the chord.

So, in our example, if we were using a Major Scale Rule, and if the previous note in the chord were C4 (Middle C), and if Noatikl chose a value of 3 as its random value; then the played note would be D#4 (Middle D#), which is the third semitone note up from Middle C. This value is used even though it is not in the current scale rule!

Units

You define the Unit of Measure by which the Delay and Delay Range parameters are interpreted. This may be one of the following values:

Seconds (thousandths of a)

The properties including Duration are all interpreted as being in thousandths of a second (i.e. Milliseconds). So, a Duration value of 1000 means one second.

Beats (60ths of a)

The properties including Duration are all interpreted as being in 60ths of a beat. In Noatikl a Beat is defined as being one crotchet; you get 3 beats in a bar of 4:4 music. So, a Duration value of 60 means one beat. A Duration value of 30 means a quaver. A Duration value of 20 means a triplet. A Duration value of 15 means a semi-quaver. A Duration value of 240 means 4 beats (which is a full bar if the Piece Meter is 4:4).

Quantized Beats (60ths of a)

This works the same way as *Beats (60ths of a)* except that where the *Delay* has a special value of 10, 15 or 20; the delay is interpreted in a special way that is very useful for some breakbeat-based music. Specifically, in this case, the calculated value for the delay is rounded to the nearest sub-multiple of the *Delay* value. So, for example, if the engine calculates a value of 43, and if *Delay* is 20, the used value for the delay is actually 40 (which is the nearest multiple of 20).

Delay

This defines the minimum delay after which the Chording Voice will play a followed note. The actual value chosen for each note is a value between Delay, and Delay plus the Delay Range. Each and every note composed for this Chording Voice will have a note whose delay is separately calculated.

Delay Range

This is combined with the Delay parameter, to determine the delay after which the Chording Voice will play a followed note. The actual value chosen for each note is a value between Delay, and Delay plus the Delay Range. Each and every note composed for this Chording Voice will have a note whose delay is separately calculated.

Shift / Interval

Used when the Strategy is either Interval Within Scale Rule or Semitone Shift.

This causes the Chording Voice to choose notes which are offset in some way from the followed note, according to the Strategy; where the offset is defined as a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values.

S/I Range

This represents the "*Shift/Interval Range*", and is used when the Strategy is either Interval Within Scale Rule or Semitone Shift.

This causes the Chording Voice to choose notes which are offset in some way from the followed note, according to the Strategy; where the offset is defined as a value randomly selected between the Shift / Interval and Shift / Interval plus Shift / Interval Range values.

Pitch Offset

This property defines the amount that the pitch of each note in the chord should be offset, in semitones, from the previous note in the chord; the actual value selected might be overridden according to the various rules that apply to the Voice, but in general, this parameter allows you to "shape" a chord to have a given range of pitch values. In combination with the *Delay*-related parameters, this allows you to create some very interesting arpeggiation effects.

For example, a value of +12 would tend to space each note in the chord by a range of 12 semitones (which is one octave), with each subsequent value in the chord being higher in pitch that the previous.

For example, a value of -12 would tend to space each note in the chord by a range of 12 semitones (which is one octave), with each subsequent value in the chord being lower in pitch that the previous.

Velocity Factor

This property allows you to specify the range of relative velocities for the notes in a chord. Each subsequent note in the chord is the defined percentage louder (for a positive value) or quieter (for a negative value) than the previous note in the chord. A value of zero means that all notes in the chord are played with the same velocity.

The Voice velocity envelope values are ignored when this parameter is applied.

For example, a value of *-30* would tell Noatikl to generate this Voice's chords such that each auto-chorded note is 30% quieter than each preceding note in the chord; giving a noticeable tailing-off effect.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

View: Voice - Rules

This view allows you to define some of the key properties that govern how your Voice works.

Name

The name of the Voice.

Mute

The Mute parameter.

Harmony Rules

Select the Harmony Rule that you want your Voice to use. If you select the value labelled ?, this tells Noatikl to use the value defined in the Piece object. If that value is in turn the ? value, then Noatikl will choose a rule to use at random when it starts playing!

Next Note Rules

Select the Next Note Rule that you want your Voice to use. If you select the value labelled ?, this tells Noatikl to use the value defined in the Piece object. If that value is in turn the ? value, then Noatikl will choose a rule to use at random when it starts playing!

Rhythm Rules

Select the Rhythm Rule that you want your Voice to use. If you select the value labelled ?, then Noatikl will choose a rule to use at random when it starts playing!

Scale Rules

Select the Scale Rule that you want your Voice to use. If you select the value labelled ?, this tells Noatikl to use the value defined in the Piece object. If that value is in turn the ? value, then Noatikl will choose a rule to use at random when it starts playing!

Harmonise?

The default value for this property is “Yes”, which means that the Voice will be considered for harmonisation with other voices. Set to “No” if for some reason you do not want other voices to harmonize with this voice!

Voice Root

Normally, you want your Voice to use the Piece Root. This is represented by the value ? For the Voice Root parameter. However, sometimes you really want to force your Voice to use a different Root note; in which case, set the Voice Root to be whatever value suits.

This allows you to work-around the following sort of problem:

Imagine that you have a sampler, wher you load-up a variety of loops against MIDI note C3 up to D3. To have your piece drive this from a Rhythmic Voice such that the sounds you hear are not affected by changes to the Piece Root, you should set the Voice Root to e.g. C3 and your Voice will then be unaffected by changes to the Piece Root. Note that in this specific example, it would probably be a good idea to set the Harmonize? Flag to No.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Scripts

This view allows you to embed small Trigger Scripts, which are small bits of code in the widely Lua language, that are triggered when various events happen when Noatikl is playing. Using trigger scripts allows you to tell Noatikl to behave in very powerful ways while it is playing.

Name

The name of the Voice.

Mute

The Mute parameter.

Start

Press this button to show the Start trigger script in the Script Editor window.

The Start Trigger Script is called once at the start of the Piece, when the piece starts playing.

```
function nt_trigger_start()
  print ("Voice start!")
end
```

Bar

Press this button to show the Bar trigger script in the Script Editor window.

The Bar Trigger Script is called at the start of every bar while the piece is playing.

```
function nt_trigger_bar(bar)
  print ("Voice Bar number", bar)
end
```

Composed

Press this button to show the Composed trigger script in the Script Editor window.

The Composed Script is called when Noatikl composes a note. Use this to emit MIDI CC events and what have you!

```
function nt_trigger_composed(noteon, channel, pitch, velocity)
  print ("Voice Composed", noteon, channel, pitch, velocity)
end
```

MIDI In CC

Press this button to show the MIDI In CC trigger script in the Script Editor window.

The MIDI In CC Trigger Script is called whenever a MIDI CC event is received by the MIDI Input device.

```
function nt_trigger_midiin_cc(channel, cc, value)
  print ("Voice MIDI In CC", channel, cc, value)
end
```

MIDI In Note

Press this button to show the MIDI In Note trigger script in the Script Editor window.

The MIDI In Note Trigger Script is called whenever a MIDI Note On or Off event is received by the MIDI Input device.

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
  print ("Voice MIDI In note ", noteon, channel, pitch, velocity)
end
```

Stop

Press this button to show the Stop trigger script in the Script Editor window.

The Stop Trigger Script is called once at the end of the Piece, just as the Piece stops playing.

```
function nt_trigger_stop()
  print ("Piece stop!")
end
```

See Also

- Scripting Overview
- Trigger Scripts
- Scripting Reference
- Noatikl as a Hyperinstrument

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice – Notes

This view allows you to store notes on your voice, including copyright information and any notes you might want to make for future reference.

Name

The name of the Voice.

Mute

The Mute parameter.

Copyright

Press this button to view the Copyright information you might want to record for the voice. In the case of a voice from a template pack, this might contain a copyright notice associated with that template.

Notes

Press this button to edit any detailed notes you might want to make about this voice for future reference.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Controllers

This view allows you to define some of the key controller values that are emitted by the Voice.

Name

The name of the Voice.

Mute

The Mute parameter.

Damper/Hold (64)

Set this value to other than the default of "-1", if you want to emit a Damper/Hold MIDI controller (MIDI CC 64) at the specified value on this Voice's MIDI line. This is a funny MIDI controller, with only two states; in that a value of 64 or greater activates Damper/Hold, and any value of 63 or less means to turn it off! Leave this value at the default of "-1" if you don't want Noatikl to emit any information for this MIDI controller.

Harmonic Content (71)

Set this value to other than the default of "-1", if you want to emit a Harmonic Content MIDI controller (MIDI CC 71) at the specified value on this Voice's MIDI line. Leave this value at the default of "-1" if you don't want Noatikl to emit any information for this MIDI controller.

Reverb (91)

Set this value to other than the default of "-1", if you want to emit a Reverb MIDI controller (MIDI CC 91) at the specified value on this Voice's MIDI line. Leave this value at the default of "-1" if you don't want Noatikl to emit any information for this MIDI controller.

Chorus (93)

Set this value to other than the default of "-1", if you want to emit a Chorus MIDI controller (MIDI CC 93) at the specified value on this Voice's MIDI line. Leave this value at the default of "-1" if you don't want Noatikl to emit any information for this MIDI controller.

Damper Release

If you are using Damper/Hold (64), then you will find that your notes can start building-up and never decay! In which case, set the Damper Release property to "Yes", which tells Noatikl to momentarily release the damper just before the end of every bar. This prevents build-up of notes and generally sounds wonderful.

Portamento (65)

Set this value to other than the default of "-1", if you want to emit a Portamento MIDI controller (MIDI CC 65) at the specified value on this Voice's MIDI line. Leave this value at the default of "-1" if you don't want Noatikl to emit any information for this MIDI controller.

MIDI Channel Sharing

The default value of "Yes" means that this Voice can share its MIDI channel with other Voices. This is only considered if you have defined the MIDI Channel property for a Voice to be 0.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

View: Voice - Micro Controller 1

This view allows you to define a very powerful Microcontroller that is associated with your Voice!

Microcontrollers are a very powerful of noatikl. You can think of them as built-in, highly configurable MIDI event generators. The can either synchronise to the tempo of Noatikl, or you can let them run free-floating. Experiment with them - they can do a huge amount to make your music interesting and dynamic!

Tip: if you want to synchronise your Microcontroller to the time-base, so that your MIDI controller is synchronised to bar boundaries in your music, you'll need to use the Beat Cycle Length property!

Name

The name of the Voice.

Mute

The Mute parameter.

MIDI CC

This tells Noatikl which MIDI controller (also referred to as the MIDI CC) to emit for this microcontroller. When the Microcontroller is active, Noatikl will emit values for this MIDI controller that change at various times, with behaviour that you define using the various parameters in this Parameter.

Mode

The Mode defines the shape of the waveform that Noatikl will use to shape this waveform.

The Mode may be one of the following values:

-1 - Off

The microcontroller is off. This is the default value.

0 - Random Drift

The microcontroller will drift between the Minimum and Minimum plus Range, changing at times specified by the Update and Update Range parameters, by an amount between the Change and Change plus Change Range parameters.

1 - LFO (Min-Max-Min)

A triangular waveform, that starts at the minimum value, works up to the maximum value, and works back to the minimum value.

2 - LFO (Max-Min-Max)

A triangular waveform, that starts at the maximum value, works down to the minimum value, and works back to the maximum value.

3 - Sawtooth (Min-Max)

A sawtooth waveform, that starts at the minimum value, works up to the maximum value, and then starts again from the minimum value.

4 - Sawtooth (Max-Min)

A sawtooth waveform, that starts at the maximum value, works down to the minimum value, and then starts again from the maximum value.

Minimum

Defines the minimum value that may be emitted by the Microcontroller.

Range

The microcontroller will emit a value between the Minimum and Minimum plus Range values.

So for example, if you define Minimum to be 20, and Range to be 100, the value that is emitted will be in the range 20 to 120 inclusive.

Change

Defines the amount by which the microcontroller will change, every time it is allowed to change. Typically set to a value of 1. If this value is set to 0, the Microcontroller will change only if the Change Range is greater than or equal to 1!

Change Range

Defines the upper limit to the amount by which the microcontroller will change, every time it is allowed to change. Typically set to a value of 1. If this value is set to 0, the Microcontroller will change only if the Change Range is greater than or equal to 1!

For example, if you define Change to be 1, and Change Range to be 3, the value that is emitted will vary by a value between 1 and (3+1)=4 each time.

Update

Defines the minimum time in milliseconds between changes in the emitted Microcontroller value. The system might not be able to emit changes as quickly as you want, if you set a very small value! If you don't want changes to happen very often, then use a large value.

Ignored if Beat Cycle Length is non-zero!

Update Range

Defines the upper limit in the time in milliseconds between changes in the emitted Microcontroller value. Use this property to apply some uncertainty in when the changes will occur!

For example, if you define Update to be 1000, and Update Range to be 500, the value that is emitted will change every 1000 to 1500 milliseconds (or in other words, every 1 to 1.5 seconds).

Ignored if Beat Cycle Length is non-zero!

Update Units

You define the Unit of Measure by which the Update and Update Range parameters are interpreted. This may be one of the following values:

Seconds (thousandths of a)

The Update and Update Range are interpreted as being in thousandths of a second (i.e. Milliseconds). So, a Update value of 1000 means one second.

Full seconds

The Update and Update Range are interpreted as being in seconds. So, a Update value of 10 means ten seconds.

Beat Cycle Length

This parameter is critical for using Noatikl to generate effects which synchronise with the bar timing of your voice! If you want to achieve an effect like a filter-sweep that synchronises to your bar boundary, then this is the property to use.

Here are some of the values you could use.

Note in Noatikl a Beat is defined as being one crotchet; you get 4 beats in a bar of 4:4 music. So, a Duration value of 60 means one beat. A Duration value of 30 means a quaver. A Duration value of 20 means a triplet. A Duration value of 15 means a semi-quaver. A Duration value of 240 means 4 beats (which is a full bar if the Piece Meter is 4:4).

Phase Shift%

Use this property if you want to start the microcontroller from a start-point other than at the very start of its cycle.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

View: Voice - Micro Controller 2

This view allows you to define a very powerful microcontroller that is associated with your Voice!

Microcontrollers are a very powerful of noatikl. You can think of them as built-in, highly configurable MIDI event generators. The can either synchronise to the tempo of Noatikl, or you can let them run free-floating. Experiment with them - they can do a huge amount to make your music interesting and dynamic!

Tip: if you want to synchronise your Microcontroller to the time-base, so that your MIDI controller is synchronised to bar boundaries in your music, you'll need to use the Beat Cycle Length property!

Name

The name of the Voice.

Mute

The Mute parameter.

MIDI CC

This tells Noatikl which MIDI controller (also referred to as the MIDI CC) to emit for this microcontroller. When the Microcontroller is active, Noatikl will emit values for this MIDI controller that change at various times, with behaviour that you define using the various parameters in this Parameter.

Mode

The Mode defines the shape of the waveform that Noatikl will use to shape this waveform.

The Mode may be one of the following values:

-1 - Off

The microcontroller is off. This is the default value.

0 - Random Drift

The microcontroller will drift between the Minimum and Minimum plus Range, changing at times specified by the Update and Update Range parameters, by an amount between the Change and Change plus Change Range parameters.

1 - LFO (Min-Max-Min)

A triangular waveform, that starts at the minimum value, works up to the maximum value, and works back to the minimum value.

2 - LFO (Max-Min-Max)

A triangular waveform, that starts at the maximum value, works down to the minimum value, and works back to the maximum value.

3 - Sawtooth (Min-Max)

A sawtooth waveform, that starts at the minimum value, works up to the maximum value, and then starts again from the minimum value.

4 - Sawtooth (Max-Min)

A sawtooth waveform, that starts at the maximum value, works down to the minimum value, and then starts again from the maximum value.

Minimum

Defines the minimum value that may be emitted by the Microcontroller.

Range

The microcontroller will emit a value between the Minimum and Minimum plus Range values.

So for example, if you define Minimum to be 20, and Range to be 100, the value that is emitted will be in the range 20 to 120 inclusive.

Change

Defines the amount by which the microcontroller will change, every time it is allowed to change. Typically set to a value of 1. If this value is set to 0, the Microcontroller will change only if the Change Range is greater than or equal to 1!

Change Range

Defines the upper limit to the amount by which the microcontroller will change, every time it is allowed to change. Typically set to a value of 1. If this value is set to 0, the Microcontroller will change only if the Change Range is greater than or equal to 1!

For example, if you define Change to be 1, and Change Range to be 3, the value that is emitted will vary by a value between 1 and (3+1)=4 each time.

Update

Defines the minimum time in milliseconds between changes in the emitted Microcontroller value. The system might not be able to emit changes as quickly as you want, if you set a very small value! If you don't want changes to happen very often, then use a large value.

Ignored if Beat Cycle Length is non-zero!

Update Range

Defines the upper limit in the time in milliseconds between changes in the emitted Microcontroller value. Use this property to apply some uncertainty in when the changes will occur!

For example, if you define Update to be 1000, and Update Range to be 500, the value that is emitted will change every 1000 to 1500 milliseconds (or in other words, every 1 to 1.5 seconds).

Ignored if Beat Cycle Length is non-zero!

Update Units

You define the Unit of Measure by which the Update and Update Range parameters are interpreted. This may be one of the following values:

Seconds (thousandths of a)

The Update and Update Range are interpreted as being in thousandths of a second (i.e. Milliseconds). So, a Update value of 1000 means one second.

Full seconds

The Update and Update Range are interpreted as being in seconds. So, a Update value of 10 means ten seconds.

Beat Cycle Length

This parameter is critical for using Noatikl to generate effects which synchronise with the bar timing of your voice! If you want to achieve an effect like a filter-sweep that synchronises to your bar boundary, then this is the property to use.

Here are some of the values you could use.

Note in Noatikl a Beat is defined as being one crotchet; you get 4 beats in a bar of 4:4 music. So, a Duration value of 60 means one beat. A Duration value of 30 means a quaver. A Duration value of 20 means a triplet. A Duration value of 15 means a semi-quaver. A Duration value of 240 means 4 beats (which is a full bar if the Piece Meter is 4:4).

Phase Shift%

Use this property if you want to start the microcontroller from a start-point other than at the very start of its cycle.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Micro Note Delay

This view provides fine variation in the times of Note events generated by noatikl. This can be used to give Noatikl a more "human" feel.

Name

The name of the Voice.

Mute

The Mute parameter.

Delay Range

The maximum amount of delay generated by micro note delay changes, that may be applied to note events. Zero means off (which is the default).

Delay Change

The amount of change in the micro delay that is applied by Noatikl between note on/off events. The value drifts between zero (off) and the Delay Range, changing by plus or minus the Delay Change value each time.

Delay Offset

Fixed amount of offset note delay to apply, used only when the Micro Note Delay controller is in use. The default value is zero.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Controller Micro Pitch

This view provides fine variation through use of the MIDI Pitch Wheel controller.

Tip: This is not normally used on MIDI line 10, which is the drum/percussion line!

Name

The name of the Voice.

Mute

The Mute parameter.

Bend Sensitivity

A value from 0 to 24, meaning how many semitones are controlled by the full available range of Micro Pitch parameters. The default value is 2, which represents two semitones.

Pitch Bend Offset

Fixed amount of pitch-bend to apply on this MIDI line, used to tune/de-tune an instrument.

The default value is zero, which means to apply no offset pitch bend.

From -8192 to +8191; which covers a range of pitch bend defined by the Bend Sensitivity property.

Pitch Range

The maximum amount of micro pitch change that can be applied. Zero means off (which is the default). The maximum value allowed is 8191! The value chosen is added to the pitch bend offset.

Pitch Change

The amount of change in Micro Pitch that is applied by Noatikl between “update” periods. The value drifts between zero (off) and the Pitch Range, changing by plus or minus the Pitch Change value each time.

Pitch Update

The time in milliseconds between updates to the pitch controller. The actual value chosen is selected randomly each time, to be a value somewhere between Pitch Update and Update Range.

Update Range

The upper limit of time between updates to the pitch controller. The actual value chosen is betwen Pitch Update and Update Range.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

View: Voice - Note To MIDI CC Mapping

This view allows you to tell Noatikl to emit MIDI controller data instead of MIDI note events!

Normally, Noatikl composes and emits MIDI note events. However, you can tell Noatikl to instead send MIDI controller data instead of MIDI note events.

Why would you want to do this? Well, it lets you use Noatikl as a very powerful generative MIDI event generator with a huge range of potential applications!

Name

The name of the Voice.

Mute

The Mute parameter.

CC for Note On?

If you want this Voice to emit a MIDI CC instead of note on/off events, set this parameter to Yes.

Note On CC

If you have set CC for Note On? to Yes, then instead of emitting a note on event, Noatikl will emit the specified MIDI CC, with a value equal to the composed pitch!

CC For Velocity?

If you want this Voice to emit a MIDI CC proportionate to the Velocity of the composed note (in addition to any controller defined for Note On CC), then set this parameter to Yes.

Velocity CC

If you have set CC for Velocity? to Yes, then Noatikl will (in addition to the Note On CC value) emit the specified MIDI CC, with a value equal to the composed velocity!

CC for Note Off?

If you want this Voice to emit a MIDI CC when a note off occurs, set this parameter to Yes. This applies only if CC for Note On? Is set to Yes.

Note On CC

If you have set CC for Note Off? to Yes, then instead of emitting a note off event, Noatikl will emit the specified MIDI CC, with a value equal to the composed pitch of the stopped note!

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

Credits

We have so many people to thank for their support over the years, which has helped us get to where we are today with noatikl. The list below is an ever expanding one, in no particular order, and does not include everyone (sorry to those we have missed!) :

- everyone who has supported our Noatikl development efforts by purchasing from our store – you are the people who make it all possible.
- everyone who has made contributions to the Noatikl forum, blogged about our Noatikl products and given feedback.
- those who have helped others on the Noatikl forum, for which we are especially grateful.
- our outstanding Noatikl beta testers for their testing efforts (and those who are happy to be mentioned include: www.myspace.com/sashimirecords).
- Mark Harrop, to whom we are hugely grateful to for his stirling efforts and superb Noatikl videos.
- Timothy Didymus, a longstanding friend with incredible talent whose sublime generative music creations have, on many occasions when times were hard, given us the much needed inspiration to press onwards.
- Brian Eno, for his support and kindness to us over the years – he is such an amazing and incredible man and talent.
- Our old friends and colleagues from SSEYO days including John Wilkinson, Jon Pettigrew, Jerry Leach, Paul Blampied, Jerry Swan and Simon Robertson et al.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Envelope - Velocity

This view allows you to define how the Velocity level is changed automatically for your voice! Note that the velocity defines relatively how loud each note is.

Name

The name of the Voice.

Mute

The Mute parameter.

Velocity

This parameter is shown as a graphical representation of the Velocity envelope to be followed through playback of the Noatikl piece. The piece starts with the value at the left side of the envelope, and as the piece progresses, eventually the value from the far right of the envelope is used.

The actual value used at any point, is calculated as being a value somewhere in the range from Velocity, to Velocity plus Velocity Range. The velocity can change by amount differing from the last used velocity, by a value somewhere in the range from Velocity Change, to Velocity Change plus Velocity Change Range

You can draw direct on to the envelope with your mouse.

Alternatively, you may use one of the various powerful envelope editing tools that we have made available to you.

To use the envelope editing tools:
- Right-click (win) or ctrl-click (mac) on the envelope tool
- Select the option you want. e.g. random, curve up etc.
- Select the range using the mouse...
- Then: either press space or enter, or select pop-up envelope tool to apply to the selected range.
- Select freehand mode to return to the normal click-to-paint mode.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Envelope - Velocity Range

This view allows you to define how the Velocity Range level is changed automatically for your voice! Note that the velocity defines relatively how loud each note is.

Name

The name of the Voice.

Mute

The Mute parameter.

Velocity Range

This parameter is shown as a graphical representation of the Velocity Range envelope to be followed through playback of the Noatikl piece. The piece starts with the value at the left side of the envelope, and as the piece progresses, eventually the value from the far right of the envelope is used.

The actual value used at any point, is calculated as being a value somewhere in the range from Velocity, to Velocity plus Velocity Range. The velocity can change by amount differing from the last used velocity, by a value somewhere in the range from Velocity Change, to Velocity Change plus Velocity Change Range

You can draw direct on to the envelope with your mouse.

Alternatively, you may use one of the various powerful envelope editing tools that we have made available to you.

To use the envelope editing tools:
- Right-click (win) or ctrl-click (mac) on the envelope tool
- Select the option you want. e.g. random, curve up etc.
- Select the range using the mouse...
- Then: either press space or enter, or select pop-up envelope tool to apply to the selected range.
- Select freehand mode to return to the normal click-to-paint mode.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Envelope - Velocity Change

This view allows you to define how the Velocity level is changed automatically for your voice! Note that the velocity defines relatively how loud each note is.

Name

The name of the Voice.

Mute

The Mute parameter.

Velocity Change

This parameter is shown as a graphical representation of the Velocity Change envelope to be followed through playback of the Noatikl piece. The piece starts with the value at the left side of the envelope, and as the piece progresses, eventually the value from the far right of the envelope is used.

The actual value used at any point, is calculated as being a value somewhere in the range from Velocity, to Velocity plus Velocity Range. The velocity can change by amount differing from the last used velocity, by a value somewhere in the range from Velocity Change, to Velocity Change plus Velocity Change Range

You can draw direct on to the envelope with your mouse.

Alternatively, you may use one of the various powerful envelope editing tools that we have made available to you.

To use the envelope editing tools:
- Right-click (win) or ctrl-click (mac) on the envelope tool
- Select the option you want. e.g. random, curve up etc.
- Select the range using the mouse...
- Then: either press space or enter, or select pop-up envelope tool to apply to the selected range.
- Select freehand mode to return to the normal click-to-paint mode.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Envelope - Velocity Change Range

This view allows you to define how the Velocity level is changed automatically for your voice! Note that the velocity defines relatively how loud each note is.

Name

The name of the Voice.

Mute

The Mute parameter.

Velocity Change Range

This parameter is shown as a graphical representation of the Velocity Change Range envelope to be followed through playback of the Noatikl piece. The piece starts with the value at the left side of the envelope, and as the piece progresses, eventually the value from the far right of the envelope is used.

The actual value used at any point, is calculated as being a value somewhere in the range from Velocity, to Velocity plus Velocity Range. The velocity can change by amount differing from the last used velocity, by a value somewhere in the range from Velocity Change, to Velocity Change plus Velocity Change Range

You can draw direct on to the envelope with your mouse.

Alternatively, you may use one of the various powerful envelope editing tools that we have made available to you.

To use the envelope editing tools:
- Right-click (win) or ctrl-click (mac) on the envelope tool
- Select the option you want. e.g. random, curve up etc.
- Select the range using the mouse...
- Then: either press space or enter, or select pop-up envelope tool to apply to the selected range.
- Select freehand mode to return to the normal click-to-paint mode.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Voice - Envelope - User Envelope 1 (Volume)

This view allows you to define an envelope that is used to emit a MIDI CC of your choice. The default value for this MIDI CC is 7, which is the Volume controller.

Name

The name of the Voice.

Mute

The Mute parameter.

MIDI CC

Use this to define the MIDI CC that you want to be emitted by this envelope. The default value is 7, which is the Volume controller.

Enabled?

Use this to turn your envelope on or off.

Envelope

This parameter is shown as a graphical representation of the Envelope to be followed through playback of the Noatikl piece. The piece starts with the value at the left side of the envelope, and as the piece progresses, eventually the value from the far right of the envelope is used.

You can draw direct on to the envelope with your mouse.

Alternatively, you may use one of the various powerful envelope editing tools that we have made available to you.

To use the envelope editing tools:
- Right-click (win) or ctrl-click (mac) on the envelope tool
- Select the option you want. e.g. random, curve up etc.
- Select the range using the mouse...
- Then: either press space or enter, or select pop-up envelope tool to apply to the selected range.
- Select freehand mode to return to the normal click-to-paint mode.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

View: Voice – Envelope – Micro User (Envelope 1)

This view provides fine variation in the values generated by the User Envelope 1. Any value generated by this micro controller is added to the Micro User Envelope 1 value, to give fine variation in any such envelope.

Name

The name of the Voice.

Range

The maximum amount of micro change in the User Envelope 1 that can be applied. Zero means off (which is the default).

Change

The amount of micro change in the User Envelope that is applied by Noatikl between “update” periods. The value drifts between zero (off) and the Range, changing by plus or minus the Change value each time.

Update

The time in milliseconds between updates to the Micro User Envelope value. The actual value chosen is selected randomly each time, to be a value somewhere between Update and Update Range.

Update Range

The upper limit of time between updates to the Micro User Envelope value. The actual value chosen is betwen Update and Update Range.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

View: Voice - Envelope - User Envelope 2 (Pan)

This view allows you to define an envelope that is used to emit a MIDI CC of your choice. The default value for this MIDI CC is 10, which is the Pan controller.

Name

The name of the Voice.

Mute

The Mute parameter.

MIDI CC

Use this to define the MIDI CC that you want to be emitted by this envelope. The default value is 10, which is the Pan controller.

Enabled?

Use this to turn your envelope on or off.

Envelope

This parameter is shown as a graphical representation of the Pan envelope to be followed through playback of the Noatikl piece. The piece starts with the value at the left side of the envelope, and as the piece progresses, eventually the value from the far right of the envelope is used.

You can draw direct on to the envelope with your mouse.

Alternatively, you may use one of the various powerful envelope editing tools that we have made available to you.

To use the envelope editing tools:
- Right-click (win) or ctrl-click (mac) on the envelope tool
- Select the option you want. e.g. random, curve up etc.
- Select the range using the mouse...
- Then: either press space or enter, or select pop-up envelope tool to apply to the selected range.
- Select freehand mode to return to the normal click-to-paint mode.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Scale Rule

This view allows you to define how your Scale Rules behave.

Name

The name of the Scale Rule.

Value

Alter the height of each of the bars in the display, by clicking the mouse at the appropriate position.

The relative height of each bar, indicates the relative probability of a note being chosen for a Voice using this Rule, at the indicated semitone distance from the Root note defined for this Piece.

A bar with zero height, is one that will not be chosen by the Noatikl music engine.

The text on the the left of the bar (e.g. **m2**) defines the relative position in the scale rule governed by the that scale rule bar. The blue text on the right of the bar is a value from 0 to 100, proportionate to the bar height, which is especially useful for keeping tracks of bars that have small values which might no otherwise be visible.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Harmony Rule

This view allows you to define how your Harmony Rules behave.

Name

The name of the Harmony Rule.

Value

Alter the height of each of the bars in the display, by clicking the mouse at the appropriate position.

The relative height of each bar, indicates the relative probability of a note being chosen for a Voice, where the note being considered is the indicated semitone distance from a note already selected as being composed at the considered time for any Voice other than the one being composed for.

By way of example, imagine that you have three Voices, called X, Y and Z.

Imagine that at some time in the piece, Noatikl has already chosen to play note C for Voice X, and note G for Voice Y. At that time, Noatikl thinks about composing a note for Voice Z. It looks at the notes available for it to compose, and adjusts the probabilities of choosing each of those notes, by applying the Harmony Rule bar heights for each of those two composed notes which are active at time (i.e. notes C and G).

Harmonies are always calculated based on a rising direction up from the Piece Root. So, if Noatikl is using a Piece Root of B, and Noatikl is considering if it can compose note D for Voice Z, and it looks at the note it needs to harmonize with which is note C for Voice X; then Noatikl figures-out the Harmony Rule values for Voice Z from the C of Voice X, up and through the Octave (i.e. E, F, F#, G, G# etc.).

Confused? Well, it should all become clear when you start playing with it!

A bar with zero height, is one that will not be chosen by the Noatikl music engine.

The text on the the left of the bar (e.g. **m2**) defines the relative position in the harmony rule governed by that harmony rule bar. The blue text on the right of the bar is a value from 0 to 100, proportionate to the bar height, which is especially useful for keeping tracks of bars that have small values which might no otherwise be visible.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

View: Next Note Rule

This view allows you to define how your Next Note Rules behave.

Name

The name of the Next Note Rule.

Value

Alter the height of each of the bars in the display, by clicking the mouse at the appropriate position.

The relative height of each bar, indicates the relative probability of a note being chosen at the indicated semitone distance from the last note composed for a Voice using this rule. A bar with zero height, is one that will not be chosen by the Noatikl music engine.

The text on the the left of the bar (e.g. +**1**) defines the relative position in semitones goverbed by that next note rule bar. The blue text on the right of the bar is a value from 0 to 100, proportionate to the bar height, which is especially useful for keeping tracks of bars that have small values which might no otherwise be visible.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

View: Rhythm Rule

This view allows you to define how your Rhythm Rules behave.

Name

The name of the Rhythm Rule.

Value

Alter the height of each of the bars in the display, by clicking the mouse at the appropriate position.

The relative height of each bar, indicates the relative probability of a note duration being chosen for a Voice using this Rule.

A bar with zero height, is one that will not be chosen by the Noatikl music engine.

The note durations you have available to you are as follows. Note that we give various definitions of each duration, as different people have different names for the same thing!

The text on the the left of the bar (e.g. **1/4.**) defines the duration governed by that rhythm rule bar. The blue text on the right of the bar is a value from 0 to 100, proportionate to the bar height, which is especially useful for keeping track of bars that have small values which might no otherwise be visible.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Piece - Basics

This view allows you to define some of the key properties that govern your composition at the Piece level.

Name

Always shows the text "Piece" . You can't change this value!

Piece Length

The minimum duration, in seconds, for which to play this piece before restarting. The duration chosen is somewhere in the range of Piece Length, to Piece Length plus Piece Length Range; rounded up to the nearest bar boundary.

Piece Length Range

Defines an upper range in values, for which to play this piece before restarting

Piece Rest

The minimum duration, in seconds, for which this piece will rest with silence before restarting. The duration chosen by Noatikl is a randomly selected value in the range of Piece Rest, to Piece Rest plus Piece Rest Range; rounded up to the nearest bar boundary.

Piece Rest Range

Defines an upper range in values, for which this piece will rest with silence before restarting.

Auto Restart?

Defines if, when the piece has reached the end and applied any rest, it should restart automatically or not. If set to Yes, the Piece will keep playing with appropriate random variations. If set to No, the piece will simply stop playing at the end until you tell it to Stop.

Meter

Defines the Meter to be used by this Piece, such 4:4 or 3:4 or 6:8. A Voice will generally use this Meter, but the Meter value to be used for each Voice may actually override this setting. This approach allows Voices to be configured to work with a completely different Meter, which can be used for interesting polyphonic effects!

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Piece - Tempo

This view allows you to define the tempo properties for your Noatikl piece.

The tempo parameters are used in these two situations:
- when "cooking" a MIDI file from any Noatikl variant
- when "real-time" playing standalone and not synced

In all other circumstances, the Tempo parameters in this view are ignored! When not cooking a MIDI file, and when using the Noatikl Plug-in or Noatikl standalone in Sync? mode, the tempo is provided by the sequencer, and all of the Tempo parameters are ignored!

Name

Always shows the text "Piece" . You can't change this value!

Tempo

The minimum Tempo to use for this piece, in Beats Per Minute.

The actual Tempo to use is a value between Tempo, and Tempo plus Range.

Range

A value defining the maximum tempo to use, in Beats Per Minute.

The actual Tempo used by Noatikl is a randomly-selected value between Tempo, and Tempo plus Range.

Change

A value that defines if the tempo should change value according to an envelope defined by Envelope. When selected, the tempo value used for each bar is offset by the Envelope value scaled by the Envelope Range.

Envelope

A value showing the range of envelope value changes that this piece should be follow, used if Change is set to Yes. Use this if you want to apply a dynamic tempo envelope to your piece.

Envelope Range

A value defining what range of values are indicated by the Envelope. Defined in Beats Per minute.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Piece – Rules

This view allows you to define the default Rules used by the Noatikl Piece.

Name

Always shows the text "Piece" . You can't change this value!

Scale Rules

Set this to define the default Scale Rule to use when the piece plays. Individual Voices are allowed to override this setting if they so wish.

Harmony Rules

Set this to define the default Harmony Rule to use when the piece plays. Individual Voices are allowed to override this setting if they so wish.

Next Note Rules

Set this to define the default Next Note Rule to use when the piece plays. Individual Voices are allowed to override this setting if they so wish.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

View: Piece – Roots

This view allows you to define some of the key properties that govern your Noatikl composition.

Name

Always shows the text "Piece" . You can't change this value!!

Piece Roots

The Root Pitch to use for this Piece. For example, if you are using a Major Scale Rule, then set this value to be C, for you Piece to play in the key of C Major.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF

USER GUIDE

- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: Piece - Scripts

This view allows you to embed small Trigger Scripts, which are small bits of code in the widely Lua language, that are triggered when various events happen when Noatikl is playing. Using trigger scripts allows you to tell Noatikl to behave in very powerful ways while it is playing.

Name

Always shows the text "Piece" . You can't change this value!

Start

Press this button to show the Start trigger script in the Script Editor window.

The Start Trigger Script is called once at the start of the Piece, when the piece starts playing.

```
function nt_trigger_start()
  print ("Piece start!")
end
```

Bar

Press this button to show the Bar trigger script in the Script Editor window.

The Bar Trigger Script is called at the start of every bar while the piece is playing.

```
function nt_trigger_bar(bar)
  print ("Bar number", bar)
end
```

MIDI In CC

Press this button to show the MIDI In CC trigger script in the Script Editor window.

The MIDI In CC Trigger Script is called whenever a MIDI CC event is received by the MIDI Input device.

```
function nt_trigger_midiin_cc(channel, cc, value)
  print ("Piece MIDI In CC", channel, cc, value)
end
```

MIDI In Note

Press this button to show the MIDI In Note trigger script in the Script Editor window.

The MIDI In Note Trigger Script is called whenever a MIDI Note On or Off event is received by the MIDI Input device.

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
  print ("Piece MIDI in note ", noteon, channel, pitch, velocity)
end
```

Stop

Press this button to show the Stop trigger script in the Script Editor window.

The Stop Trigger Script is called once at the end of the Piece, just as the Piece stops playing.

```
function nt_trigger_stop()
  print ("Piece stop!")
end
```

See Also

- Scripting Overview
- Trigger Scripts
- Scripting Reference
- Noatikl as a Hyperinstrument

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

View: File

This view allows you to define some bookkeeping properties of your file. None of these properties affect the way that you piece sounds.

Name

An internal value representing your File. Always a value of File Details. You can't change this value!

Title

The title of your Piece.

Author

The author of your Piece.

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Patterns: Examples

Noatikl has its own pattern format, which allows it to play specified notes and rests in different ways. Patterns are affected by the Noatikl rules. Below are a number of example patterns. Copy and paste these into Noatikl to try them out:

Rhythm: <100 R 60 60 60.127 15>

Both: <100 B 60.15-30 1 60 2 60.127 3 15 7>

Forced: <100 F60 60.127 1 60 4 30 5 15.70-120 7>

Sequence: <S100.M R 1.1 2-0.1-0 3-0.1-0 >

Two patterns. Select randomly from these two each time!

<100 B 60 1 60 2 60 3 60 4>

<100 B 30 9 30 8 30 7 30 6 30 5 30 4 30 3 30 2>

Two sequenced sub-patterns. Play 1 once, then 2 once...

<S100 R 1.1 2.1>

<100 B 60 1 60 2 60 3 60 4>

<100 B 30 9 30 8 30 7 30 6 30 5 30 4 30 3 30 2>

Two sequenced sub-patterns. Play 1 twice, then 2 twice...

<S100 R 1.2 2.2>

<100 B 60 1 60 2 60 3 60 4>

<100 B 30 9 30 8 30 7 30 6 30 5 30 4 30 3 30 2>

Two sequenced sub-patterns. Play 1 or 2 twice, then 1 or 2 twice...

<S100 R 1-1.2 1-1.2>

<100 B 60 1 60 2 60 3 60 4>

<100 B 30 9 30 8 30 7 30 6 30 5 30 4 30 3 30 2>

Two sequenced sub-patterns. Play 1 once, then 2 twice, the one or 2 once, then 2 once...

<S100 R 1.1 2.2 1-1.1 2.1>

<100 B 60 1 60 2 60 3 60 4>

<100 B 30 9 30 8 30 7 30 6 30 5 30 4 30 3 30 2>

Two sequenced sub-patterns. Play 1 once, then 2 forever...

<S100 R 1.1 2.0>

<100 B 60 1 60 2 60 3 60 4>

<100 B 30 9 30 8 30 7 30 6 30 5 30 4 30 3 30 2>

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

Patterns: General Syntax

<[prob][.M] pattype {[dur][.vff[-vffr]] [scaleint]}*>

Where:

- prob : the relative probability that this sub-pattern is selected; relevant only where there is more than one sub-pattern! The default value is 100%.
- M : Flag indicating that the sub-pattern is to be "muted", i.e. not allowed to be selected. This can be useful for testing of individual sub patterns; where you might want to "solo" a sub-pattern by muting out all the others.
- Pattype : One of:
  - B: Both pattern
  - R: Rhythm pattern
- Froot: Force Frequency pattern type, followed by the root note in MIDIpitch. e.g. F60. Not yet implemented!
- dur : Note Duration.

Note in Noatikl a Beat is defined as being one crotchet; you get 4 beats in a bar of 4:4 music. So, a Duration value of 60 means one beat. A Duration value of 30 means a quaver. A Duration value of 20 means a triplet. A Duration value of 15 means a semi-quaver. A Duration value of 240 means 4 beats (which is a full bar if the Piece Meter is 4:4).

Note that in Rhythm Patterns, a negative duration indicates a rest for that time! If a note sub-pattern is not an exact even number of bars (e.g. 2 and and half bars at the current meter!) then the engine will pad to silence to the end of the nearest bar boundary.

vff : Velocity Force Factor percent (optional, from 0 [silent] to 100 [normal] and beyond; e.g. 120 would boost the velocity by 20%)

vffr Velocity Force Factor range (optional, used with the above.

scaleint: Scale interval (not present for Rhythm patterns).

B rule: interpreted as being the first valid note in current voices scale; i.e. the first element in the voice's current scale rule which does not have a zero value. "1" is therefore usually the root note (c.f. the Pitch parameter). “0” has the special meaning of indicating a “rest” note!

F rule: distance in semitones up from the root note.

B/F rules: a zero value indicates a REST for that time!

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Patterns: Sequence Sub Patterns

Examples

<S100 R 1.20 2.1 1-2.1-4 2.1>

Syntax

<[S][prob][.M] R {[seqnum[-seqnumrange].[repeattimes[-repeattimesrange]] [seqnum[-seqnumrange].[repeattimes[-repeattimesrange]]}* >

Where:

S : identifies a sequence sub-pattern

prob : relative probability of being chosen when there is more than one sequence sub-pattern.

M : Mute the sub-pattern (i.e. prevent it being selected!). If none can be selected, then a non-sequence sub-pattern is chosen to play at random as usual each time.

R : Rhythm pattern type (always required)

seqnum Sequence Number. The index of the non-sequence sub-pattern to play. Default is 1. The sub-patterns are numbered from 1 up!

seqnumrange : Sequence Number Range. Default is 0.

repeattimes : Repeat Times Minimum. The number of times to repeat this sub-pattern, when selected. A value of "0", will cause the sub-pattern (when selected) to keep playing forever until the end of the piece! Default is 1.

repeattimesrange Repeat Times Range. Default is 0.

Noatikl™ 1.5
Generative Music

Noatikl 1.5
Overview
Download
User Quotes
Recordings
Templates
FAQ
Generative Music
Recipes & Ideas
EULAs
Forum
User Guide PDF
USER GUIDE
Contents
General
Views - Voice
Views - Controllers
Views - Envelopes
Views - Rules
Views - Piece
Patterns
Scripting
Credits

Noatikl User Guide

back | next

Scripting: Overview

Noatikl contains a powerful scripting engine, which while entirely optional to use, offers tremendous power.

Introduction to Noatikl Scripting

Noatikl contains several very powerful scripting features, which can be used in two separate contexts.

The most common way to use scripting in Noatikl, it to embed small Trigger Scripts, which are small bits of code in the widely used Lua language, attached to the Piece or to Voice objects. These scripts are triggered when various events happen when Noatikl is playing. Using trigger scripts allows you to tell Noatikl to behave in very powerful ways while it is playing.

The other way to use scripting in Noatikl, is to use the Noatikl Script Window to create and run scripts written in the Lua language, to manipulate Noatikl editor windows. This is done through the Noatikl Script Window. This allows you to, for example, dynamically create random Noatikl compositions.

Noatikl scripts are written using the widely used Lua scripting langauge. Lua is very powerful, fast and easy to learn. Lua is a very popular embedded scripting langage, and is widely used in the gaming and multimedia worlds. Noatikl adds various noatikl-specific functions to the Lua language. These extensions are outlined later in this document. We give you lots of examples to get you started. If you ever get stuck, or if you want to share your cunning scripts with fellow Noatikl users, then please make use of the Noatikl forums!

Note that Noatikl supports the table, string and math standard Lua libraries. It does **not** support the *io* or *os* libraries.

Why would you want to use Noatikl Scripting?

With careful use of trigger scripts, you can use Noatikl to build a custom hyperinstrument!

You can, if you wish, create trigger scripts to run at the following times:

- start of playback
- every bar
- when a note is composed by noatikl
- end of playback
- in response to MIDI CC via e.g. keyboard controller
- in response to MIDI Note on/off via e.g. keyboard controller

Every script can do pretty much anything to the Noatikl engine in real-time, such as manipulating music rules, playing with mutation, and what have you.

We also have a new hyperinstrument mode, and allow the Note on/off commands to control parameters and/or trigger "listening" voices within Noatikl, that can in turn trigger off other musical events such as auto-chording events or following voices.

The scripts are in the very widely used Lua scripting language.

A note on *print*

Note that the standard *Lua* function *print* may be used to display text in the Noatikl console window (if you happen to have that displayed; this is very useful for debugging your scripts!

A note on Comments in Lua script

Single-line comments in Lua script start with the two minus characters:

```
-- This is a comment
```

To comment-out a block of text in Lua, enclose the text in --[[and]] as in this example

```
print ("Hello")
--[[
This is a big block
of text that I want to comment out!
]]
print ("World")
```

Trigger Scripts ("Triggers")

Noatikl Triggers can be assigned to most Noatikl objects, including the Piece and Voice objects.

Not all Noatikl script functions are available to you within Noatikl trigger scripts; specifically, the functions that open and manipulate windows are not available from Triggers (these are reserved for use from the Noatikl Script Window). You can enter these scripts using the following Noatikl views:

- Voice – Scripts
- Piece – Scripts

There are a variety of *Trigger Scripts* types that you may invoke.

Start

The Start Trigger Script is called once at the start of the Piece, when the piece starts playing.

```
function nt_trigger_start()
  print ("Piece start!")
end
```

Bar

The Bar Trigger Script is called at the start of every bar while the piece is playing.

```
function nt_trigger_bar(bar)
  print ("Bar number", bar)
end
```

Composed

The Composed Script is called when Noatikl composes a note. Only available to voice objects! Use this to emit MIDI CC events and what have you!

Advanced users might also be interested in reading about using this script with embedded calls to *noatikl_Trigger_EmitListeningNote*, to help build hyperinstruments.

```
function nt_trigger_composed(noteon, channel, pitch, velocity)
  print ("Composed", noteon, channel, pitch, velocity)
end
```

MIDI In CC

The MIDI In CC Trigger Script is called whenever a MIDI CC event is received by the MIDI Input device.

```
function nt_trigger_midiin_cc(channel, cc, value)
  if (cc == 1)
  then
    print ("MIDI In CC", channel, cc, value)
  end
end
```

NOTE on *Omni*: Noatikl delivers all MIDI CC events to this trigger from all MIDI channels, even if this trigger is a voice trigger script with a voice targeting a different MIDI channel. This is known as *Omni* behaviour. If you want your voice trigger to respond only if the supplied MIDI CC is from the same channel as your voice, use script like this:

```
function nt_trigger_midiin_cc(channel, cc, value)
  if (channel == noatikl_Trigger_GetMidiChannel())
  then
    -- Matches our voice channel!
    print ("MIDI In CC", channel, cc, value)

    if (cc == 1)
    then
      noatikl_Trigger_Parameter_Set("Patch", value)
    end
  end
end
```

MIDI In Note

The MIDI In Note Trigger Script is called whenever a MIDI Note On or Off event is received by the MIDI Input device. See the Noatikl hyperinstrument guide for a detailed discussion.

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
  print ("note ", noteon, channel, pitch, velocity)
end
```

NOTE on *Omni*: Noatikl delivers all MIDI Note events to this trigger from all MIDI channels, even if this trigger is a voice trigger script with a voice targeting a different MIDI channel. This is known as *Omni* behaviour. If you want your voice trigger to respond only if the supplied MIDI Note is from the same channel as your voice, use script like this:

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
 if (channel == noatikl_Trigger_GetChannel())
  then
    print ("note ", noteon, channel, pitch, velocity)
  end
end
```

Stop

The Stop Trigger Script is called once at the end of the Piece, just as the Piece stops playing.

```
function nt_trigger_stop()
  print ("Piece stop!")
end
```

The Script Editor Window

When you decide to edit a *Trigger Script* property, you are presented with a *Script Editor* window.

The text panels support the usual context-sensitive text editor menus and keyboard accelerators (allowing for fast copy and paste operations).

The Script Editor window allows you to edit trigger scripts associated with your object.

Test

Pressing this button will compile any Lua script that you type in the large text area at the top of the window, and display any results in the bottom panel. Use the *Test* to quickly check your script for obvious syntax errors!

Help

Pressing the *Help* button displays help on the Noatikl Scripting system!

Clear

Pressing the *Clear* button quickly erases the text in the top panel.

Default

Pressing the *Default* button will replace any text in the top panel, with a new, empty *default* trigger script appropriate to the Noatikl script property you are currently viewing.

OK

Pressing the *OK* button will save your changes. These are used by Noatikl it next starts playing.

Cancel

Pressing the *Cancel* button will discard your changes.

Test Results

Any test results are displayed in the *Test Results* area at the bottom.

Clear

Pressing the *Clear* button in the bottom area, quickly erases the text in the bottom panel.

Scripting Reference

To find out about all the available objects, parameters, and functions, you should refer to the complete Noatikl scripting reference.

Trigger Script Cookbook

A great way to start thinking about Noatikl trigger scripts, is to look at various real examples of how to do various interesting things with Noatikl trigger scripts!

Example: Change the scale to use depending on time of day

This is a script that you could use as a piece-level *Bar* trigger script.

```
  function nt_trigger_start()
    -- Noatikl script to get hour of day as 24-hour clock value
    -- ... and select scale accordingly!
    local lTime = tonumber(noatikl_GetDate("%H"))
    if ((lTime > 20) or (lTime < 8))
    then
      noatikl_Trigger_Parameter_Set("Scale Rules", "early morning scale")
    else
      noatikl_Trigger_Parameter_Set("Scale Rules", "middle of day scale")
    end
  end
```

Example: randomly select a pattern to use for a voice

This is a voice-level *Bar* trigger script.

```
function nt_trigger_start()
  -- Get a random integer between 1 and 3 inclusive.
  local lValue = noatikl_GetRandomFromTo (1, 3)

  print ("lValue", lValue)

  local lPattern = ""
  if (lValue == 1)
  then
    lPattern = "<100 B 240 1>"
  elseif (lValue == 2)
  then
    lPattern = "<100 B 120 1 120 2>"
  else
    lPattern = "<100 B 60 1 60 2 60 3 60 4>"
  end

  noatikl_Trigger_Parameter_Set("Patterns", lPattern)
end
```

Example: script trigger that allows Noatikl trigger scripts to emit MIDI CC events, with an optional delay

This is a script that you could use as a piece-level *Bar* trigger script.

```
  function nt_trigger_bar(bar)

    -- Voice trigger...
    -- Apply a pan sweep through the bar from left to right,
    -- to show-off the use of noatikl_Trigger_EmitMidiCC.

    local lDuration = noatikl_Trigger_GetBarDuration()

    local lMidiChannel = noatikl_Trigger_GetChannel()
    local lDelay = 0
    local lCC = 10 -- Pan controller!
    local lValue = 0
    while lDelay <= lDuration do
      local lValue = (127 * lDelay) / lDuration
      -- Note that the "lDelay" is optional; we're using the in this specific
      -- demo, to get a sweep effect from start to end of the bar.
      noatikl_Trigger_EmitMidiCC(lMidiChannel, lCC, lValue, lDelay)
      lDelay = lDelay + 20
    end
  end
```

Example: piece bar trigger, that automatically adjusts the piece root every 10 bars

```
function nt_trigger_bar (bar)

  if ((bar % 10) == 0)
  then
    -- Every 10 bars, change the root at random,
    -- from all those available!
    local lRoots = {"A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"}
    local lIndex = noatikl_GetRandomFromTo(1,12)
    local lRoot = lRoots[lIndex]
    print ("Root", lRoot)
    noatikl_Trigger_Parameter_Set("Piece Roots", lRoot)

    -- Un-comment the following line to set tempo at random, too,
    -- to a value related to the selected root!
    -- noatikl_Trigger_Parameter_Set("Tempo", 100 + lIndex * 5)
  end
end
```

Example: piece start trigger, that automatically adjusts the tempo to suit the time of day

```
function nt_trigger_start()
  print("Piece!")

  -- Noatikl script to get hour of day as 24-hour clock value
  -- ... and adjust tempo accordingly!
  local lTime = tonumber(noatikl_GetDate("%H"))
  if ((lTime > 20) or (lTime < 8))
  then
    print ("late night / early morning!")
    local lRandom = noatikl_GetRandomFromTo(0,50)
    noatikl_Trigger_Parameter_Set("Tempo", 50 + lRandom)
  else
    print ("middle of day!")
    local lRandom = noatikl_GetRandomFromTo(0,50)
    noatikl_Trigger_Parameter_Set("Tempo", 100 + lRandom)
  end
end
```

Example: voice start trigger, that automatically cycles the patch every time it starts playback

```
function nt_trigger_start()
  print("Voice!")

  local lPatch = noatikl_Trigger_Parameter_Get_AsNumber("Patch")
  lPatch = lPatch + 1
```

```
  if (lPatch > 100)
  then
    lPatch = 0
  end
  noatikl_Trigger_Parameter_Set("Patch", lPatch)
end
```

Example: voice trigger, that automatically changes patch every bar

```
function nt_trigger_bar(bar)
  print("Voice1 ", bar)
  local lPatch = noatikl_Trigger_Parameter_Get_AsNumber("Patch")
  lPatch = lPatch + 1
  if (lPatch > 100)
  then
    lPatch = 0
  end
  noatikl_Trigger_Parameter_Set("Patch", lPatch)
end
```

Example: bringing voices into the mix

In this example, imagine that you have a mutating drum Voice that you want to have muted at the start of the piece and come in with a bang at the start of bar 20. And you don't want the mutations to start until bar 21...

How would that work in the world of script?

Here is the Voice Start trigger:

```
function nt_trigger_start()
  -- When we start, disable mutation for this voice!
  -- And set mute, too!
  noatikl_Trigger_Parameter_Set("Mute", "Yes")
  noatikl_Trigger_Parameter_Set("Mutation Factor", 0)
end
```

Here is the Voice Bar trigger:

```
function nt_trigger_bar(bar)
  if (bar == 20)
  then
    -- At bar 20, unmute the voice!
    noatikl_Trigger_Parameter_Set("Mute", "No")
  end

  if (bar == 21)
  then
    -- At bar 21, adjust the mutation factor!
    noatikl_Trigger_Parameter_Set("Mutation Factor", 20)
  end
end
```

The power of using scripts like this, is that:

- The drums alway start on the bar specified.
- They don't morph until they are audible.
- The drum elements can now make staggered entry despite all bing on the same midi channel.

Example: Emit MIDI CC in response to composed note events

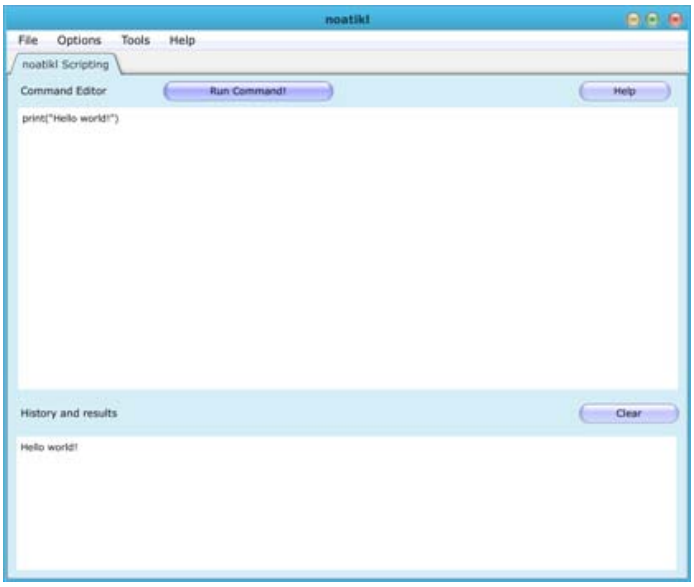
```
function nt_trigger_composed(noteon, channel, pitch, velocity)
  -- Keep track of last composed note using the global (for this voice)
  -- called GLastNote ...
  -- Note that the "pitch" parameter should be ignored when noteon is false.

  lMidiChannel = noatikl_Trigger_GetChannel()

  if (noteon)
  then
    if (GLastNote ~= nil)
    then
      noatikl_Trigger_EmitMidiCC(lMidiChannel, 11, GLastNote, 10)
    end

    GLastNote=pitch
    noatikl_Trigger_EmitMidiCC(lMidiChannel, 11, pitch, 0)
  else
    if (GLastNote ~= nil)
    then
      noatikl_Trigger_EmitMidiCC(lMidiChannel, 11, GLastNote, 10)
      GLastNote = nil
    end
  end
end
```

Noatikl Script Window (the Noatikl Script Console)

The Noatikl Script Window is used to create and run scripts to manipulate Noatikl editor windows; for example, to dynamically create random Noatikl compositions.

It is also used for displaying messages from *print* calls within your trigger scripts!

The text panels support the usual context-sensitive text editor menus and keyboard accelerators (allowing for fast copy and paste operations).



Run Command!

Pressing this button will run any Lua script that you type in the large text area at the top of the window, and display any results in the bottom panel

An example you might like to try would be this:

```
print ("Hello world!")
```

History and Results

Any results are displayed in the *History and Results* area at the bottom.

This is where the output from any *print* statements in your scripts are displayed (including those within trigger scripts).

Help

Pressing the *Help* button displays help on the Noatikl Scripting system!

Clear

Pressing the *Clear* button quickly erases the bottom panel.

Console Script Cookbook

A great way to start thinking about Noatikl console scripting, is to look at various real examples of how to do various interesting things with Noatikl console scripting!

Example: the simplest possible start

```
print ("Hello world")
```

Example: open all Noatikl files in a sub-folder with a .txt extension, adjust some properties and re-save with .Noatikl file extension.

```
local lFileList = noatikl_FindFilePaths("myfolder1", "myfolder2")

local index = 0
while true do

  index = index + 1
  local lFilePath = lFileList[index];
  if (lFilePath == nil)
  then
    break
  end

  if (string.find(lFilePath, '.txt'))
  then
    local lWindow = noatikl_Window_OpenPath(lFilePath)

    -- Change some file properties
    noatikl_Window_Object_Parameter_Set(lWindow, "File", 1, "Author", "my company")
    noatikl_Window_Object_Parameter_Set(lWindow, "File", 1, "Midi Output Device", "?")
    noatikl_Window_Object_Parameter_Set(lWindow, "File", 1, "Midi Input Device", "?")
    noatikl_Window_Object_Parameter_Set(lWindow, "File", 1, "Midi Sync?", "No")

    -- Change some properties for every voice in the file...
    local lCount = noatikl_Window_Object_GetCount(lWindow, 'Voice')

    local lIndex = 1
    while lIndex <= lCount do

      noatikl_Window_Object_Parameter_Set(lWindow, "Voice", lIndex, "Copyright", "Copyright my
company")

      lIndex = lIndex + 1
    end

    local lSaveToPath = string.gsub(lFilePath, '.txt', '.noatikl')

    noatikl_Window_SaveToPath(lWindow, lSaveToPath)

    print ("Saved", lFilePath)

    noatikl_Window_Close(lWindow)
```

```
  end
end
print ('Done!')
```

See Also

- Piece – Scripts
- Voice – Scripts
- Scripting Reference
- Noatikl as a Hyperinstrument

Noatikl™ 1.5 Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Scripting: Noatikl as a Hyperinstrument

Noatikl can be used as a customisable hyperinstrument.

You might also want to refer to the scripting overview and the scripting reference.

What is a hyperinstrument?

A hyperinstrument can be defined as being an instrument that can have extraordinary, semi-automated response to relatively simple real-time inputs.

Noatikl is a hyperinstrument in many ways. For example from simple interaction in real-time through mouse and keyboard, you can have it generate extraordinary, ever-changing music that responds to your inputs. Noatikl also directly responds to any real-time MIDI note data fed into it, for example from a MIDI keyboard or other MIDI controller; where you feed MIDI data into Noatikl, you will find that Noatikl automatically harmonises its voices with all incoming MIDI note data. You can also configure Noatikl to respond in many advanced ways to both MIDI note events and MIDI control events (otherwise known as "*MIDI CCs*").

So, Noatikl is a hyperinstrument for a number of reasons; Noatikl directly harmonises with incoming MIDI note events, and noatikl's generative engine is an instrument that creates hugely more than the sum of your direct inputs with keyboard, mouse, and MIDI. In other words, noatikl's generative engine, MIDI control, scripting and direct MIDI input harmonisation all work together to make Noatikl itself greater than the sum of its parts; a **customisable generative hyperinstrument**!

See the Noatikl Hyperinstrument - Quick Start if you want to get started quickly!

How does Noatikl work as a hyperinstrument?

You can respond to any incoming MIDI CC however you want; for example, to change the current scale or adjust a harmony rule in real-time in response to incoming MIDI CC events!

You can also respond to any incoming MIDI Note on/off event however you want; for example, you could use some notes (such as accidentals in a scale) to change parameter values, and pass-through the others to the Noatikl engine. In other words, you can use certain specific notes to trigger various complex responses by Noatikl, for example; or to enable/disable the chording properties only for certain notes (immediately that a note is triggered!); or to 'fix' "invalid" notes; or to change scale; or pretty much whatever you can imagine!

All note on/off events received by Noatikl via the MIDI input device (or plug-in input) are normally passed through automatically to the destination MIDI output device (or plug-in output); with the exception that where you have a Listening voice with a Note On trigger script, the note on/off events for that voice's MIDI channel are **not** immediately passed-through by noatikl; in this case, the only way to get them out is via a call to noatikl_Trigger_EmitListeningNote within the nt_trigger_midiin_note trigger script for a voice! Note that the noatikl_Trigger_EmitListeningNote will work *only* if your voices is a Listening voice, *and* if Listen? is ticked.

To re-iterate, for lines that are not being listened to with a voice that has a Note Trigger Script: the any note on/off events effectively bypass noatikl's internal engine and are passed-on with minimum latency (though the Noatikl engine does harmonize against them). On lines that are marked as listening voice type, and where there is a note trigger script with embedded, then you'd need to have a call to this function within that script:

```
noatikl_Trigger_EmitListeningNote(noteon, pitch, velocity)
```

which results in note on/off events on that line being sent via noatikl's internal composition engine.

Here is real, working example of a Voice MIDI In Note trigger script that you could use in your own Noatikl piece. Note that the call to noatikl_Trigger_EmitListeningNote will have an effect only if the voice is a Listening voice:

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
  if (channel == noatikl_Trigger_GetChannel())
  then
    -- Simply pass-through all notes to the listening voice.
    -- If we wanted to, we could respond to certain pitches to do different
    -- things!
    noatikl_Trigger_EmitListeningNote(noteon, pitch, velocity)
  end
end
```

When you call this function, this causes the note to be emitted as though the listening voice were "following" that note in accordance with the "following voice" parameters. So, if you want the exact note to be passed through, then using the follows strategy of semitone shift and set the offset to be zero.

The note emitted by Noatikl after it applys the "following" rules, is auto-chorded (if you have this set-up); and

any Noatikl voice that you have configured to follow this listening voice, will follow that note.

The above comments apply if you play into the listening voice with a monophonic feed. If you're supplying a polyphonic feed, then slightly more complex rules apply. In this case, only the first note in any ongoing chord from the controller is treated with the following voice parameters by noatikl; all other notes active in any chord are harmonised with, and passed out directly to the MIDI stream. So as you play solo/chords on your controller, the system caters for that polyphony in a good way; the first active note in the chord is processed by Noatikl (and auto-chorded or followed as required), and all other notes in the chord are passed straight through with minimal latency (and are, of course, harmonized with by noatikl!).

Note: we recommend use of noatikl_Trigger_EmitListeningNote with the Standalone version of noatikl: some sequencer hosts can too much latency to make this mode effective with the plug-in (Reaper is effective!). And note, that if using this mode with the plug-in, it is likely to work best when getting input from a MIDI port or direct from the device, and outputting data to a port (e.g. MIDI Yoke or IAC port). If you find otherwise, please let us know!

Note that the MIDI input could come from MIDI backing track, in sequencer, instead of course; depending on what routing options are available to you in your host sequencer.

Noatikl Hyperinstrument - Quick Start

To have a voice "listen" to incoming MIDI data, you need to follow the instructions below.

Note: you can simply select *File -> New* and select *Standard Templates -> Hyperinstrument*, to automatically create a new file which contains the following settings; you'll first have to change the MIDI output/input devices to suit your requirements. The created ppiece assumes your instrument is on MIDI channel 1; the first voice is a listening voice set to auto-chord the first note of any chord you play on your MIDI input device; having first applied any pitch shift required through the following voice parameters. It also has a couple of extra voices to harmonize along with your input.

- define a MIDI input device
- tick the Listen? checkbox (please follow the link and read the comments about MIDI feedback!)
- set a Noatikl voice to be the one to "listen" to your guitar by setting the voice type to Listening (this makes it a "Listening Voice")
- select Voice - Scripts
- set this to be your voice's MIDI In Note trigger script:

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
  if (channel == noatikl_Trigger_GetChannel())
  then
    -- Simply pass-through all notes to the listening voice.
    -- If we wanted to, we could respond to certain pitches
    -- to do different things!
    noatikl_Trigger_EmitListeningNote(noteon, pitch, velocity)
  end
end
```

Harmonising with a MIDI file - an example using Cubase SE 3

The following example (tested with Cubase SE 3 on Windows) shows how to use Noatikl standalone to harmonise with a MIDI file held in Cubase SE 3.

Cubase

1. Start Cubase
2. Select *File -> Import -> MIDI File*, and select a a "type 0" MIDI file (which will load all data into one track), as that is by far the easiest to work with.
3. Set the "in:" to be "Not Connected" ... this is to prevent MIDI feedback, which would crash Cubase!
4. Set the the "out:" to be MIDI Yoke NT: 2 (for example)
5. Ensure that your MIDI Sync options are to emit MIDI Sync on MID Yoke NT 2. (*Transport -> Sync Setup*)
6. Press play; you should hear nothing as the MIDI data isn't going to any synth. Press Stop, and move on to prepare your Noatikl piece for MIDI harmonisation!

noatikl

1. Start noatikl
2. Create a simple piece, maybe with just 1 voice to start with (or as many as you want, I guess). You probably want to assign to MIDI channels not used by your MIDI file.
3. Set MIDI output to Microsoft GM Wavetable Synth
4. Set MIDI input to MIDI Yoke 2
5. Tick both "Sync?" and "Listen?"
6. Press Play (Noatikl will then wait for you to press the Cubase play button)

Cubase

1. Press the play button on the transport control - Noatikl will start playing all the data from the MIDI file, and merge-in its own composed data that harmonises with the incoming MIDI file. Excellent!

noatikl

1. While it is all playing, go back to Noatikl and look at the "Harmony Rule" view, just to show visually that Noatikl is harmonising with the MIDI input data from the MIDI file.

Cubase - crashes on saving/re-opening the project

If I save this Cubase project and re-open it, the "in:" setting is not saved by Cubase. If I try changing this, my copy of cubase *always* crashes(!), presumably as it has caused itself someway to suffer from MIDI feedback; we hope this doesn't happen with your copy of Cubase!

See Also

- Piece - Scripts
- Voice - Scripts
- Scripting Overview
- Trigger Scripts
- Scripting Reference

Noatikl™ 1.5
Generative Music

- Noatikl 1.5
- Overview
- Download
- User Quotes
- Recordings
- Templates
- FAQ
- Generative Music
- Recipes & Ideas
- EULAs
- Forum
- User Guide PDF
- USER GUIDE
- Contents
- General
- Views - Voice
- Views - Controllers
- Views - Envelopes
- Views - Rules
- Views - Piece
- Patterns
- Scripting
- Credits

Noatikl User Guide

back | next

Scripting: Reference

Noatikl contains many powerful scripting features. This page lists all the available functions, objects, and parameters.

Scripting Function Reference

Note that Noatikl supports the table, string and math standard Lua libraries. It does **not** support the *io* or *os* libraries.

A note on *print*

Note that the standard *Lua* function *print* may be used to display text in the Noatikl console window (if you happen to have that displayed; this is very useful for debugging your scripts!

The following functions can be used from any Noatikl script.

noatikl_GetRandom ()

This function returns a random integer.

Example:

```
local lValue = noatikl_GetRandom ()
```

noatikl_GetRandomFromTo (minimum, maximum)

This function returns a random integer in the range of the two supplied values.

The function has two arguments:

- *minimum* the minimum value that can be returned
- *maximum* the maximum value that can be returned

Example:

```
-- Get a random integer between 0 and 50 inclusive.
local lValue = noatikl_GetRandomFromTo (0, 50)
```

noatikl_GetTime ([table])

Returns the current time. It works the same way as the the `Lua os.time` function.

noatikl_GetDate ([format [, time])

Returns the current date. It works the same way as the the `Lua os.date` function.

Example:

```
-- Noatikl script to get hour of day as 24-hour clock value
-- ... and adjust tempo accordingly!
local lTime = tonumber(noatikl_GetDate("%H"))
if ((lTime > 20) or (lTime < 8))
then
  print ("late night / early morning!")
  local lRandom = noatikl_GetRandomFromTo(0,50)
  noatikl_Trigger_Parameter_Set("Tempo", 50 + lRandom)
else
  print ("middle of day!")
  local lRandom = noatikl_GetRandomFromTo(0,50)
  noatikl_Trigger_Parameter_Set("Tempo", 100 + lRandom)
end
```

The following functions can *only* be used from any Noatikl trigger script

noatikl_Trigger_Parameter_Set (parameter_name, newvalue)

This function is used to set the object's parameter to the supplied value, for the object within whose trigger script you place this call.

The function has two arguments:

- *parameter_name* the name of the parameter for which you want to set the value.
- *newvalue* the new value to use.

Example:

```
function nt_trigger_start()
  -- Set the current patch (assuming we make this call from within a
  -- Voice's trigger script!)
  local lPatch = 20
  noatikl_Trigger_Parameter_Set("Patch", lPatch)
end
```

---

noatikl_Trigger_Parameter_Get (parameter_name)

This function returns the value of the named parameter, for the object within whose trigger script you place this call. Note that the value returned is always a *string*, which you can convert to a number using *tonumber()*.

The function has one argument:

- *parameter_name* the name of the parameter for which you want to determine the current value.

Example:

```
function nt_trigger_start()
  -- Get the current patch (assuming we make this call from within a
  -- Voice's trigger script!)
  local lPatch = noatikl_Trigger_Parameter_Get("Patch")
  print("Patch", lPatch)
end
```

---

noatikl_Trigger_Parameter_Get_AsNumber (parameter_name)

This function returns the value of the named parameter, for the object within whose trigger script you place this call. Note that the value returned is always a *number*, so you should use this call only when you know that the parameter in question can be treated as a number.

The function has one argument:

- *parameter_name* the name of the parameter for which you want to determine the current value.

Example:

```
function nt_trigger_start()
  -- Get the current patch as a number (assuming we make this call from within a
  -- Voice's trigger script!)
  local lPatch = noatikl_Trigger_Parameter_GetAsNumber("Patch")
  print("Patch", lPatch)
end
```

---

noatikl_Trigger_GetChannel()

This function returns the MIDI channel (assuming the script is called from within a voice script!). The value returned is a value between 1 and 16.

Example:

```
function nt_trigger_start()
  local lChannel = noatikl_Trigger_GetChannel ()
  print ("Channal", lChannel)
end
```

---

noatikl_Trigger_GetRuleElement (object_type, object_name, element_index)

This function returns the rule element value of the specified index, for the named object. The value returned is a value between 0 and 127.

The function has the following arguments:

- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *element_index* the index of the element; starting at zero for the initial element.

Example:

```
function nt_trigger_start()
  -- Dump out the scale rule element values, for the major scale.
  local lIndex = 0
  while (lIndex < 12)
  do
    local lValue = noatikl_Trigger_GetRuleElement ("Scale Rule", "Major", lIndex)
    print ("lValue", lValue)
    lIndex = lIndex + 1
  end
end
```

---

noatikl_Trigger_SetRuleElement (object_type, object_name, element_index, newvalue)

This function set the rule element value of the specified index, for the named object, to the specified value. The

value must be between 0 and 127.

The function has the following arguments:

- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *element_index* the index of the element; starting at zero for the initial element.
- *newvalue* the new element value to use, in a range from 0 to 127.

Example:

```
function nt_trigger_start()
  -- Set the scale rule element values, to odd settings (!), for the major scale.
  local lIndex = 0
  while (lIndex < 12)
  do
    noatikl_Trigger_SetRuleElement ("Scale Rule", "Major", lIndex, (lIndex * 127) / 12)
    lIndex = lIndex + 1
  end
end
```

---

noatikl_Trigger_EmitListeningNote(noteon, pitch, velocity)

This function emits the specified note via the current voice; the function is allowed only within a nt_trigger_midiin_note trigger script. See the Noatikl hyperinstrument guide for a detailed discussion. Note that this function will have an effect only if will work *only* if your voices is a Listening voice, *and* if Listen? is ticked.

The function has the following arguments:

- *noteon* *true* for a note on event, *false* for a note off event.
- *channel* the MIDI channel to use; from 1 to 16.
- *pitch* the MIDI pitch to use; from 0 to 127.
- *velocity* the MIDI velocity to use, from 0 to 127.

Example:

```
function nt_trigger_midiin_note(noteon, channel, pitch, velocity)
  if (channel == noatikl_Trigger_GetChannel())
  then
    noatikl_Trigger_EmitListeningNote(noteon, pitch, velocity)
  end
end
```

---

noatikl_Trigger_EmitMidiCC (channel, cc, value [, delay])

This function emits the specified MIDI CC event.

The function has the following arguments:

- *channel* the MIDI channel to use; from 1 to 16.
- *cc* the MIDI CC to use; from 0 to 127.
- *value* the controller value to use, from 0 to 127.
- *delay* an optional delay to use, from 0 up, defaulting to 0; in Noatikl pattern time units (where 60 represents one crotchet or quarter note). The delay is relative to the current Noatikl timebase relevant to the trigger script in question.

Example:

```
function nt_trigger_bar(bar)
  -- Voice trigger...
  -- Apply a pan sweep through the bar from left to right,
  -- to show-off the use of noatikl_Trigger_EmitMidiCC.

  local lDuration = noatikl_Trigger_GetBarDuration()

  local lMidiChannel = noatikl_Trigger_GetChannel()
  local lDelay = 0
  local lCC = 10 -- Pan controller!
  local lValue = 0
  while lDelay <= lDuration do
    local lValue = (127 * lDelay) / lDuration
    -- Note that the "lDelay" is optional; we use this in this specific
    -- demo, to get a sweep effect from start to end of the bar.
    noatikl_Trigger_EmitMidiCC(lMidiChannel, lCC, lValue, lDelay)
    lDelay = lDelay + 20
  end
end
```

---

noatikl_Trigger_GetBarDuration ()

This function returns the duration of the piece bar, in Noatikl pattern time units; where 240 represents four crotchets or one whole note or a bar of 4:4 time.

Example:

```
function nt_trigger_start()
  local lDuration = noatikl_Trigger_GetBarDuration()
  print ("lDuration", lDuration)
end
```

---

noatikl_Trigger_Object_GetCount (object)

Returns the number of objects of the specified type, in the current Noatikl file.

The function has the following arguments:

- *object* the object of interest.

Example:

```
function nt_trigger_start()
  lCount = noatikl_Trigger_Object_GetCount('Voice')
  print ('Voice objects=', lCount)
end
```

---

noatikl_Trigger_Object_GetName (object_type, object_index)

Returns the name of the specified object.

The function has the following arguments:

- *object_type* the type of the object of interest.
- *object_index* the index of the object of interest.

Example:

```
function nt_trigger_start()
  local lCount = noatikl_Trigger_Object_GetCount('Voice')
  print ('Voice objects=', lCount)
  local lIndex
  while (lIndex <= lCount)
  do
    print (noatikl_Trigger_Object_GetName('Voice', lIndex)
    lIndex = lIndex + 1
  end
end
```

---

noatikl_Trigger_Object_Parameter_Get (object_type, object_index, parameter_name)

Returns the value of the parameter, for the specified object index of the specified object type, in the playing Noatikl file.

The function has the following arguments:

- *object_type* the type object of interest.
- *object_index* the index of the object of interest.
- *parameter_name* the parameter of interest.

Example:

```
function nt_trigger_start()
  local lVal = noatikl_Trigger_Object_Parameter_Get("File", 1, "Author")
  print ('Author=', lVal)
end
```

---

noatikl_Trigger_Object_Parameter_Set (object_type, object_index, parameter_name, newvalue)

Sets the parameter to the the supplied value, for the specified object index of the specified object type, in the playing Noatikl file.

The function has the following arguments:

- *object_type* the type object of interest.
- *object_index* the index of the object of interest.
- *parameter_name* the parameter of interest.
- *newvalue* the new value to use.

Example:

```
function nt_trigger_start()
  noatikl_Trigger_Object_Parameter_Set("File", 1, "Author", "John Smith")
end
```

---

noatikl_Trigger_GetRuleElement (object_type, object_name, element_index)

This function returns the rule element value of the specified index, for the named object. The value returned is a value between 0 and 127.

The function has the following arguments:

- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *element_index* the index of the element; starting at zero for the initial element.

Example:

```
function nt_trigger_start()
  local lCount = noatikl_Trigger_Object_GetCount(l'Scale Rule')
  print ('Scale Rule objects=', lCount)

  local lIndex = 1
  while (lIndex <= lCount)
  do
    -- Dump out the scale rule element values, for the scale!
    local lName = noatikl_Trigger_Object_GetName('Scale Rule', lIndex)
    print ('Scale', lName)

    local lItemIndex = 0
    while (lItemIndex < 12)
    do
      local lValue = noatikl_Trigger_GetRuleElement ("Scale Rule", lName, lItemIndex)
      print ("lItemIndex", lItemIndex, "lValue", lValue)
      lItemIndex = lItemIndex + 1
    end

    lIndex = lIndex + 1
  end
end
```

---

noatikl_Trigger_SetRuleElement (object_type, object_name, element_index, newvalue)

This function set the rule element value of the specified index, for the named object, to the specified value. The value must be between 0 and 127.

The function has the following arguments:

- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *element_index* the index of the element; starting at zero for the initial element.
- *newvalue* the new element value to use, in a range from 0 to 127.

Example:

```
function nt_trigger_start()
  local lCount = noatikl_Trigger_Object_GetCount('Scale Rule')
  print ('Scale Rule objects=', lCount)

  local lIndex = 1
  while (lIndex <= lCount)
  do
    -- Set the scale rule element values, for the scale, to stupid values!
    local lName = noatikl_Trigger_Object_GetName('Scale Rule', lIndex)
    print ('Scale', lName)

    local lItemIndex = 0
    while (lItemIndex < 12)
    do
      local lValue = (lItemIndex * 127) / 12
      noatikl_Trigger_SetRuleElement ("Scale Rule", lName, lItemIndex, lValue)
      lItemIndex = lItemIndex + 1
    end

    lIndex = lIndex + 1
  end
end
```

---

noatikl_Trigger_GetEnvelopePercent (object_type, object_name, par_name, percent)

This function returns the percent value for the specified envelope at the given percent position. The value returned is a value between 0 and 127.

The function has the following arguments:

- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *par_name* the name of the envelope parameter (e.g. *Volume*).
- *percent* the percent value; from 0 to 100.

Example:

```
function nt_trigger_start()
  local lCount = noatikl_Trigger_Object_GetCount('Voice')
  print ('Voice objects=', lCount)

  local lIndex = 1
  while (lIndex <= lCount)
  do
    -- Dump out the voice volumes envelopes!
    local lName = noatikl_Trigger_Object_GetName('Voice', lIndex)
    print ('Voice', lName)

    local lPercent = 0
    while (lPercent <= 100)
    do
      local lValue = noatikl_Trigger_GetEnvelopePercent ("Voice", lName, "Volume", lPercent)
      print ("Volume percent", lPercent, "value", lValue)
      lPercent = lPercent + 1
    end

    lIndex = lIndex + 1
  end
end
```

---

noatikl_Trigger_SetEnvelopePercent (object_type, object_name, par_name, percent, newvalue)

This function sets the envelope value at the specified percent, for the named object, to the specified value. The

value must be between 0 and 127.

The function has the following arguments:

- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *par_name* the name of the envelope parameter (e.g. *Volume*).
- *percent* the percent value; from 0 to 100.
- *newvalue* the new element value to use, in a range from 0 to 127.

Example:

```
function nt_trigger_start()
  local lCount = noatikl_Trigger_Object_GetCount('Voice')
  print ('Voice objects=', lCount)

  local lIndex = 1
  while (lIndex <= lCount)
  do
    -- Ramp-up every voice volume envelope!
    local lName = noatikl_Trigger_Object_GetName('Voice', lIndex)
    print ('Voice', lName)

    local lPercent = 0
    while (lPercent <= 100)
    do
      local lValue = (lPercent * 127) / 100
      noatikl_Trigger_SetEnvelopePercent ("Voice", lName, "Volume", lPercent, lValue)
      lPercent = lPercent + 1
    end

    lIndex = lIndex + 1
  end
end
```

The following functions can *only* be used from the Noatikl script console window.

---

noatikl_Window_New ()

Creates a new Noatikl file; the user is first prompted with the template selection dialog. Returns a file handle that you must use in other related function calls; returns 0 in the case of an error.

Example:

```
  -- Create a new file Window, and keep it open!
  local lWindow = noatikl_Window_New()
```

---

noatikl_Window_OpenPath ([filepath])

Opens the Noatikl file from the specified path. Returns a file handle that you must use in other related function calls; returns 0 in the case of an error.

The function has the following arguments:

- *filepath* the path of the Noatikl file to open. If this argument is not supplied, the user is first presented with a file browser to select a file to open.

Example:

```
  -- Open a file Window, and keep it open!
  local lWindow = noatikl_Window_OpenPath("/myfolder/myfile.noatikl")
```

---

noatikl_Window_Close (window)

Closes the Noatikl file window returned by an earlier call to noaikl_Window_Open.

The function has the following arguments:

- *window* the window handle of the file to be closed.

Example:

```
  -- Open a file Window
  local lWindow = noatikl_Window_OpenPath("/myfolder/myfile.noatikl")
  -- and immediately close it!
  noatikl_Window_Close(lWindow)
```

---

noatikl_Window_GetCount (window)

Returns the number of currently open Noatikl file windows.

Example:

```
local lCount = noatikl_Window_GetCount()
print ("lCount", lCount)
```

---

noatikl_Window_GetWindow (index)

Returns the window handle for the specified window index; returns 0 if no window exists for that index.

The function has the following arguments:

- *index* the index of the window for which the handle is being queried.

Example:

```
local lCount = noatikl_Window_GetCount()
print ("lCount", lCount)

local lIndex = 1
while (lIndex <= lCount)
do
  local lWindow = noatikl_Window_GetWindow(lIndex)
  print (lWindow)
  local lPath = noatikl_Window_GetPath(lWindow)
  print (lPath)
  lIndex = lIndex + 1
end
```

---

noatikl_Window_GetPath (window)

Returns the file path used by the window in question.

The function has the following arguments:

- *window* the window handle of the file being queried.

Example:

```
-- Run through all open Noatikl file windows,
-- and print out the file paths.
local lCount = noatikl_Window_GetCount()

local lIndex = 1
while (lIndex <= lCount)
do
  local lWindow = noatikl_Window_GetWindow(lIndex)
  local lPath = noatikl_Window_GetPath(lWindow)
  print (lPath)
  lIndex = lIndex + 1
end
```

---

noatikl_Window_SaveToPath (window, path)

Save the Noatikl file window returned by an earlier call to noaikl_Window_Open, to the specified path.

The function has the following arguments:

- *window* the window handle of the file to be saved.
- *path* the path where the file should be saved.

Example:

```
  -- Try to save the file with a different name... immediately close it!
  noatikl_Window_SaveToPath("/myfolder/myfilenew.noatikl")
```

---

noatikl_Window_FindFilePaths (subfolder [, subfolder2]

Find all Noatikl files relative to the Noatikl document root. Return as an array of file paths.

The function has the following arguments:

- *subfolder* the subfolder under which to search.
- *subfolder2* option subfolder to search under the subfolder.

Example:

```
local lFileList = noatikl_FindFilePaths("templates", "timdidymus1")

local index = 0
while true do

  index = index + 1
  local lFilePath = lFileList[index];
  if (lFilePath == nil)
  then
    break
  end

  print ("File path", lFilePath)
end
```

---

noatikl_Window_Object_GetCount (window, object)

Returns the number of objects of the specified type, in the specified Noatikl file.

The function has the following arguments:

- *window* the window handle of the file to be saved.
- *object* the object of interest.

Example:

```
local lCount = noatikl_Window_Object_GetCount(lWindow, 'Voice')
print ('Voice objects=', lCount)
```

---

noatikl_Window_Object_GetName (window, object_type, object_index)

Returns the name of the specified object.

The function has the following arguments:

- *window* the window handle of the file to be saved.
- *object_type* the type of the object of interest.
- *object_index* the index of the object of interest.

Example:

```
local lCount = noatikl_Window_Object_GetCount(lWindow, 'Voice')
print ('Voice objects=', lCount)
local lIndex
while (lIndex <= lCount)
do
  print (noatikl_Window_Object_GetName(lWindow, 'Voice', lIndex)
  lIndex = lIndex + 1
end
```

---

noatikl_Window_Object_SetName (window, object_type, object_index, newname)

Sets the name of the specified object.

The function has the following arguments:

- *window* the window handle of the file to be saved.
- *object_type* the type of the object of interest.
- *object_index* the index of the object of interest.
- *newname* the new name to use for the object; this must be unique, and must be a valid name.

Example:

```
local lCount = noatikl_Window_Object_GetCount(lWindow, 'Voice')
print ('Voice objects=', lCount)
local lIndex
while (lIndex <= lCount)
do
  -- Make-up a new name, based on the index number!
  lNewName = lIndex
  print (noatikl_Window_Object_SetName(lWindow, 'Voice', lIndex, lNewName)
  lIndex = lIndex + 1
end
```

---

noatikl_Window_Object_Parameter_Get (window, object_type, object_index, parameter_name)

Returns the value of the parameter, for the specified object index of the specified object type, in the specified Noatikl file.

The function has the following arguments:

- *window* the window handle of the file to be saved.
- *object_type* the type object of interest.
- *object_index* the index of the object of interest.
- *parameter_name* the parameter of interest.

Example:

```
local lVal = noatikl_Window_Object_Parameter_Get(lWindow, "File", 1, "Author")
print ('Author=', lVal)
```

---

noatikl_Window_Object_Parameter_Set (window, object_type, object_index, parameter_name, newvalue)

Sets the parameter to the the supplied value, for the specified object index of the specified object type, in the specified Noatikl file.

The function has the following arguments:

- *window* the window handle of the file to be saved.
- *object_type* the type object of interest.
- *object_index* the index of the object of interest.
- *parameter_name* the parameter of interest.
- *newvalue* the new value to use.

Example:

```
noatikl_Window_Object_Parameter_Set(lWindow, "File", 1, "Author", "John Smith")
```

---

noatikl_Window_GetRuleElement (window, object_type, object_name, element_index)

This function returns the rule element value of the specified index, for the named object. The value returned is a

value between 0 and 127.

The function has the following arguments:

- *window* the window handle of the file
- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *element_index* the index of the element; starting at zero for the initial element.

Example:

```
local lWindow = noatikl_Window_New()

local lCount = noatikl_Window_Object_GetCount(lWindow, 'Scale Rule')
print ('Scale Rule objects=', lCount)

local lIndex = 1
while (lIndex <= lCount)
do
  -- Dump out the scale rule element values, for the scale!
  local lName = noatikl_Window_Object_GetName(lWindow, 'Scale Rule', lIndex)
  print ('Scale', lName)

  local lItemIndex = 0
  while (lItemIndex < 12)
  do
    local lValue = noatikl_Window_GetRuleElement (lWindow, "Scale Rule", lName, lItemIndex)
    print ("lItemIndex", lItemIndex, "lValue", lValue)
    lItemIndex = lItemIndex + 1
  end

  lIndex = lIndex + 1
end
```

---

noatikl_Window_SetRuleElement (window, object_type, object_name, element_index, newvalue)

This function set the rule element value of the specified index, for the named object, to the specified value. The value must be between 0 and 127.

The function has the following arguments:

- *window* the window handle of the file
- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *element_index* the index of the element; starting at zero for the initial element.
- *newvalue* the new element value to use, in a range from 0 to 127.

Example:

```
local lWindow = noatikl_Window_New()

local lCount = noatikl_Window_Object_GetCount(lWindow, 'Scale Rule')
print ('Scale Rule objects=', lCount)

local lIndex = 1
while (lIndex <= lCount)
do
  -- Set the scale rule element values, for the scale, to stupid values!
  local lName = noatikl_Window_Object_GetName(lWindow, 'Scale Rule', lIndex)
  print ('Scale', lName)

  local lItemIndex = 0
  while (lItemIndex < 12)
  do
    local lValue = (lItemIndex * 127) / 12
    noatikl_Window_SetRuleElement (lWindow, "Scale Rule", lName, lItemIndex, lValue)
    lItemIndex = lItemIndex + 1
  end

  lIndex = lIndex + 1
end
```

---

noatikl_Window_GetEnvelopePercent (window, object_type, object_name, par_name, percent)

This function returns the percent value for the specified envelope at the given percent position. The value returned is a value between 0 and 127.

The function has the following arguments:

- *window* the window handle of the file
- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *par_name* the name of the envelope parameter (e.g. *Volume*).
- *percent* the percent value; from 0 to 100.

Example:

```
local lWindow = noatikl_Window_New()

local lCount = noatikl_Window_Object_GetCount(lWindow, 'Voice')
print ('Voice objects=', lCount)

local lIndex = 1
```

```
while (lIndex <= lCount)
do
  -- Dump out the voice volumes envelopes!
  local lName = noatikl_Window_Object_GetName(lWindow, 'Voice', lIndex)
  print ('Voice', lName)

  local lPercent = 0
  while (lPercent <= 100)
  do
    local lValue = noatikl_Window_GetEnvelopePercent (lWindow, "Voice", lName, "Volume", lPercent)
    print ("Volume percent", lPercent, "value", lValue)
    lPercent = lPercent + 1
  end

  lIndex = lIndex + 1
end
```

---

noatikl_Window_SetEnvelopePercent (window, object_type, object_name, par_name, percent, newvalue)

This function sets the envelope value at the specified percent, for the named object, to the specified value. The value must be between 0 and 127.

The function has the following arguments:

- *window* the window handle of the file
- *object_type* the object type (e.g. *Scale Rule*).
- *object_name* the name of the object (e.g. *Major*).
- *par_name* the name of the envelope parameter (e.g. *Volume*).
- *percent* the percent value; from 0 to 100.
- *newvalue* the new element value to use, in a range from 0 to 127.

Example:

```
local lWindow = noatikl_Window_New()

local lCount = noatikl_Window_Object_GetCount(lWindow, 'Voice')
print ('Voice objects=', lCount)

local lIndex = 1
while (lIndex <= lCount)
do
  -- Ramp-up every voice volume envelope!
  local lName = noatikl_Window_Object_GetName(lWindow, 'Voice', lIndex)
  print ('Voice', lName)

  local lPercent = 0
  while (lPercent <= 100)
  do
    local lValue = (lPercent * 127) / 100
    noatikl_Window_SetEnvelopePercent (lWindow, "Voice", lName, "Volume", lPercent, lValue)
    lPercent = lPercent + 1
  end

  lIndex = lIndex + 1
end
```

Objects and Parameters

The following table defines all parameters available to Noatikl scripts, listing the object name, the parameter name (as supplied to functions, and which is always unique for a given object), and the parameter name (as shown to the user in any view, which is usually shorter than the name that must be supplied to functions, and which is not always unique), and the range of legal values that you may supply.

Object: *"File"*

View	Displayed Name	Script Parameter Name	Notes
File	Title	"Title"	
	Author	"Author"	
	Midi Output Device	"Midi Output Device"	
	Midi Input Device	"Midi Input Device"	
	Midi Sync?	"Midi Sync?"	Yes or No
	Notes	"Notes"	

Object: *"Frequency Jump Rule"*

View	Displayed Name	Script Parameter Name	Notes
Frequency Jump Rule	Value	"Value"	

Object: *"Harmony Rule"*

	Displayed	Script Parameter	

View	Name	Name	Notes
Harmony Rule	Value	"Value"	

Object: *"Rhythm Rule"*

View	Displayed Name	Script Parameter Name	Notes
Rhythm Rule	Value	"Value"	

Object: *"Scale Rule"*

View	Displayed Name	Script Parameter Name	Notes
Scale Rule	Value	"Value"	

Object: *"Piece"*

View	Displayed Name	Script Parameter Name	Notes
Piece – Basics	Piece Length	"Piece Length"	Value in range 1, 32000
	Piece Length Range	"Piece Length Range"	Value in range 0, 32000
	Piece Rest	"Piece Gap"	Value in range 0, 10
	Piece Rest Range	"Piece Gap Range"	Value in range 0, 20
	Auto Restart?	"Piece Auto Restart"	Yes or No
	Meter	"Meter"	
Piece – Tempo	Tempo	"Tempo"	Value in range 1, 400
	Range	"Tempo Range"	Value in range 0, 400
	Change?	"Tempo Change"	Yes or No
	Envelope	"Tempo Envelope"	
	Envelope Range	"Tempo Envelope Range"	Value in range 0, 400
Piece – Rules	Scale Rules	"Scale Rules"	
	Harmony Rules	"Harmony Rules"	
	Next Note Rules	"Next Note Rules"	
Piece – Roots	Piece Roots	"Piece Roots"	
Piece – Scripts	Start	"Script_Start"	
	Bar	"Script_Bar"	
	MIDI In CC	"Script_MidiIn_CC"	
	MIDI In Note	"Script_MidiIn_Note"	
	Stop	"Script_Stop"	

Object: *"Voice"*

View	Displayed Name	Script Parameter Name	Notes
All voice			

views!	Mute	"Mute"	Yes or No
Voice – Basics	Patch	"Patch"	
	Use Patch?	"Use Patch?"	Yes or No
	MIDI Channel	"MIDI Channel"	Value in range 0, 16
	Voice Type	"Voice Type"	
	Pitch	"Pitch"	
	Pitch Range	"Pitch Range"	Value in range 11, 127
	Phrase Length	"Phrase Length"	Value in range 1, 256; also in Voice – Ambient
	Phrase Length Range	"Phrase Length Range"	Value in range 0, 256; also in Voice – Ambient
	Phrase Gaps	"Phrase Gaps"	Value in range 0, 256; also in Voice – Ambient
	Phrase Gaps Range	"Phrase Gaps Range"	Value in range 0, 256; also in Voice – Ambient
	Note Rest %	"Phrase Note Rest %"	Value in range 0, 100; also in Voice – Ambient
Voice – Ambient	Units	"Ambient Units"	
	Duration	"Ambient Duration"	Value in range 0, 32000
	Duration Range	"Ambient Duration Range"	Value in range 0, 32000
	Gap Minimum	"Ambient Gap Min"	Value in range 0, 32000
	Gap Range	"Ambient Gap Range"	Value in range 0, 32000
Voice – Following	Follow Voice	"Follow Named Voice"	
	Percent	"Follow Percent"	Value in range 0, 100
	Strategy	"Follow Strategy"	
	Units	"Follow Delay Unit"	
	Delay	"Follow Delay"	Value in range 0, 32000
	Delay Range	"Follow Delay Range"	Value in range 0, 32000
	Shift/Interval	"Follow Shift/Interval"	Value in range -60, +60

	S/I Range	"Follow Shift/Interval Range"	Value in range -60, +60
Voice - Repeat	Voice	"Repeat Specific Voice"	
	Percent	"Repeat Bars Percent"	Value in range 0, 100
	Bars	"Repeat For Bars"	Value in range 1, 100
	History Range	"Repeat Bar History Range"	Value in range 0, 100
	History	"Repeat Bar History"	Value in range 1, 100
	Bars Range	"Repeat For Bars Range"	Value in range 0, 100
Voice - Patterns	Patterns	"Patterns"	
	Use Percent	"Patterns Use Percent"	Value in range 0, 100
	Mutation Factor	"Mutation factor"	
	Bars Between	"Mutate No. Bars"	Value in range 0, 100
	Bars Range	"Mutate No. Bars Range"	Value in range 0, 100
	Mutate Rhythm?	"Mutation of Rhythm"	Yes or No
	Meter	"Meter"	
Voice - Chords	Depth	"Chord Depth"	Value in range 1, 32
	Pitch Offset	"Chord Pitch Offset"	Value in range -60, +60
	Delay	"Chord Delay"	Value in range 0, 32000
	Depth %	"Chord Depth Percent"	Value in range 0, 100
	Delay Unit	"Chord Delay Unit"	
	Velocity Factor	"Chord Velocity Factor"	Value in range -100, +100
	Delay Range	"Chord Delay Range"	Value in range 0, 32000
	Depth Range	"Chord Depth Range"	Value in range 0, 32
	Strategy	"Chord Strategy"	
	Shift/Interval	"Chord Shift/Interval"	Value in range -60, +60
	S/I Range	"Chord Shift/Interval Range"	Value in range -60, +60
Voice - Rules	Harmony Rules	"Harmony Rules"	
	Next Note Rules	"Next Note Rules"	
	Rhythm Rules	"Rhythm Rules"	

	Scale Rules	"Scale Rules"	
	Harmonize?	"Harmonize?"	Yes or No
	Voice Root	"Voice Root"	
Voice - Scripts	Start	"Script_Start"	
	Bar	"Script_Bar"	
	Composed	"Script_Composed"	
	MIDI In CC	"Script_MidiIn_CC"	
	MIDI In Note	"Script_MidiIn_Note"	
	Stop	"Script_Stop"	
Voice - Notes	Copyright	"Copyright"	
	Notes	"Notes"	
Voice - Controllers	Damper/Hold (64)	"Damper/Hold (64)"	Value in range -1, 127
	Harmonic Content (71)	"Harmonic Content (71)"	Value in range -1, 127
	Reverb (91)	"Reverb (91)"	Value in range -1, 127
	Chorus (93)	"Chorus (93)"	Value in range -1, 127
	Damper Release	"Damper Release"	Yes or No
	Portamento (84)	"Portamento (65)"	Value in range -1, 127
	MIDI Channel Sharing	"MIDI Channel Reallocation"	Yes or No
Voice - Micro Controller 1	MIDI CC	"User Controller 1 Midi Command"	
	Mode	"User Controller 1 Mode"	
	Minimum	"User Controller 1 Minimum"	Value in range 0, 127
	Range	"User Controller 1 Range"	Value in range 0, 127
	Change	"User Controller 1 Change"	Value in range 0, 127
	Change Range	"User Controller 1 Change Range"	Value in range 0, 127
	Update	"User Controller 1 Update"	Value in range 0, 10000
	Update Range	"User Controller 1 Update Range"	Value in range 0, 10000
	Update Units	"User Controller 1 Update Unit"	
	Beat Cycle Length	"User Controller 1 Beat Cycle Length"	Value in range 0, 32000
	Phase Shift%	"User Controller 1 Phase Shift"	Value in range 0, 100

Voice – Micro Controller 2	MIDI CC	"User Controller 2 Midi Command"	
	Mode	"User Controller 2 Mode"	
	Minimum	"User Controller 2 Minimum"	Value in range 0, 127
	Range	"User Controller 2 Range"	Value in range 0, 127
	Change	"User Controller 2 Change"	Value in range 0, 127
	Change Range	"User Controller 2 Change Range"	Value in range 0, 127
	Update	"User Controller 2 Update"	Value in range 0, 10000
	Update Range	"User Controller 2 Update Range"	Value in range 0, 10000
	Beat Cycle Length	"User Controller 2 Beat Cycle Length"	Value in range 0, 32000
	Update Units	"User Controller 2 Update Unit"	
	Phase Shift%	"User Controller 2 Phase Shift"	Value in range 0, 100
Voice – Micro Note Delay	Delay Range	"Micro Note Delay Range"	Value in range 0, 1000
	Delay Change	"Micro Note Delay Change"	Value in range 0, 1000
	Delay Offset	"Micro Note Delay Offset"	Value in range -1000, +1000
Voice – Micro Pitch	Bend Sensitivity	"Pitch Bend Sensitivity"	Value in range 0, 24
	Pitch Bend Offset	"Pitch Bend Offset"	Value in range -8192, +8191
	Pitch Range	"Micro Pitch Range"	Value in range 0, 8191
	Pitch Change	"Micro Pitch Change"	Value in range 0, 1000
	Pitch Update	"Micro Pitch Update"	Value in range 0, 10000
	Update Range	"Micro Pitch Update Range"	Value in range 0, 10000
Voice – Micro Pitch	Range	"Micro Volume Range"	Value in range 0, 127
	Change	"Micro Volume Change"	Value in range 0, 127
	Update	"Micro Volume Update"	Value in range 0, 1000
	Update Range	"Micro Volume Update Range"	Value in range 0, 10000
Voice –			

Note to MIDI CC Mapping	CC for Note On?	"MIDI CC instead of Note?"	Yes or No
	Note On CC	"MIDI CC Note On Value"	
	CC for Velocity?	"MIDI CC for Note On Velocity?"	Yes or No
	Velocity CC	"MIDI CC Note On Velocity"	
	CC for Off?	"MIDI CC for Note Off? "	Yes or No
	Note Off CC	"MIDI CC Note Off Value"	
Voice – User Envelope 1 (Volume)	MIDI CC	"User Envelope 1 MIDI CC"	
	Enabled?	"User Envelope 1 Enabled"	Yes or No
	Envelope	"Volume"	
Voice – User Envelope 2 (Pan)	MIDI CC	"User Envelope 2 MIDI CC"	
	Enabled?	"User Envelope 2 Enabled"	Yes or No
	Envelope	"Pan (10)"	
Voice – Envelope – Velocity	Velocity	"Velocity"	
Voice – Envelope – Velocity Range	Velocity Range	"Velocity Range"	
Voice – Envelope – Velocity Change	Velocity Change	"Velocity Change"	
Voice – Envelope – Velocity Change Range	Velocity Change Range	"Velocity Change Range"	

See Also

- Scripting Overview
- Trigger Scripts
- Noatikl as a Hyperinstrument