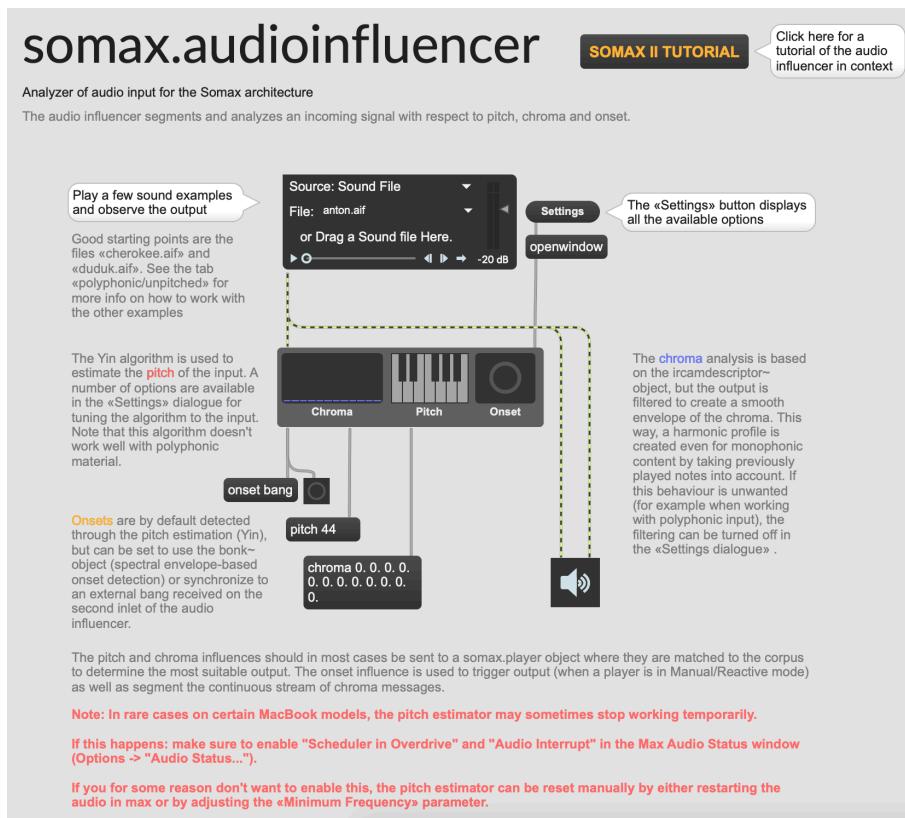


# **Appendix A**

# **Max Documentation**

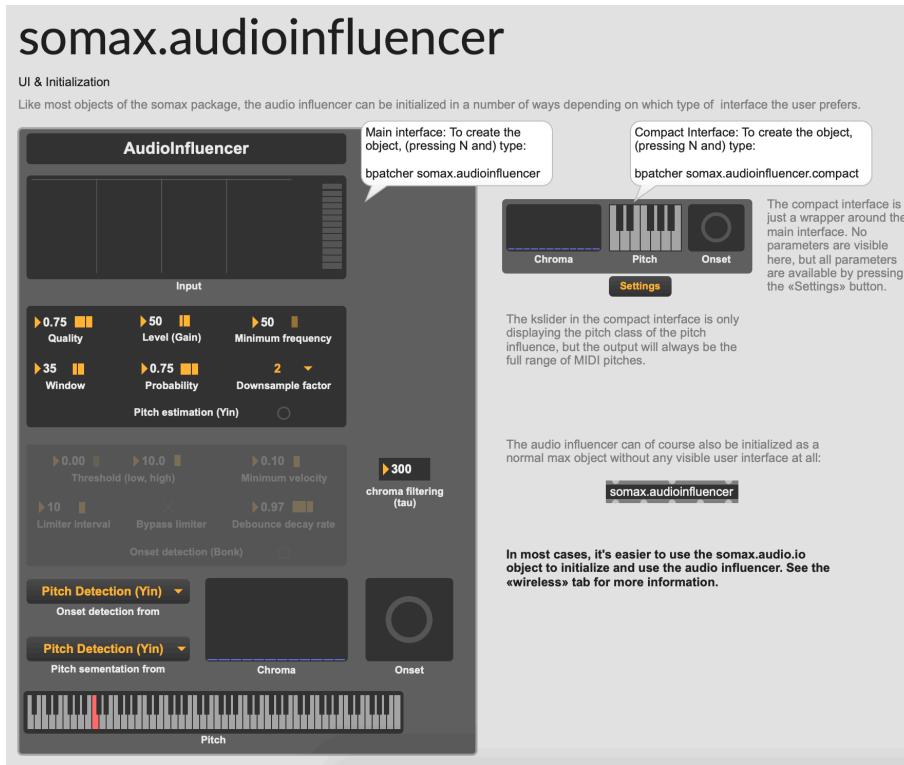
This appendix outlines the user documentation which is available inside the Somax package when running Max as a set of screenshots. This is only intended to give an overview of the existing documentation - the proper way to use the documentation is of course to open it in Max and listen to and/or interact with the examples.

## The Somax Software Architecture



The first tab of the somax.audioinfluencer object.

## The Somax Software Architecture



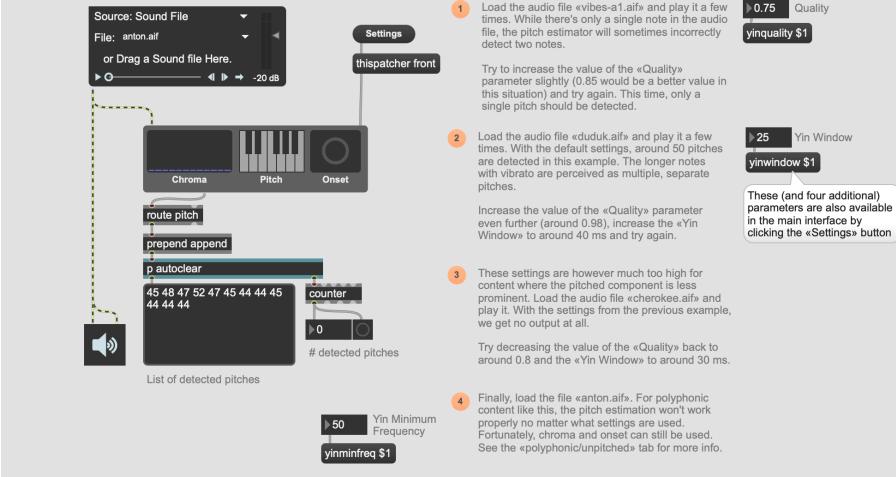
The second tab of the somax.audioinfluencer object.

## The Somax Software Architecture

### somax.audioinfluencer

#### Handling monophonic input

Depending on the character of the monophonic input, tuning of the internal parameters of the pitch estimator may be necessary.

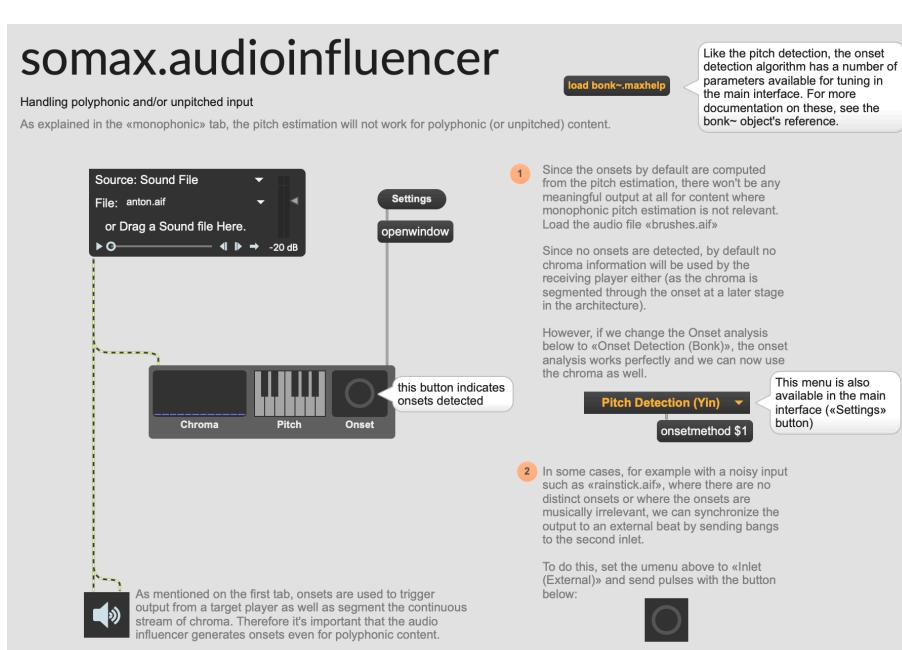


The third tab of the somax.audioinfluencer object.

### somax.audioinfluencer

#### Handling polyphonic and/or unpitched input

As explained in the «monophonic» tab, the pitch estimation will not work for polyphonic (or unpitched) content.

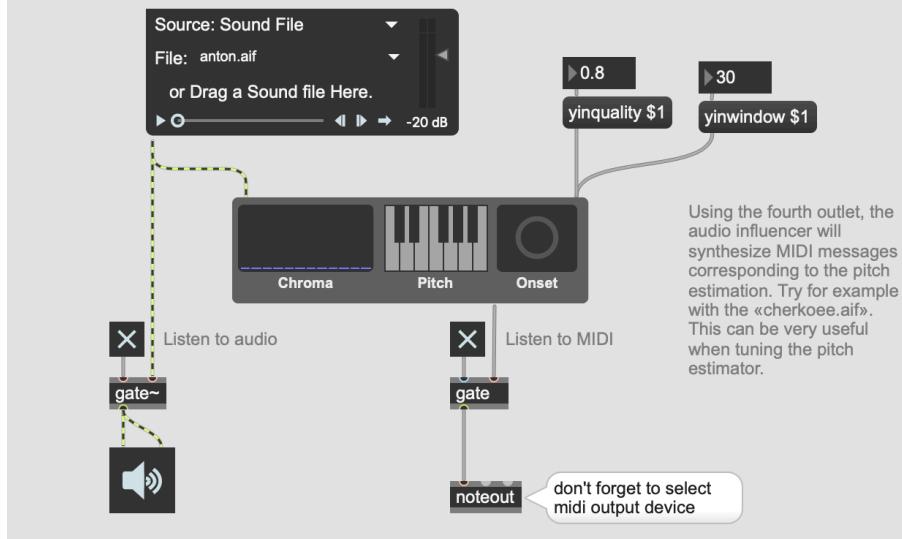


The fourth tab of the somax.audioinfluencer object.

## somax.audioinfluencer

MIDI synthesis and output

The fourth outlet is generating MIDI messages synthesized from the pitch estimation.



The fifth tab of the somax.audioinfluencer object.

# somax.audioinfluencer

Sending influences wirelessly

Somax can send influences between influencers and players wirelessly (i.e. without max patch cords).

To create a wireless audio influencer, press N and type:  
bpatcher somax.audio.io

the somax.audio.io object is just a convenient wrapper around the audio influencer with some additional user interface to handle input and output

The wireless influencer can be given a name to differentiate it from other influencers using the bpatcher @args attribute, i.e.:  
bpatcher somax.audio.io @args <name>

If no argument is provided, the influencer will automatically be assigned a unique name

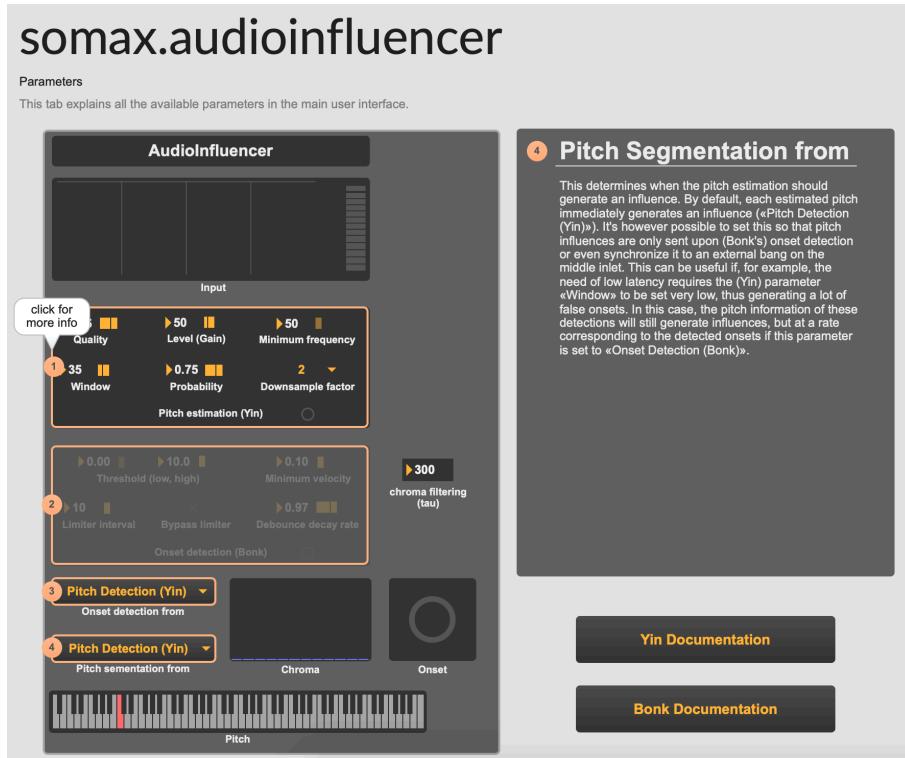
The name attribute can also be set using the «Argument(s)» field in the Inspector

The name of the influencer (and all of its influences) will be available in the routing section of the somax.player.io module.

<None> See [somax.player](#) for more information on wireless routing

The sixth tab of the somax.audioinfluencer object.

## The Somax Software Architecture



The seventh tab of the `somax.audioinfluencer` object.

## The Somax Software Architecture

**somax.midiinfluencer**

Analyzer of MIDI input for the Somax architecture

The MIDI influencer segments and analyzes an incoming stream of midi messages with respect to pitch, chroma and onset.

press some keys on the polyphonic keyboard to see the behaviour of the influencer

The **pitch** is by default selected as the top note in the received input, but there are a number of other options available in the «Settings» dialog.

By default, an «**onset influence**» is sent each time a note on is received. This isn't always ideal for polyphonic input, for which a workaround is described in the «polyphony» tab.

The «**Settings**» button displays all the available options

Click here for a tutorial of the MIDI influencer in context

The «**Settings**» button displays all the available options

For the **chroma**, a set of harmonics are created for each MIDI note before computing a pseudo-spectrogram for the MIDI input. The purpose of this is to create a smoother interaction between influence data from MIDI and audio respectively

The changes in chroma are also not instantaneous but decays slowly once a note has been released. This behaviour can be turned off in the «Settings».

The pitch and chroma influences should in most cases be sent to a somax.player object where they are matched to the corpus to determine the most suitable output. The onset influence is used to trigger output (when a player is in Manual/Reactive mode) as well as segment the continuous stream of chroma messages.

The first tab of the `somax.midiinfluencer` object.

**somax.midiinfluencer**

UI & Initialization

Like most objects of the somax package, the MIDI influencer can be initialized in a number of ways depending on which type of interface the user prefers.

Main interface: To create the object, (pressing N and) type:  
`bpatcher somax.midiinfluencer`

Compact interface: To create the object, (pressing N and) type:  
`bpatcher somax.midiinfluencer.compact`

The compact interface is just a wrapper around the main interface. No parameters are visible here, but all parameters are available by pressing the «Settings» button.

The MIDI influencer can of course also be initialized as a normal max object without any visible user interface at all:

`somax.midiinfluencer`

In most cases, it's easier to use the `somax.midi.io` object to initialize and use the MIDI influencer. See the «wireless» tab for more information.

The second tab of the `somax.midiinfluencer` object.

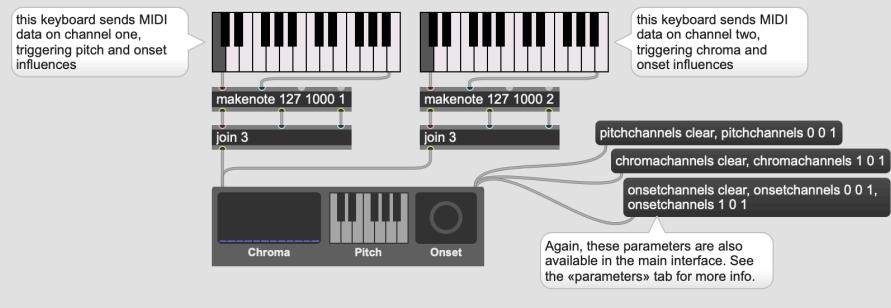
## The Somax Software Architecture

---

### somax.midiinfluencer

#### Multiple Sources

It's possible to use multiple MIDI sources and specify which ones should send harmonic, melodic and/or onset influences accordingly. This is also useful when combining audio influences and MIDI influences. For example, if a violinist and a pianist are interacting with the system and they want the chroma influences to be analyzed from MIDI input from the pianist, while pitch influences should be sent from the violinist's input, but both should be able to send onsets.

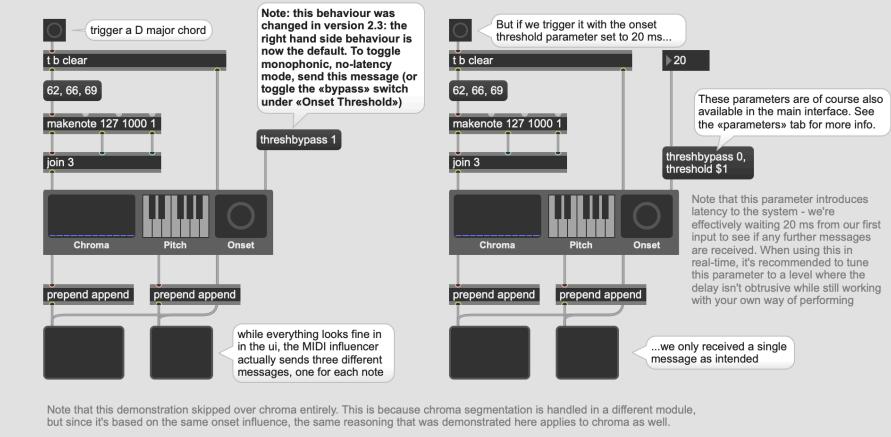


The third tab of the somax.midiinfluencer object.

### somax.midiinfluencer

#### Polyphony

When working with polyphonic input, the idea that each Note On message triggers an onset is not always ideal. This means that when you play a chord, the MIDI Influencer will send multiple sets of influences instead of just one.

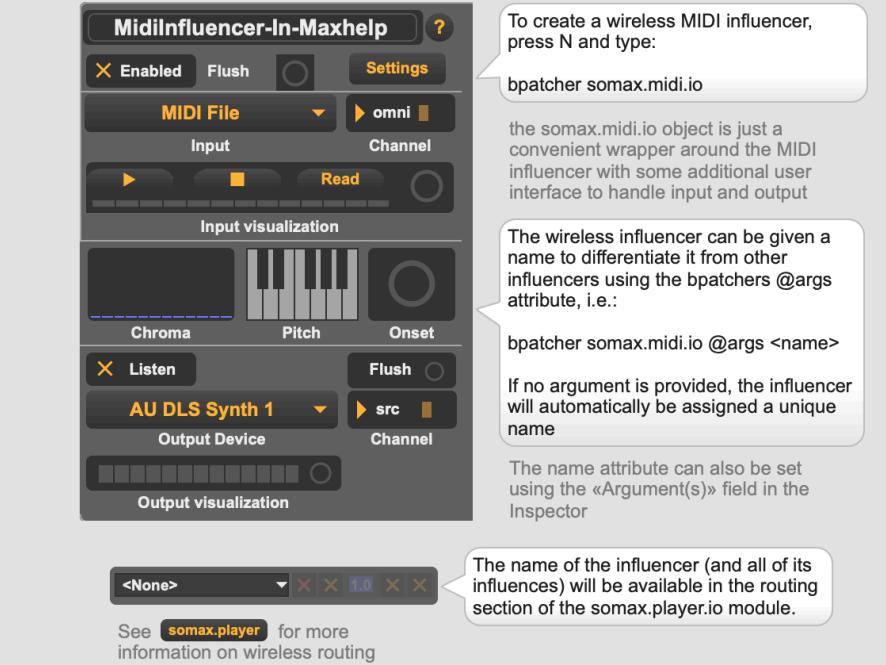


The fourth tab of the somax.midiinfluencer object.

# somax.midiinfluencer

Sending influences wirelessly

Somax can send influences between influencers and players wirelessly (i.e. without max patch cords).



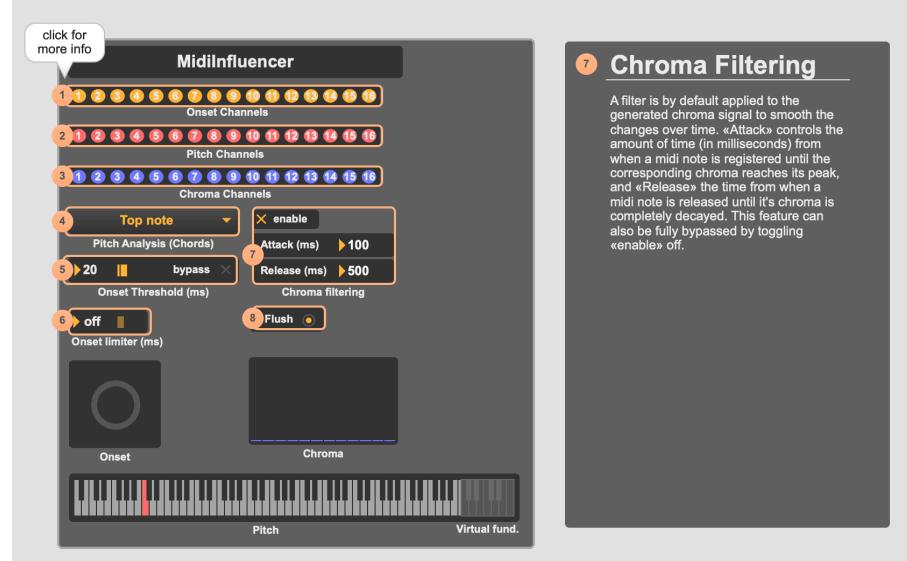
The fifth tab of the somax.midiinfluencer object.

## The Somax Software Architecture

### somax.midiinfluencer

#### Parameters

This tab explains all the available parameters in the main user interface.



The sixth tab of the somax.midiinfluencer object.

### somax.server

#### The server of the Somax architecture

The somax.server object handles the communication with the server (written in Python) where all the players are stored. The server also handles the scheduling of events and all aspects of timing and tempo, as well as the construction of new corpora through the Corpus Builder module.

The server is a standalone Python application that (in most cases) will run in a terminal window on the same machine as the max code. The first time the server is launched you may need to give it permissions to run, as described in the README and the tutorial below.

[SOMAX II TUTORIAL](#)

Click here for a tutorial  
of the server in context

The first tab of the somax.server object.

## The Somax Software Architecture

---

# somax.server

**The Corpus Builder**  
Module for constructing corpora from MIDI files.



The main generative agent of somax, the somax.player, generates output by using pre-existing MIDI files as a source. However, the player cannot operate directly on the MIDI file, a «Corpus» has to be constructed from the MIDI file.

To build a corpus:

1. Make sure that the server is started.
2. Drop a MIDI file in the box above (or use «Read» to navigate)
3. Select which MIDI channels of the corpus that should be used when comparing the corpus to incoming pitch and chroma influences respectively (you can read more about this in the pdf «A Gentle Introduction to Somax»)
4. Give the corpus a name
5. Build it

The corpus will automatically be listed and available in player's interface.

**Note:** The Corpus Builder in this particular tutorial is not connected to a server, so it's not possible to use this to construct a corpus. Use the regular Corpus Builder in the somax.server.io module to construct a corpus.

The second tab of the somax.server object.

# somax.server

Sending influences wirelessly  
Somax can send influences between influencers and players wirelessly (i.e. without max patch cords).

When using the wireless architecture, one server object must always exist somewhere in the patcher. This module handles necessary communication with the server and adds means to control the server's transport (time and tempo).

**Do not attempt to create multiple servers - this will cause communication problems between Max and the server. For this reason, below is just a picture of the server interface, clicking the buttons is not possible.**



The status window will display whether the server is ready, running, offline, etc.

The first step is to start the server. When the server is running, this button will not be visible.

The beat tracker is only visible if «Tempo Source» is set to an influencer. In that case, it will listen to the onsets of that particular influencer to set the tempo.

A number of convenient modules exists for recording MIDI, flushing all MIDI devices and building corpora. See the «corpus builder» tab for more info regarding the latter.

If the tempo source is set to a somax.player, the tempo will be set from the corpus of the player. If the tempo source is set to a somax.audioinfluencer or somax.midiinfluencer, the tempo will be set through the beat tracker.

The third tab of the somax.server object.

## The Somax Software Architecture



The first tab of the somax.player object.

## The Somax Software Architecture

---

**somax.player**

**UI & Initialization**

Like most objects of the somax package, the player can be initialized in a number of ways depending on what interface the user prefers.

Main interface: To create the object, (pressing N and) type:  
`bpatcher somax.player`

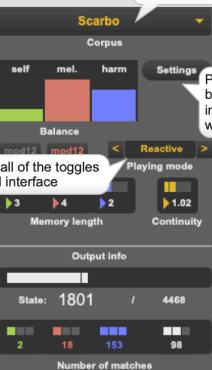


To assign the player a name MyName:  
`bpatcher somax.player @args MyName`

The «Playing mode» controls all of the toggles on the left-hand side in the full interface

All of the other parameters have a one-to-one correspondance between the full and compact interfaces

Compact Interface: To create the object, (pressing N and) type:  
`bpatcher somax.player.compact`



Press the «Settings» button to access the full interface in a new window

The compact interface is just a wrapper around the main interface. Only a subset of the main interface's parameters are displayed here

Updating parameters in the compact interface will send the changes to the main interface as well (and vice versa).

Initializing somax.player without graphical user interface:

The player can of course also be initialized as a normal max object without any visible user interface at all by (pressing N and) typing:  
`somax.player`

In most cases, it's easier to use the `somax.player.io` object to initialize and use the player. See the «wireless» tab for more information.

The second tab of the `somax.player` object.

## The Somax Software Architecture

---



The third tab of the somax.player object.

## The Somax Software Architecture

---

**somax.player**

**Parameters**

While the player is fully usable with the default settings, the quality of the output is largely determined by how well the parameters are tuned to the input and the corpus. Here, each parameter is described in detail. Note that apart from setting the corpus, all of these parameters can (unless otherwise specified in their descriptions) be controlled in real-time as a means of interacting with the player.



click any of the numbered buttons for more info

**Player**

- 1 – click to refresh –
- 2 Corpus
- 3 Playing Mode: Manual
- 4 Ignore Phase: True
- 5 Note Durations: Hold
- 6 Onsets: Synchronized
- 7 Decay Basis: Event
- 8 Continuity Factor: ▶ 1.50
- 9 Quality Threshold: ▶ off
- 10 -5 -4 -3 -2 0 1 2 3 4 5 6 Active transpositions (num semitones)
- 11 bypass self melodic harmonic
- 12 Balance mod12 mod12
- 13 mod12 mod12
- 14 mod12 mod12
- 15 Memory length (n-gram order) 2.20 2.20 2.20
- 16 Decay time (unit: # events) 2.20 2.20 2.20
- 17 Reinstantiate
- 18 OSC Port: ▶ 1236
- 19 Reset influence

Should absolutely not be used in real-time!

**15 Layer Decay Time**

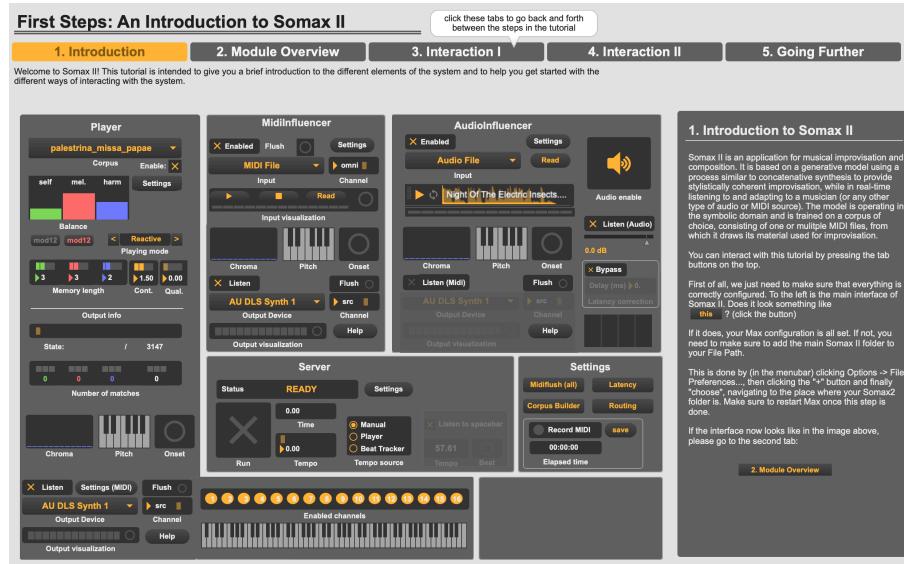
As briefly described under 'Decay Basis', matches from a previous time step has a degree of impact on the generation process for consecutive time steps. This parameter controls for how many events/how many beats (again, see 'Decay Basis' for more information) previous matches impact the generation process.

Note: Several parameters consist of both a slider and a number box that can be manipulated. As a general rule, sliders have a lower and upper bound specifying a reasonable range for the parameter. It is however in many cases possible to use the numbox to set the parameter to a value outside of this range.

The fourth tab of the `somax.player` object.

## Appendix B

# Max Tutorial



The first section of the tutorial.

## The Somax Software Architecture

**First Steps: An Introduction to Somax II**

click these tabs to go back and forth between the steps in the tutorial

1. Introduction    2. Module Overview    3. Interaction I    4. Interaction II    5. Going Further

Somax II consists of four main modules: The Player, the Midi Influencer, the Audio Influencer and the Server. This slide is intended to give a brief overview of the different objects and their corresponding roles.

For an overview of the interaction between the objects, see [this diagram](#).

The second section of the tutorial.

**First Steps: An Introduction to Somax II**

click these tabs to go back and forth between the steps in the tutorial

1. Introduction    2. Module Overview    3. Interaction I    4. Interaction II    5. Going Further

This slide will present a brief introduction to how to interact with somax - the steps required to get some output from the Player and how to use the influencers to interact with the generated output.

It's divided into two parts: «Interaction I» describing how to start the server and produce output and «Interaction II» describing how to interact with the output

If you've already started the server and have a terminal window running, this button will not be present and you can disregard this step.

**Step 1 : Starting the server**  
Press this button to start the server. A terminal window will open (and should not be closed until the end of the session), displaying some text indicating that the server has been successfully started. The status of the server is displayed in the Max Console. Note: The first time you start the server, mac OS may ask you for permissions to run the application. This is normal, simply click 'allow' when prompted.

**Step 2 : Instantiating the player**  
The second step is to instantiate all existing Players (which in our case is just the single one). This is done by once again clicking this button, which now should say

**Step 3 : Load a corpus**  
Before we can start the scheduler, we need to load a corpus. To do this, click the 'File' menu and selecting one of the pre-built corpora available. This step may take some time (5-20 seconds depending on the corpus length).

**Step 4 : Interact (microphone)**  
Finally, enable audio input by making sure that **①** and **②** are enabled and **③** is set to 'Audio Input'. Then click the 'Record MIDI' button to start the server by toggling **④**. Now, hum a few notes (or say a few words) and you should hear Somax providing accompaniment to your voice based on the corpus you selected!

[Continue to the next step >](#)

The third section of the tutorial.

## The Somax Software Architecture

**First Steps: An Introduction to Somax II**

click these tabs to go back and forth between the steps in the tutorial

1. Introduction    2. Module Overview    3. Interaction I    4. Interaction II    5. Going Further

This is the second part of the introduction to interaction in somax. Make sure that you have completed all the steps in [Interaction I](#) so that the system is in a state ready for interaction.

**Step 4 : Interact (MIDI)**

Next, we'll interact with the system using MIDI.

First of all, make sure that you've completed the steps in the previous tab (if not, go back to [Interaction I](#)). You can do this by clicking the [Interaction I](#) tab. Make sure that [Enabled](#) is enabled, [set to emit \(External\)](#) and [is enabled](#) is enabled. Now, play a few notes on the keyboard below. You should be able to hear them, as well as the matching output from the player.

Ideally, you should also notice that the player starts to output notes in a more matches and therefore a higher probability of jumps. For more information on these parameters, once again see [help](#).

A final step is to set the playing mode at [Continuous](#). In this mode, the player will regardless of input continuously generate output and jump in a similar manner as above when input is received.

[continue to the next step >](#)

The first section of the tutorial.

**First Steps: An Introduction to Somax II**

click these tabs to go back and forth between the steps in the tutorial

1. Introduction    2. Module Overview    3. Interaction I    4. Interaction II    5. Going Further

**5. Going Further**

This tutorial intended to give you a quick introduction to Somax and to serve as a starting point for both audio and MIDI-based interaction.

There are however a number of other aspects of the system that you can for example connecting a MIDI keyboard, audio file or a sequencer (using an IAC bus) as an input, using multiple players who listen to each other, collecting data from sensors and interacting with the tempo and beat tracker, and many other possibilities.

In the end, the quality of the output will always depend largely on how well the influencers and the player(s) are turned with respect to the corpus and the input, so it's highly encouraged to experiment with the different parameters. The detailed descriptions of the options in each module will provide a good starting point for going further, outlining a few use cases as well as providing detailed descriptions about each individual parameter.

**A note on the user interface:**

When you open the real `somax2.patcher`, you will notice that it doesn't look exactly the same as in this tutorial. In this patcher, the four main modules have been moved around, and each player has a routing module attached to it.

In the real patcher, it's possible to duplicate any patcher by unlocking the patcher and, selecting the object and pressing **cmd** (alternatively **ctrl**)-drag the object). This way, it's possible to create multiple instances of a patcher and connect them to each other. To make a player p1 listen to a player (or influencer) p2, simply select `p2` as the source in `p1`'s routing module. It's also possible to write your own routings for the players to listen to. Consult the `somax.player` help file for more information.

The final section of the tutorial.