Voks manual

1 Introduction

Voks is a music instrument which allows you to play a voice using non-vocal gestures.

Contrary to digital instruments such as Cantor Digitalis, it does not generate sound from scratch; instead, the performer uses their gestures to control the rhythm, melody, and vocal quality of a prerecorded voice sample.

2 Quick Start

This section explains how to test Voks using on a Mac, using a tablet and the default voice data. You will probably want to provide your own voice data and/or use other controllers; please refer to the subsequent sections for more details.

2.1 Downloads

To play Voks, you need the app itself as well as some data files (mainly, labeled voice samples). A directory `data` containing some example data for use with Voks is available.

Both the Voks app and the `data` directory can be downloaded from http://chorus-digitalis.lam.jussieu.fr/voks.html

2.2 Launching the app

If your Mac is configured to allow apps from the App Store and identified developers, the first time you launch this app, it will ask you to confirm that you really want to open it. If the problem persists, you can follow these steps: https://support.apple.com/fr-fr/HT202491

To use a Wacom graphics tablet to control Voks, you need to first install this driver: https://www.wacom.com/fr-fr/support/product-support/drivers

2.3 Loading default data

Click the *Load utterance file…* button on the left panel. A file dialog opens; navigate to the `data` directory and select the `utterances.txt` file.

2.4 Selecting the right controller

In the *Controller selection* panel, click on the *Rhythm* tab and select the spacebar. Click on the *Pitch* tab, select *Tablet* as the *Pitch controller* and find your tablet in the *Selected device* list.

2.5 Playing

- Make sure that the *Toggle audio status* icon is on (black).
- Place the tip of the stylus in contact with the surface of the tablet.
- Repeatedly press the spacebar.

3 Using Custom Samples

The default version of Voks comes with a number of voice recordings, but you will probably want to use your own ones. To use custom voice recordings, you need to do the following:

1. Provide a voice recording in the correct format,
2. Provide a syllabic labeling of that recording,
3. Edit the utterance file to indicate the location of those files.

3.1 The Voice Recording

To synthesize any sound, Voks needs to be provided with an audio sample of a voice recording. That recording needs to meet a number of conditions:

- **General features**: The sample should be a recording of a monophonic voice sample. The result is more likely to be satisfactory if your sample is:
  - As dry (*i.e* without reverb) as possible,
  - Uttered relatively fast (this makes the time control algorithm more reactive to your gestures), but with each sound articulated clearly,
  - Contained in a relatively short pitch range, so as to avoid pitch-range related timbre variations (if manageable, singing on a truly constant pitch can be a good idea; see next point).
- **Pitch**: Unless you do not intend to control the pitch of your sample, the pitch of your sample should be a constant, known value. You can achieve this either by “singing” on a constant note, or by using pitch-correction software to force the pitch of your recording *a posteriori*.
- **Format**: The recording **must** be a *mono* file, with a sample rate of 44100 Hz, in the .wav format. Failing to comply with this may result in a number of issues, ranging from Voks refusing to play any sound to weird timing errors.

Those are the rules that ensure Voks works as close as possible to the way it was intended to work. Naturally, you are encouraged to break them to see what the results are, *e.g* by using non-monophonic, heavily-preprocessed, or even non-vocal samples.

3.2 The Labeling File

With any voice sample, you need to provide a file, also known as a *labeling*, that contains information about the temporal repartition of syllables in that sample. This section details about the exact nature of that information, and the way to encode that information (see *Labeling file format specification*).

A labeling is composed of timestamps, which can be seen as anchors which Voks' virtual playhead will target. You will need to provide:

- A pair of timestamps corresponding respectively to the start and the end of your audio sample,
- as well as, for each syllable in your sample, a pair of timestamps located respectively in a consonant and a consecutive vowel.

Such timestamps can be obtained using a visual sound editor such as Audacity or Praat.

The **starting and ending timestamps** should be located just before and after the sound in your sample; that is, in silent zones, but relatively close (a few dozen ms) to the sound. If the audio file is cropped tightly around the sound, you can choose 0 ms and the total duration of your sample as starting and ending timestamps, but explicitly setting those bounds will allow you to also deal with files that contain long silent zones before/after the sound, as well as files with sound that you do not wish to resynthesize.

The **syllabic control points** should be located in spectrally stable zones of your signal, that is, zones where the timbre of the sample does not vary much. For (monophtong) vowels, this corresponds to the whole vowel; for plosives, to the quiet zone before the explosion.

There are a few special cases to consider here:

- When a phone (typically, a consonant such as /l/, or a semi-vowel) does not feature a spectrally stable zone, just put the timestamp where the spectral features of the phone are most salient (often in the middle of the phone).
- When there are several consecutive consonants (a *consonant cluster*), choose one, and ask yourself what consonant you would stop at, were you asked to stop between two vowels.

Example: If you were asked to stop in between the two vowel of the word "express", you might say something like "ek—", where you'd stop just before the /k/ explosion, and then later say "—kspress". So you should place the control point just before the explosion of the /k/.

- When the stable zone of a phone (typically, a vowel) is longer than a few dozen ms, put the point at a late point in that zone, leaving just a few dozen ms of stable sound after the point.

Again, those are the rules that we had in mind while developing Voks. You are encouraged to experiment with labelings, *e.g* by labeling based on units different from the syllable.

Labeling File Format Specification

A labeling file is a text file composed of a series of lines, each composed of:

- An identifier, which is either `start`, `end`, or a whole number,
- A comma,
- One or more floating-point timestamps, expressed in second, and separated by a space if needed,
- A semicolon,
- And a newline character

Lines that start with the `start` or `end` identifier comprise a single timestamp (the starting/ending timestamp), lines that start with a whole identifier comprise two (a consonant timestamp followed by a vowel timestamp).

Whole identifiers should be numbered from 1 to the total number of pairs, in strictly increasing order; the timestamps associated with whole identifiers should themselves appear in strictly increasing order, and be strictly contained in the interval defined by the `start` and `end` timestamps.

Any timestamp except the `start` and `end` ones may be replaced by the `null` or `next` keyword. When encountering `null`, Voks will simply do nothing special, and when encountering `next`, it will look up the next control point with the same type (consonantic/vocalic) instead. This is useful when dealing with samples that start with a vowel (use `null` in lieu of the nonexistent starting consonant), end with a consonant (use `null` in lieu of the nonexistent ending vowel), or with consecutive vowels that belong to separate syllables (use `null` in lieu of the nonexistent separating consonant).

Here is an example of a valid labeling file:

```
start, 0.000;
end, 8.000;
1, null 0.100;
2, 0.300 0.500;
3, 0.650 0.700;
4, 0.701 0.750;
```

Note that the starting timestamp is 0. but doesn't need to be. Also notice that the points 0.700 and 0.701 are separated by a millisecond so as to ensure that timestamps are in strictly increasing order.

3.3 The Utterance File

The utterance file is a convenient way to quickly access a bunch of sample-labeling pairs of files at once. Its syntax resembles that of labeling files (both are instances of the Max language's `coll` syntax).

An utterance file is a text file whose each line comprises:
- A whole number identifier,
- A comma,
- Three strings, each delimited by a pair of quotes `"`, and separated by spaces:

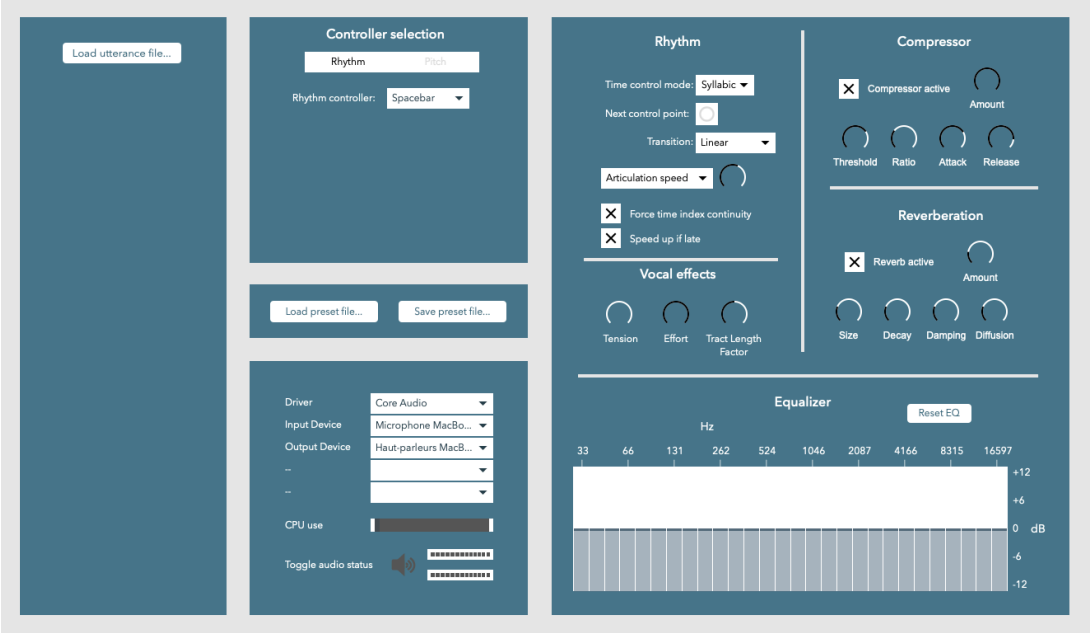
- An identifier which can be any string you want; it will appear in Voks' graphical interface,
- The path to your sample audio file (see above),
- The path to your labeling text file (see above);

- A newline character.

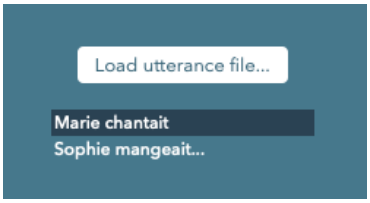
Here is an example of a valid utterance file:

```
1, "Marie" "/path/to/samps/marie.wav" "/path/to/labs/marie.txt";
2, "Sophie" "/path/to/samps/sophie.wav" "/path/to/labs/sophie.txt";
```

4 Interface



Utterance file

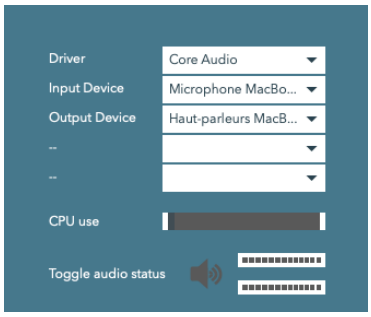


- `Load utterance file` opens a dialog box that allows you to specify the path to the utterance file. This file contains a list of audio and analysis files that can be used in Voks.

Presets

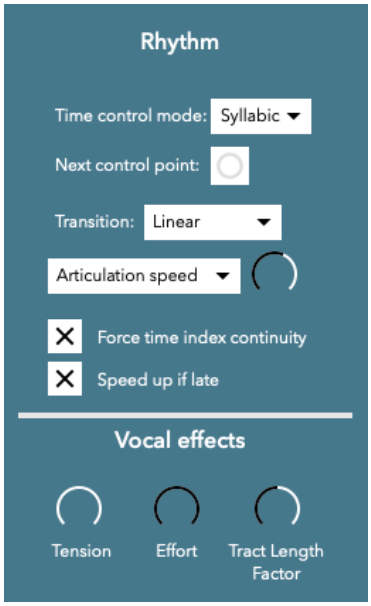
- `Load preset file` opens a dialog box to specify the path to a preset file.
- `Save preset file` saves the actual state in a preset file.

Driver



- `Driver` specifies wich audio drivers is used.
- `Input Device` specifies the audio input device.
- `Output Device` specifies the output device.
- `CPU use` shows the global CPU use.
- `Toggle audio status` sets the audio input/output on/off.

Rhythm



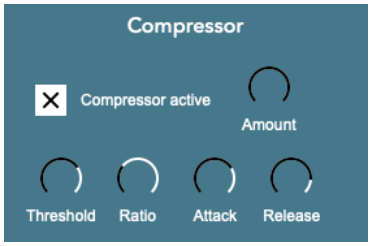
- `Time control mode` specifies how to control rhythm:
  - `Scrub` allows you to control directly the playhead position.
  - `Speed` allows you to control directly the speed.
  - `Syllabic` allows you to control the syllabic advancement in real-time:
    - `Next control point` triggers advancement to the next control point with the specified type.
    - `Transition` specifies the transition shape between consecutive control point:
      - `Linear`: the playback position follows a linear evolution between two consecutive control point.
      - `Parabolic`: the playback position follows a parabolic evolution between two consecutive control point.
      - `Direct control`:

- `Articulation speed` specifies the playback speed between two consecutive control point.
- `Force time continuity` forces the time index continuity in syllabic mode.
- `Speed up if late` accelerates the playback speed.

Vocal effects

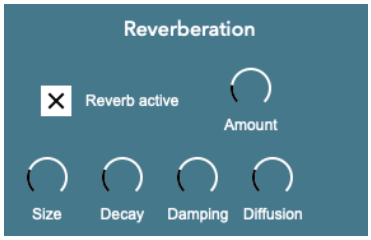
- `Tension`
- `Effort`
- `Tract Length Factor` sets the vocal tract length stretching/shrinking factor

Compressor



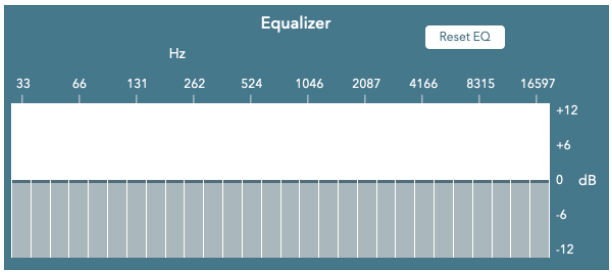
- `Compressor active` activates the compressor.
- `Amount` sets the dry/wet level of the compressor (range 0-1).
- `Threshold` sets the threshold level where the compressor applies level reduction (range Odb -36db).
- `Ratio` set the numerator of the compressor gain reduction ration (range 0-100).
- `Attack` sets the rate at which the compressor is engaged when the signal exceeds the threshold (range 0-150 ms).
- `Release`sets the compressor release rate.

Reverberation



- `Reverb active` 0-reverb off ; 1-reverb on
- `Amount` specifies the dry (0) /wet (1) level of the reverberation.
- `Size` specifies the reverberation size (range 0-127).
- `Decay` specifies the reverberation decay (range 0-127).
- `Damping` specifies the damping (range 0-127).
- `Diffusion`

Equalizer



- `Reset EQ` sets each band at 0dB.

5 Controlling Voks Without a Graphical Interface

Voks can be controlled programmatically, either using Max messages or OSC messages. Both types of messages follow the same syntax, which is detailed in this section.

- Max messages should be sent to the `voksEngine.maxpat` object which you can obtain by downloading Voks' source code. This is how Voks works under the hood: a main patch is in charge of displaying the graphical user interface (GUI) and collecting data from the GUI and the controllers; the main patch then sends that data to `voksEngine.maxpat`, which takes care of the sound synthesis. If you go the Max route, be aware that `voksEngine.maxpat` depends on other patchers (included in the Max project source) as well as the `supervp.scrub~` external.
- OSC messages should be sent to port 7400 on the IP of the machine running Voks.

5.1 Message Syntax

Messages follow the OSC format: they are composed of an address, optionally followed by a value. An address is a sequence of strings separated with slashes `/`, which also starts with a slash. A value can be another string, an integer, or a floating-point number. Here are examples of valid Voks input messages:

```
/param/articulationSpeed 1.5
/param/pitchMode absolute
/rhythm/speed/reset
```

Such messages can be classified into two categories:

- **Parameter messages** are used to set the value of a parameter, such as rhythm control mode, pitch, etc. (those are the parameters whose value will be saved/recalled upon saving/recalling a preset file). Parameter messages are of the form:

```
/param/<name of the parameter> <value of the parameter to set>
```

The first two message examples above are of this type.

- **Non-parameter messages** are used to send one-off messages to Voks. The third example above is of this type.

The remainder of this section contains a list of all parameters, as well as a list of all Voks messages.

5.2 Voks Parameters

To set the value of any parameter in this section, send a `/param` message. For instance, to set the value of `exampleParam` to `exampleValue`, send

```
/param/exampleParam exampleValue
```

Pitch and Timbre Parameters

- `pitchMode` Specifies how to control pitch:
  - `absolute`: by specifying pitch directly (needs the original pitch to be specified too! see below)
  - `relative`: by specifying a transposition value. Warning: if in relative mode, turn vocal tension off, since it relies on pitch.
- `pitch` Pitch in MIDI semitones, if in `absolute` pitch mode. The original pitch of the sample needs to be specified too, using the parameter `originalPitch`.
- `originalPitch` Pitch of the original sample.
- `transposition` Transposition from original sample, in semitones, if in `relative` pitch mode. Replaces the need for providing a pitch and an original pitch. **Warning**: if no original pitch is provided, vocal tension should be switched off, since it relies on pitch.
- `tension`
- `tractLengthFactor` Vocal tract length stretching ($> 1.$) / shrinking ($< 1.$) factor

Time Control-Related Parameters

- `rhythmMode` Rhythm control mode: `scrub`: scrub mode; `speed`: speed mode, `syllabic`: syllabic mode.
- `sampleStart` Starting point of the loaded sample. Useful if there is no labeling file (which may happen in scrub or speed mode), discouraged otherwise.
- `sampleEnd` Ending point of the loaded sample. Useful if there is no labeling file (which may happen in scrub or speed mode), discouraged otherwise.
- `loop` 0 or 1.
  - 0 Utterances only play once in syllabic mode.
  - 1 Playback restarts once the whole utterance has been played in syllabic mode.
- `firstType` Integer.
  - If 0, no effect.
  - Else:
    - If controlling syllabic mode with bangs, the first bang received will query the labeling for the first control point **with specified type**, instead of any control point. Subsequent bangs work as expected.
    - If controlling syllabic mode with ints, any int different from the specified value is ignored, until that value is received. After that, ints work as expected.
- `continuous` 0 or 1. Only relevant in syllabic mode.

    - 0 Allow time index to jump to a higher value if the player is tapping too fast in syllabic mode.
    - 1 Force continuity of the time index in syllabic mode.
- `haste` 0 or 1. Determines what happens if user taps too fast. Only relevant when `articulationMode` is `speed` (if in duration mode, the duration parameter is always used) and `continuous` is 1 (if not continuous, we won't run late because the index can just jump to where it should be). Determines what happens if user taps too fast.
    - 0 Keep playing the signal at specified speed, possibly accumulating delay.
    - 1 Accelerate signal to keep up with the user's taps.
- `shape`
    - `fader` Set to fader mode when in syllabic mode: time index follows continuous (float) fader index given as input.
    - `linear` Time index is a series of line segments.
    - `parabolic` Time index is a series of parabola sections.
- `articulationMode` Determines if the inter-point duration is set directly (`duration`), or relative to the original sample (`speed`). See below.
- `articulationSpeed` In (non-fader) syllabic mode, if `articulationMode` is set to `speed`, time index speed is constant and equal to this parameter.
- `articulationDuration` In (non-fader) syllabic mode, if `articulationMode` is set to `duration`, time index inter-controlpoint duration is constant and equal to this parameter.

5.3 Sound Effect Parameters

- `preGain` Pre-FX gain.

Reverberation Parameters

- `reverb` 1 to apply, 0 to bypass.
- `reverbAmount` Wet/dry. 0. is original signal, 1. is completely wet signal.
- `reverbSize` Reverberation size, between 0. and 127.
- `reverbDecay` Reverberation decay, between 0. and 127.
- `reverbDamping` Reverberation damping, between 0. and 127.
- `reverbDiffusion` Reverberation diffusion, between 0. and 127.

Compressor Parameters

- `compressor` 1 to apply, 0 to bypass.
- `compAmount` Wet/dry. 0. is original signal, 1. is completely wet signal.
- `compThreshold`
- `compRatio`
- `compAttack`
- `compRelease`

Equalizer Parameters

- `equalizer` 1 to apply, 0 to bypass.
- `eqAmount` Wet/dry. 0. is original signal, 1. is completely wet signal.
- `eqGain01`, `eqGain02`, ..., `eqGain28` Gain of the 1st, 2nd, ..., 28th band, in dB.

5.4 Non-Parameter Messages

Those are the messages that you may send to Voks, but which do not set the value of a specific parameter.

- `/sample/read <optional filepath>` Load specified audio file for resequencing. Currently, only mono, 44100Hz .wav files are guaranteed to work properly. If no file path is specified, a file dialog will open to let the user select one.
- `/rhythm/scrub/read <optional filepath>` Load specified text file path to set sample bounds in scrub mode. File must obey the labeling specification, although only the values of the sample bounds will be read. If no file path is specified, a file dialog will open to let the user select one.
- `/rhythm/speed/read <optional filepath>` Load specified text file path to set sample bounds in speed mode. File must obey the labeling specification, although only the values of the sample bounds will be read. If no file path is specified, a file dialog will open to let the user select one.
- `/rhythm/syllabic/read filepath` Load specified text file path as a labeling. File must obey the labeling specification. If no file path is specified, a file dialog will open to let the user select one.
- `/rhythm/speed/play <int>` 0 stops playback, 1 resumes playback.
- `/rhythm/speed/reset` Restarts playback from beginning of sample.
- `/rhythm/syllabic/reset` Go back to beginning of utterance.
- `/rhythm <float>`
  - If in scrub mode: Scrub index, in the [0., 1.] range.
  - If in speed mode: Speed index, in the [0., 1.] range. The index is remapped continuously to around [-13., 13.] down the line: 0. gets mapped to around -13., 1. gets mapped to around 13., and there is a range with positive width around 0.5 which maps to 0.
  - If in syllabic mode: Fader index, in the [0., 1.] range.
- `/rhythm <int>`
  - If in syllabic mode: Trigger advancement to the next control point with the specified type.
- `/rhythm bang`
  - If in syllabic mode: Trigger advancement to the next control point regardless of type.

6 Development Information

This section is intended for users who want to understand/modify Voks under the hood. Voks' source code is available on this repository: https://github.com/LAM-IJLRA/Voks.

6.1 General information and architecture

Voks is written in the proprietary Max visual programming language, which means that you will need a Max license to be able to save and/or compile it.[1] The rest of this section assumes familiarity with Max.

The entrypoint is `Voks.maxproj`, which specifies which source files are part of the software, as well as some other information relevant to the project. In particular, it specifies that the file `voks.maxpat`[2] is to be opened on project load.

`voks.maxpat` contains Voks' graphical interface, and also instantiates the `voksEngine.maxpat` Max object[3]. `voksEngine.maxpat` is where the application logic and sound processing happens.

6.2 Coding conventions

To help navigate the project, some objects types have been given a distinguished color:

- Objects that contain other data (that is, subpatchers, abstractions, `coll`s, etc.) have a hot pink border.
- `value` objects corresponding to global parameters (see next section) have a sky blue border.
- Other `value` objects have a green border.

6.3 The parameter system

Overview

Parameter management is achieved mainly in the `parameters` subpatcher in `voksEngine.maxpat`. The `parameters` subpatcher receives messages in its only inlet when a parameter is updated through a controller or the graphical user interface. In turn, upon updating a parameter, it tells the rest of the application about it using the `value` (`v` for short) Max object and global messages, in Max fashion.

Preset management, that is, saving the value of all parameters on disk at once, and recalling it, is achieved through the `presetManager.maxpat` patcher, which communicates with `parameters`.

Adding and using a new parameter

To add a parameter `myParameter` to the application, edit the `parameters` subpatcher inside the `voksEngine.maxpat` patcher: add an object named `param myParameter` and connect its input to the input of the containing `parameter` subpatcher. Upon saving the

[1] `.maxpat` and `.maxproj` files use the JSON syntax, so *technically*, you could edit them by hand or using a JSON tool, but this is likely to be impractical.

[2] All `.maxpat` files are contained in the `patchers/` subdirectory.

[3] Hidden by default, toggle patching mode on using Option+Cmd+E (Apple)/Alt+Ctl+E (Windows) to see it.

patcher, `myParameter` is added to the application parameters; it can now be updated, used, and will be saved and recalled automatically by the preset manager.

To update your new parameter `myParameter` with a new value `newValue`, just send the message `/param/myParameter newValue` to `voksEngine`.

To query the value of `myParameter` anywhere in your code, create an object named `[v myParameter-PARAM]` (`v` is short for `value` here); it will output the value of the parameter any time a bang is sent to it.

If you need to be notified whenever the value of `myParameter` changes, create an object named `[r myParameter-UPDATE]` (`r` is short for `receive` here) and connect its output to the input of `[v myParameter-PARAM]`. `[r myParameter-UPDATE]` will output a bang any time the parameter is changed, making `[v myParameter-PARAM]` output the new value.

Preset management

Preset management happens in the `presetManager.maxpat` patcher. Parameter values get stored automatically as soon as the `param` object has been added (see above) and a value is sent.

Sending `write` to `presetManager`'s inlet opens a file dialog, allowing you to save a preset file on your filesystem. Sending `read` opens a file dialog, allowing you to select a previously saved preset file and recall the corresponding parameter values.