

Welcome to NRCI!

NRCI (Networked Resources for Collaborative Improvisation) is a suite of Pd tools designed to facilitate laptop ensemble performance. The NRCI project has several goals:

1. to make Pd and laptop improvisation more approachable for beginners;
2. to facilitate and stimulate creativity for advanced users;
3. to facilitate experiments with live coding (the design of software during performance);
4. and to create a simple, useful, and exciting testbed for experiments with musical networks.

Feel free to hack, modify, and abuse this code as you see fit!

Getting started:

There are four steps to loading NRCI and getting it ready for performance (plus two optional steps that may be helpful if you are using specialized audio hardware, or if you want to test to make sure that you are "getting sound").

Required:

1. make a copy of the "-workspace.pd" file
2. rename the copied file to something appropriate.
3. Launch the Pd-extended application (available from puredata.info/downloads).
4. Choose "Open" from the "File" menu to load your renamed workspace file.

Optional (for audio hardware):

5. If you are using an audio interface, you may need to choose "Preferences / Audio settings..." from the Pd-extended menu, and then select your audio interface.
6. To test your audio input/output and MIDI connections, choose "Test Audio and MIDI" from the "Media" menu.

Using the workspace:

There are two sections of the workspace:

1. The NRCI library: a series of small boxes, one instance of each type of object in the library
2. The network interface: a large box with *net-main* in the upper left-hand corner

Pd has two user interaction modes: "run" and "edit". In "run" mode you can flip switches, push buttons, drag faders, and otherwise "perform" a Pd patch. In "edit" mode you can create objects, connect them to one another, and drag them around the screen. To switch back and forth between "run" and "edit", choose "Edit Mode" from the "Edit" menu, or type "command-e".

The NRCI library:

The NRCI library is a set of modules that you can interconnect to create performance interfaces for laptop musicmaking. This is the basic principle of Pd - interconnect objects to make music. NRCI tries to make things easy by creating standardized types of objects which interconnect in standardized ways. As you learn more about Pd, you can mingle "basic" Pd objects with NRCI objects as you see fit.

Many of the NRCI modules require "default values" - user-specified settings that you type into the object after the module name. You can shift-click or right-click on a module and select "Open" to open it up and see what kinds of default values it requires; all the modules have explanatory comments about their inlets, outlets, and required default values. Examples of appropriate default values are given in all the objects loaded into the workspace.

Note that you can delete any library objects from the workspace that you aren't using; they're just there by default to remind you of what's available.

There are two varieties of modules: "control" and "audio". Control objects output occasional numbers; audio objects output continuous streams of audio data. Typically we'll design connections so that control modules tell audio modules how to behave.

Pd patches "obey the law of gravity" - information flows from the top of the screen to the bottom. Inputs to an object are at the top of the object box; outputs are at the bottom. Connect the output of one module to the input of another module below. Note that audio connections are thick black lines, while control connections are thin lines. Audio inputs can't accept control outputs, and vice versa, so if Pd refuses to let you make a connection, it's probably because you are mixing audio and control. Don't worry - you can't "break" the software - in the worst case you might get an unpleasant sound.

Control modules:

There are three types of control modules - *ctime*, *cdata*, and *ui*. *ctime* modules are rhythmic generators. They tell *cdata* modules when to produce a new data value. All *ctime* modules have an on/off toggle switch embedded; they will only produce rhythms when turned on.

cdata modules generate data values every time they receive a new timing event from a *ctime* module. They don't have on/off switches - they produce a new value whenever they receive a request for such a value from one (or many) *ctime* modules.

ui modules generate data values based upon user input, either from the mouse, or from the keyboard. Because they are rhythmically driven by the user, there's no need (or possibility) to connect *ctime* modules to *ui* modules.

Audio modules:

There are two types of audio modules - *synth~* and *sfx~*. *synth~* modules are sound generators. They take control messages which are numbers between 0 and 100 - usually that message is interpreted as a pitch, though sometimes it's used in other ways (for instance, as an index location to a soundfile). *synth~* modules produce audio output. All *synth~* modules have an on/off switch; some also have a "bang" button, which can be pressed to load a soundfile into the module.

sfx~ modules are sound processors. They take an audio input (left inlet) and a control input (right inlet). The control input should be numbers between 0 and 100; depending on the function of the module, that number may be interpreted in a number of ways, including pitch, feedback percentage, gain amount, etc. *sfx~* modules also have on/bypass switches; when the module is switched off, it will still pass through any input audio.

There are two "terminating" *sfx~* modules - *sfx-output~*, and *sfx-record~*. *sfx-output~* passes sound to the audio interface for audition; it also feeds data to the networking code that reports audio status for each connected performer. It provides an on/off switch and a number box which controls output volume in dB. *milo-sfx-record~* allows for audio to be saved to disk; click the "bang" button to specify the file name and location, then switch the module on for recording.

Structuring your patch:

Start with one or more *ctime* modules.

Connect their outputs to one or more *cdata* modules.

Connect the *cdata* outputs to *synth~* modules and to the right inlets of *sfx~* modules.

Connect *ui* modules to *synth~* modules, and to the right inlets of *sfx~* modules.

Connect the outputs of *synth~* modules to the left inlets of *sfx~* modules.

The audio outputs of *sfx~* modules can also be connected to the inputs of additional *sfx~* modules.

You can branch these structures as often as you like; you can also feed two outputs to one input.

End your structure with one (or several) *milo-sfx-output~* modules.

User interface:

In addition to clicking on the on/off toggles for *ctime*, *synth*, and *sfx* modules, you can also use the keyboard for performance. Type the backquote key (top left of the keyboard - `) to toggle between the different keyboard input modes. “Chat” mode sends your keystrokes to the chat module for communication over the network. “Modules” mode sends your keystrokes to any modules for which you have specified a default keystroke to toggle the on/off switches. And “Fretboard” mode sends keystrokes to any instances of *milo-ui-fretboard-pitches* and *milo-ui-fretboard-0100* - these modules output control values based upon the keystrokes they receive, and can be used in place of *ctime* / *cdata* pairs. “Off” mode is self-explanatory.

Setting up networking:

Make sure you are connected to the locally available wireless network.

Enter “edit mode” and change the “/username” string at the top left of the “net-main” object to your username. You may need to save, close, and reopen your workspace in order for this change to take effect.

Description of the network interface:

Click the toggles in *net-request-handler* to request pitch, amplitude, or rhythmic (duration and onset) data from other ensemble members. Requested data comes out of the four rightmost outlets of the *net-main* object; you can connect those outlets to any object that will accept control data (*cdata*, *synth~*, and *sfx~* types).

Enter “chat mode” by toggling the backquote key to type chat messages (as described above under “User interface”. Backspace deletes your entire chat message unsent; the return key sends your message. Chat messages appear in the main Pd window.

ctime modules described:

ctime-gdecrease: geometrically decreasing durations from a high threshold to a low threshold

ctime-gincrease: geometrically increasing durations from a low threshold to a high threshold

ctime-ldecrease: linearly decreasing durations from a high threshold to a low threshold

ctime-lincrease: linearly increasing durations from a low threshold to a high threshold

ctime-periodic: periodic durations

ctime-random: random durations between a low and high threshold

ctime-randomperiodic: randomly chosen periodic duration which changes randomly

ctime-rlinearhi: random durations skewed towards longer durations (within given thresholds)

ctime-rlinearlo: random durations skewed towards shorter durations (within given thresholds)

ctime-rtriangle: random durations skewed towards average durations (within given thresholds)

ctime-sinusoid: sinusoidally oscillating duration

ctime-triangular: triangularly oscillating duration
ctime-tuplet: randomly chosen tuplet subdivisions of a periodic base duration

***cdata* modules described:**

cdata-constant: outputs a constant value
cdata-cycle: outputs a repeating sequence of values, randomly changing values occasionally
cdata-heap: choose randomly between a small set of values, randomly changing them occasionally
cdata-random: outputs random values between a low and high threshold
cdata-rlinearhi: outputs random values along a linear distribution (biased towards high values)
cdata-rlinearlo: outputs random values along a linear distribution (biased towards low values)
cdata-rtriangle: outputs random values on a triangular distribution (biased towards center values)
cdata-sinusoid: outputs sinusoidally oscillating values
cdata-triangular: outputs triangularly oscillating values

***ssynth~* modules described:**

synth-flute~: noisy flute-like physical model
synth-fm~: fm synthesis
synth-grain~: granular synthesis
synth-osc~: sine oscillator
synth-playback~: sample playback (click “bang” button to choose a soundfile)
synth-pluck~: Karplus-Strong plucked-string physical model
synth-pulsenoise~: pulsed noise generator
synth-pulsetrain~: pulse-train generator (tends to be quieter than other *ssynth~*s)

***sfx~* modules described:**

sfx-ampenv~: amplitude envelope
sfx-bandpass~: bandpass filter
sfx-combfilter~: comb filter
sfx-distortion~: hyperbolic tangent waveshaping
sfx-feedback~: idiosyncratic feedback module and noise injector
sfx-ffm~: feedback FM applied to arbitrary input
sfx-fm~: FM applied to arbitrary input
sfx-gate~: switches between muting/passthrough which each data value received
sfx-highpass~: highpass filter
sfx-lowpass~: lowpass filter
sfx-output~: output module - should be the end of most signal chains!
sfx-randombpass~: bandpass filter with randomly varying filter steepness/resonance
sfx-randomcomb~: comb filter with randomly varying feedback
sfx-randomdelay~: echo with randomly varying feedback
sfx-record~: disk recorder (click circle to name soundfile, then turn on to record)
sfx-reverb~: reverberation
sfx-ringmod~: ring modulation
sfx-scrub~: delays a signal, then “sweeps” through it at faster/slower than normal rates

***ui* modules described:**

ui-fretboard-pitches: each keyboard row maps to an octave of pitches; use shift to jump 4 octaves
ui-fretboard-0100: each column of the keyboard maps to 2.5, 5, 7.5, and 10, shifting by ten to the right
ui-mouse: outputs x- and y-axis mouse positions from first two outlets; third outlet produces a “bang” message on mouse click