r_cycle

NewInControlEnvironment

r_cycle is a set of tools that enables users to easily integrate Novation MIDI controllers in their code through the Novation InControl protocol. This allows for communication with the hardware in use, quick design of UI layouts and development of customised interactions for multimedia applications. It is open source and freely available here.

Supported Hardware

- Launchpad MK2
- Launchpad Pro MK1
- Launchpad X
- Launchpad Mini MK3
- Launchpad Pro MK3

Supported Environments / Programming Languages

- Pure Data

How To Install - GLOBAL

The **r_cycle** folder must be placed in *Pd/externals* or in a folder that is listed as part of the search path in Pd (to check in Pd go to: *Preferences->Path*).

r_cycle objects can then be accessed in the following ways:

- adding **[declare -path r_cycle]** to the patch. **RECOMMENDED**
- using the following syntax: r_cycle/name_of_the_object_ - ie **[r_cycle/COLORS]**. This is **NOT RECOMMENDED** because the length of the name of some objects could affect their graphical representation (ie *widgets*).

How To Install - LOCAL

It is also possible to use **r_cycle** locally (ie make it only available to specific patches in a specific folder). In order to do that the user must place the r_cycle folder in the main folder where the patch lives and then use [declare -path r_cycle] to *import* the library.

For example, assuming the user is working on a new patch called *mysequencer.pd*, that lives in the *myapp* folder, in order to use **r_cycle** the user must:

- place the r_cycle folder in the *myapp* folder
- use [declare -path r_cycle] in *mysequencer.pd*

This approach has the advantage of making the project easier to share (compared to the GLOBAL approach from above), since the user can share the folder *myapp* without worrying about providing instructions about how to install r_cycle (see *How To Install - GLOBAL*). The downside of having a *local* version of **r_cycle** though, is that it is easy to end up with several copies of **r_cycle** in different places on the hard drive (to be noticed that **r_cycle** is less than 1 MB though - audio files excluded).

List of objects

Widgets

- *doc* **[LP_GUI]** The middle man between LP and Pd objects - required for adding objects to the surface of a LP and receiving messages from a LP
    - arg 1: (float) LP_GUI_id - unique identifier
    - inlet 1:
        - **[set *list*(** (NOT IN USE - SHOULD BE AVOIDED) set specific pad to a color - list of 3 floats: pad_id, color, status (0-1)
        - **[reset(** reset surface status
        - **[_HB_off(** turn off *heartbeat*
        - **[_buffers(** show the internal buffers
        - **[device(** Launchpad device in use: *LP2*, *LP_Pro*, *LP_X*, *LPM_MK3*, *LP_Pro_MK3*
        - **[daw_mode *float*(** enables/disables *daw mode* on devices that support it
        - **[programmer_mode *float*(** enables/disables *programmer mode* on devices that support it
- *doc* **[DRUM_PADS]** needs **[LP_GUI]** - creates grids of drum pads on LP
    - arg 1: (float) x-offset
    - arg 2: (float) y-offset
    - arg 3: (float) size of the square
    - arg 4: (float) color (0-127)
    - arg 5: (float) array mode - when *float* is different from 0 pads are displayed sequentially (left-right bottom-top)
    - arg 6: (float) LP_GUI_id
    - inlet 1:
        - (bang) reset the object
        - **[map *list*(** map pads to specific numbers (generally MIDI notes) - ie for a 2x2 grid: **[map 10 20 11 21(** - pads are numbered from bottom-top left-right
        - **[reset_map(** use original pad indexing
        - **[color_bg *float*(** background color
        - **[color_fg *float*(** foreground color (ie selected pad)
        - **[print_args(** print to the console a list of all the arguments
    - outlet 1: (float) pressed pad id
    - outlet 2: (float) pad velocity
    - outlet 3: (float) pad aftertouch
    - outlet 3: (float) pad polytouch

- *doc* **[KEYBOARD]** needs **[LP_GUI]** - creates a chromatic keyboard on LP surface
    - arg 1: (float) keyboard row (0-6 - bottom to top)
    - arg 2: (float) pitch offset in semitones
    - arg 3: (float) LP_GUI_id
    - inlet 1:
        - (bang) reset the object
        - **[transpose *float*(** pitch offset in semitones
        - **[octave *float*(** octave offset
        - **[color_bg *float*(** background color
        - **[color_fg *float*(** foreground color (ie selected pad)
        - **[print_args(** print to the console a list of all the arguments
    - outlet 1: (float) pressed pad id - MIDI note
    - outlet 2: (float) pad velocity
    - outlet 3: (float) pad aftertouch
    - outlet 3: (list) 2 floats - pad polytouch
- *doc* **[BUTTON]** needs **[LP_GUI]** - sets a pad to behave like a button (trigger)
    - arg 1: (float) pad nr (on LP)
    - arg 2: (float) color (0-127)
    - arg 3: (symbol) symbol to output when pad is pressed
    - arg 4: (float) LP_GUI_id
    - inlet 1:
        - (bang) reset the object
        - **[color_bg *float*(** background color
        - **[color_fg *float*(** foreground color (ie selected pad)
        - **[toggle *float*(** toggle mode off/on
        - **[off_on *float*(** turn button off and on
        - **[print_args(** print to the console a list of all the arguments
    - outlet 1: (symbol) symbol passed as argument
    - outlet 2: (float) status: 0. released, 1. pressed
- *doc* **[RADIO]** needs **[LP_GUI]** - sets pads/buttons to behave like a radio button / slider
    - arg 1: (float) pad nr (on LP) - represents the starting point of the UI object
    - arg 2: (float) length (1-8)
    - arg 3: (float) ON color (0-127)
    - arg 4: (float) OFF color (0-127)
    - arg 5: (float) orientation - horizontal/vertical (0-1)
    - arg 6: (float) LP_GUI_id
    - inlet 1:
        - (bang) reset the object
        - **[color_bg *float*(** background color
        - **[color_fg *float*(** foreground color (ie selected pad)
        - **[select *float*(** allows to select value
        - **[print_args(** print to the console a list of all the arguments
    - outlet 1: (float) absolute value (0-1)

    - outlet 2: (float) value (0-(length-1))
    - outlet 3: (float) status: 0. released, 1. pressed

- *doc* **[GRID]** needs **[LP_GUI]** - sets pads/buttons to behave like a 'sequence' (a programmable series of steps)

    - arg 1: (float) pad nr (on LP) - represents the starting point of the UI object
    - arg 2: (float) length (1-8)
    - arg 3: (float) OFF color (0-127)
    - arg 4: (float) LP_GUI_id
    - inlet 1:
        - **[print_args(** print a list of all arguments in the Pd console
        - **[clear(** clear sequence
        - **[dump(** dump grid status from outlet 2
        - **[get_size(** return size of the grid - useful for [PATTERN]
        - **[steps list(** list of pairs of floats where 1st is the step (== x6 ticks) number and the 2nd is its value (==0 or !=0)
        - **[raw_ticks list(** list of ticks values - no tick number is needed (ie 1 0 1 0 would set tick 1 to 1, tick 2 to 0, tick 3 to 1 and tock 4 to 0) - mainly useful when dumping values out of [PATTERN]
        - **[pointer *float*(** expects a step nr to highlight it white
        - **[color_bg *float*(** background color
        - **[color_fg *float*(** foreground color (ie selected pad)
    - outlet 1: (list) step nr + value, when value change
    - outlet 2: (list) grid status, list of pairs: step nr + value

Audio-MIDI

- *doc* **[TO_LPP]**

    - inlet 1:
        - (float) pad/button nr - see HW layouts below
        - (list) pairs of elem_id+color
        - (symbol) button name
        - **[clear(** clear the whole surface
        - **[column *float*(** light up a specific column
        - **[row *float*(** light up a specific row
        - **[setall *float*(** set all the pad to a specific color
        - **[mode *float*(** 0.static, 1.flash, 2.pulse
        - **[mode *symbol*(** *static*, *flash*, *pulse*
        - **[text *symbol float float*(** text to print, loop (0-1), speed (1-7)
        - **[text_stop(** stop text scrolling
        - **[rgb *list*(** elem_id, R, G, B - this can be repeated up till 81 times in the same list - RGB in a range 0-63
    - inlet 2: (float) color

- *doc* **[FROM_LPP]**

    - arg 1: (float) MIDI channel (1-16)

    - arg 2: (float) note/CC loopback (0-1) - visual feedback when pressing pads/buttons
    - arg 3: (float) ON color (0-127)
    - inlet 1:
        - (float) OFF/ON (0-1) - activate/deactivate the object
        - **[print_args(** print a list of all arguments in the Pd console
    - outlet 1: (float) row (of a pressed pad/button)
    - outlet 2: (float) column (of a pressed pad/button)
    - outlet 3: (symbol) button name
    - outlet 4: (float) note
    - outlet 5: (float) velocity
    - outlet 6: (list) polytouch - pair of floats: 1.pad nr, 2.value
    - outlet 7: (float) aftertouch
    - outlet 8: (float) cc value
    - outlet 9: (float) cc number

- *doc* **[TO_LP2]**

    - inlet 1:
        - (float) pad/button nr - see HW layouts below
        - (list) pairs of elem_id+color
        - (symbol) button name
        - **[clear(** clear the whole surface
        - **[column *float*(** light up a specific column
        - **[row *float*(** light up a specific row
        - **[setall *float*(** set all the pad to a specific color
        - **[mode *float*(** 0.static, 1.flash, 2.pulse
        - **[mode *symbol*(** *static*, *flash*, *pulse*
        - **[text *symbol float float*(** text to print, loop (0-1), speed (1-7)
        - **[text_stop(** stop text scrolling
        - **[rgb *list*(** elem_id, R, G, B - this can be repeated up till 81 times in the same list - RGB in a range 0-63
    - inlet 2: (float) color

- *doc* **[FROM_LP2]**

    - arg 1: (float) MIDI channel (1-16)
    - arg 2: (float) note/CC loopback (0-1) - visual feedback when pressing pads/buttons
    - arg 3: (float) ON color (0-127)
    - inlet 1:
        - (float) OFF/ON (0-1) - activate/deactivate the object
        - **[print_args(** print a list of all arguments in the Pd console
    - outlet 1: (float) row (of a pressed pad/button)
    - outlet 2: (float) column (of a pressed pad/button)
    - outlet 3: (symbol) button name
    - outlet 4: (float) note
    - outlet 5: (float) velocity
    - outlet 6: (list) polytouch - pair of floats: 1.pad nr, 2.value
    - outlet 7: (float) aftertouch

    - outlet 8: (float) cc value
    - outlet 9: (float) cc number

- *doc* **[TO_LPP3]**

    - inlet 1:
        - (float) pad/button nr - see HW layouts below
        - (list) pairs of elem_id+color
        - (symbol) button name
        - **[clear(** clear the whole surface
        - **[column *float*(** light up a specific column
        - **[row *float*(** light up a specific row
        - **[setall *float*(** set all the pad to a specific color
        - **[mode *float*(** 0.static, 1.flash, 2.pulse
        - **[mode *symbol*(** *static*, *flash*, *pulse*
        - **[text *symbol float float*(** text to print, loop (0-1), speed (0-127 - 2s complement - from 64 to 127 is reverse scrolling)
        - **[text_stop(** stop text scrolling
        - **[rgb *list*(** elem_id, R, G, B - this can be repeated up till 81 times in the same list - RGB in a range 0-127
    - inlet 2: (float) color

- *doc* **[FROM_LPP3]**

    - arg 1: (float) MIDI channel (1-16)
    - arg 2: (float) note/CC loopback (0-1) - visual feedback when pressing pads/buttons
    - arg 3: (float) ON color (0-127)
    - inlet 1:
        - (float) OFF/ON (0-1) - activate/deactivate the object
        - **[print_args(** print a list of all arguments in the Pd console
    - outlet 1: (float) row (of a pressed pad/button)
    - outlet 2: (float) column (of a pressed pad/button)
    - outlet 3: (symbol) button name
    - outlet 4: (float) note
    - outlet 5: (float) velocity
    - outlet 6: (list) polytouch - pair of floats: 1.pad nr, 2.value
    - outlet 7: (float) aftertouch
    - outlet 8: (float) cc value
    - outlet 9: (float) cc number

- *doc* **[TO_LP_X]**

    - inlet 1:
        - (float) pad/button nr - see HW layouts below
        - (list) pairs of elem_id+color
        - (symbol) button name
        - **[clear(** clear the whole surface
        - **[column *float*(** light up a specific column

            - **[row *float*(** light up a specific row
            - **[setall *float*(** set all the pad to a specific color
            - **[mode *float*(** 0.static, 1.flash, 2.pulse
            - **[mode *symbol*(** *static*, *flash*, *pulse*
            - **[text *symbol float float*(** text to print, loop (0-1), speed (0-127 - 2s complement - from 64 to 127 is reverse scrolling)
            - **[text_stop(** stop text scrolling
            - **[rgb *list*(** elem_id, R, G, B - this can be repeated up till 81 times in the same list - RGB in a range 0-127
        - inlet 2: (float) color

- *doc* **[FROM_LP_X]**

    - arg 1: (float) MIDI channel (1-16)
    - arg 2: (float) note/CC loopback (0-1) - visual feedback when pressing pads/buttons
    - arg 3: (float) ON color (0-127)
    - inlet 1:
        - (float) OFF/ON (0-1) - activate/deactivate the object
        - **[print_args(** print a list of all arguments in the Pd console
    - outlet 1: (float) row (of a pressed pad/button)
    - outlet 2: (float) column (of a pressed pad/button)
    - outlet 3: (symbol) button name
    - outlet 4: (float) note
    - outlet 5: (float) velocity
    - outlet 6: (list) polytouch - pair of floats: 1.pad nr, 2.value
    - outlet 7: (float) aftertouch
    - outlet 8: (float) cc value
    - outlet 9: (float) cc number

- *doc* **[TO_LPM_MK3]**

    - inlet 1:
        - (float) pad/button nr - see HW layouts below
        - (list) pairs of elem_id+color
        - (symbol) button name
        - **[clear(** clear the whole surface
        - **[column *float*(** light up a specific column
        - **[row *float*(** light up a specific row
        - **[setall *float*(** set all the pad to a specific color
        - **[mode *float*(** 0.static, 1.flash, 2.pulse
        - **[mode *symbol*(** *static*, *flash*, *pulse*
        - **[text *symbol float float*(** text to print, loop (0-1), speed (0-127 - 2s complement - from 64 to 127 is reverse scrolling)
        - **[text_stop(** stop text scrolling
        - **[rgb *list*(** elem_id, R, G, B - this can be repeated up till 81 times in the same list - RGB in a range 0-127
    - inlet 2: (float) color

- *doc* **[FROM_LPM_MK3]**
    - arg 1: (float) MIDI channel (1-16)
    - arg 2: (float) note/CC loopback (0-1) - visual feedback when pressing pads/buttons
    - arg 3: (float) ON color (0-127)
    - inlet 1:
        - (float) OFF/ON (0-1) - activate/deactivate the object
        - **[print_args(** print a list of all arguments in the Pd console
    - outlet 1: (float) row (of a pressed pad/button)
    - outlet 2: (float) column (of a pressed pad/button)
    - outlet 3: (symbol) button name
    - outlet 4: (float) note
    - outlet 5: (float) velocity
    - outlet 6: (float) cc value
    - outlet 7: (float) cc number
- *doc* **[SIMPLE_PLAYER]** simple sample playback
    - arg 1: (symbol) audio file
    - inlet 1:
        - (bang) to play sample
        - **[load(** to select a file to load
        - **[read *symbol*(** to load a specific audio file
    - outlet 1: audio signal - sample playback
- *doc* **[MULTI_SAMPLER]** multi-sample player
    - arg 1: (float) number of voices
    - inlet 1:
        - (float) pad_id
        - **[load(** to select a file to load
        - **[read *float text*(** to load a specific audio file (ie [read 2 ../audio/kick.wav(where the *float* is the voice nr and the text the file to load)
    - inlet 2: (float) velocity
    - outlet 1: audio signal - sample playback (all samples)
- *doc* **[MONO_MIDI]** ignore note off messages in case don't come from last pressed key
    - inlet 1: (float) note on/off
    - inlet 2: (float) velocity
    - outlet 1: (float) note on/off
    - outlet 2: (float) velocity
- *doc* **[OSCILLOSCOPE]** digital scope with zoom functions - useful for displaying audio signals
- *doc* **[BAND_LIMITED]** band-limited mono synth (sine/triangle/sawtooth/square)
    - inlet 1: (float) MIDI note (internally clipped to 106 for aliasing reasons)
    - inlet 2: (float) volume (0-1)
    - inlet 3: (float) waveform (0-3)

    - outlet 1: audio signal

- *doc* **[TAPE_DELAY]** interpolated feedback delay - time parameter is interpolated to emulate the tape effect

    - inlet 1: audio signal to delay
    - inlet 2:
        - **[time *float*(** delay time in msec (0-1000)
        - **[feedback *float*(** delay feedback (0-1)
        - **[dry_wet *float*(** dry/wet (0-1)

- *doc* **[G_SYNTH]** 4 grains granular module

    - inlet 1:
        - **[glissando float(** set glissando in msec (0-10000)
        - **[pitch float(** set pitch (0-1) (this is the ratio - ie 1 is original pitch, 0.5 one octave down, 2 one octave up)
        - **[spray float(** set spray (0-1) - distribution of single grains in the stereo field
        - **[offset float(** set offset (0-1) - reading point in the audio buffer
        - **[volume float(** set volume (0-1)
        - **[reset_phase(** reset internal phasors
        - **[read text(** audio file to read (ie [read ../audio/my_sample.wav( )
    - outlet 1: left audio signal
    - outlet 2: right audio signal

- *doc* **[WAVESHAPER]** simple distortion using transfer functions

    - inlet 1: audio signal
    - inlet 2:
        - **[mix float(** wet-dry parameter (0-1)
        - **[gain float(** audio input gain (default 1)
        - **[function float(** waveshaper function (0-6) - 0 LINEAR - 1 POWER FUNCTION - 2 EXPONENTIAL FUNCTION - 3 CHEBYCHEV_2 - 4 CHEBYCHEV_3 - 5 CHEBYCHEV_4 - 6 CHEBYCHEV_5
    - outlet 1: audio signal

- *doc* **[ENVELOPE]** simple attack/release envelope - outputs an audio signal

    - inlet 1:
        - (float) when ==0 output signal goes from 1 to 0 in 'release' time, when !=0 output signal goes from 0 to 1 in 'attack' time
        - **[attack *float*(** attack time in msec
        - **[release *float*(** release time in msec
    - outlet 1: audio signal - envelope

Misc

- *doc* **[QUERY]** queries al r_cycle objects

    - outlet 1: return print_inlets, print_outlets, print_args and print_methods in accord with the button pressed

- *doc* **[BPM2MS]** converts from bpm to milliseconds and the other way around

    - inlet 1: (float) number to convert
    - outlet 1: (float) number converted

- *doc* **[DEVICE_SELECT]** opens dialog window to select AUDIO/MIDI device inputs and outputs

- *doc* **[SQUARE]** generates squares to send to **[LP_GUI]**

    - inlet 1: (list) starting point on LP (bottom-left corner) + square size in nr of pads
    - inlet 2: (float) color (0-127)
    - outlet 1: (list) pairs of pads (id) + colors

- *doc* **[COLORS]** preview 128 LP colors and return color values

    - outlet 1-2-3: (float) R G B values (0-255)
    - outlet 4: (float) color representation for Pd GUI objects

- *doc* **[COLORS_NO_GUI]** same as **[COLORS]** but without visual feedback (no graph-on-parent)

    - outlet 1-2-3: (float) R G B values (0-255)
    - outlet 4: (float) color representation for Pd GUI objects

- *doc* **[DSP_TOGGLE]** turns on/off Pure Data DSP (audio on/off)

    - inlet 1: (float) off/on (0-1)

- *doc* **[INC_DEC]** increment/decrement number

    - inlet 1: (bang) to increment
    - inlet 2: (bang) to decrement
    - inlet 3: (float) set next number
    - outlet 1: (float) number

- *doc* **[EU_R]** quasi-euclidean generator

    - inlet 1:
        - **[print(** print sequence to console
        - **[length *float*(** sequence length (0-16)
        - **[ones *float*(** number of onsets in the sequence (0-length)
        - **[offset *float*(** sequence offset (0-length)
        - **[get *float*(** return value of the selected step (0-1) - this is generally used to read the sequence
    - outlet 1: (float) value (0-1) of the requested step (see *get $1* above)

- *doc* **[PRINT_2]** allows to visualize text in the patch instead of on the console

    - inlet 1: can be everything - float, symbol, etc. **[_clean(** to clean the text

- *doc* **[REPEAT]** repeats a floating point value an *n* number of times every *t* msec

    - inlet 1:
      - *float* to be repeated - list to be repeated (circular buffer) - **[print_args(** print all arguments to the console - **[msec $1(** repetition rate in msec - **[repeat(** number of repetitions (>0) - if <1 it would still generate 1 output - **[stop(** immediately stop repetitions
    - outlet 1: (float) value == input

- *doc* **[CLOCK]** sequencer clock with transport and (optional) MIDI out - MIDI based with 96 ticks per bar

    - arg 1: (float) unique identifier
    - inlet 1:
      - **[play(** send 'play' msg and start clock
      - **[stop(** send 'stop' msg and stop clock
      - **[continue(** send 'continue' msg and stop clock
      - **[midi *float*(** enable/disable MIDI output (0-1)
      - **[bpm *float*(** set bpm
    - outlet 1: (bang) outputs a 'bang' per tick
    - outlet 2: (symbol) outputs 'play/stop/continue' in accord with the CLOCK status

- *doc* **[PATTERN]** monophonic pattern - needs to be linked to a [CLOCK]

    - arg 1: (float) unique identifier of the [CLOCK] [PATTERN] is linked to
    - inlet 1:
      - **[size *float*(** size of the sequence in steps (1/16th)
      - **[clear(** clear sequence
      - **[direction *float*(** 0: normal, 1: reverse
      - **[set *float float*(** set values in ticks (ie [0 1, 24 20(- sets tick 0 to 1 and tick 24 to 20)
      - **[step *float float*(** set values in steps (ie [0 5, 14 35(- sets step 0 to 5 and step 14 to 35)
      - **[steps *list*(** list of pairs of floats where 1st is the step (== x6 ticks) number and the 2nd is the value (==0 or !=0)
      - **[sync_rate *float*(** integer in a range 0-11, corresponding to the following sync rates: 1, 1T, 2, 2T, 4, 4T, 8, 8T, 16, 16T, 32, 32T
    - outlet 1: (float) outputs the current step nr
    - outlet 2: (bang) outputs a 'bang' per tick
    - outlet 3: (float) outputs the value of the current tick
    - outlet 4: (list) outputs a list of pairs of floats, where the 1st is the ID and the 2nd its value

- *doc* **[PROBE]** display audio signal value

    - inlet 1: audio signal

Guidelines

Here are some **r_cycle** *standards*, also useful to better understand the overall architecture. Respecting these standards makes our lives easier 😃

- [send] and [return] names always have a the following form: $0_name, unless they need to be GLOBAL

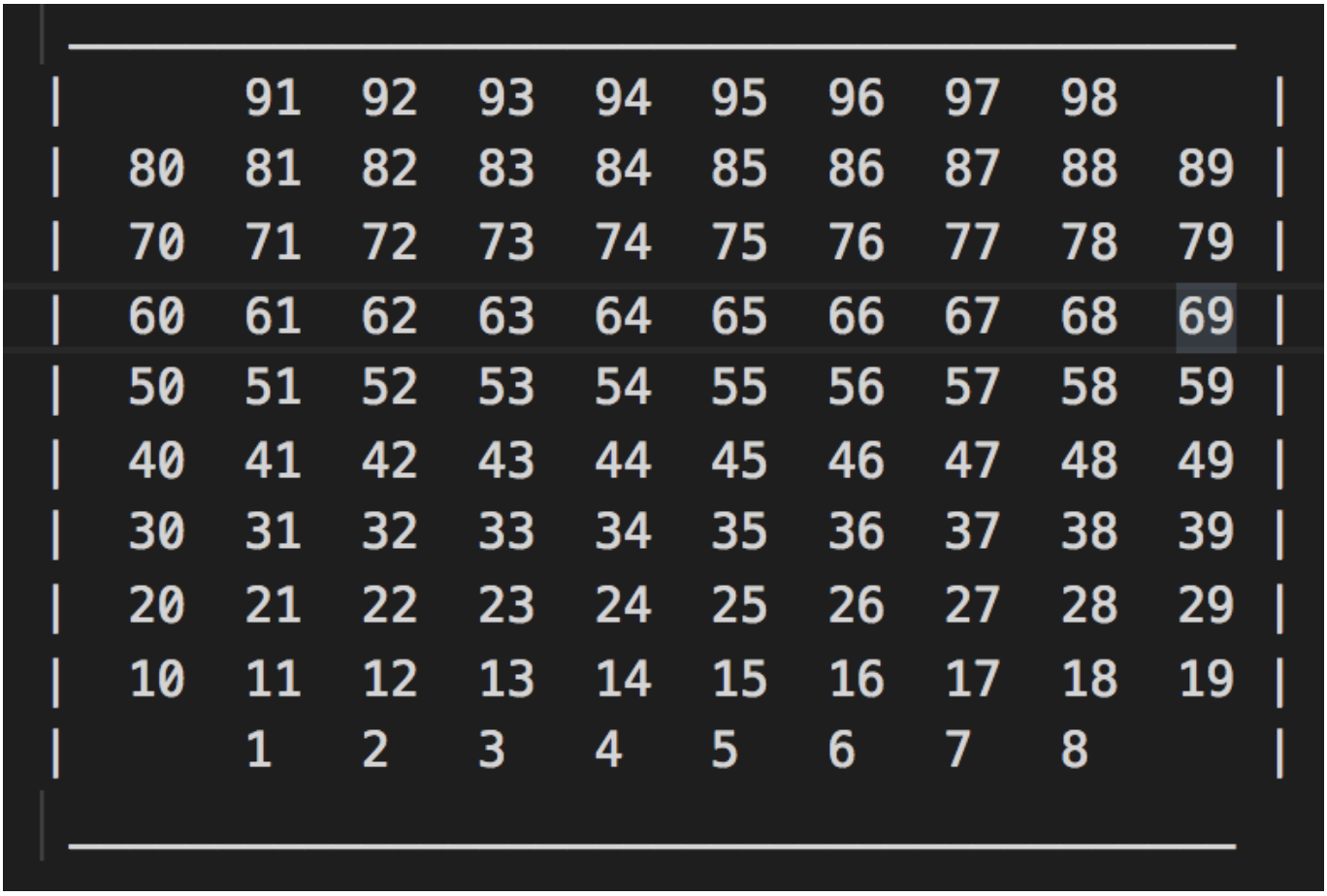
- When Pd GUI objects (ie [bng], [vradio], etc.) use internal sends and returns (right-click->Properties), these always use names with the following structure (unless they need to be GLOBAL):
    - SEND: $0_from_g_name
    - RETURN: $0_to_g_name where **g** stands for GUI. This makes easier interacting with them.
- Try to avoid to use [s~] and [r~] (it's fine for control signals)
- comments, comments, comments, comments, comments, ... use as many comments as you can, let other people understand what's going on in your head!
- we normally use 2 numeric ranges and do the **scaling** inside the various objects:
    - (int) with range 0-127 - MIDI (to be noted that in Pd all numbers are floats)
    - (float) with range 0-1 basically when it's not MIDI is better to have a **standard** so that it's easier to connect different objects without worrying about having to scale numbers
- **abstractions** and **subpatches** respect the right-to-left order of operations
- all r_cycle objects must have a **print_args** method, so all arguments specific to an object can be printed in the console when a [print_args(message is received(
- all r_cycle objects must have a **print_methods** method, so all methods specific to an object can be printed in the console when a [print_methods(message is received
- all r_cycle objects must have a **print_outlets** method, so the function of all outlets of an object can be printed in the console when a [print_outlets(message is received
- always use [trigger], don't connect 1 outlet to multiple inlets without using this object. It's important for readibility and debugging
- place all audio files in the *audio* folder
- abstractions that use visible native Pd GUI objects with labels (via graph-on-parent), must use methods that match the labelling (ie the method to control a slider labelled 'freq' must be [freq $1( )
- methods (ie [*methodname* $1( ) must be sent to the leftmost inlet. The only exception is audio objects in which the rightmost inlet is used
- all object descriptions in README.md must use the keyword _*doc*_ in order to be found by [LP_GUI]'s *print_objects* method
- all the subpatches must have all the [inlet]s visible at the top of the patch and all the [outlet]s visible at the bottom of the patch

Compatibility

- Pure Data >=0.50

Hardware Layout - LP Pro

Pad/Button IDs in decimal



	91	92	93	94	95	96	97	98	
80	81	82	83	84	85	86	87	88	89
70	71	72	73	74	75	76	77	78	79
60	61	62	63	64	65	66	67	68	69
50	51	52	53	54	55	56	57	58	59
40	41	42	43	44	45	46	47	48	49
30	31	32	33	34	35	36	37	38	39
20	21	22	23	24	25	26	27	28	29
10	11	12	13	14	15	16	17	18	19
	1	2	3	4	5	6	7	8	

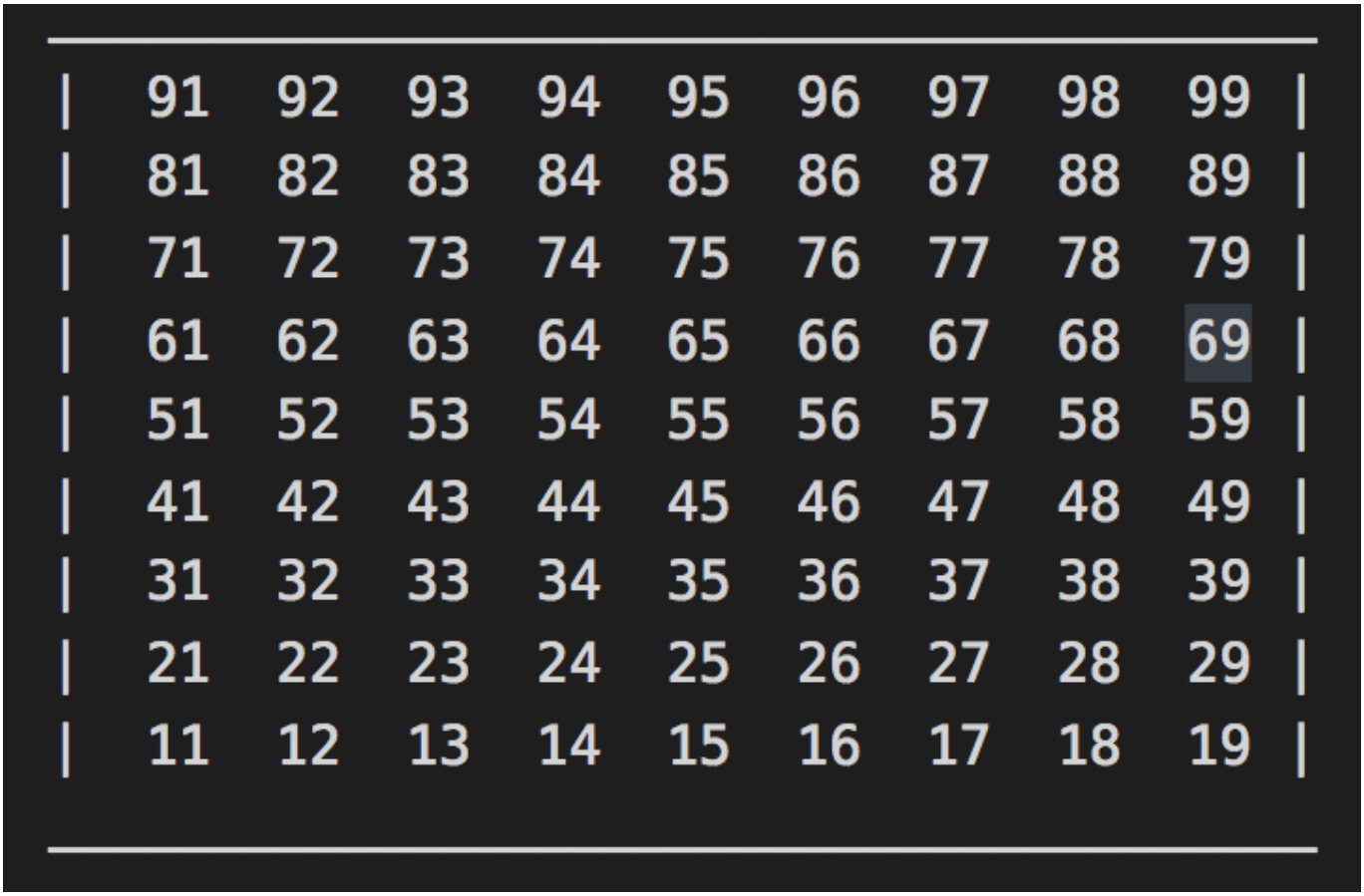
Hardware Layout - LP Pro MK3

Pad/Button IDs in decimal

90	91	92	93	94	95	96	97	98	99
80	81	82	83	84	85	86	87	88	89
70	71	72	73	74	75	76	77	78	79
60	61	62	63	64	65	66	67	68	69
50	51	52	53	54	55	56	57	58	59
40	41	42	43	44	45	46	47	48	49
30	31	32	33	34	35	36	37	38	39
20	21	22	23	24	25	26	27	28	29
10	11	12	13	14	15	16	17	18	19
	101	102	103	104	105	106	107	108	
0	1	2	3	4	5	6	7	8	

Hardware Layout - LP Mini MK3 / X

Pad/Button IDs in decimal



Pure Data SDK

Architecture Overview

[LP_GUI] is the core object used to talk to the hardware. It is the only object that has knowledge of the hardware and talks with it directly. It is possible to use multiple **[LP_GUI]** objects simultaneously by setting a unique identifier as an argument. Objects like **[KEYBOARD]** and **[DRUM_PADS]** are widgets that allow the hardware to interact with Pure Data (ie **[KEYBOARD]** could control a synth). These are also in charge of generating visual feedback on the hardware. When MIDI events are received **[LP_GUI]** talks to the right object which decides what to do with that info and lets **[LP_GUI]** know (ie pressing a specific pad lights up the whole surface)

Widget Architecture

Widgets must reply to **[LP_GUI]** *heartbeat* with their *obj_unique_id* - **[list obj_$0]**

Widgets must only communicate their attributes once, when requested by **[LP_GUI]** (this means objects shouldn't send anything to **[LP_GUI]** when created - ie using **[loadbang]** - see **[KEYBOARD]** and **[DRUM_PADS]**)

In all the objects the real-time processing happens in **[pd PROCESSING]**

- here the object receives info from **[LP_GUI]** when a pad/button is pressed on the HW in the form of a list: *elem_id*, *color*, *status* plus special messages: [velocity $1(, [polytouch $1 $2(and [aftertouch $1(-

these are sent prior to the list
- the object must reply with a list of: *elem_id*, *color*. This allows to create custom animations (ie pressing 1 pad makes the whole surface pulse for 1 sec)
- Objects must have a *print_args* method that prints a list of all the arguments for the object in the console
- Objects must have a *print_methods* method that prints a list of all the methods for the object in the console
- Objects must have a *print_outlets* method that prints a list of the functions of all outlets of the object in the console

Communication Protocol - LP_GUI <-> Widgets

The communication between **[LP_GUI]** (the hub) and the widgets is bi-directional.

[LP_GUI] talks to the objects in the following ways:

- sends a *heartbeat* to all the objects to check they are alive
- initialize the objects to get all their attributes (only once)
- when receiving MIDI from hardware, sends values to the widgets in the form of a list containing: *elem_id* (pads/buttons), color, status (pressed/released) plus special messages: [velocity $1(, [polytouch $1 $2((**aftertouch** is a global send [s _aftetouch] since it cannot be associated to a specific pad)- these are sent before the list

All widgets report the info associated to them in a form of a list containing first the *obj_unique_id* (symbol/string) the 1 or more pairs of integers representing *elem_id* (pads/buttons) and color.

All widgets, when receiving values from **[LP_GUI]** (ie because a button has been pressed on the hardware) and performing necessary processing, send a list of integers containing: *elem_id* and *color* back to **[LP_GUI]**.

Useful Links

- r_cycle
- Launchpad MK2 Programmer's Reference Guide
- Launchpad Pro Programmer's Reference Guide
- Launchpad X Programmer's Reference Guide
- Launchpad Mini MK3 Programmer's Reference Guide
- Novation Components
- Pure Data

Copyright

Except as otherwise noted:

License

see LICENSE