norns script reference

This script reference is a supplement to the complete norns API. Here, you'll find examples which explore single components of the API, which can be copied/pasted into your own scripts and manipulated for your needs. These examples assume familiarity with the scripting concepts covered in the norns studies.

To access the complete norns API, you can either:

- connect to maiden and navigate to `norns.local/doc` in your browser
- visit the static API

▼ sections

modules

Linked module names below go to extended reference examples, which illustrate how to use the commands in different contexts. **(api)** links go to the static API.

Module	API	Description
arc	(api)	Connect a script to a hardware arc
audio	(api)	Directly set system audio levels
clock	(api)	Coroutine system which executes functions on beat-synced and free-running schedules
controlspec	(api)	PARAM menu control constructor with presets
crow	(api)	Connect a script to a hardware crow
encoders	(api)	Decipher the norns on-board encoders
engine	(api)	Register a SuperCollider engine
gamepad	(api)	Connect a script to a gamepad controller
grid	(api)	Connect a script to a hardware grid
hid	(api)	Connect a script to HID hardware

keyboard	(api)	Decipher keyboard (typing, not piano) input
metro	(api)	High-resolution time-based counter
midi	(api)	Connect a script to MIDI hardware
norns	(api)	System utilities
osc	(api)	Connect a script to OSC streams
params.binary	(api)	Binary (toggling) parameters
params.control	(api)	Granular control over parameter values
params.file	(api)	An easy way to load files into scripts via parameters
params.group	(api)	Groups of parameters
params.number	(api)	Defines parameters which require discrete incrementing values
params.option	(api)	Select a parameter from a list
params.separator	(api)	Separator between parameters
params.taper	(api)	Non-linear parameter using @catfact's taper function
params.text	(api)	Parameter for text entry
params.trigger	(api)	Parameter for an "on/off" action trigger
paramset	(api)	Create script parameters, such as in the PARAMETERS menu
poll	(api)	System polling for CPU, incoming/outgoing amplitude, and incoming pitch
screen	(api)	Draw to the norns on-board screen
softcut	(api)	Two audio buffers which can be recorded into and played by six individual voices
lib/beatclock	(api)	Older clock library - see clock instead
lib/elca	(api)	Elementary cellular automata generator
lib/envgraph	(api)	Envelope graph drawing module
lib/er	(api)	Euclidean rhythm generator
lib/fileselect	(api)	File select utility
lib/filtergraph	(api)	Filter graph drawing module
lib/filters	(api)	Value smoother
lib/formatters	(api)	PARAM menu formatter functions
lib/graph	(api)	Graph drawing module
lib/intonation	(api)	Library of various tunings, including 12 tone and gamuts
lib/lattice	(api)	Simple and extensible sequencers driven by a superclock
lib/lfo	(api)	Single-clock framework for generating movement (beat-synced or free) inside of a script
lib/listselect	(api)	List select utility
lib/musicutil	(api)	Utility module for common music maths
lib/pattern_time	(api)	Timed-event pattern recorder / player
lib/reflection	(api)	Record clock-synced changes to data over time, with variable-rate playback, overdubbing, and pattern management tools
lib/sequins	(api)	Build and modify sequencers + arpeggiators
lib/tabutil	(api)	Table utilities, also available in global variable `tab`
lib/timeline	(api)	Sequence events in time
lib/textentry	(api)	Text entry UI
lib/ui	(api)	UI widgets module
lib/util	(api)	Helpful utility functions, also available in global variable `util`

lib/voice	(api)	Experimental voice allocation module

folder structure

Scripts are located in `~/dust/code/` , and are what make norns do things. A script consists of at least a Lua file but can additionally also contain supporting Lua libraries, SuperCollider engines and data.

```
myscript/
  myscript.lua -- main version, shows up as MYSCRIPT
  mod.lua -- alt version, shows up as MYSCRIPT/MOD
  README.md -- main docs/readme
  data/
    myscript-01.pset -- pset, loaded via params:read(1) or via menu
  lib/
    somelib.lua -- arbitrary lib, imported via require 'lib/somelib'
    some-engine.sc -- engine file
    some-engine.lua -- engine lib, require lib/some-engine'
  docs/ -- more documentation, won't be shown in SELECT
    more-docs.md
```

basic script

```
-- scriptname: short script description
-- v1.0.0 @author
-- llllllll.co/t/22222

engine.name = 'PolySub'

function init()
  -- initialization
end

function key(n,z)
  -- key actions: n = number, z = state
end

function enc(n,d)
  -- encoder actions: n = number, d = delta
end

function redraw()
  -- screen redraw
end

function cleanup()
  -- deinitialization
end
```

setting minimum required version

Each norns update brings new features, both for scripting and general usability. If you share a script which relies on the features of a particular norns update, you can specify a minimum version requirement which will display a message on the norns screen if the script is loaded on a previous version of the software.

To specify a minimum version required, set `norns.version.required` to the **YYMMDD** value before the `init()` of your script, eg:

```
-- minimum required version check

norns.version.required = 221214 -- can be a number or string, formatted YYMMDD

function init()
  print('hello, welcome to the script')
```

```
end
```

If this script is loaded on a norns running software earlier than `221214`, you'll see this on the norns screen:

```
error: version 221214 required
try 'SYSTEM > UPDATE'
```

And this will print to maiden:

```
### SCRIPT ERROR: version 221214 required
### try 'SYSTEM > UPDATE'
### or check for new disk image
```

helpful system commands and variables

There are some commands and variables which are helpful for common scripting tasks. Please note that while these tables may contain additional commands and variables, any not mentioned in this section will not be useful for standard scripting.

These semicolon-preceded commands are shortcuts which are run from maiden directly:

command	description
`;restart`	When executed in maiden, this command restarts the environment – run this command in the `matron` tab to restart the Lua layer, or run it in the `supercollider` tab to restart SuperCollider
`;install GITHUB_URL`	When executed in maiden, this command installs a GitHub repository – see these docs for more information

The `norns` table contains useful system commands and variables:

command	description
`norns.expand_filesystem()`	Expands the filesystem to the full capacity of the disk, useful after a fresh installation : function
`norns.blank()`	Turns off the norns screen : function
`norns.rerun()`	Rerun the currently-loaded script : function
`norns.shutdown()`	SLEEP norns (no UI state information will be shown, however) : function
`norns.system_cmd(command)`	Asynchronously executes a system `command` : function

The `norns.script` table contains functions for script lifecycle management:

command	description
`norns.script.load(filename)`	Runs the script at `filename` : function
`norns.script.clear()`	Clears the currently-running script : function

The `norns.state` table contains useful state data:

variable	description
`norns.state.path`	The `code` folder path for the currently-running script : string
`norns.state.data`	The `data` folder path for the currently-running script : string
`norns.state.lib`	The `lib` folder path for the currently-running script : string
`norns.state.shortname`	The name of the script, used as the default name by the PSET + PMAP filesystems : string
`norns.state.name`	The name of the script as shown in the SELECT menu and on the HOME screen : string

The `_path` table contains useful aliases for common file-paths:

variable	description

`_path.code`	Alias for `/home/we/dust/code/` : string
`_path.data`	Alias for `/home/we/dust/data/` : string
`_path.audio`	Alias for `/home/we/dust/audio/` : string
`_path.tape`	Alias for `/home/we/dust/audio/tape/` : string

libraries

Lua libraries can be used by using `include("path/to/library")`. Remember to *not* include `.lua` in the library name.

local libraries

`include()` will first look in the directory of the current script. This allows using relative paths to use libraries local to the script. For example, with the following structure:

```
myscript/
  myscript.lua
  lib/
    somelib.lua
```

`myscript.lua` can include `somelib.lua` using:

```
include("lib/somelib")
```

third-party libraries

Third party libraries can be included using their full path starting from the `~/dust/code/` directory. For example, with the following structure in `~/dust/code/`:

```
myscript/
  myscript.lua
  lib/
    somelib.lua
otherscript/
  otherscript.lua
  lib/
    otherlib.lua
```

`myscript.lua` can include `otherlib.lua` using:

```
include("otherscript/lib/otherlib")
```

engine

Specify an engine at the top of your script. See the engine docs for more details.

```
engine.name = 'PolySub'
```

If you want to use an engine from another project make sure to install that project first. If the engine comes with an accompanying Lua file make sure to import it:

```
engine.name = 'R'

local R = require 'r/lib/r'
```

- `engine.list_commands()` shows the commands.

For example to set the command `cutoff` to 500:

```
engine.cutoff(500)
```

To see a list of all locally installed engines:

```
tab.print(engine.names)
```

screen

The screen API handles drawing on the norns screen. See the screen docs for more details.

```
function redraw()
  screen.clear()
  screen.move(10,10)
  screen.text("hello world")
  screen.update()
end
```

metro

The metro API allows for high-resolution scheduling. See the metro docs for more details.

```
re = metro.init()
re.time = 1.0 / 15
re.event = function()
  redraw()
end
```

- `re:start()`, starts metro.
- `re:stop()`, stops metro.

paramset

The paramset API allows to read and write temporary data and files. See the paramset docs for more details.

A parameter can be installed with the following:

```
params:add{
  type = "number",
  id = "someparam",
  name = "Some Param",
  min = 1,
  max = 48,
  default = 4,
}
```

- `params:set(index, value)`, writes a parameter.
- `params:get(index)`, reads a parameter.

system globals

Do not overwrite these variables. Doing so may break things.

System globals:

```
_G, _VERSION, assert, bit32, collectgarbage, dofile , error, getmetatable, ipairs, io, load,
loadfile, next, math, os, pairs, pcall, print, rawequal, rawget, rawlen, rawset, require,
select, setmetatable, tonumber, tostring, table, type, utf8, xpcall
```

norns globals:

```
_menu, _norns, _path, _startup, arc, audio, cleanup, clock, controlspec, coroutine, crow,
debug, enc, engine, grid, hid, include, inf, key, metro, midi, mix, norns, osc, package,
params, paramset, paths, poll, redraw, screen, softcut, string, tab, util, wifi
```

devices

grid

`grid.connect(n)` to create device, returns object with handler. See the grid docs for more details.

```
g = grid.connect()
```

- `g:led(x, y, val)`, sets state of single LED on this grid device.
- `g:all(val)`, sets state of all LEDs on this grid device.
- `g:refresh()`, update any dirty quads on this grid device.
- `g.key(x, y, state)`, key event handler function.

arc

`arc.connect(n)` to create device, returns object with handler. See the arc docs for more details.

```
a = arc.connect()
```

- `a:led(ring, x, val)`, sets state of single LED on this arc device.
- `a:all(val)`, sets state of all LEDs on this arc device.
- `a:segment(ring, from, to, level)`, creates an anti-aliased point to point arc - segment/range on a specific LED ring.
- `a:refresh()`, updates any dirty quads on this arc device.
- `a.delta(n, delta)`, encoder event handler function.

midi

`midi.connect(n)` to create device, returns object with handler. See the midi docs for more details.

```
m = midi.connect()
```

- `m:note_on(value,velocity,channel)`, sends `note_on` message.
- `m:note_off(value,velocity,channel)`, sends `note_off` message.
- `m.event`, midi event handler function.

hid

`hid.connect(n)` to create device, returns object with handler. See the hid docs for more details.

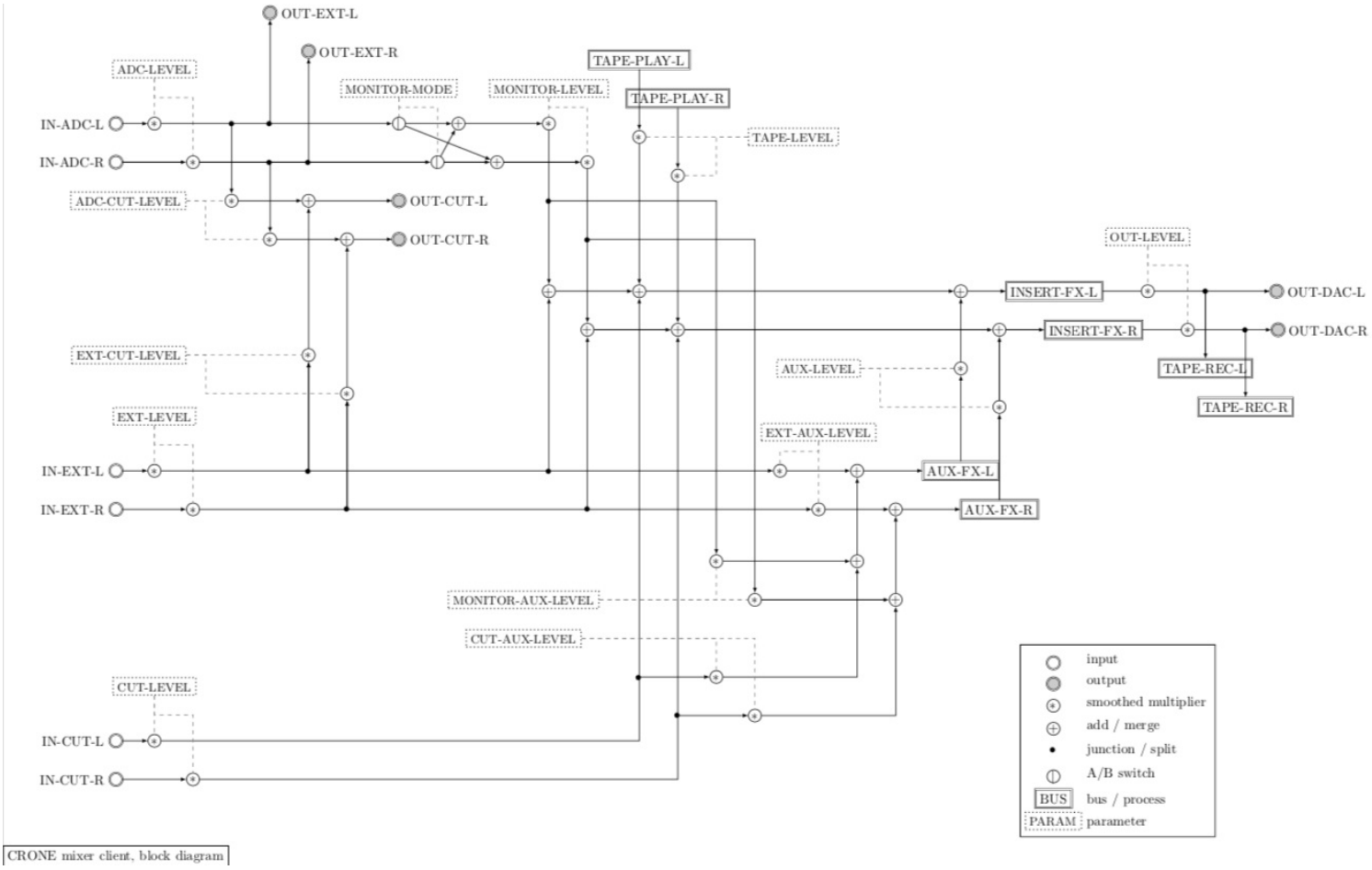
```
h = hid.connect()
```

osc

Send networked data via Open Sound Control. See the osc docs for more details.

- `osc.send(to, path, args)`, sends osc event.
- `osc.event(path, args, from)` handler function called when an osc event is received.

crone



CRONE mixer client, block diagram

pdf version

contribute

Contributions to the script reference are welcomed + we are very grateful for any assistance covering these topics.

Have questions about how to contribute? Please feel free to email help@monome.org and we can collaborate!

Many thanks to `@fardles` (site) for building the reference pages for `er`, `fileselect`, `musicutil`, `textentry`, and `ui`.

---

help