extending norns

sometimes desired functionality is impossible or inefficient to achieve in a norns script, but can become possible by modifying the source code of the norns system itself. norns (the sound computer) comes with everything you need to edit and recompile norns (the software suite), allowing you to add new functions to the core lua library and/or change how lua interacts with the hardware and the audio DSP. this tutorial will demonstrate how to extend norns by adding a new routine to the `screen` graphics library. if you make some additions or enhancements you'd like to see in the main norns code base, please send a pull request on github!

this tutorial assumes you've already figured out how to compile norns, and also assumes some familiarity with the C programming language.

where are we going with this?

the `screen` library on norns allows drawing a lot of interesting graphics using a shape-based set of drawing functions, but occasionally we'd like to get the values of pixels that have already been drawn to the screen. we're going to implement a `screen.peek(x, y, w, h)` lua function that allows determining the values of pixels in a certain area of the screen.

detour: norns programs

during normal operation, a few different programs run on norns all at once.

- *`maiden`*: a web server that serves a javascript app implementing an in-browser editor / IDE for writing and installing norns scripts. we won't be talking about maiden much in this tutorial, making changes to maiden works somewhat differently and it's pretty independent from everything else. the maiden browser app communicates with `matron` and `sclang` (the supercollider interpreter) via websockets. run `maiden help` in an SSH session to see available options.
- *`maiden-repl`* meanwhile is a C program (built with ncurses) that can make a websocket connection to `matron` or `sclang`. this allows you to access a REPL from an SSH session into norns. you can access this on norns by running `maiden repl` in an SSH session.
- *`matron`*: the "front of house" application you typically interact with when using norns. evaluates lua code, including user scripts and the REPL. draws to the screen and checks for key / encoder input. handles device detection and MIDI. sends control messages to the audio parts. this will be the focus of our discussion today.
- *`crone`*: the audio stuff manager. handles OSC messages for mixing, tape, and softcut. you may also wish to add other OSC commands here for e.g. adding functionality to softcut but this is a slightly different topic.
- *`ws-wrapper`*: `matron` and `sclang` just interact with stdin / stdout (reading from / writing to the terminal), they can't communicate with a remote application like `maiden` on their own. `ws-wrapper` is a utility program that creates a websocket server and passes websocket messages to and from the wrapped program.

`crone`, `maiden`, and `ws-wrapper`'d versions of `matron` and `sclang` are started on bootup by systemd services defined in `/etc/systemd/system`. together these make up the norns software stack.

to make sure we've compiled the current code, run `waf`:

```
we@norns.local:~/norns$ ./waf -j 1
Waf: Entering directory `/home/we/norns/build'
...
Waf: Leaving directory `/home/we/norns/build'
'build' finished successfully
```

at the moment, we haven't changed anything, so there's no need to restart anything. but after recompiling, for changes to take effect you'll also need to restart any modified programs or just restart norns entirely. you can restart maiden in a couple of different ways, including putting the `;restart` command in the maiden repl or running, for instance:

```
we@norns.local:~/norns$ systemctl restart norns-matron.service
```

or by SLEEPing or RESETing the norns system.

this is the build cycle we'll be using to test changes made to matron – run `./waf` to rebuild, then restart matron.

lua / C interface

all the main norns objects available to lua (`grid`, `screen`, `midi`, …) are implemented in a lua files in the `/home/we/norns/lua/core` folder. internally these libraries are able to interact with matron's C code using functions on the `_norns` table. while these functions are callable from lua, they are in fact implemented in C, and their implementations can be found in `norns/matron/src/weaver.c`. specifically the `w_init` function contains many lines like this:

```
    // register screen funcs
    lua_register_norns("screen_update", &_screen_update);
    lua_register_norns("screen_save", &_screen_save);
    lua_register_norns("screen_restore", &_screen_restore);
```

adding a new entry here will make another C function accessible from lua by adding a new value to the `_norns` table, e.g. `_norns.screen_update`. all such functions have the same C signature. we need to add a function declaration (toward the top of the file):

```
static int _screen_peek(lua_State *l);
```

we also add a function definition, down with the other `_screen_` function definitions. these need to get whatever arguments were passed from lua, do some work, and possibly give a result back. we also include specially formatted comments that provide documentation for the arguments these functions expect. for a function that returns a value, we need to tell lua we left 1 value on the stack by returning `1`. for more info, see this page about using the lua stack.

```
/***
 * screen: peek
 * @function s_peek
 * @tparam integer x screen x position (0-127)
 * @tparam integer y screen y position (0-63)
 * @tparam integer w rectangle width to grab
 * @tparam integer h rectangle height to grab
 */
int _screen_peek(lua_State *l) {
    // get the args passed in to _norns.screen_peek
    lua_check_num_args(4);
    int x = luaL_checkinteger(l, 1);
    int y = luaL_checkinteger(l, 2);
    int w = luaL_checkinteger(l, 3);
    int h = luaL_checkinteger(l, 4);
    // release the stack space used for the arguments
    lua_settop(l, 0);
    if ((x >= 0) && (x <= 127)
     && (y >= 0) && (y <= 63)
     && (w > 0)
     && (h > 0)) {
        uint8_t* buf = screen_peek(x, y, &w, &h);
        if (buf) {
            // return the results to lua by putting
            // a string value on the stack
            lua_pushlstring(l, buf, w * h);
            // lua_pushlstring copies the buffer to lua's memory,
            // so we can free it now
            free(buf);
            return 1;
        }
    }
    return 0;
}
```

tip: to print something out for debugging in matron, you generally do it like this:

```
fprintf(stderr, "some value: %d\n", 1234);
```

we'll also need to modify the `screen` lua library to expose the new function. in `norns/lua/core/screen.lua`. it's easier to hand in default values here and so on.

```
--- get a rectangle of screen content.
-- @tparam number x x position
-- @tparam number y y position
-- @tparam number w width, default 1
-- @tparam number h height, default 1
Screen.peek = function(x, y, w, h)
  return _norns.screen_peek(x, y, w or 1, h or 1)
end
```

that takes care of handing values back and forth between lua and C. then we need to implement the `screen_peek` function.

the screen buffer

first we need to declare the signature for our `screen_peek` function in the `screen.h` header file so it can be used from `weaver.c`. in `norns/matron/src/hardware/screen.h`:

```
extern char *screen_peek(int x, int y, int *w, int *h);
```

then in `norns/matron/src/hardware/screen.c` we'll write the definition. the details of this specific example function aren't necessary to understand, other C extensions could be much simpler or totally different from this.

```
uint8_t *screen_peek(int x, int y, int *w, int *h) {
    // this is a macro that bails out if the screen
    // hasn't been set up yet
    CHECK_CRR
    // need to ensure that the requested width and height
    // are in bounds
    *w = (*w <= (128 - x)) ? (*w) : (128 - x);
    *h = (*h <= (64 - y))  ? (*h) : (64 - y);
    // allocate a big enough destination buffer
    uint8_t *buf = malloc(*w * *h);
    if (!buf) {
        return NULL;
    }
    // finish any pending drawing
    cairo_surface_flush(surface);
    // get the raw pixel data for the screen
    uint32_t *data = (uint32_t *)cairo_image_surface_get_data(surface);
    if (!data) {
        return NULL;
    }

    // now convert ARGB32 pixels to 4-bit grayscale
    // each component of the ARGB32 values is the same
    char *p = buf;
    for (int j = y; j < y + *h; j++) {
        for (int i = x; i < x + *w; i++) {
            // get the i-th column in the j-th row of the screen buffer
            // only keep the lowest 4 bits to get a 0-15 brightness value
            // put this in the current cell in the output buffer
            *p = data[j * 128 + i] & 0xF;
            // advance the pointer into the output buffer
            p++;
        }
    }
    return buf;
}
```

we should now be able to recompile and restart `matron`, and test out the new norns function in a lua repl:

```
>> s = screen.peek(0, 10, 128, 1)
<ok>
>> string.byte(s, 1)
15
```

help