Hello, Emacsy!
A Minimal Emacsy Example Program

Shane Celis
shane.celis@gmail.com

1 Introduction

I have received a lot of questions asking, what does Emacsy[1] actually do? What restrictions does it impose on the GUI toolkit? How is it possible to not use any Emacs code? I thought it might be best if I were to provide a minimal example program, so that people can see code that illustrates Emacsy API usage.

2 Embedders' API

Here are a few function prototypes defined in `emacsy.h`.

⟨*emacsy.h* 1a⟩≡
```
/* Initialize Emacsy. */
int  emacsy_initialize(void);

/* Enqueue a keyboard event. */
void emacsy_key_event(int char_code,
                      int modifier_key_flags);

/* Run an iteration of Emacsy's event loop
   (will not block). */
int emacsy_tick();

/* Return the message or echo area. */
char *emacsy_message_or_echo_area();

/* Return the mode line. */
char *emacsy_mode_line();

/* Terminate Emacsy, runs termination hook. */
int  emacsy_terminate();
```

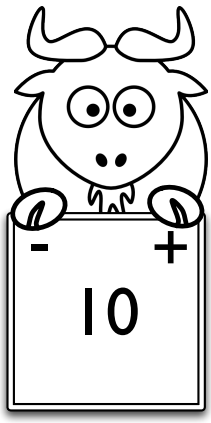

Figure 1: Emacsy integrated into the simplest application ever!

3 The Simplest Application Ever

Let's exercise these functions in a minimal GLUT program we'll call `hello-emacsy`.[2] This simple program will display an integer, the variable `counter`, that one can increment or decrement. The code will be organized as follows.

⟨*hello-emacsy.c* 1b⟩≡
```
⟨Headers 4a⟩
⟨State 1c⟩
⟨Functions 2b⟩
⟨Primitives 2e⟩
⟨Register primitives. 3b⟩
⟨Main 2a⟩
```

Our application's state is captured by one global variable.

⟨*State* 1c⟩≡ (1b)
```
int counter = 0; /* We display this number. */
```

[1] Kickstarter page http://kck.st/IY0Bau

[2] Note: Emacsy does not rely on GLUT. One could use Qt, Cocoa, or ncurses.

Let's initialize everything in `main` and enter our runloop.

⟨*Main* 2a⟩≡ (1b)
```
int main(int argc, char *argv[]) {
  int err;
  ⟨Initialize GLUT. 4d⟩
  scm_init_guile();    /* Initialize Guile. */
  scm_c_primitive_load("../../line-pragma.scm");
  /* Initialize Emacsy. */
  err = emacsy_initialize();
  if (err)
    exit(err);
  primitives_init();   /* Register primitives. */
  ⟨Load config. 3d⟩
  glutMainLoop();      /* We never return. */
  return 0;
}
```

4 Runloop Interaction

Let's look at how Emacsy interacts with your application's runloop since that's probably the most concerning part of embedding. First, let's pass some input to Emacsy.

⟨*Functions* 2b⟩≡ (1b) 2d▷
```
void keyboard_func(unsigned char glut_key,
                   int x, int y) {
  /* Send the key event to Emacsy
     (not processed yet). */
  int key;
  int mod_flags;
  ⟨Get modifier key flags. 4f⟩
  ⟨Handle control modifier. 2c⟩
  emacsy_key_event(key,
                   mod_flags);
  glutPostRedisplay();
}
```

The keys `C-a` and `C-b` returns 1 and 2 respectively. We want to map these to their actual character values.

⟨*Handle control modifier.* 2c⟩≡ (2b)
```
key = mod_flags & EY_MODKEY_CONTROL
  ? glut_key + ('a' - 1)
  : glut_key;
```

The function `display_func` is run for every frame that's drawn. It's effectively our runloop, even though the actual runloop is in GLUT.

⟨*Functions* 2b⟩+≡ (1b) ◁2b 4b▷
```
/* GLUT display function */
void display_func() {
  ⟨Setup display. 4c⟩
  ⟨Display the counter variable. 4e⟩

  /* Process events in Emacsy. */
  if (emacsy_tick() & EY_QUIT_APPLICATION) {
    emacsy_terminate();
    exit(0);
  }

  /* Display Emacsy message/echo area. */
  draw_string(0, 5, emacsy_message_or_echo_area());
  /* Display Emacsy mode line. */
  draw_string(0, 30, emacsy_mode_line());

  glutSwapBuffers();
}
```

At this point, our application can process key events, accept input on the minibuffer, and use nearly all of the facilities that Emacsy offers, but it can't change any application state, which makes it not very interesting yet.

5 Plugging Into Your App

Let's define a new primitive Scheme procedure `get-counter`, so Emacsy can access the application's state. This will define a `C` function `SCM scm_get_counter(void)` and a Scheme procedure `(get-counter)`.

⟨*Primitives* 2e⟩≡ (1b) 3a▷
```
SCM_DEFINE (scm_get_counter, "get-counter",
            /* required arg count    */ 0,
            /* optional arg count    */ 0,
            /* variable length args? */ 0,
            (),
            "Returns value of counter.")
{
  return scm_from_int(counter);
}
```

Let's define another primitive Scheme procedure to alter the application's state.

⟨*Primitives* 2e⟩+≡ (1b) ◁2e
```
SCM_DEFINE (scm_set_counter_x, "set-counter!",
          /* required, optional, var. length? */
             1, 0, 0,
             (SCM value),
             "Sets value of counter.")
{
  counter = scm_to_int(value);
  glutPostRedisplay();
  return SCM_UNSPECIFIED;
}
```

Once we have written these primitive procedures, we need to register them with the Scheme runtime.

⟨*Register primitives.* 3b⟩≡ (1b)
```
void primitives_init()
{
#ifndef SCM_MAGIC_SNARFER
  #include "hello-emacsy.c.x"
#endif
}
```

We generate the file `hello-emacsy.x` by running the command: `guile-snarf hello-emacsy.c`. Emacsy can now access and alter the application's internal state.

6 Changing the UI

Now let's use these new procedures to create interactive commands and bind them to keys by changing our config file `.hello-emacsy`.

⟨*.hello-emacsy* 3c⟩≡ 3e▷
```
(use-modules (emacsy emacsy))

(define-interactive (incr-counter)
 (set-counter! (1+ (get-counter))))

(define-interactive (decr-counter)
 (set-counter! (1- (get-counter))))

(define-key global-map
 (kbd "=") 'incr-counter)
(define-key global-map
 (kbd "-") 'decr-counter)
```

We load this file in `main` like so.

⟨*Load config.* 3d⟩≡ (2a)
```
scm_c_primitive_load(".hello-emacsy");
```

We can now hit `-` and `=` to decrement and increment the `counter`. This is fine, but what else can we do with it? We could make a macro that increments 5 times by hitting `C-x ( = = = = = C-x )`, then hit `C-e` to run that macro.

Let's implement another command that will ask the user for a number to set the counter to.

⟨*.hello-emacsy* 3c⟩+≡ ◁3c 3f▷
```
(define-interactive (change-counter)
 (set-counter!
   (string->number
     (read-from-minibuffer
       "New counter value: "))))
```

Now we can hit `M-x change-counter` and we'll be prompted for the new value we want. There we have it. We have made the simplest application ever more *Emacs-y*.

7 Changing it at Runtime

We can add commands easily by changing and reloading the file. But we can do better. Let's start a REPL we can connect to.

⟨*.hello-emacsy* 3c⟩+≡ ◁3e
```
(use-modules (system repl server))

;; Start a server on port 37146.
(spawn-server)
```

Now we can `telnet localhost 37146` to get a REPL.

8 Conclusion

We implemented a simple interactive application that displays a number. We embedded Emacsy into it: sending events to Emacsy and displaying the minibuffer. We implemented primitive procedures so Emacsy could access and manipulate the application's state. We extended the user interface to accept new commands `+` and `-` to change the state.

A Plaintext Please

Here are the plaintext files: emacsy.h, hello-emacsy.c, emacsy-stub.c, and .hello-emacsy. Or

B Uninteresting Code

Not particularly interesting bits of code but necessary to compile.

⟨*Headers* 4a⟩≡ (1b)
```
#ifndef SCM_MAGIC_SNARFER
#include <GLUT/glut.h>
#include <stdlib.h>
#include <emacsy.h>
#endif
#include <libguile.h>

void draw_string(int, int, char*);
```

⟨*Functions* 2b⟩+≡ (1b) ◁2d
```
/* Draws a string at (x, y) on the screen. */
void draw_string(int x, int y, char *string) {
  glLoadIdentity();
  glTranslatef(x, y, 0.);
  glScalef(0.2, 0.2, 1.0);
  while(*string)
    glutStrokeCharacter(GLUT_STROKE_ROMAN,
                        *string++);
}
```

Setup the display buffer the drawing.

⟨*Setup display.* 4c⟩≡ (2d)
```
glClear(GL_COLOR_BUFFER_BIT);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 500.0, 0.0, 500.0, -2.0, 500.0);
gluLookAt(0,   0,   2,
          0.0, 0.0, 0.0,
          0.0, 1.0, 0.0);

glMatrixMode(GL_MODELVIEW);
glColor3f(1, 1, 1);
```

⟨*Initialize GLUT.* 4d⟩≡ (2a)
```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE);
glutInitWindowSize(500, 500);
glutCreateWindow("Hello, Emacsy!");
glutDisplayFunc(display_func);
glutKeyboardFunc(keyboard_func);
```

Our application has just one job.

⟨*Display the counter variable.* 4e⟩≡ (2d)
```
char counter_string[255];
sprintf(counter_string, "%d", counter);
draw_string(250, 250, counter_string);
```

⟨*Get modifier key flags.* 4f⟩≡ (2b)
```
int glut_mod_flags = glutGetModifiers();
mod_flags = 0;
if (glut_mod_flags & GLUT_ACTIVE_SHIFT)
   mod_flags |= EY_MODKEY_SHIFT;
if (glut_mod_flags & GLUT_ACTIVE_CTRL)
   mod_flags |= EY_MODKEY_CONTROL;
if (glut_mod_flags & GLUT_ACTIVE_ALT)
   mod_flags |= EY_MODKEY_META;
```

C Noweb Index

C.1 Defined Fragments

⟨*.hello-emacsy* 3c⟩
⟨*Display the counter variable.* 4e⟩
⟨*emacsy.h* 1a⟩
⟨*Functions* 2b⟩
⟨*Get modifier key flags.* 4f⟩
⟨*Handle control modifier.* 2c⟩
⟨*Headers* 4a⟩
⟨*hello-emacsy.c* 1b⟩
⟨*Initialize GLUT.* 4d⟩
⟨*Load config.* 3d⟩
⟨*Main* 2a⟩
⟨*Primitives* 2e⟩
⟨*Register primitives.* 3b⟩

⟨*Setup display.* 4c⟩
⟨*State* 1c⟩