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

These are the proposed function prototypes defined in `emacsy.h`.

```
"emacsy.h" 1a≡
    /* Initialize Emacsy. */
    int  emacsy_init(void);

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

    /* Enqueue a mouse event. */
    void emacsy_mouse_event(int x, int y,
                            int button, int state);

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

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

    /* Return the mode line */
    char *emacsy_mode_line();◇
```

3 The Simplest Application Ever

Let's exercise these functions in a minimal GLUT program we'll call `hello-emacsy`. Note: Emacsy does not rely on GLUT; you could use Qt, Cocoa or ncurses. This program will display an integer, the variable `counter`.

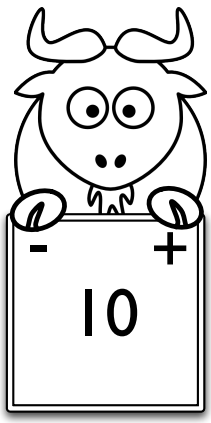

Figure 1: Emacsy integrated into the simplest application ever!

```
"hello-emacsy.c" 1b≡
    ⟨ Include Headers 3b ⟩
    int counter = 0; /* We display this number. */
    ⟨ Functions 2a, ... ⟩
    ⟨ Main 1c ⟩◇
```

Let's start with the main function.

```
⟨ Main 1c ⟩ ≡
    int main(int argc, char *argv[]) {
      ⟨ GLUT Initialization 3e ⟩
      scm_init_guile();  /* Initialize Guile. */
      emacsy_init();     /* Initialize Emacsy. */
      /* Load config. */
      //scm_c_primitive_load(".hello_emacsy");
      glutMainLoop();  /* We never return. */
      return 0;
    }◇
```

Fragment referenced in 1b.

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

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 2a ⟩ ≡
    void keyboard_func(unsigned char key,
                       int x, int y) {
      if (key == 'q')
        exit(0);
      /* Send the key event to Emacsy
         (not processed yet). */
      emacsy_key_event(glutGetModifiers(), key);
      glutPostRedisplay();
    }◇
```

Fragment defined by 2abcd, 3c.
Fragment referenced in 1b.

The function `display_func` is run for every frame that's drawn. It's effectively our runloop.

```
⟨ Functions 2b ⟩ ≡
    /* GLUT display function */
    void display_func() {
      ⟨ Display Setup 3d ⟩
      ⟨ Display the counter variable 3f ⟩

      /* Process events in Emacsy. */
      emacsy_tick();

      /* 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();
    }◇
```

Fragment defined by 2abcd, 3c.
Fragment referenced in 1b.

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 internal state.

```
⟨ Functions 2c ⟩ ≡
    SCM_DEFINE (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);
    }◇
```

Fragment defined by 2abcd, 3c.
Fragment referenced in 1b.

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

```
⟨ Functions 2d ⟩ ≡
    SCM_DEFINE (set_counter_x, "set-counter!",
                /* required arg count     */ 1,
                /* optional arg count     */ 0,
                /* variable length args? */ 0,
                (SCM value),
                "Sets value of counter.")
    {
      counter = scm_to_int(value);
      return SCM_UNSPECIFIED;
    }◇
```

Fragment defined by 2abcd, 3c.
Fragment referenced in 1b.

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" 2e≡
    (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)◇
```

File defined by 2e, 3a.

This is fine, but what else can we do with it? Let's implement another command that will ask the user for a number to set the counter to.

```
".hello-emacsy" 3a≡
    (define-interactive (change-counter)
     (set-counter!
      (read-from-minibuffer "New counter value: "↩
    )))◇
```

File defined by 2e, 3a.

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 emacs-y.

A Plaintext Please

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

B Uninteresting Code

Not particularly interesting bits of code but necessary to compile.

```
⟨ Include Headers 3b ⟩ ≡
    #include <GLUT/glut.h>
    #include <stdlib.h>
    #include <libguile.h>
    #include "emacsy.h"

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

Fragment referenced in 1b.

```
⟨ Functions 3c ⟩ ≡
    /* 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++);
    }◇
```

Fragment defined by 2abcd, 3c.
Fragment referenced in 1b.

Setup the display buffer the drawing.

```
⟨ Display Setup 3d ⟩ ≡
    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);◇
```

Fragment referenced in 2b.

```
⟨ GLUT Initialization 3e ⟩ ≡
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Minimal Emacsy Example");
    glutDisplayFunc(display_func);
    glutKeyboardFunc(keyboard_func);◇
```

Fragment referenced in 1c.

```
⟨ Display the counter variable 3f ⟩ ≡
    char counter_string[255];
    sprintf(counter_string, "%d", counter);
    draw_string(250, 250, counter_string);◇
```

Fragment referenced in 2b.

```
"emacsy-stub.c" 3g≡
    /* emacsy-stub.c */

    int  emacsy_init(void) { return 0; }

    void emacsy_key_event(int modifier_key_flags,
                          int key_code) {
      extern int counter;
      /* Fake the primitive scheme functions. */
      if (key_code == '=')
        counter++;
      else if (key_code == '-')
        counter--;
    }

    void emacsy_tick() { }

    char *emacsy_message_or_echo_area() {
      return "No commands defined.";
    }

    char *emacsy_mode_line() {
      return "-:%*-  Simplest Application Ever";
    }◇
```