Emacsy Reference Manual

An embeddable Emacs-like library using GNU Guile.

Shane Celis

Edition 0.3.21-4a1a
29 June 2019

Table of Contents

Emacsy

This document describes Emacsy version 0.3.21-4a1a, An embeddable Emacs-like library using GNU Guile.

Preface

This project is an experiment, actually two experiments. Firstly, it's an experiment to see whether there's any interest and utility in an embeddable Emacs-like environment. Secondly, I'd like to see how literate programming fares in comparison to the conventional approach. Let me elaborate a little on each.

Emacs is the extensible programmer's text editor. For decades, it's gobbled up functionality that sometimes seems far removed from text editing. I will expand upon why I believe this is the case and what particular functionality I hope to replicate later. I'd like to discuss a little about why I'm bothering to start with Emacs rather than just writing something entirely new. Emacs has fostered a community of people that are comfortable using, customising, and extending Emacs while its running. The last part is most important in my mind. Extending Emacs is a natural part of its use; it's a tinkerer's dream toy. And I want to grease the rails for people who already *get* what kind of tool I'm trying to provide. Had I chosen another perfectly competent language like Lua instead of a Lisp, that would erect a barrier to that track. Were I to write a completely different API, that's yet another barrier. Were I to "modernize" the terminology used by Emacs, e.g., say "key shortcut" instead of "key binding", or "window" instead of "frame", that's a barrier to drawing the community of people that already *get it* to try this out.

Let me say a little about why I'm choosing to do this as a literate program[1]. I've written a lot of code, none of which was written literately. Recently I had an experience that made me want to try something different. I began a group project. There wasn't *that* much code. Yet not too far into the project, it had become opaque to one of the original contributors. This was a small codebase with someone who was there from the start, and already we were having problems. Maybe the code was bad. Maybe we were bad programmers (Eek!). Whatever the case, assuming there's no simple fix for opaque code, it is something that can be addressed. Better communication about the code may help. So I would like to invest a good faith effort in attempting to write this program in a literate fashion.

A few notes on my personal goals for this document and the code. The writing style I'm leaving as informal for purposes of expediency and lowering the barrier of contribution. Also for expediency, my initial interest is in fleshing out the functionality. I'm not concerned about optimality of the implementation yet. Only in cases where the design cannot be reimplemented to be more efficient would I be concerned. If we can make a useable system, optimization will follow and hopefully be informed by profiling.

There's a ton of work left to do! Please feel free to contribute to the effort.

[1] Emacsy has since been converted from a literate noweb program to plain Guile Scheme and this Info document

1 Introduction

Emacsy is inspired by the Emacs text editor, but it is not an attempt to create another text editor. This project "extracts" the kernel of Emacs that makes it so extensible. There's a joke that Emacs is a great operating system—lacking only a decent editor. Emacsy is the Emacs OS sans the text editor. Although Emacsy shares no code with Emacs, it does share a vision. This project is aimed at Emacs users and software developers.

1.1 Vision

Emacs has been extended to do much more than text editing. It can get your email, run a chat client, do video editing[1], and more. For some the prospect of chatting from within one's text editor sounds weird. Why would anyone want to do that? Because Emacs gives them so much control. Frustrated by a particular piece of functionality? Disable it. Unhappy with some unintuitive key binding? Change it. Unimpressed by built-in functionality? Rewrite it. And you can do all that while Emacs is running. You don't have to exit and recompile.

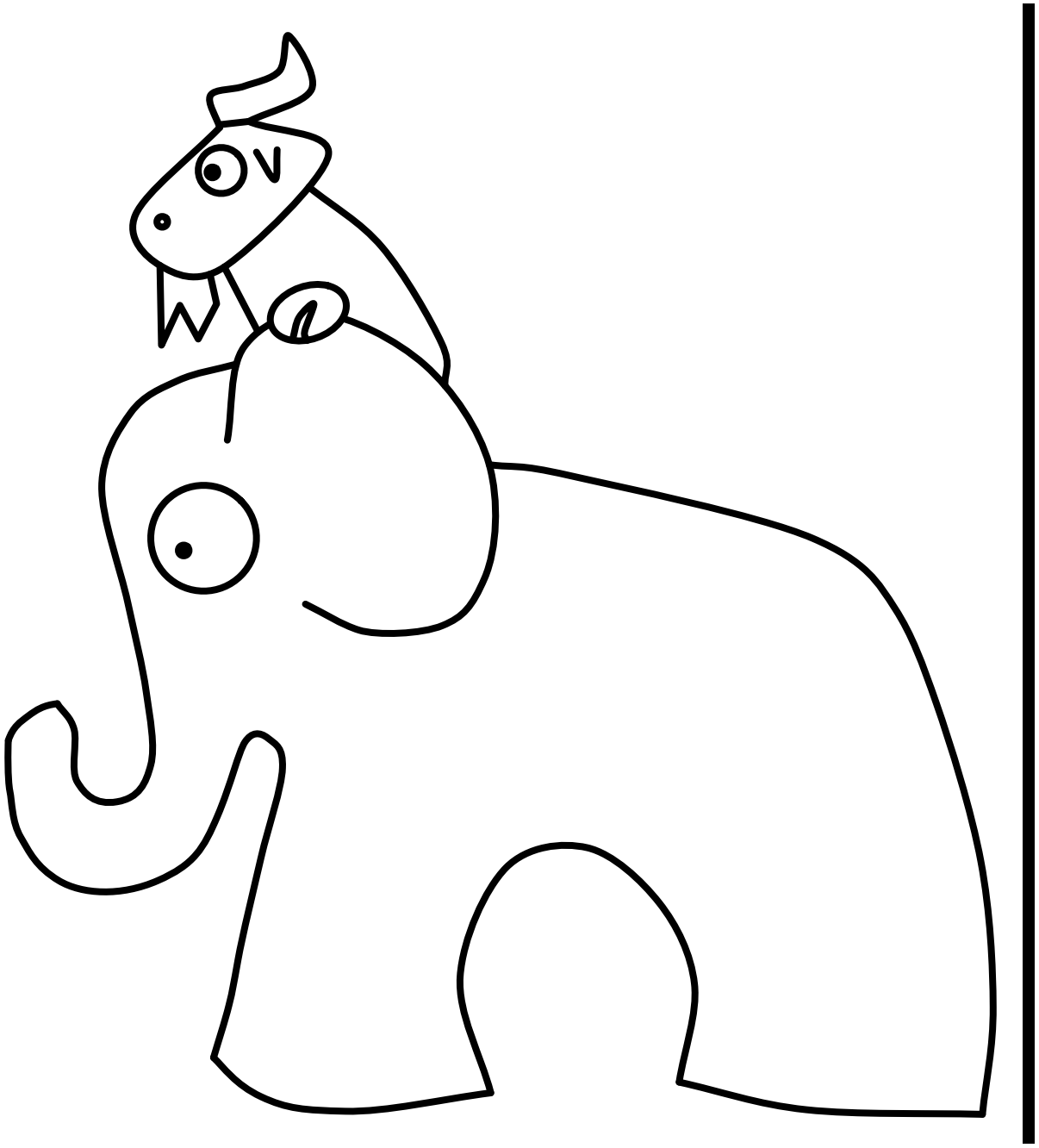
The purpose of Emacsy is to bring the Emacs way of doing things to other applications natively. In my mind, I imagine Emacs consuming applications from the outside, while Emacsy combines with applications from the inside—thereby allowing an application to be Emacs-like without requiring it to use Emacs as its frontend. I would like to hit \verb|M-x| in other applications to run commands. I would like to see authors introduce a new version: "Version 3.0, now extendable with Emacsy." I would like hear power users ask, "Yes, but is it Emacsy?"

1.1.1 Motivation

This project was inspired by my frustration creating interactive applications with the conventional edit-run-compile style of development. Finding the right abstraction for the User Interface (UI) that will compose well is not easy. Additionally, If the application is a means to an end and not an end in itself (which is common for academic and in-house tools), then the UI is usually the lowest development priority. Changing the UI is painful, so often

[1] `http://1010.co.uk/gneve.html`

mediocre UIs rule. Emacsy allows the developer—or the user—to reshape and extend the UI and application easily at runtime.



1.1.2 Overlooked Treasure

Emacs has a powerful means of programmatically extending itself while it is running. Not many successful applications can boast of that, but I believe a powerful idea within Emacs has been overlooked as an Emacsism rather than an idea of general utility. Let me mention another idea that might have become a Lispism but has since seen widespread adoption.

The Lisp programming language introduced the term Read-Eval-Print-Loop (REPL, pronounced rep-pel), an interactive programming feature present in many dynamic languages: Python, Ruby, MATLAB, Mathematica, Lua to name a few. The pseudo code is given below.

<<Lisp REPL>>= (while #t (print (eval (read)))) The REPL interaction pattern is to enter one complete expression, hit the return key, and the result of that expression will be displayed. It might look like this:

```
> (+ 1 2)
3
```

The kernel of Emacs is conceptually similar to the REPL, but the level of interaction is more fine grained. A REPL assumes a command line interface. Emacs assumes a keyboard interface. I have not seen the kernel of Emacs exhibited in any other applications, but I think it is of similar utility to the REPL—and entirely separate from text editing. I'd like to name this the Key-Lookup-Execute-Command-Loop (KLECL, pronounced clec-cull).

1.1.3 Emacsy

Long-time Emacs users will be familiar with this idea, but new Emacs users may not be. For instance, when a user hits the 'a' key, then an 'a' is inserted into their document. Let's pull apart the functions to see what that actually looks like with respect to the KLECL.

```
> (read-key)
#\a
> (lookup-key #\a)
self-insert-command
> (execute-command 'self-insert-command)
#t
```

Key sequences in Emacs are associated with commands. The fact that each command is implemented in Lisp is an implementation detail and not essential to the idea of a KLECL.

Note how flexible the KLECL is: One can build a REPL out of a KLECL, or a text editor, or a robot simulator (as shown in the video). Emacs uses the KLECL to create an extensible text editor. Emacsy uses the KLECL to make other applications similarly extensible.

1.1.4 Goals

The goals of this project are as follows.

1. Easy to embed technically

   Emacsy will use Guile Scheme to make it easy to embed within C and C++ programs.

2. Easy to learn

   Emacsy should be easy enough to learn that the uninitiated may easily make parametric changes, e.g., key 'a' now does what key 'b' does and *vice versa*. Programmers in any language ought to be able to make new commands for themselves. And old Emacs hands should be able to happily rely on old idioms and function names to change most anything.

3. Opinionated but not unpersuadable

   Emacsy should be configured with a sensible set of defaults (opinions). Out of the box, it is not *tabla rasa*, a blank slate, where the user must choose every detail, every time. However, if the user wants to choose every detail, they can.

4. Key bindings can be modified

   It wouldn't be Emacs-like if you couldn't tinker with it.

5. Commands can be defined in Emacsy's language or the host language

   New commands can be defined in Guile Scheme or C/C++.

6. Commands compose well

   That is to say, commands can call other commands. No special arrangements must be considered in the general case.

7. A small number of *interface* functions

   The core functions that must be called by the embedding application will be few and straightforward to use.

8. Bring KLECL to light

1.1.5 Anti-goals

Just as important as a project's goals are its anti-goals: the things it is not intended to do.

1. Not a general purpose text editor

   Emacsy will not do general purpose text editing out of the box, although it will have a minibuffer.

2. Not an Emacs replacement

   Emacs is full featured programmer's text editor with more bells and whistles than most people will ever have the time to fully explore. Emacsy extracts the Emacs spirit of application and UI extensibility to use within other programs.

3. Not an Elisp replacement

   There have been many attempts to replace Emacs and elisp with an newer Lisp dialect. Emacsy is not one of them.

4. Not source code compatible with Emacs

   Although Emacsy may adopt some of naming conventions of Emacs, it will not use elisp and will not attempt to be in any way source code compatible with Emacs.

5. Not a framework

   I will not steal your runloop. You call Emacsy when it suits your application not the other way around.

1.1.6 Emacsy Features

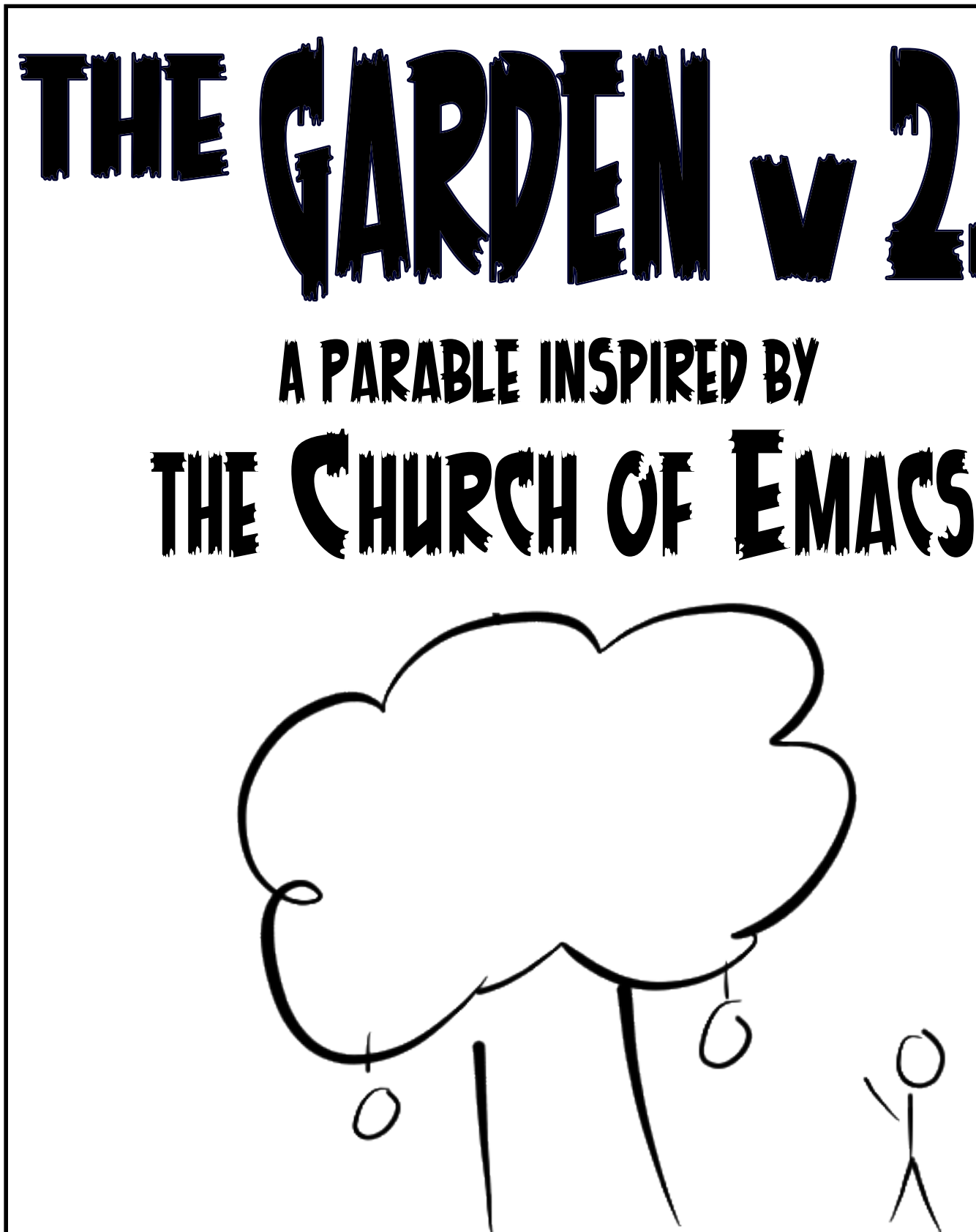
These are the core features from Emacs that will be implemented in Emacsy.

1. keymaps
2. minibuffer
3. recordable macros
4. history

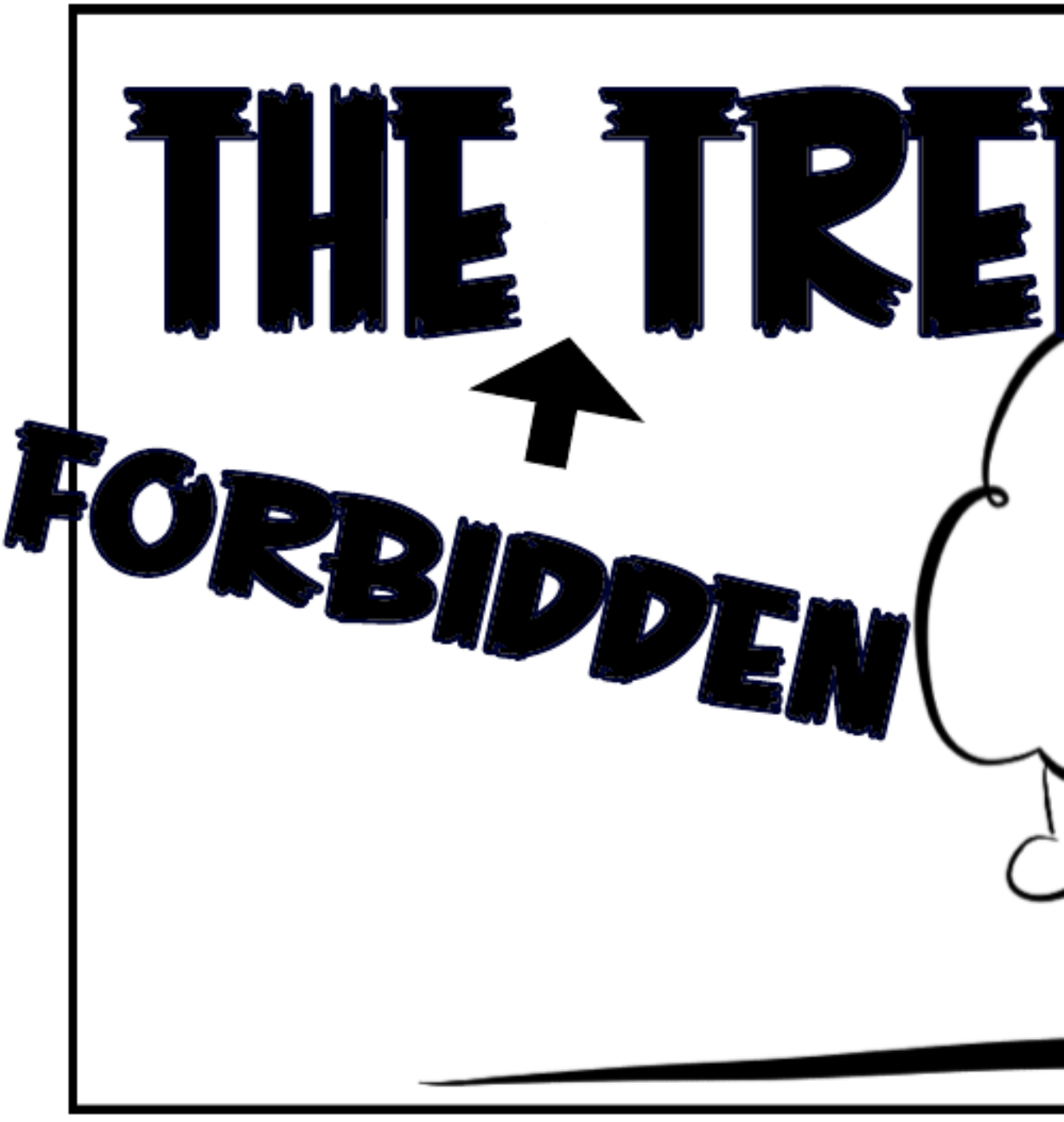
5. tab completion
6. major and minor modes

2 The Garden

Now for a little entertainment.




IN THE BEGINNING THE DEVELOPER CREATED THE GARDEN, A PROGRAM. AND IT WAS GOOD.
$ git commit -m "And


THE TRE
FORBIDDEN

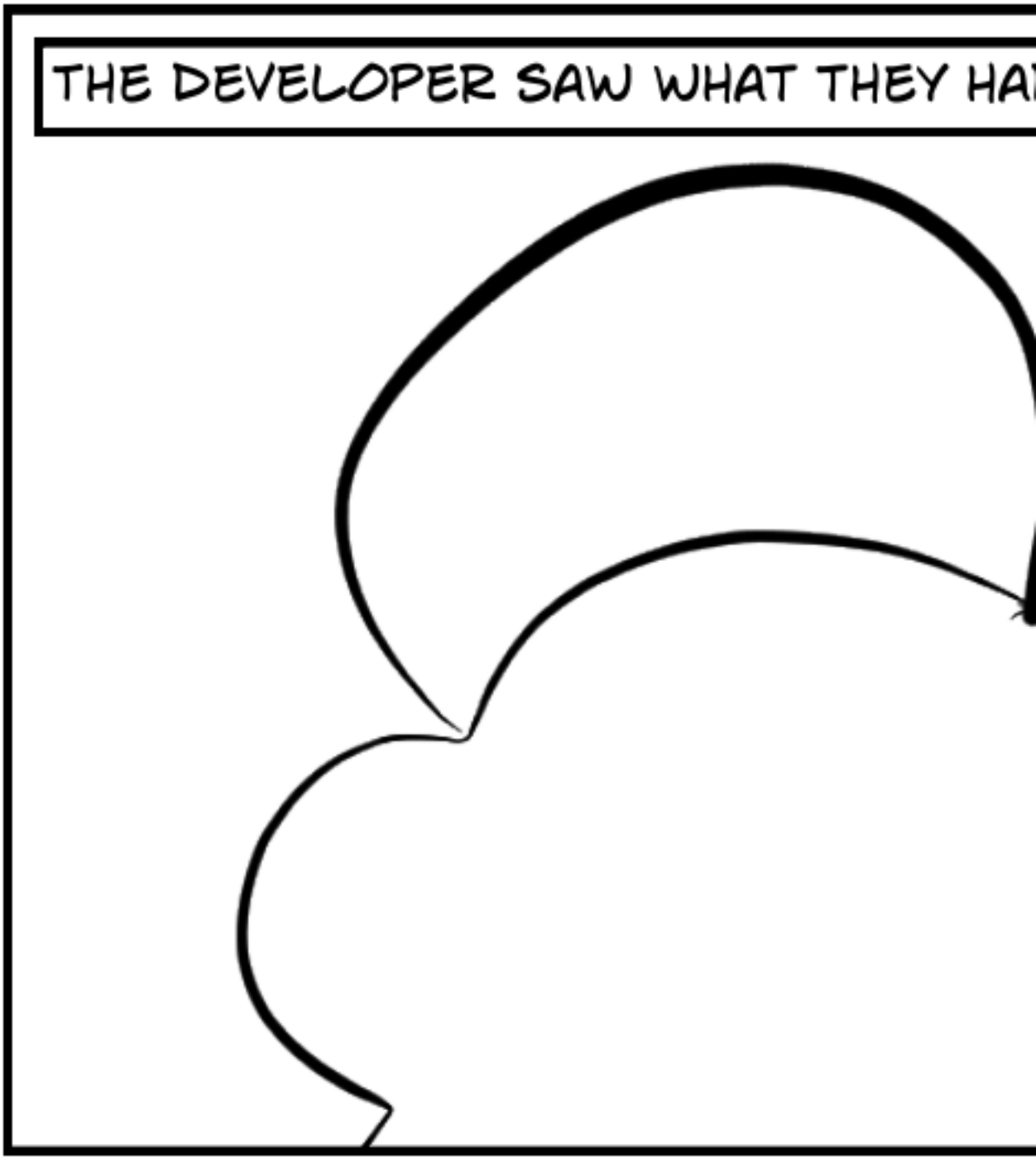

EMPOWERED, ALICE GREW DISCONT
This will not do.

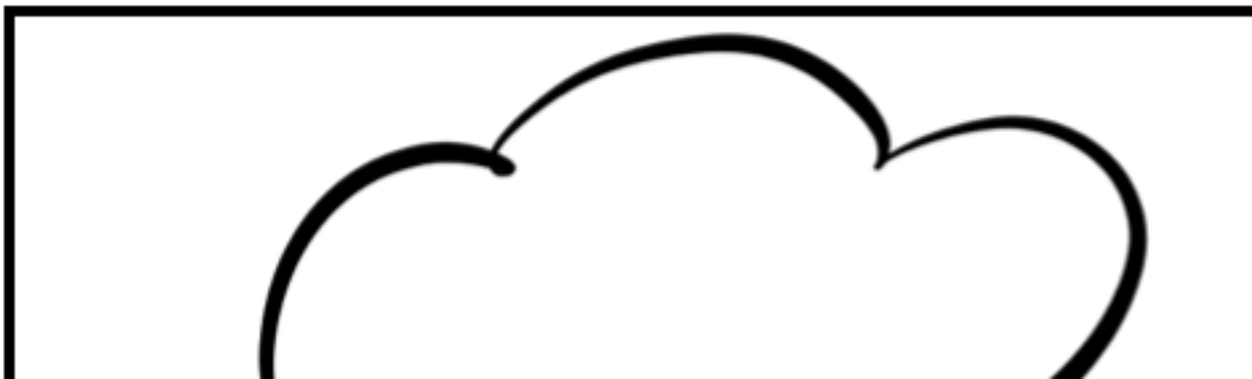

Delete tent.


$ make
LATER... ST
BUT, HOW?


YOUR FEATURE REQUEST IS ANSWERED, USER.
POOF


DISILLUSION ERODED STEVE'S RESOLVE.
ALICE!


THE DEVELOPER SAW WHAT THEY HA




THE
Parables reveal the tr

3 Installation

Emacsy is available for download from its website at `http://www.gnu.org/pub/gnu/emacsy/`. This section describes the software requirements of Emacsy, as well as how to install it and get ready to use it.

3.1 Requirements

This section lists requirements when building Emacsy from source. The build procedure for Emacsy is the same as for GNU software, and is not covered here. Please see the files `README` and `INSTALL` in the Emacsy source tree for additional details.

Emacsy depends on the following packages:

- GNU Guile (`http://gnu.org/software/guile/`), version 2.2.4 is known to work.
- Guile-Lib (`http://nongnu.org/guile-lib/`), version 0.2.1.6 is known to work.
- GNU Make (`http://www.gnu.org/software/make/`).

The following dependencies are optional:

- Autoconf (`http://www.gnu.org/software/autoconf`), Automake (`http://www.gnu.org/software/automake`), Libtool (`http://www.gnu.org/software/libtool`), and pkg-config (`https://www.freedesktop.org/wiki/Software/pkg-config`) to build from git.
- Installing Texinfo (`https://savannah.gnu.org/projects/texinfo`), will allow you to build the documentation.
- Installing FreeGLUT (`https://freeglut.sourceforge.net`), will allow you to build the Hello Emacsy example.
- Installing WebKitGTK (`https://webkitgtk.org`), will allow you to build the bare bones Emacsy Web browser examples.

3.2 Running the Test Suites

After a successful `configure` and `make` run, it is a good idea to run the test suites.

```
make check
```

4 Hello Emacsy

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.

4.1 Embedder's API: Unlimited power.

Here are a few function prototypes defined in `emacsy.h`, see Section 5.1 [C Api], page 34.

`int emacsy_initialize` (*int init_flags*) [C Function]
Initialize Emacsy.

`void emacsy_key_event` (*int char_code, int modifier_key_flags*) [C Function]
Enqueue a keyboard event.

`int emacsy_tick` () [C Function]
Run an iteration of Emacsy's event loop, does not block.

`char *emacsy_mode_line` () [C Function]
Return the mode line.

`int emacsy_terminate` () [C Function]
Terminate Emacsy; run termination hook.

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

4.2 The Simplest Application Ever

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

`int` *counter = 0;* [Variable]
Hello Emacsy's state is captured by one global variable. Hello Emacsy will display this number.

Initialize everything in *main* and enter our runloop.

`glutInit (&argc,` *argv*); [C Function]
Initialize GLUT.

`void scm_init_guile` (); [C Function]
Initialize Guile.

`emacsy_initialize (...);` [C Function]
Initialize Emacsy.

`primitives_init ();` [C Function]
Register primitives.

`char *` *try_load_startup* (. . .); [C Function]
Try to load `hello-emacsy.scm`

`void glutMainLoop` (); [C Function]
Enter GLUT main loop, not return.

[2] Note: Emacsy does not rely on FreeGLUT. One could use Gtk+, Ncurses, Qt, or whatever

4.2.1 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.

`void keyboard_func` (*unsigned char glut_key, int x, int y*) [C Function]
Send key events to Emacsy.

int key; // The Key event (not processed yet).

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

`void display_func ()` [C Function]
The function *display_func* is run for every frame that's drawn. It's effectively our runloop, even though the actual runloop is in FreeGLUT.

Our application has just one job: Display the counter variable.

`glClear (GL_COLOR_BUFFER_BIT);` [C Function]
Setup the display buffer the drawing.

Process events in Emacsy.

Display Emacsy message/echo area.

Display Emacsy mode line.

`void draw_string` (*int x, int y, char *string*) [C Function]
Draw a string function. Draws a string at (x, y) on the screen.

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.

4.2.2 Plugging Into Your App

`get-counter` [Scheme Procedure]
`SCM` *scm_get_counter* () [C Function]
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)`.

`set-counter!` *value* [Scheme Procedure]
`SCM` *scm_set_counter_x* (*SCM value*) [C Function]
Let's define another primitive Scheme procedure to alter the application's state.

`void primitives_init ()` [C Function]
Once we have written these primitive procedures, we need to register them with the Scheme runtime.

`char *` *try_load_startup* (*char const* prefix, char const* dir, char const* startup_script*) [C Function]
Locate the `hello-emacsy.scm` Guile initialization and load it.

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

`incr-counter` *#:optional* (*n* (*universal-argument-pop!*)) [Interactive Procedure]

`decr-counter` *#:optional* (*n* (*universal-argument-pop!*)) [Interactive Procedure]

`global-map` [Scheme Procedure]
Bind *inc-counter* to `=`.

`global-map` [Scheme Procedure]
Bind *inc-counter* to `-`.

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

`change-counter` [Interactive Procedure]
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*.

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. `example/hello-emacsy.scm`.

```
(use-modules (system repl server))
(spawn-server)
```

Start a server on port 37146.

4.3 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.

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

4.4 Plaintext Please

4.4.1 hello-emacsy.c

```
/*
   Emacsy --- An embeddable Emacs-like library using GNU Guile

   Copyright (C) 2012, 2013 Shane Celis <shane.celis@gmail.com>
   Copyright (C) 2019, Jan (janneke) Nieuwenhuizen <janneke@gnu.org>

   This file is part of Emacsy.

   Emacsy is free software:  you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
```

```
   (at your option) any later version.

   Emacsy is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with Emacsy.    If not, see <http://www.gnu.org/licenses/>.
 */

/*
 * Let's exercise these functions in a minimal FreeGLUT program we'll call
 * @verb{|hello-emacsy|}.@footnote{Note:  Emacsy does not rely on FreeGLUT.
 * One could use Gtk+, Ncurses, Qt, or whatever}.    This simple program
 * will display an integer, the variable @var{counter}, that one can
 * increment or decrement.
 *
 * @image{images/minimal-emacsy-example,,,,.png}
 */

#ifndef SCM_MAGIC_SNARFER
#include <libgen.h>
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#include <stdlib.h>
#include <emacsy.h>
#endif
#include <libguile.h>

void display_func ();
void keyboard_func (unsigned char glut_key, int x, int y);
void draw_string (int, int, char*);
char * try_load_startup (char const* prefix, char const* dir, char const* startup_script);
void primitives_init ();

/*
 * @defvar int counter = 0;
 * Hello Emacsy's state is captured by one global variable.
 * Hello Emacsy will display this number.
 * @end defvar
 */
int counter = 0;
int interactive = 1;
```

```
/*
 * Initialize everything in @var{main} and enter our runloop.
 */
int
main (int argc, char *argv[])
{
  int err;
/* glutInit (&argc, argv);
 * Initialize GLUT.
 */
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_RGB|GLUT_DOUBLE);
  glutInitWindowSize (500, 500);
  glutCreateWindow ("Hello, Emacsy!");
  glutDisplayFunc (display_func);
  if (interactive)
    glutKeyboardFunc (keyboard_func);
/* void scm_init_guile ();
 * Initialize Guile.
 */
  scm_init_guile ();
/* emacsy_initialize (@dots {});
 * Initialize Emacsy.
 */
  if (argc == 2 && strcmp ("--batch", argv[1]) == 0)
    interactive = 0;
  err = emacsy_initialize (interactive
                           ?  EMACSY_INTERACTIVE
                           :  EMACSY_NON_INTERACTIVE);
  if (err)
    exit (err);
/* primitives_init ();
 * Register primitives.
 */
  primitives_init ();

/* char * try_load_startup (@dots{});
 * Try to load @file{hello-emacsy.scm}
 */
  char const *startup_script = "hello-emacsy.scm";

  char prefix[PATH_MAX];
  strcpy (prefix, argv[0]);
  if (getenv ("_"))
    strcpy (prefix, getenv ("_"));
  dirname (dirname (prefix));
```

```
  if (!try_load_startup (0, 0, startup_script)
      &&!try_load_startup (getenv ("EMACSY_SYSCONFDIR"), "/", startup_script)
      &&!try_load_startup (prefix, "/", startup_script)
      &&!try_load_startup (prefix, "/etc/emacsy/", startup_script))
    fprintf (stderr, "error:  failed to find '%s'.\n", startup_script);
/* void glutMainLoop ();
 * Enter GLUT main loop, not return.
 */
  glutMainLoop ();
  return 0;
}

/*
 * @subsection 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.
 */

/* void keyboard_func (unsigned char glut_key, int x, int y)
 * Send key events to Emacsy.
 */
void
keyboard_func (unsigned char glut_key, int x, int y)
{
/*
 * int key; // The Key event (not processed yet).
 */
  int key;
  int mod_flags;
  int glut_mod_flags = glutGetModifiers ();
  mod_flags = 0;
  if (glut_mod_flags & GLUT_ACTIVE_SHIFT)
    mod_flags |= EMACSY_MODKEY_SHIFT;
  if (glut_mod_flags & GLUT_ACTIVE_CTRL)
    mod_flags |= EMACSY_MODKEY_CONTROL;
  if (glut_mod_flags & GLUT_ACTIVE_ALT)
    mod_flags |= EMACSY_MODKEY_META;
  if (glut_key == 8)
    glut_key = 127;
  else if (glut_key == 127)
    {
      glut_key = 4;
      mod_flags += EMACSY_MODKEY_CONTROL;
    }
```

```
/*
 * The keys @verb{|C-a|} and @verb{|C-b|} return @code{1} and @code{2}
 * respectively.    We want to map these to their actual character values.
 */
  key = mod_flags & EMACSY_MODKEY_CONTROL
    ?  glut_key + ('a' - 1)
    :  glut_key;
  emacsy_key_event (key, mod_flags);
  glutPostRedisplay ();
}

/* void display_func ()
 * The function @var{display_func} is run for every frame that's
 * drawn.  It's effectively our runloop, even though the actual runloop is
 * in FreeGLUT.
 *
 * Our application has just one job:  Display the counter variable.
 */
void
display_func ()
{
/* glClear (GL_COLOR_BUFFER_BIT);
 * Setup the display buffer the drawing.
 */
  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);

  char counter_string[255];
  sprintf (counter_string, "%d", counter);
  draw_string (250, 250, counter_string);

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

```
    }
  glutSetWindowTitle (emacsy_current_buffer ());

/*
 * 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 ();
}

/* void draw_string (int x, int y, char *string)
 *
 * Draw a string function.
 * 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++);
}

/*
 * 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.
 */

/*
 * @subsection Plugging Into Your App
 */

//

/*
 * @deffn {Scheme Procedure} get-counter
```

```
 * @deffnx {C Function} SCM scm_get_counter ()
 * Let's define a new primitive Scheme procedure @var{get-counter}, so
 * Emacsy can access the application's state.    This will define
 * a @var{C} function @code{SCM scm_get_counter (void)} and a Scheme procedure
 * @code{(get-counter)}.
 *
 * @end deffn
 */

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);
}

/*
 * @deffn {Scheme Procedure} set-counter!  value
 * @deffnx {C Function} SCM scm_set_counter_x (SCM value)
 * Let's define another primitive Scheme procedure to alter the
 * application's state.
 * @end deffn
 */

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;
}

/* void primitives_init ()
 * Once we have written these primitive procedures, we need to register
 * them with the Scheme runtime.
 */
void
primitives_init ()
{
#ifndef SCM_MAGIC_SNARFER
#include "hello-emacsy.c.x"
```

```
#endif
}

/* char * try_load_startup (char const* prefix, char const* dir, char const* startup_script
 * Locate the @file{hello-emacsy.scm} Guile initialization and load it.
 */
char *
try_load_startup (char const* prefix, char const* dir, char const* startup_script)
{
  static char file_name[PATH_MAX];
  if (prefix)
    strcpy (file_name, prefix);
  if (dir)
    strcat (file_name, dir);
  strcat (file_name, startup_script);

  if (access (file_name, R_OK) != -1)
    {
      fprintf (stderr, "Loading '%s'.\n", file_name);
      scm_c_primitive_load (file_name);
      return file_name;
    }
  else
    fprintf (stderr, "no such file '%s'.\n", file_name);

  return 0;
}
```

4.4.2 hello-emacsy.c.x

```
/* cpp arguments:  -pthread -I/gnu/store/9alic3caqhay3h8mx4iihpmyj6ymqpcx-guile-2.2.4/inclu
scm_c_define_gsubr (s_scm_get_counter, 0, 0, 0, (scm_t_subr) scm_get_counter);;
scm_c_define_gsubr (s_scm_set_counter_x, 1, 0, 0, (scm_t_subr) scm_set_counter_x);;
```

4.4.3 hello-emacsy.scm

```
;;; Emacsy --- An embeddable Emacs-like library using GNU Guile
;;;
;;; Copyright (C) 2012, 2013 Shane Celis <shane.celis@gmail.com>
;;;
;;; This file is part of Emacsy.
;;;
;;; Emacsy is free software:  you can redistribute it and/or modify
;;; it under the terms of the GNU General Public License as published by
;;; the Free Software Foundation, either version 3 of the License, or
;;; (at your option) any later version.
;;;
;;; Emacsy is distributed in the hope that it will be useful,
```

```
;;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;;; GNU General Public License for more details.
;;;
;;; You should have received a copy of the GNU General Public License
;;; along with Emacsy.    If not, see <http://www.gnu.org/licenses/>.

;; We generate the file @file{example/hello-emacsy.c.x} by running the
;; command:  @code{guile-snarf example/hello-emacsy.c}.    Emacsy can now
;; access and alter the application's internal state.
;;.

;; @subsection Changing the UI Now let's use these new procedures to
;; create interactive commands and bind them to keys by changing our
;; config file @file{example/hello-emacsy.scm}.
(use-modules (emacsy emacsy))

;;.
(define-interactive (incr-counter #:optional (n (universal-argument-pop!)))
 "Increment the counter."
 (set-counter!  (+ (get-counter) n)))

;;.
(define-interactive (decr-counter #:optional (n (universal-argument-pop!)))
 "Decrement the counter."
 (set-counter!  (- (get-counter) n)))

;; Bind @var{inc-counter} to @code{=}.
(define-key global-map "=" 'incr-counter)
;; Bind @var{inc-counter} to @code{-}.
(define-key global-map "-" 'decr-counter)

;; We can now hit @verb{|-|} and @verb{|=|} to decrement and increment the
;; @var{counter}.  This is fine, but what else can we do with it?    We could
;; make a macro that increments 5 times by hitting
;; @verb{|C-x ( = = = = = C-x )|}, then hit @verb{|C-e|} to run that macro.
;; (set!  debug-on-error?  #t)

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

;; Now we can hit @verb{|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 @emph{Emacs-y}.
(define-interactive (change-counter)
 "Change the counter to a new value."
```

```
 (set-counter!
   (string->number
     (read-from-minibuffer
       "New counter value:  ")))

;; @subsection 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.
;; @file{example/hello-emacsy.scm}.
;;.

;; @example
;; (use-modules (system repl server))
;; (spawn-server)
;; @end example
;; Start a server on port 37146.
;;.

;; Start a server on port 37146.
(use-modules (system repl server))
(spawn-server)
```

4.4.4 emacsy.h

```
/*
  Emacsy --- An embeddable Emacs-like library using GNU Guile

  Copyright (C) 2012, 2013 Shane Celis <shane.celis@gmail.com>
  Copyright (C) 2019, Jan (janneke) Nieuwenhuizen <janneke@gnu.org>

  This file is part of Emacsy.

  Emacsy is free software:  you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  Emacsy is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with Emacsy.    If not, see <http://www.gnu.org/licenses/>.
*/
```

```
#ifndef __EMACSY_H
#define __EMACSY_H 1

#ifdef __cplusplus
 extern "C" {
#endif

#include <libguile.h>

/* Here are the constants for the C API.                                    */
/*                                                                          */
/*                                                                          */
/* <emacsy-c-api:Defines>=                                                  */
#define EMACSY_MODKEY_COUNT   6

#define EMACSY_MODKEY_ALT     1 // A
#define EMACSY_MODKEY_CONTROL 2 // C
#define EMACSY_MODKEY_HYPER   4 // H
#define EMACSY_MODKEY_META    8 // M
#define EMACSY_MODKEY_SUPER  16 // s
#define EMACSY_MODKEY_SHIFT  32 // S

#define EMACSY_MOUSE_BUTTON_DOWN  0
#define EMACSY_MOUSE_BUTTON_UP    1
#define EMACSY_MOUSE_MOTION       2

#define EMACSY_INTERACTIVE        1
#define EMACSY_NON_INTERACTIVE    0

/* Here are the return flags that may be returned by \verb|emacsy_tick|.   */
/*                                                                          */
/*                                                                          */
/* <emacsy-c-api:Defines>=                                                  */
#define EMACSY_QUIT_APPLICATION_P       1
#define EMACSY_ECHO_AREA_UPDATED_P      2
#define EMACSY_MODELINE_UPDATED_P       4
#define EMACSY_RAN_UNDEFINED_COMMAND_P  8

/*
 * Emacsy provides a C API to ease integration with C and C++
 * programs.  The C API is given below.
 */

/* Initialize Emacsy.  */
int  emacsy_initialize (int init_flags);

/* Enqueue a keyboard event.  */
```

```
void emacsy_key_event (int char_code,
                       int modifier_key_flags);

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

/* Run an iteration of Emacsy's event loop, does 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 ();

/* Return the name of the current buffer.  */
char *emacsy_current_buffer ();

/* Run a hook.  */
int  emacsy_run_hook_0 (char const *hook_name);

/* Return the minibuffer point.  */
int  emacsy_minibuffer_point ();

/* Terminate Emacsy; run termination hook.  */
int  emacsy_terminate ();

/* Attempt to load a module.  */
int emacsy_load_module (char const *module_name);

/* Load a file in the emacsy environment.  */
//int emacsy_load(const char *file_name);

/* Convert the modifier_key_flags into a Scheme list of symbols.  */
SCM modifier_key_flags_to_list(int modifier_key_flags);

/* SCM scm_c_string_to_symbol (char const* str) */
SCM scm_c_string_to_symbol (char const* str);

/* Ref @var{name} from emacsy module.  */
SCM scm_c_emacsy_ref (char const* name);

#ifdef __cplusplus
 }
```

```
#endif

#endif // __EMACSY_H
```

5 Api

I expounded on the virtues of the Key Lookup Execute Command Loop (KLECL) in see Section 1.1.2 [Overlooked Treasure], page 4. Now we're going to implement a KLECL, which requires fleshing out some concepts. We need events, keymaps, and commands. Let's begin with events.

5.1 C Api

Emacsy provides a C API to ease integration with C and C++ programs. The C API is given below.

`int emacsy_initialize` (*int init_flags*) [C Function]
Initialize Emacsy.

`void emacsy_key_event` (*int char_code, int modifier_key_flags*) [C Function]
Enqueue a keyboard event.

`void emacsy_mouse_event` (*int x, int y, int state, int button, int modifier_key_flags*) [C Function]
Enqueue a mouse event.

`int emacsy_tick` () [C Function]
Run an iteration of Emacsy's event loop, does not block.

`char *emacsy_message_or_echo_area` () [C Function]

`char *emacsy_mode_line` () [C Function]
Return the mode line.

`char *emacsy_current_buffer` () [C Function]

`int emacsy_run_hook_0` (*char const *hook_name*) [C Function]
Run a hook.

`int emacsy_minibuffer_point` () [C Function]
Return the minibuffer point.

`int emacsy_terminate` () [C Function]
Terminate Emacsy; run termination hook.

`SCM load_module_try` (*void* data*) [C Function]
Attempt to load a module.

The function *scm_c_use_module* throws an exception if it cannot find the module, so we have to split that functionality into a body function *load_module_try* and an error handler *load_module_error*.

`SCM load_module_error` (*void *data, SCM key, SCM args*) [C Function]

`int emacsy_load_module` (*char const *module*) [C Function]
Attempt to load a module. Returns 0 if no errors, and non-zero otherwise.

`SCM modifier_key_flags_to_list` (*int modifier_key_flags*) [C Function]

`SCM scm_c_string_to_symbol` (*char const* str*) [C Function]

`modifier-key-flags->list` *flags* [Scheme Procedure]
`SCM` *scm_modifier_key_flags_to_list* (*flags*) [C Function]
Convert integer *flags* to a list of symbols.

`SCM scm_c_emacsy_ref` (*char const* name*) [C Function]
Ref *name* from emacsy module.

5.2 Emacsy Facade

So that users of our library don't have to import all of our nicely partitioned modules individually, we'll expose a facade module that re-exports all of the public interfaces for each module. Just use

```
(use-modules (emacsy emacsy))
```

or

```
#:use-module (emacsy emacsy)
```

5.3 Event

One of the idioms we want to capture from Emacs is this.

```
(define-key global-map "M-f" 'some-command)
```

They [[keymap]] and [[command]] module will deal with most of the above, except for the [[kbd]] procedure. That's something events will be concerned with. One may define a converter for a [[kbd-entry]] to an event of the proper type. Note that a [[kbd-string]] is broken into multiple [[kbd-entries]] on whitespace boundaries, e.g., "C-x C-f" is a [[kbd-string]] that when parsed becomes two [[kbd-entries]] "C-x" and "C-f".

`<event>` [Class]
Basic event class.

`<modifier-key-event>` [Class]
Event to capture key strokes, including the modifier keys.

`<key-event>` [Class]
Event to capture key strokes, including the modifier keys.

`<mouse-event>` [Class]
Event to capture mouse events.

`<drag-mouse-event>` [Class]
Event to capture mouse drag events.

`<dummy-event>` [Class]

`kbd-converter-functions` [Variable]
Now we have the function [[kbd-entry->key-event]]. [[kbd]] needs to know about this and any other converter function. So let's register it.

`kbd-entry->key-event` *kbd-entry* [Scheme Procedure]
Let's write the converter for the [[<key-event>]] class that will accept the same kind of strings that Emacs does. If the [[kbd-entry]] does not match the event-type, we return false [[#f]].

`modifier-char->symbol` *char* [Scheme Procedure]
For the modifier keys, we are going to emulate Emacs to a fault.

`register-kbd-converter` *function-name function* [Scheme Procedure]

`kbd->events` *kbd-string* [Scheme Procedure]

`canonize-event!` (*event* <*key-event*>) [Scheme Procedure]

`event->kbd` (*event* <*key-event*>) [Scheme Procedure]
Now we convert the [[<key-event>]] back to a [[kbd-entry]].

`event->kbd` (*event* <*modifier-key-event*>) [Scheme Procedure]

`modifier-symbol->char` *sym* [Scheme Procedure]
Instead of using [[define-generic]] I've written a convenience macro [[define-generic-public]] that exports the symbol to the current module. This mimics the functionality of [[define-public]]. In general, any *-public macro will export the symbol or syntax to the

`write` (*obj* <*key-event*>) *port* [Scheme Procedure]
Display the <*key-event*> in a nice way.

`kbd-entry->mouse-event` *kbd-entry* [Scheme Procedure]
The *kbd-entry* for mouse events is similar to key events. The regular expression is `^(([ACHMsS]-)*)((up-|down-|drag-)?mouse-([123]))\$`.

`up-mouse-event?` *e* [Scheme Procedure]

`down-mouse-event?` *e* [Scheme Procedure]

`drag-mouse-event?` *e* [Scheme Procedure]

`click-mouse-event?` *e* [Scheme Procedure]

`motion-mouse-event?` *e* [Scheme Procedure]

5.4 Keymap

The keymap stores the mapping between key strokes—or events—and commands. Emacs uses lists for its representation of keymaps. Emacsy instead uses a class that stores entries in a hash table. Another difference for Emacsy is that it does not convert `S-C-a` to a different representation like `[33554433]`; it leaves it as a string that is expected to be turned into a canonical representation "C-A".

Here is an example of the keymap representation in Emacs.

```
> (let ((k (make-sparse-keymap)))
    (define-key k "a"         'self-insert-command)
    (define-key k "<mouse-1>" 'mouse-drag-region)
```

```
    (define-key k "C-x C-f"   'find-file-at-point)
    k)

(keymap
 (24 keymap
     (6 . find-file-at-point))
 (mouse-1 . mouse-drag-region)
 (97 . self-insert-command))
```

When I initially implemented Emacsy, I replicated Emacs' keymap representation, but I realized it wasn't necessary. And it seems preferrable to make the representation more transparent to casual inspection. Also, Emacsy isn't directly responsible for the conversion of keyboard events into [[key-event]]s—that's a lower level detail that the embedding application must handle. Here is the same keymap as above but in Emacsy.

```
> (let ((k (make-keymap)))
    (define-key k "a"       'self-insert-command)
    (define-key k "mouse-1" 'mouse-drag-region)
    (define-key k "C-x C-f" 'find-file-at-point)
    k)

#<keymap
  a self-insert-command
  C-x #<keymap
        C-f find-file-at-point>
  mouse-1 mouse-drag-region>
```

There are a few differences in how the keymap is produced, and the representation looks slightly different too. For one thing it's not a list.

Our keymap class has a hashtable of entries and possibly a parent keymap.

`<keymap>` [Class]

`lookup-key` *keymap keys #:optional* (*follow-parent? #t*) [Scheme Procedure]

`lookup-key?` *keymap keyspec #:optional* (*keymap-ok? #f*) [Scheme Procedure]

`define-key` *keymap key-list-or-string symbol-or-procedure-or-keymap* [Scheme Procedure]

`keymap?` *obj* [Scheme Procedure]

`make-keymap` *#:optional* (*parent #f*) [Scheme Procedure]

`write` (*obj* `<`*keymap*`>`) *port* [Scheme Procedure]

`write-keymap` *obj port #:optional* (*keymap-print-prefix 0*) [Scheme Procedure]

`lookup-key-entry?` *result* [Scheme Procedure]

5.5 Command

> If words of command are not clear and distinct, if orders are not thoroughly understood, then the general is to blame.

—*Sun Tzu*

The command module is responsible for a couple things. In Emacs one defines commands by using the special form [[(interactive)]] within the body of the procedure. Consider this simple command.

```
(defun hello-command ()
  (interactive)
  (message "Hello, Emacs!"))
```

Emacsy uses a more Scheme-like means of defining commands as shown below.

```
(define-interactive (hello-command)
  (message "Hello, Emacsy!"))
```

One deviation from Emacs I want to see within Emacsy is to have the commands be more context sensitive. To illustrate the problem when I hit `M-x TAB TAB` it autocompletes all the available commands into a buffer. In my case that buffer contains 4,840 commands. This doesn't seem to hurt command usability, but it does hurt the command discoverability.

I want Emacsy to have command sets that are analogous to keymaps. There will be a global command set [[global-cmdset]] similar to the global keymap [[global-map]]. And in the same way that major and minor modes may add keymaps to a particular buffer, so too may they add command maps.

The class holds the entries, a string completer for tab completion, and potentially a parent command map.

`module-command-interface` *mod* [Scheme Procedure]

`module-export-command!` *m names* [Scheme Procedure]

`in-what-command` [Variable]

`this-command` [Variable]

`last-command` [Variable]

`kill-rogue-coroutine?` [Variable]

`seconds-to-wait-for-yield` [Variable]

`this-interactive-command` [Variable]

`command-contains?` (*cmap* <*command-set*>) *command-symbol* [Scheme Procedure]

We have accessors for adding, removing, and testing what's in the set. Note that the parent set is never mutated.

`command-add!` (*cmap* <*command-set*>) *command-symbol* [Scheme Procedure]

`command-remove!` (*cmap* <*command-set*>) *command-symbol* [Scheme Procedure]

`register-interactive` *name proc* [Scheme Procedure]

`command->proc` *command* [Scheme Procedure]

`command-name` *command* [Scheme Procedure]

`command?` *object* [Scheme Procedure]

`set-command-properties!` *proc #:optional (name #f)* [Scheme Procedure]

`what-command-am-i?` [Scheme Procedure]

`command-execute` *command . args* [Scheme Procedure]

`call-interactively` *command . args* [Scheme Procedure]

`called-interactively?` *#:optional (kind (quote any))* [Scheme Procedure]

5.6 Block

`<blocking-continuation>` [Class]
We're going to capture these blocking continuations into a class.

`blocking-continuations` [Variable]
[[call-blockable]] will handle any aborts to the [['block]] prompt. If the thunk aborts, it adds an instance of the class [[<blocking-continuation>]] to a list of such instances.

`block-yield` [Scheme Procedure]

`call-blockable` *thunk* [Scheme Procedure]

`block-tick` [Scheme Procedure]
To possibly resume these continuations, we're going to call [[block-tick]]. Additionally, continuations come in two flavors: serial and non-serial. The constraints on resuming are different. A non-serial block can be resumed whenever the [[continue-when?]] thunk return true. A serial block, however, will only be resumed after every other serial block that has a greater number, meaning more recent, has been resumed.

`blocking?` [Scheme Procedure]

`maybe-continue` (*obj* <*blocking-continuation*>) [Scheme Procedure]

`block-until` *condition-thunk #:optional (serial? #f)* [Scheme Procedure]
In addition to simply yielding we can block until a particular condition is met.

`block-while` *condition-thunk #:optional (serial? #f)* [Scheme Procedure]
And if we have [[block-until]], it's easy to write [[block-while]].

`block-kill` (*obj* <*blocking-continuation*>) [Scheme Procedure]
Sometimes we may just want to kill a blocking continuation. One could just forget the reference and let it be garbage collected. Here, we're going to throw an exception such that whatever the continuation was doing can potentially be cleaned up.

5.7 KLECL

> A box without hinges, key, or lid, yet golden treasure inside is hid.
> —*The Hobbit – J. R. R. Tolkien*

We finally have all the pieces to properly build the KLECL. First, we have to accept input.

`event-queue` [Variable]

`read-event-hook` [Variable]

`emacsy-interactive?` [Variable]

With the command loop I've also adopted a prefix of [[primitive-]] which signifies that it does not do any error handling. The command loop sets up a fair amount of state.

`this-command-event` [Variable]

`last-command-event` [Variable]

`pre-command-hook` [Variable]

`post-command-hook` [Variable]

`emacsy-ran-undefined-command?` [Variable]

`command-loop-count` [Variable]
Each command loop is given a different number.

`emacsy-event` *event* [Scheme Procedure]

`emacsy-key-event` *char #:optional* (*modifier-keys* (*quote*)) [Scheme Procedure]
This is a convenience procedure to enqueue a key event.

`emacsy-mouse-event` *position button state #:optional* (*modifier-keys* (*quote*)) [Scheme Procedure]

`emacsy-discard-input!` [Scheme Procedure]
And mainly for testing purposes we also want to discard all input. Or there are cases where we want to unread an event and push it to the front of the queue rather than the rear.

`emacsy-event-unread` *event* [Scheme Procedure]
[[read-event]] is the lowest-level procedure for grabbing events. It will block if there are no events to read.

`read-event` *#:optional* (*prompt #f*) [Scheme Procedure]

`read-key` *#:optional* (*prompt #f*) [Scheme Procedure]
read-key-sequence #:optional prompt #:key keymaps

`quit-key?` *aKey keymaps* [Scheme Procedure]
We also check all the maps for a quit key, typically defined as `C-g`.

`default-klecl-maps` [Scheme Procedure]

`message` . *args* [Scheme Procedure]
I find it convenient to begin emitting messages in case of error. However, I would like for there to be a clean separation between Emacsy and its KLECL such that someone may write a clean vim-y using it if they so chose. So this message will merely go to the stdout\; however, it will be redefined later.

primitive-command-tick #:optional prompt #:key keymaps undefined-command XXX Rename this to klec, for Key-Lookup-Execute-Command (KLEC)—just missing the loop component?

`command-tick` *#:key* (*keymaps* (*default-klecl-maps*)) [Scheme Procedure]

`primitive-command-loop` *#:optional* (*continue-pred* (*const #t*)) [Scheme Procedure]

Now let's write the command loop without any error handling. This seems a little messy with the continue predicate procedure being passed along. I'm not sure yet, how best to organize it.

`keyboard-quit` [Interactive Procedure]

We have finished the KLECL. Note that although we have used Emacs-like function names, we have not implemented the Emacs-like UI yet. We have not defined any default key bindings. I want to encourage people to explore different user interfaces based on the KLECL, and one can start from this part of the code. If one wanted to create a modal UI, one could use the [[(emacsy klecl)]] module and not have to worry about any "pollution" of Emacs-isms.

5.8 Kbd-Macro

. . .

—. . .

We will now add a keyboard macro facility familiar to Emacs users. We hook into the [[read-event]] procedure using a hook.

`defining-kbd-macro?` [Variable]

`last-kbd-macro` [Variable]

`executing-kbd-macro?` [Variable]

`kbd-macro-termination-hook` [Variable]

`executing-temporal-kbd-macro-hook` [Variable]

`kbd-read-event-hook` *event* [Scheme Procedure]

XXX This also may record the key event that stops the keyboard macro, which it shouldn't.

`kmacro-start-macro` [Interactive Procedure]

`kmacro-end-macro` [Interactive Procedure]

`kmacro-end-and-call-macro` [Interactive Procedure]

FIXME

`execute-temporal-kbd-macro` *#:optional* (*kbd-macro last-kbd-macro*) [Interactive Procedure]

In addition to regular keyboard macros, Emacsy can execute keyboard macros such that they reproduce the keys at the same pace as they were recorded.

5.9 Buffer

> And when you gaze long into an abyss the abyss also gazes into you.
> —*Beyond Good and Evil, Friedrich Nietzsche*

A buffer in Emacs represents text, including its mode, local variables, etc. A Emacsy buffer is not necessarily text. It can be extended to hold whatever the host application is interested in. Emacs' concepts of buffer, window, and mode are directly analogous to the model, view, and controller respectively—the MVC pattern.

`with-buffer ...` [Macro]
A convenience macro to work with a given buffer.

`save-excursion ...` [Macro]
A convenience macro to do some work

`<buffer>` [Class]

`before-buffer-change-hook` [Variable]

`after-buffer-change-hook` [Variable]

`buffer-stack` [Variable]

`last-buffer` [Variable]

`aux-buffer` [Variable]

`buffer-name` [Scheme Procedure]
Buffer's have a name, and there is always a current buffer or it's false. Note that methods do not work as easily with optional arguments. It seems best to define each method with a different number of arguments as shown below.

`buffer-name` (*buffer* `<`*buffer*`>`) [Scheme Procedure]

`set-buffer-name!` *name* [Scheme Procedure]

`set-buffer-name!` *name* (*buffer* `<`*buffer*`>`) [Scheme Procedure]

`buffer-modified?` [Scheme Procedure]

`buffer-modified-tick` [Scheme Procedure]

`write` (*obj* `<`*buffer*`>`) *port* [Scheme Procedure]

`current-local-map` [Scheme Procedure]

`use-local-map` *keymap* [Scheme Procedure]

`buffer-list` [Scheme Procedure]

`current-buffer` [Scheme Procedure]

`add-buffer!` *buffer* [Scheme Procedure]

`remove-buffer!` *buffer* [Scheme Procedure]

`next-buffer` *#:optional* (*incr 1*) [Interactive Procedure]

`prev-buffer` *#:optional* (*incr 1*) [Interactive Procedure]

`set-buffer!` *buffer* [Scheme Procedure]

This is scary, we will override it when we have <text-buffer>.

`other-buffer` *#:optional (count 1)* [Interactive Procedure]

`switch-to-buffer` [Variable]

`local-var-ref` *symbol* [Scheme Procedure]

`local-var-set!` *symbol value* [Scheme Procedure]

`local-var` [Variable]

5.9.1 Mru-stack

The buffers are kept in a most recently used stack that has the following operators: add!, remove!, contains?, recall!, and list.

`<mru-stack>` [Class]

`mru-add!` (*s* <*mru-stack*>) *x* [Scheme Procedure]

`mru-remove!` (*s* <*mru-stack*>) *x* [Scheme Procedure]

`mru-recall!` (*s* <*mru-stack*>) *x* [Scheme Procedure]

`mru-set!` (*s* <*mru-stack*>) *x* [Scheme Procedure]

`mru-ref` (*s* <*mru-stack*>) [Scheme Procedure]

`mru-list` (*s* <*mru-stack*>) [Scheme Procedure]

`mru-empty?` (*s* <*mru-stack*>) [Scheme Procedure]

`mru-contains?` (*s* <*mru-stack*>) *x* [Scheme Procedure]

`mru-next!` (*s* <*mru-stack*>) *count* [Scheme Procedure]

The order of the elements may not change yet the index may be moved around.

`mru-prev!` (*s* <*mru-stack*>) *count* [Scheme Procedure]

`mru-prev!` (*s* <*mru-stack*>) [Scheme Procedure]

`mru-next!` (*s* <*mru-stack*>) [Scheme Procedure]

5.10 Text

Editing and stuff.

`buffer-string` [Scheme Procedure]

`point` [Scheme Procedure]

`point-min` [Scheme Procedure]

`beginning-of-buffer` *#:optional arg* [Interactive Procedure]

`point-max` [Scheme Procedure]

`end-of-buffer` *#:optional arg* [Interactive Procedure]

`mark` *#:optional force* [Scheme Procedure]

`set-mark` *pos* [Scheme Procedure]

`set-mark-command` *#:optional arg* [Interactive Procedure]

`mark-whole-buffer` [Interactive Procedure]

`exchange-point-and-mark` [Interactive Procedure]

`char-after` *#:optional* (*point* (*point*)) [Scheme Procedure]

`goto-char` *#:optional* (*point* (*point*)) [Interactive Procedure]

`forward-char` *#:optional* (*n 1*) [Interactive Procedure]

`backward-char` *#:optional* (*n 1*) [Interactive Procedure]

`beginning-of-line` *#:optional* (*n 1*) [Interactive Procedure]

`end-of-line` *#:optional* (*n 1*) [Interactive Procedure]

`move-beginning-of-line` *#:optional* (*n 1*) [Interactive Procedure]

`move-end-of-line` *#:optional* (*n 1*) [Interactive Procedure]

`re-search-forward` *regex #:optional* (*bound #f*) (*no-error? #f*) (*repeat 1*) [Interactive Procedure]

`re-search-backward` *regex #:optional* (*bound #f*) (*no-error? #f*) (*repeat 1*) [Interactive Procedure]

`forward-word` *#:optional* (*n 1*) [Interactive Procedure]

`backward-word` *#:optional* (*n 1*) [Interactive Procedure]

`forward-line` *#:optional* (*n 1*) [Interactive Procedure]

`backward-line` *#:optional* (*n 1*) [Scheme Procedure]

`insert-char` *char* [Scheme Procedure]

`insert` *#:rest args* [Interactive Procedure]

`self-insert-command` *#:optional* (*n 1*) [Interactive Procedure]

`kill-ring` [Variable]

.

`delete-forward-char` *#:optional* (*n 1*) [Interactive Procedure]

`forward-delete-char` *#:optional* (*n 1*) [Interactive Procedure]

Alias for delete-forward-char

`delete-backward-char` *#:optional* (*n 1*) [Interactive Procedure]

`backward-delete-char` *#:optional* (*n 1*) [Interactive Procedure]

Alias for delete-backward-char

`delete-region` *#:optional* (*start* (*point*)) (*end* (*mark*)) [Interactive Procedure]

`kill-region` *#:optional* (*start* (*point*)) (*end* (*mark*)) [Interactive Procedure]

`delete-line` *n* [Scheme Procedure]

`kill-line` *#:optional* (*n 1*) [Interactive Procedure]

`delete-word` *n* [Scheme Procedure]

`kill-word` *#:optional* (*n 1*) [Interactive Procedure]

`backward-kill-word` *#:optional* (*n 1*) [Interactive Procedure]

A child of <buffer>, such as <text-buffer>, <minibuffer> or a custom UI buffer may override these, for efficiency or otherwise.

`buffer:line-length` (*buffer* <*buffer*>) [Scheme Procedure]

`buffer:current-column` (*buffer* <*buffer*>) [Scheme Procedure]

`buffer:beginning-of-line` (*buffer* <*buffer*>) *n* [Scheme Procedure]

`buffer:end-of-line` (*buffer* <*buffer*>) *n* [Scheme Procedure]

5.10.1 Editing for Gap Buffer

<*text-buffer*> inherits from buffer and implements the simplest text editing for the Gap Buffer.

`buffer:buffer-string` (*buffer* <*text-buffer*>) [Scheme Procedure]

`buffer:goto-char` (*buffer* <*text-buffer*>) *pos* [Scheme Procedure]

`buffer:point` (*buffer* <*text-buffer*>) [Scheme Procedure]

`buffer:point-min` (*buffer* <*text-buffer*>) [Scheme Procedure]

`buffer:point-max` (*buffer* <*text-buffer*>) [Scheme Procedure]

`buffer:set-mark` (*buffer* <*text-buffer*>) *pos* [Scheme Procedure]

`buffer:mark` (*buffer* <*text-buffer*>) [Scheme Procedure]

`buffer:char-before` (*buffer* <*text-buffer*>) *point* [Scheme Procedure]

`buffer:char-after` (*buffer* <*text-buffer*>) *pos* [Scheme Procedure]

`buffer:insert-string` (*buffer* <*text-buffer*>) *string* [Scheme Procedure]

`buffer:insert-char` (*buffer* <*text-buffer*>) *char* [Scheme Procedure]

`buffer:delete-char` (*buffer* <*text-buffer*>) *n* [Scheme Procedure]

`buffer:delete-region` (*buffer* <*text-buffer*>) *start end* [Scheme Procedure]

5.11 Minibuffer

The minibuffer provides a rich interactive textual input system. It offers `TAB` completion and history. The implementation of it inherits from the <*text-buffer*>.

`<minibuffer>` [Class]

`minibuffer-local-map` [Variable]

We define a keymap with all the typical self-insert-commands that would be expected in an editable buffer

`minibuffer` [Variable]

`emacsy-display-minibuffer?` [Variable]

`ticks-per-second` [Variable]

`minibuffer-history` [Variable]

When we show the minibuffer, we'll show the prompt, the contents (user editable), and the minibuffer-message if applicable.

`buffer:buffer-string` (*buffer* <*minibuffer*>) [Scheme Procedure]

`minibuffer-contents` *#:optional* (*buffer minibuffer*) [Scheme Procedure]

`delete-minibuffer-contents` *#:optional* (*buffer minibuffer*) [Scheme Procedure]

`minibuffer-message` *string . args* [Scheme Procedure]

5.11.1 read-from-minibuffer

history can be #f, a symbol, or a <cursor-list>.

`try-completion` *string collection #:optional* (*predicate* (*const #t*)) [Scheme Procedure]

`all-completions` *string collection #:optional* (*predicate* (*const #t*)) [Scheme Procedure]

`collection->completer` *collection #:optional* (*predicate* (*const #t*)) [Scheme Procedure]

`completing-read` *prompt collection #:key predicate* (*const #t*) (*require-match? #f*) (*initial-input #f*) (*history* (*what-command-am-i?*)) (*to-string #f*) [Scheme Procedure]

`apropos-module` *rgx module* [Scheme Procedure]

`command-completion-function` *text cont?* [Scheme Procedure]

We want to be able to look up filenames.

`files-in-dir` *dirname* [Scheme Procedure]

`read-file-name` *prompt #:key dir default-file-name initial predicate history* [Scheme Procedure]

5.11.2 Minibuffer History

`make-history` *#:optional* (*list* (*quote*)) (*index #f*) [Scheme Procedure]

`history-insert!` *history value* [Scheme Procedure]

`history-ref` *history* [Scheme Procedure]

`history-set!` *history value* [Scheme Procedure]

`exit-minibuffer` [Interactive Procedure]

`minibuffer-complete` [Interactive Procedure]

`next-match` [Interactive Procedure]

`previous-match` [Interactive Procedure]

`minibuffer-complete-word` [Interactive Procedure]

`minibuffer-completion-help` [Interactive Procedure]
Some commands for manipulating the minibuffer history.

`previous-history-element` *#:optional* (*n 1*) [Interactive Procedure]

5.12 Core

Now we're going to put in place some core functionality that makes Emacsy an Emacs-like library.

`global-map` [Variable]
We need a global keymap.

`special-event-map` [Variable]

`emacsy-quit-application?` [Variable]

`messages` [Variable]

`emacsy-send-mouse-movement-events?` [Variable]
Sometimes we may want to track the motion events generated by a mouse. We don't do this all the time because it seems unnecessarily taxing.

`current-active-maps` [Scheme Procedure]

`universal-argument-ref` [Scheme Procedure]

`universal-argument-pop!` [Scheme Procedure]

`universal-argument-push!` *arg* [Scheme Procedure]

`switch-to-buffer` *#:optional buffer* [Interactive Procedure]

`emacsy-echo-area` [Scheme Procedure]

`current-message` [Scheme Procedure]

`emacsy-message` . *args* [Scheme Procedure]

`clear-echo-area` [Scheme Procedure]
When the minibuffer is entered, we want to clear the echo-area. Because the echo-area is defined in core, it seems best to deal with it in core rather than placing echo-area handling code in minibuffer.

`emacsy-message-or-echo-area` [Scheme Procedure]
These are most of the C API calls.

`emacsy-mode-line` [Scheme Procedure]
method

`emacsy-mode-line` (*buffer* <*buffer*>) [Scheme Procedure]
method XXX this should be moved into the (emacsy buffer) module.

`emacsy-minibuffer-point` [Scheme Procedure]

`emacsy-run-hook` *hook . args* [Scheme Procedure]

`emacsy-terminate` [Scheme Procedure]

`emacsy-tick` [Scheme Procedure]

`emacsy-initialize` *interactive?* [Scheme Procedure]

`eval-expression` *#:optional epression* [Interactive Procedure]
There is one command that I consider fundamental for an Emacs-like program. Whenever I'm presented with a program that claims to be Emacs-like, I try this out `M-: (+ 1 2)`. If it doesn't work then it may have Emacs-like key bindings, but it's not Emacs-like. That command is [[eval-expression]]. Let's write it.

`execute-extended-command` *#:optional* (*n 1*) [Interactive Procedure]
The second fundamental command is [[execute-extended-command]] invoked with `M-x`.

`quit-application` [Interactive Procedure]

`universal-argument` [Interactive Procedure]
This [[universal-argument]] command is written using a different style than is typical for interative Emacs commands. Most Emacs commands are written with their state, keymaps, and ancillary procedures as public variables. This style has a benefit of allowing one to manipulate or extend some pieces; however, there are some benefits to having everything encapsulated in this command procedure. For instance, if the minibuffer were written in this style, one could invoke recursive minibuffers.

`load-file` *#:optional file-name* [Interactive Procedure]
We want to be able to load a scheme file.

The *scratch* buffer.

Override kill-buffer; make sure the buffer list does not become empty.

5.13 Advice

> Wise men don't need advice. Fools won't take it.
> —*Benjamin Franklin*
>
> No enemy is worse than bad advice.
> —*Sophocles*

Emacs has a facility to define "advice" these are pieces of code that run before, after, or around an already defined function. This article (`"http://electricimage.net/cupboard/2013/05/04/on-defadvice/"`) provides a good example.

`<record-of-advice>` [Record]
How will this work? Before we try to make the macro, let's focus on building up the functions. We want to have a function that we can substitute for the original function which will have a number of before, after, and around pieces of advice that can be attached to it.

`<piece-of-advice>` [Record]

5.14 Window

Emacsy aims to offer the minimal amount of intrusion to acquire big gains in program functionality. Windows is an optional module for Emacsy. If you want to offer windows that behave like Emacs windows, you can, but you aren't required to.

`<window>` [Class]
The window class contains a renderable window that is associated with a buffer.

`<internal-window>` [Class]
The internal window class contains other windows.

`root-window` [Variable]

`window-configuration-change-hook` [Variable]

`current-window` [Variable]

`initialize` (*obj* `<`*internal-window*`>`) *initargs* [Scheme Procedure]

`window?` *o* [Scheme Procedure]

`window-live?` *o* [Scheme Procedure]

`frame-root-window` [Scheme Procedure]

`edges->bcoords` *edges* [Scheme Procedure]
Emacs uses the edges of windows `(left top right bottom)`, but I'm more comfortable using bounded coordinate systems `(left bottom width height)`. So let's write some converters.

`bcoords->edges` *coords* [Scheme Procedure]

`window-clone` (*window* `<`*window*`>`) [Scheme Procedure]

`selected-window` [Scheme Procedure]

`update-window` (*window* `<`*internal-window*`>`) [Scheme Procedure]

`window-tree` (*w* `<`*internal-window*`>`) [Scheme Procedure]

`window-tree` (*w* `<`*window*`>`) [Scheme Procedure]

`window-list` *#:optional* (*w root-window*) [Scheme Procedure]

`split-window` *#:optional* (*window* (*selected-window*)) (*size 0.5*) (*side* (*quote below*)) [Interactive Procedure]
Be careful with `deep-clone`. If you deep clone one window that has references to other windows, you will clone entire object graph.

`split-window-below` *#:optional* (*size 0.5*) [Interactive Procedure]

`split-window-right` *#:optional* (*size 0.5*) [Interactive Procedure]

`delete-window` *#:optional* (*window* (*selected-window*)) [Interactive Procedure]

`other-window` *#:optional* (*count 1*) [Interactive Procedure]

5.15 Help

`describe-variable` *#:optional symbol* [Interactive Procedure]

`describe-command` *#:optional symbol* [Interactive Procedure]

5.16 Self-doc

Emacs offers a fantastic comprehensive help system. Emacsy intends to replicate most of this functionality. One distinction that would be nice to make is to partition Scheme values into procedures, variables, and parameters. In Scheme, all these kinds of values are the handled the same way. In Emacs, each are accessible by the help system distinctly. For instance, [[C-h f]] looks up functions, [[C-h v]] looks up variables. In addition to defining what kind of value a variable holds, this also allows one to include documentation for values which is not included in Guile Scheme by default. (XXX fact check.)

`variable-documentation` *variable-or-symbol* [Scheme Procedure]

XXX Rename from variable-documentation to just documentation.

`emacsy-collect-kind` *module kind #:optional* (*depth 0*) [Scheme Procedure]

We also want to be able to collect up all the variables in some given module.

Parameters behave similarly to variables; however, whenever they are defined, their values are set.

6 Contributing

6.1 Building from Git

If you want to hack Emacsy itself, it is recommended to use the latest version from the Git repository:

```
git clone git://git.savannah.gnu.org/emacsy.git
```

The easiest way to set up a development environment for Emacsy is, of course, by using Guix! The following command starts a new shell where all the dependencies and appropriate environment variables are set up to hack on Emacsy:

```
GUIX_PACKAGE_PATH=guix guix environment -l .guix.scm
```

Finally, you have to invoke `make check` to run tests (see Section 3.2 [Running the Test Suites], page 17). If anything fails, take a look at installation instructions (see Chapter 3 [Installation], page 17) or send a emacsysage to the `bug-emacsy@gnu.org` mailing list.

6.2 Running Emacsy From the Source Tree

First, you need to have an environment with all the dependencies available (see Section 6.1 [Building from Git], page 51), and then simply prefix each command by `./pre-inst-env` (the `pre-inst-env` script lives in the top build tree of Emacsy).

6.3 The Perfect Setup

The Perfect Setup to hack on Emacsy is basically the perfect setup used for Guile hacking (see Section "Using Guile in Emacs" in *Guile Reference Manual*). First, you need more than an editor, you need Emacs (`http://www.gnu.org/software/emacs`), empowered by the wonderful Geiser (`http://nongnu.org/geiser/`).

Geiser allows for interactive and incremental development from within Emacs: code compilation and evaluation from within buffers, access to on-line documentation (docstrings), context-sensitive completion, `M-.` to jump to an object definition, a REPL to try out your code, and more (see Section "Introduction" in *Geiser User Manual*).

6.4 Coding Style

In general our code follows the GNU Coding Standards (see *GNU Coding Standards*). However, they do not say much about Scheme, so here are some additional rules.

6.4.1 Programming Paradigm

Scheme code in Emacsy is written in a purely functional style.

6.4.2 Formatting Code

When writing Scheme code, we follow common wisdom among Scheme programmers. In general, we follow the Riastradh's Lisp Style Rules (`http://mumble.net/~campbell/scheme/style.txt`). This document happens to describe the conventions mostly used in Guile's code too. It is very thoughtful and well written, so please do read it.

If you do not use Emacs, please make sure to let your editor knows these rules.

Additionally, in Emacsy we prefer to format `if` statements like this

```
(if foo? trivial-then
    (let ((bar (the-longer ...)))
      more-complicated
      ...
      else))
```

6.5 Submitting Patches

Development is done using the Git distributed version control system. Thus, access to the repository is not strictly necessary. We welcome contributions in the form of patches as produced by `git format-patch` sent to the `guile-user@gnu.org` mailing list.

Please write commit logs in the ChangeLog format (see Section "Change Logs" in *GNU Coding Standards*); you can check the commit history for examples.

6.5.1 Reporting Bugs

Encountering a problem or bug can be very frustrating for you as a user or potential contributor. For us as Emacsy maintainers, the preferred bug report includes a beautiful and tested patch that we can integrate without any effort.

However, please don't let our preference stop you from reporting a bug. There's one thing *much* worse for us than getting a bug report without a patch: Reading a complaint or rant online about your frustrations and how our work sucks, without having heard directly what you experienced.

So if you report a problem, will it be fixed? And **when**? The most honest answer is: It depends. Let's curry that informationless honesty with a more helpful and more blunt reminder of a mantra of free software:

> **Q:** When will it be finished?
>
> **A:** It will be ready sooner if you help.
>
> —*Richard Stallman*

Join us on `#guile` on the Freenode IRC network or on `guile-user@gnu.org` to share your experience—good or bad.

Please send bug reports with full details to `guile-user@gnu.org`.

7 Acknowledgments

We would like to thank the following people for their help:

8 Resources

- Emacsy (`https://github.com/shanecelis/emacsy/`) GSOC
- #guile (`irc.freenode.net`) The Guile community home at the freenode IRC network.
- `guile-user@gnu.org` The Guile user mailing list, where it all started. guile-user archives (`https://lists.gnu.org/archive/html/guile-user/`).

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any,

be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their

titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

   Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Programming Index

(

*

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Keyboard command Index

S

U

Concept Index