CSC 195.01 2013S *Technologies for Mediascripting*

Class 13: Racket Internals (1): Overview

Held: Thursday, 2 May 2013

Summary: We do a quick introduction to writing extensions for Racket.

Related Pages:

- EBoard.

Notes:

- You're probably busy enough this week that there is *no homework*!
- For what we're doing today, it's helpful to have a reference page up.
- Today's convo is cool.
- Please attend next week's class for debriefing! **Overview:**
  - About Racket
  - It's Going to Seem Familiar
  - Extension Basics
  - Compilation
  - Loading

About Racket

- Racket is a variant of the Scheme programming language, designed for a variety of purposes. Racket includes strong support for teaching computer science.
- Racket has a nice IDE called DrRacket. We will be migrating 151 to using Racket and DrRacket.
  - Racket is the next generation of the Scheme implementation we use already.
  - DrRacket provided some inspiration for the MediaScheme IDE, but is also different in many ways.
- Our focus will primarily be extending Racket with a variety of features
  - Some DBus communcation
  - Some Gimp stuff
  - Some simplification
  - More?
- We may also look at embedding a Racket interpreter into the GIMP as an alternative to TinyScheme.
  - We have something like that implemented now.
- Some documentation that may help as you do this stuff is at http://docs.racket-lang.org/inside/.

It’s Going to Seem Familiar

- We’ll need to learn some standard template code.
- We’ll need to learn about a new compound type.
- We’ll need to learn the form of a function.
- We’ll need to learn the instructions for adding a function.
- We’ll need to learn copmilation commands.
- ...

Extension Basics

- We need to provide 3 basic functions that Racket uses to communicate with our code.
  - `scheme_initialize`, called when the code is first loaded.
  - `scheme_module_name`, called when it has to query our code.
  - `scheme_reload`, called when the code gets reloaded.
- All of these return `Scheme_Object` values. As you might guess, a `Scheme_Object` is Racket’s beautiful generic type.
- All of the extensions (new functions) you write will have the form

```
Scheme_Object *
function (int argc, Scheme_Object *argv[])
{
} // function
* You can check types with fun things like SCHEME_INTP
  SCHEME_DOUBLEP, and the ilk.
* You can extract values with SCHEME_INT_VAL,
  SCHEME_DOUBLE_VAL, and such.
* You install functions in two steps; First you make a function object
  and then you tell the environment about it.  (This code typically goes in
  the reload function.)
  + scheme_make_prim_w_arity (proc, internal-name, min-arity, max-arity)
  + scheme_add_global (extenal-name, result-of-prev-fun, env)
* Note that we have at least three names associated with the function.  We’ll
  play with changing each of these so that you can understand the
  difference.
```

Compilation

- [Side note: We’re supposed to deal with garbage collection, but we’ll avoid that issue for now. Our programs may leak.]
- Two steps: Compile to object file, link to shared object (rather than executable).
- Use raco ctool for both tasks, just with different flags.
  - --cc flag for compilation.
  - --ld flag for linking.

Loading

- [We're doing some simple instructions for loading code. When we get down to the nitty-gritty details, it's a bit more complex.]

- (load-extension "*prefix*.so")
