ScriptingLanguageSpecification

Matthew Bregg

December 4, 2015

Contents

1 Overview of Scripting Language

For the scripting language, we have choosen a lisp style language. While far from a complete lisp, it has a syntax similar to scheme, and could easily be extended, without breaking existing scripts.

1.1 Built in functions

1.1.1 define

```
(define name value)
(define fooConstant 3)
(define (sqr x) (* x x))
```

Assigns to the name the given value. This version does support defining functions atm. (See `http://www.scheme.com/tspl2d/binding.html`)

- Right now, defines may be nested.
- A variable name can't start with a number, to make it determinable from an int atom.
  - It must start with an alpha character.

1.1.2 Basic number ops

Basic number ops, including +,-,/,*,^. Takes in two args, returns a third with the value.

```
(+ 1 2)
(- 1 2)
(* 1 2)
(/ 1 2)
(^ 1 2)
```

1.1.3 concat-string

Takes in n string, or numbers, returns one string of all those concattenated

```
(concat-string "foo" 3 "bar")
(concat-string "Foo" "bar")
```

1.1.4 cons

Takes two arguments, returns a pair holding the two arguments as one object

```
(cons a b)
(cons 1 2)
```

(See `http://download.plt-scheme.org/doc/4.2.4/html/guide/Pairs__Lists__and_Scheme_Syntax.html`)

1.1.5 card

Takes a card-size, name, and two layouts, one for the front, one for the back, and then the shape of the card.

```
(card card-size "name" frontLayout backLayout shape)
```

When a card is rendered, it will pair the layout with the wholeCard position-scaled (0 0 100 100)

1.1.6 render

Takes in a single card, a list of cards, or a pair of cards, and 0-n families. Renders them.

```
(render card family)
(render cards family)
(render (cons carda cardb) family)
(render some-cards family0 ... familyn-1)
```

Note: The family is an optional argument, leaving it empty is the same as calling

```
(render cards (family))
```

Which runs render with an empty family.

1.1.7 list

Takes in a n arguments, and returns a list of them.

```
(list N0 ... Nn-1)
(list 1 2 3 4 5 6 7)
```

1.1.8 position-scaled

Takes in a x-offset%, y-offset%, and a scale-width% and scale-height%, and returns a position-scaled object.

- The two scale arguments are optional, default to 100.

```
(position-scaled x-offset% y-offset% scale-width% scale-height%)
(position-scaled 0 0 50 50)
(define wholeCard (position-scaled 0 0 100 100))
```

1.1.9 leaf-layout

Returns a layout. Takes in a decal, or a string. In the event a string is given, the decal will be looked up in the family. This layout can then be used with the above layout function.

```
(layout image)
(layout foobarImage)
(layout "foo")
(layout (color-decal "white"))
```

1. Leaf-Layout options A Leaf-Layout can be given a third argument, to determine some extra behavior. Takes an extra parameter, either a W, or an H, A, or O.

    - If W, width will be at most maximimum width of an image.
    - If H, height will be at most, maximum height of given image
    - If A, the original aspect ratio will be maintained.
    - If O, original size will be mantained.
    - IF S, stretch to fit.
    - All the options aside S, which does need to, will add transparent padding to return a size render desires.
    - The default is "S", so calling with the "S" argument is the same as not having a third argument

1.1.10 layout

Creates a Layout object. A layout contains 0-n tuples of layouts position-scaleds, and shapes. Takes 0-n tuples of layouts position-scaleds and shapes as arguments.

```
(Layout
  (list layout0 position-scaled0 shape0)
  (list layout1 position-scaled1 shape1)
  ...
  (list layoutn-1 position-scaledn-1 shapen-1))

(Layout
  (list layoutFoo position-scaledFoo rectanglebar)
  (list (layout foobarImage) wholeCard rectanglefoo)
  (list (layout "foo") (circle 3.14))
)
```

1.1.11 family

Creates a map of strings to decals, a family. Takes in a name, and N pairs.

- Requires a family name.
  - The family name is added to the card name when a card is rendered, to avoid name collisions when rendering the same card with multiple families.
  - If a multiple families given, append the names of all the families.

```
(family name pair0 ... pairn-1)
(family "fooFamily" (cons "foo" fooImage) (cons "bar" barImage))
```

1.1.12 eval-file

Takes in n filepaths, evals each file in given order

```
(eval-file "filename.filename")
(eval-file "foo.script")
(eval-file "foo.script" "bar.script")
```

Evals foo.script. Returns null.

1.1.13 Decals

- Image Decal

```
(image "filepath.[jpg|png|etc]")
(image "foo.jpg")
```

- Color Decal
  - A decal takes in a color

    ```
    (color-decal "color")
    (color-decal "white")
    ```

- String decal

A string from a given font.

```
(string "StringText" "Font" "Color" Size)
(string "Hello World!" "Arial.font" "Red" 12)
(string "1" "Arial.font" "Red" 12)
```

1.1.14 Shapes

- Rectangle

```
(rectangle width height)
(rectangle 100 200)
```

- Triangle

```
(triangle lengthA lengthB lengthC)
(triangle  100 200 300)
```

- AnyShape
  - Connect point0 -> point1, and then pointn-1 -> point0 to make a shape

```
(any-shape point0x point0y point1x point1y ... pointn-1x pointn-1y)
(any-shape  100 100 200 200 300 300)
```

- Circle

```
(circle radius)
(Circle 100)
```

1.1.15 Position-Scaleds

A position-scaled that can be used in the script

```
(position-scaled x-offset% y-offset% scale-width% scale-height%)
(position-scaled 0 0 100 100)
```

1.1.16 Size

A size is used by a card to determine how many pixels it will be.

```
(size width height)
```

2 Config file

- Allows one to set various options
- Current options are
  - script-file
    - Specify the script to run
    - No default, can be overridden by terminal args
  - output-format
    - Specify what format to output in
    - Defaults to png
  - output-file
    - Specify where to output the result to
    - Defaults to ./
  - logfile
    - Specify where to log to
    - Defaults to .cardlog
  - load-builder
    - Takes in a name, and the path to a builder java file.
    - Loads said builder into script evaluator
      `(set-option "load-builder" "name" "path")`
- Each option is enter in this format

```
(set-option "option-name" values)
```

-So for example

```
(set-option "output-dir" "./")
```