Binding With Metatable And Closures

Description

Here is an example of how to make a Lua binding to Thomas Boutell's GD Graphics Library [1] using closures containing a metatable. The original example of how to use GD in C can be found on Tom's web page [2].

`Image.new` is a constructor that returns a userdata containing the image to be maninpulated. The userdata's metatable has a garbage collection event to destroy the image, and a *index* event for calling the methods needed to draw the image and save the image to a file. Only three methods are implimented: `ColorAllocate`, `Line` and `PNG`.

C code

```
#include <stdio.h>

#include "lua.h"
#include "lauxlib.h"
#include "lualib.h"
#include "gd.h"

/*
** ============================================================================
**  Example Lua bindings for GD
** ============================================================================
*/

#define IMAGE "Image"

static gdImagePtr checkimage (lua_State *L, int index)
{
  luaL_checktype(L, index, LUA_TUSERDATA);
  lua_getmetatable(L, index);
  if( ! lua_equal(L, lua_upvalueindex(1), -1) )
    luaL_typerror(L, index, IMAGE);  /* die */
  lua_pop(L, 1);
  return (gdImagePtr)lua_unboxpointer(L, index);
}
```

```
static gdImagePtr pushimage (lua_State *L, gdImagePtr im)
{
  lua_boxpointer(L, im);
  lua_pushvalue(L, lua_upvalueindex(1));
  lua_setmetatable(L, -2);
  return im;
}


static int image_new (lua_State *L)
{
  int x = luaL_checkint(L, 1);
  int y = luaL_checkint(L, 2);
  pushimage(L, gdImageCreate(x, y));
  return 1;
}

static int image_destroy (lua_State *L)
{
  gdImagePtr im = (gdImagePtr)lua_unboxpointer(L, 1);
  if (im) gdImageDestroy(im);
  return 0;
}

static int image_color_allocate (lua_State *L)
{
  gdImagePtr im = checkimage(L, 1);
  int r = luaL_checkint(L, 2);
  int g = luaL_checkint(L, 3);
  int b = luaL_checkint(L, 4);
  lua_pushnumber(L, gdImageColorAllocate(im, r, g, b));
  return 1;
}

static int image_line (lua_State *L)
{
  gdImagePtr im = checkimage(L, 1);
  int x1     = luaL_checkint(L, 2);
  int y1     = luaL_checkint(L, 3);
  int x2     = luaL_checkint(L, 4);
  int y2     = luaL_checkint(L, 5);
  int colour = luaL_checkint(L, 6);
  gdImageLine(im, x1, y1, x2, y2, colour);
  return 0;
```

```
}

static int image_png (lua_State *L)
{
  /* Output the image to the disk file in PNG format. */
  gdImagePtr im    = checkimage(L, 1);
  const char *name = luaL_checkstring(L, 2);
  FILE *pngout     = fopen( name, "wb");
  gdImagePng(im, pngout);
  fclose(pngout);
  return 0;
}

static const luaL_reg meta_methods[] = {
{"__gc", image_destroy },
{0,0}
};

static const luaL_reg image_methods[] = {
{"new",           image_new},
{"colorallocate", image_color_allocate},
{"line",          image_line},
{"PNG",           image_png},
{0,0}
};

#define newtable(L) (lua_newtable(L), lua_gettop(L))

int Image_register (lua_State *L)
{
  int metatable, methods;

  lua_pushliteral(L, IMAGE);         /* name of Image table */
  methods   = newtable(L);           /* Image methods table */
  metatable = newtable(L);           /* Image metatable */
  lua_pushliteral(L, "__index");     /* add index event to metatable */
  lua_pushvalue(L, methods);
  lua_settable(L, metatable);        /* metatable.__index = methods */
  lua_pushliteral(L, "__metatable"); /* hide metatable */
  lua_pushvalue(L, methods);
  lua_settable(L, metatable);        /* metatable.__metatable = methods */
  luaL_openlib(L, 0, meta_methods,  0); /* fill metatable */
  luaL_openlib(L, 0, image_methods, 1); /* fill Image methods table */
  lua_settable(L, LUA_GLOBALSINDEX); /* add Image to globals */
  return 0;
```

```
}


int main(int argc, char *argv[])
{
  lua_State *L = lua_open();

  luaopen_base(L);
  luaopen_table(L);
  luaopen_io(L);
  luaopen_string(L);
  luaopen_math(L);
  luaopen_debug(L);

  Image_register(L);

  if(argc>1) lua_dofile(L, argv[1]);

  lua_close(L);
  return 0;
}
```

Compiling the Code

This code can be compiled for Lua 5.0 as follows:

```
gcc gd.c  -L/usr/local/lib/ -llua -llualib -lgd -lpng
```

Lua Test Code

```
for n,v in pairs(Image) do print(n,v) end

size = 256

im = Image.new(size, size)

print(im)

white = im:colorallocate(255, 255, 255)

for i = 0,size-1,1 do
  c = im:colorallocate(0, i, i)
```

```
  im:line(0, 0, size-1 , i, c)
end

im:PNG'test.png'

-- debug.debug()
```

Test Code Output

```
$ ./a gd.lua
line    function: 0x100553b8
PNG     function: 0x10055500
colorallocate   function: 0x10055418
new     function: 0x10054ff8
userdata: 0x10055e98
```

This is the image created by the test code.

The next example shows how to use the *index* and *newindex* events to read and write structure members BindingWithMembersAndMethods