API Design in Lua and C Applications

By Mitchell
Lua Workshop 2016

Outline

- Introduction
  - Extending C with Lua
- API Design
  - User interface
  - Event loop
  - Custom code
  - General considerations
- Q & A

Introduction

- Lua is an embedded language
  - C library
- C applications embed Lua
  - Define custom objects, functions, etc.
  - Execute Lua code

Simple Example

- Defines a single global variable
- Not useful by itself
- Application logic
  - Button clicks
  - Key presses
  - Attempts to quit

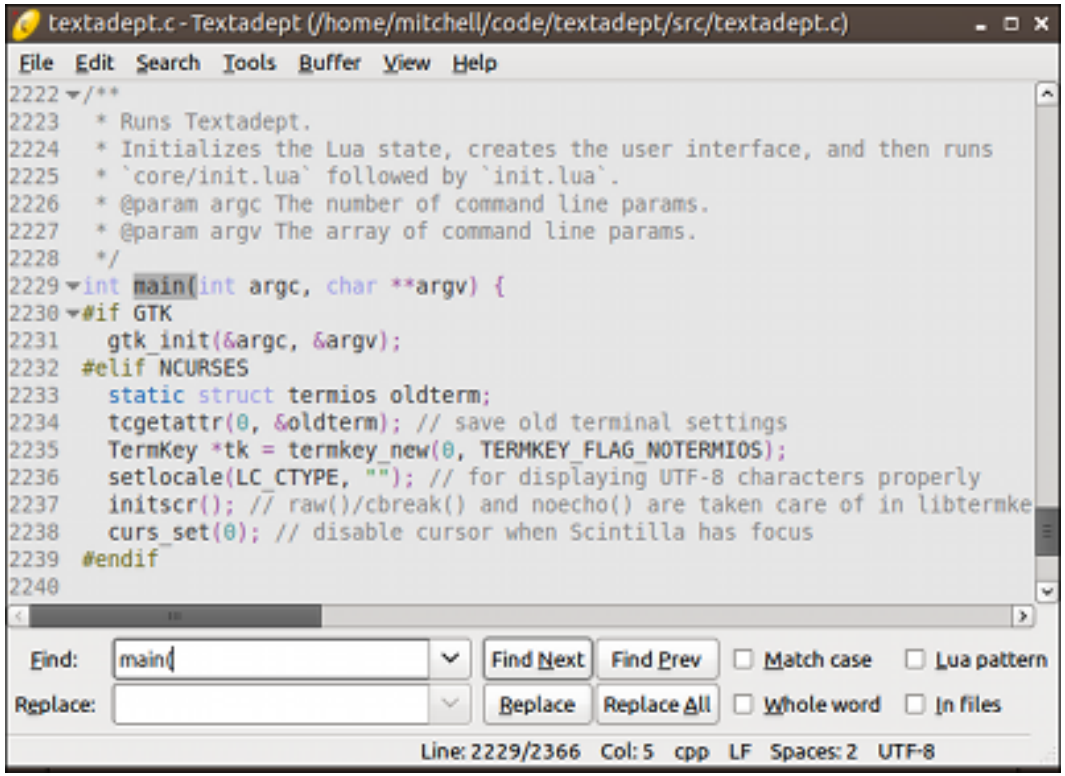
```c
#include <lua.h>
#include <lauxlib.h>

/* ... */

int main(int argc, char **argv) {
  lua_State *L = luaL_newstate();
  lua_pushstring(L, "hello!");
  lua_setglobal(L, "hello");
  /* ... application logic ... */
  lua_close(L);
  return 0;
}
```

Textadept

- Fast, minimalist, remarkably extensible
- Tiny C core
  - Embeds Lua
  - Skeleton UI
  - Editing component
  - That’s it!

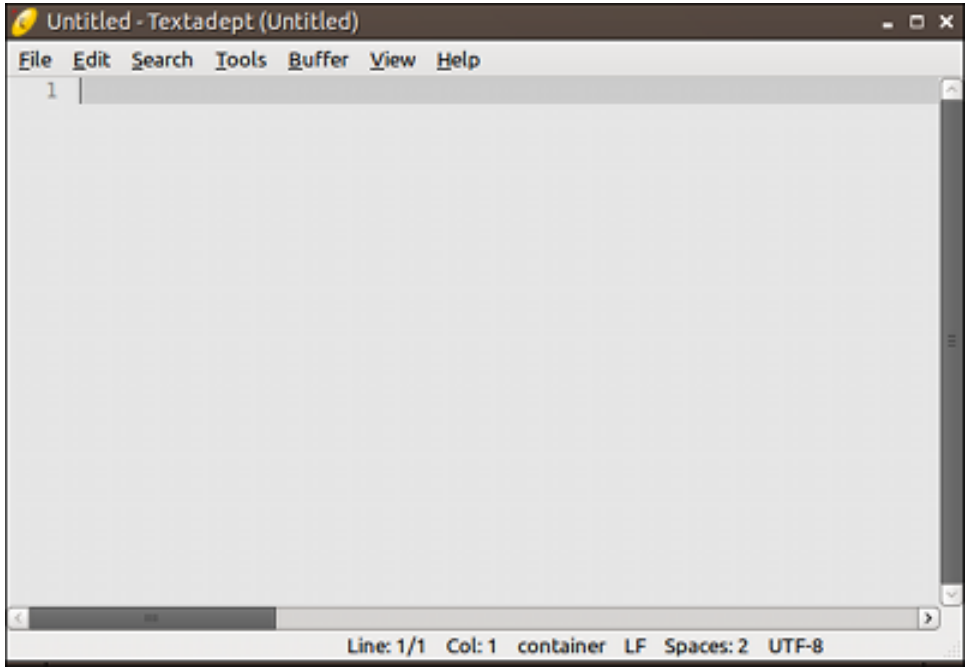


Sample Textadept API

- C core provides Lua with:
  - WIN32, OSX, LINUX, BSD, and CURSES
  - ui.find.find_next() and ui.find.find_prev()
  - ui.size = {width, height}
  - buffer.new() and buffer.char_at[pos]
  - view:split() and view.size
- A bit of everything: globals, modules, functions, objects, and metamethods

User Interface

- Text entries, buttons, menus, labels, etc.
- Events
  - Button clicks
  - Key presses
  - Menu selections
  - Files dropped
  - Life-cycle
    - Init, quit, suspend, resume, etc.



UI Elements

- Represented with empty table and metatable

```
ui.find.entry_text = "..."
ui.find.match_case = true
-- Perform the Find.
```

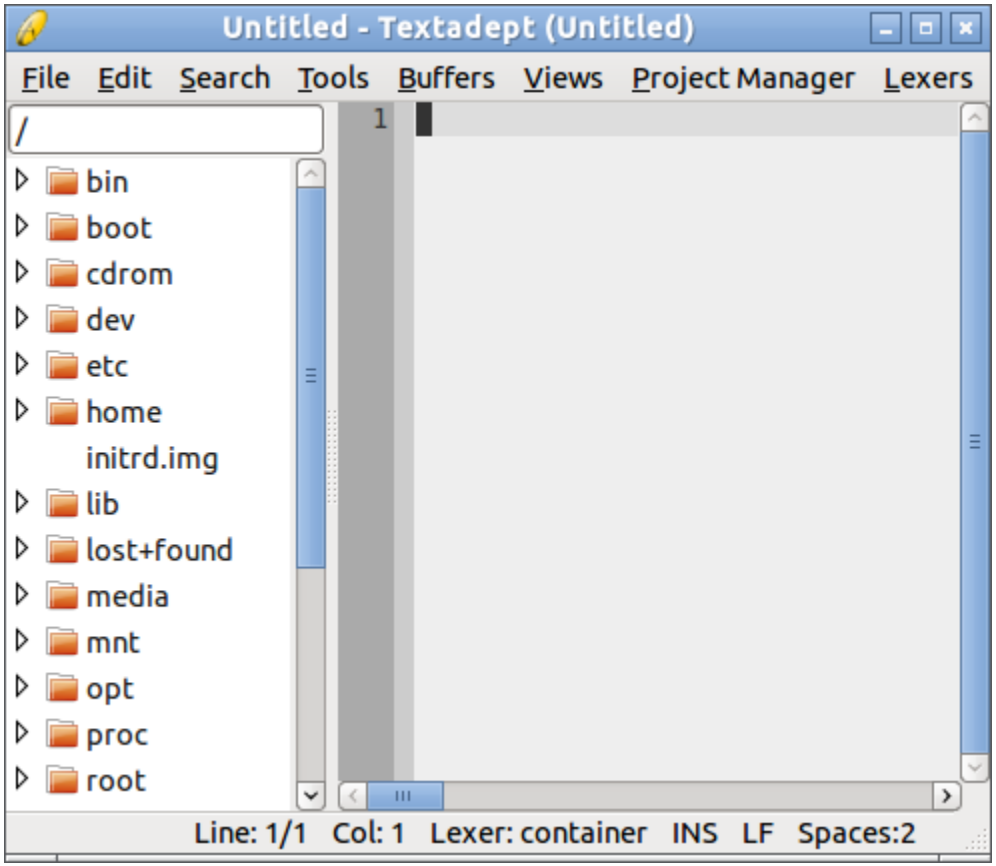
```
static int ui_find_index(lua_State *L) {
  const char *key = lua_tostring(L, 2);
  if (strcmp(key, "entry_text") == 0)
    /* push Find text */
  else if (strcmp(key, "match_case") == 0)
    /* push Match Case state */
  return 1;
}

static int ui_find_newindex(lua_State *L) {
  const char *key = lua_tostring(L, 2);
  if (strcmp(key, "entry_text") == 0)
    /* set Find text from 3rd arg */
  else if (strcmp(key, "match_case") == 0)
    /* set Match Case state from 3rd arg */
  return 0;
}
```

```
lua_getglobal(L, "ui");
lua_newtable(L);
/* ... set metatable and metamethods ... */
lua_setfield(L, -2, "find");
```

Beware

- Easy to go overboard
  - Complex widgets
  - Needless Lua interfaces
- Keep it simple, minimalist



Aside: External Components

- C or C++ with own API
- Scintilla editing component
  - API has Windows roots
  - SCI_*MESSAGE*(lParam, wParam)
    - `SCI_INSERTTEXT(int pos, char *text)`
    - `SCI_SETEOLMODE(int eolmode, void)`
    - `SCI_GETCHARAT(int pos, void)`

Lua Mapping

- Object-oriented Scintilla → Lua API
  - `SCI_INSERTTEXT(int pos, char *text)`
    `→ buffer:insert_text(0, "foo")`
  - `SCI_SETEOLMODE(int eolmode, void)`
    `→ buffer.eol_mode = 2`
  - `SCI_GETCHARAT(int pos, void)`
    `→ buffer.char_at[0] --> "f"`

Lua Mapping Details

- buffers are tables with metamethods
- For a given key, look up Scintilla message, arguments
  - For functions, return callable closure
  - For simple properties, get/set them
  - For tabular properties, return a new, empty table with metamethods that get/set based on key

Illustration

- `buffer:insert_text(0, "foo")`
  - Key is `"insert_text"`
  - Lookup yields SCI_INSERTTEXT with `int` and `char*` arguments

```
/* from buffer's __index metamethod */
lua_pushcclosure(L, closure, 1); /* key lookup value */

static int closure(lua_State *L) {
  int message = lua_tonumber(lua_upvalueindex(1));
  int pos = luaL_checknumber(L, 2); /* buffer is at 1 */
  char *text = luaL_checkstring(L, 3);
  return (SendScintillaMessage(message, pos, text), 0);
}
```

Event Loop

- C callbacks
- Can notify Lua
  - Lua functions
  - Event dispatch

```c
static void keypress(/* ... */) {
  lua_getglobal(L, "on_keypress");
  if (lua_isfunction(L, -1)) {
    /* push key and modifiers */
    lua_pcall(L, /*nargs*/, 0, 0);
  } else lua_pop(L, 1);
}
```

vs.

```c
static void keypress(/* ... */) {
  lua_getglobal(L, "events");
  lua_getfield(L, -1, "emit");
  lua_pushstring(L, "keypress");
  /* push key and modifiers */
  lua_pcall(L, /*nargs*/, 0, 0);
  lua_pop(L, 1); /* events */
}
```

Event Dispatch

- Superior to hard-coded callbacks
  - Multiple listeners
  - Custom overrides
    - Regex find ↔ Lua pattern find

```
events.connect("keypress", function(...)
  if condition then --[[ Handle keypress. ]] end
end)

events.connect("find", function(...)
  -- Override default Find behavior.
  return true
end, 1)
```

Running Lua Scripts

- Two-stage process
  - Application scripts
  - User scripts

```
int main(int argc, char **argv) {
  /* ... */
  lua_State *L = luaL_newstate();
  /* ... populate Lua environment ... */
  luaL_dofile(L, APP_HOME "/core/init.lua"); /* app scripts */
  /* ... any extra initialization ... */
  luaL_dofile(L, USER_HOME "/init.lua"); /* user scripts */
  /* ... main loop ... */
  lua_close(L);
  return 0;
}
```

General API Design Considerations

- Globals pollution
- Modularizing/Namespacing
  - Evolution is normal
- 3rd-party Lua modules
  - Naming conflicts
- Intuitive?
  - Developers vs. Users

Wrap Up

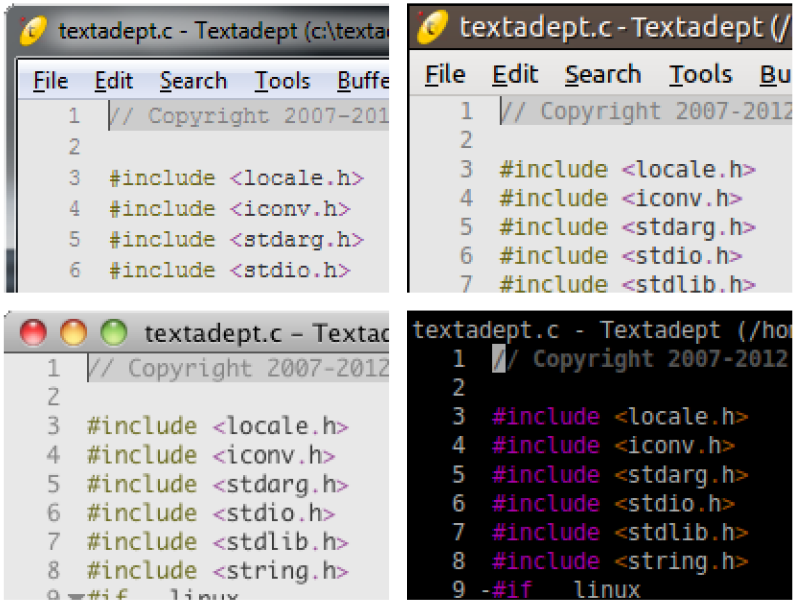
- Extend your C applications with Lua
- Design an API Lua can use
  - Interact with UI & external components
  - Take full advantage of event loop
  - Run custom user code
  - Be mindful of Lua environment

Final Thoughts

- Applicable to any application with embedded languages
  - Python
  - JavaScript

Thank You

- Questions?



Textadept: http://foicica.com/textadept