celedev

Using Lua for Responsive Programming of iOS apps

Workshop 2013

Jean-Luc Jumpertz

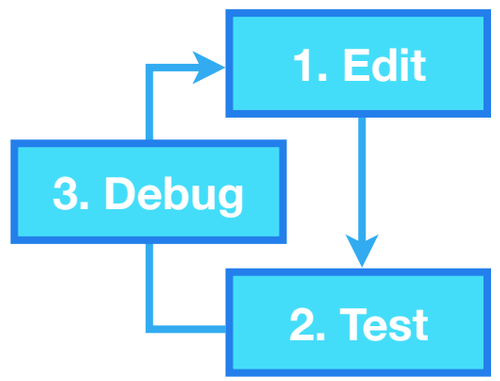
www.celedev.com

celedev

Responsive Programming on iOS

- The iOS ecosystem
  - powerful mobile devices: iPads, iPhones
  - very large and complex SDK
    - to build apps with sophisticated User Experience
  - large number of developpers

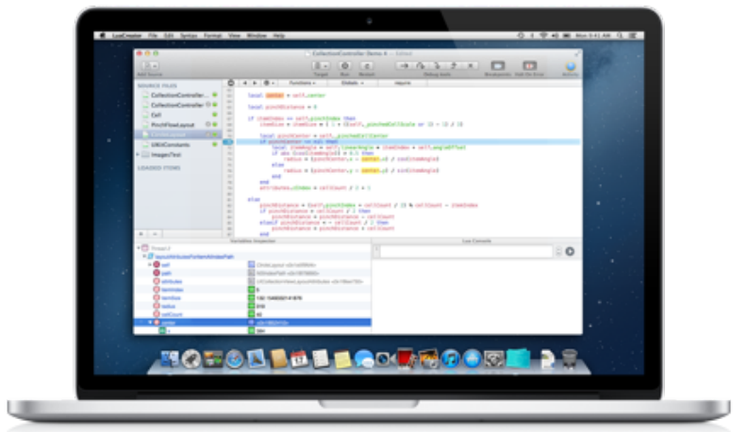
- Responsive Programming
  - all about interactivity between a developer and his application
  - while giving access to the entire iOS SDK
  - more than just live-coding



- Major benefits
  - fast prototyping and fine-tuning of apps
  - fun to use, encourage experimentation, enable creativity

celedev

System Components



Development Environment

- Lua Code Editor
- App Resources Manager
- App Monitor / Debugger
- Device manager

OS X

Host Computer

Application

Dynamic code (Lua)

Compiled code

iOS SDK Bindings

X Bindings

Celedev Runtime

X SDK

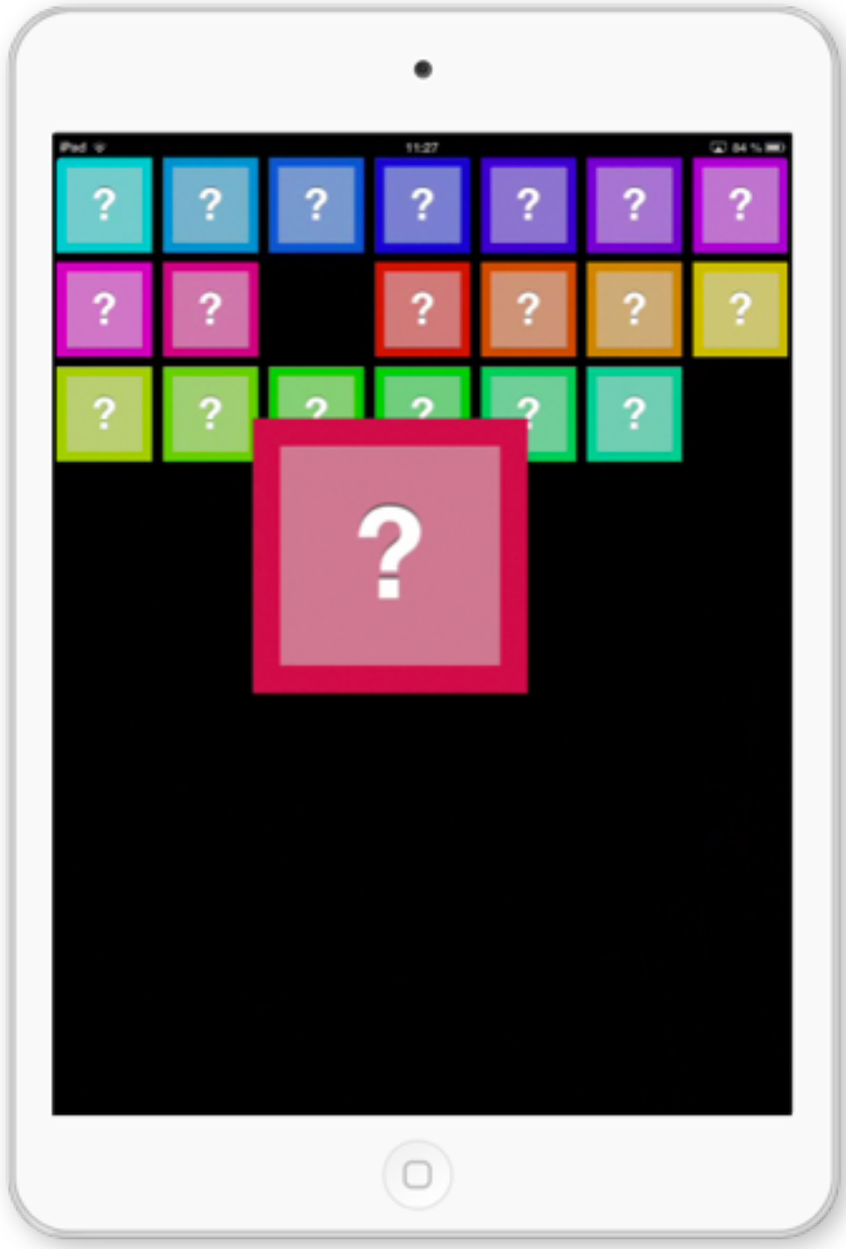
OS + SDK iOS

Target iOS device

celedev

Demo : live collections

- Ultra-simple app with a CollectionView-based screen
- Entirely written in Lua
- 3 Lua classes inheriting from ObjC SDK classes:
  - controller,
  - cell,
  - layout



celedev

Why Lua?

- Clean, easy-to-learn, powerful syntax
- Dynamic language
- Lua C API
- Small memory footprint and performance
- Instantiable VM
- Easy-to-sandbox
- Business-friendly open-source license
- Not bloated with overkilling standard libraries
  - The «battery not included» option is perfect in Celedev's case as tons of «batteries» are provided by the IOS SDK

celedev

Integrating Lua

- System characteristics
  - event-triggered execution of Lua code
    - e.g. user interaction, timer…
  - no active waiting or polling
    - as it would impact the battery life
- Software design choices
  - Latest Lua version (5.2.2)
  - Runtime code entirely written in C, using the Lua C API
  - Lua is for application code
  - Avoid weird twisting of Lua syntax ;)
  - Keep the Lua VM code unmodified
    - except where absolutely needed :)

celedev

Integrating Lua

- Objective C API
  - provides a simplified view of a Lua State to an iOS developper
  - Sample code

```
CIMLuaContext* luaContext = [[CIMLuaContext alloc] initWithName:@"MyLua"];

// Set a Lua global
luaContext [@"foo"] = @"Hello World";

// Same as Lua: self.rootController = require "MyModule"
[luaContext loadLuaModuleNamed:@"MyModule"
            withCompletionBlock:^(id result) {

    if ([result isKindOfClass:[UIViewController class]])
    {
        self.rootController = result;
    }
}];
```

celedev

Multi-threading Lua

- Why bothering with multiple threads?
  - The external world (the native app code) is multi-threaded
  - The native code calls the Lua Context
    - potentially from various threads
  - The Lua Context calls the native code
    - some native functions shall be called in specific threads
  - Lua code shall not slow down user interaction in the app
    - Better to run Lua code out of the main thread (user events loop)
  - Lua GC can be executed faster when CPU is idle
    - i.e. in a low-priority background thread
- But Lua can only run safely from a single thread

celedev

Multi-threading Lua

- Internally Lua has everything we need (almost)
  - a thread structure: lua_State,
  - lua_newthread function in the C API
  - macros to track Lua threads creation and deletion: `luai_userstatethread`…
- What we need to add to make it work
  - a simple lock to serialize the execution of Lua from multiple threads
  - well-chosen descheduling points
    - good candidates: debug hook, C function call from Lua, Lua thread function return
  - a basic asynchronous messaging service
- Avoiding pitfalls
  - keep a reference to secondary Lua threads to prevent Garbage Collection
  - … but do not leak Lua threads
  - carefully design the Lua threads scheduler to avoid deadlocks
  - make it invisible to the executed Lua code

celedev

Object-Oriented Framework

- Goals:
  - integrate Lua code transparently with iOS SDK and Objective C runtime
  - provide a unified and simple model for Lua and Objective C objects
  - support dynamic code update by design
- Main Features
  - expose a Lua object model fully compatible with ObjC concepts
  - symmetrical model
    - Lua can call any method of an Objc instance or class
    - ObjC can call any published method of a Lua instance or Class
  - create a Lua class as a subclass of an ObjC class (or of a Lua class)
  - declare that a Lua class conforms to an ObjC protocols
    - this publishes the Lua methods defined in this protocol to ObjC

celedev

Object-Oriented Framework

- Code example

```
local UIView = require "UIKit.UIView"
local UIFont = objc.UIFont
local UIColor = objc.UIColor

local Cell = class.createClass ("LabelCell", objc.UICollectionViewCell)

function Cell:setAppearance (cellIndex, cellCount)

    -- ensure that params are not nil
    cellIndex, cellCount = cellIndex or 0, cellCount or 1

    local contentView = self.contentView
    local contentSize = contentView:bounds().size

    -- Text label
    local label = self.label

    label.frame = { x = 0, y = contentSize.height / 4,
                    width = contentSize.width, height = contentSize.width }
    label.font = UIFont:boldSystemFontOfSize (46.0)
    label.textColor = UIColor.whiteColor
end

return Cell
```

celedev

Object-Oriented Framework

- Implementation
  - in C, using Lua C API
  - classical (yet complex) Lua objects implementation based on metatables and `__index` & `__newindex` metamethods
  - internally, 2 different kinds of objects (hidden from the user)
    - Lua-only objects implemented as tables
    - Lua-objc objects implemented as userdata + uservalue
  - Objects lifecycle compliant with both worlds
    - reference-counted ObjC objects
    - garbage-collected Lua objects
- Performance aspects
  - the heavily-used metamethods are the most critical regarding performance
  - avoid pushing C strings to Lua in performance-critical code
  - replaced by upvalues where appropriate, and lua_rawgetp or lua_rawgeti elsewhere

celedev

Dynamic Code Update

- Dynamic code update is managed by Celedev IDE in association with the runtime
  - managed at the Lua module level
- A Lua module is updated when
  - it is syntactically correct
  - the module syntax has been changed since the last loaded version
- Lua require() function rewritten for dynamic update
  - get the latest version of a module from
    - the connected Celedev IDE, if present
    - the application package otherwise

celedev

The debugger

- Good debug tools are essential for serious software development
- The Celedev remote Debugger has been specifically designed for supporting the Responsive Programming environment
  - fully multi-threads aware
    - including multi-threaded debug of Lua coroutines
  - integrated with the Object-Oriented framework
    - includes a full-featured class-hierarchy inspector
  - integrated with the Dynamic code update feature
    - can debug functions in module old versions when needed
- Demo

celedev

Conclusion

- All this works pretty well!
- Lua is extremely well-designed for embedding
  - excellent C API stack model, small and readable source code
  - the only language I know for which 50% of the ref. manual is about C integration
  - highly useful hooks for advanced integration: `luai_userstatexxx`, `luai_writestring`, `lua_assert...`
- No necessary feature missing
  - see Occam's razor «*Entia non sunt multiplicanda praeter necessitatem*»
- Places for improvement
  - long integer values for bridging with 64 bits systems (Lua 5.3?)
  - better parser errors detection: range-based, more accurate diagnostics…
  - garbage collection: could it be made it more *transparent*?

celedev

Conclusion

Lua is a *scripting* language…

…capable of running complex applications

celedev

Conclusion

Lua is a *scripting* language…

…capable of running complex applications

Lua is a *small* language…

…but it's bigger on the inside

celedev

Conclusion

Lua is a *scripting* language…

…capable of running complex applications

Lua is a *small* language…

…but it’s bigger on the inside

celedev

Thank You!

For more information about Celedev:

- website:www.celedev.com
- mail: jean-luc@celedev.eu
- twitter: @celedev

My Lua public projects:

- LuaSyntaxer: https://bitbucket.org/jean_luc/luasyntaxer
- Lua 5.2 + JL patches: https://bitbucket.org/jean_luc/lua-5.2-jl-patches