The History & Design of LuaCocoa

Eric Wing
@ewingfighter
http://playcontrol.net/opensource/LuaCocoa

Background: Lots of different hats to wear

- Worked on Globalstar (orbital satellite global communication system)
- Cross-platform: Linux, Mac, Windows, iOS, Android, FreeBSD, Solaris, IRIX
- Scientific Visualization (OpenGL)
- Native Cocoa Application development (Mac, iOS)
- Video Game Engines
- Open Source: SDL, OpenSceneGraph, CMake, OpenAL
- C, C++, Obj-C, Lua, Perl, Java, JavaScript, Swift
- Book author (Beginning iPhone Games Development)
- Startup cofounder

GIST CANCER

The Biggest GIST CANCER Event of the Year!

GIST (Gastrointestinal Stromal Tumor) is an orphan cancer affecting over 5,000+ patients in the U.S. alone. It is ruthless with no known cure. Please join us & walk to find a Cure & bring Hope to all **GIST** patients who are waiting for a Miracle.

OCTOBER 23, 2016

RESEARCH FUND

PLACE

Almaden Lake Park
San Jose, CA

REGISTRATION

No Fee
Registration Required

Registration: 9:30 AM
Walk Start: 10:30 AM

WALK FOR A CURE

Objective-C

- Small, ***pure superset*** of C to add OO features to C
  - (funny syntax was deliberately picked to not conflict with C, C++ so they can be intermixed)
- Created in the 80’s, around the same time as C++, but SmallTalk influences
- Closest thing to SmallTalk you can get with C
- Messaging another core concept of Obj-C

Objective-C

- Likes late-binding and dynamic dispatch
- Pretty small runtime
- Pretty efficient introspection & reflection
  - Not usually a concern unlike in Java, C#

Objective-C

- Unusual blend of statically compiled C typed language + dynamic features found in scripting languages

- Runtime C functions

```
#import <objc/objc.h>
#import <objc/runtime.h>
id objc_msgSend(id self, SEL op, …);
```

Cocoa

- Huge standard library for Obj-C
- Foundation for non-GUI
- AppKit for GUI on Mac
  - UIKit for iOS
- Originally from NeXTStep
  - Foundation + AppKit named Cocoa as pun of Java by Apple
- Lots of other frameworks in addition to Cocoa
  - Some C, some Obj-C
  - CoreData, CoreGraphics, AudioUnits, OpenGL, etc

Cocoa designed for GUI development

- Exploits a lot of crazy dynamic runtime features of Obj-C under the hood
- Responder Chain
- Key Value Coding
- Key Value Observing
    - isa-swizzling (aka method swizzling)
- Cocoa Bindings
- Core Data
- NSUndoManager
- Interface Builder

LuaObjCBridge (Tom McClean, ~2005)

- Automatically binds classes & methods at runtime (no compile time glue code)
- Simple: Originally 1000-1500 lines including #ifdefs for GNUStep/Apple
- Used the only Obj-C 1.0 runtime
  - No additional things like libffi
- Used Lua 5.0

LuaObjCBridge limitations

- No subclassing in Lua
- Couldn't handle non-object types
  - C structs, functions, enums
- No automatic memory management
  - (had to call retain/release/autorelease in Lua code)
- No metamethods
  - (accidental design mistake used lightuserdata)

LuaObjCBridge method calls

- In Obj-C
    - [my_obj doSomethingWithObj1:obj1 andObj2:obj2];
- In LuaObjCBridge
    - my_obj:doSomethingWithObj1_andObj2_(obj1, obj2)
- Colons are replaced with underscores
    - Trailing underscore is optional in LuaObjCBridge and RubyCocoa
    - PyObjC makes an impassioned argument that this is bad (ambiguous case, etc) which I eventually was convinced of
- Trivia: Obj-C doesn't have "true" named parameters. Order matters and is part of the method name.

LuaCore (Gus Mueller ~2006)

- Cocoa uses a lot of struct types (NSRect, NSPoint, NSSize) in graphics APIs
- LuaCore introduces manual bindings for common types and C APIs in CoreGraphics
- Could be used in conjunction with LuaObjCBridge
- I took both and started enhancing them
    - Lots of experimentation with structs in particular
    - Became kind of a proving ground precursor to LuaCocoa
    - Also eventually ended up being maintainer for these projects

Apple WWDC 2006 (Everything changes)

- Objective-C 2.0
- BridgeSupport: fill the holes for language bridges
- Scripting Bridge
- PyObjC & RubyCocoa officially supported by Apple
  - Ships on the system
  - Xcode & Interface Builder integration

Objective-C 2.0

- Breaking changes to Obj-C
- Introduces 64-bit Obj-C support
    - Need to support PowerPC, Intel, 32-bit, 64-bit moving forward
    - Apple has been transitioning to first Intel, and then 64-bit
        - Opportunity to create a new ABI, so best time to make breaking language changes
- Runtime functions LuaObjCBridge depended on were deprecated/unavailable
    - LuaObjCBridge needed a complete rewrite
- Obj-C 2.0 also introduces optional Garbage Collection

BridgeSupport: Make full binding at runtime possible (No static bindings needed)

- XML metadata for things which introspection cannot discover
  - C structs, C functions, enums, #defines, global variable names
- Intent of an API
  - BOOL is signed char, but really means boolean
  - in and out parameters
    - (LuaCocoa will use multiple return values for out parameters)
- Also provides .dylib with missing symbols for inline functions
- Open source command line tool ‘gen_bridge_metadata’ can be run on 3rd-party headers

BridgeSupport

- Apple used to ship both “fullbridgesupport” and (non-full) “bridgesupport” files
- Apple later removed “full”
- LuaCocoa originally used “full” so some migration pains
  - Probably better though because RAM and parse times went way down
- Non-full tool sometimes misses some needed APIs
  - Cobbled together some APIs to allow suppling additional metadata to fill holes

Scripting Bridge
(not to be confused with BridgeSupport)

- Allows programs to communicate with others through AppleScript dictionary in other languages
  - Nobody knows how to write AppleScript (read-only language)
  - Objective-C API which LuaCocoa can go through
- Examples
  - iTunes (play a song, get info)

```
 LuaCocoa.import("ScriptingBridge")
 local itunes_application =
SBApplication:applicationWithBundleIdentifier_("com.apple.iTunes")
 itunes_application:pause()
 itunes_application:playpause()
 local itunes_track = itunes_application:currentTrack()
 NSLog("Currently playing: Name:%@, Artist:%@, Year:%d",
itunes_track:name(), itunes_track:artist() ,itunes_track:year())
```

  - Xcode (test automation)
    - http://playcontrol.net/ewing/jibberjabber/automated-mobile-test-part2.html
- LuaCocoa also comes with a command line tool: ‘luacocoa’ (slight modification of ‘lua’) to also allow command line scripting

LuaCocoa philosophy

- Like Lua: “Mechanisms, not policy”
  - Use as a library (not a framework)
    - Use as you see fit, as little or as much as you want
- Support BOTH sides of the bridge
  - Other bridges: “Write everything in the language”
  - LuaCocoa: “You *could* write everything, but may miss the point”
    - Xcode, Interface Builder, clang, llvm, lldb are powerful tools
- Also means optional Obj-C GC must be supported

Consequences of philosophy

- Like most bridges, need to support all the Lua/scripting side stuff as expected
- Unlike other bridges, also need good C/Obj-C support to make interacting from the C/Obj-C side easier
- Obj-C optional Garbage Collection must be supported
    - You make the call on GC, not me

Additional Research (Lua side)

- LuaObjectiveC (Steve Dekorte)
  - Lua 3.1, ~2001
  - Trivia: Gave very first talk at first Lua Workshop on IO Language
- LuaBridge (Richard Kiss)
  - Lua 4.0, ~2001
- Adobe Lightroom (Mark Hamburg and his team)
  - Lua Workshop 2005

Additional Research (Obj-C side)

- PyObjC (talk by Bill Bumgarner (Apple Obj-C runtime))
- RubyCocoa (Laurent Sansonetti (Apple), MacRuby, RubyMotion)
  - Sat down with me one afternoon at WWDC to explain how RubyCocoa was implemented via BridgeSupport and libffi
- JSCocoa (Patrick Geiller)
- Perl CamelBones: Sherm Pendley (Rest In Peace)

LuaCocoa Examples

- Hybrid live coding (Core Animation)
- LuaBork: Swedish Chef Translator (LPeg)
    - Or email address validation
- Android package name validation (Java BNR)
- Lua table <=> NSDictionary <=> Info.plist serialization

LuaCocoa Begins (MIT License)

- PowerPC & Intel
  - 32-bit & 64-bit
- Traditional (Reference Counting) & Obj-C Garbage Collection
- Started with Lua 5.0
  - Eventually migrated to Lua 5.1

Patches to Lua

- LNUM (Asko Kauppi)
  - 64-bit integers cannot be stored in double without loss
  - long double didn't work (and probably bad anyway)
  - LNUM tries to preserve integers and floating types behind the scenes
  - Intended for numeric computation, author amused by my use
  - No Lua 5.2 support
  - Lua 5.3 supports integers!

Patches to Lua

- Objective-C exception handling (me)
  - pcall doesn't know anything about Obj-C exceptions
  - Lua has macro for C++ exceptions, but Obj-C doesn't perfectly fit
  - Useful for all Lua on Mac/iOS environments, not just LuaCocoa
    - Correctness / Safety
    - Performance: Provides "zero-cost" exceptions
    - Bonus: Also handles C++ exceptions
  - Wish list: Lua refactors exception handling code to make writing this easier or incorporates patch directly
    - (I already have some new patches for Lua 5.3 for other projects)

libffi

- Foreign Function Interface Library
  - Low-level (assembly) implementation to provide common C interface to call functions
- Using actual libffi directly, not LuaJIT 2.0 or LuaFFI
  - LuaCocoa predates these by years
  - Mentioned because I’ve gotten confused on the Lua mailing list about libffi questions where they presume LuaFFI or LuaJIT

libffi

- LuaObjCBridge just used objc_msgSend family
  - LuaObjCBridge didn't handle C functions
- Obj-C 2.0 removed a bunch of functions that were needed for the bridge
- Lots of corner cases about which objc_msgSend to call
  - depends on architecture and size of parameters & return values
- libffi more robust and easier to marshall parameters
  - LuaCocoa built around libffi
  - Mac ships with public libffi included
- (For those who know of NSInvocation, it is awful.)

libffi

- Generality:
  - Lua => C/Obj-C functions/methods/blocks uses libffi
  - C/Obj-C => Lua goes through normal Lua C API
    - Other bridges like PyObjC/RubyCocoa don’t have such a powerful API and may resort to libFFI in both directions

LuaCocoa.import()

- LuaCocoa.import(“Foundation")
  - Loads BridgeSupport data and dlopen framework and extra .dylib
- Kind of like require, and would like to unify
- But used extra parameter to work around some cases
  - LuaCocoa.import("CoreGraphics", "/System/Library/Frameworks/ApplicationServices.framework/Frameworks")
- Results in needing a separate standalone executable “luacocoa” instead of providing just “lua”
  - Though needing an autorelease pool also results in this
- TODO: Revisit and fix

C Functions

- BridgeSupport + libffi + dlopen + dlsym allows us to automatically bind C functions to Lua at runtime
- Inline functions are solved by loading an extra .dylib provided by BridgeSupport which provides symbols to access
- Yes, variadics are supported
  - Metadata says whether null terminated or printf token based. LuaCocoa handles appropriately.

```
local array = NSArray:arrayWithObjects_(obj1, obj2, nil)

NSLog("array: %@: ", array)
```

structs

- Now using BridgeSupport
- Can get the proper names of fields
- Also overloaded a bunch convenience features

structs

- Definition in C

```
typedef struct _NSPoint {
    CGFloat x;
    CGFloat y;
} NSPoint;

typedef struct _NSSize {
    CGFloat width;
    CGFloat height;
} NSSize;

typedef struct _NSRect {
    NSPoint origin;
    NSSize size;
} NSRect;
```

- Definition in BridgeSupport

```
<struct name='NSRect' type='{_NSRect=&quot;origin&quot;
{_NSPoint=&quot;x&quot;f&quot;y&quot;f}&quot;size&quot;
{_NSSize=&quot;width&quot;f&quot;height&quot;f}}' type64='{CGRect=&quot;origin&quot;
{CGPoint=&quot;x&quot;d&quot;y&quot;d}&quot;size&quot;
{CGSize=&quot;width&quot;d&quot;height&quot;d}}'/>
```

- Tricky:
    - In 32-bit ABI, NSRect & CGRect (and friends) are different definitions
    - 64-bit ABI, Apple redefined NSRect to be a typedef to CGRect
    - Subtle implications for implementation, but don’t have time to cover

structs

- Can represent as tables, nested tables, array access, flat ordered values, etc.

```
local ns_rect = NSMakeRect(300, 400, 500, 600)
ns_rect.origin = {1000, 2000}
ns_rect.size.width = 1000
ns_rect({ {1001, 2002}, {3003, 4004}})
ns_rect({ 1011, 2022, 3033, 4044})
ns_rect(1111, 2222, 3333, 4444)
ns_rect({1110, 2220}, {3330, 4440})
ns_rect({x=1010, y=2020}, {width=3030,
height=4040})
ns_rect({x=0010, [2]=0020}, {[1]=0030,
height=0040})
ns_rect({origin = {x=10101, y=20202}, size =
{width=30303, height=40404}})
```

Objects revisited

- Unlike LuaObjCBridge, now use full user data to use metamethods
    - __tostring, __eq, __gc
- Specialized user data for NSArray, NSMutableArray, NSDictionary, NSMutableDictionary, NSNumber, NSNull, NSString, NSBlock
    - __len, __index, __newindex, __tonumber, __concat, __call
- __gc is the most important since it is the basis for automatic memory management

NSArray/NSDictionary

- calling NSArray objectAtIndex: counts from 0: array:objectAtIndex_(0)
  - But calling through Lua brackets counts from 1: array[1]
- APIs and auto-coercion for Lua tables and NS
  - Must copy/convert when this happens
  - When converting to NS, because Lua tables can be either, both data NS structures get created until one can be discarded
- NS* can only hold objects
  - Numbers auto-boxed in NSNumber
  - Lua functions can now be boxed in Blocks
    - (requires signature)
  - Non-object user data gets dropped

Proper Memory Management

- 2 sides of the LuaCocoa bridge: Coding on both sides is supported
  - Lua side is expected to be Lua garbage collected
  - Obj-C side is either classic reference counting, Obj-C garbage collection, or ARC
- Thought Exercises:
  - We create an object in Obj-C and push it to Lua
    - If the object goes away in Obj-C, but still active in Lua, it must stay alive
    - What if we push the same Obj-C in Lua twice?
      - (Same user data or different user data?)
  - We create an object in Lua and push it to Obj-C
    - If the object goes away in Lua, but still active in Obj-C, it must stay alive

Obj-C Optional Garbage Collection (2006-2012)

- GC is now deprecated. However, dealing with it had a big impact on LuaCocoa's implementation so it is worth discussing
- Obj-C GC is runtime activated
  - Burden placed on library writers to write code such that their code would work either way
- LuaCocoa was the first to support dual-mode. Most did not.
- LuaCocoa's commitment to supporting code on both sides of the bridge forced extra careful thought and design

Obj-C Optional Garbage Collection (2006-2012)

- In Obj-C there was kind of a dance to support dual mode
  - In GC-mode, retain/release/autorelease become no-ops
  - But CFRetain/CFRelease are still meaningful (needed for Apple's bridged C-types)

```
// Creation dance
NSObject* the_object = [[NSObject alloc] init];
CFRetain(the_object); // Always meaningful
[the_object release]; // Only in non-GC

// When finally done:
CFRelease(the_object); // Always meaningful
```

- Also other complicated edge cases I won't get into

Memory Management

- *We create an object in Obj-C and push it to Lua*
    - *If the object goes away in Obj-C, but still active in Lua, it must stay alive*
- Solution: Every time we create a new userdata to push to Lua, we call CFRetain(). *(Note: Do not use Obj-C retain which is a no-op in GC)*
    - When the __gc finalize gets called, we call CFRelease()

Memory Management

- *What if we push the same Obj-C object in Lua twice?*
  - *(Same user data or different user data?)*
- Solution: We reuse the same user data
  - All Obj-C objects will have a 1-to-1 relationship with a Lua user data at a given time which helps simplify some things
  - Keep a map between Obj-C object address and user data in a weak table (in the Lua Registry)
    - Weak table prevents us from accidentally rooting the object and leaking
    - If object is already in Lua, we reuse that user data
    - If not, we create a new user data
    - When the object leaves Lua and gets garbage collected, the user data and weak table should go away.
  - Also helps minimize a performance problem of creating too many temporary objects that need to be collected

Memory Management

- *We create an object in Lua and push it to Obj-C*
  - *If the object goes away in Lua, but still active in Obj-C, it must stay alive*

- Solution: This one mostly works itself out if the object is pure Obj-C. Obj-C side is expected to use normal Obj-C memory semantics to keep it alive while in use. If pushed back into Lua, a new user data will be created.
  - If the object is a subclass created in Lua, things get nastier. Too much detail for this talk.

Memory Management

- Summary:
  - I am very grateful for Lua weak tables
  - I am very grateful for how simple and straight-forward the __gc metamethod rules work
    - (I haven’t dealt with Lua 5.2 and resurrection. Should I be worried?)
  - The Lua Registry and the stack API of Lua was also really nice to have here so I could refer to values easily already in Lua without having even more pointers to deal with and risk making more mistakes with the Obj-C side of the memory life-cycles (is it GC or non-GC, etc)

Subclassing in Lua

- Cocoa is an object-oriented framework so some classes require you to subclass
- However, early on Cocoa realized the pitfalls of subclassing and started preferring delegation (pattern)
- But to use delegates in Cocoa, you still need to create your own class
- Very grateful Lua is not object-oriented by default
  - PyObjC, RubyCocoa have to deal with multiple-inheritance headaches because of the two OO systems colliding
  - Really drove home how well designed Lua is for embedding + co-existing
    - Can embed and embrace this foreign environment without creating this clash/conflict
    - Metamethods allow conformance to OO features as needed

Defining a new Class

- Second parameter can be a Class or string
    - SimpleLuaOpenGLView = LuaCocoa.CreateClass("SimpleLuaOpenGLView", NSOpenGLView)
- Protocols are listed at the end
    - OpenPanelDelegate = LuaCocoa.CreateClass("OpenPanelDelegate", NSObject, "NSOpenSavePanelDelegate")

Subclassing in Obj-C

```
#import <Cocoa/Cocoa.h>
#import <OpenGL/gl.h>

@interface SimpleLuaOpenGLView : NSOpenGLView
{
}
- (void) drawRect:(NSRect)the_rect;
@end

@implementation

- (void) drawRect:(NSRect)the_rect
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(1.0, 0.85, 0.35);
    glBegin(GL_TRIANGLES);
        glVertex3f(  0.0,  0.6, 0.0);
        glVertex3f( -0.2, -0.3, 0.0);
        glVertex3f(  0.2, -0.3, 0.0);
    glEnd();

    glFlush();
}

@end
```

Subclassing in LuaCocoa

```
LuaCocoa.import("Cocoa")
LuaCocoa.import("OpenGL")

SimpleLuaOpenGLView = LuaCocoa.CreateClass("SimpleLuaOpenGLView", NSOpenGLView)

SimpleLuaOpenGLView["drawRect_"] =
{
    “-v@:{CGRect={CGPoint=dd}{CGSize=dd}}",

    function (self, the_rect)
        glClearColor(0, 0, 0, 0);
        glClear(GL_COLOR_BUFFER_BIT);

        glColor3f(1.0, 0.85, 0.35);
        glBegin(GL_TRIANGLES);
            glVertex3f(  0.0,  0.6, 0.0);
            glVertex3f( -0.2, -0.3, 0.0);
            glVertex3f(  0.2, -0.3, 0.0);
        glEnd();

        glFlush();
    end
}
```

Categories

- Objective-C has the ability to add new methods to classes you don't own
  - Handy for adding new helper methods without replacing all instances of a class with a new subclass
  - Also handy for implementing methods that have dependencies on otherwise unrelated classes
  - Can also be abused to override/replace an existing definition

Categories

```
NSView["rightMouseDown_"] =
{
   "-v@:@",
   function(self, the_event)
      NSLog("rightMouseDown_ %@", the_event)
   end
}
```

- Reuses same syntax as before
- TODO: Add API for Swizzling
  - Not commonly done in Obj-C (requires low-level runtime APIs)
  - But would allow for calling the existing implementation before replacing it
    - (Subclass overriding without subclassing)
  - Swizzling is used in LuaCocoa implementation for some things like dealloc/finalize

Subclassing Behind the Scenes

- Lots of tricky things
- A new Class (and an instance of it) is compromised of a Obj-C part and a Lua part (put in environment table)
    - Obj-C class registration is global/singleton so the Lua part needs to stick around
    - Because LuaCocoa supports multiple lua_States, the Lua part is kept around in a global list, with a map between lua_State & implementation
        - Assumption is that all scripts that use the class should define the class, and in exactly the same way.
            - But if the lua_State is closed (live-coding?), the global map will fall back to the next definition in the global list
                - This can break down if your implementation uses non-constant values local only to a particular lua_State
                    - You must decide on the trade-offs here

Subclassing Behind the Scenes

- Obj-C dealloc & finalize are tricky
  - There is extra clean-up LuaCocoa needs to do for every object
    - Can’t just call ‘super’ because there is no LuaCocoa intermediate object…user directly subclasses real Obj-C object
    - Also, finalize and some classes for dealloc trigger on a background thread which can be big trouble with Lua
  - Solution: Method swizzling is done to invoke the proper dealloc/finalize
    - Also compares the thread the lua_State was created on vs. the current thread. Attempts to redirect as necessary.

Subclassing Behind the Scenes

- ‘super’ doesn’t work the way you would hope
- [super dealloc];
  - Compiler figures out super at compile time. Doesn’t work for runtime bridge.
- class_getSuperclass & objc_msgSendSuper complicated interaction…hard to explain. Obj-C mailing list for help.
  - But trying to subclass a Lua subclass will break (i.e. 2+ levels)
    - Either incorrect implementation or infinite recursion
    - (Other Lua/Obj-C bridge implementations I looked at never get this right.)
  - Solution needed ffi_prep_closure (blocks unfortunately were not powerful enough)
  - Also ended up requiring you to explicitly name the super class you want to invoke in Lua
    - This might be omit-able, but just getting this far was a “miracle” so I gave up
    - self:super(NSNumberFormatter):decimalSeparator()

Blocks (2008)
(aka closures for C & Obj-C)

- Peter Norvig: Design patterns are sign of a missing feature / deficiency in a language
- Blocks are the most significant change to Obj-C (IMHO)
  - Changes the semantics of how you use the language
  - Completely eliminates the need for Cocoa Delegate Pattern
- New APIs require blocks, so LuaCocoa must support

Blocks Almost perfect, but not quite

- Could have replaced need for ffi_prep_closure and used blocks instead
- Hard to create on-the-fly with arbitrary signatures
  - Actually need ffi_prep_closure to create runtime blocks
- Not enough runtime information to get everything needed through introspection
- Hence still need BridgeSupport and libffi

LuaCocoa Blocks

- Lua functions can now be wrapped in Blocks and treated as Obj-C objects across the bridge
  - Creating blocks in Lua require an Obj-C method signature to represent the types of parameters and return value
    - Defining as a parameter to a function can automatically use BridgeSupport to find signature
    - Creating standalone blocks in Lua requires explicit manual signature
- Blocks can be invoked as functions in Lua (__call)
- Blocks need to be memory managed similar to other Obj-C objects. Also need to pay attention to Lua function life-cycle when wrapping Lua function. Don't have time to talk about this.
- Some blocks APIs invoke on background threads
  - LuaCocoa attempts to re-route to origin thread
    - However, deadlock issues with GCD concurrency options
    - Set concurrent options to NO if given the choice

LuaCocoa Obj-C side

- Public API in LuaCocoa.h
- Objective-C LuaCocoa class to get you going:

```
   LuaCocoa* lua_cocoa = [[LuaCocoa alloc] init];
   struct lua_State* lua_state = [lua_cocoa luaState];
   NSString* the_path = [[NSBundle mainBundle]
pathForResource:@"MyScript" ofType:@"lua"];
   luaL_loadfile(lua_state, [the_path
fileSystemRepresentation]);
   lua_pcall(lua_state, 0, 0, 0);
```

LuaCocoa Obj-C side

- Also contains a bunch of C APIs reminiscent of lua.h, but for Obj-C types

```
void LuaCocoa_PushInstance(struct lua_State* lua_state,
id the_object);
id LuaCocoa_ToInstance(struct lua_State* lua_state, int
stack_index);
bool LuaCocoa_IsInstance(struct lua_State* lua_state,
int stack_index);
```

LuaCocoa Obj-C side
(Example: Implement a class in Lua)

```
LuaCocoa.import("Foundation")

MyLuaClass = LuaCocoa.CreateClass("MyLuaClass", NSObject)

MyLuaClass["doSomethingWithaBool_aDouble_anInteger_aString_anId_"] =
{
	function (self, a_bool, a_double, an_integer, a_string, an_id)
		print("in subclass doSomething:", self, a_bool, a_double, an_integer, 
a_string, an_id)
		local ret_string = NSString:stringWithUTF8String_(a_string)
		return ret_string
	end,

	"-@@:Bdi*@"
}
```

LuaCocoa Obj-C side
(Use Lua class from Obj-C)

```
    Class MyLuaClass = NSClassFromString(@"MyLuaClass");

    id new_instance = [[MyLuaClass alloc] init];

    NSString* ret_string = [new_instance doSomethingWithaBool:true
aDouble:2.0 anInteger:3 aString:"hello world" anId:the_path];
```

App Sandbox
Mac 10.7 Lion (2011)

- Security => Principle of least privilege
- Required for Mac App Store
  - But also available outside (Developer ID / GateKeeper)
    - Generally a good idea if you can support it
- Early on, libffi (mprotect?) was triggering console warnings under App Sandbox
  - Suggestion that Apple was going to fix this
  - Not sure of status today (is LuaJIT allowed on Mac App Store?)
    - libffi fork for iOS has workaround for mprotect if a problem for MAS
- LuaCocoa seems to work under App Sandbox (with caveats)

App Sandbox gotcha:
Duck Typing

- App Sandbox locks down the file system
- NSOpenPanel/NSSavePanel have changed into a contract that infers the user has granted permission to a file
- But Apple pulled a fast-one and did a new ground up “secure” implementation of the file panels
    - Instead of making programmers write to the new classes, if App Sandbox is active, Apple secretly returns an instance of these new classes. (Class Clusters)
    - These classes are not related to the original classes in the class hierarchy
    - Duck Typing is used instead. To Obj-C programmers, you are never the wiser.
    - Under “clean” API usage, you would never notice in LuaCocoa either
        - But the NS*Panel APIs use Blocks APIs with non-object parameters so Bridge Support metadata is needed
        - But since these APIs are private/secret details, there is no Bridge Support metadata and since these classes are unrelated to the NS*Panel classes, there is no way to correlate them.

App Sandbox gotcha: Duck Typing

- Workaround: Explicitly define the block signature yourself with your own metadata.

```
function ShowOpenPanel(thewindow)
    local panel = NSOpenPanel:openPanel()
    panel:setCanChooseFiles_(true)
    panel:setAllowedFileTypes_({ "jpg", "bmp", "png" })

    local my_completion_function = function(returncode)
        if returncode == NSFileHandlingPanelOKButton then
            print("User hit OK")
        elseif returncode == NSFileHandlingPanelCancelButton then
            print("User hit cancel")
        else
            print("This code shouldn't be possible")
        end
    end

    local my_completion_block = LuaCocoa.toblock(my_completion_function,
[[<arg> <arg type='i' type64='q'/> <retval type='v'/> </arg>]])

    panel:beginSheetModalForWindow_completionHandler_(GetMainWindow(),
my_completion_block)
end
```

- (Could also create your own Bridge Support file.)

- This problem isn't actually unique to App Sandbox. Other APIs in Cocoa could do something like this.

Porting to iOS?

- Has become feasible
- BridgeSupport not on iOS but could generate metadata ourselves and ship in bundle
  - gen_bridge_metadata still on Mac and open source
  - dynamic libraries allowed for inline symbols?
- libffi iOS fork seems to have a workaround for mprotect PROT_EXEC
  - JSCocoa for iOS proves this works
- NSXMLDocument still not available
  - Had started experiment moving to TBXML

Future ideas

- iOS
- Lua 5.3
- Objective-Lua (David Given)
    - Uses LPEG+LEG to create superset of Lua with Obj-C/SmallTalk like syntax
- slua (offshoot of llvm-lua) (Robert G. Jakabosky)
    - Use compiler to generate static bindings
        - Use Objective-Lua or annotations to help compiler resolve ambiguities
    - Kind of like RubyMotion?
- Cross-platform native GUI in Lua, using the following for platform specific backends
    - LuaCocoa (Cocoa)
    - LuaInterface (C#)
    - LuaJava or JNLua or etc (Java)
- DTrace probes to show Lua script-level info instead of C level (see Ruby & Python)
- NSArray/NSDictionary subclass/class cluster implemented to share Lua table data instead of needing to copy/convert

Apple State of the Union

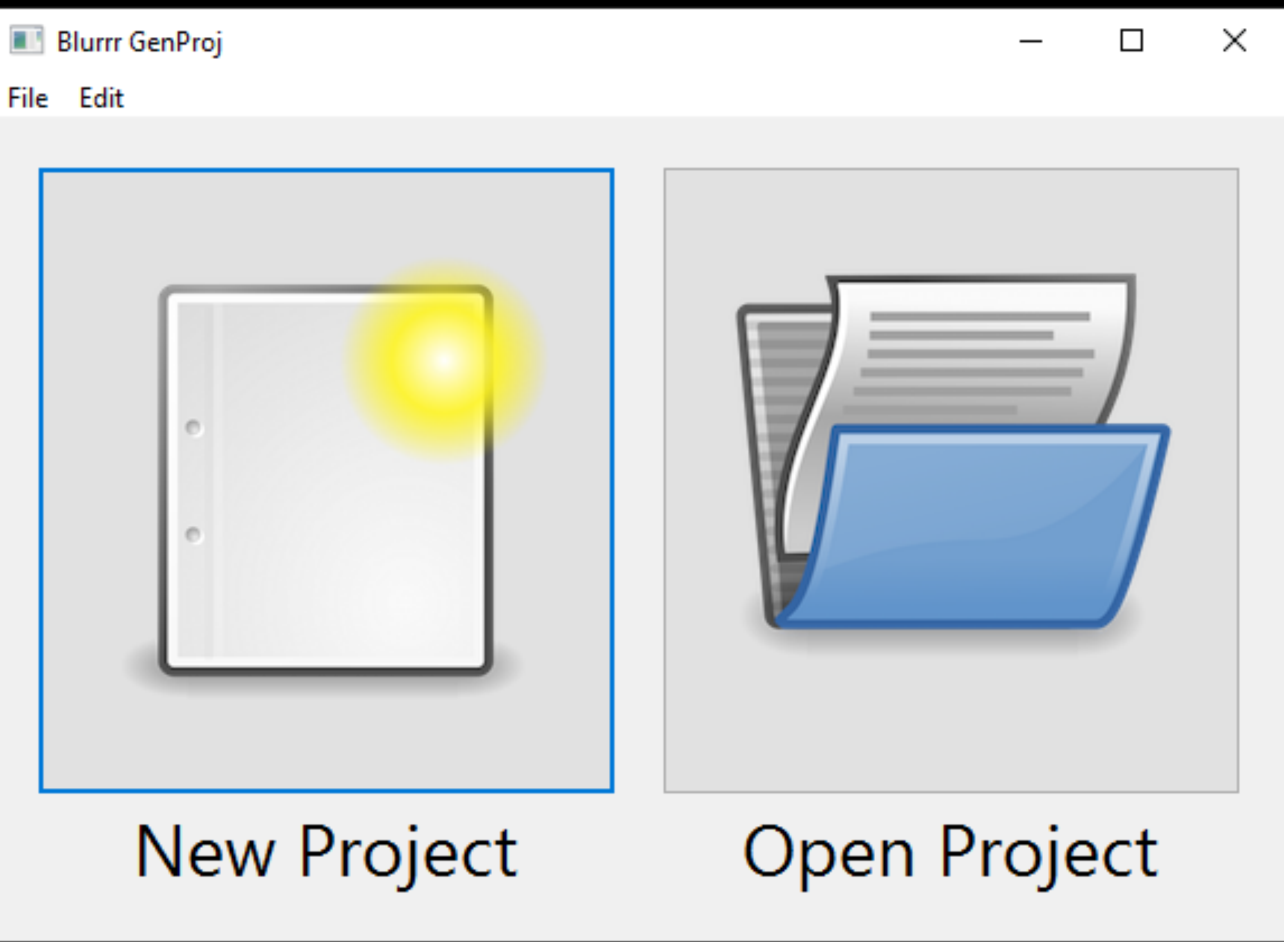
- iOS App Store (2008)
  - No BridgeSupport on iOS. 3rd party dynamic libraries (dlopen) & mprotect PROT_EXEC forbidden.
- Xcode 4 (2011)
  - Breaks most of AppleScript dictionary
    - Finally revisited in Xcode 8 (2016)
  - PyObjC & MacRuby Interface Builder integration broken/removed
- Mac App Store (2011) sandboxing disincentivizes app scripting
- Laurent Sansonetti leaves Apple to create RubyMotion (2011)
- Mac 10.9 (Mavericks 2013) adds official Obj-C API to JavaScriptCore
- Mac 10.10 (Yosemite 2014) officially adds JavaScript scripting support for “Mac Automation”
  - JSCocoa made redundant on Mac?
- iOS 8 (2014) dynamic linking made available for “App Extensions”
- Swift Language debut (2014)
  - Swift open sourced with initial Linux port (December 2015)
  - Swift 3.0 just released (fall 2016)

LuaCocoa is Sleeping

- Not dead
- Very personal to me so won’t go away
- But not a high priority due to the current environment
- Contributions still welcome
- Need stuff? Contact me

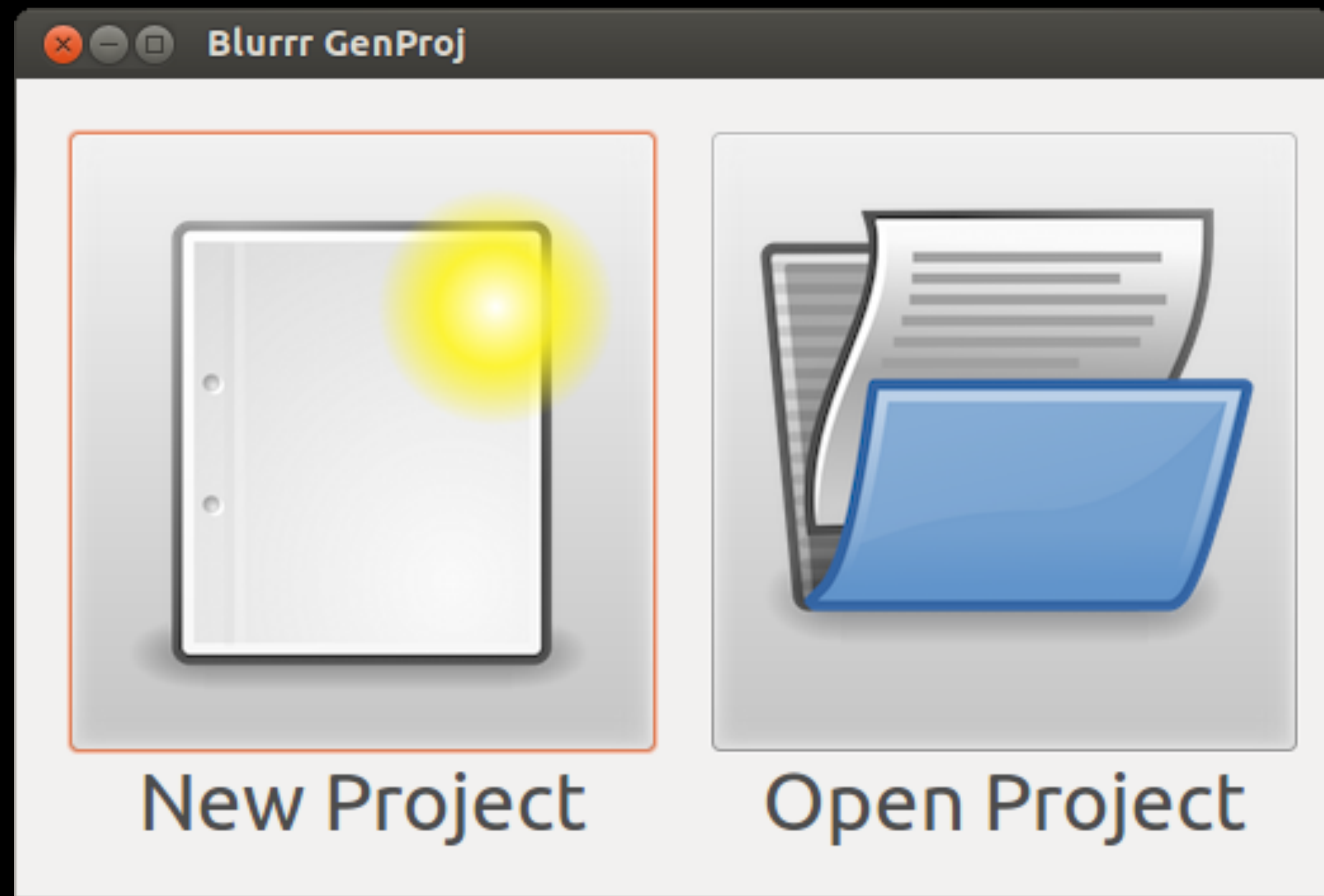
Off topic (Call for Help): IUP (cross-platform GUI library)

- IUP is a cross-platform native GUI library from Tecgraf/PUC-Rio (where Lua is from)
- Small, lightweight, native, fast
- Native Windows support
- GTK2, GTK3, Motif for Linux, etc
- But no Cocoa…

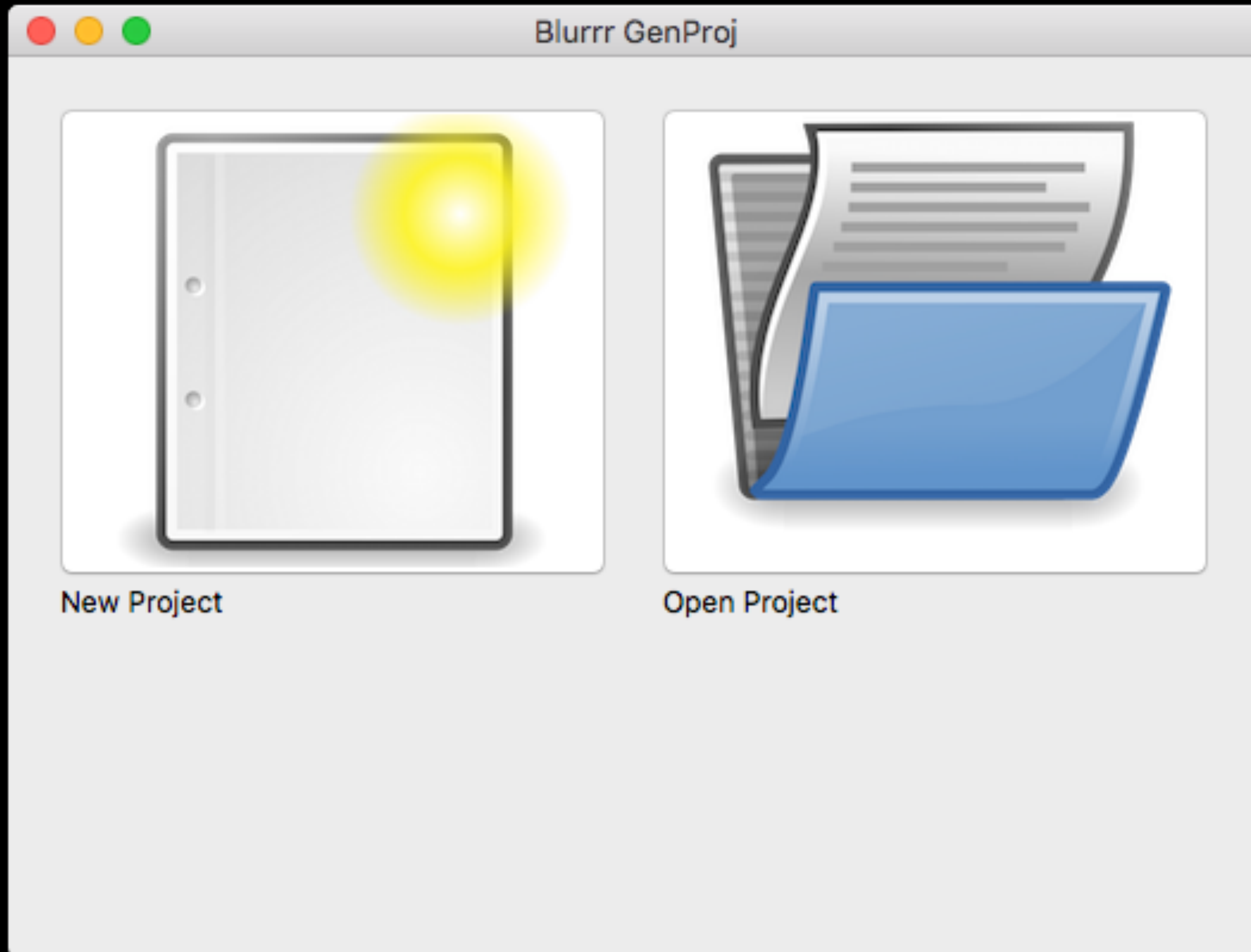


Windows

Linux



Mac
(IUP Cocoa)



Call to Arms
IUP for Cocoa

- We can make this happen!
- Stepping stone to iOS implementation
    - Then Android
- IUP abstraction may already be far enough to deal with mobile-isms
    - No “Window” type, but “Dialog”
- Lua bindings are first class citizens in IUP
- Pure C API means other languages can benefit too
- Looking for volunteers, funding, or companies to sponsor or drive

Links

- LuaCocoa
  - http://playcontrol.net/opensource/LuaCocoa
- Eric Wing (@ewingfighter)
  - Website: http://playcontrol.net
  - YouTube: https://www.youtube.com/user/ewmailing
    - Now playing: "***Why we loved Sierra Games***"

An Opinionated Review

by a crotchety Sierra fan
who remembers 1984

- Blurrr SDK (my current project)
  - https://BlurrrSDK.com, @BlurrrSDK
- IUP Cocoa (use Cocoa branch)
  - https://github.com/ewmailing/IupCocoa
- GIST Cancer Research Fund
  - http://www.gistinfo.org