Mobile Go, part 2: Building an iOS app with `go build`

Part 2 of my quest to learn Go for use in my iOS and Android app. Here's how to use Go in an iOS app.

Starting from first principles

It must be possible to build an iOS app with Go.

Why do I think this? Because after years of iOS development, I know that Objective C is really just C code with a small runtime that enables the language's bells & whistles. And the support goes both ways: while the Objective C compiler supports calling C functions without any special configuration, many people don't know that you can also call Objective C methods from C code:

```
// Objective-C example
void createStringInObjectiveC() {
    [[NSString alloc] init];
}


// C equivalent
void createStringInC() {
    void* cls = objc_getClass("NSString");
    void* obj = objc_msgSend(cls,
NSSelectorFromString(CFSTR("alloc")));
    obj = objc_msgSend(obj, NSSelectorFromString(CFSTR("init")));
}
```

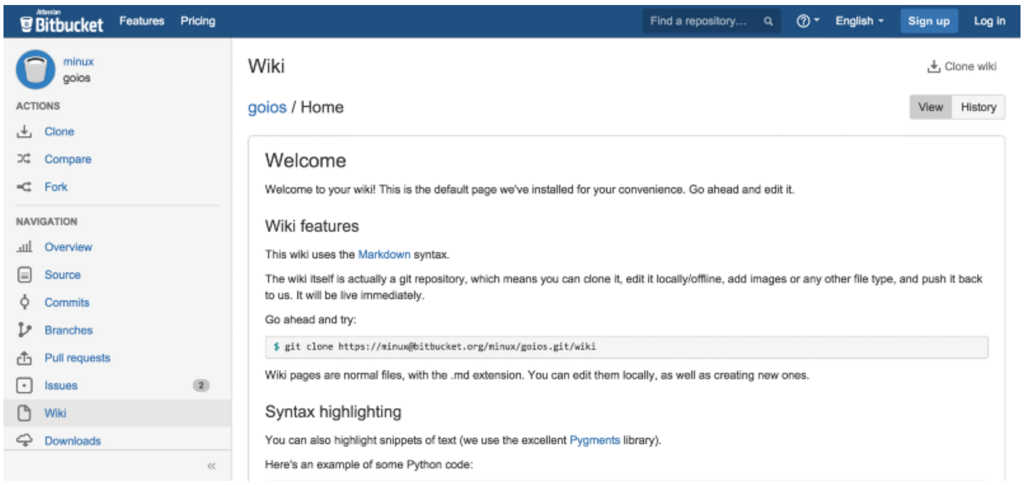
It's not pretty, but it just goes to show that at heart, Objective C is not radically different from C.

What about Go? Well, we can also see that Go is really just C code with a medium-sized runtime. Granted, no one has yet developed a Go→C transpiler, but can see that because it's compiled to machine code rather than byte code, it must be more or less compatible with C. And as I mentioned in Part 1 of this series, Go interoperates with C in both directions via the *cgo* part of the Go toolchain.

So if both Go and Objective C offer two-way calling into and out of C, it's only logical to assume that we can call between the two languages, using C as a middle man. Right??

Diving down the rabbit hole

A quick Google search for "build Go on iOS" pretty much only has one relevant result, a StackOverflow post entitled Go language on iPhone. The top answer refers to an iOS Go port being maintained by someone named Minux.



Wiki

goios / Home

Welcome

Welcome to your wiki! This is the default page we've installed for your convenience. Go ahead and edit it.

Wiki features

This wiki uses the Markdown syntax.

The wiki itself is actually a git repository, which means you can clone it, edit it locally/offline, add images or any other file type, and push it back to us. It will be live immediately.

Go ahead and try:

```
$ git clone https://minux@bitbucket.org/minux/goios.git/wiki
```

Wiki pages are normal files, with the .md extension. You can edit them locally, as well as creating new ones.

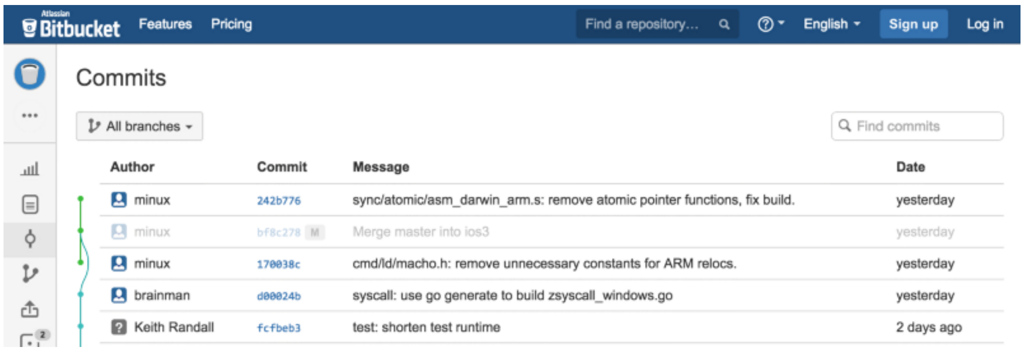
Syntax highlighting

You can also highlight snippets of text (we use the excellent Pygments library).

Here's an example of some Python code:

https://bitbucket.org/minux/goios/wiki/Home

Well, I appreciate the explanation of the Wiki features, but I'm guessing this is not a wildly popular, well-supported project, with a vibrant community☺



Commits

Author	Commit	Message	Date
minux	242b776	sync/atomic/asm_darwin_arm.s: remove atomic pointer functions, fix build.	yesterday
minux	bf8c278 M	Merge master into ios3	yesterday
minux	170038c	cmd/ld/macho.h: remove unnecessary constants for ARM relocs.	yesterday
brainman	d00024b	syscall: use go generate to build zsyscall_windows.go	yesterday
Keith Randall	fcfbeb3	test: shorten test runtime	2 days ago

Luckily, it looks like it *is* being actively maintained, if by only one person. After a little digging, I found that the author recently posted an announcement on the Go mailing list in November 2014:

> *Hi gophers,*

> *I'm happy to announce that the iOS port of Go recently gained full cgo and external linking support. I've also got a simple C based application working on a real iPad…*

> *I intend to propose for inclusion into Go 1.5… I believe it really will happen.*

> *The iOS port is my first Go OS port, but it's the only one that takes almost 3 years to complete.* ☺

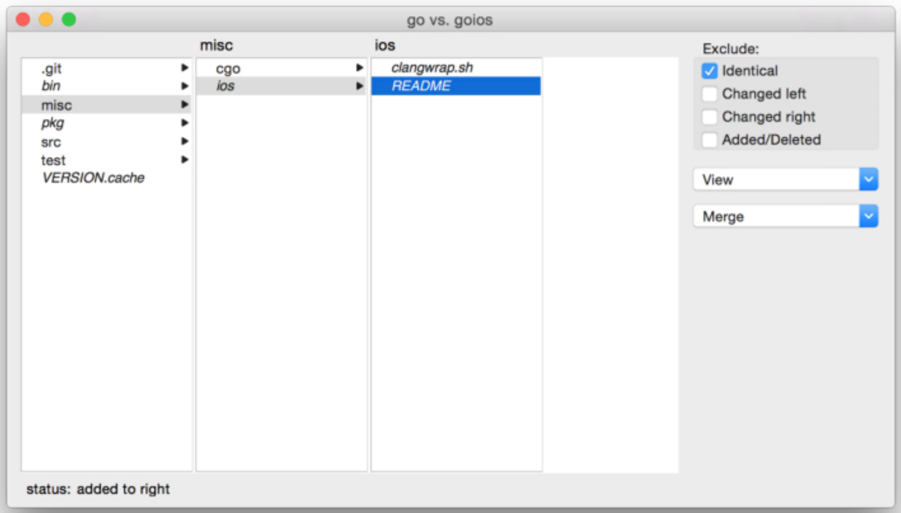
> *Go build iOS Apps. Let's see when can we see a Go based App in the store.*

> *Minux*

So exciting! I guess I picked an opportune time to start this project.

Building the Go-iOS toolchain

After checking out the goios repo, I used FileMerge to find which files Minux had changed. I'm sure there's a way to do this with Git or SourceTree, but I didn't care about change history, just which files were different.



status: added to right

A-ha! There's a file called *ios/README*! Let's see what it says.

```
To build a cross compiling toolchain for iOS on OS X, first
modify clangwrap.sh in misc/ios to match your setup. And then
run:
```

```
GOARM=7 CGO_ENABLED=1 GOARCH=arm
CC_FOR_TARGET=`pwd`/../misc/ios/clangwrap.sh \
 CXX_FOR_TARGET=`pwd`/../misc/ios/clangwrap.sh ./make.bash


To build a program, use the normal go build command:


CGO_ENABLED=1 GOARCH=arm go build import/path
```

Well this seems pretty easy. I don’t totally understand what clangwrap.sh is for, but I understand that we need to cross-compile to the target machine type —iOS devices run on ARM, not x86 like my laptop does. So what’s in clangwrap.sh, and what do we need to modify?

```
#!/bin/sh
# This uses the latest available iOS SDK, which is recommended.
# To select a specific SDK, run ‘xcodebuild -showsdks’
# to see the available SDKs and replace iphoneos with one of
them.
SDK=iphoneos
SDK_PATH=`xcrun -sdk $SDK -show-sdk-path`
export IPHONEOS_DEPLOYMENT_TARGET=7.0
# cmd/cgo doesn’t support llvm-gcc-4.2, so we have to use clang.
CLANG=`xcrun -sdk $SDK -find clang`
exec “$CLANG” -arch armv7 -isysroot “$SDK_PATH” “$@”
```

Let’s just try those instructions he gave, without modifying anything:

```
$ cd src
$ GOARM=7 CGO_ENABLED=1 GOARCH=arm \
    CC_FOR_TARGET=`pwd`/../misc/ios/clangwrap.sh \
    CXX_FOR_TARGET=`pwd`/../misc/ios/clangwrap.sh \
    ./make.bash
##### Building C bootstrap tool.
cmd/dist


##### Building compilers and Go bootstrap tool for host,
darwin/amd64.
lib9
libbio
liblink
cmd/gc
...
```

Looking good… Uh oh, some errors are starting to pop up:

```
cmd/pprof/internal/commands
# cmd/yacc
exec ldid: No such file or directory
# cmd/cgo
exec ldid: No such file or directory
cmd/pprof/internal/driver
crypto/x509
net/textproto
log/syslog
mime/multipart
net/mail
# crypto/x509
crypto/x509/root_cgo_darwin.go:46:37: error: use of undeclared
identifier ‘kSecFormatX509Cert’
 err = SecKeychainItemExport(cert, kSecFormatX509Cert,
kSecItemPemArmour, NULL, &data);
 ^
1 error generated.
```

Well, I remember something from the *README* file:

> ** crypto/x509 won’t build, I don’t yet know how to get system root on iOS.*

> ** Because I still want to be able to do native build, CGO_ENABLED=1 is not the default, yet.*

> ** You can safely ignore the “exec ldid: No such file or directory” error message. I will remove the automatic “ldis -S” support when upstreaming the code. Its only purpose is for native development on (jailbroken) iOS.*

Okay, I don’t know what #2 means, but #1 and # mean we can safely ignore all those errors! Now I have my go binary installed at *../bin/go*. Great!

Preparing to build an iOS app

I think it’d be best to start small, and fail fast. So, my first iOS Go app will simply display “1+1=2” on the screen. That’s it?? Well, the “2″ will be computed by calling into a Go function from Objective C:

```
func add(a int, b int) int {
   return a + b
}
```

Minux provided an example of how to do this a bit later on his announcement thread:

> *You can rename the main function of the original app to iosmain, rearranging the files into a single directory, and adding a new main.go:*

```
package main


/*
// adjust LDFLAGS if necessary
#cgo LDFLAGS: -framework UIKit -framework Foundation -framework
CoreGraphics
extern void iosmain(int argc, char *argv[]);
*/
import "C"

func main() {
   C.iosmain(0, nil)
}
```

I actually emailed the author for more clarity on building this, and he suggested the following:

> *Build your app binary with:*

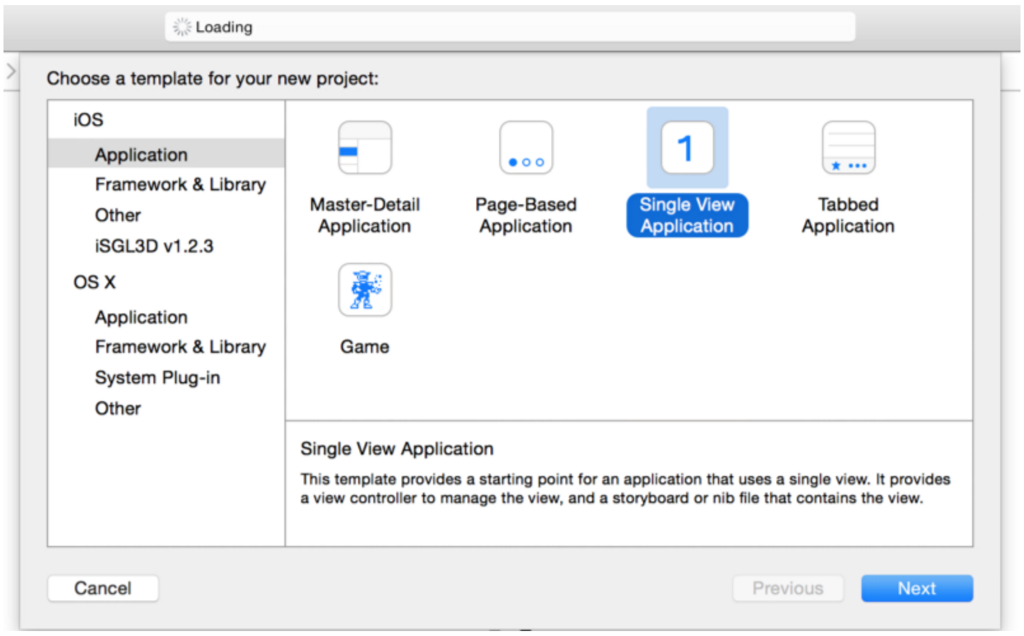
> ***GOARM=7 CGO_ENABLED=1 GOARCH=arm go build***

> *(Note, building apps requires cross compiling on OS X because it requires the whole iPhoneOS SDK.)*

This all looks great and I think I'm ready to actually do it!

Creating the Xcode project and building from the command line

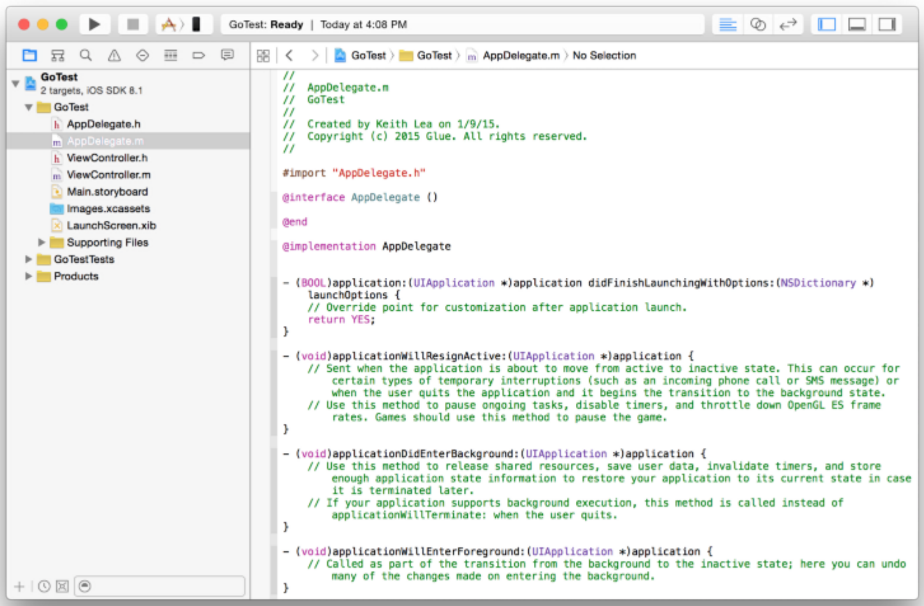
Okay, let's just start by creating a normal iOS app.



Single View Application

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view, and a storyboard or nib file that contains the view.

Blah blah blah… Okay great:

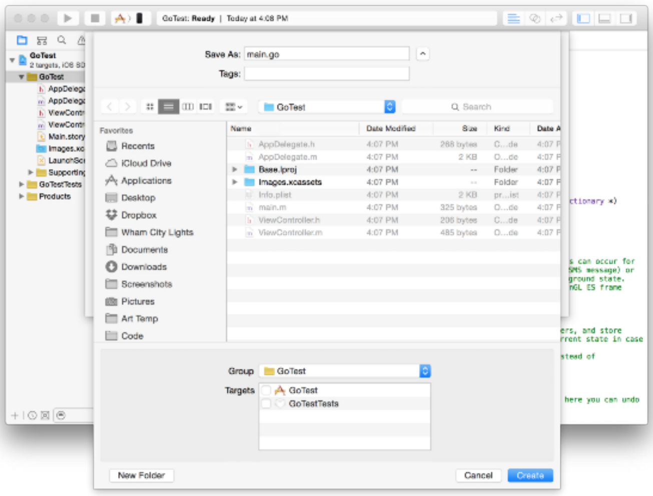


```
//
//  AppDelegate.m
//  GoTest
//
//  Created by Keith Lea on 1/9/15.
//  Copyright (c) 2015 Glue. All rights reserved.
//

#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)
    launchOptions {
    // Override point for customization after application launch.
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can occur for
        certain types of temporary interruptions (such as an incoming phone call or SMS message) or
        when the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame
        rates. Games should use this method to pause the game.
}

- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and store
        enough application state information to restore your application to its current state in case
        it is terminated later.
    // If your application supports background execution, this method is called instead of
        applicationWillTerminate: when the user quits.
}

- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the inactive state; here you can undo
        many of the changes made on entering the background.
}
```

I'm just gonna dive right in here and try compiling this with my shiny new iOS cross-compiling version of *go build:*

```
$ export PATH=/Users/keith/…/goios/bin:$PATH
$ cd GoTest
$ GOARM=7 CGO_ENABLED=1 GOARCH=arm go build
can't load package: package .: no buildable Go source files in
/Users/keith/…/GoTest
```

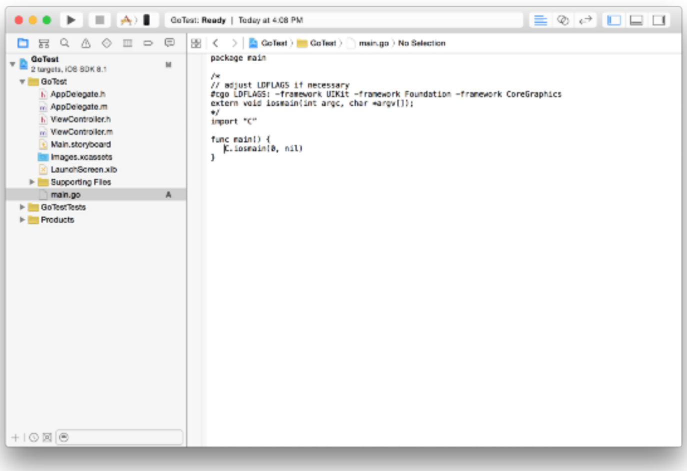
Well duh, so let's create that Go file that Minux recommended earlier!



Save As: main.go

Group: GoTest

Targets: GoTest, GoTestTests



```
package main

/*
// adjust LDFLAGS if necessary
#cgo LDFLAGS: -framework UIKit -framework Foundation -framework CoreGraphics
extern void iosmain(int argc, char *argv[]);
*/
import "C"

func main() {
    C.iosmain(0, nil)
}
```

Someone should file a Apple Radar ticket for Go syntax highlighting ☺️

Let’s save and try again!

```
$ GOARM=7 CGO_ENABLED=1 GOARCH=arm go build
# _/Users/keith/…/GoTest
Undefined symbols for architecture armv7:
 “_iosmain”, referenced from:
 __cgo_d4b2332342b4_Cfunc_iosmain in main.cgo2.o
 (maybe you meant: __cgo_d4b2332342b4_Cfunc_iosmain)
ld: symbol(s) not found for architecture armv7
clang: error: linker command failed with exit code 1 (use -v to
see invocation)
```

We’re getting so close! It compiled and attempted to link! Now we have to do that trick Minux mentioned: renaming our Xcode-generated main() function to iosmain():

```
//
// main.m
// GoTest
//
// Created by Keith Lea on 1/9/15.
// Copyright (c) 2015 Keith Lea. All rights reserved.
//

#import <UIKit/UIKit.h>
#import “AppDelegate.h”
```

```
int iosmain(int argc, char * argv[]) {
   @autoreleasepool {
      return UIApplicationMain(argc, argv, nil,
              NSStringFromClass([AppDelegate class]));
   }
}
```

And let's try again!

```
$ GOARM=7 CGO_ENABLED=1 GOARCH=arm go build
# _/Users/keith/.../GoTest
exec ldid: No such file or directory
```

That *ldid* error is okay—Minux's README told us to ignore it. But… what happens now? I assume Go generated a binary—yep, it's called *GoTest*, as expected. Let's try running it just for fun.

```
$ ./GoTest
-bash: ./GoTest: Bad CPU type in executable
```

Great—that makes sense—it shouldn't run on my machine because it was built for an ARM chip. But… how am I supposed to launch this on my iPhone?

Go vs. static linking vs. dynamic linking

So I've built a Go binary for armv7—but how do I run it on my phone? I'm pretty sure can't just start any old executable on iOS—after all, how would you invoke it? The executable needs to be packaged up as an iOS app bundle. I've always looked at Xcode's app packaging process as a magical black box, and I have been content never needing to touch it. Well, I knew making a Go mobile app was going to be a long, treacherous journey. So let's take a deep breath and find a way to get this binary running on a phone—hopefully using Xcode's built-in build process as much as possible.

Xcode's build system seems complex—the fact that it has its own command line tool, *xcodebuild*, speaks to this. To minimize mucking around with Apple's tools, it would be *great* if we could use *go build* for the Go code, but allow Xcode to compile the rest—and link it all together into an app binary as usual. This idea seemed great for another reason: I'd love to be able to build Go libraries that can be included in other people's mobile apps, without having to dramatically alter their build process.

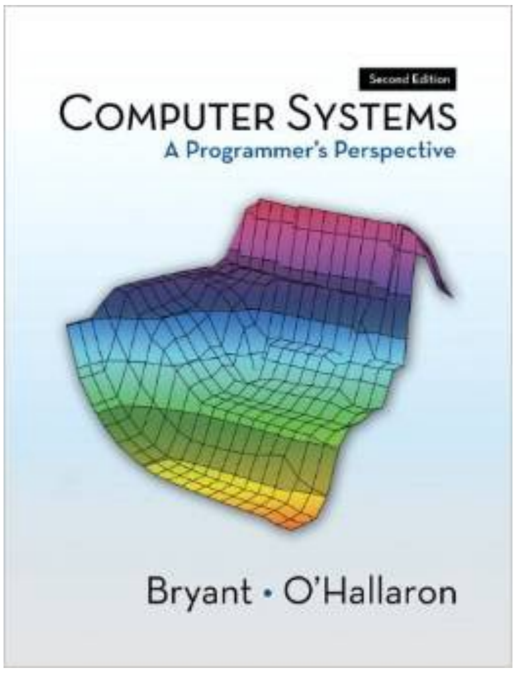
But this leads to a problem we've been dancing around for this whole blog series—how to build Go code as a library? I've built pure C++ libraries commercially before, and distributing it to clients was easy—I just sent over a fat static library (*.a) for iOS, and some architecture-specific dynamic libraries (*.so) for Android. This worked great, but it's not going to be as easy here, for a few reasons:

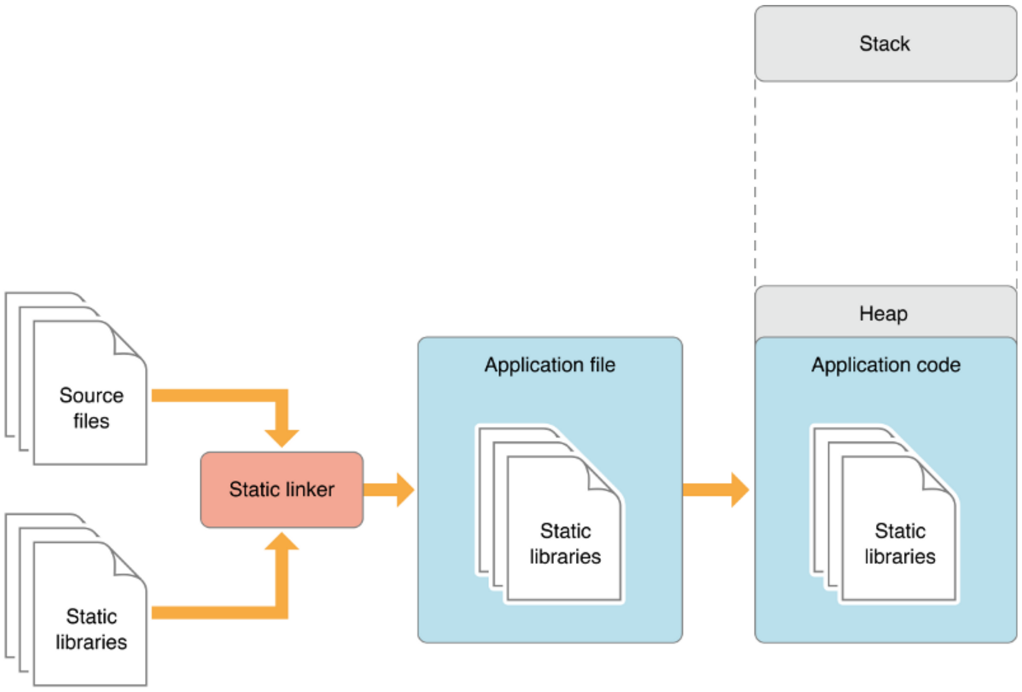
1. Go cannot build static libraries
2. Go cannot build dynamic libraries
3. iOS 7 and prior cannot load dynamic libraries

It was at this point that I started realizing I need to know what linkers do, and what the difference between static and dynamic linking is. As a Java developer I never had to really understand how this stuff worked. I vaguely understood that Java code was "dynamically linked," and I only knew this because of the LGPL Java controversy of the early 2000's.

So at this point I asked my dad, who at one point was really into C++, before he switched gears and began focusing on Java concurrency. He recommended a book, but said at this point in history I may get more information from Wikipedia.

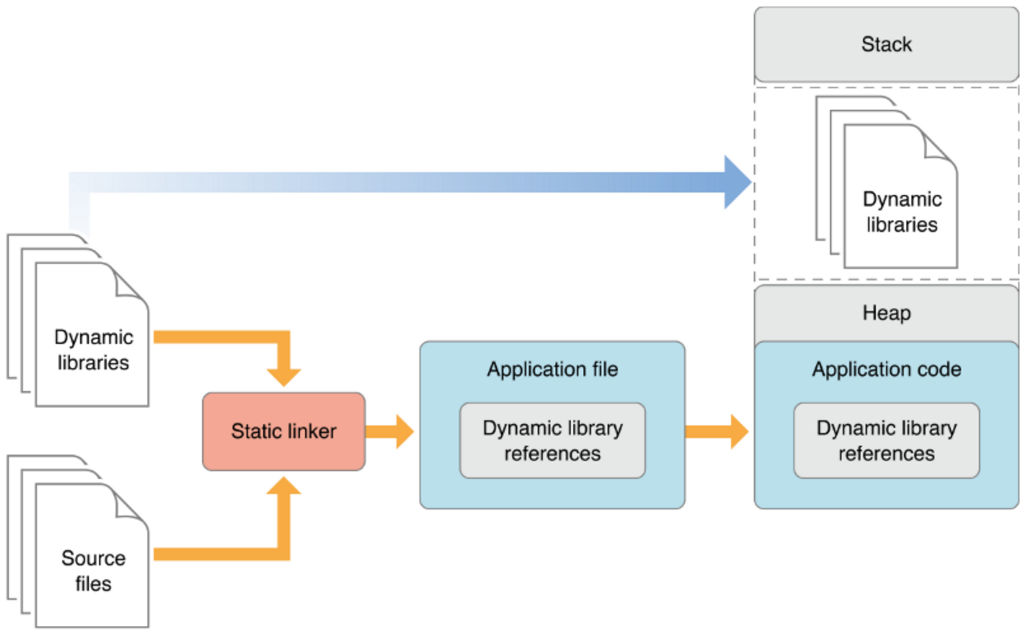
The Wikipedia page was pretty sparse, but after some Googling, I found an even better source of information: Apple's own Mac developer documentation. Look at this great diagram!





Static linking, courtesy of Apple

. . .



Dynamic linking, courtesy of Apple

This all makes sense, but the combination of iOS's limitations and Go's limitations left me feeling stuck. So I contacted Michael Hudson-Doyle from Canonical, as I heard he's trying to build Go packages as dynamic libraries. He's doing this in order to make Linux packaging & distribution easier on Ubuntu. Michael wrote back quickly and was very helpful:

> *The external linking support in the Go toolchain's linker can already turn a bunch of Go code into a .o file, so all you need to do is put that in a .a file instead of an executable.*

> *There will be some other things, like persuading the linker to accept input that doesn't define a main() function, and probably some worries about symbol visibility (in particular, I don't know at all what would or should happen if you tried to link against two .a files that both contained a Go runtime).*

What a great idea! I actually remember seeing those **.o* files when I was running *cgo* earlier. And I remember from my C++ days that a static library (**.a*) is just a collection of **.o* files, which can be created with the Unix *ar* tool.

In the meantime, I had also written to Minux, developer of the Go iOS port, and he wrote back with similar advice:

> *try:*

> mkdir tmp
> go build -ldflags '-tmpdir ./tmp -linkmode external' import/path

> *under* ./tmp*, there should be multiple object files, numbered files is those compiled by gcc, and* go.o *is the result of Go code.*

> *Incorporate these into your project. The main function will still be controlled by* go.o*, but at least you can use Go code without go build building the whole project.*

This is all sounding pretty good. I don't *love* the idea of Go generating *main()*, but at least we've found a way to get Go to generate **.o* files, which I can then package up into a **.a* file! Yay!

One question remained—will this strategy work on Android? As I mentioned earlier, I'm comfortable distributing Android code as dynamic libraries (*.so), but it appears that Go can't build these. Well, after a bit of research, I'm hopeful, as Go 1.4 shipped in December 2014 with official Android support. And this situation is mentioned specifically in the release notes:

> *"Go 1.4 can build binaries for ARM processors running the Android OS. It can also build a .so library that can be loaded by an Android app."*

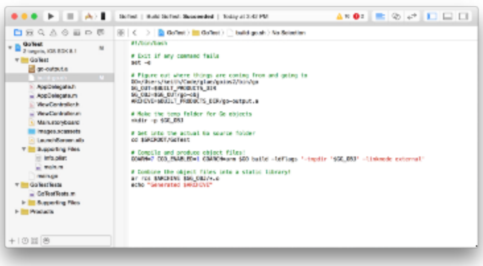
That sounds great—looks like we can keep plugging away at iOS, and figure out Android later.

Plugging Go into the Xcode build system

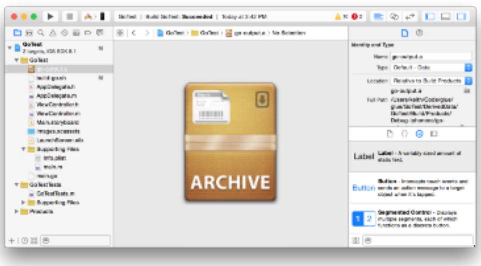
I think finally we have a plan. I'll create a Run Script build step in Xcode, so that the build looks like this:

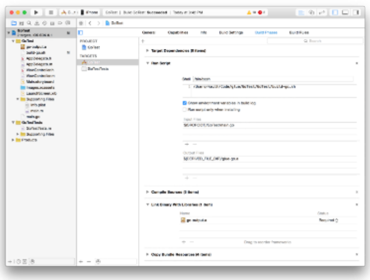
1. Run *go build* in order to compile the Go code and produce **.o* files containing compiled object code
2. Run *ar* to package the **.o* object files into *go-output.a* (a static library!)
3. Place *go-output.a* in Xcode's output folder
4. Xcode compiles my Objective-C and C++ code
5. When linking, Xcode statically links against *go-output.a*
6. Xcode produces the ARM binary
7. Xcode packages it up into an iOS app and deploys to my phone

Let's try it! First we need to set up the Xcode project and write the shell script.

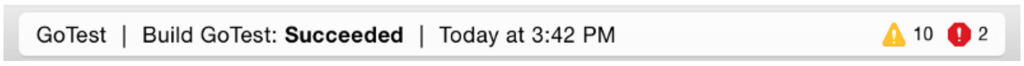


```
#!/bin/bash

# Exit if any command fails
set -e

# Figure out where things are coming from and going to
GO=/Users/keith/Code/glam/goios2/bin/go
GG_OUT=$BUILT_PRODUCTS_DIR
GG_OBJ=$GG_OUT/go-obj
ARCHIVE=$BUILT_PRODUCTS_DIR/go-output.a

# Make the temp folder for Go objects
mkdir -p $GG_OBJ

# Get into the actual Go source folder
cd $SRCROOT/GoTest

# Compile and produce object files!
GOARM=7 CGO_ENABLED=1 GOARCH=arm $GO build -ldflags '-tmpdir '$GG_OBJ' -linkmode external'

# Combine the object files into a static library!
ar rcs $ARCHIVE $GG_OBJ/*.o
echo "Generated $ARCHIVE"
```



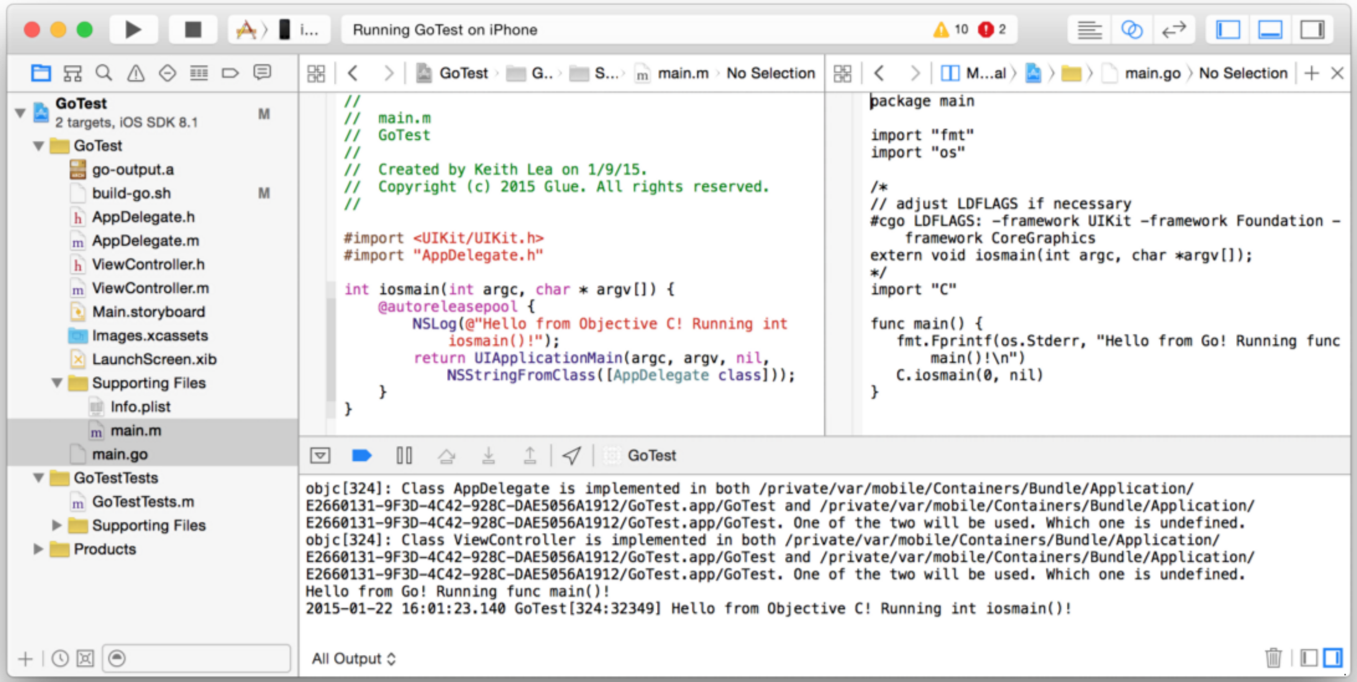


The moment of truth—let's try building and running on my iPhone!



GoTest | Build GoTest: **Succeeded** | Today at 3:42 PM ⚠ 10 ❗ 2

It works! The app does launch on my phone. But is Go really being loaded? Let's modify our source a little to make sure. Now Go will print a message to the Xcode console, and the Objective C code will do the same.



```
//
//  main.m
//  GoTest
//
//  Created by Keith Lea on 1/9/15.
//  Copyright (c) 2015 Glue. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "AppDelegate.h"

int iosmain(int argc, char * argv[]) {
    @autoreleasepool {
        NSLog(@"Hello from Objective C! Running int
            iosmain()!");
        return UIApplicationMain(argc, argv, nil,
            NSStringFromClass([AppDelegate class]));
    }
}
```

```
package main

import "fmt"
import "os"

/*
// adjust LDFLAGS if necessary
#cgo LDFLAGS: -framework UIKit -framework Foundation -
    framework CoreGraphics
extern void iosmain(int argc, char *argv[]);
*/
import "C"

func main() {
   fmt.Fprintf(os.Stderr, "Hello from Go! Running func
       main()!\n")
   C.iosmain(0, nil)
}
```

```
objc[324]: Class AppDelegate is implemented in both /private/var/mobile/Containers/Bundle/Application/
E2660131-9F3D-4C42-928C-DAE5056A1912/GoTest.app/GoTest and /private/var/mobile/Containers/Bundle/Application/
E2660131-9F3D-4C42-928C-DAE5056A1912/GoTest.app/GoTest. One of the two will be used. Which one is undefined.
objc[324]: Class ViewController is implemented in both /private/var/mobile/Containers/Bundle/Application/
E2660131-9F3D-4C42-928C-DAE5056A1912/GoTest.app/GoTest and /private/var/mobile/Containers/Bundle/Application/
E2660131-9F3D-4C42-928C-DAE5056A1912/GoTest.app/GoTest. One of the two will be used. Which one is undefined.
Hello from Go! Running func main()!
2015-01-22 16:01:23.140 GoTest[324:32349] Hello from Objective C! Running int iosmain()!
```

Finally! Success!

Conclusions

You can indeed build an iOS app using the Go toolchain, without throwing out the baby with the bathwater—i.e., without abandoning Xcode's otherwise great build system.

- You can build Go code for iOS and ARM7 using an actively maintained fork of the Go toolchain. This code may be integrated into mainline Go, in version 1.5, which is due mid-2015.
- Building Go code as a library is the best option for me (but YMMV).
- Even though iOS 8 and Android support loading dynamic libraries (*.so), supporting iOS 7 and earlier requires using static libraries, especially if you want to distribute libraries for use in other apps.
- Go's toolchain *can* be modified to build a static library—the only caveat is that your app's *main()* function still needs to be written in Go. While

it's strange to have your application's *main()* implemented in a library, it's easy to deal with.

- Xcode happily links Go-generated object files alongside its own build output—and produces an executable that contains Go code which runs on a real iPhone!

Stay tuned for the next installment of this blog. Some issues are still yet unresolved, including:

1. Calling into Go code from Objective-C/C++
2. Calling Objective-C code from Go
3. Building 64-bit binaries (which is now required on iOS)
4. Building on Android