Mobile Go, Part 4: Calling Objective C from Go

Part 4 of my quest to learn Go for use in my iOS and Android app. Here's how to call Objective C methods from your Go code.

Triggering a dialog box via HTTP request

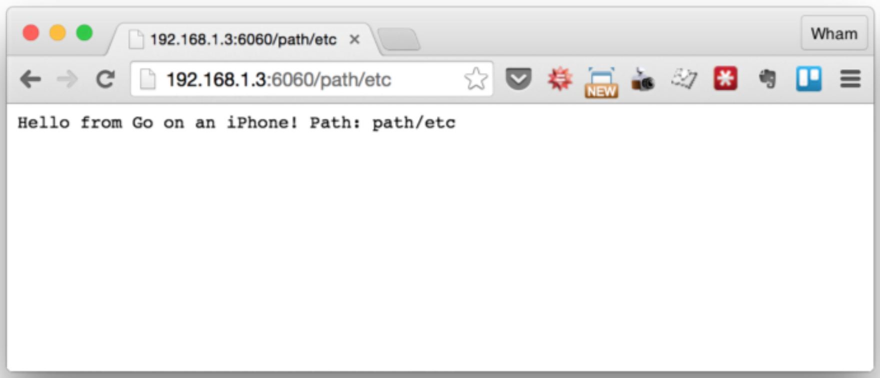
I need a good test case for calling from Go into Objective C. My first idea is to use Go's *net/http/Server* web server to trigger an alert box on my iPhone. Then I can use my computer's web browser to remotely pop up a dialog box on my phone. It's not a super realistic example, but it's a great chance to figure out how to call Objective C code from Go.

Starting the Go web server

A neat feature of Go is that it comes with a built-in web server that seems pretty full-featured and very easy to use. You can start it with a few lines of code:

```
package main

import “fmt”
import “os”
import “net/http”

/*
#cgo LDFLAGS: -Wl,-U,_iosmain
extern void iosmain(int argc, char *argv[]);
*/
import “C”

//export AddUsingGo
func AddUsingGo(a int, b int) int {
   return a + b
}

func HandleHttpRequest(w http.ResponseWriter, r *http.Request) {
   fmt.Fprintf(w, “Hello from Go on an iPhone! Path: %s”,
               r.URL.Path[1:])
}

func StartGoServer() {
   fmt.Fprintf(os.Stderr, “Starting net/http/Server...\n”)
   http.HandleFunc(“/”, HandleHttpRequest)
   http.ListenAndServe(“:6060", nil)
}

func main() {
   fmt.Fprintf(os.Stderr, “Hello from Go! I’m in func main()!\n”)
   go StartGoServer()
   C.iosmain(0, nil)
}
```

And now when I visit *http://<my phone's IP address>:6060* I see the following:



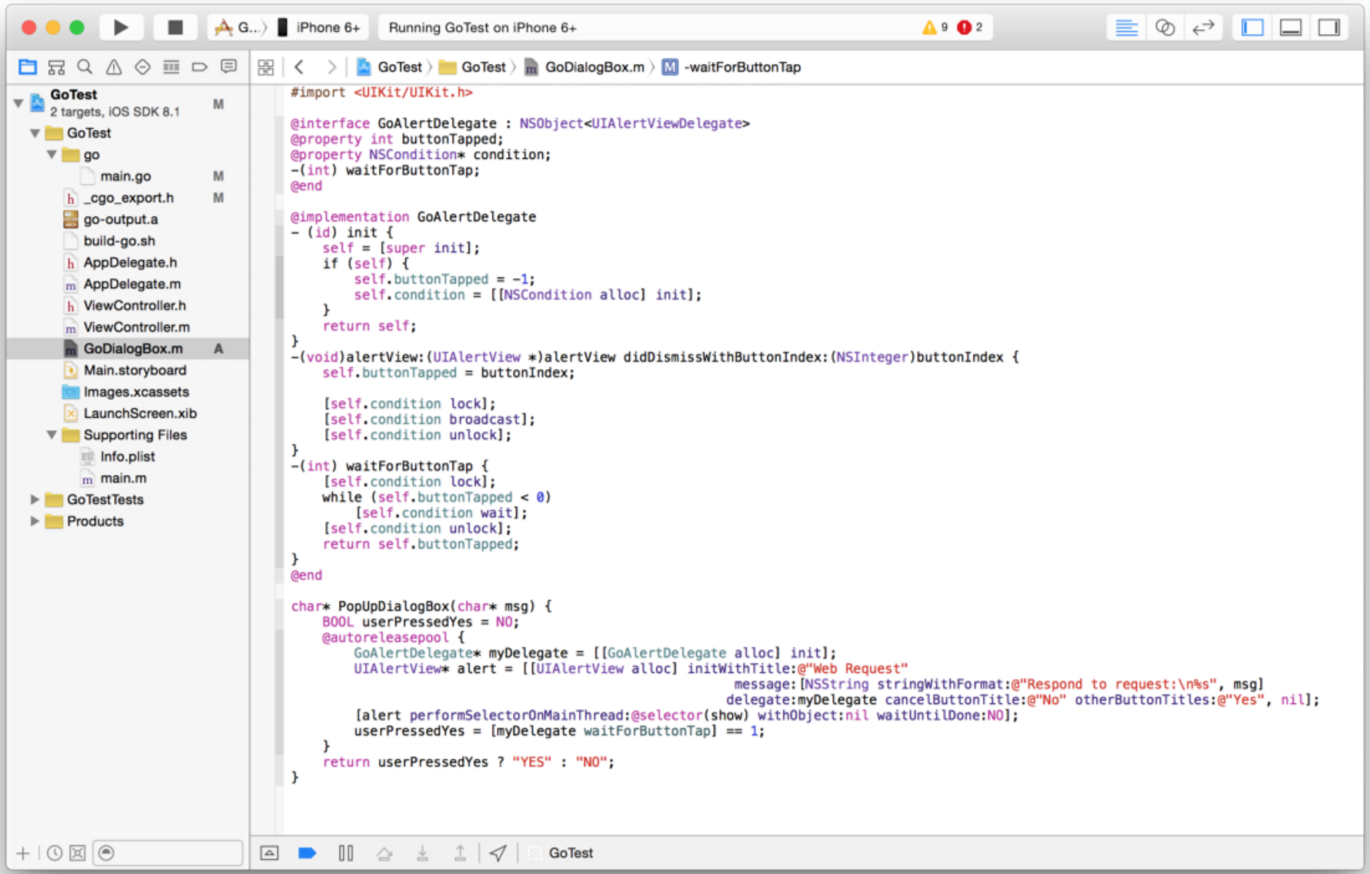
```
Hello from Go on an iPhone! Path: path/etc
```

Wow! That was easy!

Popping up a dialog box, the easy way

Now I want a dialog box to show up when I receive the request. There's an easy way to do this and a hard way.

I'd like to start with the easy way. I think we can simply define a C function called *char* PopUpDialogBox(char* message)*, and implement it in Objective-C. Unfortunately this immediately gets a little tricky due to threading:



```
#import <UIKit/UIKit.h>

@interface GoAlertDelegate : NSObject<UIAlertViewDelegate>
@property int buttonTapped;
@property NSCondition* condition;
-(int) waitForButtonTap;
@end

@implementation GoAlertDelegate
- (id) init {
    self = [super init];
    if (self) {
        self.buttonTapped = -1;
        self.condition = [[NSCondition alloc] init];
    }
    return self;
}
-(void)alertView:(UIAlertView *)alertView didDismissWithButtonIndex:(NSInteger)buttonIndex {
    self.buttonTapped = buttonIndex;

    [self.condition lock];
    [self.condition broadcast];
    [self.condition unlock];
}
-(int) waitForButtonTap {
    [self.condition lock];
    while (self.buttonTapped < 0)
        [self.condition wait];
    [self.condition unlock];
    return self.buttonTapped;
}
@end

char* PopUpDialogBox(char* msg) {
    BOOL userPressedYes = NO;
    @autoreleasepool {
        GoAlertDelegate* myDelegate = [[GoAlertDelegate alloc] init];
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Web Request"
                                                        message:[NSString stringWithFormat:@"Respond to request:\n%s", msg]
                                                       delegate:myDelegate cancelButtonTitle:@"No" otherButtonTitles:@"Yes", nil];
        [alert performSelectorOnMainThread:@selector(show) withObject:nil waitUntilDone:NO];
        userPressedYes = [myDelegate waitForButtonTap] == 1;
    }
    return userPressedYes ? "YES" : "NO";
}
```

You can see here that, when receiving an HTTP request, we:

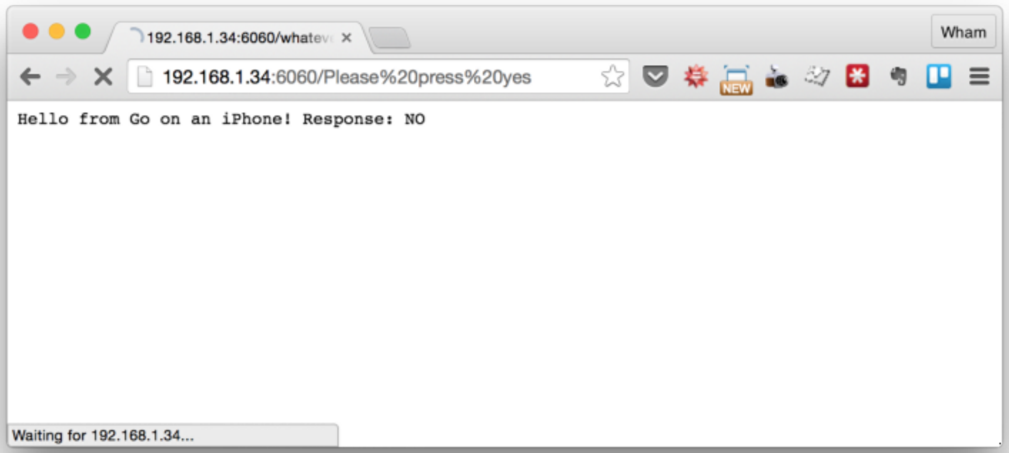
1. Start a temporary autorelease pool for Objective C's Automatic Reference Counting (ARC) runtime
2. Create the alert dialog, with text from the requested URL itself
3. Show the alert on the main iOS run loop
4. Wait for the user to press a button
5. Return a value based on which button the user pressed

Now we just need to call this from our Go code:

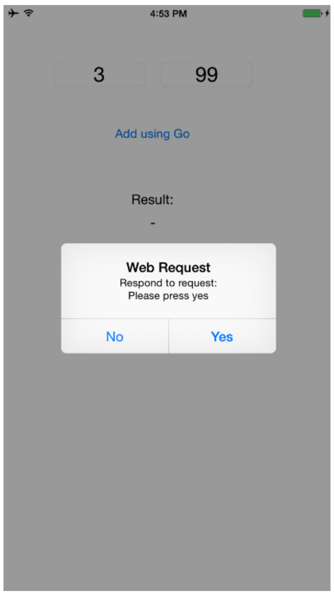
```
package main
```

```
import “fmt”
import “os”
import “net/http”


/*
#cgo LDFLAGS: -Wl,-U,_iosmain,-U,_PopUpDialogBox
extern void iosmain(int argc, char *argv[]);
extern char* PopUpDialogBox(char* msg);
*/
import “C”


//export AddUsingGo
func AddUsingGo(a int, b int) int {
   return a + b
}


func HandleHttpRequest(w http.ResponseWriter, r *http.Request) {
   userInput :=
C.GoString(C.PopUpDialogBox(C.CString(r.URL.Path[1:])))
   fmt.Fprintf(w, “Hello from Go on an iPhone! Response: %s”,
userInput)
}


func StartGoServer() {
   fmt.Fprintf(os.Stderr, “Starting net/http/Server…\n”)
   http.HandleFunc(“/”, HandleHttpRequest)
   http.ListenAndServe(“:6060", nil)
}


func main() {
   fmt.Fprintf(os.Stderr, “Hello from Go! I’m in func main()!\n”)
   go StartGoServer()
   C.iosmain(0, nil)
}
```

Let’s try it!


192.168.1.34:6060/whatev
Wham
192.168.1.34:6060/Please%20press%20yes
Waiting for 192.168.1.34...

```
Hello from Go on an iPhone! Response: NO
```



Wow! It really works! It's a hacky solution for a lot of reasons, but it does the job. In the spirit of agile, I'm going to mark this Story as Delivered, and leave a better solution for another day ☺

Conclusion

Calling from Go into Objective C was actually quite easy, as we've already do it for *iosmain()*. Our main hurdle was simply the clunkiness of Cocoa threading itself.

In the next part in this series, we'll talk about taking a more serious and scalable approach—allowing our Objective C code to register a delegate with the Go server. Then when a HTTP request is received on that URL, the Objective C delegate is called to take action and provide a response.