Building a (Core) Foundation

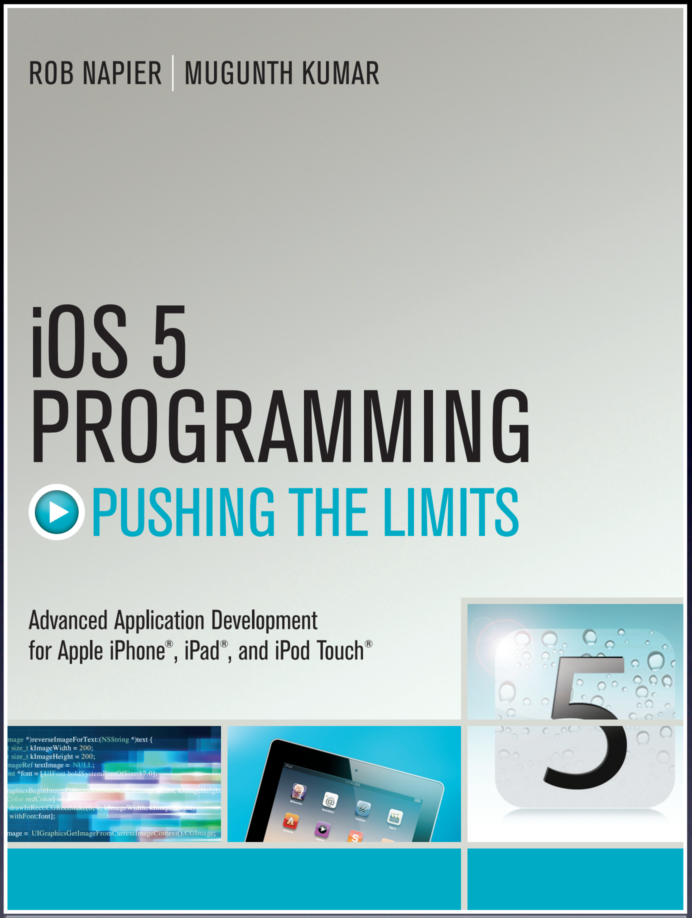
Rob Napier

A little background

- Mac OS X since 10.4
- iPhoneOS since release
- Cisco Jabber, The Daily, RNCryptor
- Focus on low-level
- Today: Mac developer for...

DELL

DELL KACE

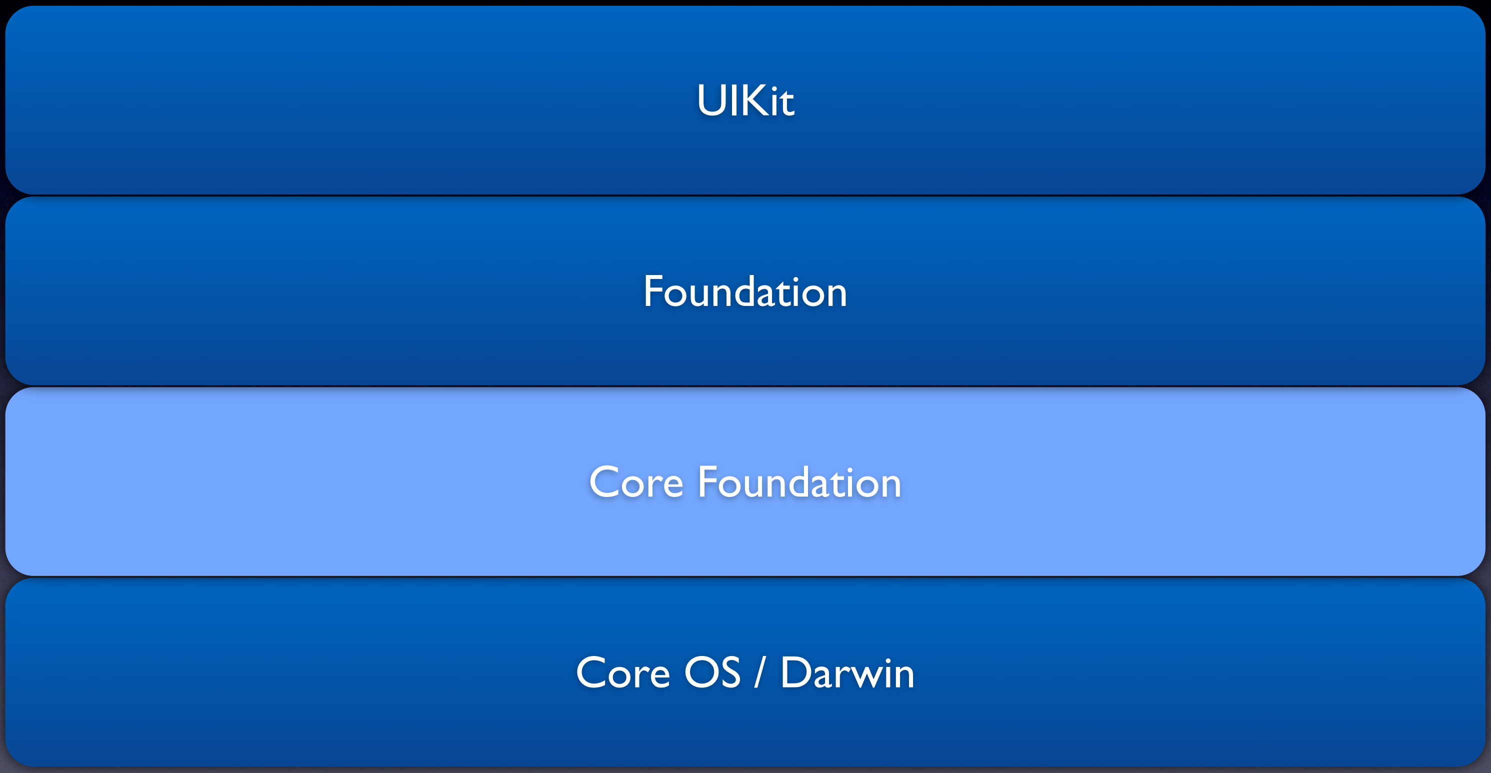

ROB NAPIER | MUGUNTH KUMAR
iOS 5 PROGRAMMING
PUSHING THE LIMITS
Advanced Application Development for Apple iPhone®, iPad®, and iPod Touch®

Chapter 19

Core Foundation?

- Data structures for all those powerful frameworks with “Core” in their name.
- The awesomeness of Objective-C. The speed of C.

Where are we?*



*Simplified, but close enough

Who cares?

- You want to use those powerful frameworks, right?
- Did I mention, it can do a lot of things Cocoa can't?
- And C is fast. Yes, very fast.

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

Real Types

```
typedef void * CFTypeRef;

typedef const struct __CFString * CFStringRef;

typedef struct __CFString * CFMutableStringRef;
```

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

Memory Management

- If you `Create` or `Copy` an object, you are an owner
- If you do not `Create` or `Copy` an object, you are not an owner.
- If you want to prevent the object from being destroyed, you must become an owner by calling `CFRetain()`
- If you are an owner of an object, you must call `CFRelease()` when you are done with it

`CFRelease != -release`

- `CFRelease` makes us cry
- `CFRelease(NULL)` crashes
- There are a dozen wrappers on `CFRelease()` that fix that
- You will certainly make your own
- And call them `SAFE_RELEASE()` like everyone else

autorelease?

- There is no autorelease

:(

Allocators

- How you want your memory?
  `CFCreateBlah(`**`Allocator`**`, param, param)`
- 99.9% of the time you want `NULL`
- Sometimes you want `kCFAllocatorMalloc` for memory that was created with `malloc()`
- Occasionally you want `kCFAllocatorNull` to do nothing
- Everything else is incredibly obscure

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

Introspection

- `CFGetTypeID() <=> CF`*`Array`*`GetTypeID()`
- `CFCopyDescription()`
- `CFShow()`
- `CFShowStr()`

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

Strings

- Constants: `CFSTR()`

```
CFStringRef foo = CFSTR(“foo”);
```

- `CFStringCreateWithCString()`

Convert `CFStringRef` to cstring

```
char * MYCFStringCopyUTF8String(CFStringRef aString) {
    if (aString == NULL) {
       return NULL;
     }
     CFIndex length = CFStringGetLength(aString);
     CFIndex maxSize =
       CFStringGetMaximumSizeForEncoding(length,
                                         kCFStringEncodingUTF8);
     char *buffer = (char *)malloc(maxSize);
     if (CFStringGetCString(aString, buffer, maxSize,
                            kCFStringEncodingUTF8)) {
       return buffer;
     }
     free(buffer);
     return NULL;
}
```

Convert non-const cstring to `CStringRef`

Consider the ownership:

```
const char *cstr = “Hello”;
char *bytes = malloc(strlen(cstr) + 1);
strcpy(bytes, cstr);


CFStringRef str =
   CFStringCreateWithCStringNoCopy(NULL, bytes,
                                   kCFStringEncodingUTF8,
                                   kCFAllocatorMalloc);
CFShow(str);
CFRelease(str);
```

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

CFArray

```
CFStringRef strings[3] =
     { CFSTR(“One”), CFSTR(“Two”), CFSTR(“Three”) };
CFArrayRef array = CFArrayCreate(NULL, (void *)strings, 3,
                                 &kCFTypeArrayCallBacks);
CFShow(array);
CFRelease(array);

CFMutableArrayRef array = CFArrayCreateMutable(NULL, 0,
                                   &kCFTypeArrayCallBacks);
```

CFDictionary

```
#define kCount 3
CFStringRef keys[kCount] =
         { CFSTR(“One”), CFSTR(“Two”), CFSTR(“Three”) };
CFStringRef values[kCount] =
         { CFSTR(“Foo”), CFSTR(“Bar”), CFSTR(“Baz”) };
CFDictionaryRef dict =
  CFDictionaryCreate(NULL,
                     (void *)keys,
                     (void *)values,
                     kCount,
                     &kCFTypeDictionaryKeyCallBacks,
                     &kCFTypeDictionaryValueCallBacks);
```

Others

- `CFTree`
- `CFBinaryHeap`
- `CFBitVector`

Callbacks

- `retain`
- `release`
- `copyDescription`
- `equal`
- `hash`

Non-retaining CFArray

```
CFArrayCallBacks nrCallbacks = kCFTypeArrayCallBacks;
nrCallbacks.retain = NULL;
nrCallbacks.release = NULL;
CFMutableArrayRef nrArray = CFArrayCreateMutable(NULL, 0,
                                                 &nrCallbacks);

CFStringRef string =
  CFStringCreateWithCString(NULL, “Stuff”,
                            kCFStringEncodingUTF8);
CFArrayAppendValue(nrArray, string);
CFRelease(nrArray);
CFRelease(string);
```

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

Toll-free Bridging

```
NSArray *nsArray = [NSArray arrayWithObject:@”Foo”];
printf(“%ld\n”, CFArrayGetCount((__bridge CFArrayRef)nsArray));

CFMutableArrayRef cfArray =
     CFArrayCreateMutable(NULL, 0, &kCFTypeArrayCallBacks);
CFArrayAppendValue(cfArray, CFSTR(“Foo”)); NSLog(@”%ld”,
[(__bridge id)cfArray count]); CFRelease(cfArray);
```

How does that even work?

ObjC:

```
typedef struct objc_object {
  Class isa;
} *id;
```

CF:

```
typedef struct __CFRuntimeBase {
  uintptr_t _cfisa
  ...
}
```

The CF Magic

```
CFIndex CFStringGetLength(CFStringRef str) {
  CF_OBJC_FUNCDISPATCH0(__kCFStringTypeID,
                        CFIndex, str,
                        "length");
  __CFAssertIsString(str);
  return __CFStrLength(str)
}
```

The Path

- The *real* types
- Memory management
- Introspection
- Strings
- Collections
- Toll-free Bridging
- ARC

Converting to ARC

```
- (NSString *)firstName {
  CFStringRef cfString = CFStringCreate...;
  return CFBridgingRelease(cfString);
}

CFStringRef cfStr = CFBridgingRetain([nsString copy]);
...
CFRelease(cfStr);
```

Bringing It Home

- Core Foundation is your friend
- 90% of Core Foundation is Foundation minus autorelease (and minus ARC)
- Core Foundation, as a rule, is more flexible and faster than the ObjC equivalent
- Go Forth and Core!

http://iosptl.com