CFHipsterRef Low-Level Programming on iOS & Mac OS X

Jia

Published with GitBook

Table of Contents

My Book

Welcome in my book!

Kernel

Objective-C Runtime

libobjc

```
#import <objc/runtime.h>
```

Message Sending

Consider the following Objective-C code:

```
[object message];
```

The compiler will translate this into an equivalent objc_msgSend call:

```
objc_msgSend(object, @selector(message));
```

Metaprogramming with Properties

```
#pragma mark - NSCoding
- (id)initWithCoder:(NSCoder *)decoder
{
    self = [super init];
    if (!self)
    {
        return self;
    }

    unsigned int count;
    objc_property_t *properties = class_copyPropertyList([self class], &count);
    for (NSUInteger i = 0; i < count; i++)
    {
        objc_property_t property = properties[i];
        NSString *key = [NSString stringWithUTF8String:property_getName(property)];
        [self setValue:[decoder decodeObjectForKey:key] forKey:key];
    }
    free(properties);

    return self;
}

- (void)encodeWithCoder:(NSCoder *)coder
{
    unsigned int count;
    objc_property_t *properties = class_copyPropertyList([self class], &count);
    for (NSUInteger i = 0; i < count; i++)
    {
        objc_property_t property = properties[i];
        NSString *key = [NSString stringWithUTF8String:property_getName(property)];
        [coder encodeObject:[self valueForKey:key] forKey:key];
    }

    free(properties);
}
```

Associated Objects

- objc_setAssociatedObject
- objc_getAssociatedObject
- objc_removeAssociatedObjects

Dynamically Adding a Method

```
Class c = [NSObject class];

IMP greetingIMP = imp_implementationWithBlock((NSString *)^(id self, NSString *name){
    return [NSString stringWithFormat:@"Hello, %@!", name];
});

const char *greetingTypes = [[NSString stringWithFormat:@"%s%s%s", @encode(id), @encode(i

class_addMethod(c, @selector(greetingWithName:), greetingIMP, greetingTypes);
```

Method Swizzling

```
#import <objc/runtime.h>

@implementation UIViewController (Tracking)

+ (void)load {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        Class class = [self class];

        SEL originalSelector = @selector(viewWillAppear:);
        SEL swizzledSelector = @selector(xxx_viewWillAppear:);

        Method originalMethod = class_getInstanceMethod(class, originalSelector);
        Method swizzledMethod = class_getInstanceMethod(class, swizzledSelector);

        // When swizzling a class method, use the following:
        // Class class = object_getClass((id)self);
        // ...
        // Method originalMethod = class_getClassMethod(class, originalSelector);
        // Method swizzledMethod = class_getClassMethod(class, swizzledSelector);

        BOOL didAddMethod =
            class_addMethod(class,
                originalSelector,
                method_getImplementation(swizzledMethod),
                method_getTypeEncoding(swizzledMethod));

        if (didAddMethod)
        {
            class_replaceMethod(class,
                swizzledSelector,
                method_getImplementation(originalMethod),
                method_getTypeEncoding(originalMethod));
        }
        else
        {
            method_exchangeImplementations(originalMethod, swizzledMethod);
        }
    });
}

#pragma mark - Method Swizzling

- (void)xxx_viewWillAppear:(BOOL)animated
{
    [self xxx_viewWillAppear:animated];
    NSLog(@"viewWillAppear: %@", self);
}

@end
```

+load vs. +initialize

Swizzling should always be done in +load.

There are two methods that are automatically invoked by the Objective-C runtime for each class. **+load is sent when the class is initially loaded,** while **+initialize is called just before the application calls its first method on that class or an instance of that class.** Both are optional, and are executed only if the method is implemented.

Because method swizzling affects global state, it is important to minimize the possibility of race conditions. +load is guaranteed to be loaded during class initialization, which provides a modicum of consistency for changing system-wide behavior. By contrast, +initialize provides no such guarantee of when it will be executed—in fact, it may never be called, if that class is never messaged directly by the app.

dispatch_once

Swizzling should always be done in a dispatch_once.

Again, because swizzling changes global state, we need to take every precaution available to us in the runtime. Atomicity is one such precaution, as is a guarantee that code will be executed exactly once, even across different threads. Grand Central Dispatch's dispatch_once provides both of these desirable behaviors, and should be considered as much a standard practice for swizzling as they are for initializing singletons.

Selectors, Methods, & Implementations

In Objective-C, selectors, methods, and implementations refer to particular aspects of the runtime, although in normal conversation, these terms are often used interchangeably to generally refer to the process of message sending.

Here is how each is described in Apple's Objective-C Runtime Reference:

1. **Selector (typedef struct objc_selector *SEL):** Selectors are used to represent the name of a method at runtime. A method selector is a C string that has been registered (or "mapped") with the Objective-C runtime. Selectors generated by the compiler are automatically mapped by the runtime when the class is loaded .

2. **Method (typedef struct objc_method *Method):** An opaque type that represents a method in a class definition.

3. **Implementation (typedef id (*IMP)(id, SEL, ...)):** This data type is a pointer to the start of the function that implements the method. This function uses standard C calling conventions as implemented for the current CPU architecture. The first argument is a pointer to self (that is, the memory for the particular instance of this class, or, for a class method, a pointer to the metaclass). The second argument is the method selector. The method arguments follow.

Invoking _cmd

It may appear that the following code will result in an infinite loop:

```
- (void)xxx_viewWillAppear:(BOOL)animated
{
    [self xxx_viewWillAppear:animated];
    NSLog(@"viewWillAppear: %@", NSStringFromClass([self class]));
}
```

Surprisingly, it won't. In the process of swizzling, xxx_viewWillAppear: has been reassigned to the original implementation of UIViewController -viewWillAppear:. It's good programmer instinct for calling a method on self in its own implementation to raise a red flag, but in this case, it makes sense if we remember what's really going on. However, if we were to call viewWillAppear: in this method, it would cause an infinite loop, since the implementation of this method will be swizzled to the viewWillAppear: selector at runtime.

Dynamically Creating a Class

```
@interface Product : NSObject
@property (readonly) NSString *name;
@property (readonly) double price;

- (instancetype)initWithName:(NSString *)name price:(double)price;
@end

@implementation Product
- (instancetype)initWithName:(NSString *)name price:(double)price
{
    self = [super init];
    if(!self)
    {
        return nil;
    }

    self.name = name;
    self.price = price;
    return self;
}
@end
```

1. First, a class is allocated with objc_allocateClassPair, which specifies the class's superclass and name.

   ```
   Class c = objc_allocateClassPair([NSObject class], "Product", 0);
   ```

2. After that, instance variables are added to the class using class_addIvar. at fourth argument is used to determine the variable's minimum alignment, which depends on the ivar's type and the target platform architecture.

   ```
   class_addIvar(c, "name", sizeof(id), log2(sizeof(id)), @encode(id));
   class_addIvar(c, "price", sizeof(double), sizeof(double), @encode(double));
   ```

3. The next step is to define the implementation for the initializer, with imp_implementationWithBlock:. To set name, the call is simply object_setIvar. price is set by performing a memcpy at the previously-calculated offset.

   ```
   Ivar nameIvar = class_getInstanceVariable(c, "name");
   ptrdiff_t priceIvarOffset = ivar_getOffset(class_getInstanceVariable(c, "price"));

   IMP initIMP = imp_implementationWithBlock( ^(id self, NSString *name, double price)
   {
       object_setIvar(self, nameIvar, name);

       char *ptr = ((char *)(__bridge void *)self) + priceIvarOffset;        memcpy(ptr
   ```

```
    return self;
});

const char *initTypes = [[NSString stringWithFormat:@"%s%s%s%s%s%s", @encode(id), @en

class_addMethod(c, @selector(initWithFirstName:lastName:age:),
initIMP, initTypes);
```

4. Adding methods for the ivar getters follows much the same process.

```
IMP nameIMP = imp_implementationWithBlock(^(id self) { return object_getIvar(self, na
});

const char *nameTypes =
[[NSString stringWithFormat:@"%s%s%s",
@encode(id), @encode(id), @encode(SEL)] UTF8String];
class_addMethod(c, @selector(name), nameIMP, nameTypes);
```

```
IMP priceIMP = imp_implementationWithBlock(^(id self) {
    char *ptr = ((char *)(__bridge void *)self) + priceIvarOffset;        double pri

    memcpy(&price, ptr, sizeof(price));

    return price;
});

const char *priceTypes = [[NSString stringWithFormat:@"%s%s%s", @encode(double), @enc
class_addMethod(c, @selector(age), priceIMP, priceTypes);
```

5. Finally, once all of the methods are added, the class is registered with the runtime. And from that point on, Product can be interacted with in Objective-C like any other class:

```
objc_registerClassPair(c);

Product *widget = [[Product alloc] initWithName:@"Widget" price:50.00];
NSLog(@"%@: %g", widget.name, widget.price);
```

Clang

传统的编译器通常分为三个部分，前端(frontEnd)，优化器(Optimizer)和后端(backEnd)。在编译过程中：

1. 前端主要负责词法和语法分析，将源代码转化为抽象语法树；
2. 优化器则是在前端的基础上，对得到的中间代码进行优化，使代码更加高效；
3. 后端则是将已经优化的中间代码转化为针对各自平台的机器代码。Clang则是以LLVM为后端的一款高效易用，并且与IDE结合很好的编译前端。

先来看看 C 标准是怎么规定一个 C Compiler 在翻译过程中应该完成哪些步骤的（主要参考自ISO/IEC 9899（俗称C99）标准 "5.1.1.2 Translation phases"） 第一阶段：对字符集进行转换；三并字（Trigraph sequences）被替换为单个字符。

第二阶段：行末的反斜杠 \ 被删除，实现行连接。

第三阶段：整个源文件被识别为一系列的预处理记号（preprocessing tokens）和空白字符（含注释）；一块注释替换为一个空格；换行符被保留。

第四阶段：预处理指令被执行，宏调用展开，_Pragma 一元运算符表达式执行；#include 预处理指令所涉及的文件，也被按照当前所描述的阶段一至阶段四进行处理，且处理过程是递归的；以上完毕后，预处理指令被删除。

第五阶段：字符常量（character constants）和字符串字面量（string literals）中的字符集成员和转义序列被转换。

第六阶段：连接毗邻的字符串字面量记号。

第七阶段：预处理记号（preprocessing tokens）转换为一般记号（token）；并对转换后的一般记号进行语法及语义上的分析，然后转换为翻译单元（translation unit）。

第八阶段：解析对外部对象和函数的引用；链接库组件（library components）， 获得可执行的程序映像文件（program image）。 上面八个阶段中，前六个阶段至第七阶段的前半部分，大体上对应了大多数程序员熟知的“预处理过程”，而第七个阶段的后半部分则是“编译过程”（实则涵盖了文法分析、语义分析、等许多细分阶段）， 最后的第八阶段则是“链接过程”。

libclang

libclang is the C interface to the Clang LLVM front-end. It's a powerful way for C & Objective-C programs to introspect their own internal structure and composition.

Clang Components

libsupport	Basic support library, from LLVM.
libsystem	System abstraction library, from LLVM.
libbasic	Diagnostics, SourceLocations, SourceBuffer abstraction, file system caching for input source files.
libast	Provides classes to represent the C AST, the C type system, builtin functions, and various helpers for analyzing and manipulating the AST (visitors, pretty printers, etc).
liblex	Lexing and preprocessing, identifier hash table, pragma handling, tokens, and macro expansion.
libparse	Parsing. is library invokes coarse-grained Actions provided by the client (e.g. libsema builds ASTs) but knows nothing about ASTs or other client-specific data structures.
libsema	Semantic Analysis. is provides a set of parser actions to build a standardized AST for programs.
libcodegen	Lower the AST to LLVM IR for optimization & code generation.
librewrite	Editing of text buffers (important for code rewriting transformation, like refactoring).
libanalysis	Static analysis support.

Translation Units

The first step to working with source code is to load it into memory. Clang operates on translation units, which are the ultimate form of C or Objective-C source code, after all of the **#include, #import,** and any other preprocessor directives have been evaluated.

A translation unit is created by passing a path to the source code file, and any command-line compilation arguments:

```
CXIndex idx = clang_createIndex(0, 0);
const char *filename = "path/to/Source.m";
int argc;
const char *argv[];

CXTranslationUnit tu = clang_parseTranslationUnit(idx, filename, 0, argv, argc, 0, CXTran
{ ... }
clang_disposeTranslationUnit(tu);
clang_disposeIndex(idx);
```

AST

In order to understand the structure of a translation unit, Clang constructs an Abstract Syntax Tree (AST). ASTs are the platonic ideal of what code, derived by distilling constructs from their representation.

ImageIO

Image I/O is a powerful, albeit lesser-known framework for working with images. Independent of Core Graphics, it can read and write between between many different formats, access photo metadata, and perform common image processing operations. e framework offers the fastest image encoders and decoders on the platform, with advanced caching mechanisms and even the ability to load images incrementally.

Supported Image Types

According to the official docs, Image I/O supports "most image formats". Rather than take the docs at their word and guess what exactly that entails, this information can be retrieved programmatically.

CGImageSourceCopyTypeIdentifiers returns list of UTIs for image types supported:

Writing to a File

```
UIImage *image = ...;
NSURL *fileURL = [NSURL fileURLWithPath:@"/path/to/output.jpg"];
NSString *UTI = @"public.jpeg";
NSDictionary *options = @{
    (__bridge id) kCGImageDestinationLossyCompressionQuality: @(0.75),
    (__bridge id)kCGImagePropertyOrientation: @(4), (__bridge id)kCGImagePropertyHasAlpha
    };

CGImageDestinationRef imageDestinationRef =
    CGImageDestinationCreateWithURL((__bridge CFURLRef)fileURL,
        (__bridge CFStringRef)UTI, 1, NULL);

CGImageDestinationAddImage(imageDestinationRef,[imageCGImage],(__bridge CFDictionaryRef)o

CGImageDestinationFinalize(imageDestinationRef);
CFRelease(imageDestinationRef);
```

Reading from a File

```
NSURL *fileURL = [NSURL fileURLWithPath:@"/path/to/input.jpg"];
NSDictionary *options = @{
    (__bridge id)kCGImageSourceTypeIdentifierHint: @"{public.jpeg",
    (__bridge id)kCGImageSourceShouldCache: @(YES),
    (__bridge id)kCGImageSourceShouldAllowFloat: @(YES),
    };

CGImageSourceRef imageSourceRef =
    CGImageSourceCreateWithURL((__bridge CFURLRef)fileURL, NULL);
CGImageRef imageRef =
    CGImageSourceCreateImageAtIndex(imageSourceRef, 0, (__bridge CFDictionaryRef)options)

UIImage *image = [UIImage imageWithCGImage:imageRef];

CFRelease(imageRef);
CFRelease(imageSourceRef);
```

Incrementally Reading an Image

```
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask *)dataTask didR
{
    [self.mutableResponseData appendData:data];

    CGImageSourceUpdateData(self.imageSourceRef,
        (__bridge CFDataRef)self.mutableResponseData,
        [self.mutableResponseData length]
        [dataTask countOfBytesExpectedToReceive]);

    if (CGSizeEqualToSize(self.imageSize, CGSizeZero))
    {
        NSDictionary *properties =
            (__bridge_transfer NSDictionary *)CGImageSourceCopyPropertiesAtIndex(self.ima
    0, NULL);

        if (properties)
        {
            NSNumber *width = properties[(__bridge id)kCGImagePropertyPixelWidth];
            NSNumber *height = properties[(__bridge id)kCGImagePropertyPixelHeight];

            if (width && height)
            {
                self.imageSize = CGSizeMake([width floatValue],
            }
        }
    }

    CGImageRef imageRef =
        CGImageSourceCreateImageAtIndex(self.imageSourceRef, 0,
    NULL);
    UIImage *image = [UIImage imageWithCGImage:imageRef];
    CFRelease(imageRef);

    dispatch_async(dispatch_get_main_queue(), ^{
    // delete or block callback to update with image
    });
}
```

Image Metadata

Metadata is divided into several different dictionaries, which can be specified with any of the following keys:

- kCGImagePropertyTIFFDictionary
- kCGImagePropertyGIFDictionary
- kCGImagePropertyJFIFDictionary
- kCGImagePropertyExifDictionary
- kCGImagePropertyPNGDictionary
- kCGImagePropertyIPTCDictionary
- kCGImagePropertyGPSDictionary
- kCGImagePropertyRawDictionary
- kCGImagePropertyCIFFDictionary
- kCGImageProperty8BIMDictionary
- kCGImagePropertyDNGDictionary
- kCGImagePropertyExifAuxDictionary

```
NSDictionary *properties =
    (__bridge_transfer NSDictionary *)CGImageSourceCopyPropertiesAtIndex(self.imageSource
    NULL);

NSDictionary *EXIF = properties[(__bridge id)kCGImagePropertyExifDictionary];

if (EXIF)
{
    NSString *Fnumber = EXIF[(__bridge id)kCGImagePropertyExifFNumber];
    NSString *exposure = EXIF[(__bridge id)kCGImagePropertyExifExposureTime];
    NSString *ISO = EXIF[(__bridge id)kCGImagePropertyExifISOSpeedRatings];

    NSLog(@"Shot Information: %@ %@ %@", Fnumber, exposure, ISO);
}

NSDictionary *GPS = properties[(__bridge id)kCGImagePropertyGPSDictionary];
if (GPS)
{
    NSString *latitude = GPS[(__bridge id)kCGImagePropertyGPSLatitude];
    NSString *latitudeRef = GPS[(__bridge id)kCGImagePropertyGPSLatitudeRef];
    NSString *longitude = GPS[(__bridge id)kCGImagePropertyGPSLongitude];
    NSString *longitudeRef = GPS[(__bridge id)kCGImagePropertyGPSLongitudeRef];

    NSLog(@"GPS: %@ %@ / %@ %@", latitude, latitudeRef, longitude, longitudeRef);
}
```

Accelerate

Benchmarking Performance

1. Populating an Array

    ```
    NSUInteger count = 10000000;
    float *array = malloc(count * sizeof(float));

    for (NSUInteger i = 0; i < count; i++)
    {
        array[i] = i;
    }
    ```

    ```
    float initial = 0;
    float increment = 1;
    vDSP_vramp(&initial, &increment, array, 1, count);
    ```

2. Multiplying an Array

    ```
    for (NSUInteger i = 0; i < count; i++)
    {
        array[i] *= 2.0f;
    }
    ```

    ```
    cblas_sscal(count, 2.0f, array, 1);
    ```

3. Summing an Array

    ```
    float sum = 0;
    for (NSUInteger i = 0; i < count; i++)
    {
        sum += array[i];
    }
    ```

    ```
    float sum = cblas_sasum(count, array, 1);
    ```

4. Searching

    ```
    for (NSUInteger i = 0; i < count; i++)
    {
        array[i] = (float)arc4random();
    }

    NSUInteger maxLocation = 0;
    for (NSUInteger i = 0; i < count; i++)
    ```

```
{
    if (array[i] > array[maxLocation])
    {
        maxLocation = i;
    }
}
```

```
NSUInteger maxLocation = cblas_isamax(count, array, 1);
```

vecLib

vecLib is comprised of the following 9 headers:

cblas.h / vBLAS.h	Interface for BLAS functions
clapack.h	Interface for LAPACK functions
vectorOps.h	Vector implementations of the BLAS routines.
vBasicOps.h	Basic algebraic operations. 8-, 16-, 32-, 64-, 128-bit division, saturated addition / subtraction, shiϐ / rotate, etc.
vfp.h	Transcendental operations (sin, cos, log, etc.) on single vector floating point quantities.
vForce.h	Transcendental operations on arrays of floating point quantities.
vBigNum.h	Operations on large numbers (128-, 256-, 512-, 1024-bit quantities)
vDSP.h	Digital signal processing algorithms including FFTs, signal clipping, filters, and type conversions.

vDSP.h

Fast-Fourier Transform (FFT) is the fundamental digital signal processing algorithm. It decomposes a sequence of values into components with different frequencies.

Although they have wide-ranging applications across mathematics and engineering, most application developers encounter FFTs for audio or video processing, as a way of determining the critical values in a noisy signal.

```
int x = 8;
int y = 8;

int dimensions = x * y;

int log2_x = (int)log2((double)x);
int log2_y = (int)log2((double)y);

DSPComplex *data = (DSPComplex *)malloc(sizeof(DSPComplex) * dimensions);
for (NSUInteger i = 0; i < dimensions; i++)
{
    data[i].real = (float)i;
    data[i].imag = (float)(dimensions - i) - 1.0f;
}

DSPSplitComplex input = {
    .realp = (float *)malloc(sizeof(float) * dimensions),
    .imagp = (float *)malloc(sizeof(float) * dimensions),
};

vDSP_ctoz(data, 2, &input, 1, dimensions);

FFTSetup weights = vDSP_create_fftsetup(fmax(log2_x, log2_y), kFFTRadix2);
vDSP_fft2d_zip(weights, &input, 1, 0, log2_x, log2_y, FFT_FORWARD);
```

```
vDSP_destroy_fftsetup(fft_weights);

vDSP_ztoc(&input, 1, data, 2, dimensions);

for (NSUInteger i = 0; i < dimensions; i++)
{
    NSLog(@"%g %g", data[i].real, data[i].imag);
}

free(input.realp);
free(input.imagp);
free(data);
```

vImage

vImage is comprised of 6 headers:

Alpha.h	Alpha compositing functions.
Conversion.h	Converting between image format (e.g. Planar8 to PlanarF, ARGB8888 to Planar8).
Convoluton.h	Image convolution routines (e.g. blurring and edge detection).
Geometry.h	Geometric transformations (e.g. rotate, scale, shear, affine warp).
Histogram.h	Functions for calculating image histograms and image normalization.
Morphology.h	Image morphology procedures (e.g. feature detection, dilation, erosion).
Tranform.h	Image transformation operations (e.g. gamma correction, colorspace conversion).

Alpha.h

Alpha compositing is a process of combining multiple images according to their alpha components. For each pixel in an image, the alpha, or transparency, value is used to determine how much of the image underneath it will be shown.

vImage functions are available for blending or clipping. The most common operation is compositing a top image onto a bottom image:

```
UIImage *topImage, *bottomImage = ...;
CGImageRef topImageRef = [topImage CGImage];
CGImageRef bottomImageRef = [bottomImage CGImage];

CGDataProviderRef topProvider = CGImageGetDataProvider(topImageRef);
CFDataRef topBitmapData = CGDataProviderCopyData(topProvider);

size_t width = CGImageGetWidth(topImageRef);
size_t height = CGImageGetHeight(topImageRef);
size_t bytesPerRow = CGImageGetBytesPerRow(topImageRef);

vImage_Buffer topBuffer = {
    .data = (void *)CFDataGetBytePtr(topBitmapData),
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};

CGDataProviderRef bottomProvider = CGImageGetDataProvider(bottomImageRef);
CFDataRef bottomBitmapData = CGDataProviderCopyData(bottomProvider);

vImage_Buffer bottomBuffer = {
    .data = (void *)CFDataGetBytePtr(bottomBitmapData),
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};
```

```
void *outBytes = malloc(height * bytesPerRow);
vImage_Buffer outBuffer = {
    .data = outBytes,
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};

vImage_Error error = vImagePremultipliedAlphaBlend_ARGB8888(&topBuffer, &bottomBuffer, &o

if (error)
{
    NSLog(@"Error: %ld", error);
}
```

Conversion.h

```
UIImage *image = ...;
CGImageRef imageRef = [image CGImage];

size_t width = CGImageGetWidth(imageRef);
size_t height = CGImageGetHeight(imageRef);
size_t bitsPerComponent = CGImageGetBitsPerComponent(imageRef);
size_t bytesPerRow = CGImageGetBytesPerRow(imageRef);

CGDataProviderRef sourceImageDataProvider = CGImageGetDataProvider(imageRef);
CFDataRef sourceImageData = CGDataProviderCopyData(sourceImageDataProvider);
vImage_Buffer sourceImageBuffer = {
    .data = (void *)CFDataGetBytePtr(sourceImageData),
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};

uint8_t *destinationBuffer = malloc(CFDataGetLength(sourceImageData));
vImage_Buffer destinationImageBuffer = {
    .data = destinationBuffer,
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};

const uint8_t channels[4] = {0, 3, 2, 1}; // ARGB -> ABGR
vImagePermuteChannels_ARGB8888(&sourceImageBuffer,&destinationImageBuffer,channels, kvIma

CGColorSpaceRef colorSpaceRef = CGColorSpaceCreateDeviceRGB();
CGContextRef destinationContext =
CGBitmapContextCreateWithData(destinationBuffer, width,
height, bitsPerComponent, bytesPerRow, colorSpaceRef, kCGBitmapByteOrderDefault | kCGImag
NULL);

CGImageRef permutedImageRef = CGBitmapContextCreateImage(destinationContext);
UIImage *permutedImage = [UIImage imageWithCGImage:permutedImageRef];

CGImageRelease(permutedImageRef); CGContextRelease(destinationContext);
```

```
CGColorSpaceRelease(colorSpaceRef);
```

Convolution.h

Image convolution is the process of multiplying each pixel and its adjacent pixels by a kernel, or square matrix with a sum of 1. Depending on the kernel, a convolution operation can either blur, sharpen, emboss, or detect edges.

Except for specific situations where a custom kernel is required, convolution operations would be better-served by the Core Image framework, which utilizes the GPU. However, for a straightforward CPU-based solution, vImage delivers:

Blurring an Image

```
UIImage *inImage = ...;
CGImageRef inImageRef = [inImage CGImage];

CGDataProviderRef inProvider = CGImageGetDataProvider(inImageRef);
CFDataRef inBitmapData = CGDataProviderCopyData(inProvider);

vImage_Buffer inBuffer = {
    .data = (void *)CFDataGetBytePtr(inBitmapData),
    .width = CGImageGetWidth(inImageRef),
    .height = CGImageGetHeight(inImageRef),
    .rowBytes = CGImageGetBytesPerRow(inImageRef),
};

void *outBytes = malloc(CGImageGetBytesPerRow(inImageRef) * CGImageGetHeight(inImageRef))
vImage_Buffer outBuffer = {
    .data = outBytes,
    .width = inBuffer.width,
    .height = inBuffer.height,
    .rowBytes = inBuffer.rowBytes,
};

uint32_t length = 5; // Size of convolution
vImage_Error error =
    vImageBoxConvolve_ARGB8888(&inBuffer,
        &outBuffer,
        NULL,
        0,
        0,
        length,
        length,
        NULL,
        kvImageEdgeExtend);

if (error)
{
    NSLog(@"Error: %ld", error);
}

CGColorSpaceRef colorSpaceRef = CGColorSpaceCreateDeviceRGB(); CGContextRef c =
    CGBitmapContextCreate(outBuffer.data,
        outBuffer.width,
        outBuffer.height,
        8,
        outBuffer.rowBytes,
        colorSpaceRef,
        kCGImageAlphaNoneSkipLast);
CGImageRef outImageRef = CGBitmapContextCreateImage(c);
UIImage *outImage = [UIImage imageWithCGImage:outImageRef];

CGImageRelease(outImageRef);
CGContextRelease(c);
CGColorSpaceRelease(colorSpaceRef);
CFRelease(inBitmapData);
```

Geometry.h

Resizing an image is another operation that is perhaps more suited for another, GPU-based framework, like Image I/O. For a given vImage buffer, it might be more performant to scale with Accelerate using vImageScale_* rather than convert back and forth between a CGImageRef:

Resizing an Image

```
double scaleFactor = 1.0 / 5.0;
void *outBytes = malloc(trunc(inBuffer.height * scaleFactor) * inBuffer.rowBytes);
vImage_Buffer outBuffer = {
    .data = outBytes,
    .width = trunc(inBuffer.width * scaleFactor),
    .height = trunc(inBuffer.height * scaleFactor),
    .rowBytes = inBuffer.rowBytes,
};

vImage_Error error =
    vImageScale_ARGB8888(&inBuffer,
        &outBuffer,
        NULL,
        kvImageHighQualityResampling);

if (error)
{
    NSLog(@"Error: %ld", error);
}
```

Histogram.h

Detecting if an Image Has Transparency

```
UIImage *image;
CGImageRef imageRef = [image CGImage];
CGDataProviderRef dataProvider = CGImageGetDataProvider(imageRef); CFDataRef bitmapData =
vImagePixelCount a[256], r[256], g[256], b[256];

vImagePixelCount *histogram[4] = {a, r, g, b};
```

```
 vImage_Buffer buffer = {
    .data = (void *)CFDataGetBytePtr(bitmapData),
    .width = CGImageGetWidth(imageRef),
    .height = CGImageGetHeight(imageRef),
    .rowBytes = CGImageGetBytesPerRow(imageRef),
};

vImage_Error error = vImageHistogramCalculation_ARGB8888(&buffer, histogram, kvImageNoFla

if (error)
{
    NSLog(@"Error: %ld", error);
}
BOOL hasTransparency = NO;
for (NSUInteger i = 0; !hasTransparency && i < 255; i++)
{
    hasTransparency = histogram[3][i] == 0;
}

CGDataProviderRelease(dataProvider);
CFRelease(bitmapData);
```

Morphology.h

Dilating an Image

```
size_t width = image.size.width;
size_t height = image.size.height;

size_t bitsPerComponent = 8;
size_t bytesPerRow = CGImageGetBytesPerRow([image CGImage]);

CGColorSpaceRef colorSpaceRef = CGColorSpaceCreateDeviceRGB();
CGContextRef sourceContext =
    CGBitmapContextCreate(NULL,
        width,
        height,
        bitsPerComponent,
        bytesPerRow,
        colorSpaceRef,
        kCGBitmapByteOrderDefault | kCGImageAlphaPremultipliedFirst);

CGContextDrawImage(sourceContext, CGRectMake(0.0f, 0.0f, width, height),[image CGImage]);

void *sourceData = CGBitmapContextGetData(sourceContext);
vImage_Buffer sourceBuffer = {
    .data = sourceData,
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};

size_t length = height * bytesPerRow;
void *destinationData = malloc(length);
vImage_Buffer destinationBuffer = {
    .data = destinationData,
    .width = width,
    .height = height,
    .rowBytes = bytesPerRow,
};

static unsigned char kernel[9] =
{
    1, 1, 1,
    1, 1, 1,
    1, 1, 1,
};

vImageDilate_ARGB8888(&sourceBuffer,
    &destinationBuffer,
    0,
    0,
    kernel,
    9,
    9,
    kvImageCopyInPlace);

CGContextRef destinationContext =
    CGBitmapContextCreateWithData(destinationData,
    width,
    height,
    bitsPerComponent,
    bytesPerRow,
    colorSpaceRef,
    kCGBitmapByteOrderDefault | kCGImageAlphaPremultipliedFirst,
    NULL,
    NULL);

CGImageRef dilatedImageRef = CGBitmapContextCreateImage(destinationContext);
UIImage *dilatedImage = [UIImage imageWithCGImage:dilatedImageRef];

CGImageRelease(dilatedImageRef);
CGContextRelease(destinationContext);
CGContextRelease(sourceContext);
CGColorSpaceRelease(colorSpaceRef);
```

Security

The Security framework can be divided up into Keychain Services, Cryptographic Message Syntax, Security Transform Services, and CommonCrypto.

Keychain Services

Keychain is the password management system on iOS & OS X. It stores certificates and private keys, as well as passwords for websites, servers, wireless networks, and other encrypted volumes.

Interactions with the Keychain are mediated through queries, rather than direct manipulation. The queries themselves can be quite complicated, and cumbersome with a C API.

A query is a dictionary consisting of the following components:

- The class of item to search for, either "Generic Password", "Internet Password", "Certificate", "Key", or "Identity".
- The return type for the query, either "Data", "Attributes", "Reference", or "Persistent Reference".
- One or more attribute key-value pairs to match on.
- One or more search key-value pairs to further refine the results, such as whether to match strings with case sensitivity, only match trusted certificates, or limit to just one result or return all.

Getting Keychain Items

```
NSString *service = @"com.example.app";
NSString *account = @"username";
NSDictionary *query =
@{
    (__bridge id)kSecClass: (__bridge id)kSecClassGenericPassword, (__bridge id)kSecAttrS
    (__bridge id)kSecAttrAccount: key,
    (__bridge id)kSecMatchLimit: kSecMatchLimitOne,
};

CFTypeRef result;
OSStatus status =
SecItemCopyMatching((__bridge CFDictionaryRef)query, &result);
```

In this example, the query tells the keychain to find all generic password items for the service com.example.app with the matching username. kSecAttrService defines the scope of credentials, while kSecAttrAccount acts as a unique identifier. e search option kSecMatchLimitOne is passed to ensure that only the first match is returned, if any.

If status is equal to errSecSuccess (0), then result should be populated with the matching credential.

Adding and Updating Keychain Items

```
NSData *data = ...;

if (status == errSecSuccess)
```

```
{
    NSDictionary *updatedAttributes = @{(__bridge id)kSecValueData: data};

    SecItemUpdate((__bridge CFDictionaryRef)query, (__bridge CFDictionaryRef)updatedAttri
}
else
{
    NSMutableDictionary *attributes = [query mutableCopy];
    attributes[(__bridge id)kSecValueData] = data;
    attributes[(__bridge id)kSecAttrAccessible] = (__bridge id)kSecAttrAccessibleAfterFir
    SecItemAdd((__bridge CFDictionaryRef)attributes, NULL);
}
```

Cryptographic Message Syntax

Cryptographic Message Syntax is the IETF’s standard for public key encryption and digital signatures for S/MIME messages. Apple’s Cryptographic Message Syntax Services in the Security framework provide APIs that implement these industry standard algorithms.

Messages can either be signed, encrypted, or both, by any number of signers or recipients. To sign a message is to allow the recipient to verify its sender. To encrypt a message is to ensure that it kept secret from everyone but the recipients, who alone are able to decrypt the message’s content. These two operations are orthogonal, but cryptographically related.

Encoding a Message

```
NSData *data;
SecCertificateRef certificateRef;
CMSEncoderRef encoder;
CMSEncoderCreate(&encoder);

// Encrypt
CMSEncoderUpdateContent(encoder, [data bytes], [data length]);
CMSEncoderAddRecipients(encoder, certificateRef);

// Sign
SecIdentityRef identityRef = nil;
SecIdentityCreateWithCertificate(nil, certificateRef, &identityRef);
CMSEncoderUpdateContent(encoder, [data bytes], [data length]);
CMSEncoderAddSigners(encoder, identityRef);
CFRelease(identityRef);
CMSEncoderUpdateContent(encoder, [data bytes], [data length]);

MSEncoderAddSignedAttributes(encoder, kCMSAttrSmimeCapabilities);
CFDataRef encryptedDataRef;
CMSEncoderCopyEncodedContent(encoder, &encryptedDataRef);
NSData *encryptedData = [NSData dataWithData:(__bridge NSData *)encryptedDataRef];
CFRelease(encoder);
```

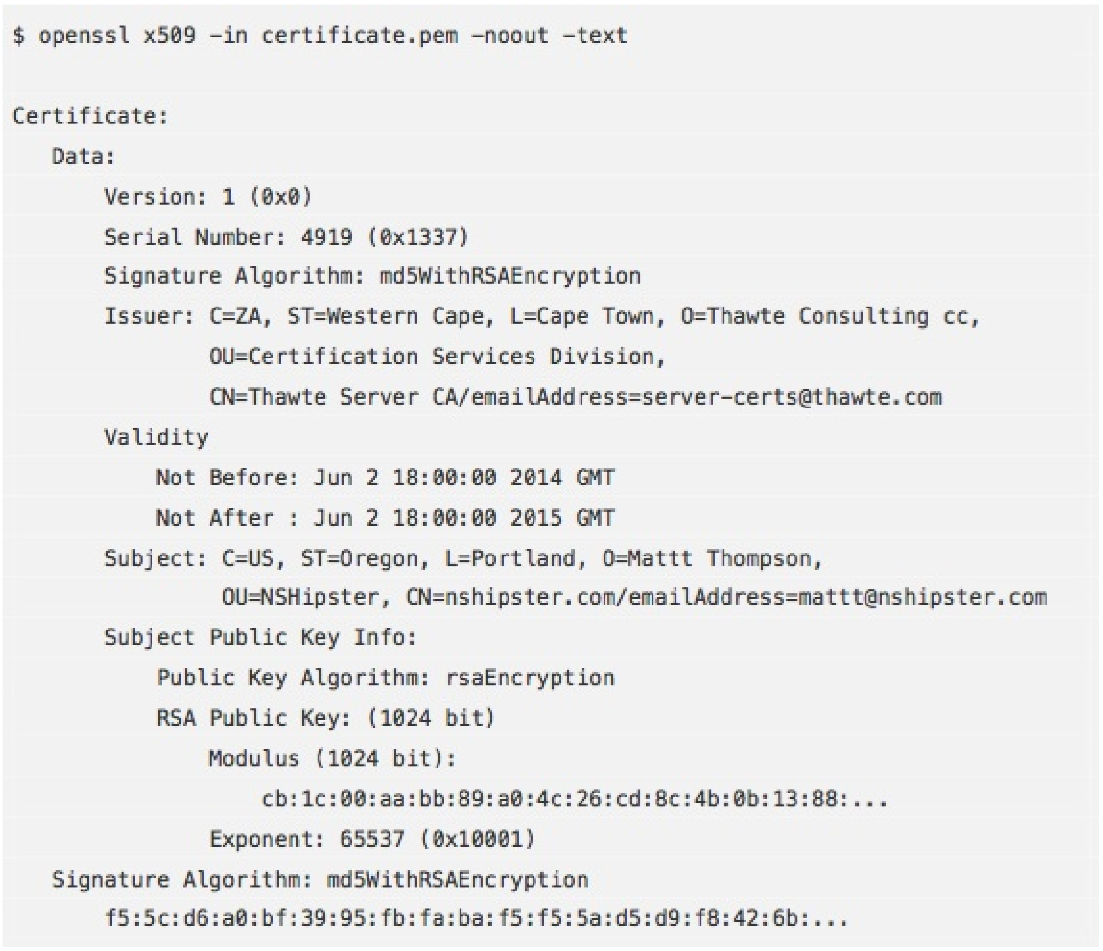
Decoding a Message

```
CMSDecoderRef decoder;
CMSDecoderCreate(&decoder);
CMSDecoderUpdateMessage(decoder, [encryptedData bytes], [encryptedData length]);? ;
CMSDecoderFinalizeMessage(decoder);

CFDataRef decryptedDataRef;
CMSDecoderCopyContent(decoder, &decryptedDataRef);
NSData *decryptedData = [NSData dataWithData:(__bridge NSData *)decryptedDataRef];

CFRelease(decryptedDataRef);
CFRelease(decoder);
```

Certificate, Key, and Trust Services



```
$ openssl x509 -in certificate.pem -noout -text

Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 4919 (0x1337)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jun 2 18:00:00 2014 GMT
           Not After : Jun 2 18:00:00 2015 GMT
       Subject: C=US, ST=Oregon, L=Portland, O=Mattt Thompson,
                OU=NSHipster, CN=nshipster.com/emailAddress=mattt@nshipster.com
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   cb:1c:00:aa:bb:89:a0:4c:26:cd:8c:4b:0b:13:88:...
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       f5:5c:d6:a0:bf:39:95:fb:fa:ba:f5:f5:5a:d5:d9:f8:42:6b:...
```

Scanning through the plain text output, several pieces of information come to the surface:

- Certificate issuer
- Validity period
- Certificate holder
- Public key of the owner
- Digital signature from the certification authority

Certificates are the basis for the cryptographic infrastructure used to secure the internet. One of the most common interactions an iOS or OS X developer has certificates is an authentication challenge from a URL request:

```
NSURLAuthenticationChallenge *challenge = ...;
SecTrustRef trust = challenge.protectionSpace.serverTrust;
SecPolicyRef X509Policy = SecPolicyCreateBasicX509();
SecTrustSetPolicies(serverTrust, (__bridge CFArrayRef)@[(__bridge id)X509Policy]);

SecTrustResultType result;
assert(SecTrustEvaluate(trust, &result) == errSecSuccess);
```

Security Transform Services

Base64 Encoding

```
SecTransformRef transform = SecEncodeTransformCreate(kSecBase64Encoding, NULL);
SecTransformSetAttribute(transform, kSecTransformInputAttributeName,
(__bridge CFDataRef)data,
NULL);

NSData *encodedData =
(__bridge_transfer NSData *)SecTransformExecute(transform, NULL);

CFRelease(transform);
```

Base64 Decoding

```
SecTransformRef transform = SecEncodeTransformCreate(kSecBase64Decoding, NULL);

NSData *decodedData =
(__bridge_transfer NSData *)SecTransformExecute(transform, NULL);

CFRelease(transform);
```

Randomization Services

Cryptography is predicated on unpredictable, random values. Without such a guarantee, it's all just security theater. SecRandomCopyBytes reads from /dev/random, which generates cryptographically-secure random bytes. /dev/random is a special file on Unix systems that streams entropy based on the environmental noise of the device.

```
NSUInteger length = 1024;

NSMutableData *mutableData = [NSMutableData dataWithLength:length];

OSStatus success = SecRandomCopyBytes(kSecRandomDefault,
length, mutableData.mutableBytes);

__Require_noErr(success, exit);
```

CommonCrypto

Digests

To calculate a checksum in code, use the CC_SHA1 function:

```
NSData *data = ...;

uint8_t output[CC_SHA1_DIGEST_LENGTH]; CC_SHA1(data.bytes, data.length, output);

NSData *digest = [NSData dataWithBytes:output length:CC_SHA1_DIGEST_LENGTH];
```

HMAC

```
NSData *data, *key;

unsigned int length = CC_SHA1_DIGEST_LENGTH; unsigned char output[length];

CCHmac(kCCHmacAlgSHA1, key.bytes, key.length, data.bytes, data.length, output);
```

Symmetric Encryption

1. The first step is to create a function that generates a PBKDF2 key from a salted password. A salt is random data used as an additional input to a one-way function performed on a password.

   ```
   static NSData * AES128PBKDF2KeyWithPassword(NSString *password, NSData *salt,
   NSError * __autoreleasing *error)
   {
       NSCParameterAssert(password);
       NSCParameterAssert(salt);

       NSMutableData *mutableDerivedKey = [NSMutableData dataWithLength:kCCKeySizeAES12

       CCCryptorStatusstatus =
           CCKeyDerivationPBKDF(kCCPBKDF2,
               [password UTF8String],
               [passwordlengthOfBytesUsingEncoding:NSUTF8StringEncoding],
               [salt bytes],
               [salt length],
               kCCPRFHmacAlgSHA256,
               1024,
               [mutableDerivedKey mutableBytes],
               kCCKeySizeAES128);

       NSData *derivedKey = nil;
       if (status != kCCSuccess)
       {
           if (error)
           {
   ```

```
            *error = [[NSError alloc] initWithDomain:nil code:status userInfo:nil];
        }
    }
    else
    {
        derivedKey = [NSData dataWithData:mutableDerivedKey];
    }

    return derivedKey;
}
```

2. Next, a function to encrypt the data can be created, which takes the data to encrypt and password, and returns the generated salt and initialization, as well as any error encountered in performing the operation, as out arguments:

```
static NSData * AES128EncryptedDataWithData(NSData *data, NSString *password,
NSData * __autoreleasing *salt, NSData * __autoreleasing *initializationVector,
NSError * __autoreleasing *error)
{
    NSCParameterAssert(initializationVector);
    NSCParameterAssert(salt);

    uint8_t *saltBuffer = malloc(8);
    SecRandomCopyBytes(kSecRandomDefault, 8, saltBuffer);
    *salt = [NSData dataWithBytes:saltBuffer length:8];

    NSData *key = AES128PBKDF2KeyWithPassword(password, *salt, error);

    uint8_t *initializationVectorBuffer = malloc(kCCBlockSizeAES128);
    SecRandomCopyBytes(kSecRandomDefault,kCCBlockSizeAES128, initializationVectorBuf
    *initializationVector = [NSData dataWithBytes:initializationVector length:kCCBlo

    size_t size = [data length] + kCCBlockSizeAES128;
    void *buffer = malloc(size);

    size_t numberOfBytesEncrypted = 0;
    CCCryptorStatusstatus =
        CCCrypt(kCCEncrypt,
            kCCAlgorithmAES128,
            kCCOptionPKCS7Padding,
            [key bytes],
            [key length],
            [*initializationVector bytes],
            [data bytes],
            [data length],
            buffer,
            size,
            &numberOfBytesEncrypted);

    NSData *encryptedData = nil;
    if (status != kCCSuccess)
    {
        if (error)
        {
            *error = [[NSError alloc] initWithDomain:nil code:status userInfo:nil];
        }
    }
    else
    {
        encryptedData = [[NSData alloc] initWithBytes:buffer length:numberOfBytesEnc
    }

    return encryptedData;
}
```

3. Finally, to decrypt the data, do the same process in reverse, this time passing the data and password along with the salt and initialization vector generated from the encryption function:

```
staticNSData*AES128DecryptedDataWithData(NSData*data,NSString*password, NSData *salt
{
    NSData *key = AES128PBKDF2KeyWithPassword(password, salt, error);

    size_t size = [data length] + kCCBlockSizeAES128;
    void *buffer = malloc(size);

    size_t numberOfBytesDecrypted = 0;
    CCCryptorStatusstatus =
        CCCrypt(kCCDecrypt,
            kCCAlgorithmAES128,
            kCCOptionPKCS7Padding,
            [key bytes],
            [key length],
            [initializationVector bytes],
            [data bytes],
            [data length],
            buffer,
            size,
            &numberOfBytesDecrypted);

    NSData *encryptedData = nil;
    if (status != kCCSuccess)
    {
        if (error)
        {
            *error = [[NSError alloc] initWithDomain:nil code:status userInfo: nil];
        }
    }
    else
    {
        encryptedData = [[NSData alloc] initWithBytes:buffer length:numberOfBytesDec
    }

    return encryptedData;
}
```

Determining Network Reachability Synchronously

Like any networking, establishing reachability should not be done synchronously.

```
@import SystemConfiguration;

SCNetworkReachabilityRef networkReachability = SCNetworkReachabilityCreateWithName(kCFAll
[@"www.apple.com" UTF8String]);

SCNetworkReachabilityFlags flags = SCNetworkReachabilityGetFlags(networkReachability, &fl

// Use flags to determine reachability
CFRelease(networkReachability);
```

SCNetworkReachabilityRef is the data type responsible for determining network reachability. It can be created by either passing host name, like in the previous example, or a sockaddr address:

```
BOOL ignoresAdHocWiFi = NO;

struct sockaddr_in ipAddress;
bzero(&ipAddress, sizeof(ipAddress));
ipAddress.sin_len = sizeof(ipAddress);
ipAddress.sin_family = AF_INET;
ipAddress.sin_addr.s_addr = htonl(ignoresAdHocWiFi ? INADDR_ANY : IN_LINKLOCALNETNUM);

SCNetworkReachabilityRef networkReachability = SCNetworkReachabilityCreateWithAddress(kCF
```

```
BOOL isReachable =
((flags & kSCNetworkReachabilityFlagsReachable) != 0);

BOOL needsConnection =
((flags & kSCNetworkReachabilityFlagsConnectionRequired) != 0);

BOOL canConnectionAutomatically =
(((flags & kSCNetworkReachabilityFlagsConnectionOnDemand ) != 0) ||
((flags & kSCNetworkReachabilityFlagsConnectionOnTraffic) != 0));

BOOL canConnectWithoutUserInteraction = (canConnectionAutomatically &&
(flags & kSCNetworkReachabilityFlagsInterventionRequired) == 0);

BOOL isNetworkReachable = (isReachable &&
(!needsConnection || canConnectWithoutUserInteraction));
```

```
if (isNetworkReachable == NO)
{
    // Not Reachable
}
#if TARGET_OS_IPHONE
```

```
else if ((flags & kSCNetworkReachabilityFlagsIsWWAN) != 0)
{
    // Reachable via WWAN
}
#endif
else
{
    // Reachable via WiFi
}
```

Calling SCNetworkReachabilityFlags on the main thread invokes a DNS lookup with a 30 second timeout. This is bad. Don’t make blocking, synchronous calls to SCNetworkReachabilityFlags.

Determining Network Reachability Asynchronously

Thankfully, the System Configuration framework provides a set of APIs for monitoring reachability changes asynchronously.

```
static void ReachabilityCallback( SCNetworkReachabilityRef target, SCNetworkConnectionFla
{
    // ...
}
```

```
SCNetworkReachabilityContext context = {0, NULL, NULL, NULL, NULL};

SCNetworkReachabilitySetCallback(networkReachability, ReachabilityCallback,
&context));

SCNetworkReachabilityScheduleWithRunLoop(reachability, CFRunLoopGetMain(),
kCFRunLoopCommonModes));
```

Xcode Toolchain

Xcode Tools

1. xcode-select

   ```
   xcode-select install
   ```

   This will install the Command Line Tools, which are necessary for compiling Objective-C code.

2. xcrun

   xcrun is the fundamental Xcode command line tool. With it, all other tools are invoked.

   ```
   xcrun xcodebuild
   ```

   In addition to running commands, xcrun can find binaries and show the path to an SDK:

   ```
   xcrun --find clang
   xcrun --sdk iphoneos --find pngcrush
   xcrun --sdk macosx --show-sdk-path
   ```

   Because xcrun executes in the context of the active Xcode version (as set by xcode-select), it is easy to have multiple versions of the Xcode toolchain co-exist on a single system.

   Using xcrun in scripts and other external tools has the advantage of ensuring consistency across different environments. For example, Xcode ships with a custom distribution of Git. By invoking `$ xcrun git` rather than just `$ git`, a build system can guarantee that the correct distribution is run.

3. xcodebuild

   Without passing any build settings, xcodebuild defaults to the scheme and configuration most recently used by Xcode.app:

   However, everything from scheme, targets, configuration, destination, SDK, and derived data location can be configured:

   ```
   xcodebuild -workspace NSHipster.xcworkspace -scheme "NSHipster"
   ```

   There are six build actions that can be invoked in sequence:

   `build` `analyze` `archive` `archive` `installsrc` `install` `clean`

4. genstrings

The genstrings utility generates a .strings file from the specified C or Objective-C source files. A .strings file is used for localizing an application in different locales, as described under "Internationalization" in Apple’s Cocoa Core Competencies.

```
genstrings -a \ /path/to/source/files/*.m
```

For each use of the NSLocalizedString macro in a source file, genstrings will append the key and comment into the target file. It’s up to the developer to then create a copy of that file for each targeted locale and have that file translated.

fr.lproj/Localizable.strings

```
/* No comment provided by engineer. */
"Username"="nom d'utilisateur";

/* {User First Name}'s Profile */
"%@'s Profile"="profil de %1$@";
```

5. ibtool

6. iprofiler

   iprofiler measure an app’s performance without launching Instruments.app:

```
iprofiler -allocations -leaks -T 15s -o perf -a NSHipster
```

7. xed

   This command simply opens Xcode.

```
xed NSHipster.xcworkspace
```

8. agvtool

   agvtool can be used to version Xcode projects, by reading and writing the appropriate values in the Info.plist file.

Compilation & Assembly

- clang: Compiles C, C, Objective-C, and Objective-C source files.
- lldb: Debugs C, C, Objective-C, and Objective-C programs
- nasm: Assembles files.
- ndisasm: Disassembles files.
- symbols: Displays symbol information about a file or process.
- strip: Removes or modifies the symbol table attached to the output of the assembler and link editor.

- atos: Converts numeric addresses to symbols of binary images or processes.

Processors

- unifdef: Removes conditional #ifdef macros from code.
- ifnames: Finds conditionals in C++ files.

Libraries

- ld: Combines object files and libraries into a single file.
- otool: Displays specified parts of object files or libraries. * ar: Creates and maintains library archives.
- libtool: Creates a library for use with the link editor, ld. *ranlib: Updates the table of contents of archive libraries.* mksdk: Makes and updates SDKs.
- lorder: Lists dependencies for object files.

Scripting

- sdef: Scripting definition extractor.
- sdp: Scripting definition processor.
- desdp: Scripting definition generator.
- amlint: Checks Automator actions for problems. Packages
- installer: Installs OS X packages.
- pkgutil: Reads and manipulates OS X packages. * lsbom: List contents of a bom (Bill of Materials).

Documentation

- headerdoc: Processes header documentation.
- gatherheaderdoc: Compiles and links headerdoc output.
- headerdoc2html: Generates HTML from headerdoc output.
- hdxml2manxml: Translates from headerdoc XML output to a file for use with xml2man
- xml2man: Converts Man Page Generation Language (MPGL) XML files into manual pages. Core Data
- momc: Compiles Managed Object Model (.mom) files
- mapc: Compiles Core Data Mapping Model (.xcmappingmodel) files

Third-Party Tools

appledoc

In Objective-C, the documentation tool of choice is appledoc 66. Using a Javadoc-like syntax, appledoc is able to generate HTML and Xcode-compatible .docset docs from .h files that look nearly identical Apple’s official documentation.

```
brew install appledoc
```

- @param [param] [Description]: Describes what value should be passed or this parameter
- @return [Description]: Describes the return value of a method
- @see [selector]: Provide "see also" reference to related item
- @discussion [Discussion]: Provide additional background
- @warning [description]: Call out exceptional or potentially dangerous behavior

To generate documentation, execute the appledoc command within the root directory of an Xcode project, passing metadata such as project and company name:

```
appledoc --project-name CFHipsterRef \
    --project-company "NSHipster" \
    --company-id com.nshipster \
    --output ~/Documents \
    .
```

xctool

xctool is a drop-in replacement for xcodebuild, the utility underlying Xcode.app itself.

```
brew install xctool
```

Every step of the build process is neatly organized and reported in a way that is understandable and visually appealing, with ANSI colorization and a splash of Unicode ornamentation, but xctool's beauty is not just skin-deep: build progress can also be reported in formats that can be read by other tools:

```
xctool -reporter plain:output.txt build
```

- pretty: (default) a text-based reporter that uses ANSI colors and unicode symbols for pretty output.
- plain: like pretty, but with with no colors or Unicode.
- phabricator: outputs a JSON array of build/test results which can be fed into the Phabricator code-review tool.
- junit: produces a JUnit / xUnit compatible XML -ile with test results.

- json-stream: a stream of build/test events as JSON dictionaries, one -er line (example output).
- json-compilation-database: outputs a JSON Compilation Database of build events which can be used by Clang Tooling based tools, e.g. OCLint.

OCLint

OCLint is a static code analyzer that inspects C code for common sources of problems, like empty **if/else/try/catch/finally** statements,unused local variables and parameters, complicated code with high NCSS (Non Commenting Source Statements) or cyclomatic / NPath complexity, redundant code, code smells, and other bad practices.

```
brew cask install oclint
```

```
$ xctool -workspace NSHipster.xcworkspace \
    -scheme "NSHipster" \
    -reporter json-compilation-database \
    build > compile_commands.json

$ oclint-json-compilation-database
```

xcpretty

```
gem install xcpretty
```

Nomad

```
gem install nomad-cli
```

Cupertino

Shenzhen

Houston

Venice

Dubai

CocoaPods

Installing CocoaPods

```
sudo gem install cocoapods
```

Managing Dependencies

```
$ pod init

$ pod 'X', '~> 1.1'

$ pod 'Y', :git => 'https://github.com/NSHipster/Y.git', :commit => 'b4dc0ffee'

$ pod install

$ pod update
```

Trying Out a CocoaPod

Creating a CocoaPod

NSHipsterKit.podspec

```
Pod::Spec.new do |s|
    s.name = 'NSHipsterKit'
    s.version = '1.0.0'
    s.license = 'MIT'
    s.summary = "A pretty obscure library. You've probably never heard of it."
    s.homepage = 'http://nshipster.com'
    s.authors = { 'Mattt Thompson' =>
    'mattt@nshipster.com' }
    s.social_media_url = "https://twitter.com/mattt"
    s.source = { :git => 'https://github.com/nshipster/NSHipsterKit.git', :tag => '1.0.0'
    s.source_files = 'NSHipsterKit'
end
```

A .podspec file can be useful for organizing internal or private dependencies as well:

```
pod 'Z', :path => 'path/to/directory/with/podspec'
```

Specification

Publishing a CocoaPod