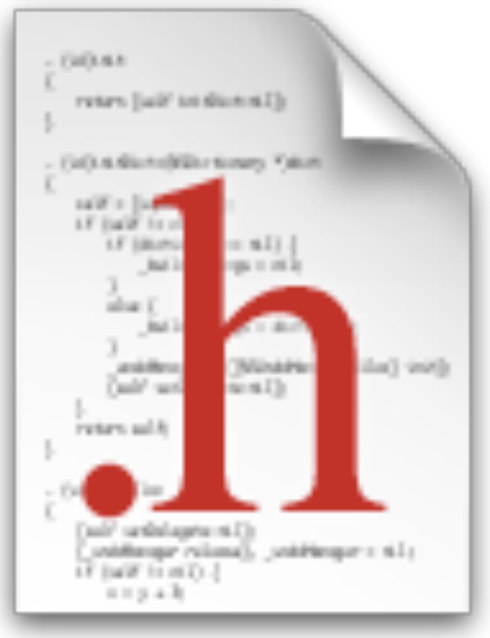
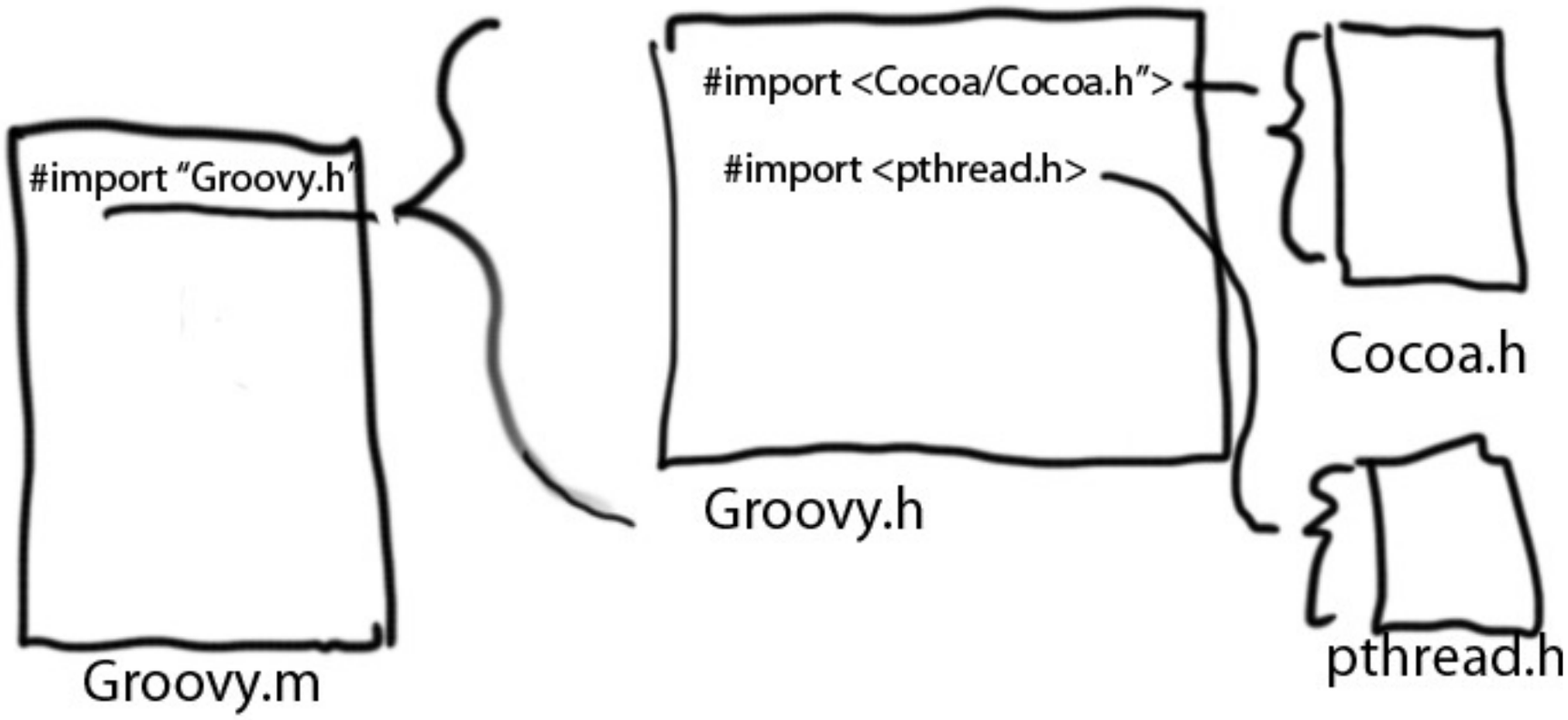


The Humble Header

Put bork line here when publishing for real

The ‘umble ‘eader

`#import “Groovy.h”`



We see them. We use them all the time. Rarely a day goes by where I don't type `#import "SomeGroovyness.h"`

Fundamentally, header files, also known as include files, are chunks of text that get pasted into our source code before it gets sent to the compiler. When the C/C++ preprocessor sees `#include "SomeGroovyness.h"` it stops what it's doing, opens `SomeGroovyness.h`, performs its preprocessing steps, which can include stopping what it's doing and opening and processing another header file. Then it blorts out all that material to the compiler and continues processing the file.

Fundamentally, that's it. It's merely a tool for getting a bunch of stuff into the pipeline before the compiler gets to our wonderful code.

Like just about everything else in programming, we people layer all sorts of traditions and conventions onto header files. That's what I'm here to talk about.

Why Are We Here?

- I'm an old curmudgeon
- One that's been cringing lately

Friday, March 23, 12

I've developed some opinions on header files over the years, influenced as a designer of APIs and libraries, and as a user. I've also been influenced by working inside of large systems. I cringe when I see something that I think is inappropriate.

Dude. They're just headers

- From a Popular Intro iPhone Book

```
#define kStateComponent    0
#define kZipComponent      1
```

Indexes for two sections of a UIPickerView

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

@interface WhateverViewController : UIViewController
@property (weak, nonatomic) IBOutlet UILabel *messageLabel;
...
- (void)updateLabelsFromTouches:(NSSet *)touches;

@end
```


First, I'd make them an enum, not a #define. enums have better compiler and debugger support. They also don't belong in the header. I as a user of this class don't care.

Similarly, yes, updateLabelsFromTouches is a useful method we'll be writing, but again as a user of this class, I don't care. Would I really be generating a synthetic set of touches to feed to this thing? Good lord i hope not

So is this stuff terrible? Of course not. But it tickled this rant.

Where did they come from?

- From the early days of C
- Advertise interfaces to (static) libraries

What's the simplest thing we can do...

Stuff that would be in the interfaces

for the user; what are the functions, what are the available data types? what are the available structures or opaque types

for the computer: what is the type, order, and size of parameters? Contents and size of a struct ? Practical nuts and bolts "how do I glom this and your code together.

Are they still useful?

- Non-interesting question for Fruit programmers
- Newer languages (Java / C#) look inside of libraries for the interfaces *and not-so-newer, like PL/I*
- I like comments in header more than Header/ JavaDoc *I know, I'm weird*
- Sometimes the best Apple docs are in the headers

PL/I has a PACKAGE statement, for instance

What are they ***for***?

- Communication
  - With the compiler *how many bytes do I deal with?*
  - With other ugly giant bags of mostly water

what is this and how do I use it?

I like to keep them minimal

My ideal header

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

@class GRIndoorCyclingDensitometer;

// Groovy is a class about ...
@interface Groovy : NSObject

// User's greeblation threshold, in milli-marklars
@property (strong, nonatomic) NSString *greeblation;

// UI stuff
@property (weak, nonatomic) IBOutlet UILabel *uberDisplayLabel;
- (IBAction) lanchNukes: (id) sender;

// The user's marklar density can affect the sub-greeblation threshold.
- (void) setDdensitometer: (GRIndoorCyclingDensitometer *) denselWashington;

@end // Groovy
```


It's a thing of beauty. Nothing extraneous. It has all the information in it. Comments on what it is. How to use it. No extra header files

What are they for?

- The public interface for the class

This is my most important point. Headers are for the public interface. Anything in the header which doesn't have to be there syntactically, or isn't part of the public interface, should not be there.

For each line, think "does this line honor the reader's time, and does it convey useful information for using this class".

I have enough to do, and would rather not slog through data I don't need

Everything before this is colored by this thought, and everything after.

Headers for Unit Tests

GronkTest.h

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

@interface GronkTest : SenTestCase

@end
```

GronkTest.m

```
#import “GronkTest.h”

@implementation GronkTest

@end
```


So, let’s apply that logic to unit test headers.
When you make a new unit test, you get this header and a stub implementation file.
You can make setup, teardown, and test methods. But you never touch the header.

You might add instance variables. Is this part of the public interface? Frankly, will anybody be using this class directly? No. The testing framework grovels around in the objC runtime. So this header file has no use.

Back at GOOG, we had test classes for every class, and had an explosion of these little turd header files. After awhile we just got rid of them. And today that’s what I do.

BUILD -> HEADER DISAPPEARS

More Testing

- Public and private headers for your class

Sometimes you need a back-channel to your class. Put those into a separate header file. Groovy.h and Groovy-Private.h. That way day-to-day users of your class won't be exposed to, and bothered by, stuff that's implementation details.

Breaking Cycles

.. Cycling Class.h

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

#import “GRIndoorCyclingRideSegment.h”

@interface GRIndoorCyclingClass : NSObject
...
- (GRIndoorCyclingRideSegment *) segmentAtIndex: (int) blah;
@end
```

.. Ride Segment.h

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

#import “GRIndoorCyclingClass.h”

@implementation GRIndoorCyclingRidesegment : NSObject
...
- (GRIndoorCyclingClass *) owningClass;

@end
```


Cycles always seem to be a problem in computation. Look at retain cycles. The multiple inheritance diamond. A popular interview question is detecting loops in a linked list. Stuff modeled with directed acyclic graphs (DAGs). Cyclic graphs cause problems.

They can happen with header files.

Usually this is the point someone would ask a mentor or stack overflow “how do I include A.h nto B.h and B.h into A.h – not the right question.

Ask “what does the compiler need to know”. It deals with bits and bytes.

Broken Cycles

.. Cycling Class.h

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

@class GRIndoorCyclingRideSegment;

@interface GRIndoorCyclingClass : NSObject
...
...
- (GRIndoorCyclingRideSegment *) segmentAtIndex: (int) blah;
@end
```

.. Ride Segment.h

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

@class GRIndoorCyclingClass;

@implementation GRIndoorCyclingRidesegment : NSObject
...
- (GRIndoorCyclingClass *) owningClass;

@end
```


Use @class to tell the compiler “this is an objective C object. Pointer size is the same size as other stuff on the system. four bytes on iOS. The compiler now knows enough for the return types, and can go on its merry way.

Notice no header file included. So no need to process them.

Multiple Inclusion

- #import is officially deprecated by general gcc
  - Therefore a non-portable directive
- A file #included multiple times is processed multiple times.

This can be bad

Let's explore that other option.

Multiple Inclusion

- Groovy.m includes goop.h
- Groovy.m includes UIColor+Foobage.h, which includes goop.h

```
typedef struct goop {
    int blah;
} goop;
```

goop.h

```
...
// From goop.h directly.
typedef struct goop {
    int blah;
} goop;
...
// From goop.h from UIColor+Foobage.h
typedef struct goop {
    int blah;
} goop;
...
```

Groovy.m, post-preprocessing

NEXT SLIDE-> compiler error

Yay! Errors!

```
error.m:7:16: error: redefinition of 'goop'
typedef struct goop {
               ^
error.m:3:16: note: previous definition is here
typedef struct goop {
               ^
1 error generated.
```


The compiler is not happy.

Include Guards

```
#ifndef GOOP_H_INCLUDED
#define GOOP_H_INCLUDED

typedef struct goop {
    int blah;
} goop;

#endif // GOOP_H_INCLUDED
```

```
// Copyright (c) 2009 The Chromium Authors. All rights reserved.
// Use of this source code is governed by a BSD-style license that can be
// found in the LICENSE file.

#ifndef APP_CLIPBOARD_CLIPBOARD_H_
#define APP_CLIPBOARD_CLIPBOARD_H_

#include <map>
#include <string>
#include <vector>
```


This is an elegant, yet gross solution. You have to pepper your header files with this boilerplate.

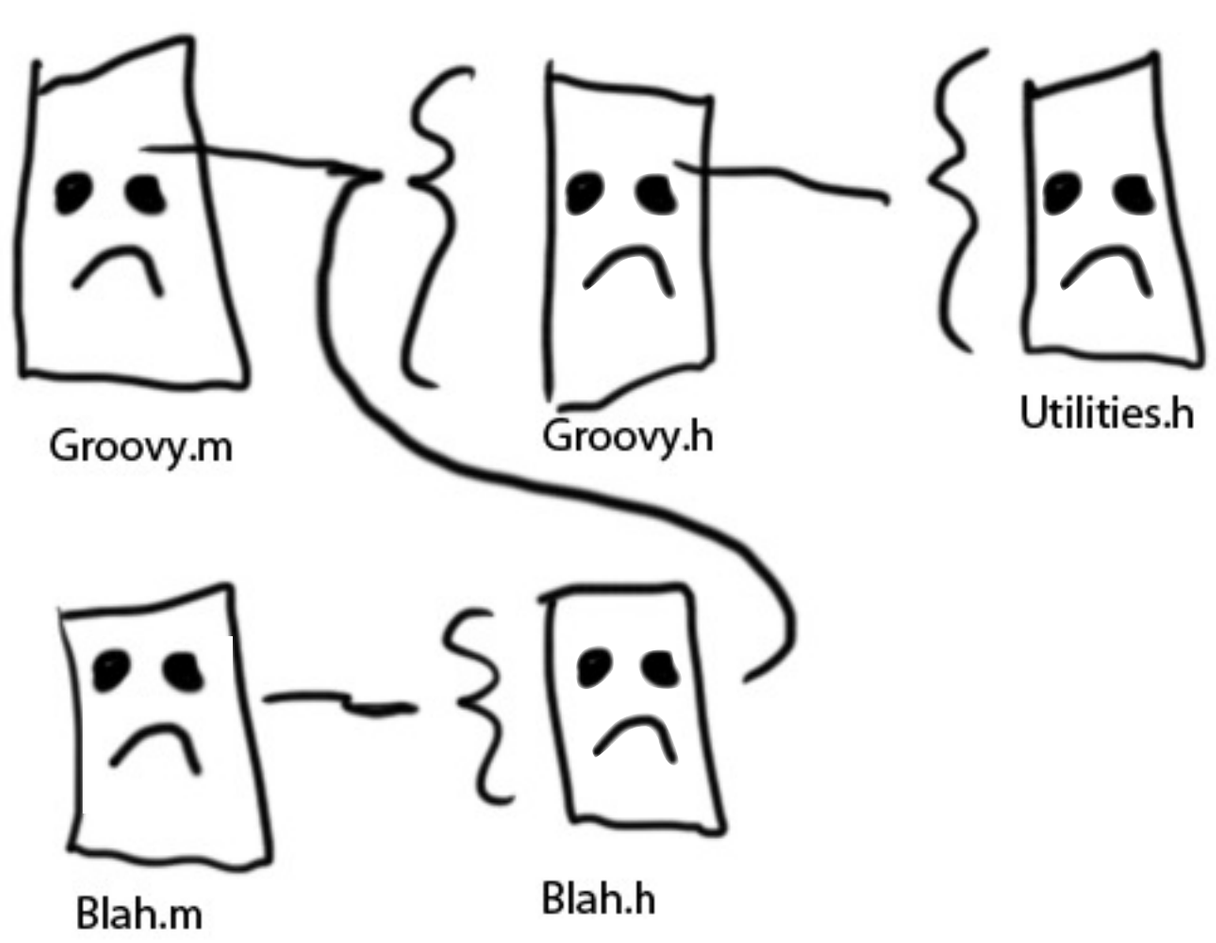
The compiler still has to scan over everything, though, just to be safe. I believe modern compilers can optimize out some scans, but I I don't trust it to optimize it 100% of the time.

Big problem is if two headers have the same include nomenclature.

Big projects, like chromium, encode the directory hierarchy in the include guard. -> BUILD

Compile Time Dependencies

- You pay a price for including headers you don't need



Friday, March 23, 12

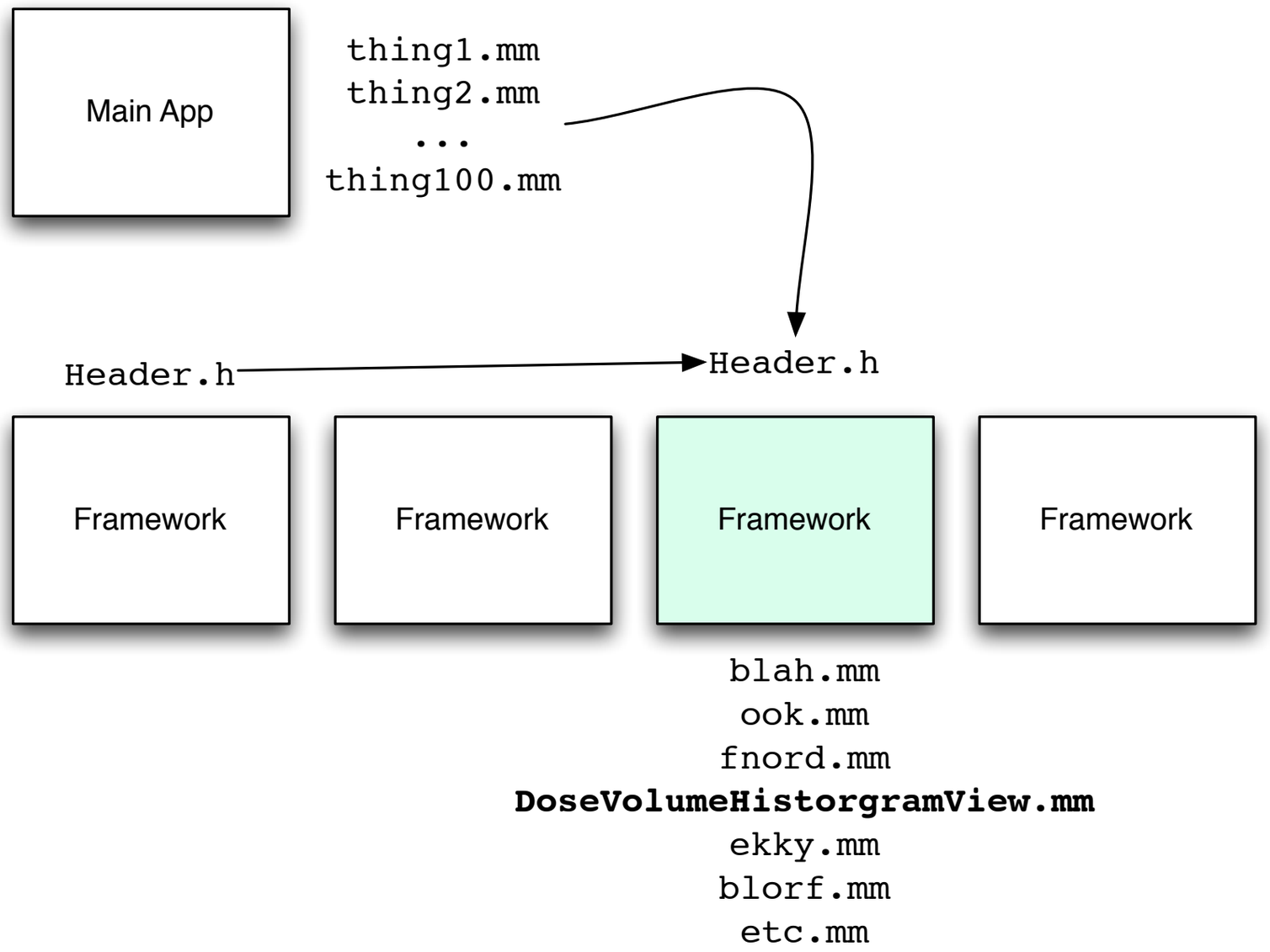
OK, so I've been harping on minimal inclusion, using @class and similar forward references. As well as the joys of #import over #include + header guards. I like to believe it's not just technical bloviation.

BUILD -> 4 SAD FACES FOR DEPENDENCIES

This is what's known as dependency management. Xcode and makefiles. makefile-replacements and makefile-replacement-replacements. Like there's make, then Ant which replaces makefiles, then Buildr which replaces ant

NEXT-> mpop / corvus

Why can this be a problem?



mpop story

Corvus story

Precompiled Headers

```
//
// Prefix header for all source files of the 'TinyPix' target
// in the 'TinyPix' project
//

#import <Availability.h>

#ifndef __IPHONE_5_0
#warning "This project uses features only available in iOS SDK 5.0 and later."
#endif

#ifdef __OBJC__
    #import <UIKit/UIKit.h>
    #import <Foundation/Foundation.h>
#endif
```


22,000 lines in uikit + foundation.
Put big, seldomly changing stuff here.
everything implicitly depends on it
Don’t put utiliities.h in here - will cause *massive* recompilation.

Order of #imports

```
// Header for this class first
#import "Groovy.h"

// Any system headers
#import <AVFoundation/AVFoundation.h>
#import <pthread.h>

// Other project headers
#import "GRFroopyViewControllerContollerView.h"
#import "GRMarklar.h"
#import "GRUtilities.h"

@implementaton Groovy
...
```

Groovy.m

Ordering of #imports borders on religious, since there's no real reason for a particular order. This is what I do. And I sort alphabetically within each section

The important thing is including the header for the class first , to make sure your headers can compile cleanly when other code includes them.

Unintentional breakage

```
#import <UIKit/UIKit.h>
@interface Groovy : NSObject
- (void) setPlayer: (AVAudioPlayer *) player;
@end
```

Groovy.h

```
#import <AVFoundation/AVFoundation.h>
#import “Groovy.h”
```

Groovy.m

```
#import “Groovy.h”
```

Snood.m

```
Compiler says “Dude!  What is this AVAudioPlayer thing?
```


BUILD -> GROOVY.m
BUILD -> SNOOD.m

Habits to Break

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

@interface Groovy : NSObject {
@private
  int this;
@protected
  int that;
@public
  int theOther;
}

...
// Private member functions
- (void) updateUI;
- (void) tweakTableView;

// Overrides
- (void) dealloc;
- (void) tableView:numberOfRowsInSection:
@end
```


I've seen unrepentant C++ programmers do this kind of stuff - way too much information. @private etc are toothless very easily circumvented with KVC. Just put the ivars into your implementation, unless you're expecting subclasses to access them. In that case you might not want to do that either.

Like I mentioned earlier, "private" methods are not really. No need to advertise them.

Same with overrides. I don't want to have to recompile my world just because you decided you're overriding something else and insist on having it in the header

Boilerplate! Lame!

```
/* FUNCTION NAME:
 * ARGUMENTS:
 * RETURN:
 * DESCRIPTION:
* SIDE-EFFECTS:
* CAVEATS:
* BY SOMELOSER ON 10/15/90
*/
```


I used to do contracting with a guy that would paste this before every function he wrote. It was on wall street, and I believe he got paid by the line. It was even uglier with a line of stars and borders and whatnot.

Most of the comments were DailyWTF category. “sortArray takes an array and sorts it”, but a lot of boilerplate that got in the way.

Xcode Header Templates

```
//
//  BIDAppDelegate.h
//  Borkinator
//
//  Created by markd on 10/14/11.
//  Copyright (c) 2011 __MyCompanyName__. All rights reserved.
//
```

Friday, March 23, 12

You can probably guess this gets the same treatment.
Filename is ok. But you can look at your editor to see what file it is. Xcode's refactoring doesn't touch it, so it's easy for it to get out of sync with the code. Like I need someting else to keep up with.
The project name is pretty worthless. If you copy the file to another project, does it matter where it came from. If you reference a file, you can see in the xcode file pane. what it's from. Plus if the project changes names? I've worked on some things that have changed name and direction 3 or 4 times. Does this need to be updated?
Creation user and date is worthless. That's what source code control is for. If you're one man project, you know who made it. In larger projects, the dude that created it is not the dude that maintains it. So more stale information.
The Copyright line is about the only useful line.
So one out seven lines is waste, and just makes you scroll more.

#endif

- Headers are for communication
- Keep them simple and minimal
- Put everything else into the implementation