

# Metaprogramming in Objective-C

Richard Warren



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Who Am I?

## Software Engineer:

- Background in DoD funded R&D
- Now focused on iOS apps

## Technical Writer:

- Creating iOS 5 Apps
- Objective-C Bootcamp
- Many MacTech Articles

## Technical Trainer:

- Senior trainer at About Objects
- Also taught classes for Future Media Concepts and MacAmerica



# Freelance Mad Science Labs

iOS Development, Technical Writing, Training and Consulting



# What is Metaprogramming?



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Wikipedia

*Metaprogramming is the writing of computer programs that write or manipulate other programs (or themselves) as their data, or that do part of the work at compile time that would otherwise be done at runtime. In some cases, this allows programmers to minimize the number of lines of code to express a solution (hence reducing development time), or it gives programs greater flexibility to efficiently handle new situations without recompilation.*



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Metaprogramming 2

Just as programming uses computers to solve real-world problems, metaprogramming uses the programming language itself to solve common programming problems.

We're writing code to help us write code.



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# How is it different from programming?

- Uses advanced techniques
  - Highly dynamic changes to classes and methods at runtime
  - Heavy use of introspection
- Provides a general solution for a broad range of problems
- Emphasizes convention over configuration
- Often feels like we're modifying the programming language itself.



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Techniques



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Introspection



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# NSObject

- isKindOfClass:
- respondsToSelector:
- conformsToProtocol:
- methodForSelector:
- etc.



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Objective-C Runtime

- objc\_getClassList()
- class\_copyIvarList()
- class\_copyMethodList()
- class\_copyPropertyList()
- class\_copyProtocolList()



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# We Can Go Deeper

- `method_getName()`
- `method_getImplementation()`
- `method_getTypeEncoding()`
- `method_getReturnType()`
- `method_getNumberOfArguments()`
- `method_getArgumentType()`



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# What Can We Do With It?

- Get the list of classes
- Look for classes whose name matches a specific pattern.
- Inside those classes, look for methods whose names match another pattern.
- Call those methods.



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Monkeying With Methods



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Categories

```
@interface NSMutableArray  
(FMSSStack)
```

- (void)FMS\_push:(id)obj;
- (id)FMS\_pop;
- (id)FMS\_peek;

```
@end
```



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



```
@implementation NSMutableArray (FMSSStack)

- (void)FMS_push:(id)obj {
    [self addObject:obj];
}

- (id)FMS_pop {
    id last = [self lastObject];
    [self removeLastObject];
    return last;
}

- (id)FMS_peek {
    return [self lastObject];
}

@end
```



# Freelance Mad Science Labs

iOS Development, Technical Writing, Training and Consulting

# Adding Methods at Runtime

BOOL class\_addMethod(Class cls, SEL name, IMP imp, const char \*types)

```
void updater(id self, SEL _cmd) {  
    NSLog(@"Calling method %@ on %@",  
          NSStringFromSelector(_cmd),  
          [self class]);  
}
```

```
success =  
class_addMethod([self class],  
                NSSelectorFromString(updateName),  
                updater,  
                "v@:");
```



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Swizzling Methods

- IMP class\_replaceMethod(Class cls, SEL name, IMP imp, const char \*types)
- void method\_exchangeImplementations(Method m1, Method m2)
- IMP method\_setImplementation(Method method, IMP imp)



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Method Swizzling

```
void swizzle(Class class, SEL originalSelector, SEL newSelector) {
    Method originalMethod = class_getInstanceMethod(class, originalSelector);
    Method newMethod = class_getInstanceMethod(class, newSelector);

    IMP originalImplementation = method_getImplementation(originalMethod);
    IMP newImplementation = method_getImplementation(newMethod);
    const char * type = method_getTypeEncoding(originalMethod);

    if(class_addMethod(class, originalSelector, newImplementation, type)) {
        class_replaceMethod(class,
                            newSelector,
                            originalImplementation,
                            type);
    } else {
        method_exchangeImplementations(originalMethod, newMethod);
    }
}
```



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



```
- (void)traceFirstName {  
  
    // Do my pre-actions  
    NSLog(@"Beginning firstName");  
  
    // The call the original  
    [self traceFirstName];  
  
    // Do my post-actions  
    NSLog(@"Ending firstName");  
}  
  
+(void)load {  
    swizzle([Person class],  
            @selector(firstName),  
            @selector(traceFirstName));  
}
```



# Freelance Mad Science Labs

iOS Development, Technical Writing, Training and Consulting

# Dynamic Subclassing

- Also called class swizzling
- Dynamically create a subclass
- Change an object's type to the subclass
- Now any method swizzling only affects that instance



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Typical Implementation

```
Class cls = objc_allocateClassPair(startingClass,  
                                   className,  
                                   0);  
  
objc_registerClassPair(cls);  
object_setClass(self, cls);
```



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Method Dispatch

- + resolveInstanceMethod:
- + resolveClassMethod:
- - (id)forwardingTargetForSelector:(SEL)aSelector
- - (void)forwardInvocation:(NSInvocation \*)anInvocation
- - (void)doesNotRecognizeSelector:(SEL)aSelector



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Magic Methods

- Useful when you need to implement a large number of very similar methods
- Dynamically implement methods based on a known naming convention
- NSObject's accessor methods



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Block Shenanigans



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Hierarchy of Block Mastery

- Using Apple's block-based APIs
- Creating methods that take block arguments
- Creating methods that return blocks
- Creating methods that have block arguments which in turn have block arguments
- And down the rabbit hole we go...



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Configuration Blocks

```
- (id) initWithConfigurationBlock:(void (^) (id _self))
configBlock {

    self = [super init];
    if (self != nil) {

        // perform default initialization here

        configBlock(self);

        // perform validation here
    }

    return self;
}
```



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Blocks that Limit Access

- Blocks can capture any values in scope.
- Let the blocks capture a local variable
- Store the blocks in a property



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



```

- (id)init {
    self = [super init];
    if (self != nil) {

        // I can only access value using
        // self.increment() or self.getter()
        // Nothing else in my class can touch this.
        __block NSUInteger value = 0;

        _inc_block = ^{
            value++;
        };

        _get_block = ^{
            return value;
        };
    }
    return self;
}

- (void)increment {
    self.inc_block();
}

- (NSUInteger)currentValue {
    return self.get_block();
}

```



# Freelance Mad Science Labs

iOS Development, Technical Writing, Training and Consulting

# Blocks and Methods

- IMP imp\_implementationWithBlock(id block)
- id imp\_getBlock(IMP anImp)
- BOOL imp\_removeBlock(IMP anImp)



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Case Studies



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# RWUserDefaultsManager

- A defaults manager that automates syncing UserDefaults using iCloud Key Value Storage
- We declare a property for each piece of data we wish to manage
- Use the property's getters and setters to access the data
- Behind the scenes, the manager sends updates to iCloud and responds to incoming changes from iCloud
- We don't want to write new getters and setters every time we add a new property



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Introspection and Magic Methods

- Defaults manager looks for any dynamic properties
- Automatically creates the getter, setter and update methods for these properties



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Sample Code

```
@interface DefaultManager : RWUserDefaultsManager
@property (assign)NSString* name;
@property (assign)NSDate* date;
@end
```

```
@implementation DefaultManager
@dynamic name;
@dynamic date;
@end
```



## Freelance Mad Science Labs

iOS Development, Technical Writing, Training and Consulting



# FMSAlertManager

- Block-based wrapper for UIAlertView
- Allows us to create buttons and define their actions in one place
- Loosely based on configuration blocks
- End up with 3-layers of nested blocks



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Usage

```
[[FMSAlertManager sharedAlertManager]
 showAlertWithTitle:@"One Button Alert"
 message:@"This is an alert with a single button"
 autoClose:YES
 configurationBlock:^(AddButtonBlock addButton) {

    addButton(@"Cancel", ^{
        NSLog(@"One Button -- Cancel Button Pressed!");
    });

}];
```



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# FMSSwizzler

- Block-based API for method swizzling and class swizzling
- + (void)FMS\_aliasInstanceMethod:(SEL)originalSelector newSelector:(SEL) newSelector;
- + (void)FMS\_replaceInstanceMethod:(SEL)methodSelector withImplementationBlock:(id)block;
- + (void)FMS\_overrideInstanceMethod:(SEL)selector oldSelector:(SEL)oldSelector implementationBlock:(id)block;
- + (FMSPseudoPropertyAdder)FMS\_generatePseudoPropertyAdderForType: (FMSPseudoPropertyType)type;
- - (void)FMS\_dynamiclySubclass;



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



# Resources

- RWUserDefaultsManager (Old code):  
[www.freelancemadscience.com](http://www.freelancemadscience.com)
- FMAlertManager:  
<https://github.com/rikiwarren/FMAlertManager>
- FMSSwizzler:  
<https://github.com/rikiwarren/FMSSwizzler>



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting

# Other Resources

- Method Replacement for Fun and Profit  
<http://www.mikeash.com/>
- Understanding the Objective-C Runtime  
<http://cocoasamurai.blogspot.com/>
- objc\_msgSend() Tour Parts 1 - 4  
<http://www.friday.com/bbum/>



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting



**Richard Warren**

rich@freelancemadscience.com

<http://freelancemadscience.com>

@rikiwarren

+Rich Warren

Questions?



**Freelance Mad Science Labs**

iOS Development, Technical Writing, Training and Consulting