Objective-C style guide

1. Visual Style

1.1. Whitespace

Indent with Spaces

Use groups of 4 spaces (instead of tabs) to denote different indentation levels.

Xcode does this by default. Double-check your settings in: `Preferences -> Text Editing -> Indentation`.

Spaces in Declarations

In `@interface` declarations, there should be one space between: the subclass name; the colon symbol; the superclass name; the adopted protocols section; any adopted protocols.

```
    // DO
    @interface PXProtocolAdoptingClass : NSObject <PXProtocolA, PXProtocolB>

    // DON'T
    @interface PXProtocolAdoptingClass:NSObject <PXProtocolA, PXProtocolB>
    @interface PXProtocolAdoptingClass : NSObject<PXProtocolA, PXProtocolB>
    @interface PXProtocolAdoptingClass : NSObject <PXProtocolA,PXProtocolB>
```

In `@property` declarations, there should be one space between: the `@property` keyword; the property attributes section; any property attributes; the property type; and the pointer asterisk.

```
    // DO
    @property (nonatomic, weak) UIView<DelegateProtocol> *delegate;

    // DON'T
    @property (nonatomic,weak) UIView <DelegateProtocol> *delegate;
    @property(nonatomic, weak) UIView <DelegateProtocol> *delegate;
    @property (nonatomic, weak) UIView<DelegateProtocol>*delegate;
```

In *method* declarations, there should be one space between: the `-` or `+` character and the `(returnType)`; an argument type and its pointer asterisk.

```
    // DO
    - (void)doSomethingWithString:(NSString *)string number:(NSNumber
*)number

    // DON'T
    - (void)doSomethingWithString:(NSString*)string number:(NSNumber *)number
    -(void)doSomethingWithString:(NSString *)string number:(NSNumber *)number
    - (void)doSomethingWithString:(NSString *)string  number:(NSNumber
*)number
```

```
    - (void) doSomethingWithString:(NSString *) string  number:(NSNumber *)
number
```

In *generic class* declaration, there should be no spaces before bracket:

```
    // DO
    @interface PXStoreObjectModel<__covariant ObjectType:PXStoreObject *> :
NSObject <PXStoreModel, PXStoreObservableObject>

    // DON'T
    @interface PXStoreObjectModel <__covariant ObjectType:PXStoreObject *> :
NSObject <PXStoreModel, PXStoreObservableObject>
```

1.2. Brackets

Opening brackets should be on the same line as the statement they refer to. Closing brackets should be on their own line, except when followed by `else`.

```
    // DO
    - (void)doSomethingWithString:(NSString *)string {
        if (condition) {
            ...
        } else {
            ...
        }
    }

    // DON'T
    - (void)doSomethingWithString:(NSString *)string
    {
        if (condition)
        {
            ...
        }
        else
        {
            ...
        }
    }
```

Always use brackets even when the conditional code is only one statement.

```
    // DO
    if (condition) {
        return;
    }

    // DON'T
    if (condition)
        return;
```

1.3. Line Wrapping

Hard wrap lines that exceed 140 characters. You can configure the column guide on *Xcode*: `Preferences -> Text Editing -> Page guide at column: 140.`

When hard wrapping method calls, give each parameter its own line. Align each parameter using the colon before the parameter (*Xcode* does this for you by default).

```
// DO
- (void)doSomethingWith:(Foo *)foo
                   rect:(NSRect)rect
               interval:(float)interval {
    ...
```

Method invocations should be formatted much like method declarations. Invocations should have all arguments on one line or have one argument per line, with colons aligned.

```
// DO
[myObject doFooWith:arg1 name:arg2 error:arg3];

[myObject doFooWith:arg1
               name:arg2
              error:arg3];

// DON'T
[myObject doFooWith:arg1 name:arg2
              error:arg3];

[myObject doFooWith:arg1
          name:arg2
          error:arg3];
```

1.4. Newlines

Use exactly one empty line to separate:

- The copyright header and the `#import` section
- The `#import` section and NS_ASSUME_NONNULL_BEGIN macro
- NS_ASSUME_NONNULL_BEGIN and the class `@interface` or `@implementation` (or associated forward declarations)
- `@interface` or `@implementation` and `@end`
- Different `@interface` or `@implementation` sections within the same file.
- Groups of related #import statements
- Groups of related statements in a single method implementation

Do not use one or more empty lines in any other cases.

Header and *implementation* files must have one, and only one, trailing empty line.

Header File Example

```
    //
    //  Created by Makarov Yury on 24/04/16.
    //  Copyright © 2016 Joom. All rights reserved.
    //

    #import "ClassA.h"
    #import "ProtocolA.h"
    #import <Foundation/Foundation.h>

    NS_ASSUME_NONNULL_BEGIN

    @protocol ProtocolB;
    @class ClassB;

    @interface MyClass : ClassA <ProtocolA>

    - (instancetype)initWithParameter1:(ClassB *)parameter1
parameter2:(id<ProtocolB>)parameter2 NS_DESIGNATED_INITIALIZER;

    - (instancetype)init NS_UNAVAILABLE;

    @end

    NS_ASSUME_NONNULL_END
```

1.5. ReactiveCocoa operators

All operators should have 4 spaces indent:

```
        // DO
        RACSignal *versionChecked = [[[[[[[RACObserve(self.deviceModel,
configuration)
        ignore:nil]
        map:^id(PXDeviceConfiguration *config) {
            return @(config.updateStatus);
        }]
        distinctUntilChanged]
        doNext:^(NSNumber *status) {
            DDLogInfo(@"Update status: %@",
NSStringFromVersionUpdateStatus(status.integerValue));
        }]
        filter:^BOOL(NSNumber *value) {
            return value.integerValue != PXUpdateStatusUpToDate &&
value.integerValue != PXUpdateStatusUnknown;
        }]
        deliverOnMainThread]
        replayLazily];

    [[[[[[RACSignal
        empty]
        deliverOn:self.scheduler]
```

```
         concat:waitForRequiredData]
         concat:[self.eventProcessor beginProcessing]]
         takeUntil:halt]
         subscribe];

        // DON'T
        RACSignal *versionChecked = [[[[[[[RACObserve(self.deviceModel,
configuration)
       ignore:nil]
       map:^id(PXDeviceConfiguration *config) {
           return @(config.updateStatus);
       }] distinctUntilChanged]
       doNext:^(NSNumber *status) {
           DDLogInfo(@"Update status: %@",
NSStringFromVersionUpdateStatus(status.integerValue));
       }] filter:^BOOL(NSNumber *value) {
           return value.integerValue != PXUpdateStatusUpToDate &&
value.integerValue != PXUpdateStatusUnknown;
       }] deliverOnMainThread]
       replayLazily];

        [[[[[[RACSignal empty]
         deliverOn:self.scheduler]
         concat:waitForRequiredData]
         concat:[self.eventProcessor beginProcessing]]
         takeUntil:halt]
         subscribe];
```

In case there's only one operator used, indent is optional:

```
RACSignal *userSignal = [RACObserve(userModel, user) ignore:nil];
```

Put a single `subscribe` on the same line with the source signal:

```
[signal subscribeNext:^(SomeObject *input) {
        // ...
}];
```

2. Code Style

2.1. Variable Declarations

Declare one variable per line even if they have the same type.

When declaring pointers, there should be a space between the asterisk and the variable type, but none between the asterisk and the variable:

```
    // DON'T
    int* variablePointer2;
    int* variablePointer, variable;
```

Always declare properties instead of ivars:

```
// DO
@interface MyClass ()

@property (nonatomic, strong, readonly, nullable) NSObject *object;

@end

// DON'T
@implementation MyClass {
        NSObject *_object;
}

@end
```

When declaring pointers, there should be a space between the asterisk and the variable type, but none between the asterisk and the variable.

```
    // DON'T
    int* variablePointer2;
    int* variablePointer, variable;
```

2.2. Forward Declarations

Use one line for each forward declarations:

```
    // DO
    @protocol forwardProtocol;
    @class aClass;
    @class anotherClass;
    @class yetAnotherClass;

    // DON'T
    @protocol forwardProtocol, anotherProtocol;
    @class aClass, anotherClass, yetAnotherClass;
```

2.3. Dot Notation

Always use dot notation when accessing properties:

```
    // DO
     [self someMethod:self.someValue];

    // DON'T
     [self someMethod:_someValue];
```

Dot notation should be used when accessing properties, but should not be used to invoke regular methods.

2.4. Property declaration

Property declarations should always precede method declarations.

Property attributes should be mentioned in the same order throughout the project.

Never omit storage attribute (strong/weak/etc). Never omit readonly and always omit readwrite specifier.

Always use `nullable/null_resettable` attributes for the corresponding properties.

Use the following options to declare a property:

```
    // DO
    @property (nonatomic, strong) NSObject *object;
    @property (nonatomic, strong, readonly) NSObject *object;
    @property (nonatomic, strong, readonly, nullable) NSObject *object;
    @property (nonatomic, assign, getter=isEnabled, readonly) BOOL enabled;

    // DON'T
    @property NSObject *object;
    @property (strong, readonly) NSObject *object;
    @property (strong, readonly, nullable, nonatomic) NSObject *object;
    @property (readonly, strong) NSObject *object;
    @property (nonatomic, assign, getter = isEnabled, readonly) BOOL enabled;
```

2.5. Imports

Includes in " " should be listed first.

```
#import "ClassA.h"
#import "ProtocolA.h"
#import <BlocksKit/BlocksKit.h>
#import <Foundation/Foundation.h>
```

2.6. NS_ASSUME_NONNULL_BEGIN/END

Each interface declaration containing pointers should include NS_ASSUME_NONNULL_BEGIN/END macro.

2.7. Designated initializers

Each designated initializer should be marked with NS_DESIGNATED_INITIALIZER. Initializers inherited from superclass, which are not either designated or convenience initializers, should be marked as unvavailble with NS_UNAVAILABLE.

2.8 Constants

```
// local constant
static const CGFloat kToolbarHeight = 44.0;

// global constant declaration
```

```
extern const CGFloat PXToolbarHeight;
```

2.9 Static functions

```
// local static function
static void someFunction() {
        // ...
}

// global function declaration
extern void PXSomeFunction();
```

2.10 Allocation/initialization

Be consistent in object allocation - always use alloc/init:

```
// DO
UIView *view = [[UIView alloc] init];


// DON'T
UIView *view = [UIView new];
```

2.11 Delegates

Declare delegate protocol below the class interface.

Each delegate method should contain a caller as the first argument:

```
// DO
@class PXObject : NSObject

@property (nonatomic, weak, nullable) id<PXDelegate> delegate;

@end

@protocol PXDelegate <NSObject>

- (void)object:(PXObject *)object didFinishTaskWithError:(NSError *)error;

@end


// DON'T
@protocol PXDelegate <NSObject>

- (void)didFinishTaskWithError:(NSError *)error;

@end
```

```
@class PXObject : NSObject

@property (nonatomic, weak, nullable) id<PXDelegate> delegate;

@end
```