# Expressive ReactiveCocoa

Dave Lee @kastiglione

## Acts

1. Start with `return`
2. Signals at the Core
3. Ultimately Expressive

## Start With `return`

## Callbacks

`return` is simple, too simple.

Necessity gave rise to callbacks.

## Callbacks

### Target-Action

```
// Setup
[button addTarget:self action:@selector(buttonTapped:)];

- (void)buttonTapped:(id)sender {
    // Callback
}
```

## Callbacks

### Delegates

```
// Setup
tableView.delegate = self;

- (void)tableView:(UITableView *)tv didSelectRowAtIndexPath:(NSIndexPath *)ip {
    // Callback
}
```

## Callbacks

### Blocks

```
[NSURLConnection
    sendAsynchronousRequest:r
    queue:q
    completionHandler:^(NSURLResponse *r, NSData *d, NSError *e) {
        // Callback for all occasions
    }];
```

## Callbacks

Iteration:

```
[xs enumerateObjectsUsingBlock:^(id o, NSUInteger i, BOOL *s) {
    // Callback for all objects
}];
```

## Callbacks

### Notifications & KVO (COMEFROMs)

```
[NSNotificationCenter.defaultCenter
    addObserverForName:n
    object:o
    queue:q
    usingBlock:^{
        // Callback
    }];


[target addObserver:self forKeyPath:keyPath options:0 context:NULL];

- (void)observeValueForKeyPath:(NSString *)kp ofObject:(id)o change:(NSDictionary *)d context:(void *)c {
    // Callback
}
```

## Callbacks

Even basic functions and methods.

```
// Using default result and return addresses:
CGFloat maxY = CGRectGetMaxY(frame);

// Explicit result and return address:
CGRectGetMaxY(frame, &maxY, $instructionPointer);
```

## Callbacks

### Hidden returns

```
// Two returns are better than one?
dispatch_async(queue, ^{});
```

## Callbacks

### Callback/return overload:

```
[manager
    GET:path
    parameters:parameters
    success:^(AFHTTPRequestOperation *operation, id responseObject) {
        // Success begets more success
    }
    failure:^(AFHTTPRequestOperation *operation, NSError *error) {
        // Error callback
    }];
```

## Callbacks

Dimensions: Time and Events

Time: sync, async (hybrid)
Events: values, completion or error

## Callbacks

Why do we have so many types of callbacks?

The ubiquitous `return` is fundamentally inadequate. How do we compose anything with so many different ways to return values.

A leaky abstraction we never really notice but influences so much of the way we compose our software.

All problems in computer science can be solved by another level of indirection.

— David Wheeler

## Callbacks

The indirection:

Instead of supplying callbacks as input *(selector, delegate, or block)*, return a "callback" object to the caller. Or, since names are always better with manager appended, let's call it a callback manager.

## Callbacks

Ok, callback managers, what now?

Before we go there, let's get to ReactiveCocoa…

## Signals

## Signals

ReactiveCocoa's core fundamental concept is `RACSignal`.

Signals are the grand unified theory of callbacks.

Signals subsume promises.

## Signals

Staying informed through subscription:

```
[signal subscribeNext:^(id value) {
    // Data handling
} error:^(NSError *error) {
    // Error handling
} completed:^{
    // Completion handling
}];
```

## Signals

Subscribing to what you need:

```
[fileWriteSignal subscribeError:^(NSError *error) {
    // Error handling
} completed:^{
    // Completion handling
}];

[cacheQuerySignal subscribeNext:^(id result) {
    // Result handling
}];
```

## Signals

Signals all around us

Networking, user input, app life cycle, queries, timers, long running computations.

## Signals

What have we gained?

1. Consistency
2. Refactored for responsibility
3. Time and event agnosticism
4. A new abstraction

## Signals at the Core

## Signal Operations

What can we do with signals?

## Signals Operations

Signals have zero or more values, and end with completion. Or error.

Sound like anything?

## Signals Operations

Simplistic but helpful analogy: lists.

Just your normal every day lists. With laziness, and asynchronicity, and errors.

## Signals Operations

Well known list operations: creation, concatenation, mapping, filtering, subdividing, etc.

Nothing to learn here.

## Signals Operations

More natural to callbacks, flow control operations:

`-distinctUntilChanged`, `-deliverOn:`, `-throttle:`, `-delay:`, `-repeat`, `-retry`.

## Signals Operations

Signals working together:

`-concat:`, `-sample:`, `-zip:`, `+combineLatest:`, `-catch:`...

## Signal Operations

Operations on signals return new signals, which can be further operated on, ad infinitum.

## Signals Operations

If normal signals are like lists, then signals of signals are like trees.

Take for example, scrolling a list of gifs.

## Signal Operations

Trees share some operations with lists, but the extra dimension gives way to some new operations.

Signals of signals have operations that are semi-analogous to traversal order: `-flatten`, `-concat`, `-switchToLatest`

Flatten is special, there's a `-flattenMap:` which is just `-map:` and then `-flatten`.

## Signals Operations

So signals are like callbacks, but they're also like data structures, with flow control.

As concepts go, I think you'll agree they're at one end of the spectrum.

## Signals Operations

Signal operations are the operators and keywords of ReactiveCocoa.

We already accept mathematical operators and language keywords mixed in with our semantically named classes and methods.

They're fiiiiiiiiiiine.

## Signal Operations

Composing and operating on signals (or, "callbacks") may seem foreign.

We now have at our disposal concrete abstractions that were previously expressed implicitly.

This is hard to explain, but I bet you've written ad-hoc imperative versions of `-distinctUntilChanged`, `-throttle:`, and `-deliverOn:`. These are just the start.

## Ultimately Expressive

## Expressiveness

What comes to mind when you think of expressive code?

## Expressiveness

What is expressiveness in the large?

## Expressiveness

Having to keep the code in your head is a code smell.

## Expressiveness

Does object oriented design reduce the amount one needs to keep their head, or increase it?

## OO Hell

## Expressiveness

Callback hell is recognizable. Do we recognize OO Hell when we see it?

## Expressiveness

OO Hell

## Expressiveness

Now scale that.

## Expressiveness

Object oriented design requires perfect communication from all involved.

It only gets harder as the amount of code scales.

## Expressiveness

Why do we have endless troubles with massive methods and classes?

Is OO a mismatch for communication structures, or do we collectively do OO wrong over and over?

OO composition is natural, we humans delegate and specialize. But can software communicate better?

## Expressiveness

We need new ways to compose communication structures.

Signals are a new way to communicate, let's try those.

## Expressiveness

Typical example, a sign-up form:

```
RAC(self.submitButton, enabled) = [RACSignal
    combineLatest:@[
        self.firstNameField.rac_textSignal,
        self.lastNameField.rac_textSignal,
        self.emailField.rac_textSignal,
        self.reEmailField.rac_textSignal
    ]
    reduce:^(NSString *first, NSString *last, NSString *email, NSString *reEmail) {
        return @(first.length > 0 && last.length > 0 && email.length > 0 && reEmail.length > 0 && [email isEqual:reEmail]);
    }];
```

## Expressiveness

## Less typical, performing a search based on location:

```
RAC(self, nearbyFriends) = [[[[[self
    rac_signalForSelector:@selector(locationManager:didUpdateLocations:)
    fromProtocol:@protocol(CLLocationManagerDelegate)]
    throttle:1]
    map:^(NSArray *locations) {
        CLLocation *location = [locations lastObject];
        CLLocationDegrees lat = round(location.latitude * 10) / 10.0
        CLLocationDegrees lon = round(location.longitude * 10) / 10.0;
        return [[CLLocation alloc] initWithLatitude:lat longitude:lon];
    }]
    distinctUntilChanged]
    flattenMap:^(CLLocation *location) {
        return [[APIClient
            searchFriendsNearbyLocation:location];
            doCompleted:^{
                [self.tableView reloadData];
            }]
    }];
```

## Expressiveness

ReactiveCocoa may be ugly to a foreign eye, but it enables a new kind of expressiveness which is sorely needed.

## Closing

## Tip of the Iceberg

Get to know ReactiveCocoa, so much more than I've covered.

## Pitfalls

Plenty: debugging/stack traces, accidentally not subscribing, replay, docs vs headers, lots of learning.

## Embarking on the journey

Straddling RAC and non-RAC is not only ok, it's impossible to avoid, we all live in the same world

By any standards, ReactiveCocoa has an amazing community of support. File an issue for anything trouble you run into. It's the best place to learn, it's like free RAC lessons :)

# Thanks, dave@kastiglione.com