

Swift?!

Alternatives to Optionals

Marius Rackwitz
mr@realm.io

COCOAPODS

The Dependency Manager for Swift & Objective-C.

Swift?!

ImplicitlyUnwrappedOptional <Optional <Swift> >

```
expect(talk).to  
  .cover(topics)
```

Overview of Different Possibilities of Return Types

Including

Optionals

But Also

Attractive Alternatives

But first of all:

**What are those
Optionals at all?**

```
enum Optional<T> : NilLiteralConvertible {
  case None
  case Some(T)

  /// Construct a `nil` instance.
  init()

  /// Construct a non-`nil` instance that stores `some`.
  init(_ some: T)

  /// Create an instance initialized with `nil`.
  init(nilLiteral: ())

  /// If `self == nil`, returns `nil`. Otherwise, returns `f(self!)`.
  func map<U>(f: @noescape (T) -> U) -> U?
}
```

– Swift Standard Library

```
/// An optional type that allows implicit member access (via compiler
/// magic).
///
/// The compiler has special knowledge of the existence of
/// ImplicitlyUnwrappedOptional<T>, but always interacts with it using the
/// library intrinsics below.
enum ImplicitlyUnwrappedOptional<T> : Reflectable, NilLiteralConvertible {
    case None
    case Some(T)

    // ...
}
```

– Swift Standard Library

**Escape Building
Deep-Nested Branch
Hierarchies to
Unwrap Optionals**

```
// Represents a StarWars film
final class Film {
  // The url of this resource
  let url: String

  // The episode number of this film.
  let episodeId: Int

  // The title of this film.
  let title: String

  // The opening crawl text at the beginning of this film.
  let openingCrawl: String

  // The director of this film.
  let director: String //="George Lucas"

  // The producer(s) of this film.
  let producer: String
}
```

```

extension Film {
  static func decode(data: AnyObject) -> Film? {
    if let url = data["url"] as? String {
      if let episodeId = data["episode_id"] as? Int {
        if let title = data["title"] as? String {
          if let openingCrawl = data["opening_crawl"] as? String {
            if let director = data["director"] as? String {
              if let producer = data["producer"] as? String {
                return Film(
                  url: url,
                  episodeId: episodeId,
                  title: title,
                  openingCrawl: openingCrawl,
                  director: director,
                  producer: producer
                )
              }
            }
          }
        }
      }
    }
  }
}

```

```
extension Film {
    static func decode(data: AnyObject) -> Film? {
        if let url = data["url"] as? String,
            let episodeId = data["episode_id"] as? Int,
            let title = data["title"] as? String,
            let openingCrawl = data["opening_crawl"] as? String,
            let director = data["director"] as? String,
            let producer = data["producer"] as? String {
                return Film(url: url, episodeId: episodeId,
                    title: title, openingCrawl: openingCrawl,
                    director: director, producer: producer)
            }
        }
    }
}
```

```
import Argo

extension Film {
  static func create(url: String)(episodeId: Int)(title: String)(openingCrawl: String)
    (director: String)(producer: String) -> Film {
    return Film(url: url, episodeId: episodeId, title: title, openingCrawl: openingCrawl,
      director: director, producer: producer)
  }

  static func decode(j: JSONValue) -> Film? {
    return Film.create
      <^> j <| "url"
      <*> j <| "episode_id"
      <*> j <| "title"
      <*> j <| "opening_crawl"
      <*> j <| "director"
      <*> j <| "producer"
  }
}
```


Optional Return Value + Error Pointer

```
func request(urlString: String, error: NSErrorPointer) -> String? { ... }

var error: NSError?
let result: NSString? = request("http://api.giphy.com/v1/gifs/search?q=doge", &error)

if let e = error {
    println("\(e)")
} else {
    if let urlString = result {
        image.url = NSURL(urlString)
    }
}
```

Pairs of Optionals

```
func request(urlString: String) -> (String?, NSError?) { ... }

let result: (String?, NSError?) = request("http://api.giphy.com/v1/gifs/search?q=doge")

if let error = result.1 {
    println("No luck today: \(error)")
} else {
    if let urlString = result.0 {
        image.url = NSURL(urlString)
    }
}
```

Named Optional Tuple Components

```
func request(urlString: String) -> (String?, NSError?) { ... }

let result: (urlString: String?, error: NSError?) = request("http://api.giphy.com/v1/gifs/search?q=doge")

if error = result.error {
    println("No luck today: \(error)")
} else {
    if urlString = result.urlString {
        image.url = NSURL(urlString)
    }
}
```

Why is this still bad?

Either Success or Failure

```
enum Result<T> {  
    case Success(T)  
    case Failure(NSError)  
}
```



error:

**unimplemented IR
generation feature
non-fixed multi-
payload enum layout**

Either Success or Failure

```
final class Box<T> {  
    let value: T  
  
    public init(_ value: T) { self.value = value }  
}  
  
enum Result<T> {  
    case Success(Box<T>)  
    case Failure(NSError)  
}
```

Either Success or Failure

```
func request(urlString: String) -> Result<String> { ... }

let result: Result<String> = request("http://api.giphy.com/v1/gifs/search?q=doge")

switch result {
    case Success(let box):
        image.url = NSURL(box.value)
    case Error(error: NSError):
        println("\(error)")
}
```


Using LlamaKit

```
import LlamaKit

func request(urlString: String, error: NSErrorPointer) -> String { ... }

let result = try { (error) in
    request("http://api.giphy.com/v1/gifs/search?q=doge", &error)
}

switch result {
    case Success(Box(urlString: String)):
        image.url = NSURL(urlString)
    case Error(error: NSError):
        println("\(error)")
}
```

Promises of Bright Futures

```
import BrightFutures // or PromiseKit, or ...

func request(urlString: String) -> Future<String> {
    let promise = Promise<String>()
    Queue.global.async {
        var error: NSError?
        let result = request(urlString, error: &error)
        if let e = error {
            promise.failure(e)
        } else {
            promise.success(result!)
        }
    }
    return promise.future
}

request("http://api.giphy.com/v1/gifs/search?q=doge").onSuccess { urlString in
    image.url = NSURL(urlString)
}.onFailure { error in
    println("\(error)")
}
```

Reactive Sneak Peek

⚠ Swift API not settled yet.

```
import ReactiveCocoa

let result: SignalProducer<String, NSError> = request("http://api.giphy.com/v1/gifs/search?q=doge")

result.start(next: { urlString in
    image.url = NSURL(urlString)
}, error: { error in
    println("\(error)")
}, completed: {
})
```

Summary:

Swift allows to ...

- Design more **meaningful** interfaces
- **Restrict** semantics
- Increase **Provability** of **Correctness**
- Reduce number of tests 🎉



Justin Spahr-Summers

@jspahrsummers

When signals are parameterized by their error type, an empty (phantom) type can guarantee that signals never error:

[github.com/ReactiveCocoa/ ...](https://github.com/ReactiveCocoa/)

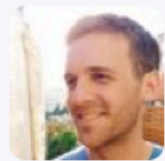


RETWEETS
7

FAVORITEN
13



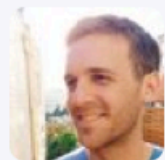
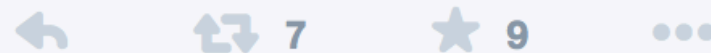
00:58 - 18. Jan. 2015



Justin Spahr-Summers @jspahrsummers · 18. Jan.

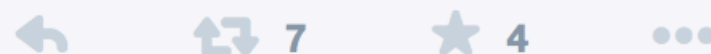
Because this “error” is impossible to use: [github.com/ReactiveCocoa/...](https://github.com/ReactiveCocoa/)

... we can prove, e.g., that bindings never error: [github.com/ReactiveCocoa/...](https://github.com/ReactiveCocoa/)



Justin Spahr-Summers @jspahrsummers · 18. Jan.

Show me a unit test that can do that!



```
for question in questions {  
    question.ask()  
}
```

**Thanks for your
attention!**

@mrackwitz
mr@realm.io