Swift Codables

Blake Merryman

What problem are we solving?

Convert between Swift data structures & archive (aka interchange) formats

Swift is strongly typed

Archival formats are loosely typed

Example: JSON

What problem are we solving?

JSON ↔️ Swift

Vocabulary 

Decode: Convert data to Swift

Encode: Convert Swift to data


SWAPI

Goal A 🥅

```
{
  "name": "Luke Skywalker",
  “eye_color": "blue",
  "url": "https://swapi.co/api/people/1/"
}
```

→

```
/// A Star Wars character.
class SWCharacter: NSObject {
    let name: String
    let eyeColor: String
    let url: String
}
```

Goal B ⚽

```
/// A Star Wars character.
class SWCharacter: NSObject {
    let name: String
    let eyeColor: String
    let url: String
}
```

Disk

👵🏻 The Old Way 👴🏻

```swift
/// A Star Wars character.
class SWCharacter: NSObject {

    /// The full name for a character.
    let name: String
    /// The name of a character's eye color.
    let eyeColor: String
    /// The unique SWAPI URL string for a character.
    let url: String

    /// The designated initializer for a character.
    init(name: String, eyeColor: String, url: String) {
        self.name = name
        self.eyeColor = eyeColor
        self.url = url
        super.init()
    }

}
```

```
func loadNext() {
    get("/people/\(nextAPICharacterIndex)") { data, response, error in
      guard let data = data,
        let newCharacter = SWDataController.createCharacter(from: data) else  {
          return
      }

      if SWDataController.save(character: newCharacter) {
        self.characters.append(newCharacter)
        self.nextAPICharacterIndex += 1
        DispatchQueue.main.async { self.delegate?.didUpdate(dataSource: self) }
      }
    }
  }
```

```swift
/// Attempts to a create a character object.
class func createCharacter(from data: Data) -> SWCharacter? {

    guard let characterJSON = JSONHelper.shared.decodeJSON(fromData: data) else {
        return nil
    }

    return SWCharacter(fromJSON: json)
}
```

```swift
func decodeJSON(fromData data: Data?) -> [String:AnyObject]? {

    guard let data = data else {
      return nil
    }

    do {
      let decoded = try JSONSerialization.jsonObject(with: data, options: [])
      return decoded as? [String:AnyObject]
    } catch let error {
      debugPrint("ERROR DECODING DATA --- \(error.localizedDescription)")
      return nil
    }
}
```

```
/// Convenience initializer used to create instance from JSON dictionary.
convenience init?(fromJSON json: [String: AnyObject]) {

    guard let name = json["name"] as? String,
      let eyeColor = json["eye_color"] as? String,
      let url = json["url"] as? String else {
        return nil
    }

    self.init(name: name, eyeColor: eyeColor, url: url)
}
```

```
func loadNext() {
    get("/people/\(nextAPICharacterIndex)") { data, response, error in
      guard let data = data,
        let newCharacter = SWDataController.createCharacter(from: data) else  {
          return
      }

      if SWDataController.save(character: newCharacter) {
        self.characters.append(newCharacter)
        self.nextAPICharacterIndex += 1
        DispatchQueue.main.async { self.delegate?.didUpdate(dataSource: self) }
      }
    }
  }
```

```
/// Attempts to save a character object to disk.
class func save(character: SWCharacter) -> Bool {
  let characterPath = documentsDirectory.appending("/\(character.name).character")
  let success = NSKeyedArchiver.archiveRootObject(character, toFile: characterPath)
  return success
}
```

```swift
extension SWCharacter: NSCoding {

  /// Allows a character to be unarchived from disk.
  required convenience init?(coder aDecoder: NSCoder) {
    guard let storedName = aDecoder.decodeObject(forKey: "name") as? String,
      let storedHome = aDecoder.decodeObject(forKey: "eye_color") as? String,
      let storedURL  = aDecoder.decodeObject(forKey: "url") as? String else {
        return nil
    }
    self.init(name: storedName, eyeColor: storedHome, url: storedURL)
  }

  /// Allows a character to be archived to disk.
  func encode(with aCoder: NSCoder) {
    aCoder.encode(self.name, forKey: "name")
    aCoder.encode(self.eyeColor, forKey: "eye_color")
    aCoder.encode(self.url, forKey: "url")
  }

}
```

Summary

Requested Character Data from API

Convert `Data` to JSON (i.e. `[String: AnyObject]`)

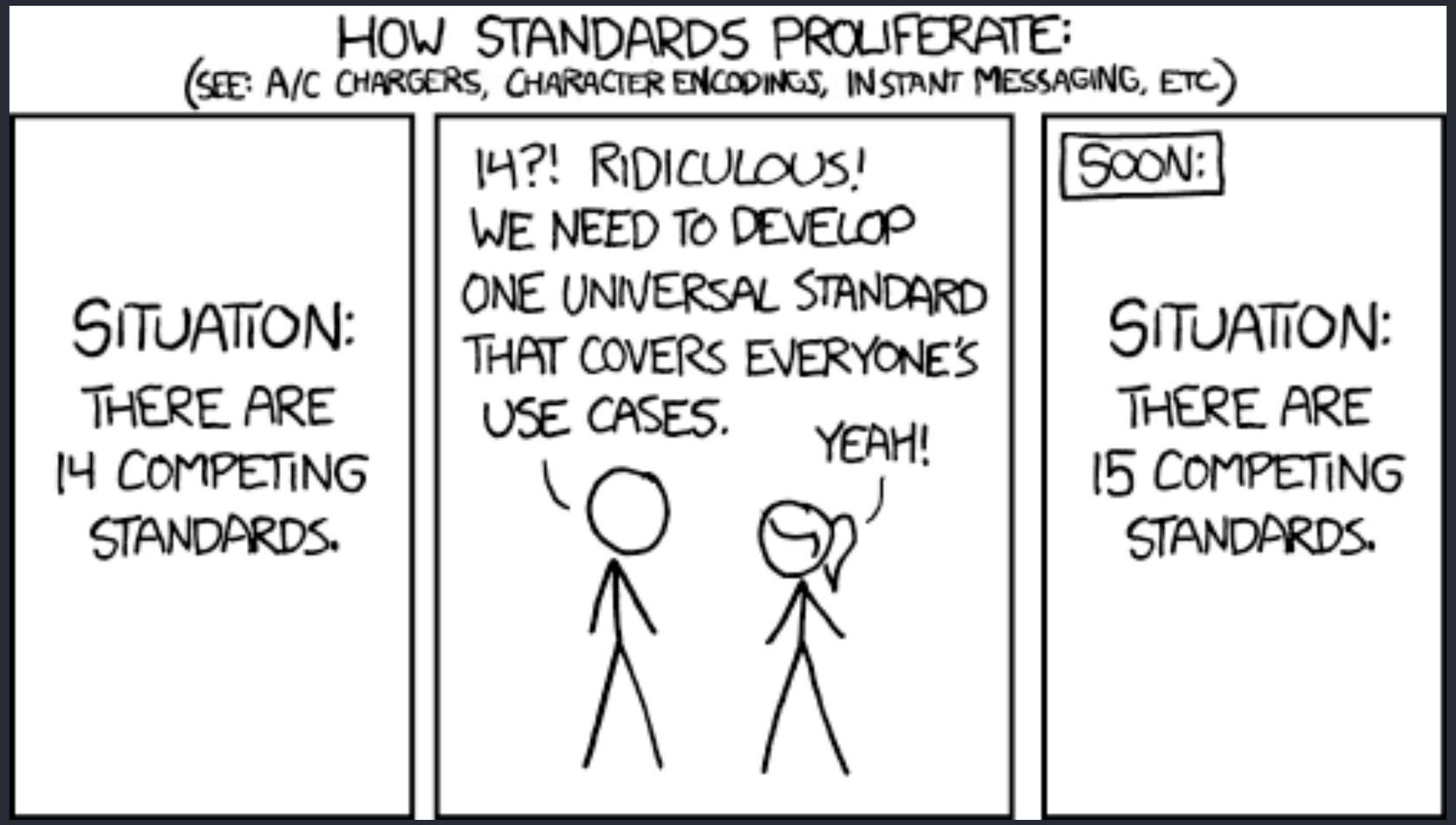
Manually map JSON key/values to `SWCharacter` properties

Persisting characters via keyed-archives (`NSKeyedArchiver`)

A Whole Lot of Bad

- ❌ Strings everywhere!
- ❌ Not type safe (and lots of work to make it so)
- ❌ Tedious & error prone
- ❌ No value semantics (`NSCoding` requires `NSObject`)

3rd Party Parser Proliferation



Sad State of Affairs 🙁

Manually implementing is tedious, error prone, unscalable

Need external dependency for something that should be built in

No platform standard (on-boarding new developers sucks)

A New Hope

```
typealias Codable = Decodable & Encodable

/// A type that can decode itself from an external representation.
protocol Decodable {
  init(from: Decoder) throws
}

/// A type that can encode itself to an external representation.
protocol Encodable {
  encode(to: Encoder) throws
}
```

Maximum Ease & Flexibility

Built into Swift Standard Library

Extremely fast and highly customizable

Supports Classes, Structs, & Enums

`NSObject` independent (but still interoperable)

Support for JSON & Property Lists out of the box

Freebies

Swift Standard Library primitives (e.g. `String`, `Int`, `Float`, etc.)

Some Foundation types (e.g. `Data`, `URL`, `Date`, etc.)

Wrapper types* (e.g. `Array`, `Dictionary`, `Optional`, etc.)

* Any type can be `Codable` if all of its stored properties are `Codable`

Containers 

Keyed	Key Value Key Value Key Value	A collection of key/value pairs
Unkeyed	[Value Value Value]	An ordered list of data
Single Value	Value	A single, primitive value

💃🏻 The `Codable` Way 🕺🏻

```swift
class SWCharacter: NSObject, NSCoding {

  let name: String
  let eyeColor: String
  let url: String

  init(name: String, eyeColor: String, url: String) {
    // ...
  }

  convenience init?(fromJSON json: [String: AnyObject]) {
    // ...
  }

  required convenience init?(coder aDecoder: NSCoder) {
    // ...
  }

  func encode(with aCoder: NSCoder) {
    // ...
  }

}
```

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: String

  enum CodingKeys: String, CodingKey {
    case name, eyeColor, url
  }

  init(from decoder: Decoder) throws {
    let container = try decoder.container(keyedBy: CodingKeys.self)
    self.name = try container.decode(String.self, forKey: .name)
    self.eyeColor = try container.decode(String.self, forKey: .eyeColor)
    self.url = try container.decode(String.self, forKey: .url)
  }

  func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)
    try container.encode(name, forKey: .name)
    try container.encode(eyeColor, forKey: .eyeColor)
    try container.encode(url, forKey: .url)
  }

}
```

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: String

  enum CodingKeys: String, CodingKey {
    case name, eyeColor, url
  }

}
```

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: String

  enum CodingKeys: String, CodingKey {
    case name, eyeColor, url
  }

}
```

```
{
  "name": "Luke Skywalker",
  “eye_color”: "blue",
  "url": "https://swapi.co/api/people/1/"
}
```

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: String

  enum CodingKeys: String, CodingKey {
    case name, eyeColor = "eye_color", url
  }

}
```

```
{
  "name": "Luke Skywalker",
  “eye_color”: "blue",
  "url": "https://swapi.co/api/people/1/"
}
```

Coming Soon...

Swift 4.1 introduces a new JSON decoding strategy:

```
keyDecodingStrategy = .convertFromSnakeCase
```

** Would allow us to omit CodingKeys in our example!*

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: String

  enum CodingKeys: String, CodingKey {
    case name, eyeColor = "eye_color", url
  }

}
```

```
{
  "name": "Luke Skywalker",
  “eye_color”: "blue",
  "url": "https://swapi.co/api/people/1/"
}
```

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: URL

  enum CodingKeys: String, CodingKey {
    case name, eyeColor = "eye_color", url
  }

}
```

```json
{
  "name": "Luke Skywalker",
  “eye_color”: "blue",
  "url": "https://swapi.co/api/people/1/"
}
```

```
func loadNext() {
    get("/people/\(nextAPICharacterIndex)") { data, response, error in
      guard let data = data,
        let newCharacter = SWDataController.createCharacter(from: data) else  {
          return
      }

      if SWDataController.save(character: newCharacter) {
        self.characters.append(newCharacter)
        self.nextAPICharacterIndex += 1
        DispatchQueue.main.async { self.delegate?.didUpdate(dataSource: self) }
      }
    }
  }
```

```swift
/// Attempts to a create a character object.
class func createCharacter(from data: Data) -> SWCharacter? {

    return try? JSONDecoder.shared.decode(SWCharacter.self, from: data)

}
```

```swift
/// Attempts to save a character object to disk.
class func save(character: SWCharacter) -> Bool {

    let path = documentsDirectory.appending("/\(character.name).character")
    let url = URL(fileURLWithPath: path)

    guard let data = try? JSONEncoder.shared.encode(character) else {
      return false // Encoding to data failed.
    }

    do {
      try data.write(to: url)
      return true
    }
    catch {
      return false
    }
}
```

Summary

Requested Character Data from API

Decode `Data` directly to a `SWCharacter`

Persisting characters by writing JSON files to disk

```swift
struct SWCharacter: Codable {

  let name: String
  let eyeColor: String
  let url: URL

  enum CodingKeys: String, CodingKey {
    case name, eyeColor = "eye_color", url
  }

}
```

```swift
struct SWCharacter: Codable {

    let url: String
    let created: Date
    let edited: Date
    let name: String
    let birthYear: String
    let gender: String
    let eyeColor: String
    let hairColor: String
    let skinColor: String
    let height: String
    let mass: String

    // MARK: - Coding Keys
    // We need this only so we can override the JSON key name.
    enum CodingKeys: String, CodingKey {
        case url
        case created
        case edited
        case name
        case birthYear = "birth_year"
        case gender
        case eyeColor = "eye_color"
        case hairColor = "hair_color"
        case skinColor = "skin_color"
```

Easily add new properties! 🎉

A Whole Lot of Good

- ✅ Minimal strings (only for container key overrides)
- ✅ Type safe
- ✅ Easy to work with and extend
- ✅ Supports value semantics

Some Assembly Required...

Codable Subclasses

Collapsing Nested Containers

Heterogenous Arrays

Anything Dynamic

Gracefully Handling Container Issues (missing or versioned data)

JSON Coding Strategies

	Decoding	Encoding
Keys	`keyDecodingStrategy`	`keyEncodingStrategy`
Dates	`dateDecodingStrategy`	`dateEncodingStrategy`
Data	`dataDecodingStrategy`	`dataEncodingStrategy`
Floats	`nonConforming FloatDecodingStrategy`	`nonConforming FloatEncodingStrategy`

Q & A

Sources

- [Encoding and Decoding Custom Types | Apple Developer Documentation](https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types)

- [Ultimate Guide to JSON Parsing with Swift 4 | Ben Scheirman](http://swiftjson.guide)

- [Swift Codable With Custom Dates | Use Your Loaf](https://useyourloaf.com/blog/swift-codable-with-custom-dates/)

- [blakemerryman/NashDevFest_SWAPI | Github](https://github.com/blakemerryman/NashDevFest_SWAPI)

- [SWAPI - The Star Wars API](https://swapi.co)

- [matteocrippa/awesome-swift | Github](https://github.com/matteocrippa/awesome-swift)

- [Swift 4.1 improves Codable with keyDecodingStrategy | Hacking with Swift](https://www.hackingwithswift.com/articles/52/swift-4-1-improves-codable-with-keydecodingstrategy)

- [What's New in Foundation | WWDC 2017](https://developer.apple.com/videos/play/wwdc2017/212/)

Codable Challenge 😏