Decoding Codable

DL

360|iDev

What this talk covers

Yup

- What JSON parsing looked like pre Swift 4
- What it looks like in Swift 4 with Codable
- Ways to deal with less than perfect JSON structures

Nope

- Exhaustive look at Codable
- A deep dive under the hood
- Step by step implementation guide
- Song and dance number

360|iDev

Ellen Mey

iOS Developer

DETROIT LABS

@el_is_for_ellen

Once upon a time...

```
struct CurrentWeather {
    let chanceOfRain: Int
    let temperature: Int
    let details: String

    init?(json: [String: Any]) {
        guard let chanceOfRain = json["chanceOfRain"] as? Int,
            let temperature = json["temperature"] as? Int,
            let details = json["description"] as? String else { return }

        self.chanceOfRain = chanceOfRain
        self.temperature = temperature
        self.details = details
    }
}
```

Once upon a time…

```
do {
    if let weatherJSON = try JSONSerialization.jsonObject(with: responseData,
        options: []) as? [String: Any],
        let weather = CurrentWeather(json: weatherJSON) {
        completionHandler(weather, nil)
    } else {
        // Handle the error if the object can't be created
    }
} catch {
    // Error converting data to JSON using JSONSerialization.jsonObject
    completionHandler(nil, error)
    return
}
```

Like this… 😬

```
struct CurrentWeather {
    let chanceOfRain: Int
    let temperatures: (farenheit: Int, celcius: Int)
    let details: String

    init?(json: [String: Any]) {
        guard let chanceOfRain = json["chanceOfRain"] as? Int,
        let temperatureJson = json["temperatures"] as? [String: Int],
            let farenheit = temperatureJson["farenheit"],
            let celcius = temperatureJson["celcius"],
            let details = json["description"] as? String else { return nil }

        self.chanceOfRain = chanceOfRain
        self.temperatures = (farenheit, celcius)
        self.details = details
    }
}
```

360|iDev

Wait we need to encode, too… 😱

```
struct CurrentWeather {
    let chanceOfRain: Int
    let temperatures: (farenheit: Int, celcius: Int)
    let details: String

    init?(json: [String: Any]) {
        guard let chanceOfRain = json["chanceOfRain"] as? Int,
        let temperatureJson = json["temperatures"] as? [String: Int],
            let farenheit = temperatureJson["farenheit"],
            let celcius = temperatureJson["celcius"],
            let details = json["description"] as? String else { return nil }

        self.chanceOfRain = chanceOfRain
        self.temperatures = (farenheit, celcius)
        self.details = details
    }

    func toJson() -> [String: Any] {
        let temps = ["farenheit: \(temperatures.farenheit)", "celcius:\
            (temperatures.celcius)"]
        var json = [String: Any]()
        json["chanceOfRain"] = chanceOfRain
        json["temperatures"] = temps
        json["details"] = details

        return json
    }
}
```

The whole thing…

```
struct CurrentWeather {

    let chanceOfRain: Int
    let temperatures: (farenheit: Int, celcius: Int)
    let description: String

    init?(json: [String: Any]) {
        guard let chanceOfRain = json["chanceOfRain"] as? Int,
        let temperatureJSON = json["temperatures"] as? [String: Int],
        let farenheit = temperatureJSON["farenheit"],
        let celcius = temperatureJSON["celcius"],
        let descriptionJSON = json["description"] as? String
            else { return nil }

        self.chanceOfRain = chanceOfRain
        self.temperatures = (farenheit, celcius)
        self.description = description
    }

    func toJSON() -> [String: Any] {

        let temps = ["farenheit: \(temperatures.farenheit)", "celcius: \(temperatures.celcius)"]
        var json = [String: Any]()
        json["chanceOfRain"] = chanceOfRain
        json["temperatures"] = temps
        json["description"] = description

        return json
    }
}

func getWeather(completionHandler: @escaping (CurrentWeather?, Error?) -> Void) {
    do {
        if let weatherJSON = try JSONSerialization.jsonObject(with: responseData, options: []) as?
            [String: Any],
            let weather = CurrentWeather(json: weatherJSON) {
            completionHandler(weather, nil)
        } else {
            // Handle the error if the object can't be created
        }
    } catch {
        // error converting data to JSON using JSONSerialization.jsonObject
        completionHandler(nil, error)
        return
    }
}


 let newWeatherReportAsJSON = self.toJSON()

 do {
 let jsonWeatherReport = try JSONSerialization.data(withJSONObject: newWeatherReportAsJSON, options:
     [])
 weatherUrlRequest.httpBody = jsonWeatherReport
 } catch {
 let error = BackendError.objectSerialization(reason: "Could not create JSON from Weather Report")
 completionHandler(error)
 return
 }
```

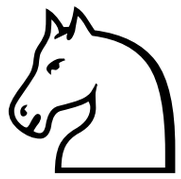
360|iDev

♘ Swift 4 ⚔️

```swift
struct CurrentWeather: Codable {
    let chanceOfRain: Int
    let temperatures: (farenheit: Int, celcius: Int)
    let details: String
```

```swift
func getWeather(completionHandler: @escaping (CurrentWeather?, Error?) -> Void) {
    let decoder = JSONDecoder()
    do {
        let weather = try decoder.decode(CurrentWeather.self, from: responseData)
        completionHandler(weather, nil)
    } catch {
        print(error)
        completionHandler(nil, error)
    }
}
```

360|iDev



Swift 4

Encoding

```
let encoder = JSONEncoder()
do {
    let newWeatherReportAsJSON = try encoder.encode(self)
    todosUrlRequest.httpBody = newWeatherReportAsJSON
} catch {
    print(error)
    completionHandler(error)
}
```

360|iDev

Magic 

- Codable automatically handles your init()
- Coding Keys are automatically created
- Cleaner and easier to read
- Takes *weakly typed* JSON and converts it to *strongly typed* Swift

360|iDev

Custom Types

```
struct CurrentWeather: Codable {
    let chanceOfRain: Int
    let temperatures: (farenheit: Int, celcius: Int)
    let details: WeatherType

    enum WeatherType: Codable {
        case fair, cloudy, sunny, windy
    }
}
```

Custom Coding Keys

```
struct CurrentWeather: Codable {
    let chanceOfRain: Int
    let temperatures: Int
    let details: WeatherType

    enum WeatherType: Codable {
        case fair, cloudy, sunny, windy
    }

    enum CodingKeys: String, CodingKey {
        case chanceOfRain, temperature
        case details = "weatherDescription"
    }
}
```

Manually Encoding and Decoding

Containers 

- Keyed Container - like a dictionary
- Unkeyed Container - ordered values, think array
- Single Value Container - raw value, no containing element
- Must be mutable, we need to write to it
- Need to specify keys

Why would we want to?

```
let decoder = JSONDecoder()
let container = try decoder.container(keyedBy: CodingKeys.self)
chanceOfRain = try container.decode(Int.self, forKey: .chanceOfRain)
temperature = try container.decode(Int.self, forKey: .temperature)
details = try container.decode(WeatherType.self, forKey: .details)
date = try container.decodeISO860Date(forKey: .date)
```

Nested Containers

```
struct CurrentWeather: Codable {
    let chanceOfRain: Int
    let temperature: Int
    let details: WeatherType
    let date: Date
    let city: String

enum WeatherType: Codable {
    case fair, cloudy, sunny, windy
}

enum CodingKeys: String, CodingKey {
    case chanceOfRain, temperature, date, address
    case details = "weatherDescription"
}

enum AddressKeys: CodingKey {
    case city
}
```

Nested Containers

```
extension CurrentWeather {
    public init(from decoder: Decoder) throws {
        let values = decoder.container(keyedBy: CodingKeys.self)
        let chanceOfRain = try values.decodeIfPresent(Int.self, forKey: .chanceOfRain)
        let temperature = try values.decode(Int.self, forKey: .temperature)
        let details = try value.decode(WeatherType.self, forKey: .details)
        let date = try container.decodeISO860Date(forKey: .date)

        let locationValues = try values.nestedContainer(keyedBy: AddressKeys.self,
                                                        forKey: .address)
        let city = try locationValues.decode(String.self, forKey: .city)

        self.init(chanceOfRain: chanceOfRain, temperature: temperature, details: details,
            date: date, city: city)
    }
}
```

Nested Containers

```
let locationValues = try values.nestedContainer(keyedBy: AddressKeys.self,
                                                forKey: .address)
let city = try locationValues.decode(String.self, forKey: .city)
```

What if the root element is an array? 🤔

```
let weather = try decoder.decode([CurrentWeather].self, from: data)
```

360|iDev

Decodable

+

Encodable

=

Codable 4Ever 💘

```
struct CurrentWeather: Decodable {
    // All the properties
}

struct WeatherReport: Encodable {}
```

keyDecodingStrategy

- .convertFromSnakeCase

```
let decoder = JSONDecoder()
decoder.keyDecodingStrategy = .convertFromSnakeCase
```

- .custom(@escaping ([CodingKey]) -> CodingKey)

```
let decoder = JSONDecoder()

decoder.keyDecodingStrategy = .custom { keys in
let lastComponent = keys.last!.stringValue.split(separator: ".").last!
return AnyKey(stringValue: String(lastComponent))!
}
```

Other things of note

- PropertyListEncoder
- Create a custom encoder
- dateDecodingStrategy is a bit limited but helpful
- Handling Floats
- Utilizing Codable with other frameworks like RxSwift

Thank You

Additional Resources and Links

- Slides will be posted at: github.com/eisforellen/talks
- Tweet me questions: @el_is_for_ellen
- https://grokswift.com/json-swift-4/
- https://www.raywenderlich.com/172145/encoding-decoding-and-serialization-in-swift-4
- https://useyourloaf.com/blog/swift-codable-with-custom-dates/
- https://www.hackingwithswift.com/articles/119/codable-cheat-sheet