

Enumerating all of the reasons why *enums* are AWESOME!

By: Conrad Stoll

—



Enumerating All The Things⁰

Liss is More

By Casey Liss

[ABOUT](#) • [CONTACT](#) • [ELSEWHERE](#) • [RSS](#)

Enumerating The Ways I Love Swift Enumerations

Saturday, 27 February 2016

⁰ <https://www.caseyliss.com/2016/2/27/swift-enums>



State Driven Development

enum

Bool

```
var instanceVariable : Bool
```

How often exactly?

```
// Regex for boolean state  
// Match Bool typed vars that aren't computed properties  
var.*:.*Bool(\n|.*=.*)  
var.*=\s?(true|false)
```



Xcode

File

Edit

View

Find

Navigate



Grocery >



iPhone 8 Plus



Find > Text > Containing

Q~|Text



■ ■ ■ In Grocery.xcodeproj

Ignoring Case ⌵

```
var active : Bool
var contentLoaded : Bool
var dataLoaded : Bool
var editing : Bool
var enabled : Bool
var expanded : Bool
var handlingTouches : Bool
var hasTakenUserAction : Bool
var isCompleted : Bool
var isRunning : Bool
var lastTouchedButtonSide : Bool
var lastTouchedRightSide : Bool
var notified : Bool
var previewEnded : Bool
var saved : Bool
var started : Bool
```

Example

Starting a Workout

```
func startWorkout() {  
    // ...  
}
```

One Variable

```
var hasLoggedIn : Bool = false
```

```
func startWorkout() {  
    guard hasLoggedIn else {  
        return  
    }  
}
```

```
// ...
```

```
}
```

Two Variables

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false

func startWorkout() {
    guard hasLoggedIn && workoutInProgress == false else {
        return
    }

    workoutInProgress = true

    // ...
}
```

The Tangled Web of State

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false
var savingWorkout : Bool = false

func stopWorkout() {
    workoutInProgress = false
    savingWorkout = true

    saveWorkout { (success) in
        self.savingWorkout = false
    }
}
```

The Tangled Web of State

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false
var savingWorkout : Bool = false

func startWorkout() {
    guard hasLoggedIn && workoutInProgress == false && !savingWorkout else {
        return
    }

    workoutInProgress = true

    // ...
}
```

The user **CANNOT** start a workout:

- Hasn't logged into the app yet
- In the middle of a workout
- Saving the last workout

The user **CAN** start a workout:

- Ready to start workout

Valid Boolean States

User Not Logged In, No Workout, Not Saving

User Logged In, No Workout, Not Saving

User Logged In, Workout In Progress, Workout Not Saving

User Logged In, No Workout, Workout Is Saving

```
var hasLoggedIn : Bool
```

```
var workoutInProgress : Bool
```

```
var savingWorkout : Bool
```

Invalid Boolean States

User Not Logged In, Workout In Progress, Not Saving

User Not Logged In, Workout In Progress, Workout Is Saving

User Not Logged In, No Workout, Workout Is Saving

User Logged In, Workout In Progress, Workout Is Saving

```
var hasLoggedIn : Bool
```

```
var workoutInProgress : Bool
```

```
var savingWorkout : Bool
```

Perfect case for an enum

- Hasn't logged into the app yet
- In the middle of a workout
- Saving the last workout
- Ready to start workout

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case readyToStart  
}
```

Enums make invalid states
impossible to represent!

Converting to Enums

```
var hasLoggedIn : Bool = false
var workoutInProgress : Bool = false
var savingWorkout : Bool = false

func startWorkout() {
    guard hasLoggedIn && workoutInProgress == false && !savingWorkout else {
        return
    }

    workoutInProgress = true

    // ...
}
```

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func startWorkout() {

    // Equivalent
    guard .readyToStart == lifecycle else {
        return
    }

    // Pattern Matching
    guard case .readyToStart = lifecycle else {
        return
    }

    lifecycle = .workoutInProgress

    // ...
}
```

Extending enums with variables and functions

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case readyToStart  
  
    var canStartWorkout : Bool {  
        get {  
            switch self {  
                case .notLoggedIn:  
                    return false  
                case .workoutInProgress:  
                    return false  
                case .savingLastWorkout:  
                    return false  
                case .readyToStart:  
                    return true  
            }  
        }  
    }  
}
```

Simplified for multiple cases

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case readyToStart  
  
    var canStartWorkout : Bool {  
        get {  
            switch self {  
                case .notLoggedIn, .workoutInProgress, .savingLastWorkout:  
                    return false  
                case .readyToStart:  
                    return true  
            }  
        }  
    }  
}
```

Simplified using *fallthrough*

```
enum WorkoutApplicationLifecycle {  
  case notLoggedIn  
  case workoutInProgress  
  case savingLastWorkout  
  case readyToStart  
  
  var canStartWorkout : Bool {  
    get {  
      switch self {  
        case .notLoggedIn: fallthrough  
        case .workoutInProgress: fallthrough  
        case .savingLastWorkout:  
          return false  
        case .readyToStart:  
          return true  
      }  
    }  
  }  
}
```

default Danger!

```
enum WorkoutApplicationLifecycle {  
  case notLoggedIn  
  case workoutInProgress  
  case savingLastWorkout  
  case readyToStart  
  
  var canStartWorkout : Bool {  
    get {  
      switch self {  
        case .readyToStart:  
          return true  
        default:  
          return false  
      }  
    }  
  }  
}
```

Enums describing the behavior

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func startWorkout() {
    guard lifecycle.canStartWorkout else {
        return
    }

    lifecycle = .workoutInProgress

    // ...
}
```

Enums describing the behavior

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func stopWorkout() {
    lifecycle = .savingLastWorkout

    saveWorkout { (success) in
        self.lifecycle = .readyToStart
    }
}
```

State Driven Development

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case readyToStart  
  
    var canStartWorkout : Bool  
    var actionTitle : String  
    var backgroundColor : UIColor  
    var barButtonItem : [UIBarButtonItem]  
}
```

Title for Buttons

```
enum WorkoutApplicationLifecycle {  
  
    // ...  
  
    var actionTitle : Bool {  
        get {  
            switch self {  
            case .notLoggedIn:  
                return "Sign In"  
            case .workoutInProgress:  
                return "End Workout"  
            case .savingLastWorkout:  
                return "Saving Workout"  
            case .readyToStart:  
                return "Start Workout"  
            }  
        }  
    }  
}
```

Simple Component Interfaces

```
@IBAction func didTapActionButton() {  
    lifecycle.actionTaken(from: self)  
}
```

Actions for Buttons

```
enum WorkoutApplicationLifecycle {  
  
    // ...  
  
    func actionTaken(from viewController : WorkoutViewController) {  
        switch self {  
        case .notLoggedIn:  
            viewController.login()  
        case .workoutInProgress:  
            viewController.stopWorkout()  
            viewController.saveWorkout()  
        case .savingLastWorkout:  
            break  
        case .readyToStart:  
            viewController.startWorkout()  
        }  
    }  
}
```

Adding Features

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case restoringWorkout  
  
    var canStartWorkout : Bool {  
        case .restoringWorkout:  
            return false  
    }  
  
    var actionTitle : Bool {  
        case .restoringWorkout:  
            return "Restoring Workout"  
    }  
  
    func actionTaken(from viewController : WorkoutViewController) {  
        case .restoringWorkout:  
            break  
    }  
}
```

Associated Values

Defining Associated Values

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case workoutInProgress(current : Workout)  
}
```

Accessing Associated Values

```
switch self {  
    case .workoutInProgress(let workout):  
        break  
}  
  
if case let .workoutInProgress(workout) = lifecycle {  
    // ...  
}
```

```
if case .workoutInProgress = lifecycle { }
```

```
if case let .workoutInProgress(_) = lifecycle {  
    // ...  
}
```

```
if case let .workoutInProgress(current: workout) = lifecycle {  
    // ...  
}
```

```
if case .workoutInProgress(let workout) = lifecycle {  
    // ...  
}
```

```
if case let .workoutInProgress(workout, user) = lifecycle {  
    // ...  
}
```

```
switch self {  
  case .workoutInProgress:  
    // ...  
  case .workoutInProgress(_):  
    // ...  
  case let .workoutInProgress(current: workout):  
    // ...  
  case let .workoutInProgress(workout, user):  
    // ...  
  case .workoutInProgress(let workout):  
    // ...  
}
```

Recommended Syntax

```
if case let .workoutInProgress(current: workout) = lifecycle {  
    // ...  
}
```

```
switch self {  
    case let .workoutInProgress(current: workout):  
        // ...  
}
```

Using Associated Values

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case workoutInProgress(current: Workout)  
  
    func actionTaken(from viewController : WorkoutViewController) {  
        switch self {  
        case let .workoutInProgress(current: workout):  
            viewController.stopWorkout()  
            viewController.saveWorkout(current: workout)  
        }  
    }  
}
```

Using Associated Values

```
enum WorkoutApplicationLifecycle {  
    // ...  
    case workoutInProgress(current: Workout)  
  
    var navigationTitle : String {  
        switch self {  
        case let .workoutInProgress(current: workout):  
            return workout.type.activityName  
        }  
    }  
}
```

Setting Associated Values

```
var lifecycle : WorkoutApplicationLifecycle = .notLoggedIn

func startWorkout() {
    guard lifecycle.canStartWorkout else {
        return
    }

    var newWorkout : Workout

    // ...

    lifecycle = .workoutInProgress(current: workout)
}
```

Table View Cell Configurations

```
enum TableViewCellConfiguration {  
    case simpleCell(title : String)  
    case detailedCell(title : String, description : String)  
    case imageCell(title : String, image : UIImage)  
    case accessoryCell(title : String,  
                        image : UIImage,  
                        accessory : UITableViewCellStyleAccessoryType)  
  
    var titleText : String  
    var descriptionText : String?  
    var image : UIImage?  
    var accessoryType : UITableViewCellStyleAccessoryType  
}
```

```
enum TableViewCellConfiguration {  
    case workoutCell(workout : Workout)  
    case userProfileCell(loggedIn : User)  
    case exerciseCell(exercise : Exercise)  
    case trainingPlanCell(plan : TrainingPlan)  
  
    var titleText : String  
    var descriptionText : String?  
    var image : UIImage?  
    var accessoryType : UITableViewCellStyleAccessoryType  
}
```

```
class CustomTableViewCell : UITableViewCell {  
  
    @IBOutlet weak var titleLabel : UILabel!  
    @IBOutlet weak var descriptionLabel : UILabel?  
    @IBOutlet weak var imageView : UIImageView?  
  
    func setup(as configuration : TableViewCellConfiguration) {  
        titleLabel.text = configuration.titleText  
        descriptionLabel?.text = configuration.descriptionText  
        imageView?.image = configuration.image  
        accessoryType = configuration.accessoryType  
    }  
}
```

Cell Configuration

```
func tableView(_ tableView: UITableView,
  cellForItemAt indexPath: IndexPath) -> UITableViewCell {
  let cell = tableView.dequeueReusableCell
    (withReuseIdentifier: reuseIdentifier,
     for: indexPath) as! CustomTableViewCell

  let exercise = exercises[indexPath.row]

  cell.setup(as: .exerciseCell(exercise: exercise))

  return cell
}
```

View Controller Modes

Viewing, Editing, New

View Controller Mode Methods

- viewSetupForTransitionToState
- cancelButtonAction
- editButtonAction
- saveButtonAction
- barButtonSetupForTransitionToState
- leftBarButtonItems
- rightBarButtonItems

View Controller Mode Methods

- didSelectRowAtIndexPath
- contentSelectionAction
- shouldHighlightCellAction
- shouldHandleLongPressGestureAction
- dragInteractionEnabled
- numberOfSections
- numberOfRowsInSection

Workout Activity Types

Running, Cycling, Swimming

Workout Activity Type Methods

- `activityName`
- `imageForActivity`
- `visibleStatsForActivity`
- `supportsGPS`
- `waterLockEnabled`

```
if activityType == .running ||  
    activityType == .walking {  
    gpsProvider.start()  
}
```

```
if activityType == .running ||  
    activityType == .walking {  
    gpsProvider.stop()  
}
```

```
if activityType == .running ||  
    activityType == .walking ||  
    activityType == .hiking {  
    gpsProvider.start()  
}
```

```
if activityType == .running ||  
    activityType == .walking {  
    gpsProvider.stop()  
}
```

```
if activityType.supportsGPS {  
    gpsProvider.start()  
}
```

```
if activityType.supportsGPS {  
    gpsProvider.stop()  
}
```

Result¹

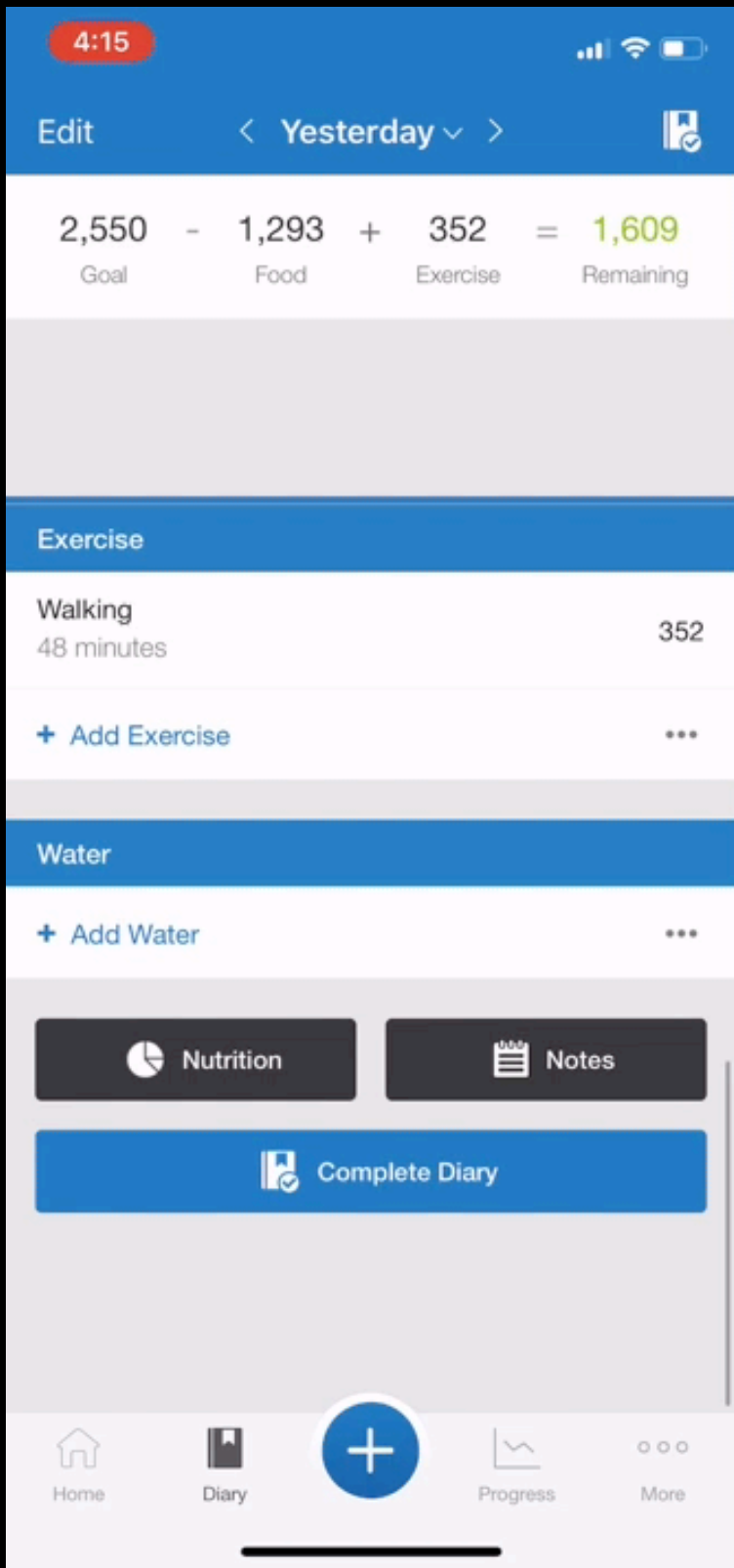
Success, Failure

```
enum Result<Value> {  
    case success(Value)  
    case failure(Error)  
}
```

¹ <https://www.swiftbysundell.com/posts/the-power-of-result-types-in-swift>

Confetti Moments

Colors, Images, 



Confetti Moments

```
public enum ConfettiMoment {  
    // Color Confetti  
    case confettiMMF  
    case confettiMFP  
    case confetti🌈  
  
    // Emoji Confetti  
    case 🌮  
  
    // Custom Confetti  
    case customColors(colors : [UIColor], name : String)  
    case customImages(images : [UIImage], name : String)  
}
```

Confetti Moment Public API

```
extension ConfettiMoment {  
    func effectScene() -> SKScene  
}
```

Protocols

```
extension ConfettiMoment : CustomStringConvertible {  
    public var description: String {  
        get {  
            switch self {  
                case .🌮: return "🌮"  
                case .confettiMMF: return "MMF"  
                case .confettiMFP: return "MFP"  
                case .confetti🏳️: return "🏳️"  
                case .customColors(_, let name): return name  
                case .customImages(_, let name): return name  
            }  
        }  
    }  
}
```

Raw Values

```
enum DefaultKeys : String {  
    case loggedInUserId  
    case selectedActivityType  
}
```

```
print(DefaultKeys.selectedActivityType.rawValue)
```

```
selectedActivityType
```

Raw Values

```
enum DefaultKeys : String {  
    case loggedInUserId  
    case selectedActivityType = "SelectedActivityTypeDefaultKey"  
}  
  
print(DefaultKeys.selectedActivityType.rawValue)  
  
SelectedActivityTypeDefaultKey
```

RawRepresentable

```
enum ApplicationTheme {  
    case defaultLight  
    case darkMode  
    case blueberry  
    case eggplant  
    case mustard  
    case broccoli  
}  
  
struct Theme : Equatable {  
    let name : String  
    let background : UIColor  
    let cell : UIColor  
    let tint : UIColor  
    let foreground : UIColor  
    let statusBar : UIStatusBarStyle  
}
```

RawRepresentable

```
enum ApplicationTheme {  
    case defaultLight(theme : Theme)  
    case darkMode(theme : Theme)  
    case blueberry(theme : Theme)  
    case eggplant(theme : Theme)  
    case mustard(theme : Theme)  
    case broccoli(theme : Theme)  
}
```

RawRepresentable

```
extension ApplicationTheme : RawRepresentable {  
    typealias RawValue = Theme  
  
    init?(rawValue: Theme) {  
        // ...  
    }  
  
    var rawValue: Theme {  
        // ...  
    }  
}
```

```
print(ApplicationTheme.broccoli.rawValue.name)
```

```
Broccoli
```

Associated vs. Raw Values

New in Swift 4.2



Enumerate Enum Cases²

² <https://oleb.net/blog/2018/06/enumerating-enum-cases/>

CaseIterable

```
enum WorkoutApplicationLifecycle : CaseIterable {  
    case notLoggedIn  
    case workoutInProgress  
    case savingLastWorkout  
    case readyToStart  
}
```

```
WorkoutApplicationLifecycle.allCases
```

```
[notLoggedIn, workoutInProgress, savingLastWorkout, readyToStart]
```

Custom CaseIterable

```
enum WorkoutApplicationLifecycle {  
    case notLoggedIn  
    case workoutInProgress(current : Workout)  
    case savingLastWorkout  
    case readyToStart  
}  
  
extension WorkoutApplicationLifecycle : CaseIterable {  
    static var allCases: [WorkoutApplicationLifecycle] {  
        return [.notLoggedIn, .savingLastWorkout, .readyToStart]  
            + WorkoutActivityType.allCases.map(.workoutInProgress)  
    }  
}
```

All the Confetti Moments

```
extension ConfettiMoment : CaseIterable {  
    static var allCases: [ConfettiMoment] {  
        return [.🌮, .confettiMMF, .confettiMFP, .confetti🏳️‍🌈]  
    }  
}
```

All the Themes

```
enum ApplicationTheme : CaseIterable {  
    case defaultLight  
    case darkMode  
    case blueberry  
    case eggplant  
    case mustard  
    case broccoli  
}
```

```
ApplicationTheme.allCases
```

```
[.defaultLight, .darkMode, .blueberry, .eggplant, .mustard, .broccoli]
```

Additional Resources

- Enum Driven Table Views³⁴
- Pattern Matching in Swift⁵

³ <https://www.natashatherobot.com/swift-enums-tableviews/>

⁴ <https://www.raywenderlich.com/5542-enum-driven-tableview-development>

⁵ <http://alisoftware.github.io/swift/pattern-matching/2016/03/27/pattern-matching-1/>

Additional Resources

- Enums and Optionals⁶
- Writing Self Documenting Swift Code⁷
- Code Encapsulation in Swift⁸
- Enumerations⁹

⁶ <http://khanlou.com/2018/04/enums-and-optionals/>

⁷ <https://www.swiftbysundell.com/posts/writing-self-documenting-swift-code>

⁸ <https://www.swiftbysundell.com/posts/code-encapsulation-in-swift>

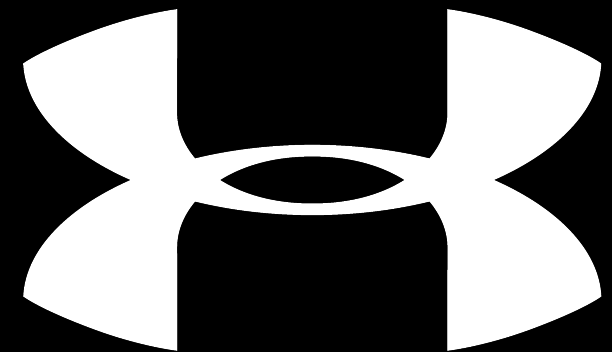
⁹ <https://ericasadun.com/2015/05/26/swift-the-hall-of-the-dwarven-enumeration-king/#more-1529>

Thanks!

—

@conradstoll

—



UNDER ARMOUR