Kicking Things Off With Swift

STL CocoaHeads

Brian Coyner
Principal Software Engineer at NISC
https://briancoyner.github.io

STL CocoaHeads

- Originally started in 2011
  - Remember “Embracing The Brackets”?
- It’s been dormant for a few years
- Getting things started again this month

STL CocoaHeads Schedule

- [ ] Meet the 2nd Thursday of each month
- [ ] Meeting at NISC
- [ ] Start at 6pm (doors “open” at 5:30)

STL CocoaHeads Goals

- [ ] Build a community
- [ ] Lots of exciting topics to cover
  - [ ] Swift, ObjC
  - [ ] iOS, tvOS, watchOS, macOS
  - [ ] Android and Kotlin (if there's interest)
  - [ ] Server technologies
- [ ] Let me know if you want to speak

About me…

About you…

Kicking Things Off With Swift

A brief survey of Swift's language features

Session Topics

- [ ] Talking To Lionel “Toy Train” Engines
- [ ] Being Lazy with `lazy var` Stored Properties
- [ ] View State Transitions
- [ ] Quick look at Server Side Swift

Session Assumptions and Expectations

- ☐ Assuming there is a mix of Swift language knowledge
- ☐ Some Swift language details are omitted from the slides
- ☐ Live Xcode Playground demos should help fill in the details
- ☐ Feel free to ask questions if anything is not clear

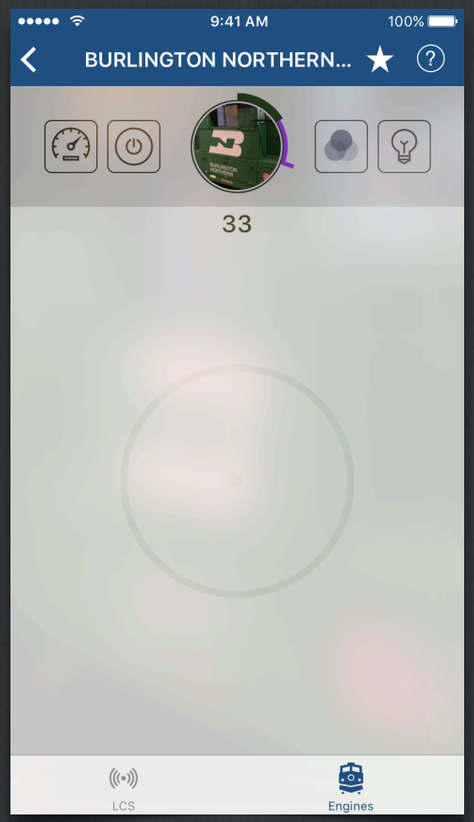
Talking To Lionel “Toy Train” Engines

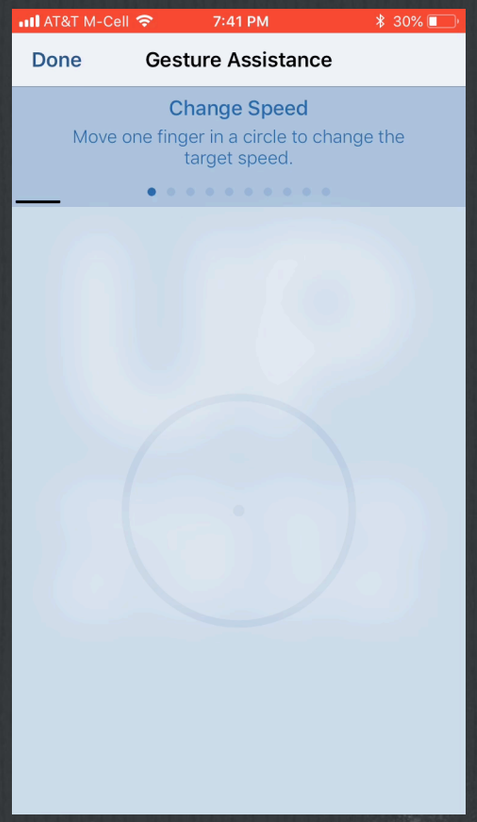
It's good to have side projects

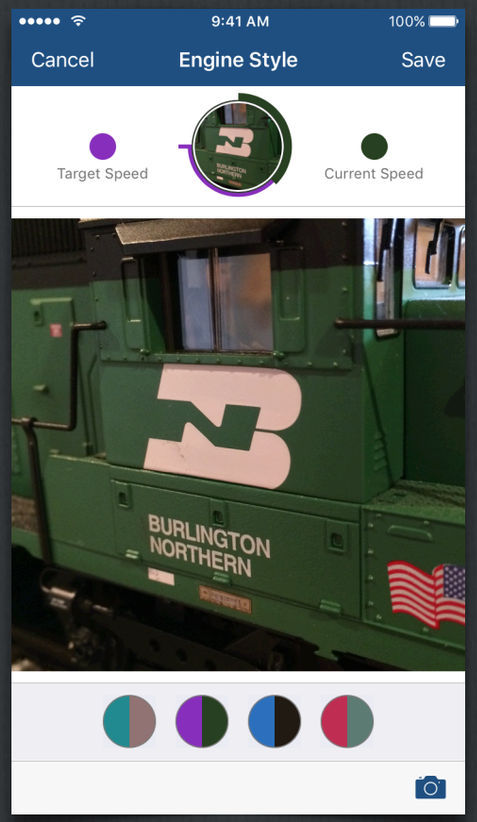
High Rail

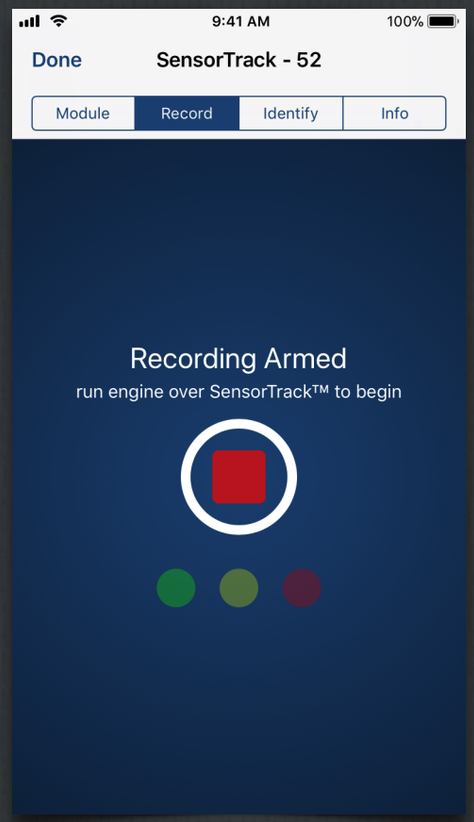
highrailcompany.com

- ☐ my "playground" app
- ☐ originally developed in ObjC
- ☐ re-wrote from the ground up in Swift
- ☐ now Swift 4.1
- ☐ 10 unique gestures
- ☐ 3D Touch engine speed control







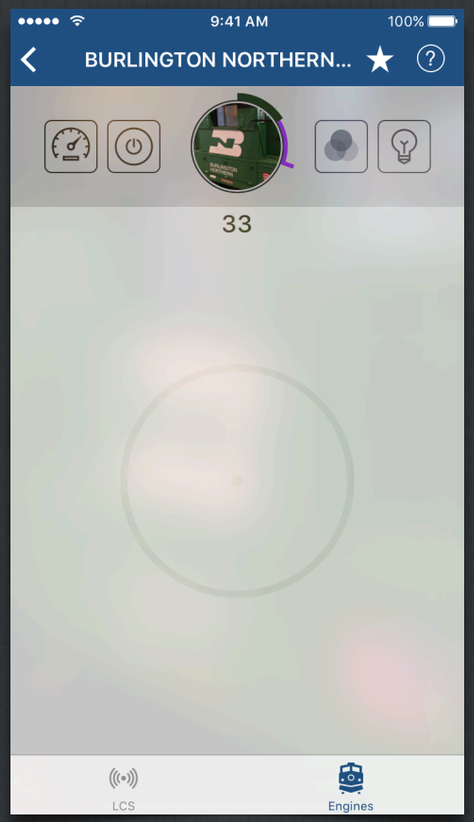


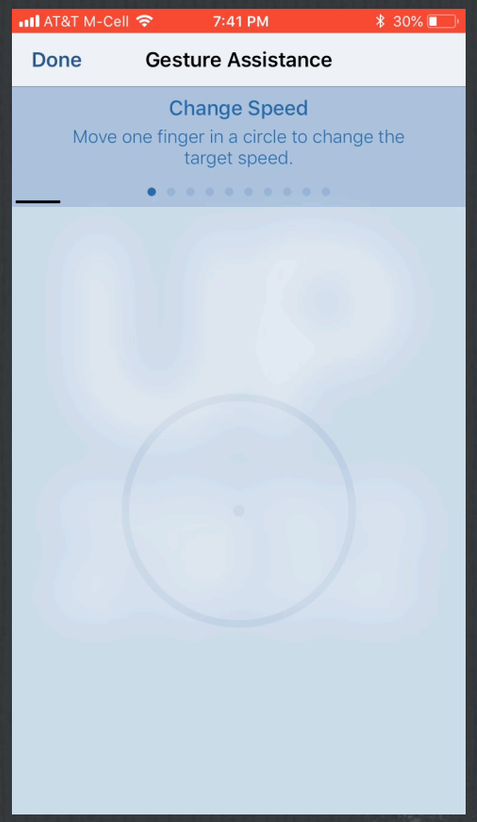
It's good to have side projects

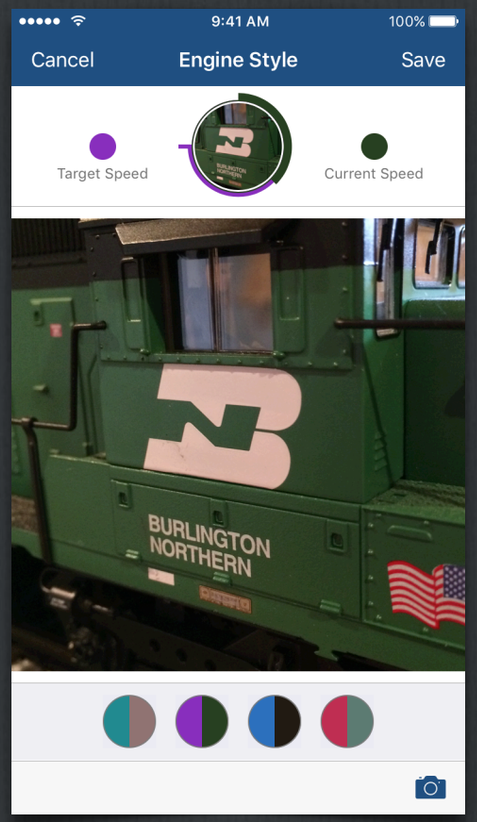
High Rail

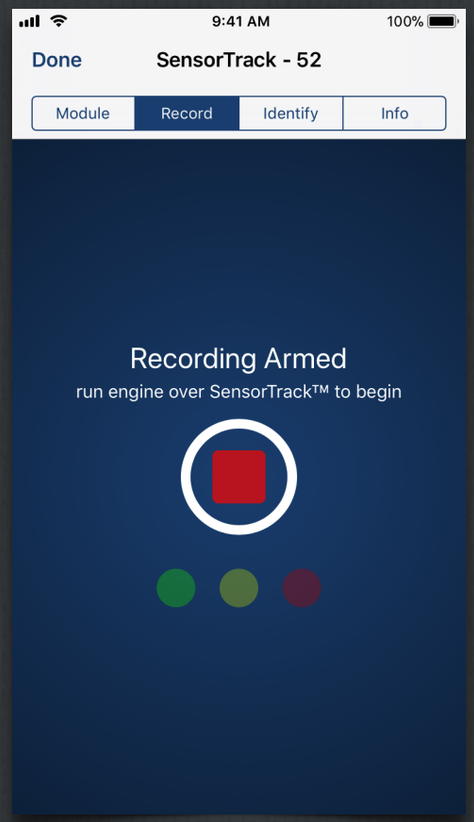
highrailcompany.com

- ☐ my "playground" app
- ☐ originally developed in ObjC
- ☐ re-wrote from the ground up in Swift
- ☐ now Swift 4.1
- ☐ 10 unique gestures
- ☐ 3D Touch engine speed control









Lionel Hardware Protocols

Lionel Hardware Protocols

- ☐ Lionel's Legacy Command protocols are freely available

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf
- Additional proprietary protocols are required to communicate with Lionel's Wi-Fi hardware

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf
- Additional proprietary protocols are required to communicate with Lionel's Wi-Fi hardware
  - establishing a Wi-Fi connection

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf
- Additional proprietary protocols are required to communicate with Lionel's Wi-Fi hardware
  - establishing a Wi-Fi connection
  - message framing

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf

- Additional proprietary protocols are required to communicate with Lionel's Wi-Fi hardware
  - establishing a Wi-Fi connection
  - message framing
  - heartbeats

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf

- Additional proprietary protocols are required to communicate with Lionel's Wi-Fi hardware
  - establishing a Wi-Fi connection
  - message framing
  - heartbeats
  - active and passive response handlers

Lionel Hardware Protocols

- Lionel's Legacy Command protocols are freely available
  - http://www.lionel.com/lcs/resources/LCS-LEGACY-Protocol-Spec-v1.22.pdf

- Additional proprietary protocols are required to communicate with Lionel's Wi-Fi hardware
  - establishing a Wi-Fi connection
  - message framing
  - heartbeats
  - active and passive response handlers
  - other hardware

We'll Touch On...

- ☐ Structs
- ☐ Memberwise initializers
- ☐ Protocols
- ☐ Protocol Extensions
- ☐ Domain Type Wrappers
- ☐ Basic Enums and Switch Statements

3-Byte Engine Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

3-Byte Engine Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

- Always 0xF8
- Identifies the message as an engine command

3-Byte Engine Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

- [ ] “A” is the engine’s ID (1 through 99)
- [ ] Requires some simple bit shifting

3-Byte Engine Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

- "C" is the action to take (i.e. ring bell, speed, etc.)
- Some actions have variable data

3-Byte Engine Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

- [ ] Command data takes up 9 bits
- [ ] The 9th bit will be treated as a special bit
- [ ] Doing so will help keep things simpler

Sample Engine Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

Immediate Stop	0	1	1	1	1	1	0	1	1
Change Speed (0 through 199)	0	D	D	D	D	D	D	D	D
Set Momentum (0 through 7)	0	1	1	0	0	1	D	D	D
Ring Bell	1	0	0	0	1	1	1	0	1
Whistle	1	1	1	1	0	D	D	D	D

Building Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol Command {
    var message: [UInt8] { get }
}
```

Building Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol Command {
    var message: [UInt8] { get }
}
```

Note:
An open-ended command definition is needed to support variable length commands, which is beyond the scope of this session.

Building Commands

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol EngineAddressableCommand: Command {
    var engineID: EngineID { get }
}
```

- [ ] Engine IDs are between 1 and 99
- [ ] A type-safe wrapper reduces bugs and simplifies unit testing

Failable Initializers

```
struct EngineID {

    /// The engine ID (value is between 1 and 99).
    let value: UInt8

    /// A failable initializer constrains the `EngineID` to
    /// values between 1 and 99.
    ///
    /// - Parameter value: the engine ID
    init?(_ value: Int) {
        switch value {
        case 1 ... 99:
            self.value = UInt8(value)
        default:
            return nil
        }
    }
}
```

Failable Initializers

```
struct EngineID {

    /// The engine ID (value is between 1 and 99).
    let value: UInt8

    /// A failable initializer constrains the `EngineID` to
    /// values between 1 and 99.
    ///
    /// - Parameter value: the engine ID
    init?(_ value: Int) {
        switch value {
        case 1 ... 99:
            self.value = UInt8(value)
        default:
            return nil
        }
    }
}
```

Initializing An Engine ID

```
let engineID: EngineID? = EngineID(2)
```

Initializing An Engine ID

```
let engineID = EngineID(2)
```

Initializing An Engine ID

```
let engineID = EngineID(2)
```

```
guard let engineID = EngineID(1) else {
    // Handle invalid engine ID
}

let command = OpenFrontCouplerCommand(engineID: engineID)
```

Equatable

https://github.com/apple/swift-evolution/blob/master/proposals/0185-synthesize-equatable-hashable.md

```swift
struct EngineID: Equatable {

    /// The engine ID (value is between 1 and 99).
    let value: UInt8

    . . .
}
```

```swift
extension EngineID {

  static func ==(lhs: EngineID, rhs: EngineID) -> Bool {
      return lhs.value == rhs.value
  }
}
```

Note:
Starting with Swift 4.1, we don’t have to implement the equals function because the compiler synthesizes the function. See link above for details.

Equatable

https://github.com/apple/swift-evolution/blob/master/proposals/0185-synthesize-equatable-hashable.md

```
struct EngineID: Equatable {

    /// The engine ID (value is between 1 and 99).
    let value: UInt8

    . . .
}
```

Note:
Starting with Swift 4.1, we don’t have to implement the equals function because the compiler synthesizes the function. See link above for details.

Defining An Engine Command

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol LegacyEngineCommand: EngineAddressableCommand {


}
```

Defining An Engine Command

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol LegacyEngineCommand: EngineAddressableCommand {


}
```

Defining An Engine Command

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol LegacyEngineCommand: EngineAddressableCommand {

    var commandField: UInt8 { get }
}
```

Defining An Engine Command

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
protocol LegacyEngineCommand: EngineAddressableCommand {
    var enableBit9: Bool { get }
    var commandField: UInt8 { get }
}
```

Protocol Extension

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
extension LegacyEngineCommand {

    var message: [UInt8] {
        var buffer = [UInt8](repeating: 0, count: 3)




        return buffer
    }
}
```

Protocol Extension

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
extension LegacyEngineCommand {

    var message: [UInt8] {
        var buffer = [UInt8](repeating: 0, count: 3)
        buffer[0] = 0xF8



        return buffer
    }
}
```

Protocol Extension

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
extension LegacyEngineCommand {

    var message: [UInt8] {
        var buffer = [UInt8](repeating: 0, count: 3)
        buffer[0] = 0xF8
        buffer[1] = (engineID.value << 1)


        return buffer
    }
}
```

Protocol Extension

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
extension LegacyEngineCommand {

    var message: [UInt8] {
        var buffer = [UInt8](repeating: 0, count: 3)
        buffer[0] = 0xF8
        buffer[1] = (engineID.value << 1) | (enableBit9 ? 1 : 0)


        return buffer
    }
}
```

Protocol Extension

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	0	0	A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

```
extension LegacyEngineCommand {

    var message: [UInt8] {
        var buffer = [UInt8](repeating: 0, count: 3)
        buffer[0] = 0xF8
        buffer[1] = (engineID.value << 1) | (enableBit9 ? 1 : 0)
        buffer[2] = commandField

        return buffer
    }
}
```

Building Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C

Immediate Stop	0	1	1	1	1	1	0	1	1
Open Front Coupler	1	0	0	0	0	0	1	0	1
Open Rear Coupler	1	0	0	0	0	0	1	1	0
Ring Bell	1	1	1	1	1	0	0	D	D

Building Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C
Immediate Stop							0	1	1	1	1	1	0	1	1

```
struct StopEngineCommand: LegacyEngineCommand {
    let engineID: EngineID
    let enableBit9 = false
    let commandField: UInt8 = 0xFB
}
```

```
let engineID: EngineID = . . .
let turnBellOffCommand = StopEngineCommand(engineID: engineID)
```

Building Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C
Open Front Coupler							1	0	0	0	0	0	1	0	1

```
struct OpenFrontCouplerCommand: LegacyEngineCommand {
    let engineID: EngineID
    let enableBit9 = true
    let commandField: UInt8 = 0x05
}
```

```
let engineID: EngineID = . . .
let command = OpenFrontCouplerCommand(engineID: engineID)
```

Building Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C
Open Rear Coupler							1	0	0	0	0	0	1	1	0

```
struct OpenRearCouplerCommand: LegacyEngineCommand {
    let engineID: EngineID
    let enableBit9 = true
    let commandField: UInt8 = 0x06
}
```

```
let engineID: EngineID = . . .
let command = OpenRearCouplerCommand(engineID: engineID)
```

Building Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C
Ring Bell							1	1	1	1	1	0	0	D	D

```
struct DynamicBellCommand: LegacyEngineCommand {
    let engineID: EngineID
    let enableBit9 = true
    let intensity: Intensity
}
```

Building Commands

Ring Bell	1	1	1	1	1	0	0	D	D

```
struct DynamicBellCommand: LegacyEngineCommand {

    enum Intensity: UInt8 {
        case soft = 1
        case moderate = 2
        case loud = 3
    }

    let engineID: EngineID
    let enableBit9 = true
    let intensity: Intensity

    var commandField: UInt8 {
        return UInt8(0xF0) | intensity.rawValue
    }
}
```

Building Commands

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A	A	A	A	A	A	A	C	C	C	C	C	C	C	C	C
Ring Bell							1	1	1	1	1	0	0	D	D

```
struct DynamicBellCommand: LegacyEngineCommand {
    let engineID: EngineID
    let enableBit9 = true
    let intensity: Intensity
}
```

```
let engineID: EngineID = . . .
let command = DynamicBellCommand(engineID: engineID, intensity: .loud)
```

Memberwise Initializers

```
struct StopEngineCommand: LegacyEngineCommand {
    let engineID: EngineID
    let enableBit9 = false
    let commandField: UInt8 = 0xFB
}
```

```
let engineID: EngineID = . . .
let turnBellOffCommand = StopEngineCommand(engineID: engineID)
```

Where did the initializer come from?

Memberwise Initializers

- ☐ Compiler generates an initializer if one is not provided for all stored properties that do not have default values

- ☐ Not exposed outside target (i.e. internal access only)

Xcode Playground Demo

Building Messages

Being Lazy with `lazy var` Stored Properties

Lazy Stored Properties

Lazy Stored Properties

- Delays the initialization of a stored property

Lazy Stored Properties

- Delays the initialization of a stored property
- Useful for (but not limited to):

Lazy Stored Properties

- [ ] Delays the initialization of a stored property
- [ ] Useful for (but not limited to):
  - [ ] expensive resources

Lazy Stored Properties

- ☐ Delays the initialization of a stored property
- ☐ Useful for (but not limited to):
  - ☐ expensive resources
  - ☐ optional resources

Lazy Stored Properties

- ☐ Delays the initialization of a stored property
- ☐ Useful for (but not limited to):
  - ☐ expensive resources
  - ☐ optional resources
  - ☐ views requiring two-step initialization

Lazy Stored Properties

- ☐ Delays the initialization of a stored property
- ☐ Useful for (but not limited to):
  - ☐ expensive resources
  - ☐ optional resources
  - ☐ views requiring two-step initialization

Lazy Stored Properties

- ☐ Delays the initialization of a stored property
- ☐ Useful for (but not limited to):
  - ☐ expensive resources
  - ☐ optional resources
  - ☐ views requiring two-step initialization
- ☐ Mutable

Lazy Stored Properties

- ☐ Delays the initialization of a stored property
- ☐ Useful for (but not limited to):
  - ☐ expensive resources
  - ☐ optional resources
  - ☐ views requiring two-step initialization
- ☐ Mutable

We'll focus on this

General Guidelines

```
fileprivate lazy var someThing = self.lazySomeThing()
```

- [ ] Only visible within the enclosing type and source file

General Guidelines

```
fileprivate lazy var someThing = self.lazySomeThing()
```

- [ ] It is what it is

General Guidelines

```
fileprivate lazy var someThing = self.lazySomeThing()
```

- [ ] Method name convention
  self.lazy<PropertyName>()

- [ ] Method location
  - [ ] shoved towards the end of the file
  - [ ] reduces file clutter for boring initialization code

General Guidelines

```
final class SomeViewController: UIViewController {

    fileprivate lazy var yesButton = self.lazyYesButton()
    fileprivate lazy var noButton = self.lazyNoButton()

      // Interesting code here
}

// Shove lazy property creation to the end of the file.

extension SomeViewController {

    fileprivate func lazyYesButton() -> UIButton {
        return makeButton(withTitle: "Yes")
    }
```

General Guidelines

```swift
// Shove lazy property creation to the end of the file.
extension SomeViewController {

    fileprivate func lazyYesButton() -> UIButton {
        return makeButton(withTitle: "Yes")
    }

    fileprivate func lazyMaybeButton() -> UIButton {
        return makeButton(withTitle: "Maybe")
    }

    fileprivate func makeButton(withTitle title: String) -> UIButton {
        let button = UIButton(type: .system)
        button.translatesAutoresizingMaskIntoConstraints = false

        button.setTitle(title, for: .normal)

        return button
    }
}
```

A note on classes

```
final class SomeViewController: UIViewController {
    . . .
}
```

- [ ] When creating reference types (i.e. classes)
  - [ ] get into the habit of typing `final class`
  - [ ] only open for subclassing when needed

- [ ] reach for other design patterns instead of subclassing
  - [ ] you know, “composition over inheritance”

Xcode Playground Demo

Being Lazy

State Transitions

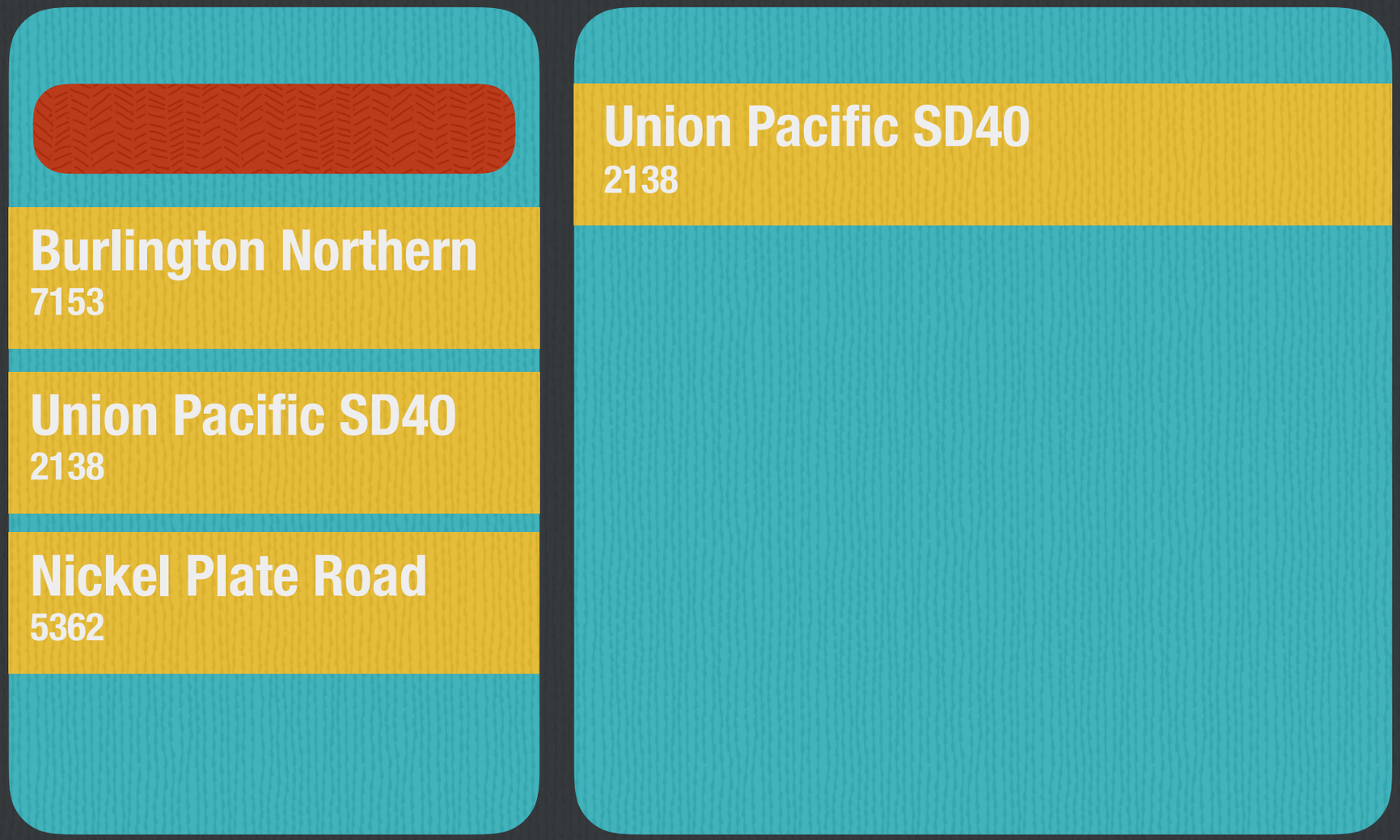
View State Transitions

- [ ] Views change as “data” changes
- [ ] Handling state transitions can be tricky
  - [ ] Utilize exhaustive switch statements
  - [ ] Switch on a tuple (optional old, optional new)

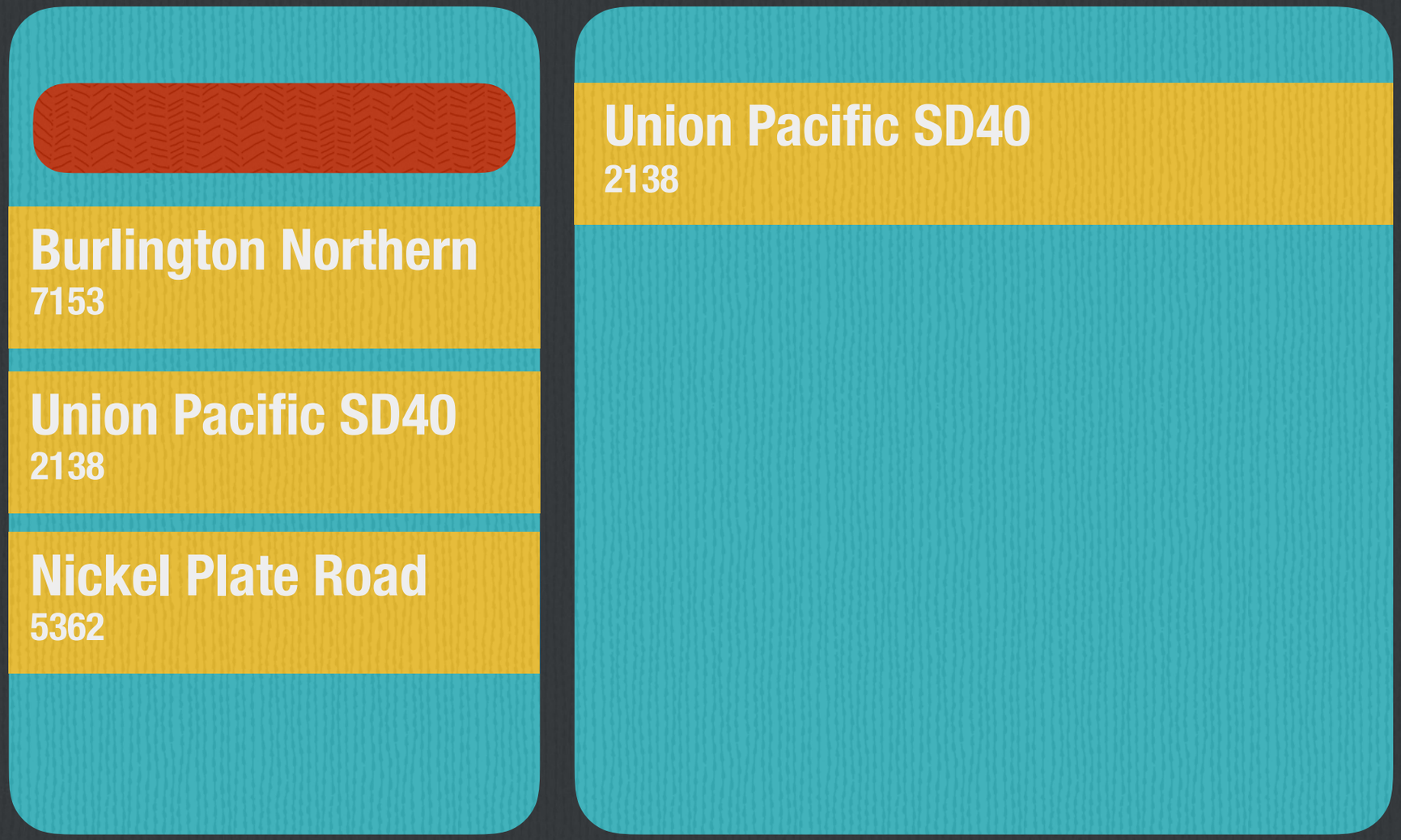
```
switch (old, new)
```

View State Transitions

View State Transitions



View State Transitions



View State Transitions

View State Transitions

View State Transitions

```
final class EngineViewSynchronizer {

    fileprivate let nameLabel: UILabel
    fileprivate let roadNumberLabel: UILabel

    var engine: Engine? {
        didSet {
            transition(from: oldValue, to: engine)
        }
    }

    init(nameLabel: UILabel, roadNumberLabel: UILabel) {
        self.nameLabel = nameLabel
        self.roadNumberLabel = roadNumberLabel
    }
}
```

View State Transitions

```
extension EngineViewSynchronizer {

    fileprivate func transition(from: Engine?, to: Engine?) {
        switch (from, to) {
        case (.none, .none):
            break
        case (.none, .some(let engine)):
            synchronizeInternalViews(with: engine)
        case (.some(_), .some(let engine)):
            synchronizeInternalViews(with: engine)
        case (.some(_), .none):
            clearInternalViews()
        }
    }
}
```

View State Transitions

```
extension EngineViewSynchronizer {

    fileprivate func transition(from: Engine?, to: Engine?) {
        switch (from, to) {
        case (.none, .none):
            break
        case (.none, .some(let engine)):
            synchronizeInternalViews(with: engine)
        case (.some(_), .some(let engine)):
            synchronizeInternalViews(with: engine)
        case (.some(_), .none):
            clearInternalViews()
        }
    }
}
```

View State Transitions

```
extension EngineTableViewCellSynchronizer {

    fileprivate func transition(from: Engine?, to: Engine?) {
        switch (from, to) {
        case (.none, .none):
            break
        case (_, .some(let engine)):
            synchronizeInternalViews(with: engine)
        case (.some(_), .none):
            clearInternalViews()
        }
    }
}
```

View State Transitions

```
extension EngineTableViewCellSynchronizer {

    private func synchronizeInternalViews(with engine: Engine) {
        nameLabel.text = engine.name
        roadNumberLabel.text = engine.roadNumber
    }

    private func clearInternalViews() {
        nameLabel.text = nil
        roadNumberLabel.text = nil
    }
}
```

General Template

```
extension YourExcitingThingThatCanTransitionState {

    fileprivate func transition(from: Foo?, to: Foo?) {
        switch (from, to) {
        case (.none, .none):
            break
        case (.none, .some(let foo)):
            setUp(with: foo)
        case (.some(let oldFoo), .some(let newFoo)):
            tearDown(oldFoo)
            setUp(newFoo)
        case (.some(let oldFoo), .none):
            tearDown(oldFoo)
        }
    }
}
```

Xcode Playground Demo

Optionals

Exhaustive Switch Statements

View State Transitions

Quick look at server-side Swift

Swift On The Server

- [ ] https://swift.org/server-apis/
- [ ] https://github.com/apple/swift-nio
- [ ] https://github.com/apple/swift-protobuf
- [ ] https://github.com/vapor/vapor
- [ ] https://github.com/ibm-swift/kitura

Vapor 3

- [ ] Built on Apple's SwiftNIO
  - [ ] Includes Futures and Promises
- [ ] Requires Swift 4.1
- [ ] Still in beta (release candidate)

Vapor 3 Demo

“Installing”

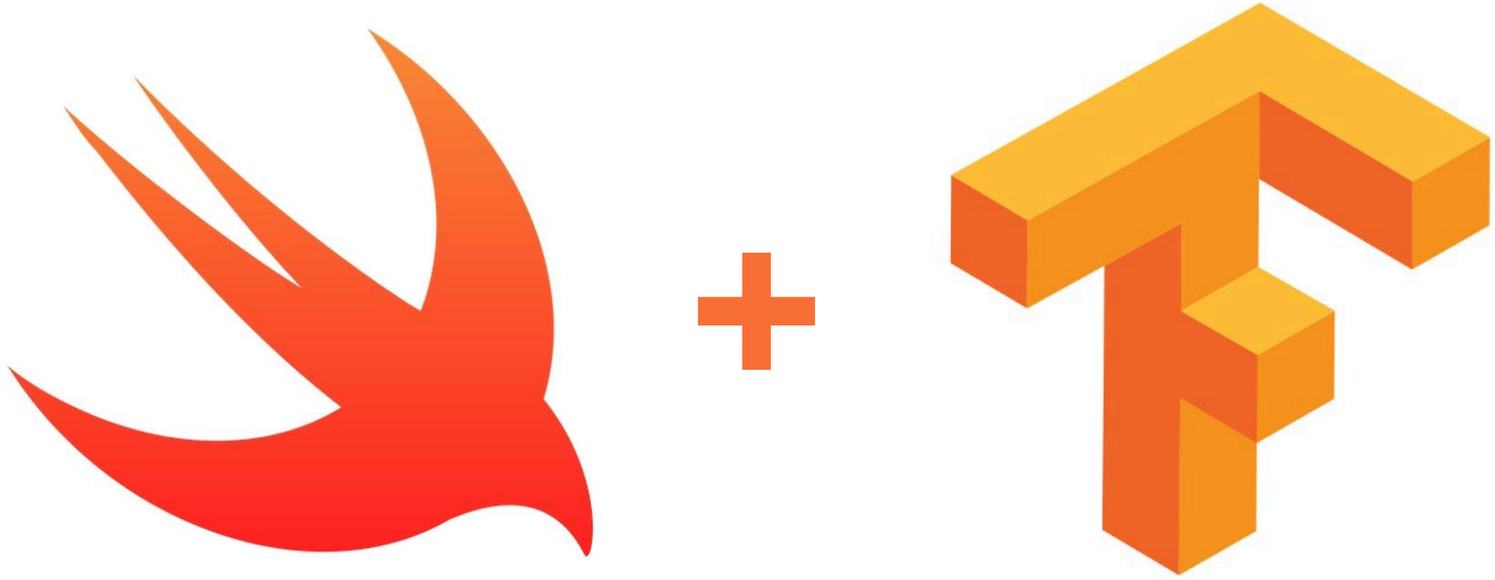
Routes

Middleware

Persistence

Swift + ML

Swift for TensorFlow



Swift and Machine Learning

- [ ] https://www.tensorflow.org/community/swift
- [ ] https://github.com/apple/swift-evolution/blob/master/proposals/0195-dynamic-member-lookup.md
- [ ] See Chris Lattner discuss
  - [ ] https://www.youtube.com/watch?v=Yze693W4MaU

Lots to discuss this year

Ideas

- ☐ Dive deeper into Vapor 3
- ☐ Promises and Futures
- ☐ Grand Central Dispatch
- ☐ Design Patterns
- ☐ Data Structures and Algorithms
- ☐ Unit Testing
- ☐ Advanced Auto Layout
- ☐ SpriteKit, SceneKit, ARKit, Vision
- ☐ Lots of UIKit and Core Animation topics
- ☐ TensorFlow for Swift + CoreML
- ☐ Unit Testing
- ☐ Swift Generics

Let me know if you would like to present

One more thing…

My team is hiring

we build neat stuff

Thank you!