Swift in Practice

Finding more issues at compile-time

Session 411

Ted Kremenek Swift Team Manager
Alex Migicovsky Sample Code Engineer

This Talk

This Talk

Take advantage of new APIs while deploying to older OS releases

This Talk

Take advantage of new APIs while deploying to older OS releases

Enforce expected application behavior using enums and protocols

Taking Advantage of New APIs

Adopting New APIs

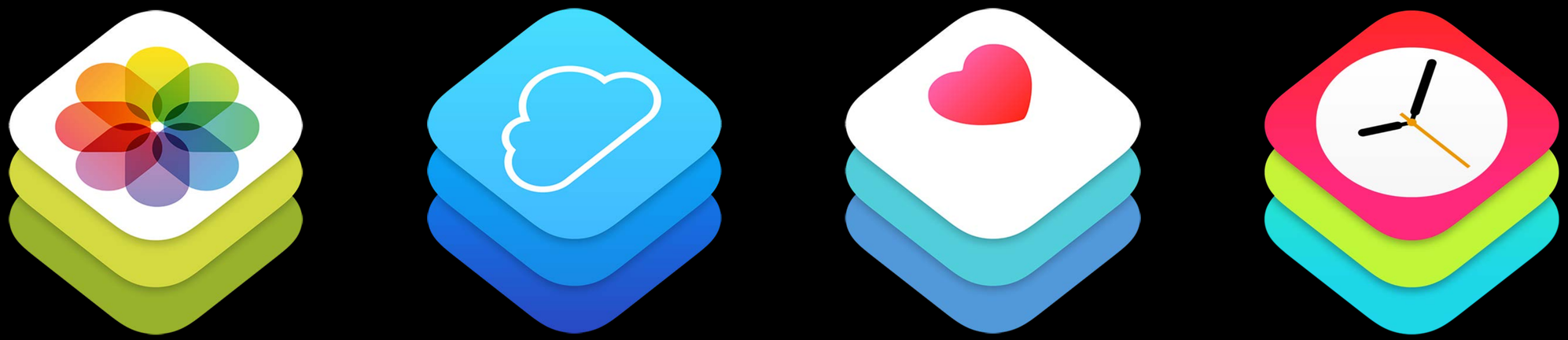
Each OS release comes up with new APIs for apps

Brings new functionality that enables richer experiences for users

Adopting New APIs

Each OS release comes up with new APIs for apps

Brings new functionality that enables richer experiences for users



Problem: Users on Different OS Releases

Problem: Users on Different OS Releases

Should you change your app to require the latest OS?

Problem: Users on Different OS Releases

Should you change your app to require the latest OS?

Should you hold back on adopting new features?

Problem: Users on Different OS Releases

Should you change your app to require the latest OS?

Should you hold back on adopting new features?

Adopt new features and support the older OS releases

Reality

Reality

It is possible to do this today...

...but now it is pain-free in Swift 2

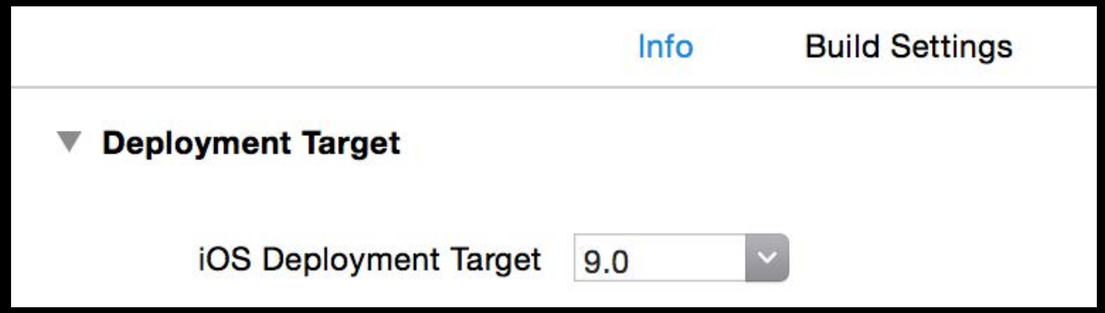
Base SDK and Deployment Target

Base SDK and Deployment Target

Always use the **Latest SDK** to access complete set of APIs...

Base SDK and Deployment Target

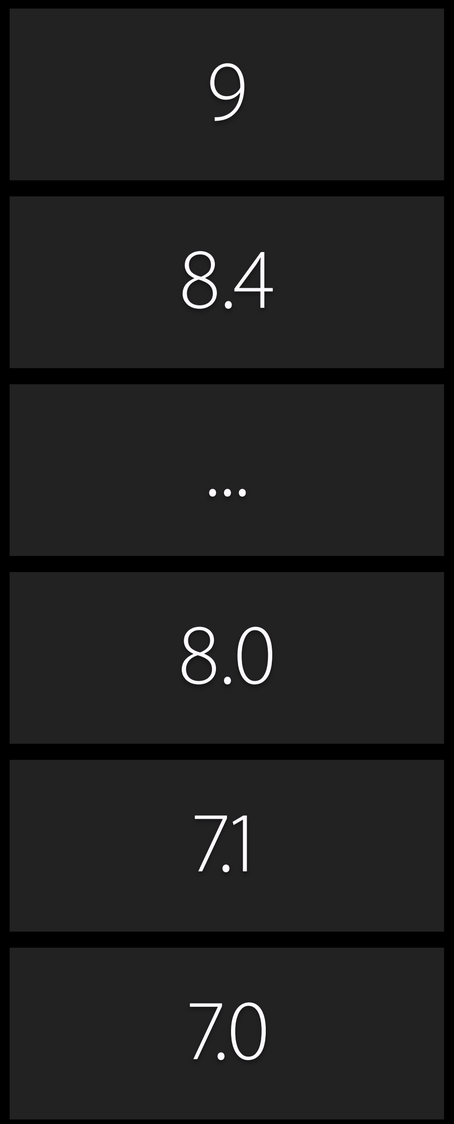
Always use the **Latest SDK** to access complete set of APIs...



... use **Deployment Target** to set an application's minimum supported OS release

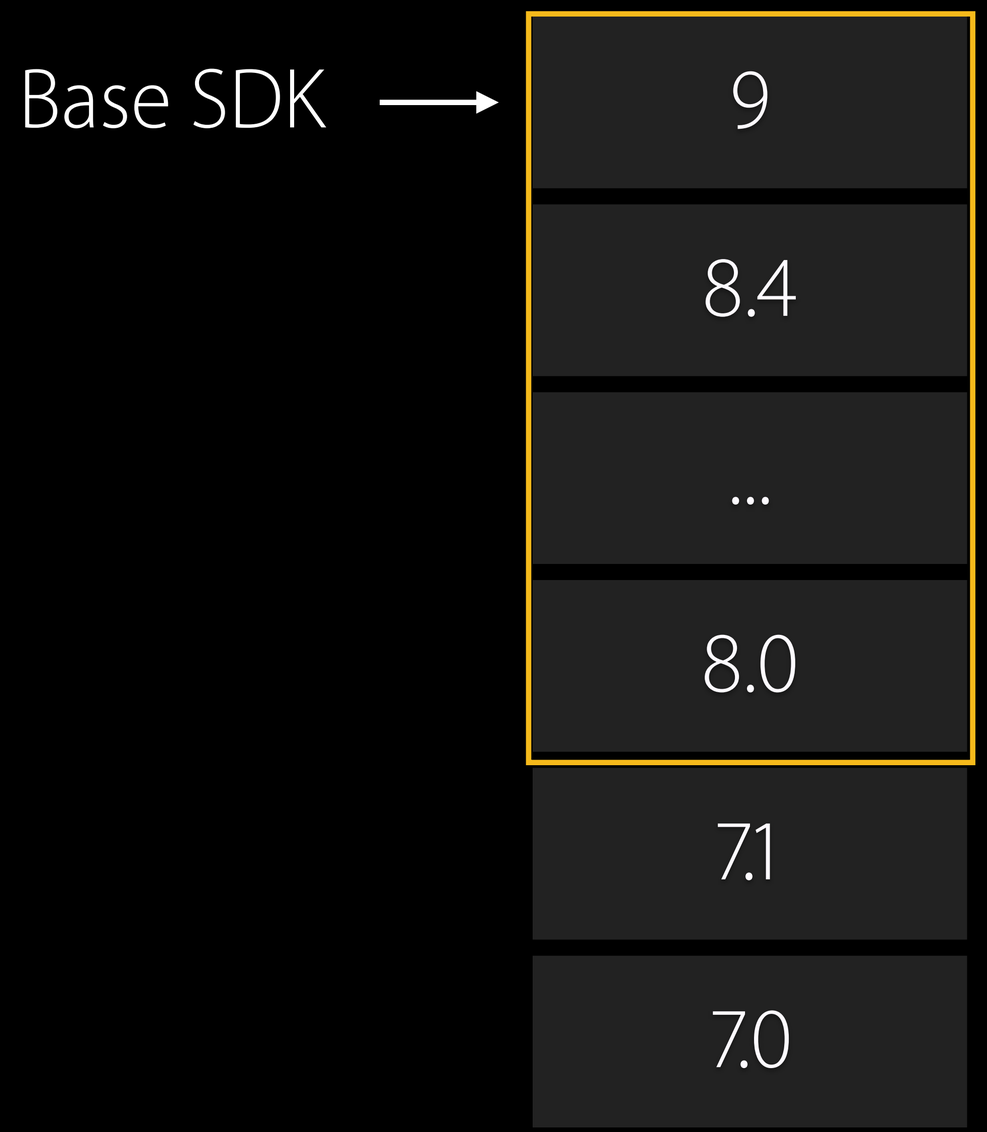
Example: Supporting Multiple iOS Releases

Example: Supporting Multiple iOS Releases



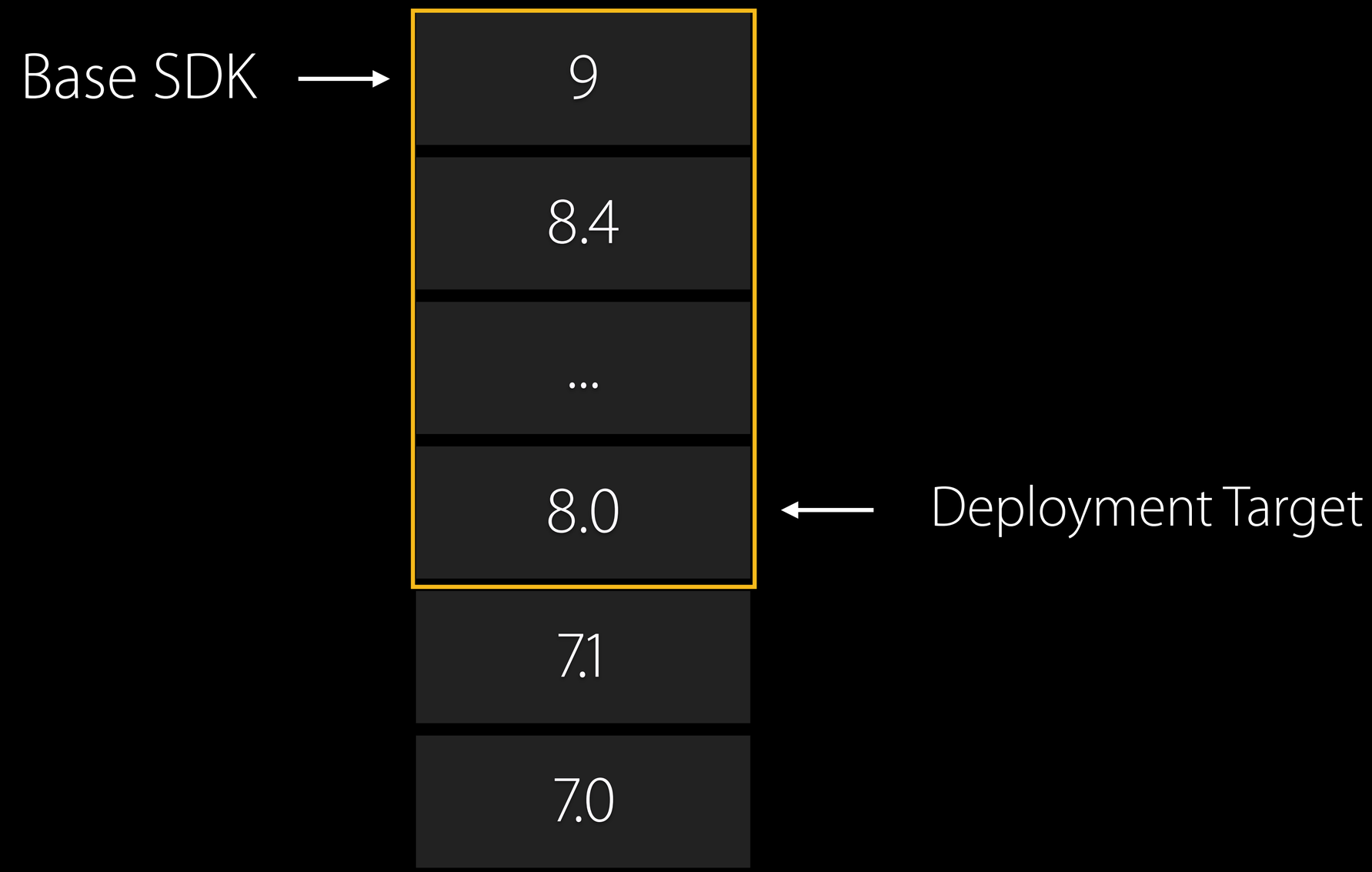
Example: Supporting Multiple iOS Releases

Example: Supporting Multiple iOS Releases



Example: Supporting Multiple iOS Releases

Example: Supporting Multiple iOS Releases



Adopting New APIs While Deploying Back

Existing very painful method

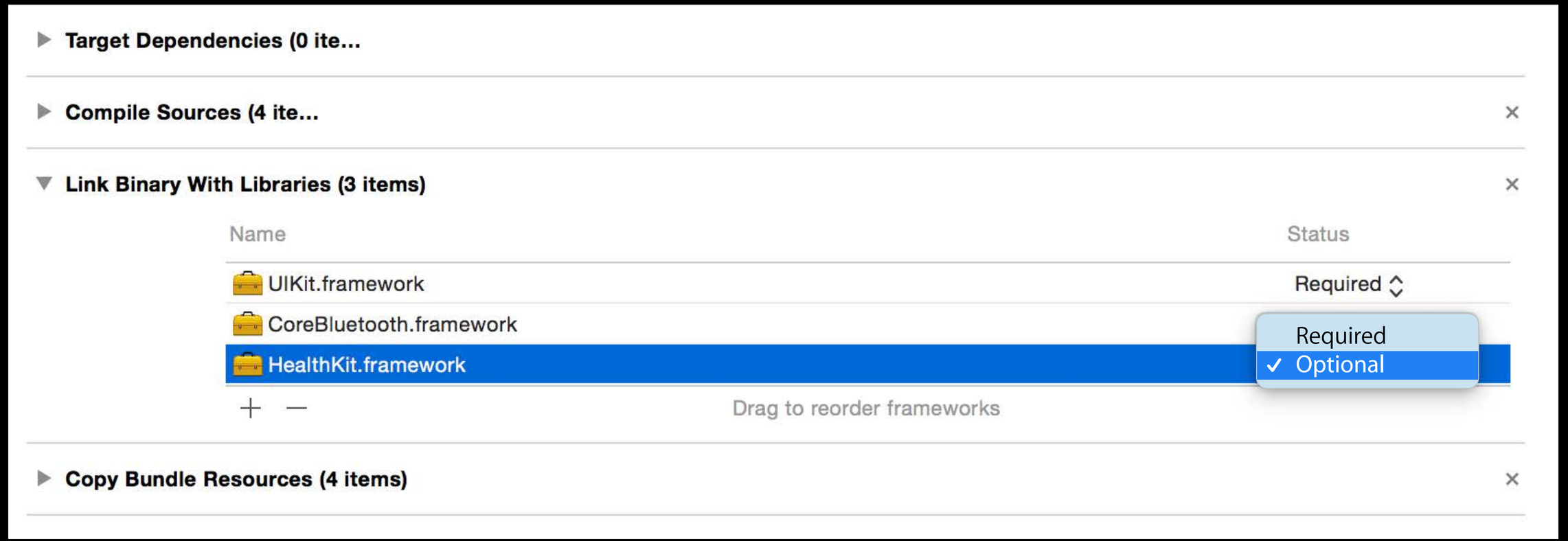
Adopting New APIs While Deploying Back

Existing very painful method

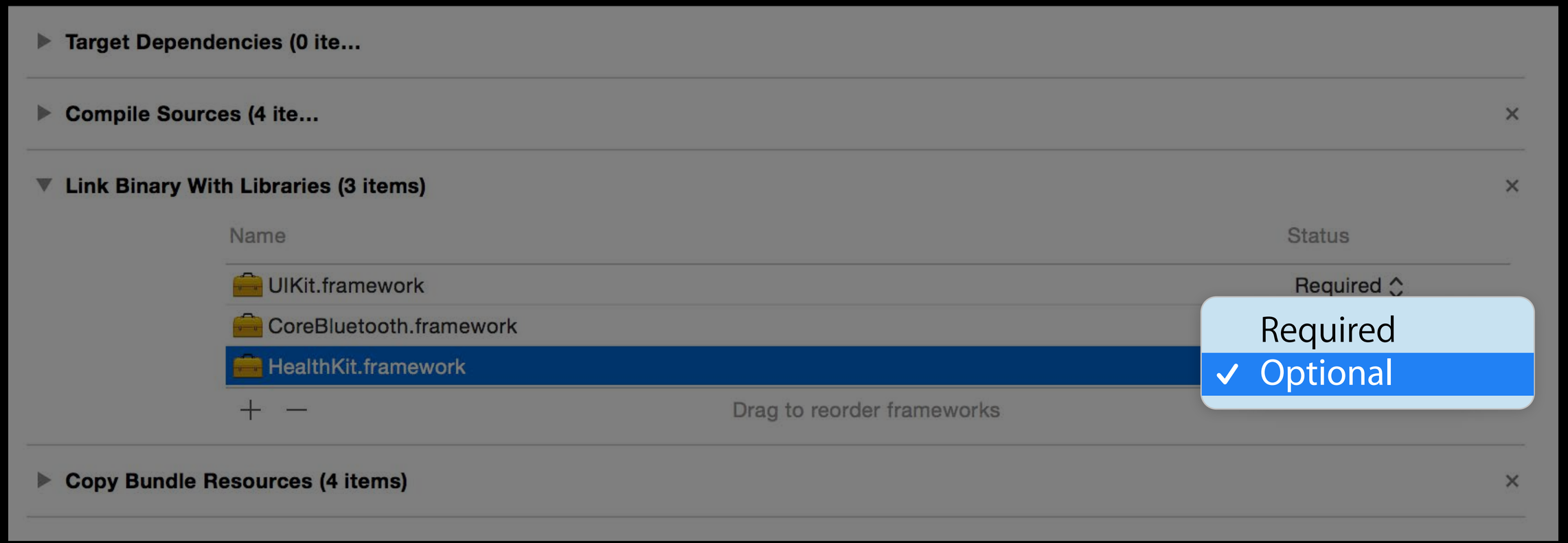
Previously you needed separate considerations for adopting each of the following:

- Frameworks
- Classes
- Methods
- Functions
- Enums

Manually Mark Frameworks as Optional



Manually Mark Frameworks as Optional



Manually Check if a Class Is Available

```
if ([NSDataAsset class]) {
    NSDataAsset *asset = [[NSDataAsset alloc] initWithName:@"Dragon"];
}
```

Manually Check if a Class Is Available

```
if ([NSDataAsset class]) {
    NSDataAsset *asset = [[NSDataAsset alloc] initWithName:@"Dragon"];
}
```

Class previously may have been internal API

Manually Check if a Class Is Available

Class previously may have been internal API

Manually Check if a Class Is Available

```
if ([NSDataAsset class]) {
    NSDataAsset *asset = [[NSDataAsset alloc] initWithName:@"Dragon"];
}
```

Manually Check if a Class Is Available

```
if ([NSData class]) {
     NSDataAsset *asset = [[NSDataAsset alloc] initWithName:@"Dragon"];
}
```

Manually Check if a Class Is Available

```
if ([NSData class]) {
     NSDataAsset *asset = [[NSDataAsset alloc] initWithName:@"Dragon"];
}
```

Easy to make mistakes

Manually Check if a Method Is Available

```
if ([view respondsToSelector:@selector(setSemanticContentAttribute:)]) {
    view.semanticContentAttribute = UISemanticContentAttributePlayback;
}
```

Manually Check if a Method Is Available

```
if ([view respondsToSelector:@selector(setSemanticContentAttribute:)]) {
    view.semanticContentAttribute = UISemanticContentAttributePlayback;
}
```

- Easy to make mistakes
- Different syntax from classes

Manually Check if a Function Is Available

```
if (&CGPathCreateWithRoundedRect) {
    path = CGPathCreateWithRoundedRect(rect, cornerWidth,
                                       cornerHeight, transform);
    ...
}
```

- ⊗ Easy to make mistakes
- ⊗ Yet another syntax!

Enums?

You are stuck with a manual OS version check

```
typedef NS_ENUM(NSInteger, UIModalPresentationStyle) {
      UIModalPresentationFullScreen = 0,
      UIModalPresentationPageSheet NS_ENUM_AVAILABLE_IOS(3_2),
      UIModalPresentationFormSheet NS_ENUM_AVAILABLE_IOS(3_2),
      UIModalPresentationCurrentContext NS_ENUM_AVAILABLE_IOS(3_2),
      UIModalPresentationCustom NS_ENUM_AVAILABLE_IOS(7_0),
      UIModalPresentationOverFullScreen NS_ENUM_AVAILABLE_IOS(8_0),
      UIModalPresentationOverCurrentContext NS_ENUM_AVAILABLE_IOS(8_0),
      UIModalPresentationPopover NS_ENUM_AVAILABLE_IOS(8_0),
      UIModalPresentationNone NS_ENUM_AVAILABLE_IOS(7_0) = -1
};
```

Enums?

You are stuck with a manual OS version check

```
typedef NS_ENUM(NSInteger, UIModalPresentationStyle) {
      UIModalPresentationFullScreen = 0,
      UIModalPresentationPageSheet NS_ENUM_AVAILABLE_IOS(3_2),
      UIModalPresentationFormSheet NS_ENUM_AVAILABLE_IOS(3_2),
      UIModalPresentationCurrentContext NS_ENUM_AVAILABLE_IOS(3_2),
      UIModalPresentationCustom NS_ENUM_AVAILABLE_IOS(7_0),
      UIModalPresentationOverFullScreen NS_ENUM_AVAILABLE_IOS(8_0),
      UIModalPresentationOverCurrentContext NS_ENUM_AVAILABLE_IOS(8_0),
      UIModalPresentationPopover NS_ENUM_AVAILABLE_IOS(8_0),
      UIModalPresentationNone NS_ENUM_AVAILABLE_IOS(7_0) = -1
};
```

You are stuck with a manual OS version check 😩

Good luck getting that right

Observations

Deploying to earlier OS is technically possible

It is easy to get wrong

Different syntax for each availability check

Failure occurs on earlier OS releases, which are less tested in practice

Adopting New APIs while Deploying Back

The new  way

Adopting New APIs while Deploying Back

The new  way

Focus on using new APIs to build features

Adopting New APIs while Deploying Back

The new  way

Focus on using new APIs to build features

Compiler emits error if API is unsafely used

- Unified syntax for conditionally using all API kinds
- No special handling of optional frameworks needed

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

9.0

8.0

7.0

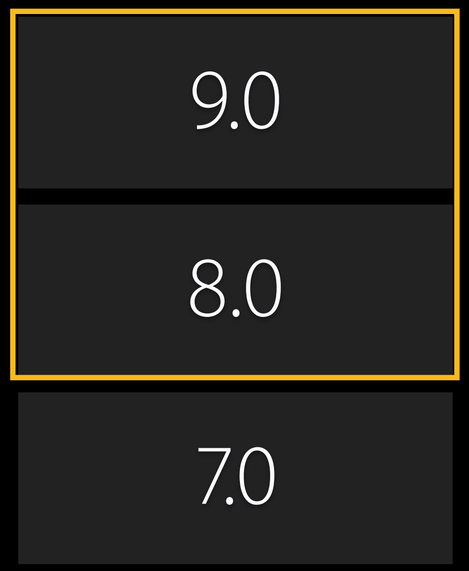
```
@available(iOS 2.0, *)
class CLLocationManager
```

```
@available(iOS 8.0, *)
func requestWhenInUseAuthorization()
```

API is always available

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

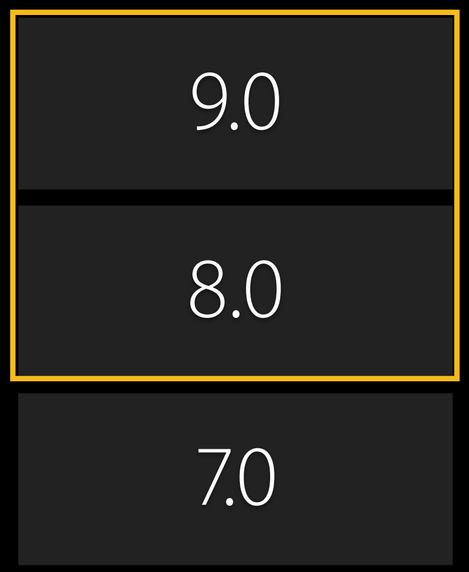



```
@available(iOS 2.0, *)
class CLLocationManager
```

```
@available(iOS 8.0, *)
func requestWhenInUseAuthorization()
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```



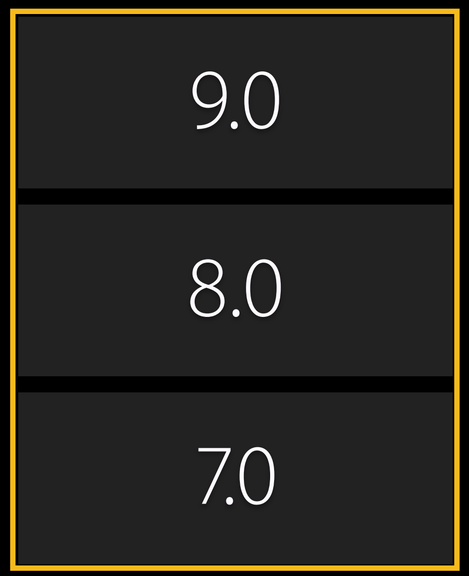

```
@available(iOS 2.0, *)
class CLLocationManager
```

```
@available(iOS 8.0, *)
func requestWhenInUseAuthorization()
```

API is always available

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```




```
@available(iOS 2.0, *)
class CLLocationManager
```

```
@available(iOS 8.0, *)
func requestWhenInUseAuthorization()
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

9.0

8.0

7.0

```
@available(iOS 2.0, *)
class CLLocationManager
```

```
@available(iOS 8.0, *)
func requestWhenInUseAuthorization()
```

API is used when it may not be available!

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

```
error: ‘requestWhenInUseAuthorization’
is only available on iOS 8.0 or newer
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
locationManager.requestWhenInUseAuthorization()
```

```
error: ‘requestWhenInUseAuthorization’
is only available on iOS 8.0 or newer
```

```
note: guard with version check?
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
if #available(iOS 8.0, *) {
    locationManager.requestWhenInUseAuthorization()
}
```

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
if #available(iOS 8.0, *) {
    locationManager.requestWhenInUseAuthorization()
}
```

Compiler generates runtime check for host version

Compile-Time API Availability Checking

```
let locationManager = CLLocationManager()
if #available(iOS 8.0, *) {
    locationManager.requestWhenInUseAuthorization()
}
```

Compiler generates runtime check for host version

Compiler infers the minimum OS version needed from the APIs used

Why Check Based on OS Version?

Why Check Based on OS Version?

Features are defined by a collection of APIs

- Checking for one API does not imply a collection of APIs are available
- Features are tied to OS versions
- Users are tied to OS versions

Why Check Based on OS Version?

Features are defined by a collection of APIs

- Checking for one API does not imply a collection of APIs are available
- Features are tied to OS versions
- Users are tied to OS versions

Compiler-enforced

- Availability checks are reliable
- Unified syntax for availability checks

Multiple Platforms

```
if #available(iOS 9.0, *) {
    let asset = NSDataAsset(name: "Dragon")
    ...
}
```

Multiple Platforms

```
if #available(iOS 9.0, OSX 10.11, *) {
    let asset = NSDataAsset(name: "Dragon")
    ...
}
```

Multiple Platforms

```
if #available(iOS 9.0, OSX 10.11, *) {
    let asset = NSDataAsset(name: "Dragon")
    ...
}
```

The * indicates “require minimum deployment target for other platforms”

Writing * is mandatory to call out control-flow

Expressing Conditional Logic

```
if #available(iOS 9.0, OSX 10.11, *) {
    let asset = NSDataAsset(name: "Dragon")
    ...
}
```

Expressing Conditional Logic

```
if #available(iOS 9.0, OSX 10.11, *) {
    let asset = NSDataAsset(name: "Dragon")
    ...
}
```

Could be a lot of code within the `if` block, but none afterwards

Expressing Conditional Logic

```
guard #available(iOS 9.0, OSX 10.11, *) else { return }
let asset = NSDataAsset(name: "Dragon")
...
```

Expressing Conditional Logic

```
guard #available(iOS 9.0, OSX 10.11, *) else { return }
let asset = NSDataAsset(name: "Dragon")
...
```

Use a `guard` statement to bail out early (when applicable)

Useful for when the code exclusively focuses on using new APIs

Factoring Your Code

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    ...
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    ...
}

iOS7API()
...
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    ...
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
func myFunctionThatUsesiOS8() {
    // Use iOS 7 (or earlier) APIs.
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
func myFunctionThatUsesiOS8() {
    // Use iOS 7 (or earlier) APIs.
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
func myFunctionThatUsesiOS8() {
    // Use iOS 7 (or earlier) APIs.
    if #available(iOS 8.0, *) {
        iOS8API()
        ...
    }
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
func myFunctionThatUsesiOS8() {
    // Use iOS 7 (or earlier) APIs.
    if #available(iOS 8.0, *) {
        iOS8API()
        ...
    }
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
@available(iOS 8.0, *)
func myFunctionThatUsesiOS8() {
    // Use iOS 7 (or earlier) APIs.
    if #available(iOS 8.0, *) {
        iOS8API()
        ...
    }
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
@available(iOS 8.0, *)
func myFunctionThatUsesiOS8() {
    // Use iOS 7 (or earlier) APIs.
    if #available(iOS 8.0, *) {
        iOS8API()
        ...
    }
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
@available(iOS 8.0, *)
func myFunctionThatUsesiOS8() {
    iOS8API()
    ...
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}

iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
@available(iOS 8.0, *)
func myFunctionThatUsesiOS8() {
    iOS8API()
    ...
    otherFunctionThatUsesiOS8()
}

@available(iOS 8.0, *)
func otherFunctionThatUsesiOS8() {
    iOS8API()
    ...
}
```

Factoring Your Code

```
// Deployment target is iOS 7.
// Use iOS 7 (or earlier) APIs.

iOS7API()
...

if #available(iOS 8.0, *) {
    // Use iOS 8 (or earlier) APIs.
    iOS8API()
    myFunctionThatUsesiOS8()
}


iOS7API()
...

if #available(iOS 9.0, *) {
    // Use iOS 9 (or earlier) APIs.
    iOS8API()
    iOS9API()
}
```

```
@available(iOS 8.0, *)
func myFunctionThatUsesiOS8() {
    iOS8API()
    ...
    otherFunctionThatUsesiOS8()
}

@available(iOS 8.0, *)
func otherFunctionThatUsesiOS8() {
    iOS8API()
    ...
    if #available(iOS 9.0, *) {
        // Use iOS 9 APIs.
        myFunctionThatUsesiOS9()
    }
}

@available(iOS 9.0, *)
func myFunctionThatUsesiOS9() {
    iOS9API()
}
```

Factoring Your Code

@available on methods

```
class MyClass {
    @available(iOS 8.0, *)
    func myMethodThatUsesiOS8() {
        ...
    }

    func otherMethod() { ... }
}
```

Factoring Your Code

@available on methods

```
class MyClass {
    @available(iOS 8.0, *)
    func myMethodThatUsesiOS8() {
        ...
    }

    func otherMethod() { ... }
}
```

Factoring Your Code

@available on methods

```
class MyClass {
    @available(iOS 8.0, *)
    func myMethodThatUsesiOS8() {
        ...
    }

    func otherMethod() { ... }
}


let myClass = MyClass()
myClass.otherMethod()

if #available(iOS 8.0, *) {
    myClass.myMethodThatUsesiOS8()
    ...
}
```

Factoring Your Code

@available on entire classes

```
class MyClass {
    @available(iOS 8.0, *)
    func myMethodThatUsesiOS8() {
        ...
    }

    func otherMethod() { ... }
}
```

Factoring Your Code

@available on entire classes

```
@available(iOS 8.0, *)
class MyClass {
    func myMethodThatUsesiOS8() {
        ...
    }

    func otherMethod() { ... }
}
```

Factoring Your Code

@available on entire classes

```
@available(iOS 8.0, *)
class MyClass {
    func myMethodThatUsesiOS8() {
        ...
    }

    func otherMethod() { ... }
}
```

```
if #available(iOS 8.0, *) {
    let myClass = MyClass()
    myClass.otherMethod()
    myClass.myMethodThatUsesiOS8()
}
```

@available and Subclassing

@available and Subclassing

```
class CustomBlurView : UIView { ... }
```

@available and Subclassing

```
class CustomBlurView : UIView { ... }


func makeBlurView() -> UIView {
    if #available(iOS 8.0, *) {
        // Use newer UIKit view when available.
        return UIVisualEffectView(...)
    }
    return CustomBlurView(...)
}
```

@available and Subclassing

```
class CustomBlurView : UIView { ... }


func makeBlurView() -> UIView {
    if #available(iOS 8.0, *) {
        // Use newer UIKit view when available.
        return UIVisualEffectView(...)
    }
    return CustomBlurView(...)
}
```

```
let blurView = makeBlurView()
```

@available and Subclassing

```
class CustomBlurView : UIView { ... }


func makeBlurView() -> UIView {
    if #available(iOS 8.0, *) {
    |   // Use newer UIKit view when available.
    |   return UIVisualEffectView(...)
    }
    return CustomBlurView(...)
}

let blurView = makeBlurView()
```

API Availability Checking

Swift's Availability checking catches unsafe uses of newer APIs at compile-time

Unified syntax for availability checking

Factor your apps logic around available APIs

Enforcing Application Constraints

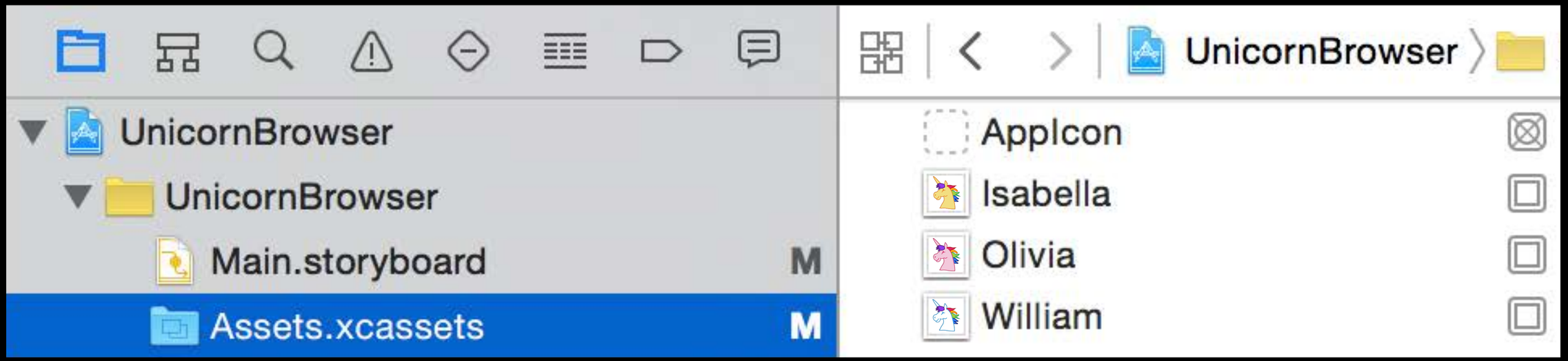
Alex Migicovsky Swift and Cocoa Lucid Dreamer

Enforcing Application Constraints

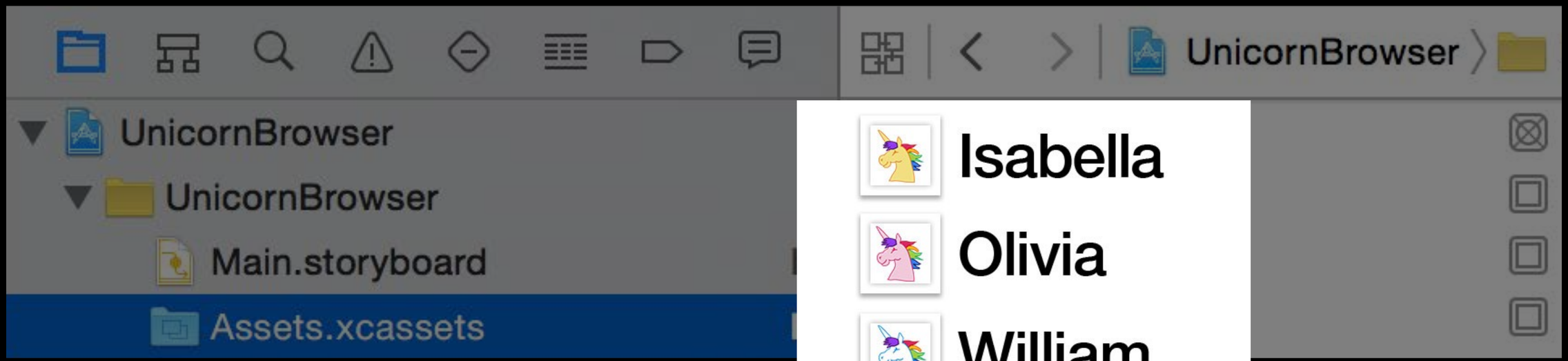
Alex Migicovsky Swift and Cocoa Lucid Dreamer

Asset Catalog Identifiers

Asset Catalog Identifiers



Asset Catalog Identifiers



Asset Catalog Identifiers

```
let isabellaImage = UIImage(named: "Isabella")!
```

```
let williamImage = UIImage(named: "William")!
```

```
let oliviaImage = UIImage(named: "Olivia")!
```

Asset Catalog Identifiers

```
let isabellaImage = UIImage(named: "Isabella")!
```

```
let williamImage = UIImage(named: "William")!
```

```
let oliviaImage = UIImage(named: "Olivia")!
```

Asset Catalog Identifiers

```
let isabellaImage = UIImage(named: "Isabella")!
```

```
let williamImage = UIImage(named: "William")!
```

```
let oliviaImage = UIImage(named: "Olivia")!
```

"William"

"Isabella"

"Olivia"

"Isabella"
"Isabella"
"Isabela"
"William"
"Olivia"
"Olivia"
"Isabella"
"Isabella"
"Oivia"
"Isabella"
"William"
"Isabella"
"William"
"William"
"Olivia"
"Olivia"
"Isabella"
"William"
"Olivia"
"Isabella"
"William"
"Olivia"
"Willim"
"Olivia"
"William"
"Olivia"
"Olivia"
"Isabella"
"Olivia"
"Isabella"
"William"
"Olivia"
"Oliia"
"Isabella"

"Isabella"
"Isabella"
"Isabela"
"William"
"Olivia"
"Olivia"
"Isabella"
"Isabella"
"Oivia"
"Isabella"
"William"
"William"
"Isabella"
"Olivia"
"Olivia"
"William"
"Isabella"
"Olivia"
"Olivia"
"William"
"Isabella"
"Olivia"
"Willim"
"Olivia"
"William"
"Olivia"
"Olivia"
"Isabella"
"Olivia"
"Isabella"
"William"
"Oliia"
"Olivia"
"Isabella"

String Constants

```
let IsabellaUnicornImageName = "Isabella"


let isabellaImage = UIImage(named: IsabellaUnicornImageName)!
```

String Constants

```
let IsabellaUnicornImageName = "Isabella"
```

```
let isabellaImage = UIImage(named: IsabellaUnicornImageName)!
```

String Constants

```
let IsabellaUnicornImageName = "Isabella"
```

```
let isabellaImage = UIImage(named: IsabellaUnicornImageName)!
```

String Constants

```
let IsabellaUnicornImageName = "Isabella"
```

```
let isabellaImage = UIImage(named: IsabellaUnicornImageName)!
```

```
let isabellaImage = UIImage(named: NSUbiquityIdentityDidChangeNotification)!
```

String Constants

```
let IsabellaUnicornImageName = "Isabella"
```

```
let isabellaImage = UIImage(named: IsabellaUnicornImageName)!
```

```
let isabellaImage = UIImage(named: NSUbiquityIdentityDidChangeNotification)!
```

```
fatal error: unexpectedly found nil while unwrapping an Optional value
```

Strings as Distinct Types

Stringly typed

Strings as Distinct Types

Stringly typed

Strings as Distinct Types

Strongly typed

Strings as Distinct Types

Strongly typed

Strings as Distinct Types

Strongly typed

Wanted:

- Mapping between strings and a new type
- UIImage non-failable init

Strings as Distinct Types

Strongly typed

Wanted:

- Mapping between strings and a new type
- UIImage non-failable init

Solution: application specific enums

Asset Catalog Identifiers

```
let isabellaImage = UIImage(assetIdentifier: .Isabella)
```

```
let williamImage = UIImage(assetIdentifier: .William)
```

```
let oliviaImage = UIImage(assetIdentifier: .Olivia)
```

Asset Catalog Identifiers

```
extension UIImage {
    enum AssetIdentifier: String {

    }
}
```

Asset Catalog Identifiers

```
extension UIImage {
    enum AssetIdentifier: String {
        case Isabella = "Isabella"
    }
}
```

Asset Catalog Identifiers

```
extension UIImage {
    enum AssetIdentifier: String {
        case Isabella = "Isabella"
        case William  = "William"
        case Olivia   = "Olivia"
    }
}
```

Asset Catalog Identifiers

```
extension UIImage {
    enum AssetIdentifier: String {
        case Isabella = "Isabella"
        case William  = "William"
        case Olivia   = "William"
    }
}
```

Asset Catalog Identifiers

```
extension UIImage {
    enum AssetIdentifier: String {
        case Isabella = "Isabella"
        case William  = "William"
        case Olivia   = "William"
    }
}
```

error: raw value for enum case is not unique

Asset Catalog Identifiers

```
extension UIImage {
    ...
    convenience init!(assetIdentifier: AssetIdentifier) {
        self.init(named: assetIdentifier.rawValue)
    }
}
```

Asset Catalog Identifiers

```
let isabellaImage = UIImage(assetIdentifier: .Isabella)
```

```
let williamImage = UIImage(assetIdentifier: .William)
```

```
let oliviaImage = UIImage(assetIdentifier: .Olivia)
```

Asset Catalog Identifiers

```
let isabellaImage = UIImage(assetIdentifier: .Isabella)
```

```
let williamImage = UIImage(assetIdentifier: .William)
```

```
let oliviaImage = UIImage(assetIdentifier: .Oliia)
```

Asset Catalog Identifiers

```
let isabellaImage = UIImage(assetIdentifier: .Isabella)
```

```
let williamImage = UIImage(assetIdentifier: .William)
```

```
let oliviaImage = UIImage(assetIdentifier: .Oliia)
```

```
error: 'UIImage.AssetIdentifier.Type' does not
have a member named 'Oliia'
```

Asset Catalog Identifiers

```
let isabellaImage = UIImage(assetIdentifier: .Isabella)
```

```
let williamImage = UIImage(assetIdentifier: .William)
```

```
let oliviaImage = UIImage(assetIdentifier: .Olivia)
```

AssetIdentifier Enum Benefits

- Centrally located constants

AssetIdentifier Enum Benefits

Centrally located constants

Doesn’t pollute global namespace

AssetIdentifier Enum Benefits

Centrally located constants

Doesn’t pollute global namespace

Must use one of the enum cases

AssetIdentifier Enum Benefits

- Centrally located constants
- Doesn't pollute global namespace
- Must use one of the enum cases
- Image initializers are not failable

Enums

Think about how you can use enums for compile time safety

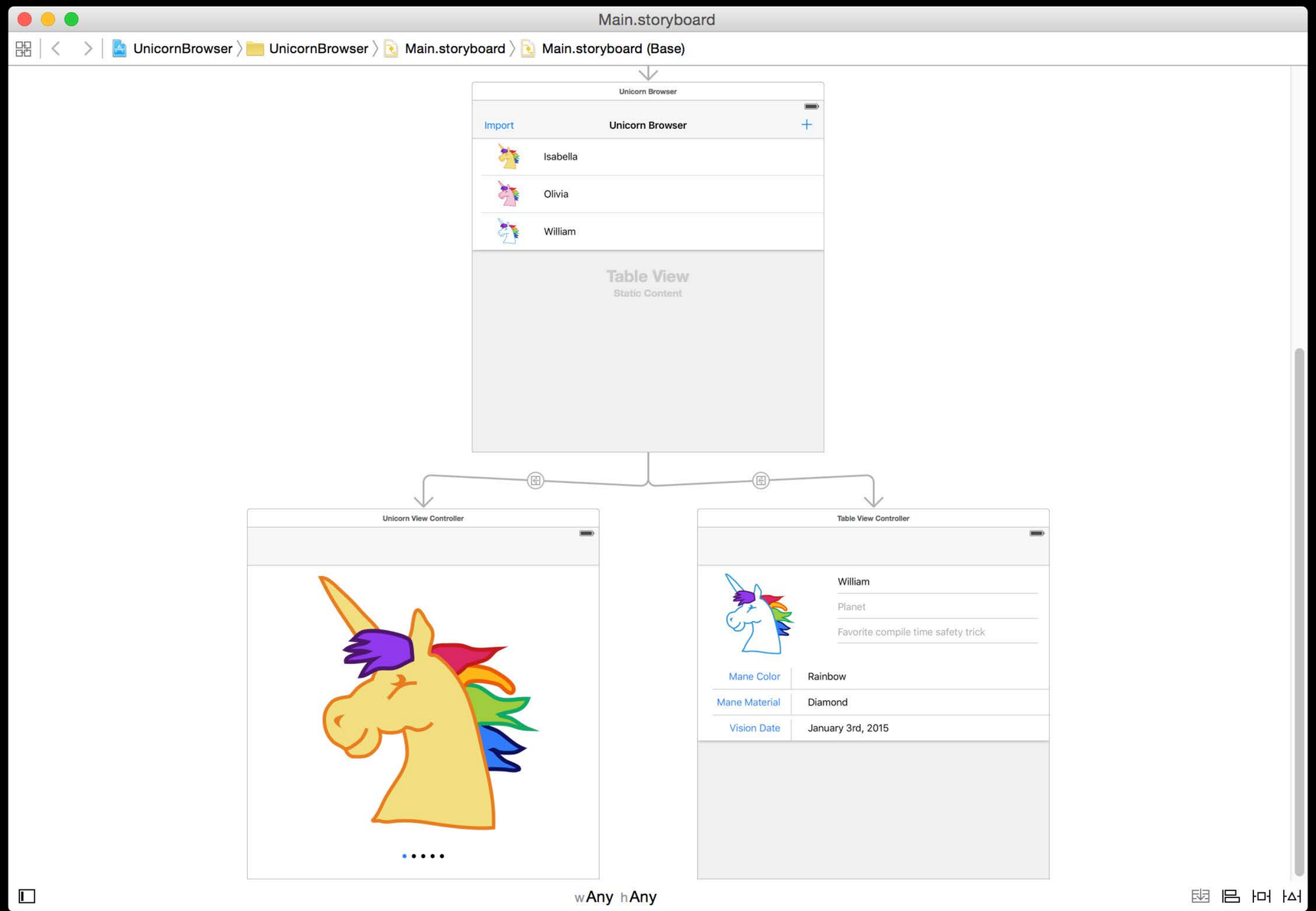
Enums

Think about how you can use enums for compile time safety

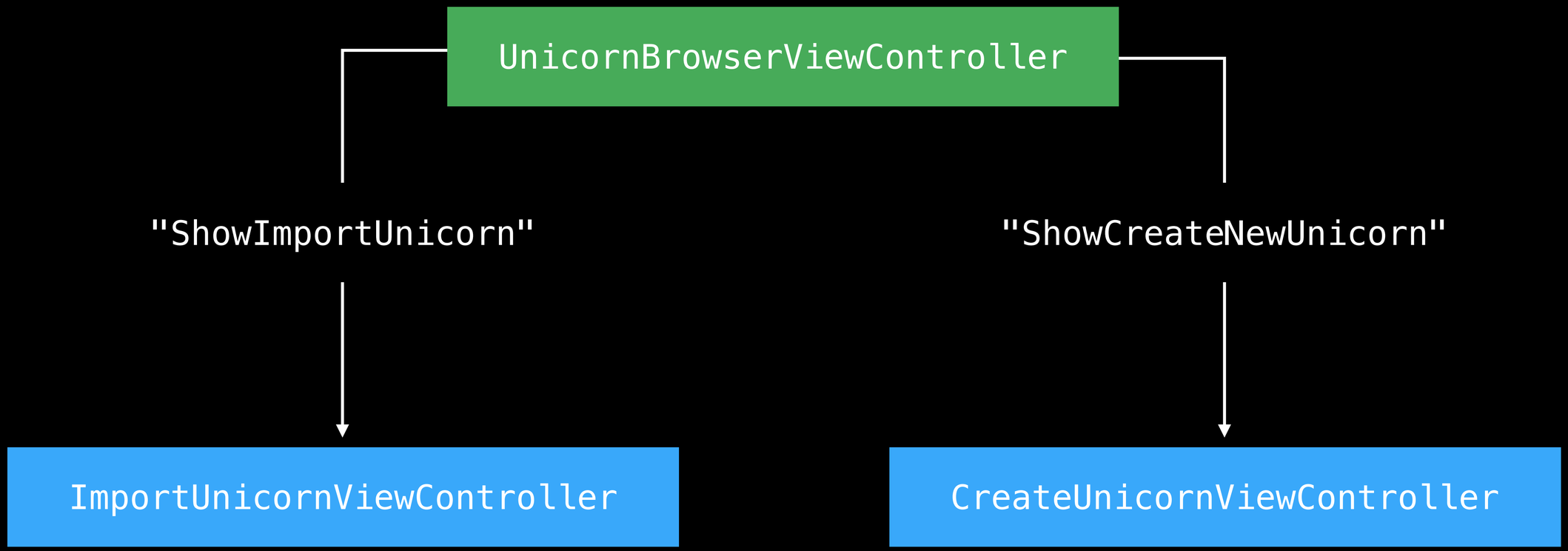
Enums can be backed by more than just String (Int, Selector, Character, Double, etc.)

Segue Identifiers

UnicornBrowser Storyboard



UnicornBrowser Storyboard



Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    switch segue.identifier {
        case "ShowImportUnicorn"?:    // Config...
        case "ShowCreateNewUnicorn"?: // Config...
    }
}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    switch segue.identifier {
        case "ShowImportUnicorn"?:    // Config...
        case "ShowCreateNewUnicorn"?: // Config...
    }
}
```

error: switch must be exhaustive, consider adding a default case

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    switch segue.identifier {
        case "ShowImportUnicorn"?:    // Config...
        case "ShowCreateNewUnicorn"?: // Config...
        default: fatalError("Invalid segue identifier \(segue.identifier).")
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController {
    enum SegueIdentifier: String {
        case ShowImportUnicorn    = "ShowImportUnicorn"
        case ShowCreateNewUnicorn = "ShowCreateNewUnicorn"
    }
}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,

}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,
                segueIdentifier = SegueIdentifier(rawValue: identifier)

}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,
                segueIdentifier = SegueIdentifier(rawValue: identifier)
        else { fatalError("Invalid segue identifier \(segue.identifier).") }



}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,
              segueIdentifier = SegueIdentifier(rawValue: identifier)
        else { fatalError("Invalid segue identifier \(segue.identifier).") }

        switch segueIdentifier {


        }
    }
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,
              segueIdentifier = SegueIdentifier(rawValue: identifier)
        else { fatalError("Invalid segue identifier \(segue.identifier).") }

    switch segueIdentifier {
        case .ShowImportUnicorn:    // Config...
        case .ShowCreateNewUnicorn: // Config...
    }
}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
enum SegueIdentifier: String {
    ...
    case ShowEditUnicorn = "ShowEditUnicorn"
}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,
              segueIdentifier = SegueIdentifier(rawValue: identifier)
        else { fatalError("Invalid segue identifier \(segue.identifier).") }

    switch segueIdentifier {
        case .ShowImportUnicorn:    // Config...
        case .ShowCreateNewUnicorn: // Config...
    }
}
```

Segue Identifiers

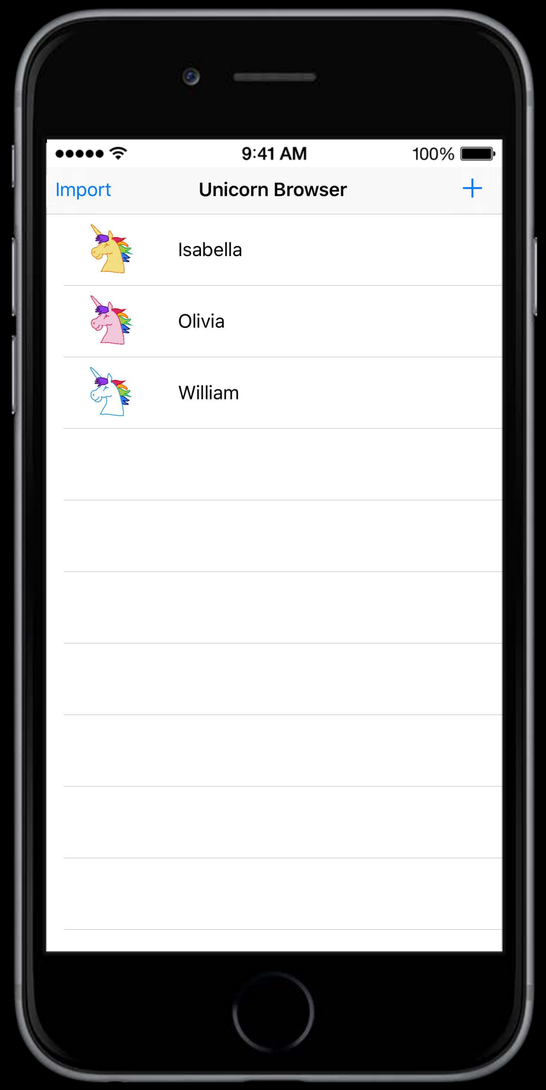
```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    guard let identifier = segue.identifier,
              segueIdentifier = SegueIdentifier(rawValue: identifier)
        else { fatalError("Invalid segue identifier \(segue.identifier).") }

    switch segueIdentifier {
        case .ShowImportUnicorn:    // Config...
        case .ShowCreateNewUnicorn: // Config...
    }
}
```

error: switch must be exhaustive, consider adding a default case

Segue Identifiers

Segues usually invoked by UIKit

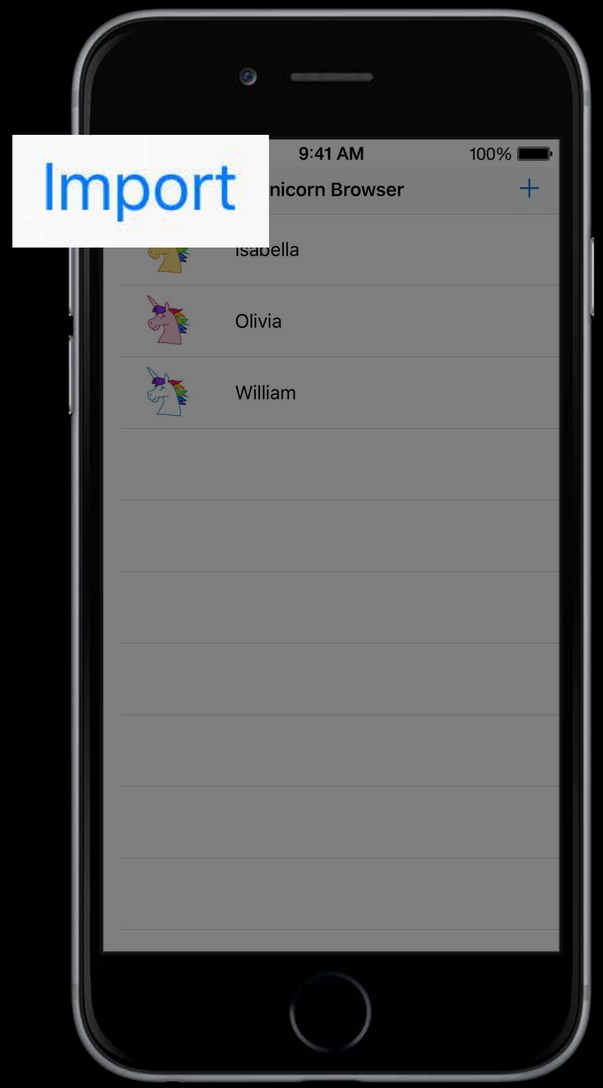
May need to invoke with API



Segue Identifiers

Segues usually invoked by UIKit

May need to invoke with API



Segue Identifiers

```
class UnicornBrowserViewController: UIViewController {
    func handleAction(sender: AnyObject?) {
        performSegueWithIdentifier("ShowImportUnicorn", sender: sender)
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController {
    func handleAction(sender: AnyObject?) {
        performSegueWithIdentifier(.ShowImportUnicorn, sender: sender)
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController {
    func performSegueWithIdentifier(segueIdentifier: SegueIdentifier,
                                    sender: AnyObject?) {
    }
}
```

Segue Identifiers

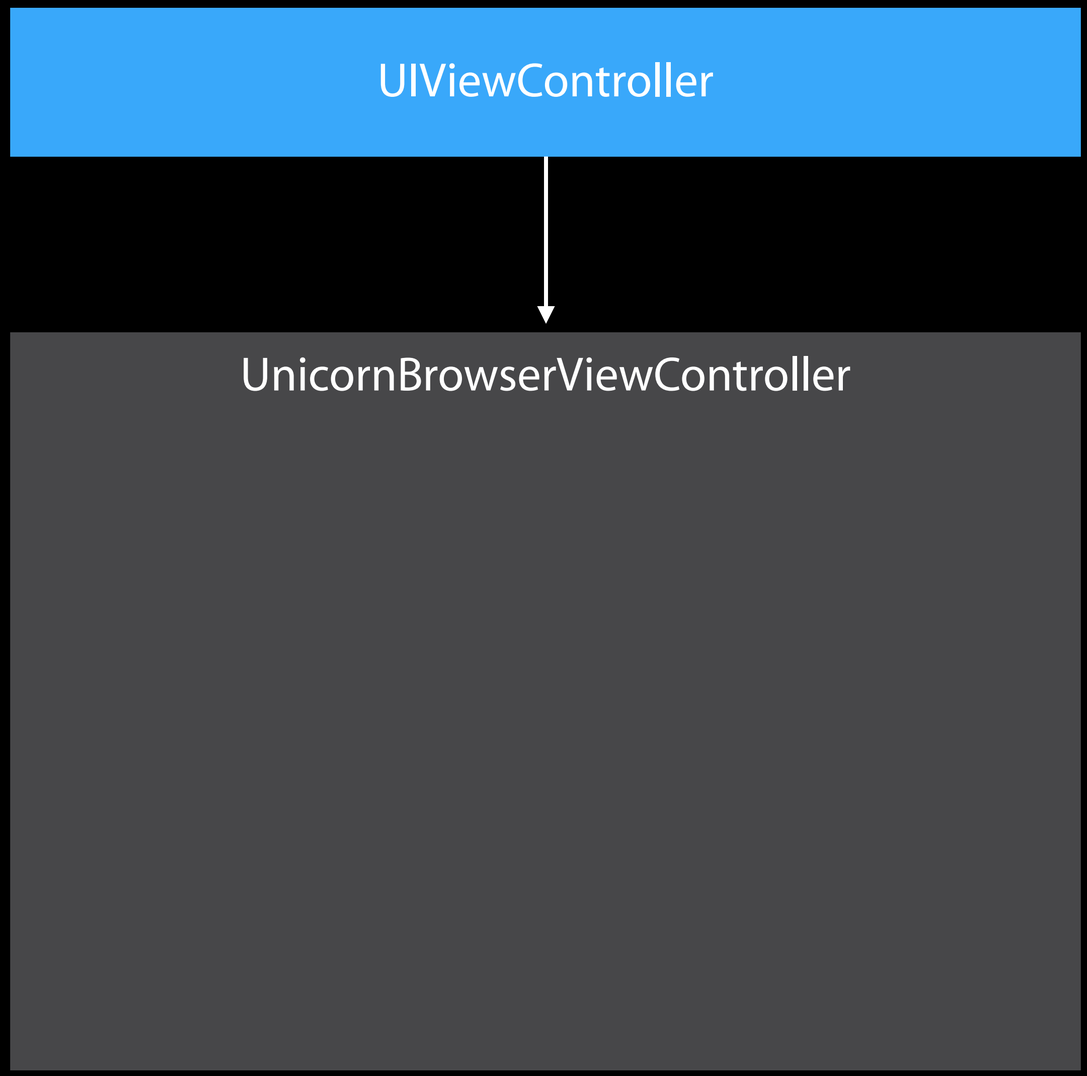
```
class UnicornBrowserViewController: UIViewController {
    func performSegueWithIdentifier(segueIdentifier: SegueIdentifier,
                                    sender: AnyObject?) {
        performSegueWithIdentifier(segueIdentifier.rawValue, sender: sender)
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController {
    func handleAction(sender: AnyObject?) {
        performSegueWithIdentifier(.ShowImportUnicorn, sender: sender)
    }
}
```

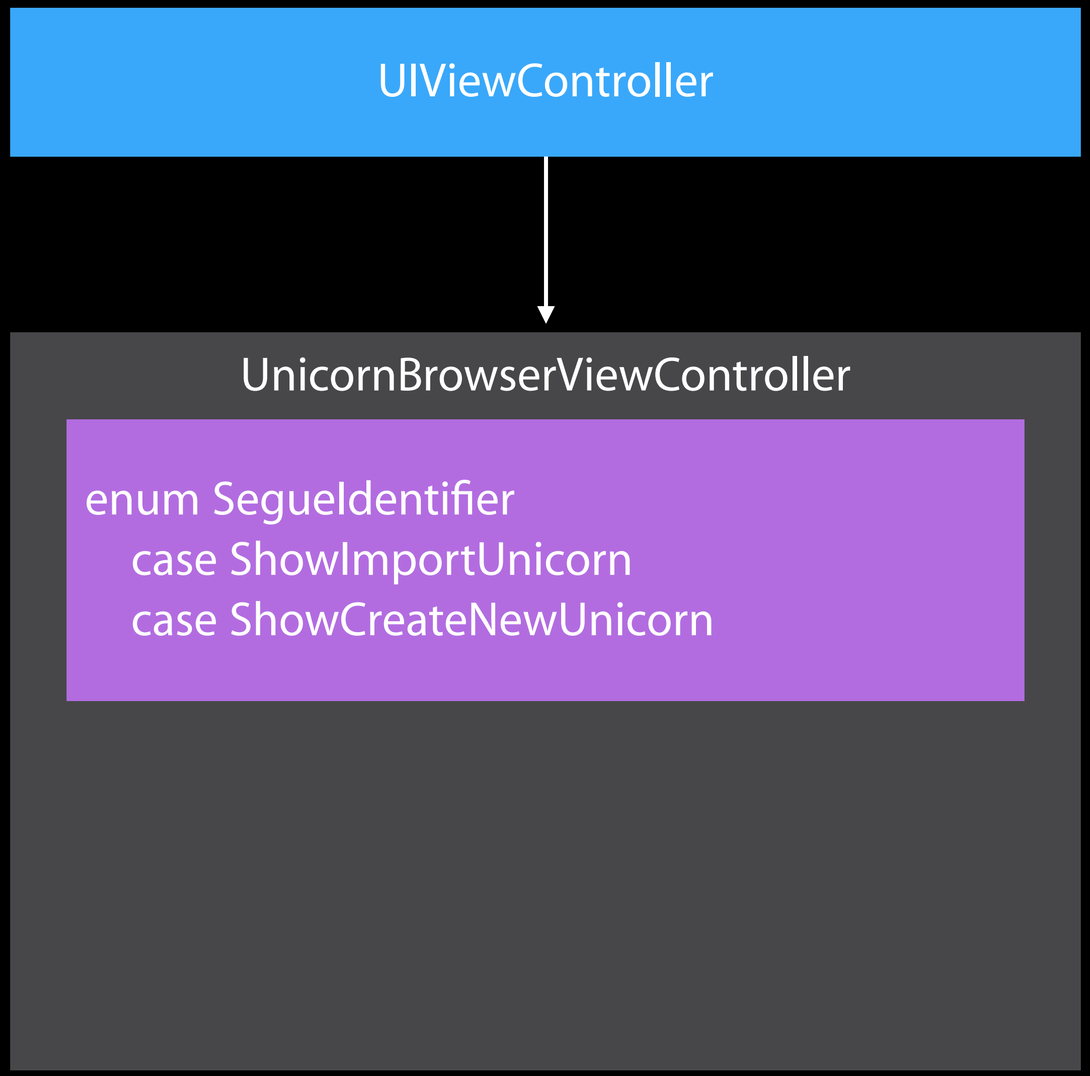
UnicornBrowserViewController Structure

UnicornBrowserViewController Structure

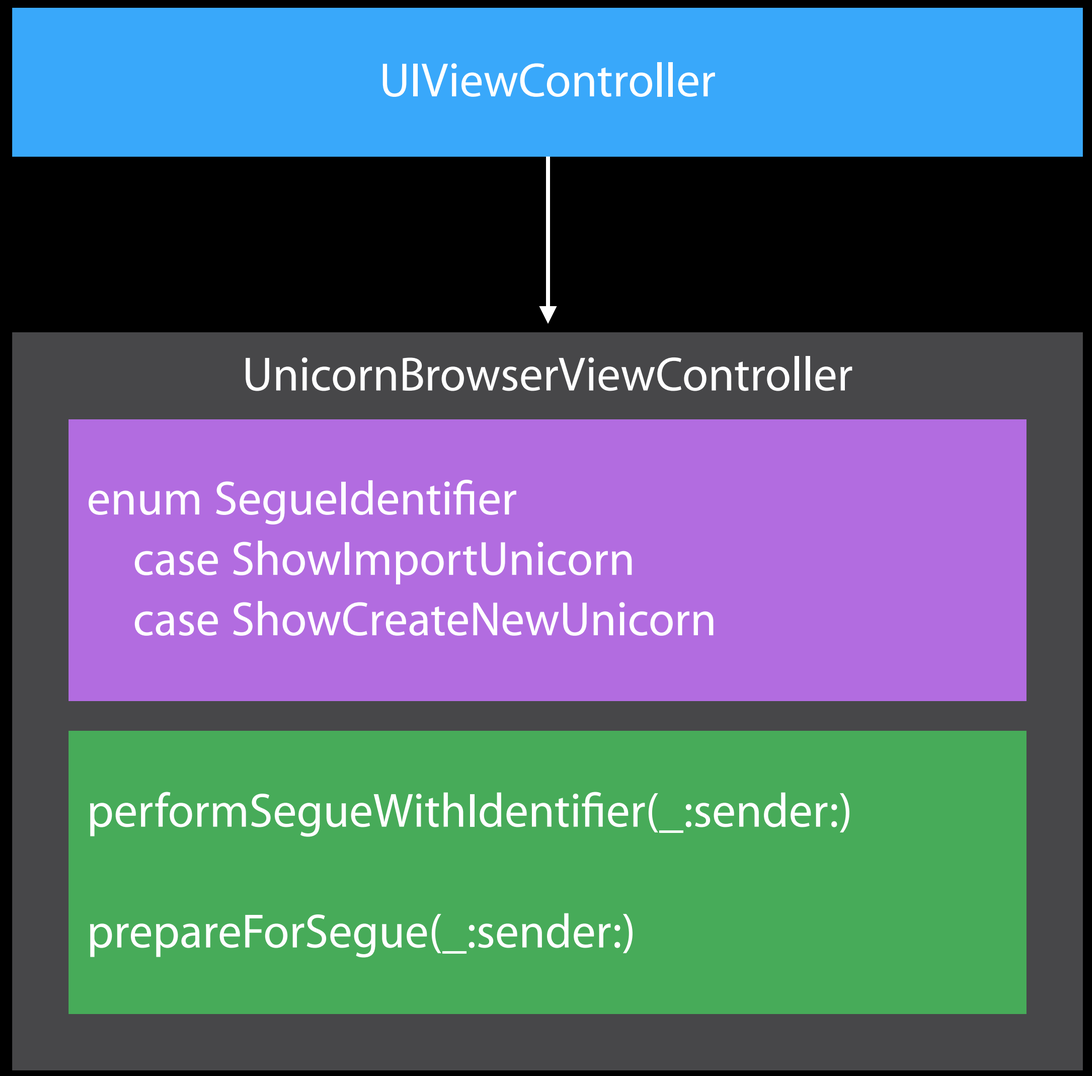


UnicornBrowserViewController Structure

Mapping



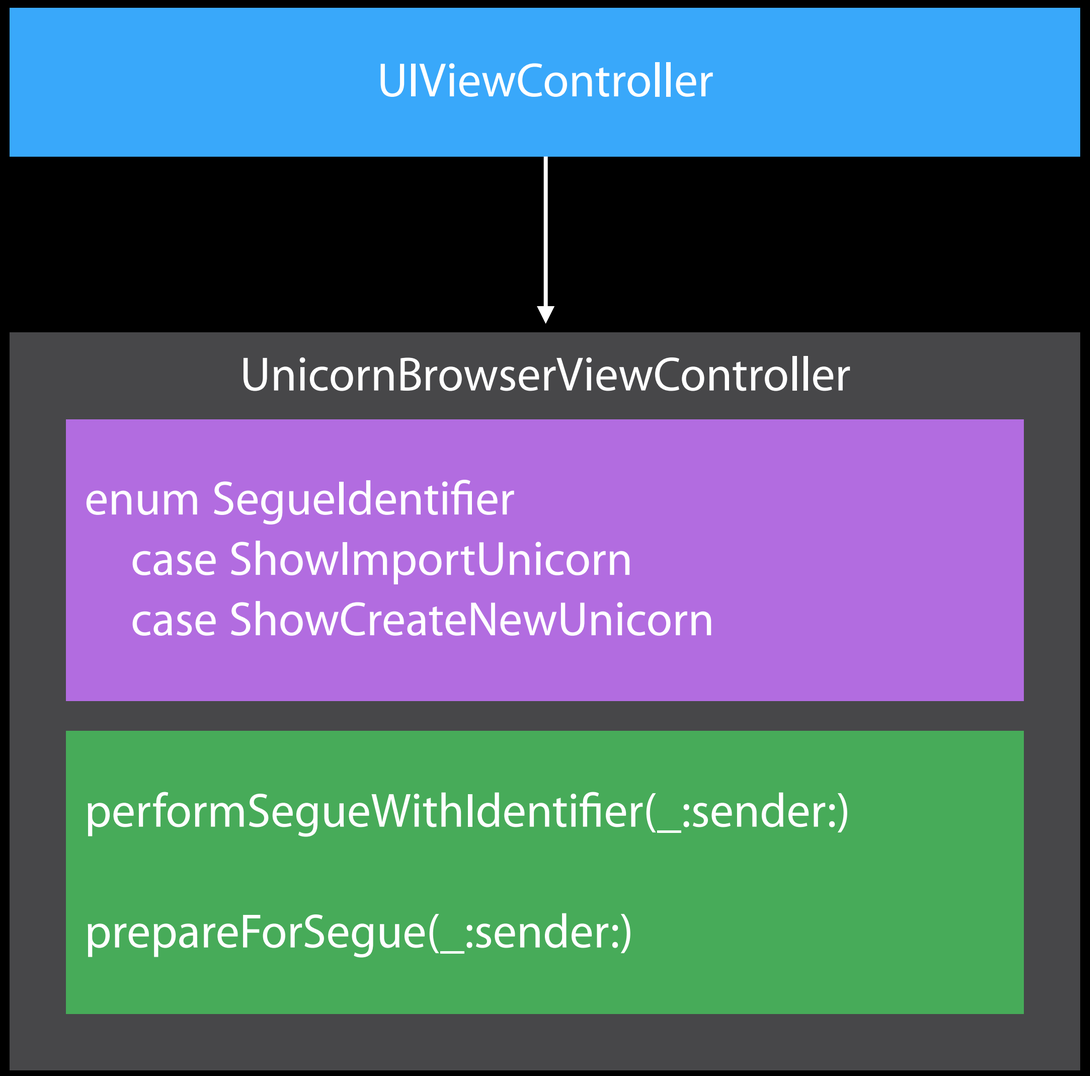
UnicornBrowserViewController Structure



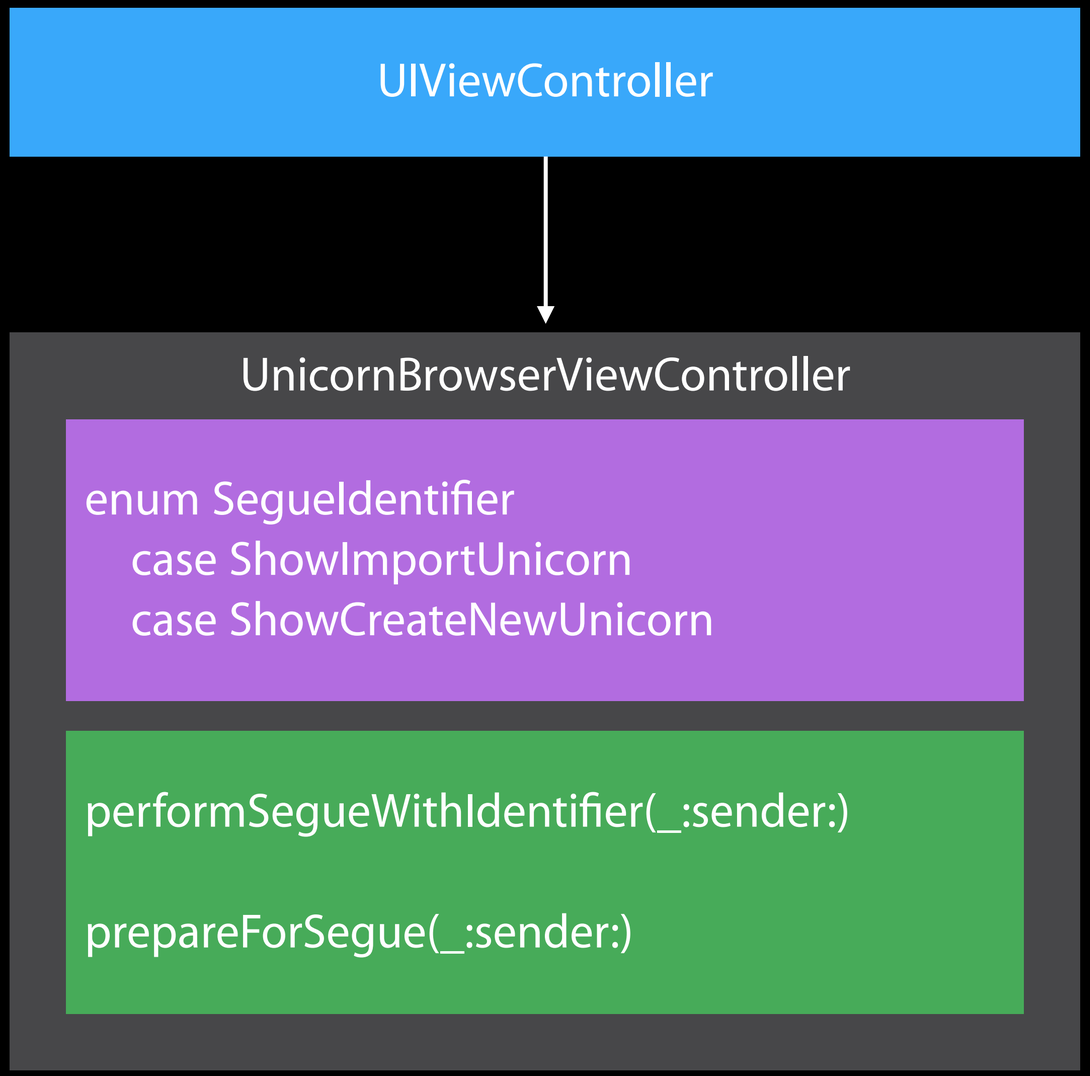
Mapping

Implementation

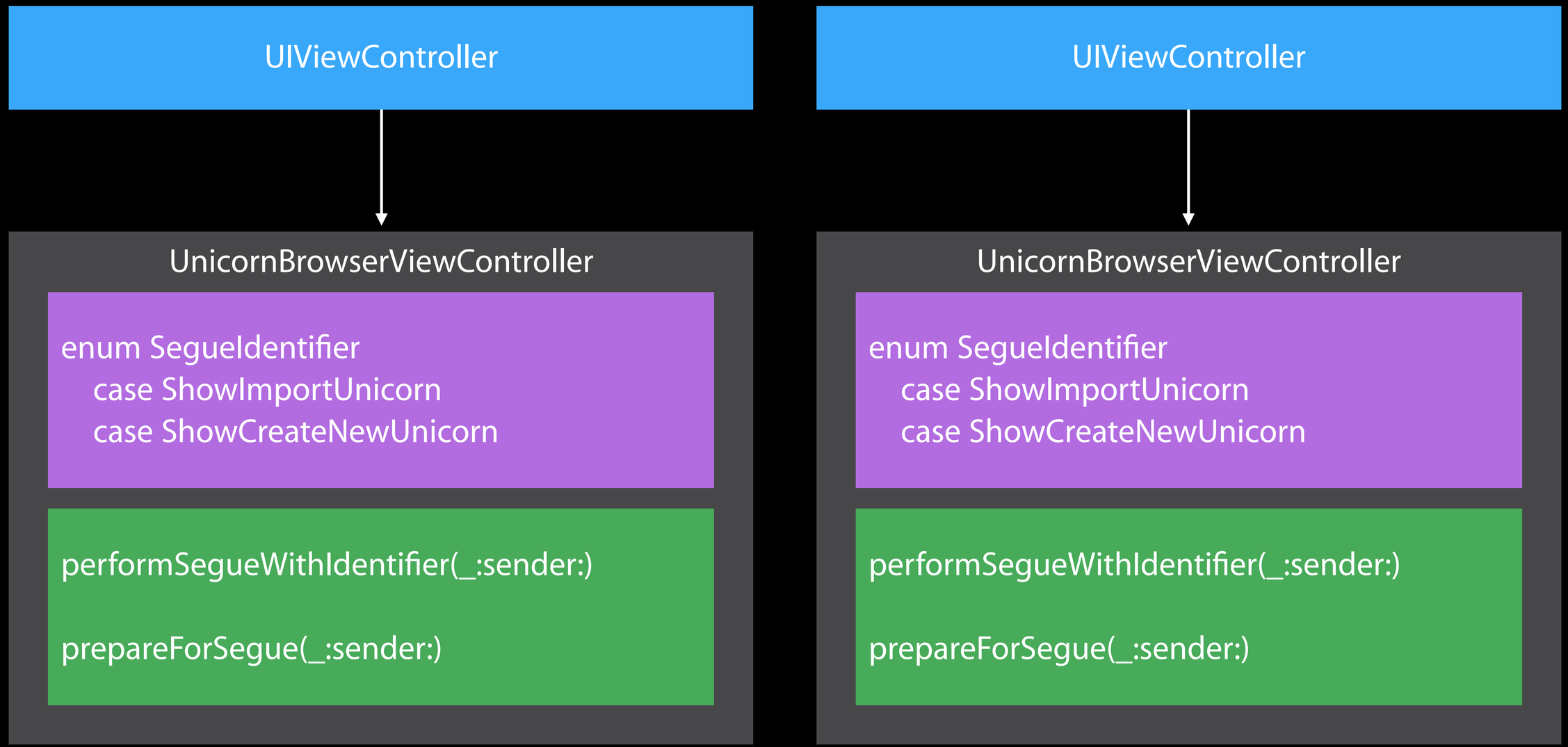
UnicornBrowserViewController Structure



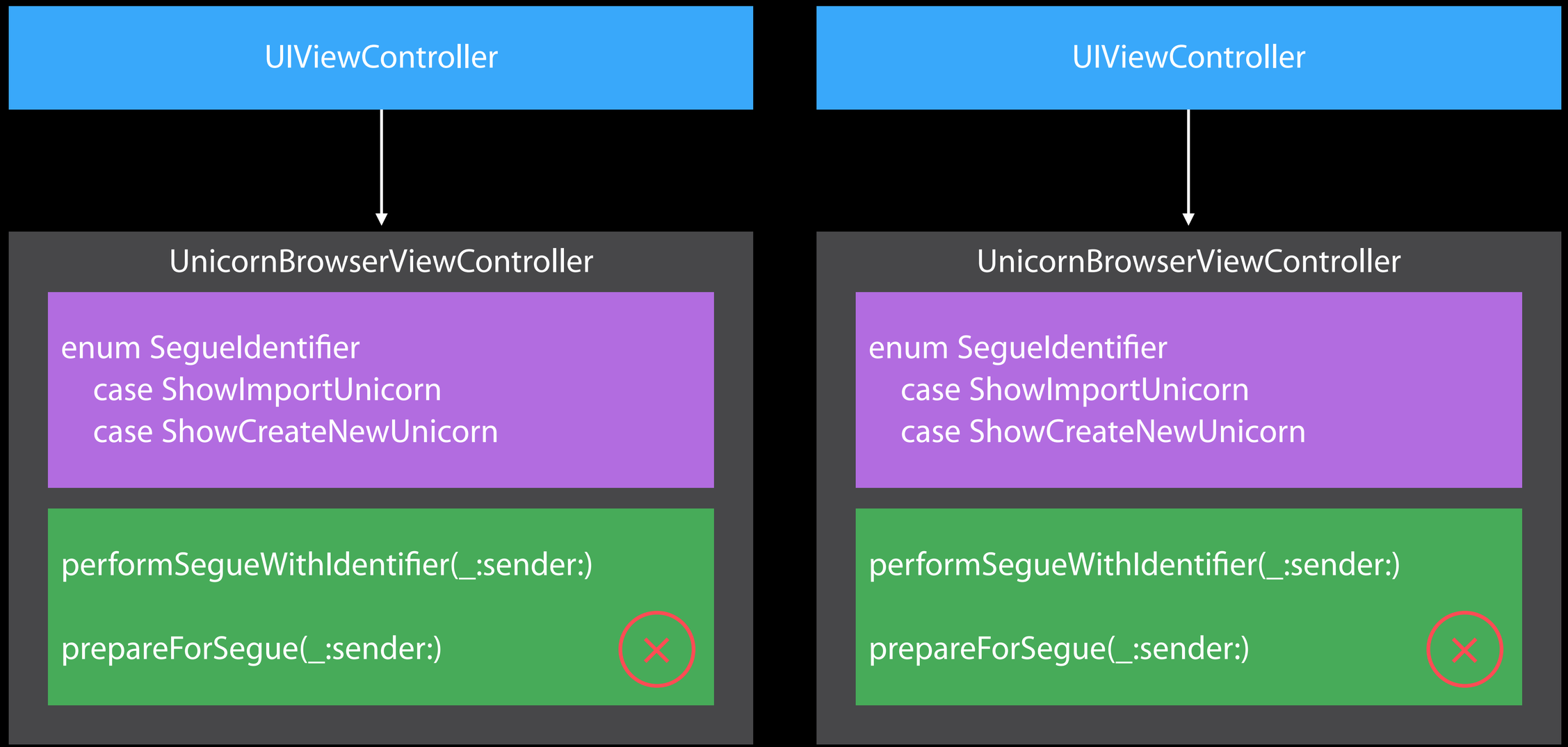
UnicornBrowserViewController Structure



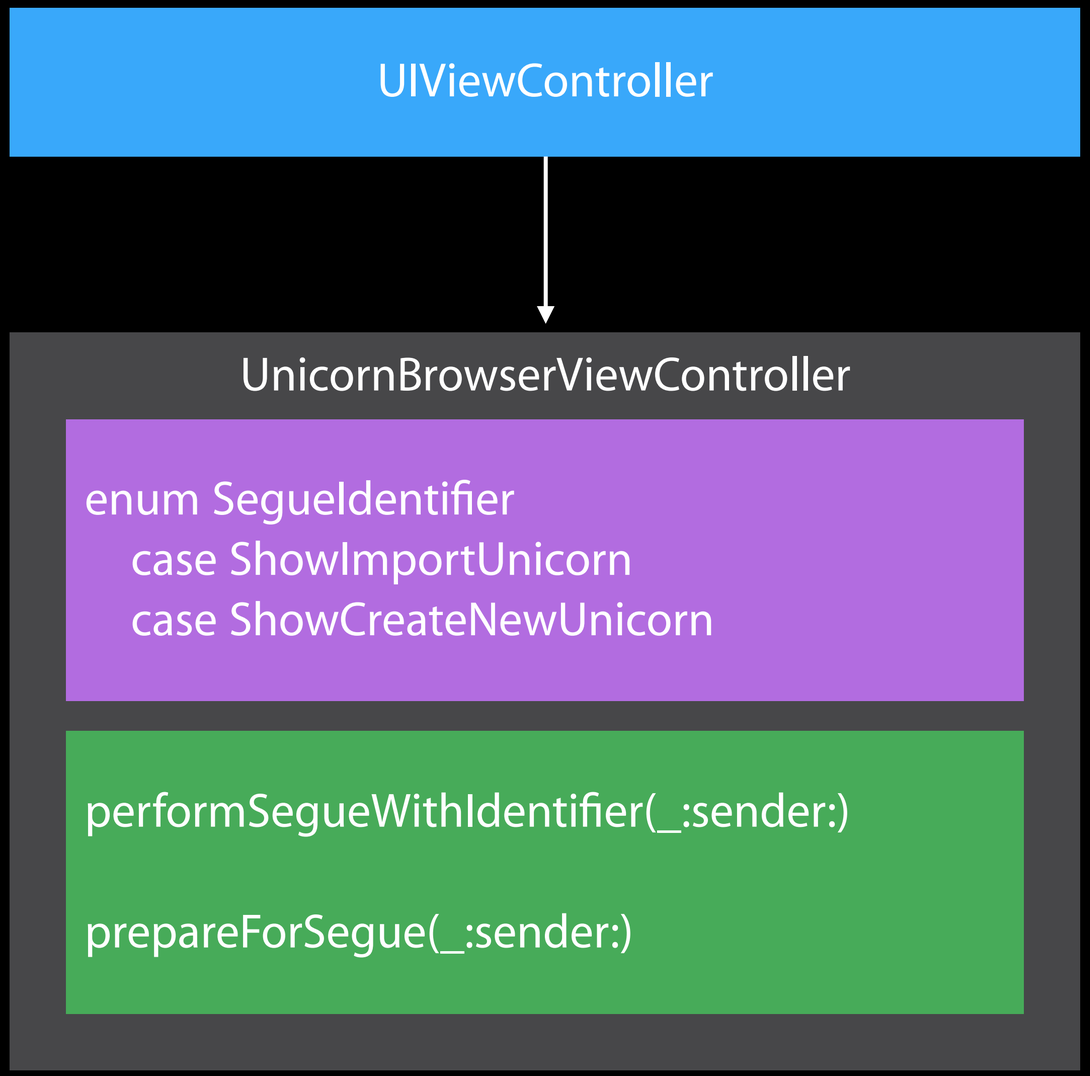
UnicornBrowserViewController Structure



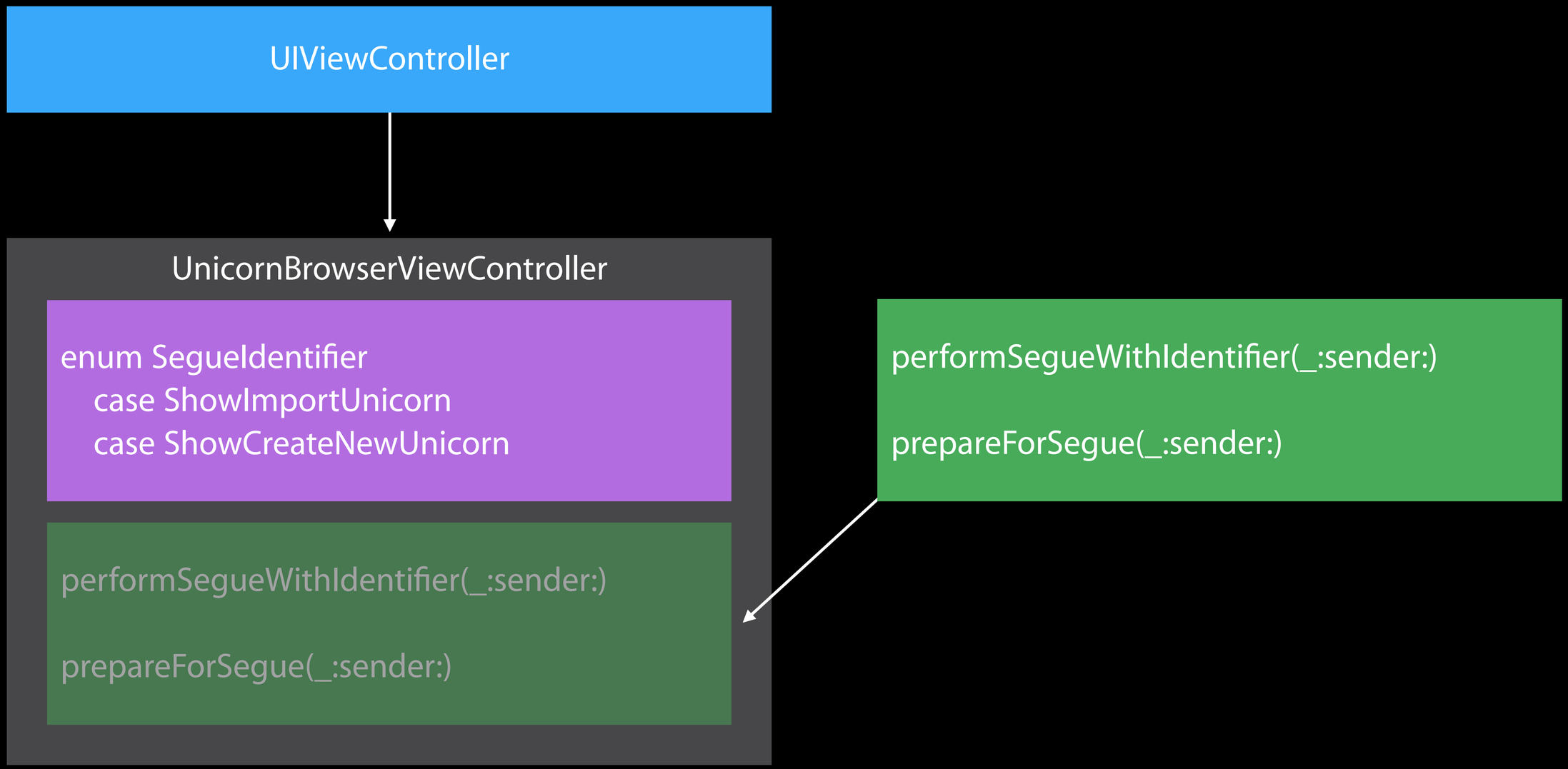
UnicornBrowserViewController Structure



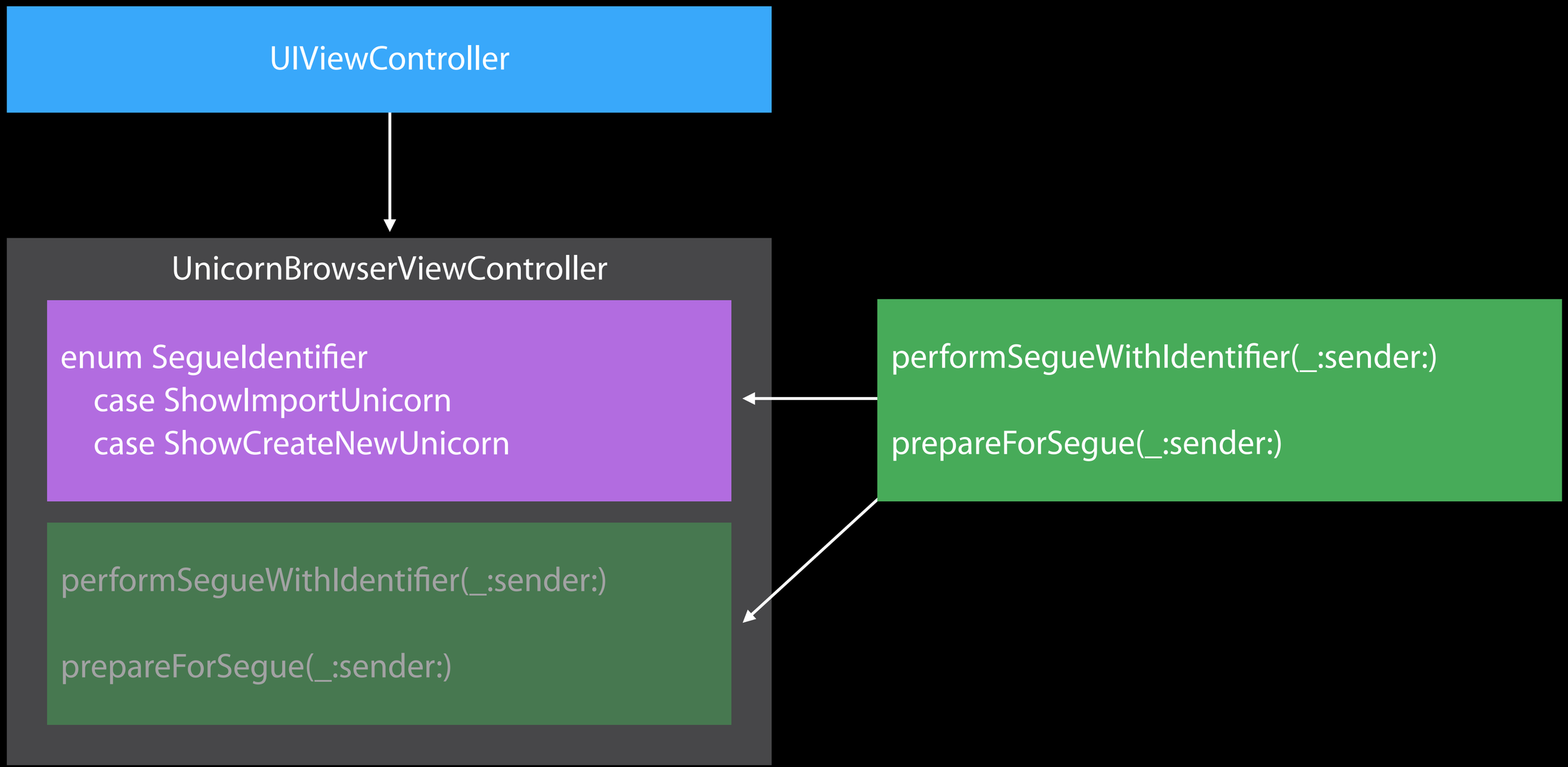
UnicornBrowserViewController Structure



UnicornBrowserViewController Structure



UnicornBrowserViewController Structure



UnicornBrowserViewController Structure

performSegueWithIdentifier(_:sender:)

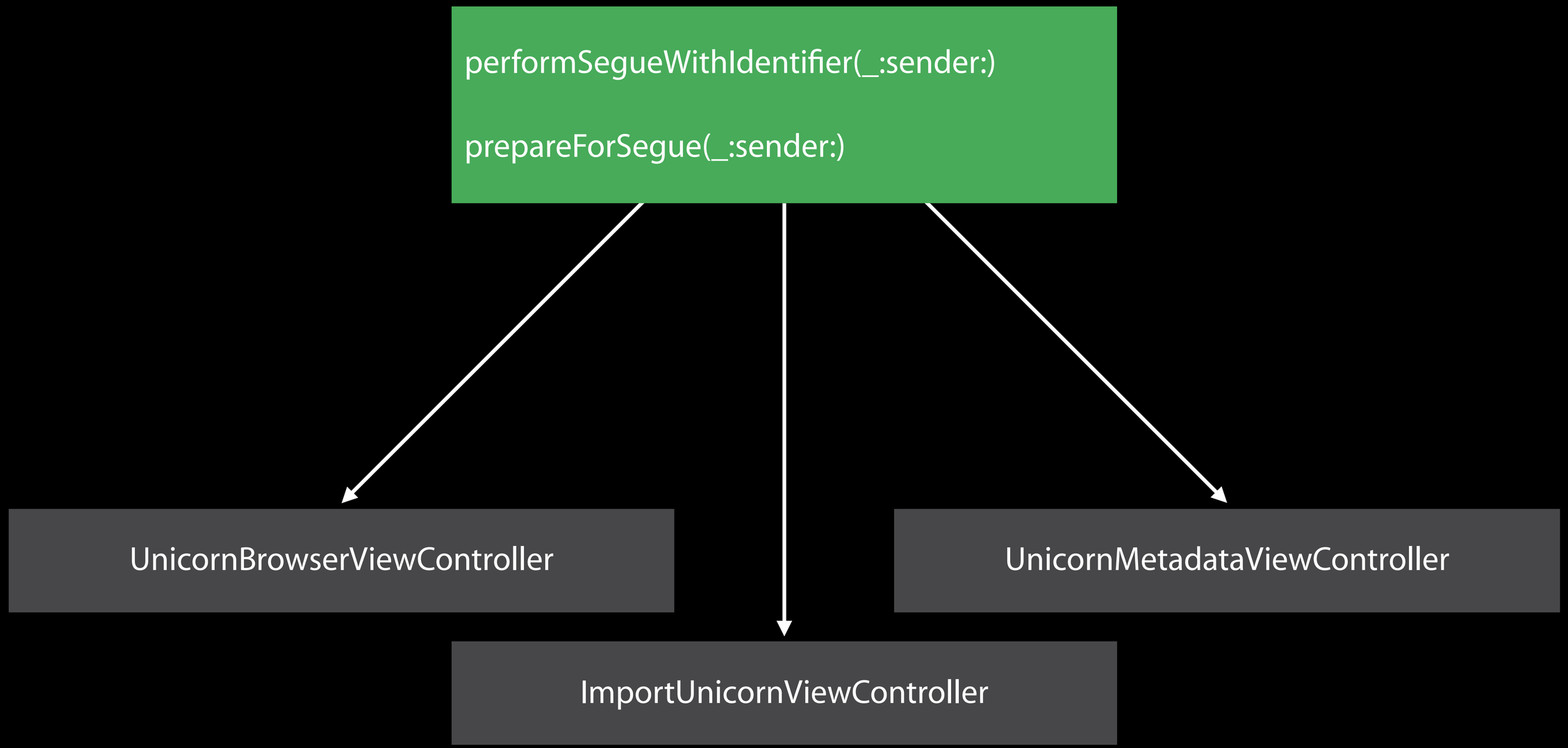
prepareForSegue(_:sender:)

UnicornBrowserViewController Structure

performSegueWithIdentifier(_:sender:)

prepareForSegue(_:sender:)

UnicornBrowserViewController Structure



Segue Identifiers

```
protocol SegueHandlerType {

}
```

Segue Identifiers

```
protocol SegueHandlerType {
    typealias SegueIdentifier
}
```

Segue Identifiers

```
protocol SegueHandlerType {
    typealias SegueIdentifier: RawRepresentable
}
```

Segue Identifiers

```
extension SegueHandlerType
```

Segue Identifiers

```
extension SegueHandlerType where
```

Segue Identifiers

```
extension SegueHandlerType where
    Self: UIViewController,
```

Segue Identifiers

```
extension SegueHandlerType where
    Self: UIViewController,
    SegueIdentifier.RawValue == String {

}
```

Segue Identifiers

```
extension SegueHandlerType where
    Self: UIViewController,
    SegueIdentifier.RawValue == String {
    func performSegueWithIdentifier(segueIdentifier: SegueIdentifier,
                                    sender: AnyObject?) {
        performSegueWithIdentifier(segueIdentifier.rawValue, sender: sender)
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController {
    enum SegueIdentifier: String {
        ...
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController, SegueHandlerType {
    enum SegueIdentifier: String {
        ...
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController, SegueHandlerType {
    enum SegueIdentifier: String {
        ...
    }
}
```

Segue Identifiers

```
class UnicornBrowserViewController: UIViewController, SegueHandlerType {
    ...
    func handleAction(sender: AnyObject?) {
        performSegueWithIdentifier(.ShowImportUnicorn, sender: sender)
    }
}
```

Segue Identifiers

```
// SegueHandlerType.swift
func segueIdentifierForSegue(segue: UIStoryboardSegue) -> SegueIdentifier {

}
```

Segue Identifiers

```
// SegueHandlerType.swift
func segueIdentifierForSegue(segue: UIStoryboardSegue) -> SegueIdentifier {
    guard let identifier = segue.identifier,
                segueIdentifier = SegueIdentifier(rawValue: identifier)
        else { fatalError("Invalid segue identifier \(segue.identifier).") }


}
```

Segue Identifiers

```
// SegueHandlerType.swift
func segueIdentifierForSegue(segue: UIStoryboardSegue) -> SegueIdentifier {
    guard let identifier = segue.identifier,
                segueIdentifier = SegueIdentifier(rawValue: identifier)
    else { fatalError("Invalid segue identifier \(segue.identifier).") }

    return segueIdentifier
}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {

}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    switch segueIdentifierForSegue(segue) {

    }
}
```

Segue Identifiers

```
// UnicornBrowserViewController.swift
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    switch segueIdentifierForSegue(segue) {
        case .ShowImportUnicorn:    // Config...
        case .ShowCreateNewUnicorn: // Config...
    }
}
```

SegueHandlerType Protocol Benefits

Compiler errors when adding new segues if the new case isn't handled

SegueHandlerType Protocol Benefits

Compiler errors when adding new segues if the new case isn't handled

Reusable

SegueHandlerType Protocol Benefits

Compiler errors when adding new segues if the new case isn't handled

Reusable

Convenient syntax

Protocols

Tighten app constraints using protocols with associated types

Protocols

Tighten app constraints using protocols with associated types

Share implementation through a constrained protocol extension

Summary

The compiler is here to help

Summary

The compiler is here to help

Safely take advantage of new APIs

Summary

The compiler is here to help

Safely take advantage of new APIs

Leverage strong typing to enforce application behavior

Related Session

Protocol-Oriented Programming in Swift	Mission	Wednesday 2:30PM

Want to Have Lucid Dreams About Swift and Cocoa?

Lister

http://developer.apple.com/library/prerelease/ios/samplecode/Lister

DemoBots

http://developer.apple.com/library/prerelease/ios/samplecode/DemoBots

More Information

Swift Language Documentation
http://developer.apple.com/swift

Apple Developer Forums
http://developer.apple.com/forums

Stefan Lesser
Developer Tools Evangelist
slesser@apple.com

WWDC15