Introduction to Swift KeyPaths

お早う!

Benedikt Terhechte

Twitter: @terhechte

Swift Guides: appventure.me

Swift Podcast: contravariance.rocks

What are KeyPaths

Swift 4

Type-Safe shortcuts to **Read** / **Write** properties

Composable

Not related to Objective-C's KeyPaths

```
[object valueForKeyPath: @“user.address.city.zip"]

object.valueForKey(#keyPath(Object.firstName))
```

What do we want to achieve?

```swift
struct ProfileSettings {
    var displayName: String
    var shareUpdates: Bool
    var score: Float
}

struct PrivacySettings {
    var passcode: Bool
    var addByID: Bool
    var blackList: [String]
}
```

```
protocol SettingsEntry {
  ???
}
```

Abstract Over Types

```
struct ProfileSettings {
    var displayName: String
    var shareUpdates: Bool
    var score: Float
}
struct PrivacySettings {
    var passcode: Bool
    var addByID: Bool
    var blackList: [String]
}
```

String, Bool, Float

Bool, Bool, [String]

KeyPaths

Protocols make it difficult to abstract over very different types

KeyPaths allow to do this

Goal

Develop generic app settings

Settings of any shape / type

To achieve that, we will learn about KeyPaths

Agenda

Intro

KeyPath Theory

Practical Example

Tips And Tricks

KeyPath Libraries

Intro

Example:

```
struct User {
  var username: String
}


var player = User(username: “Mario”]


player[keyPath: \User.username] = “Link”
```

Example:

```
player[keyPath: \User.username] =
"Link"
```

```
let nameKeyPath = \User.username

player[keyPath: nameKeyPath] = “Luigi”
```

```
let nameKeyPath = \User.username
```

Abstract the access to the property "username" into a variable that can be moved and stored

\User.username

Root: User

Value: String

KeyPath<Root, Value>

KeyPath<User, String>

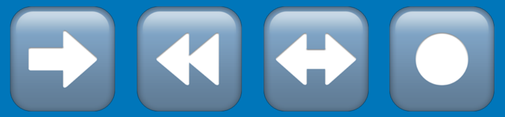
Nesting:

```
struct Address {
  let street: String
}

struct User {
  let address: Address
}

let x = \User.address.street.count
```

KeyPath<User, Int>

Theory

Types of Keypaths



KeyPath<Root, Value>

Read-Only KeyPath with Root and Value

`let` properties

KeyPath<Root, Value>

```
struct User {
  let username: String
}

let kp: KeyPath<User, String> = \User.username

let player = User(username: “Mario”)

print(player[keyPath: kp])

// Error. Read Only KeyPath
player[keyPath: kp] = “Luigi"
```

WritableKeyPath<Root, Value>

Read / Write KeyPath

`var` properties

WritableKeyPath<Type, Value>

```
struct User { var username: String }

var player = User(username: “Mario”)

player[keyPath: \User.username] = “Luigi”
```

ReferenceWritableKeyPath<Root, Value>

Read / Write `KeyPath` for `Class` types

Useful for mutating properties of `let` roots

PartialKeyPath<Root>

KeyPaths of different Values with the same Root

Also Read-Only

PartialKeyPath<Root>

```
// String
let a: PartialKeyPath<User> = \User.name

// Int
let b: PartialKeyPath<User> = \User.age

// Float
let c: PartialKeyPath<User> = \User.quote
```

```swift
func acceptKeyPath (_ keyPath: PartialKeyPath<User>) {
  …
}


acceptKeyPath(\User.age)
acceptKeyPath(\User.username)
```

AnyKeyPath

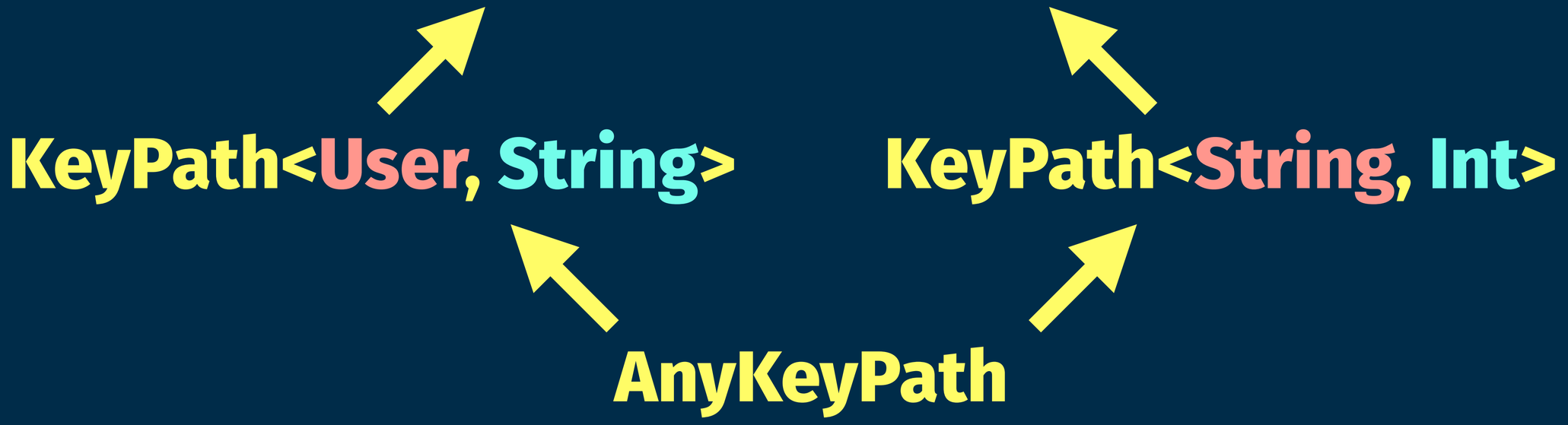
No Root, no Value

Type-Erased KeyPath.

Very useful to keep different types together

AnyKeyPath

```
let keyPaths: [AnyKeyPath]
      = [\User.username, \String.count]
```



You can cast types back

```
# AnyKeyPath as? WritableKeyPath<User, String>

# PartialKeyPath<User> as? KeyPath<User, Bool>
```

KeyPath Composition

```swift
struct User {
  let address: Address
}

struct Address {
  let street: String
}
```

KeyPath Composition

```
// User -> address
let addressKeyPath = \User.address

// Address -> street
let streetKeyPath = \Address.street

// User -> address -> street
let userStreetKeyPath = addressKeyPath
                .appending(path: streetKeyPath)
```

KeyPath Composition

```
let userStreetKeyPath = addressKeyPath
                        .appending(path: streetKeyPath)
```

User → address

+

Address → street

=

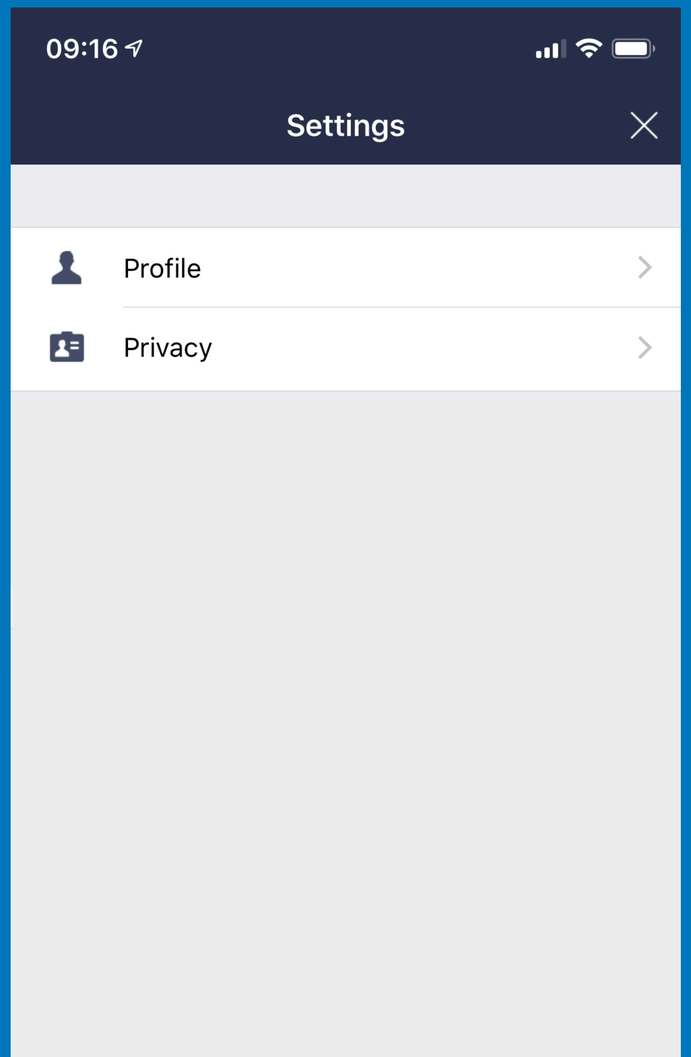
User → street

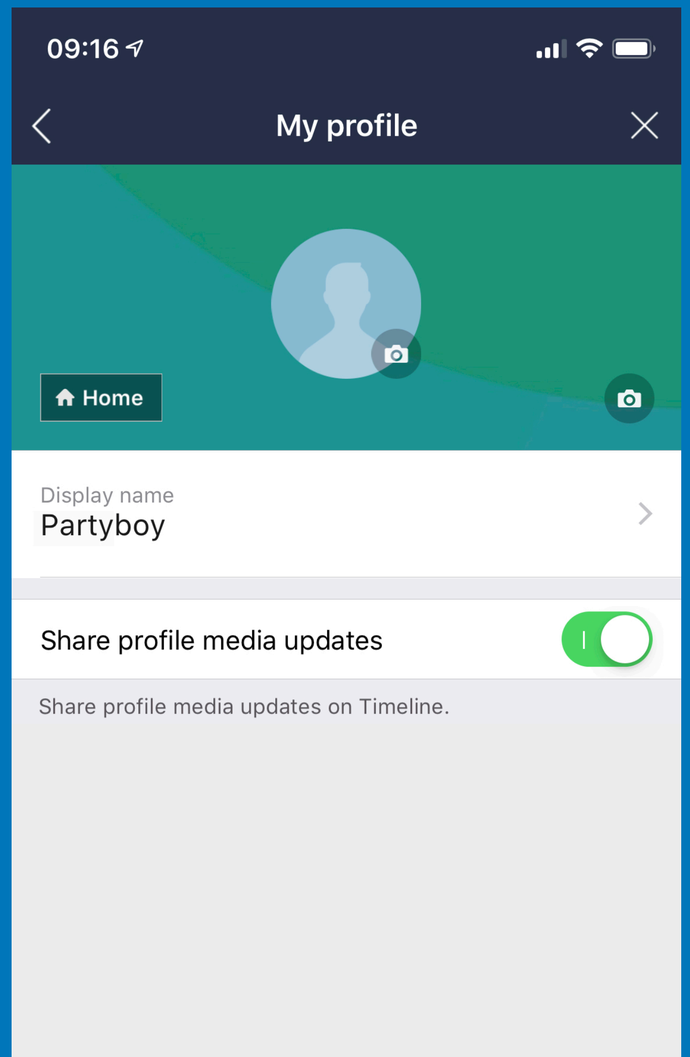
Theory

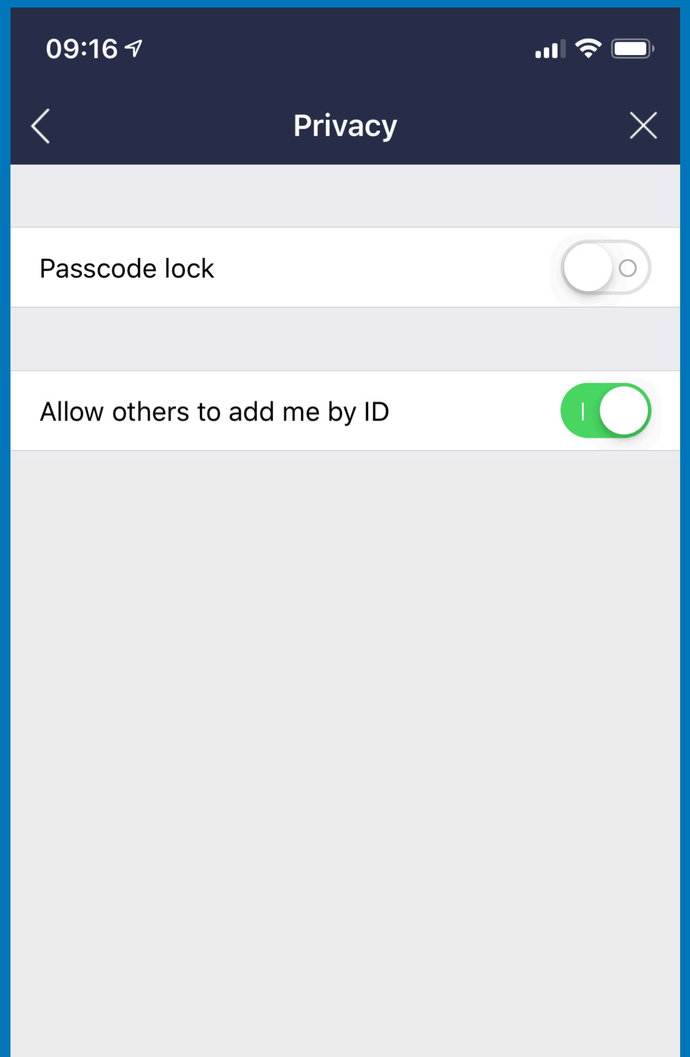
🔧

Practical Example

Generic way of handling App Settings


09:16
Settings
Profile
Privacy


09:16
My profile
Home
Display name
Partyboy
Share profile media updates
Share profile media updates on Timeline.

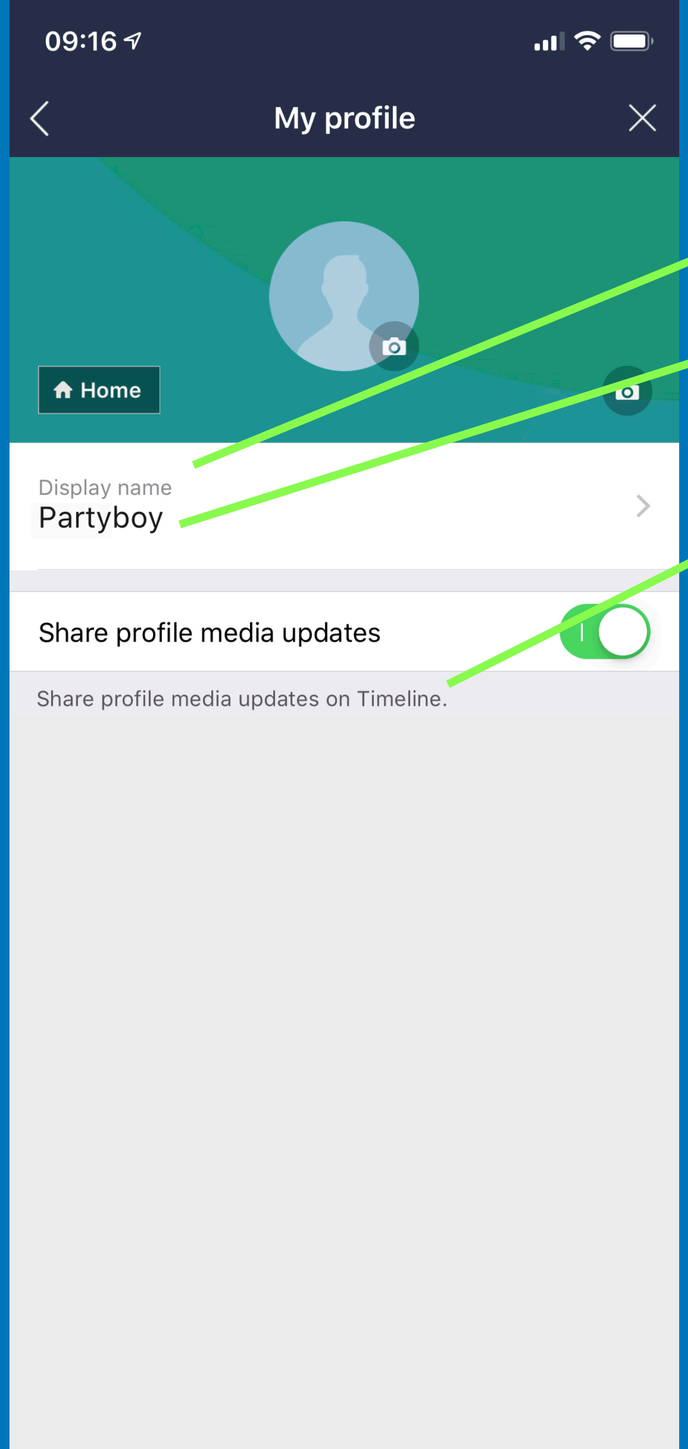

09:16
Privacy
Passcode lock
Allow others to add me by ID

```
struct Settings {
}
```

```swift
struct Settings {
    var profileSettings: ProfileSettings
    var privacySettings: PrivacySettings
}
```

```swift
struct Settings {
    var profileSettings: ProfileSettings
    var privacySettings: PrivacySettings
}
struct ProfileSettings {
    var displayName: String
    var shareUpdates: Bool
}
```

```swift
struct Settings {
    var profileSettings: ProfileSettings
    var privacySettings: PrivacySettings
}
struct ProfileSettings {
    var displayName: String
    var shareUpdates: Bool
}
struct PrivacySettings {
    var passcode: Bool
    var addByID: Bool
}
```



Title

Value / Type

Subtitle

Settings Entry Struct

```
struct SettingsEntry {
    let keyPath: AnyKeyPath
    let title: String
    let subtitle: String
    let help: String
    ...
}
```

Settings Entry Struct

```
struct SettingsEntry {
    let keyPath: AnyKeyPath <—\PrivacySettings.passcode
    let title: String
    let subtitle: String
    let help: String
    ...
}
```

Settings Entry Struct

```
struct SettingsEntry {
    let keyPath: AnyKeyPath
    let title: String <- "Lock with Passcode"
    let subtitle: String
    let help: String
    ...
}
```

Simpler Demo

```
struct SettingsEntry {
    let keyPath: AnyKeyPath
    let title: String
}
```

Allow types to return settings

```
protocol SettingsProvider {
    var settings: [SettingsEntry] { get }
}
```

```swift
struct Settings {
    var profileSettings: ProfileSettings
    var privacySettings: PrivacySettings
}
```

```
extension Settings: SettingsProvider {
 var settings: [SettingsEntry] {
  return […]
 }
}
```

```
extension Settings: SettingsProvider {
 var settings: [SettingsEntry] {
  return [
      SettingsEntry(
          keyPath: \Settings.profileSettings,
            title: "Profile"),

      SettingsEntry(
          keyPath: \Settings.privacySettings,
            title: "Privacy")
  ]
 }
}
```

```
extension ProfileSettings: SettingsProvider {
 var settings: [SettingsEntry] {
  return […]
 }
}
```

```
extension ProfileSettings: SettingsProvider {
 var settings: [SettingsEntry] {
  return [
    SettingsEntry(
       keyPath: \ProfileSettings.displayName,
         title: "Display Name"),

    SettingsEntry(
       keyPath: \ProfileSettings.shareUpdates,
         title: "Share Profile Media Updates")
  ]
 }
}
```

```
extension PrivacySettings: SettingsProvider {
 var settings: [SettingsEntry] {
  return [
      SettingsEntry(
          keyPath: \PrivacySettings.addByID,
            title: "Allow add me by ID"),

      SettingsEntry(
          keyPath: \PrivacySettings.passcode,
            title: "Passcode Lock")
  ]
 }
}
```

```
func editSettings(provider: inout SettingsProvider) {
}

var appSettings = Settings()
editSettings(provider: &appSettings)
```

```
func editSettings(provider: inout SettingsProvider) {


    for setting in provider.settings {
    }
}
```

```
func editSettings(provider: inout SettingsProvider) {

    for setting in provider.settings {
        let value = provider[keyPath: setting.keyPath]
        ...
    }
}
```

Nested Providers

```
Settings { <--- Here `[SettingsEntry]`
  profileSettings {
    displayName: String,
    shareUpdates: Bool
  },
  ...
}
```

Nested Providers

```
Settings {
  profileSettings { <--- Here `[SettingsEntry]`
    displayName: String,
    shareUpdates: Bool
  },
  ...
}
```

Nested Providers

```
Settings {
  profileSettings {
    displayName: String, <--- Here `String`
    shareUpdates: Bool
  },
  ...
}
```

```
func editSettings(provider: inout SettingsProvider) {


  for setting in provider.settings {
    let value = provider[keyPath: setting.keyPath]
    if let nested = value as? SettingsProvider {
        ...
    } else {
        ...
    }
  }
}
```

```
func editSettings(provider: inout SettingsProvider) {


  for setting in provider.settings {
    let value = provider[keyPath: setting.keyPath]
    if let nested = value as? SettingsProvider {
      for item in nested.settings {
```

```
func editSettings(provider: inout SettingsProvider) {


  for setting in provider.settings {
    let value = provider[keyPath: setting.keyPath]
    if let nested = value as? SettingsProvider {
      for item in nested.settings {

        if let joined =
          keyPath.appending(path: item.keyPath) {
```

```
func editSettings(provider: inout SettingsProvider) {


  for setting in provider.settings {
    let value = provider[keyPath: setting.keyPath]
    if let nested = value as? SettingsProvider {
      for item in nested.settings {

        if let joined =
          keyPath.appending(path: item.keyPath) {
```

\Settings.PrivacySettings

```
func editSettings(provider: inout SettingsProvider) {


  for setting in provider.settings {
    let value = provider[keyPath: setting.keyPath]
    if let nested = value as? SettingsProvider {
      for item in nested.settings {

        if let joined =
          keyPath.appending(path: item.keyPath) {
```

\PrivacySettings.passcode

```
func editSettings(provider: inout SettingsProvider) {


  for setting in provider.settings {
    let value = provider[keyPath: keyPath]
    if let nested = value as? SettingsProvider {
      for item in nested.settings {

        if let joined =
          keyPath.appending(path: item.keyPath) {
            updateSetting(keyPath: joined,
                          title: item.title)

    ...
}
```

Settings → privacySettings

+

PrivacySettings → passcode

=

Settings → passcode

```
func editSettings(provider: inout SettingsProvider) {


  func updateSetting(keyPath: AnyKeyPath,
                     title: String) {
    let value = provider[keyPath: setting.keyPath]
    if let nested = value as? SettingsProvider {
      for item in nested.settings {

        if let joined =
          keyPath.appending(path: item.keyPath) {
            updateSetting(keyPath: joined,
                          title: item.title)

    ...
}
```

```
✅ Settings: SettingsProvider {
  ✅ profileSettings: SettingsProvider {
  },
  ✅ privacySettings: SettingsProvider {
  },
}
```

```
✅ Settings: SettingsProvider {
  ✅ profileSettings: SettingsProvider {
    displayName: String,
    shareUpdates: Bool
  },
  ✅ privacySettings: SettingsProvider {
  },
}
```

```
if let nested = value as? SettingsProvider {
  for item in nested.settings {
    if let joined =
      keyPath.appending(path: item.keyPath) {
        updateSetting(keyPath: joined,
                      title: item.title)
    }
  }
} else {
}
}
```

```
if let nested = value as? SettingsProvider {
  for item in nested.settings {
    if let joined =
      keyPath.appending(path: item.keyPath) {
        updateSetting(keyPath: joined,
                      title: item.title)
    }
  }
} else {
  if let writable =
      keyPath as? WritableKeyPath<Root, Bool> {
          provider[keyPath: writable] = true
  }
}
```

```
if let nested = value as? SettingsProvider {
  for item in nested.settings {
    if let joined =
      keyPath.appending(path: item.keyPath) {
        updateSetting(keyPath: joined,
                      title: item.title)
    }
  }
} else {
  if let writable =
      keyPath as? WritableKeyPath<Root, Bool> {
          titleLabel.text = setting.title
          provider[keyPath: writable] = true
  }
}
```

Nested Providers

```
Settings {
  profileSettings {
    displayName: String,
    shareUpdates: Bool <--- Set to true
  },
  ...
}
```

The Final Code:

```swift
func editSettings<Root: SettingsProvider>(provider: inout Root) {

  func updateSetting(keyPath: AnyKeyPath, title: String) {
    let value = provider[keyPath: keyPath]
    if let nestedProvider = value as? SettingsProvider {
        for item in nestedProvider.settings {
            if let joined = keyPath.appending(path: item.keyPath) {
                updateSetting(keyPath: joined, title: item.title)
            }
        }
    } else if let writable = keyPath as? WritableKeyPath<Root, Bool> {
        provider[keyPath: writable] = true
    }
  }

  for setting in provider.settings {
      updateSetting(keyPath: setting.keyPath, title: setting.title)
  }
}
```

What did we achieve?

09:16

Settings

- Profile
- Privacy

09:16

My profile

Home

Display name
Partyboy

Share profile media updates

Share profile media updates on Timeline.

09:16

Privacy

Passcode lock

Allow others to add me by ID

Abstract over properties

Types of any shape can now be combined into our settings

Can have generic UI elements to read / write the types

They just need to describe themselves via [SettingEntry]

Three Tips for Using KeyPaths

1. Choose which Types to Erase

```
# KeyPath<A, B> = \User.age

# PartialKeyPath<A> = \User.age

# AnyKeyPath = \User.age
```

2. You can cast types back

```
# AnyKeyPath as? WritableKeyPath<User, String>

# PartialKeyPath<User> as? KeyPath<User, Bool>
```

3. KeyPaths conform to Hashable

Can be Keys in dictionaries

Useful to store more information about Keys

```swift
let meta: [PartialKeyPath<User>: String] = [
]
```

```swift
let meta: [PartialKeyPath<User>: String] = [
  \User.username: “Your Name”,
  \User.age: “Your Age”
]
```

```
let meta: [PartialKeyPath<User>: String] = [
  \User.username: “Your Name”,
  \User.age: “Your Age”
]

func renderTitle(keyPath: AnyKeyPath) {
}

renderTitle(\User.username)
```

```
let meta: [PartialKeyPath<User>: String] = [
  \User.username: “Your Name”,
  \User.age: “Your Age”
]

func renderTitle(keyPath: AnyKeyPath) {
   if let title = meta[keyPath]
     titleField.text = title
   }
}

renderTitle(\User.username)
```

KeyPath Libraries

Kuery by @k_katsumi

github.com/kishikawakatsumi/Kuery

Kuery

Type-Safe Core Data Queries

Core Data without strings

Kuery Example

```
// Before:
NSPredicate(format: "name == %@", “Mario")
NSPredicate(format: "age > %@", 20)

// After:
Query(Person.self).filter(\Person.name == “Mario")
Query(Person.self).filter(\Person.age > 20)
```

github.com/kishikawakatsumi/Kuery

KeyPathKit by @v_pradeilles

github.com/vincent-pradeilles/KeyPathKit

KeyPathKit

Useful abstractions for easier KeyPath usage

KeyPathKit

```
contacts.filter(where: \.lastName == “Webb"
    && \.age < 40)
contacts.average(of: \.age).rounded()
contacts.between(\.age, range: 20...30)
contacts.groupBy(\.lastName)
```

github.com/vincent-pradeilles/KeyPathKit

Sorting

```
users.sorted(by: .ascending(\.lastName),
                 .descending(\.address.city.name))
```

github.com/vincent-pradeilles/KeyPathKit

Recap

What did we learn

KeyPaths are type-safe, type-erased, hashable and composable

Create abstractions not possible with protocols

More Information

https://www.swiftbysundell.com/posts/the-power-of-key-paths-in-swift

https://www.klundberg.com/blog/swift-4-keypaths-and-you/

https://github.com/vincent-pradeilles/slides/blob/master/iosconfsg-2019-the-underestimated-power-of-keypaths.pdf

https://blog.slashkeys.com/practical-keypaths-in-swift-220da5ab5950

https://edit.theappbusiness.com/using-swift-keypaths-for-beautiful-user-preferences-c83c2f7ea7be

https://www.swiftbysundell.com/posts/the-power-of-key-paths-in-swift

https://github.com/makskovalko/FormValidation

Simple Swift Guides: www.appventure.me

Search or enter website name

APPVENTURE.ME Simple. Swift. Guides.

search

Guides

```
case (is Bool, let x as Int)
    where x > 10
```

🧩 Pattern Matching

In this post, we'll have a look at Pattern Matching in Swift in terms of the 'switch', 'for', 'if', and 'guard' keywords. We'll have a look at the basic syntax and at best practices and helpful examples.

```
enum Either<A, B> {
  case left(A), right(B)
}
```

🚦 Enums

When and how to use enums in Swift? This is a detailed practical overview of all the possibilities enums can offer you.

```
array.reduce(0, { $0 += $1 })
```

🗺️ Map, Reduce & more

A practical introduction into the `map`, `compactMap`, `filter`, and `reduce` functions.

```
protocol Host {
 associatedType Client
 }
```

🧙 Associated Types

Understand how to model your way around some of the issues that arise when introducing associated types into your protocols

```
(18, "Carl", [3, 5, 1])
```

🧱 Tuples

A brief introduction into Swift's tuples including a basic introduction and several useful tips for advanced usage.

```
a?.b?.c? == nil
```

⁉️ Optionals

Best practices for handling Swift's `Optional` and extending them to make them much more approachable

```
let m = Mirror(reflecting: me)
```

🔮 Reflection

In this guide we'll examine the Swift reflection API, see how fast it is, and will try to show use cases where it can be applied successfully.

SWIFT TOPICS

- All
- Swift Tricks
- Language
- Hands On
- Blog

GUIDES

- 🧩 Pattern Matching
- 🚦 Enums
- 🗺️ Map, Reduce & more
- 🧙 Associated Types
- 🧱 Tuples
- ⁉️ Optionals
- 🔮 Reflection

CONTRA VARIANCE A PODCAST

TAGS

- pattern matching 25
- switch 24
- enum 19
- associated 14
- tuples 13

@terhechte

CONTRA VARIANCE

A PODCAST

@BasThomas

Swift Weekly Brief

www.contravariance.rocks

XING

We're Hiring 😉

- German Social Network
- Based in Hamburg
- 15 Mio Users
- Native Apps on all platforms

THANKS!
ありがとう