Combine in Practice

Michael LeHew, Foundation
Ben D. Jones, Foundation
Publisher

request

Subscriber
Callbacks
Closures
Notifications
Key Value Observing
Publisher
Subscriber
A unified, declarative API for processing values over time
Introducing Combine

WWDC 2019

```
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
        where S.Input == Output, S.Failure == Failure
}
```
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
        where S.Input == Output, S.Failure == Failure
}
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
    where S.Input == Output, S.Failure == Failure
}
protocol Publisher {
  associatedtype Output
  associatedtype Failure: Error

  func subscribe<S: Subscriber>(_ subscriber: S)
  where S.Input == Output, S.Failure == Failure
}
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
    where S.Input == Output, S.Failure == Failure
}
protocol Publisher {
    associatedtype Output
    associatedtype Failure: Error

    func subscribe<S: Subscriber>(_ subscriber: S)
    where S.Input == Output, S.Failure == Failure
}
Trick Name
Description
Trick Name
Description
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)

some Publisher
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)

<table>
<thead>
<tr>
<th>Notification</th>
<th>// Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>// Failure</td>
</tr>
</tbody>
</table>
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
  .map { notification in
    return notification.userInfo?["data"] as! Data
  }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .tryMap { data in
        let decoder = JSONDecoder()
        try decoder.decode(MagicTrick.self, from: data)
    }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .tryMap { data in
        let decoder = JSONDecoder()
        try decoder.decode(MagicTrick.self, from: data)
    }
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .decode(MagicTrick.self, JSONDecoder())
Error Handling

Every Publisher describes how they can fail

Use operators to react/recover from errors
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .decode(MagicTrick.self, JSONDecoder())
    .assertNoFailure()
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .decode(MagicTrick.self, JSONDecoder())
    .assertNoFailure()
Error Handling Operators

assertNoFailure

Upstream Publisher — assertNoFailure — Subscriber
Error Handling Operators

assertNoFailure
Error Handling Operators

assertNoFailure

Upstream Publisher  assertNoFailure  Subscriber
Error Handling Operators

assertNoFailure

Thread 1: Fatal error: Publisher.assertNoFailure: Received error: ...
Failure Handling Operators

assertNoFailure
retry
catch
mapError
setFailureType
Failure Handling Operators

catch

Upstream Publisher — catch — Subscriber
Failure Handling Operators

- Catch

Upstream Publisher — catch — Subscriber
Failure Handling Operators

catch

Upstream Publisher

catch

Subscriber
Failure Handling Operators

catch

Upstream Publisher

catch

Recovery Publisher

Subscriber
Failure Handling Operators

catch

Upstream Publisher

catch

Recovery Publisher

Subscriber
Failure Handling Operators

```plaintext
Upstream Publisher

catch

Recovery Publisher

catch

Subscriber
```
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .decode(MagicTrick.self, JSONDecoder())
    .catch {
        return Just(MagicTrick.placeholder)
    }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .decode(MagicTrick.self, JSONDecoder())
    .catch {
        return Just(MagicTrick.placeholder)
    }
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .decode(MagicTrick.self, JSONDecoder())
    .catch {
        return Just(MagicTrick.placeholder)
    }
Notification
Never

Notification
Publisher
Notification

Never

Data

Never

Notification Publisher

map
Notification Never
Data Never
Notification Publisher
map
decode
MagicTrick Error
Notification Never

Notification Publisher

Data Never

map

MagicTrick Error

decode

MagicTrick Never

catch

Just placeholder
Just placeholder
Flat Map

Upstream Publisher — flatMap — Subscriber
Flat Map

Upstream Publisher -> flatMap -> Subscriber
Flat Map

Upstream Publisher

flatMap

Subscriber

Just

decode

catch
Flat Map

Upstream Publisher → flatMap → Subscriber

Just → decode → catch
Flat Map

Upstream Publisher \( \rightarrow \) flatMap \( \rightarrow \) Subscriber
Flat Map

Upstream
Publisher

flatMap

Subscriber

Just

decode

catch
Flat Map

Upstream Publisher

flatMap

Subscriber

Just

decode

catch
Flat Map

Upstream Publisher

flatMap

Subscriber

Just

decode

catch

Recovery Publisher
Flat Map

Upstream Publisher → flatMap → Subscriber

Just → decode → catch → Recovery Publisher
Flat Map

Upstream Publisher -- flatMap -- Subscriber
Flat Map

Upstream Publisher —> flatMap —> Subscriber

Recovery

decode

Just
catch
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
  .map { notification in
    return notification.userInfo?["data"] as! Data
  }
  .decode(MagicTrick.self, JSONDecoder())
  .catch {
    return Just(MagicTrick.placeholder)
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
  .map { notification in
    return notification.userInfo?"data" as! Data
  }
  .flatMap { data in
    return Just(data)
    .decode(MagicTrick.self, JSONDecoder())
    .catch {
      return Just(MagicTrick.placeholder)
    }
  }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }

MagicTrick
Never
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }
    .publisher(for: ".name")
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
  .map { notification in
    return notification.userInfo?"data" as! Data
  }
  .flatMap { data in
    return Just(data)
      .decode(MagicTrick.self, JSONDecoder())
      .catch {
        return Just(MagicTrick.placeholder)
      }
  }
  .publisher(for: \.name)
Scheduled Operators

Scheduler describes

• When
• Where

Supported by RunLoop and DispatchQueue
Scheduled Operators

delay
debounce
throttle
receive(on:)
subscribe(on:)

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
   .map { notification in
       return notification.userInfo?"data" as! Data
   }
   .flatMap { data in
       return Just(data)
           .decode(MagicTrick.self, JSONDecoder())
           .catch {
               return Just(MagicTrick.placeholder)
           }
   }
   .publisher(for: \.name)
   .receive(on: RunLoop.main)
// Using Publishers with Combine

let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?["data"] as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }
    .publisher(for: \.name)
    .receive(on: RunLoop.main)
Publishers

Recipe for an event stream

Operators describe new publishers from existing

Strongly typed values/errors over time

Can be synchronous or asynchronous

Can attach compatible Subscribers
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

    func receive(subscription: Subscription)
    func receive(_ value: Subscribers.Demand)
    func receive(completion: Subscribers.Completion<Failure>)
}
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

    func receive(subscription: Subscription)
    func receive(_ value: Subscribers.Demand)
    func receive(completion: Subscribers.Completion<Failure>)
}
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

    func receive(subscription: Subscription)
    func receive(_ value: Subscribers.Demand)
    func receive(completion: Subscribers.Completion<Failure>)
}
protocol Subscriber {
    associatedtype Input
    associatedtype Failure: Error

    func receive(subscription: Subscription)
    func receive(_ value: Subscribers.Demand)
    func receive(completion: Subscribers.Completion<Failure>)
}
Valid Streams
Subscription

Publisher

Subscriber
Valid Streams

Subscription

receive(subscription:)

Publisher  ————  Subscriber
Valid Streams
Zero or more values

Publisher ———— receive(_:) ———— Subscriber
Valid Streams
Zero or more values

Publisher  receive(_:):  Subscriber
Valid Streams
Completion

receive(completion:)

Publisher ———— Subscriber
Valid Streams

Completion

receive(completion:)
Valid Streams
Completion

receive(completion:)

Publisher

Subscriber
Valid Streams

Completion

receive(completion:)
Valid Streams
Completion

receive(completion:)
Exactly One Subscription
Exactly One Subscription
Exactly One Subscription  Zero or More Values
Exactly One Subscription

Zero or More Values
Exactly One Subscription

Zero or More Values

At Most One Completion
Exactly One Subscription

Zero or More Values

At Most One Completion

🧨
Kinds of Subscribers

Key Path Assignment
Sinks
Subjects
SwiftUI
let trickNamePublisher = NotificationCenter.default.publisher(for: .newTrickDownloaded)
    .map { notification in
        return notification.userInfo?"data" as! Data
    }
    .flatMap { data in
        return Just(data)
            .decode(MagicTrick.self, JSONDecoder())
            .catch {
                return Just(MagicTrick.placeholder)
            }
    }
    .publisher(for: \.name)
    .receive(on: RunLoop.main)
// Using Subscribers with Combine

let trickNamePublisher = ... // Publisher of <String, Never>
// Using Subscribers with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let canceller = trickNamePublisher.assign(to: ".someProperty", on: someObject)

// ...

canceller.cancel()
let trickNamePublisher = ... // Publisher of <String, Never>

let canceller = trickNamePublisher.assign(to: \.someProperty, on: someObject)

// ...

canceller.cancel()
Cancellation

Built into Combine

Terminate subscriptions early
Cancellation

Built into Combine

Terminate subscriptions early

```swift
protocol Cancellable {
    func cancel()
}

final class AnyCancellable: Cancellable {} // Calls `cancel` on deinit
```
Cancellation

Built into Combine

Terminate subscriptions early

```swift
protocol Cancellable {
    func cancel()
}

final class AnyCancellable: Cancellable {} // Calls `cancel` on deinit
```
Cancellation

Built into Combine

Terminate subscriptions early

```swift
protocol Cancellable {
    func cancel()
}

final class AnyCancellable: Cancellable {} // Calls `cancel` on deinit
```
// Using Subscribers with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let canceller = trickNamePublisher.sink { trickName in
  // Do Something with trickName
}
Subjects

Behave like both Publisher and Subscriber

Broadcast values to multiple subscribers

```swift
protocol Subject: Publisher, AnyObject {
    func send(_ value: Output)
    func send(completion: Subscribers.Completion<Failure>)
}
```
Subjects

Behave like both Publisher and Subscriber

Broadcast values to multiple subscribers

```swift
protocol Subject: Publisher, AnyObject {
    func send(_ value: Output)
    func send(completion: Subscribers.Completion<Failure>)
}
```
Subject
Subject

Subject

Subscriber

Subscriber
send(_::)

Subject

Subscriber

Subscriber
send(_:)
Kinds of Subjects

Passthrough

CurrentValue
Kinds of Subjects

Passthrough  CurrentValue
Kinds of Subjects

Passthrough

CurrentValue
Kinds of Subjects

- Passthrough
- CurrentValue
Kinds of Subjects

Passsthrough

CurrentValue
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let magicWordsSubject = PassthroughSubject<String, Never>()
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let magicWordsSubject = PassthroughSubject<String, Never>()

trickNamePublisher.subscribe(magicWordsSubject)
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let magicWordsSubject = PassthroughSubject<String, Never>()

trickNamePublisher.subscribe(magicWordsSubject)

let canceller = magicWordsSubject.sink { value in
    // do something with the value
}
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let magicWordsSubject = PassthroughSubject<String, Never>()

trickNamePublisher.subscribe(magicWordsSubject)

let canceller = magicWordsSubject.sink { value in
    // do something with the value
}

magicWordsSubject.send("Please")
// Using Subjects with Combine

let trickNamePublisher = ... // Publisher of <String, Never>

let magicWordsSubject = PassthroughSubject<String, Never>()

trickNamePublisher.subscribe(magicWordsSubject)

let canceller = magicWordsSubject.sink { value in
    // do something with the value
}

magicWordsSubject.send("Please")

let sharedTrickNamePublisher = trickNamePublisher.share()
Working with SwiftUI

SwiftUI owns the **Subscriber**

You just need to bring a **Publisher**
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
protocol BindableObject {
    associatedtype PublisherType : Publisher where PublisherType.Failure == Never

    var didChange: PublisherType { get }
}
// Combine with SwiftUI

class WizardModel {
    var trick: WizardTrick
    var wand: Wand?
}
// Combine with SwiftUI

class WizardModel: BindableObject {
    var trick: WizardTrick
    var wand: Wand?

    let didChange = PassthroughSubject<Void, Never>()
}

// Combine with SwiftUI

class WizardModel: BindableObject {
    var trick: WizardTrick
    var wand: Wand?

    let didChange = PassthroughSubject<Void, Never>()
}

// Combine with SwiftUI

class WizardModel : BindableObject {
    var trick: WizardTrick { didSet { didChange.send() } }
    var wand: Wand? { didSet { didChange.send() } }

    let didChange = PassthroughSubject<Void, Never>()
}

// Combine with SwiftUI

class WizardModel: BindableObject {
    var trick: WizardTrick { didSet { didChange.send() } }
    var wand: Wand? { didSet { didChange.send() } }

    let didChange = PassthroughSubject<Void, Never>()
}

struct TrickView: View {
    @ObjectBinding var model: WizardModel

    var body: some View {
        Text(model.trick.name)
    }
}
// Combine with SwiftUI

class WizardModel: BindableObject {
    var trick: WizardTrick { didSet { didChange.send() } }
    var wand: Wand? { didSet { didChange.send() } }

    let didChange = PassthroughSubject<Void, Never>()
}

struct TrickView: View {
    @ObjectBinding var model: WizardModel

    var body: some View {
        Text(model.trick.name)
    }
}
Many built in
• Publishers
• Subscribers
• Subjects

Common functionality in over 90 operators
Integrating Combine

Ben D. Jones, Foundation
Designed for composition
User name is valid according to server.
User name is valid according to server

Passwords Must Match
User name is valid according to server

Passwords Must Match
> 8 characters
User name is valid according to server
Enabled if username and passwords valid
Passwords Must Match > 8 characters
@IBAction func passwordChanged(_ sender: UITextField)

@IBAction func passwordAgainChanged(_ sender: UITextField)
@IBAction func passwordChanged(_ sender: UITextField)

@IBAction func passwordAgainChanged(_ sender: UITextField)
var password: String = ""

@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

var passwordAgain: String = ""

@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
var password: String = ""

@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

var passwordAgain: String = ""

@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
var password: String = ""

@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

var passwordAgain: String = ""

@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
@Published var password: String = ""

@IBAction func passwordChanged(_ sender: UITextField) {
    password = sender.text ?? ""
}

@Published var passwordAgain: String = ""

@IBAction func passwordAgainChanged(_ sender: UITextField) {
    passwordAgain = sender.text ?? ""
}
@Published

Property wrapper

Adds a publisher to any property
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink { 
    print("The published value is '
($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink {
    print("The published value is '\($0)'")
}

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink { print("The published value is '\($0)'") }

self.password = "password"
// Using @Published

@Published var password: String = ""

self.password = "1234"

let currentPassword: String = self.password

let printerSubscription = $password.sink { print("The published value is '\($0)'") }

self.password = "password"
Passwords must match
> 8 characters
@Published var password: String

$\text{password}$

@Published var passwordAgain: String

$\text{passwordAgain}$
@Published var password: String

$password

@Published var passwordAgain: String

$passwordAgain

validated
Password
@Published var password: String

@Published var passwordAgain: String

$password

$passwordAgain

? 

validated

Password ?
@Published var password: String

@Published var passwordAgain: String

$validated

CombineLatest
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
    }
    return password
}
@Published var password: String = ""

@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
    }
    return password
}
```swift
@Published var password: String = ""

@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}
```
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
    return password
} }
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: CombineLatest<Published<String>, Published<String>, String?> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
}

String?
Never
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: Map<CombineLatest<Published<String>, Published<String>, String?>> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
}

@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: Map<CombineLatest<Published<String>, Published<String>, String?>> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
}

String?

Never
@Published var password: String = ""
@Published var passwordAgain: String = ""

var validatedPassword: AnyPublisher<String?, Never> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
    .eraseToAnyPublisher()
}
@Published var password: String = ""

@Published var passwordAgain: String = ""

var validatedPassword: AnyPublisher<String?, Never> {
    return CombineLatest($password, $passwordAgain) { password, passwordAgain in
        guard password == passwordAgain, password.count > 8 else { return nil }
        return password
    }
    .map { $0 == "password1" ? nil : $0 }
    .eraseToAnyPublisher()
}
String
Never

$password

String
Never

$passwordAgain
String?
Never

String
Never

$password

String
Never

$passwordAgain

CombineLatest
CombineLatest

$password

$passwordAgain

map
CombineLatest

$password

$passwordAgain

map

eraseTo

AnyPublisher
eraseTo
AnyPublisher
var validatedPassword: AnyPublisher<String?, Never>
User name is valid according to server

Passwords must match > 8 characters

Enabled if username and passwords valid
User name is valid according to server

Enabled if username and passwords valid

Passwords must match > 8 characters
User name is valid according to server Enabled if username and passwords valid

Passwords must match > 8 characters
@Published var username: String

$username
@Published var username: String
Debounce

Upstream Publisher -> debounce -> Subscriber
Debounce

Upstream Publisher → debounce → Subscriber
Debounce

Upstream Publisher → debounce → Subscriber
@Published var username: String
@Published var username: String

$username  debounce  remove Duplicates
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {  
  return $username  
    .debounce(for: 0.5, scheduler: RunLoop.main)  
    .removeDuplicates()  
    .eraseToAnyPublisher()  
}
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}

String
Never
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .eraseToAnyPublisher()
}

// func usernameAvailable(_ username: String, completion: (Bool) -> Void)
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in

            // func usernameAvailable(_ username: String, completion: (Bool) -> Void)

        }
        .eraseToAnyPublisher()
}
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in
            // func usernameAvailable(_ username: String, completion: (Bool) -> Void)

        }
        .eraseToAnyPublisher()
}

@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
debounce(for: 0.5, scheduler: RunLoop.main)
    .removeDuplicates()
    .flatMap { username in
        return Future { promise in
            return Future { promise in

        }
    }
    .eraseToAnyPublisher()
}
@Published var username: String = ""

var validatedUsername: AnyPublisher<String, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in
            return Future { promise in
                (Result<Output, Failure>) -> Void
            }
        }
        .eraseToAnyPublisher()
}
@Published var username: String = ""

var validatedUsername: AnyPublisher<String?, Never> {
    return $username
        .debounce(for: 0.5, scheduler: RunLoop.main)
        .removeDuplicates()
        .flatMap { username in
            return Future { promise in
                self.usernameAvailable(username) { available in
                    promise(.success(available ? username : nil))
                }
            }
        }
        .eraseToAnyPublisher()
}
$username debounce remove Duplicates flatMap eraseTo AnyPublisher Future
$username \rightarrow \text{debounce} \rightarrow \text{remove Duplicates} \rightarrow \text{flatMap} \rightarrow \text{eraseTo AnyPublisher} \rightarrow \text{Future} \rightarrow \ldots \rightarrow \text{network}
$username debounce remove Duplicates flatMap eraseTo AnyPublisher
debounce
$username
remove Duplicates
Future
network
network
flatMap
eraseTo
AnyPublisher
```swift
var validatedUsername: AnyPublisher<String?, Never>
```
var validatedUsername: AnyPublisher<String?, Never>
```swift
var validatedPassword: AnyPublisher<String?, Never>
```

```
var validatedUsername: AnyPublisher<String?, Never>
```
User name is valid according to server

Enabled if username and passwords valid

Passwords must match > 8 characters
User name is valid according to server
Enabled if username and passwords valid
Passwords must match > 8 characters
var validatedCredentials: AnyPublisher<(String, String)?, Never> {
    return CombineLatest(validatedUsername, validatedPassword) { username, password in
        guard let uname = username, let pwd = password else { return nil }
        return (uname, pwd)
    }
    .eraseToAnyPublisher()
}
var validatedCredentials: AnyPublisher<(String, String)?, Never> {
    return CombineLatest(validatedUsername, validatedPassword) { username, password in
        guard let uname = username, let pwd = password else { return nil }
        return (uname, pwd)
    }
    .eraseToAnyPublisher()
}

(String, String)?
Never
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials.map { $0 != nil }.receive(on: RunLoop.main).assign(to: \.isEnabled, on: signupButton)
}
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}

@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
@IBOutlet var signupButton: UIButton!

var signupButtonStream: AnyCancellable?

override func viewDidLoad() {
    super.viewDidLoad()

    self.signupButtonStream = self.validatedCredentials
        .map { $0 != nil }
        .receive(on: RunLoop.main)
        .assign(to: \.isEnabled, on: signupButton)
}
Enabled if username and passwords valid

User name is valid according to server

Passwords Must Match
> 8 characters
User name is valid according to server

Enabled if username and passwords valid

Passwords Must Match

> 8 characters
$username

$password

$passwordAgain
Use Combine Today

Compose small parts into custom publishers

Adopt incrementally

Add a `Publisher` to a property with `@Published`

Compose callbacks and Publishers with `Future`
More Information

developer.apple.com/wwdc19/721