I've got a brand new Combine Publisher

(I'll give you the key)

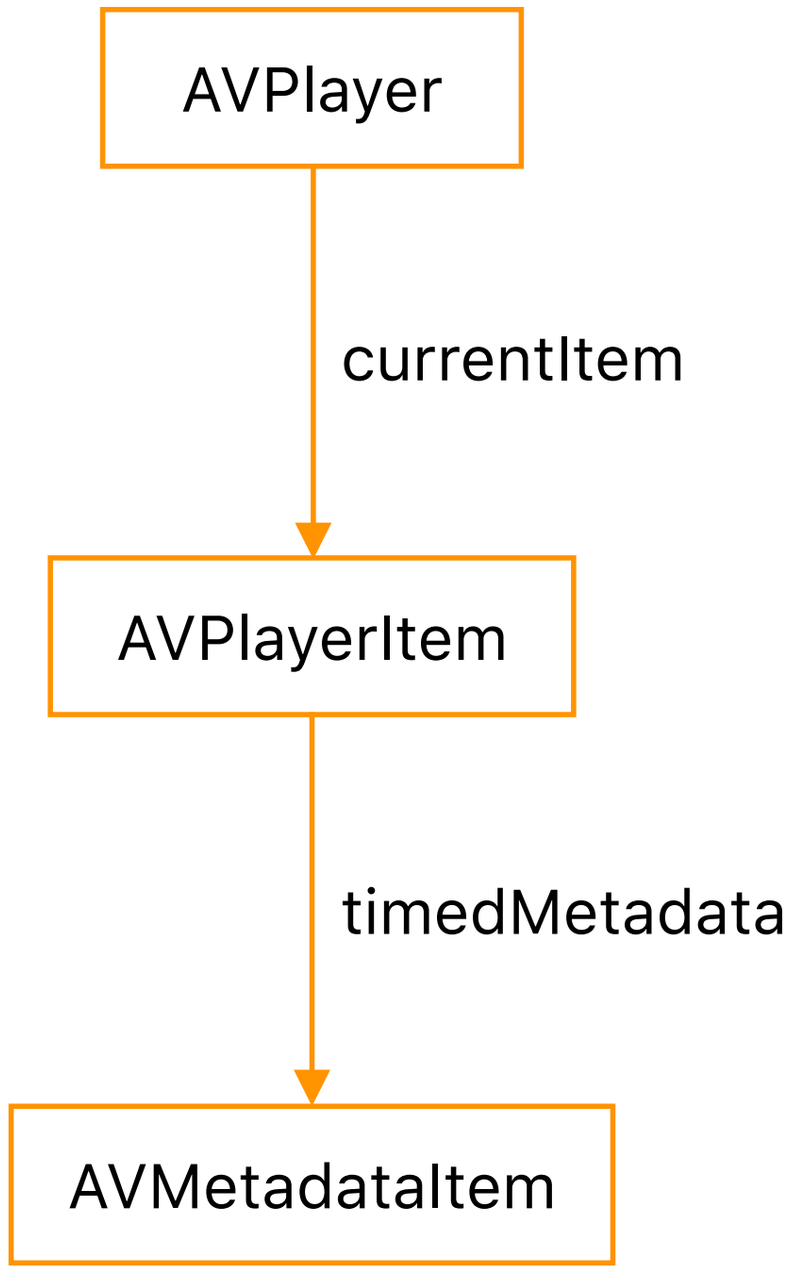
The Goal

As a listener to the **radio**

I would like to see the **details** of the **currently playing song**

So that **I appear to know about music**

Getting the timed metadata



Fetching iTunes track data

- Observe the current item of the player
- Observe the timed metadata of the current item
- Search for the title in the iTunes API
- Decode the JSON
- Capture any errors and replace with nil
- Set to a property on the view (which causes it to reload)

Publisher chaining

```
subscription = $player
    .flatMap { $0.publisher(for: \.currentItem) } // Observe currentItem
    .compactMap { $0 }
    .flatMap { $0?.publisher(for: \.timedMetadata) } // Observe timedMetadata
    .compactMap { $0?.title }
    .flatMap { term in

        urlSession
            .publisher(for: .search(term)) // Search for iTunes Item
            .map { $0.first } // Get first item
            .catch { _ in Just(nil) } // Silence error
}
.receive(on: DispatchQueue.main)
.assign(to: \.iTunesItem, on: self)
```

Ummm, thanks Apple

```
subscription = $player
    .flatMap { $0.publisher(for: \.currentItem) } // Observe currentItem
    .compactMap { $0 }
    .flatMap { $0.publisher(for: \.timedMetadata) } // Observe timedMetadata
    .compactMap { $0?.tit
    .flatMap { term in
        session
            .publisher(for: .search(term)) // Search for iTunes Item
            .map { $0.first } // Get first item
            .catch { _ in Just(nil) } // Silence error
    }
    .receive(on: DispatchQueue.main)
    .assign(to: \.currentItem, on: self)
```

'timedMetadata' was deprecated in iOS 13.0: Use AVPlayerItemMetadataOutput to obtain timed metadata

Add an extension to provide the publisher

```
extension AVPlayerItem {

    public var metadataPublisher: MetadataPublisher {
        MetadataPublisher(item: self)
    }
}
```

Create the publisher

```
public struct MetadataPublisher {
    fileprivate let item: AVPlayerItem
}
```

Conform to the Publisher protocol

```
extension MetadataPublisher: Publisher {

    public typealias Output = AVTimedMetadataGroup
    public typealias Failure = Never

    public func receive<S>(subscriber: S)
        where
    S: Subscriber,
    S.Failure == Failure,
    S.Input == Output {

        let subscription = Subscription(item: item, subscriber: subscriber)
        subscriber.receive(subscription: subscription)
    }
}
```

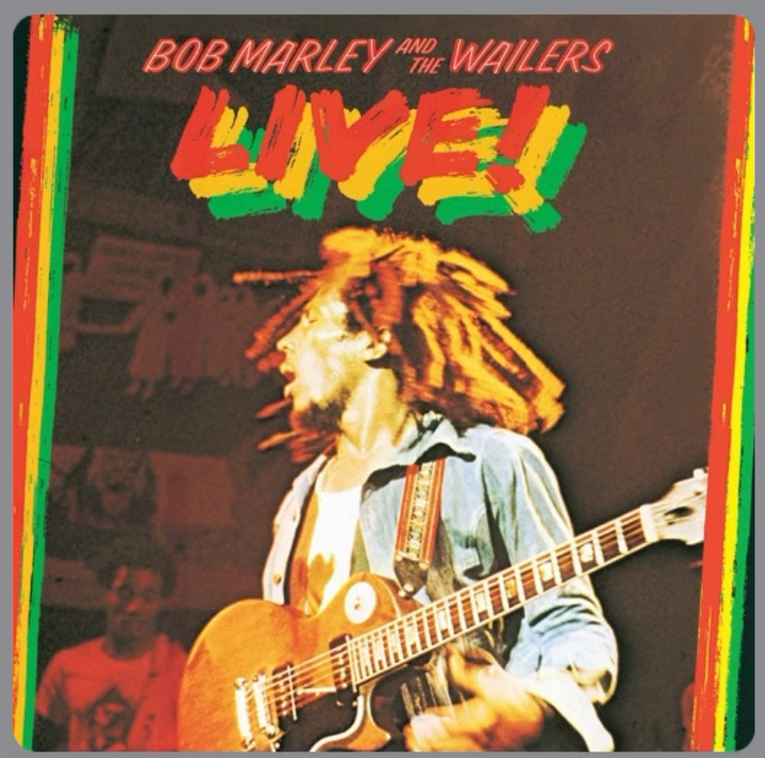
```
extension MetadataPublisher {

    fileprivate final class Subscription<Subscriber>:
        NSObject,
        AVPlayerItemMetadataOutputPushDelegate
    where
        Subscriber: Combine.Subscriber,
        Subscriber.Failure == Failure,
        Subscriber.Input == Output
    {

        private let item: AVPlayerItem
        private let subscriber: Subscriber
        private let metadataOutput = AVPlayerItemMetadataOutput()

        fileprivate init(item: AVPlayerItem, subscriber: Subscriber) {
            self.item = item
            self.subscriber = subscriber
        }

        // AVPlayerItemMetadataOutputPushDelegate

        func metadataOutput(_: AVPlayerItemMetadataOutput,
                            didOutputTimedMetadataGroups groups: [AVTimedMetadataGroup],
                            from: AVPlayerItemTrack?) {

            groups.forEach { _ = subscriber.receive($0) }
        }
    }
}
```

Conform to the Subscription protocol

```
extension MetadataPublisher.Subscription: Combine.Subscription {

    func request(_ demand: Subscribers.Demand) {
        let queue = DispatchQueue(label: "uk.co.danieltull.MetadataQueue")
        metadataOutput.setDelegate(self, queue: queue)
        item.add(metadataOutput)
    }

    func cancel() {
        item.remove(metadataOutput)
        metadataOutput.setDelegate(nil, queue: nil)
    }
}
```

Using the new publisher

```
subscription = $player
    .flatMap { $0.publisher(for: \.currentItem) } // Observe currentItem
    .compactMap { $0 }
    .flatMap { $0.metadataPublisher } // Observe metadata
    .compactMap { $0.title }
    .flatMap { term in

        urlSession
            .publisher(for: .search(term)) // Search for iTunes Item
            .map { $0.first } // Get first item
            .catch { _ in Just(nil) } // Silence error
}
.receive(on: DispatchQueue.main)
.assign(to: \.iTunesItem, on: self)
```



Lively Up Yourself
Bob Marley & The Wailers

Thank you

@danielctull