I've got a brand new Combine Publisher

(I'll give you the key)

What is Combine? 

The Combine framework provides a declarative Swift API for ***processing values*** *over* ***time***.

— Apple Documentation

synchronous functions

in

asynchronous situations

What is this all for? 

LinkPresentation

```
open class LPMetadataProvider : NSObject {

    func startFetchingMetadata(
        for URL: URL,
        completionHandler: @escaping (LPLinkMetadata?, Error?) -> Void
    )
}
```

Traditional fetching 

Traditional fetching

```
struct MetadataView: View {

    @State var title: String?
    var body: some View { Text(title ?? "") }

    func traditionalFetchMatadata(for url: URL) {

        LPMetadataProvider().startFetchingMetadata(for: url) {
            (metadata: LPLinkMetadata?, error: Error?) in

            guard let metadata = metadata else {
                print(error!.localizedDescription)
                return
            }

            DispatchQueue.main.async {
                self.title = metadata.title
            }
        }
    }
}
```

Combine Fetching 

Combine Fetching

```
struct CombineMetadataView: View {

    @State var title: String?
    var body: some View { Text(title ?? "") }

    @State var cancellable: AnyCancellable?

    func combineFetchMatadata(for url: URL) {

        cancellable = LPMetadataProvider()
            .metadata(for: url)
            .catch { _ in Empty<LPLinkMetadata, Never>() }
            .map(\.title)
            .receive(on: DispatchQueue.main)
            .assign(to: \.title, on: self)
    }
}
```

How do we implement a publisher? 

Add an extension to provide the publisher

```
extension LPMetadataProvider {

    func metadata(for url: URL) -> LinkMetadataPublisher {
        LinkMetadataPublisher(provider: self, url: url)
    }
}
```

Create the publisher

```
struct LinkMetadataPublisher {
    typealias Output = LPLinkMetadata
    typealias Failure = Error
    private let provider: LPMetadataProvider
    private let url: URL

    public init(provider: LPMetadataProvider, url: URL) {
        self.provider = provider
        self.url = url
    }
}
```

Conform to the Publisher protocol

```
extension LinkMetadataPublisher: Publisher {

    func receive<Subscriber>(subscriber: Subscriber)
        where
        Subscriber: Combine.Subscriber,
        Subscriber.Failure == Failure,
        Subscriber.Input == Output
    {
        let subscription = Subscription(subscriber: subscriber,
                                        provider: provider,
                                        url: url)
        subscriber.receive(subscription: subscription)
    }
}
```

Implement a subscription

```
extension LinkMetadataPublisher {

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

        private let provider: LPMetadataProvider
        private let url: URL
        private let subscriber: Subscriber

        fileprivate init(subscriber: Subscriber,
                         provider: LPMetadataProvider,
                         url: URL) {
            self.subscriber = subscriber
            self.provider = provider
            self.url = url
        }
    }
}
```

Conform to the Subscription protocol

```
extension LinkMetadataPublisher.Subscription: Combine.Subscription {

    func request(_ demand: Subscribers.Demand) {
        let subscriber = self.subscriber
        provider.startFetchingMetadata(for: url) { metadata, error in

            guard let metadata = metadata else {
                subscriber.receive(completion: .failure(error!))
                return
            }

            subscriber.receive(metadata)
            subscriber.receive(completion: .finished)
        }
    }

    func cancel() {}
}
```

But wait! 

What about all those other completion closure APIs??

WebKit

```
open class WKWebView : NSView {

    func evaluateJavaScript(
        _ javaScriptString: String,
        completionHandler: ((Any?, Error?) -> Void)? = nil
    )
}
```

CoreLocation

```
open class CLGeocoder : NSObject {

    func reverseGeocodeLocation(
        _ location: CLLocation,
        completionHandler: @escaping CLGeocodeCompletionHandler
    )
}

typealias CLGeocodeCompletionHandler = ([CLPlacemark]?, Error?) -> Void
```

MapKit

```
open class MKLocalSearch : NSObject {

    func start(completionHandler: @escaping CompletionHandler)

    typealias CompletionHandler = (MKLocalSearch.Response?, Error?) -> Void
}
```

The completion closures

```
(LPLinkMetadata?,                Error?  ) -> Void

((Any?,                          Error?  ) -> Void)? = nil

([CLPlacemark]?,                 Error?  ) -> Void

(MKLocalSearch.Response?, Error?  ) -> Void
```

The completion closures

```
(LPLinkMetadata?,                Error?  ) -> Void

((Any?,                          Error?  ) -> Void)? = nil

([CLPlacemark]?,                 Error?  ) -> Void

(MKLocalSearch.Response?,        Error?  ) -> Void
```



```
(Output?,                        Failure?) -> Void
```

Let's make the publisher generic! 

Specific

```
extension LinkMetadataPublisher.Subscription: Combine.Subscription {

    func request(_ demand: Subscribers.Demand) {
        let subscriber = self.subscriber
        provider.startFetchingMetadata(for: url) { metadata, error in

            guard let metadata = metadata else {
                subscriber.receive(completion: .failure(error!))
                return
            }

            subscriber.receive(metadata)
            subscriber.receive(completion: .finished)
        }
    }

    func cancel() {}
}
```

Generic

```
extension CompletionPublisher.Subscription: Combine.Subscription {

    func request(_ demand: Subscribers.Demand) {
        let subscriber = self.subscriber
        perform { output, failure in

            guard let output = output else {
                subscriber.receive(completion: .failure(failure!))
                return
            }

            subscriber.receive(output)
            subscriber.receive(completion: .finished)
        }
    }

    func cancel() {}
}
```

Specific

```
struct LinkMetadataPublisher {
    typealias Output = LPLinkMetadata
    typealias Failure = Error

    private let provider: LPMetadataProvider
    private let url: URL

    public init(provider: LPMetadataProvider, url: URL) {
        self.provider = provider
        self.url = url
    }
}
```

Generic

```
struct CompletionPublisher<Output, Failure: Error> {


    typealias Completion = (Output?, Failure?) -> Void
    private let perform: (@escaping Completion) -> Void

    public init(perform: @escaping (@escaping Completion) -> Void) {
        self.perform = perform
    }
}
```

Specific

```
extension LinkMetadataPublisher: Publisher {

    func receive<Subscriber>(subscriber: Subscriber)
        where
        Subscriber: Combine.Subscriber,
        Subscriber.Failure == Failure,
        Subscriber.Input == Output
    {
        let subscription = Subscription(subscriber: subscriber,
                                        provider: provider,
                                        url: url)
        subscriber.receive(subscription: subscription)
    }
}
```

Generic

```
extension CompletionPublisher: Publisher {

    func receive<Subscriber>(subscriber: Subscriber)
        where
        Subscriber: Combine.Subscriber,
        Subscriber.Failure == Failure,
        Subscriber.Input == Output
    {
        let subscription = Subscription(subscriber: subscriber,
                                        perform: perform)

        subscriber.receive(subscription: subscription)
    }
}
```

Specific

```
extension LinkMetadataPublisher {

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

        private let provider: LPMetadataProvider
        private let url: URL
        private let subscriber: Subscriber

        fileprivate init(subscriber: Subscriber,
                         provider: LPMetadataProvider,
                         url: URL) {
            self.subscriber = subscriber
            self.provider = provider
            self.url = url
        }
    }
}
```

Generic

```
extension CompletionPublisher {

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

        private let perform: (@escaping Completion) -> Void

        private let subscriber: Subscriber

        fileprivate init(subscriber: Subscriber,
                         perform: @escaping (@escaping Completion) -> Void) {

            self.subscriber = subscriber
            self.perform = perform

        }
    }
}
```

Specific

```
extension LPMetadataProvider {

    func metadata(for url: URL) -> LinkMetadataPublisher {
        LinkMetadataPublisher(provider: self, url: url)
    }
}
```

Generic

```
extension LPMetadataProvider {

    func metadata(for url: URL) -> CompletionPublisher<LPLinkMetadata, Error> {

        CompletionPublisher(perform: { completion in
            self.startFetchingMetadata(for: url, completionHandler: completion)
        })
    }
}
```

WebKit

```
extension WKWebView {

    func evaluateJavascript(_ javaScript: String) -> CompletionPublisher<Any, Error> {

        CompletionPublisher(perform: { completion in
            self.evaluateJavaScript(javaScript, completionHandler: completion)
        })
    }
}
```

CoreLocation

```
extension CLGeocoder {

    func reverseGeocodeLocation(_ location: CLLocation) -> CompletionPublisher<[CLPlacemark], Error> {

        CompletionPublisher(perform: { completion in
            self.reverseGeocodeLocation(location, completionHandler: completion)
        })
    }
}
```

MapKit

```
extension MKLocalSearch {

    static func publisher(for request: MKLocalSearch.Request) -> CompletionPublisher<MKLocalSearch.Response, Error> {

        let request = MKLocalSearch(request: request)
        return CompletionPublisher(perform: request.start)
    }
}
```

Thank you

@danielctull

https://danieltull.co.uk