About ‘PurchaseKit’

PurchaseKit is a solution to integrate in-app purchases into iOS apps with just a few steps and most importantly, offering you a couple of great themes to select from, which will boost your in-app purchase sales.

PurchaseKit.xcframework is the main framework that handles all the configurations, purchases, receipt validation and more. You don’t need to write any code to handle in-app purchases anymore, this framework will do the hard work for you.

PurchaseKit folder includes a few themes that will magically present the in-app purchase products, with the price, trial period and much more. You can easily edit these themes in SwiftUI.

Currently the framework supports only the following types of in-app purchases:

- **Consumable**
  A product that is used once, after which it becomes depleted and must be purchased again.

  Example: Fish food for a fishing app.

- **Non-Consumable**
  A product that is purchased once and does not expire or decrease with use.

  Example: Race track for a game app.

- **Auto-Renewable Subscription**
  A product that allows users to purchase dynamic content for a set period. This type of subscription renews automatically unless cancelled by the user.

  Example: Monthly subscription for an app offering a streaming service.

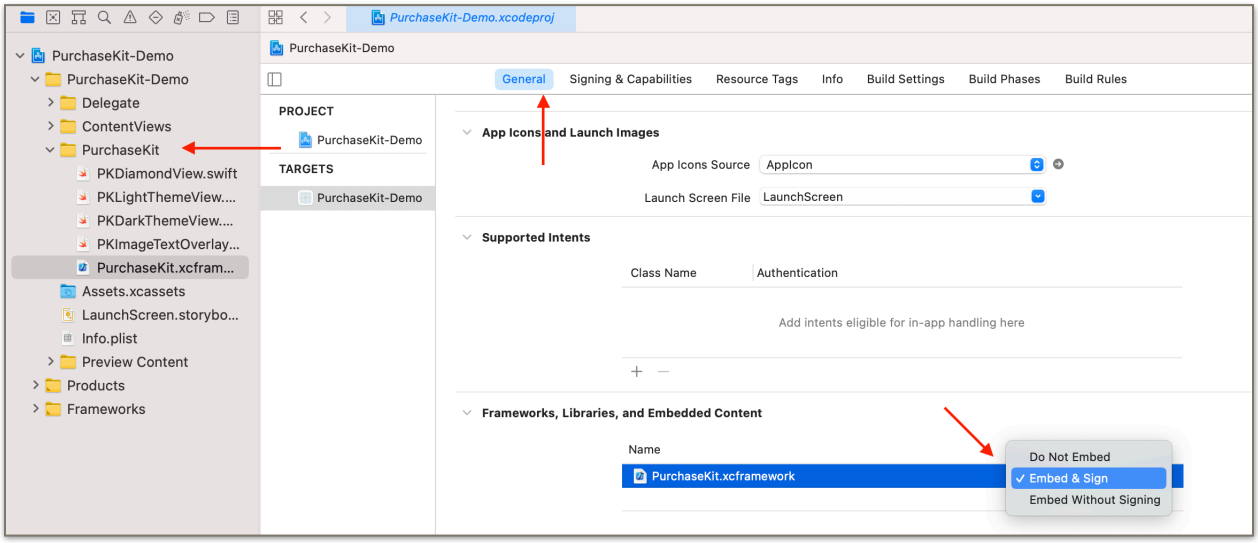
- **Non-Renewing Subscription**
  A product that allows users to purchase a service with a limited duration. The content of this in-app purchase can be static. This type of subscription does not renew automatically.

  Example: One-year subscription to a catalog of archived articles.

How to Add ‘PurchaseKit’
TO YOUR APP

STEP #1:

Drag & Drop the *PurchaseKit* folder into **your** app Xcode project. Then select your app’s target -> *General* and make sure you select *Embed & Sign* for the *PurchaseKit.xcframework*



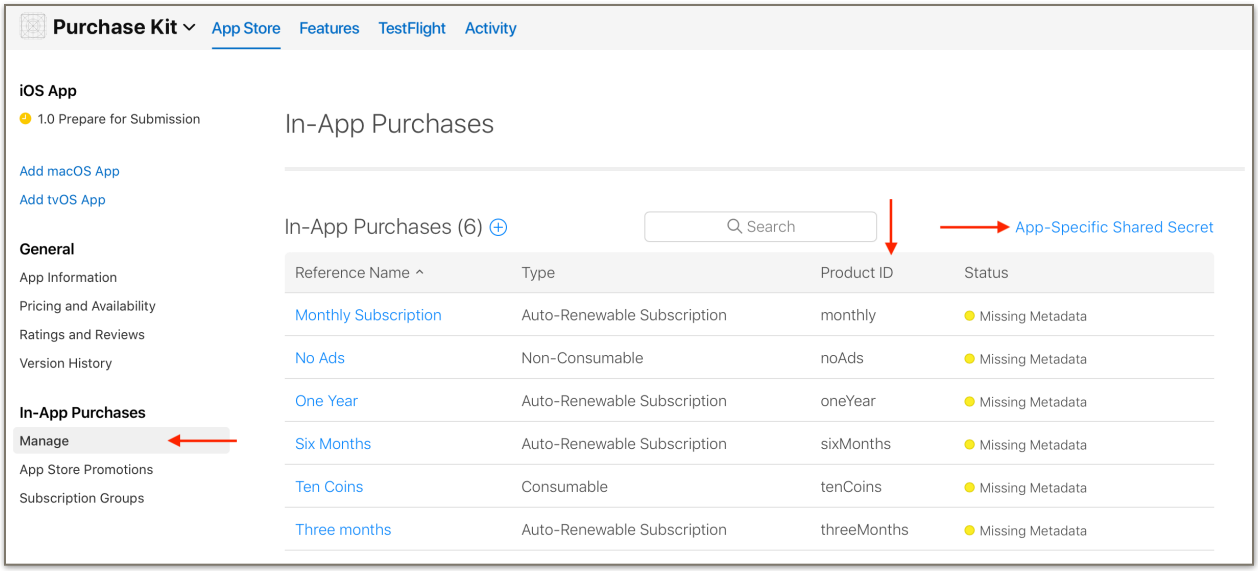
STEP #2:

Configure the *PurchaseKit* framework with your in-app purchase identifiers (*Product ID*) and *App-Specific Shared Secret.*

See image below from **App Store Connect**, to understand where to get your *App-Specific Shared Secret* and *In-App Purchase IDs.*

Make sure that the status for all Product IDs is *“Ready to Submit”.*

Now that you have you *App-Specific Shared Secret* and *Product IDs.* It’s time to configure the *PurchaseKit* inside *AppDelegate*.



Open *AppDelegate.swift* in Xcode -> import *PurchaseKit* and configure the framework as shown below:

```
import UIKit
import PurchaseKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

        PKManager.configure(sharedSecret: "b79fc31c299d4b94adc1c10f2f4aa518")
        PKManager.loadProducts(identifiers: ["tenCoins", "noAds", "monthly", "sixMonths", "oneYear", "threeMonths"])

        return true
    }

    func application(_ application: UIApplication, configurationForConnecting connectingSceneSession: UISceneSession, options:
        UIScene.ConnectionOptions) -> UISceneConfiguration {
        return UISceneConfiguration(name: "Default Configuration", sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions: Set<UISceneSession>) { }
}
```

Congratulations!
You have successfully integrated the *PurchaseKit* into your app.
Build & Run your app on a **real device** (not simulator).

You should see in Xcode Console the PurchaseKit logs:

```
✦ PurchaseKit ✦ – Requesting products with Identifiers:
✦ PurchaseKit ✦ – Found products
```

If you've setup everything correct on the App Store Connect, then you will see the products with the identifier, title and price exactly as you've set them on the ASC.

```
Found product
Identifier: monthly, Title: Monthly, Price: 1.99
```

How to Present iAP Screen

Your app should have a button or any other element that will trigged the presentation of one of the *PurchaseKit* themes.

***** THE EXAMPLE BELOW IS FOR SWIFTUI - NOT SWIFT *****
For this example, let assume you have a *ContentView.swift* file with a button and tapping it, will present the iAP (in-app purchase) screen.

```swift
/// Your screen that will present the iAP screen
struct ContentView: View {

    /// This boolean will be toggled when user taps the button which will trigger the iAP screen
    @State private var showiAPScreen: Bool = false

    /// Default rendering function
    var body: some View {
        VStack {
            Button(action: {
                self.showiAPScreen.toggle()
            }, label: {
                Text("Show iAP Screen")
            })
        }

        /// Implement this to present the iAP screen from bottom of the screen
        .sheet(isPresented: $showiAPScreen, content: {

            /// Present one of the `PurchaseKit` themes from PurchaseKit folder
            PKDarkThemeView(title: "Go Premium",
                            subtitle: "Unlock all premium features",
                            features: ["Remove Ads", "Access to all content", "Other cool features"],
                            productIds: ["noAds", "month", "oneYear"]) { (error, status, productId) in
                /// Handle in-app purchase response from Apple
                // In this statement, we check to make sure that there is no error to show
                if error == nil {
                    if status == .success {
                        /// When the purchase status was `success`, unlock the content you want
                        /// For example, set some booleans to hide ads
                    }
                }
            }
        })
    }
}
```

Have a **@State** boolean that will be toggled when the user taps the button, then implement **.sheet(isPresented:).**

See the image example below:

See the demo app for more examples and details.
If you have an app that is built using Swift instead of SwiftUI, you can use the *PurchaseKit* to present one of the iAP screens.

***** THE EXAMPLE BELOW IS FOR SWIFT - NOT SWIFT UI *****
For this example in Swift, you can easily call a function on the *PurchaseKit* framework and pass the iAP screen (theme).

You can also use the *UIHostingController* yourself if you want to present any SwiftUI view in Swift.

See the image example below:

```swift
import UIKit
import SwiftUI
import PurchaseKit

/// Your screen that will present the iAP screen
class ViewController: UIViewController {

    /// Your button action which will trigger the iAP screen
    @IBAction func showiAPScreen(_ sender: UIButton) {
        /// Present one of the `PurchaseKit` themes from PurchaseKit folder
        let theme = PKDarkThemeView(title: "Go Premium",
                                    subtitle: "Unlock all premium features",
                                    features: ["Remove Ads", "Access to all content", "Other cool features"],
                                    productIds: ["noAds", "month", "oneYear"]) { (error, status, productId) in
            /// Handle in-app purchase response from Apple
            // In this statement, we check to make sure that there is no error to show
            if error == nil {
                if status == .success {
                    /// When the purchase status was `success`, unlock the content you want
                    /// For example, set some booleans to hide ads
                }
            }
        }

        /// Present your iAP screen/theme
        PKManager.present(theme: AnyView(theme), fromController: self)
    }
}
```

Good job!
Now you know how to present the in-app purchase screen.

All purchases/restore purchases are happening via *PurchaseKit* framework and the SwiftUI themes/views.

How to unlock content

Your app should always verify the receipt for auto-renewable subscriptions, this way you can decide if you want to unlock the new features/content for that user or lock everything if their subscription expired.

Call the **PKManager.verifySubscription** function as soon as you need to verify if a given subscription for a specific Product ID is still active or expired.

```
/// If you have a subscription iAP, make sure to verify the receipt as soon as possible. Ideally when the app launches, so you could unlock the content
.onAppear(perform: {
    PKManager.verifySubscription(identifier: "monthly") { (_, subscriptionStatus, productId) in
        /// Check if the product identifier matches your subscribtion product id
        if productId == "monthly" {
            /// Check if the status for subscription is `success`, then and only then unlock the content
            if subscriptionStatus == .success {
                self.overlayThemeConsumableIAPStatus = true
            }
        }
    }
})
```

You can call this in viewDidLoad or on your screen where you have content that needs to be locked/unlock via a subscription.

That's all!

Please reach out to me for any questions you may have:
E-mail: support@apps4world.com
Skype: Apps4World