MAOR — iOS Mobile Advertising Orchestrator and Rendering

Distribution repository for the MAOR iOS framework. It ships the pre-built framework (`MAOR.xcframework`) exposed as a Swift Package, together with this integration guide.

> This is the repo to integrate into your app. For development of the MAOR SDK itself, refer to the internal source repository.
>
> CocoaPods is no longer supported. Distribution is provided exclusively via **Swift Package Manager**.

Table of contents

- Requirements
- Installation (SPM)
- Project configuration
- Initialization
- Permissions (ATT + CMP)
- User profiling
- Requesting ads
- Prefetch
- Interstitial loader
- External identity (LiveRamp / ID5)
- IMA helpers (instream)
- Error handling
- API reference
- Integration checklist

Requirements

Item	Value
Minimum iOS	**16.0**
Swift tools	**5.8+**
Distribution	Swift Package Manager (`Package.swift`)

Distribution	Swift Package Manager (`Package.swift`)
Xcode	15+ recommended

The package automatically resolves all transitive dependencies — the integrating app **does not** need to declare them manually:

- Google Mobile Ads SDK (13.x)
- Google Interactive Media Ads (3.x)
- Prebid Mobile (3.x)
- Firebase Performance (12.x)
- Promises (2.x)
- Iubenda Mobile SDK
- LiveRamp ATS SDK + Mediation Adapter
- Amazon Publisher Services (APS) + DTBiOSSDK

Installation (SPM)

From Xcode

1. `File` → `Add Package Dependencies…`
2. Enter the URL of this repository:

   ```
   <DISTRIBUTION_REPO_URL>
   ```

3. Pick a version rule (e.g. `Up to Next Major Version`) and add the **MAOR** product to your app target.

From `Package.swift`

```
dependencies: [
    .package(
        url: "<DISTRIBUTION_REPO_URL>",
        from: "4.5.0"
    )
],
targets: [
    .target(
        name: "MyApp",
        dependencies: [
            .product(name: "MAOR", package: "MAOR")
        ]
    )
]
```

```
]
```

> The package ships `MAOR.xcframework` as a binary target plus all transitive dependencies. You don't need to add GAM/Prebid/Iubenda/APS manually.

Project configuration

Info.plist

Required keys on the app target:

```
<key>GADApplicationIdentifier</key>
<string>ca-app-pub-XXXXXXXXXXXXXXXX~YYYYYYYYYY</string>

<key>NSUserTrackingUsageDescription</key>
<string>Message shown in the ATT prompt</string>

<key>SKAdNetworkItems</key>
<array>
    <!-- list of SKAdNetworkIdentifier entries required by the ad networks -->
</array>

<key>UIAppFonts</key>
<array>
    <string>MyAppFont-Regular.ttf</string>
    <string>MyAppFont-Bold.ttf</string>
</array>
```

`GADApplicationIdentifier` is required by Google Mobile Ads.
`NSUserTrackingUsageDescription` is required whenever the app triggers the ATT prompt.

Firebase

MAOR includes `FirebasePerformance`. The integrating app is responsible for initializing Firebase:

```
import FirebaseCore

FirebaseApp.configure()
```

and for adding `GoogleService-Info.plist` to the app target.

Initialization

The orchestrator is a singleton: `MobileADVOrchestrator.shared`.

Initialization is **asynchronous**: it triggers configuration download, GAM/Prebid/APS setup, and CMP bootstrap. It returns a **key** (`String`) **immediately**, to be stored and passed to all subsequent calls. An `onCompletion` closure is invoked when bootstrap finishes.

Signature

```
public func initialize(
    fontName: String,
    options: Set<InitializationOptions>?,
    isTest: Bool = false,
    storeID: String?,
    configUrl: String,
    globalConfigUrl: String?,
    liverampAppId: String?,
    onCompletion: ((Result<Void, Error>) -> Void)? = nil
) -> String
```

Parameter	Description
`fontName`	Font name (registered in `UIAppFonts`) used for autopromo/backup rendering
`options`	Set of `InitializationOptions` (ATT, CMP, analytics, yieldbird fallback)
`isTest`	`true` for test environment (test ads)
`storeID`	App Store ID of the hosting app (numeric string)
`configUrl`	URL of the per-domain ad configuration JSON
`globalConfigUrl`	URL of the global configuration JSON (ATT/CMP/limited ads/timeouts)
`liverampAppId`	LiveRamp ATS app ID (can be `nil`)
`onCompletion`	Callback with `Result<Void, Error>` once bootstrap completes

Returns the `key` to reuse in `getAds` / `prefetchAds` / `requestPermissions`.

`InitializationOptions`

```
public enum InitializationOptions: Hashable {
    case requestATT((AdTrackingStatus) -> Void)
    case requestCookies(onPermissionUpdate: (PermissionStatus, PermissionSettings?
    case analytics(onAnalytics: ((AnalyticsType, AnalyticsAd) -> Void))
```

```
    case yielbirdFallback(value: Bool)
}
```

- `requestATT` — MAOR handles the App Tracking Transparency prompt; the closure is invoked with the final status.
- `requestCookies` — MAOR handles the Iubenda CMP; the closure receives `PermissionStatus` (`.given` / `.withdrawn`) and the IAB settings.
- `analytics` — receives callbacks for relevant events (e.g. `.googleError`, `.noCMP`).
- `yielbirdFallback` — when `true`, the fallback chain routes through Yieldbird first, then autopromo.

Full example

```
import UIKit
import MAOR
import FirebaseCore

@main
final class AppDelegate: UIResponder, UIApplicationDelegate {
    static var maorKey: String = ""

    var onPermissionUpdate: (PermissionStatus, PermissionSettings?) -> Void = { _,
    var onATTUpdate: (AdTrackingStatus) -> Void = { _ in }

    func application(
        _ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsK
    ) -> Bool {
        FirebaseApp.configure()

        // Default profiling level (see "User profiling")
        MobileADVOrchestrator.shared.onProfileExpired { setNext in
            setNext(.disabled)
        }

        AppDelegate.maorKey = MobileADVOrchestrator.shared.initialize(
            fontName: "MyAppFont-Regular",
            options: [
                .requestATT { [weak self] status in
                    self?.onATTUpdate(status)
                },
                .requestCookies { [weak self] status, settings in
                    self?.onPermissionUpdate(status, settings)
                },
                .yielbirdFallback(value: true),
                .analytics { type, ad in
                    print("MAOR analytics:", type, ad)
                }
            ],
            isTest: false,
```

```
            isTest: false,
            storeID: "945341788",
            configUrl: "https://i.plug.it/mail/app/config/v8/adv.json",
            globalConfigUrl: "https://i.plug.it/adv/app/config/v14/libero_global_c
            liverampAppId: "a990478e-98b2-407b-a88a-952a25b38eca",
            onCompletion: { result in
                switch result {
                case .success:
                    print("MAOR ready")
                case .failure(let error):
                    print("MAOR init failed:", error.localizedDescription)
                }
            }
        )

        return true
    }
}
```

> The `key` can be stored anywhere (singleton, `AppDelegate`, DI container). Multiple `initialize` calls with different configurations produce different keys: multiple configurations can coexist.

Permissions (ATT + CMP)

Combined request

Shows ATT and CMP according to what is declared in `globalConfig`. Must be called on a visible view controller.

```
let (askedATT, askedCMP) = MobileADVOrchestrator.shared.requestPermissions(
    key: AppDelegate.maorKey,
    on: self
)
```

The returned `(Bool, Bool)` tuple tells whether ATT/CMP were actually requested.

Force the CMP page

```
MobileADVOrchestrator.shared.showCookiePermissionPage(on: self)
```

CMP state and checks

```
let expressed   = MobileADVOrchestrator.shared.userExpressedPermission     // Bool
let status      = MobileADVOrchestrator.shared.iubendaStatus               // giu
```

```
let status     = MobileADVOrchestrator.shared.iubendaStatus              // .giv
let settings   = MobileADVOrchestrator.shared.iubendaSettings            // Perm
let iubendaID  = MobileADVOrchestrator.shared.getIubendaID()             // Stri
let vendorOK   = MobileADVOrchestrator.shared.isConsentGivenForVendor(withId: 755
let purposeOK  = MobileADVOrchestrator.shared.isConsentGivenForPurpose(withId: 1)
```

CMP operations

```
MobileADVOrchestrator.shared.acceptCMP()       // accept all consents
MobileADVOrchestrator.shared.clearCMP()        // clear all consents
MobileADVOrchestrator.shared.resetPermission() // reset the permission handler
MobileADVOrchestrator.shared.updatePurposeConsents("BOxxx...")  // bypass only if
```

ATT state

```
let attStatus = MobileADVOrchestrator.shared.adTrackingStatus
// .authorized | .denied | .notDetermined | .restricted

let idfa = MobileADVOrchestrator.shared.getAvertisingIdentifier()
```

`PermissionSettings`

```
public struct PermissionSettings {
    public var consentTimestamp: Date
    public var consentString: String
    public var purposeConsents: String
    public var vendorConsents: String
    public var userExpressedPreferences: Bool
    public var userAcceptedGooglePersonalizedAds: Bool
    public var rand: String?
}
```

User profiling

MAOR requires a `UserProfilingLevelRequest`. The level expires over time: register a closure with `onProfileExpired` to renew it on demand.

```
public enum UserProfilingLevelRequest {
    case weak(ageRange: String, sex: String, expireDate: Date, isUnderAge: Bool, p
    case autopromo(email: String)   // serves autopromo only; no bidder requests a
    case disabled                   // profiling disabled
}
```

Example

```
MobileADVOrchestrator.shared.onProfileExpired { setNextProfile in
    // fetch or refresh the profile
    setNextProfile(.weak(
        ageRange: "5",
        sex: "m",
        expireDate: Date().addingTimeInterval(60 * 60),
        isUnderAge: false,
        ppId: "0d144fdd1f2777c08df0536d0f24afbba427e6eee1a4da250389e95c924faa5c.78
    ))
} completion: {
    print("profile updated")
}
```

> **Watch out for retain cycles:** do not strongly capture `MobileADVOrchestrator.shared` inside the closure.

`setNextProfile` must be invoked **exactly once**.

Requesting ads

Signature

```
public func getAds(
    key: String,
    id: String,
    in viewController: UIViewController,
    at position: Int,          // default 1, values must be >= 1
    widgetIndex: Int? = nil,
    completion: @escaping (Result<AnyBanner, MAORError>) -> Void
)
```

`AnyBanner`

```
public enum AnyBanner {
    case banner(BannerAdView)
    case interstitial(InterstitialAd)
    case native(IOLNativeAdView, NativeAd)
    case backup(BackupAdView, BackupAd)
}
```

> The `interstitialTeads` and `backupTeads` cases no longer exist: the Teads SDK has been removed.

Full example (banner + interstitial + brand header + backup)

```
final class BannerExampleViewController: UIViewController {

    private let adUnitId: String
    private let maorKey: String
    private var interstitialAd: InterstitialAd?

    init(adUnitId: String, maorKey: String) {
        self.adUnitId = adUnitId
        self.maorKey = maorKey
        super.init(nibName: nil, bundle: nil)
    }
    required init?(coder: NSCoder) { fatalError() }

    private func fetchBanner(at position: Int) {
        MobileADVOrchestrator.shared.getAds(
            key: self.maorKey,
            id: self.adUnitId,
            in: self,
            at: position
        ) { [weak self] result in
            guard let self else { return }
            DispatchQueue.main.async {
                switch result {
                case .success(let banner):
                    self.render(banner: banner)
                case .failure(let error):
                    self.handle(error: error)
                }
            }
        }
    }

    private func render(banner: AnyBanner) {
        switch banner {
        case .banner(let banner):
            if banner.isBrandHeader {
                self.presentBrandHeader(banner)
            } else {
                self.addBannerToView(banner)
                self.registerEvents(for: banner)
            }
        case .native(let nativeView, _):
            self.addBannerToView(nativeView)
        case .backup(let backup, _):
            self.addBannerToView(backup)
        case .interstitial(let interstitial):
            self.present(interstitial: interstitial)
        }
    }
```

```
    private func addBannerToView(_ banner: UIView) {
        self.view.addSubview(banner)
        banner.translatesAutoresizingMaskIntoConstraints = false
        NSLayoutConstraint.activate([
            banner.leadingAnchor.constraint(equalTo: self.view.leadingAnchor),
            banner.trailingAnchor.constraint(equalTo: self.view.trailingAnchor),
            banner.centerYAnchor.constraint(equalTo: self.view.centerYAnchor)
        ])

        // Dynamic height adjustment (GAM banners with fluid AdSize)
        if let bannerAd = banner as? BannerAdView {
            var sizeHandlers = bannerAd.adSizeHandlers
            sizeHandlers.willChangeAdSizeTo = { size in
                NSLayoutConstraint.activate([
                    banner.heightAnchor.constraint(equalToConstant: size.height)
                ])
            }
            bannerAd.adSizeHandlers = sizeHandlers
        }
    }

    private func registerEvents(for banner: BannerAdView) {
        banner.eventHandlers.onRecordClick              = { _ in print("click") }
        banner.eventHandlers.onRecordImpression         = { _ in print("impression
        banner.eventHandlers.onBannerWillPresentScreen  = { _ in print("will prese
        banner.eventHandlers.onBannerWillDismissScreen  = { _ in print("will dismi
        banner.eventHandlers.onBannerDidDismissScreen   = { _ in print("did dismis
        banner.eventHandlers.onBrandHeaderSet           = { _ in print("brand head
    }

    private func present(interstitial: InterstitialAd) {
        // The interstitial MUST be kept alive: store it in a property
        self.interstitialAd = interstitial

        interstitial.eventHandlers.onDidPresentFullScreen        = { _ in print("p
        interstitial.eventHandlers.onDidDismissFullScreen        = { [weak self] _
        interstitial.eventHandlers.onFailureToPresentFullScreen  = { [weak self] _
            print("failed:", err)
            self?.interstitialAd = nil
        }

        interstitial.present(on: self)
    }

    private func presentBrandHeader(_ banner: BannerAdView) {
        // Embed banner.brandHeaderAdView inside a dedicated VC; do NOT add it to
        let vc = BrandHeaderContainerVC(banner: banner)
        self.navigationController?.pushViewController(vc, animated: true)
    }

    private func handle(error: MAORError) {
        switch error {
```

```
        case .appTrackingTransparencyNotResolved(let backupBanner, _):
            self.addBannerToView(backupBanner)
        default:
            print("MAOR error:", error.localizedDescription)
        }
    }
}
```

Default-position overload

```
MobileADVOrchestrator.shared.getAds(
    key: key,
    id: "top",
    widgetIndex: nil,
    in: self
) { result in
    /* ... */
}
```

is equivalent to `position = 1`.

Notes on banners

- `BannerAdView` is a `UIView` to be inserted directly into your view hierarchy.
- `BannerAdView.brandHeaderAdView` is populated **after** the ad is received. Use `onBrandHeaderSet` or `isBrandHeader` to react.
- `IOLNativeAdView` derives from `GoogleMobileAds.NativeAdView`.
- `BackupAdView` is populated with autopromo/news/mail content depending on the configured `formatADV`.

Prefetch

Preload an ad and keep it until consumption (`.consumable`) or reuse it (`.persistent`).

```
// Preload
MobileADVOrchestrator.shared.prefetchAds(
    key: AppDelegate.maorKey,
    id: "top",
    in: self,
    at: 1,
    widgetIndex: nil,
    type: .consumable        // or .persistent
)

// Later, getAds returns the cached ad without issuing a new request
```

```
MobileADVOrchestrator.shared.getAds(key: AppDelegate.maorKey, id: "top", in: self,

// Cleanup
MobileADVOrchestrator.shared.removePrefetchAds(
    key: AppDelegate.maorKey,
    id: "top",
    at: 1,
    widgetIndex: nil
)
MobileADVOrchestrator.shared.removeAllPrefetchAds()
```

Internal prefetch timeout: **5 s**. Past that, `getAds` falls back to the normal flow.

Interstitial loader

A full-screen loader to display while waiting for an interstitial:

```
let loader = MobileADVOrchestrator.shared.presentInterstitialLoader(
    in: self,
    autoDismiss: true,
    timeoutAction: {
        print("timeout: ad not received in time")
    },
    timeout: 5.0
)

// When the interstitial is ready:
loader.dismiss(animated: false)
interstitial.present(on: self)
```

External identity (LiveRamp / ID5)

Associate an external identifier (e.g. logged-in email) to enrich the bid request.

```
// Set: send email to LiveRamp + ID5
MobileADVOrchestrator.shared.setExternalUserId(email: "user@example.com")

// Local reset
MobileADVOrchestrator.shared.resetExternalUserId()

// Delete the LiveRamp envelope for that email
let deleted = MobileADVOrchestrator.shared.deleteEnvelope(email: "user@example.com
```

`setExternalUserId` automatically invalidates the prefetch cache.

IMA helpers (instream)

Register "friendly" views on top of instream ads:

```
let container = MobileADVOrchestrator.shared.getIMAAdDisplayContainer(
    adContainer: self.view,
    viewController: self
)

MobileADVOrchestrator.shared.registerIMAFriendlyObstructionNotVisibile(
    container: container,
    friendView: overlayView,
    detailedReason: "Hidden overlay loader"
)
```

Error handling

```
public enum MAORError: Error, LocalizedError {
    case couldNotParseADVTokenStrongProfiling
    case adUnitNotPresent(for: AdType)
    case bidderFailure
    case cannotConfigure
    case auctionTimeout
    case profileExpired
    case noAdsSpecified
    case noResponse
    case appTrackingTransparencyNotResolved(backupBanner: BackupAdView, BackupAd)
    case configurationNotCompleted
    case bannerFetchingFailure
    case unknown
    case configNotFound(for: String)
    case adUnitNotPresentById(for: String)
    case noCMP
    case limitedAdsBidderNotnabled
    case id5InvalidUniversalId
}
```

`errorDomain = "it.italiaonline.MAOR.Error"`. A numeric `errorCode` is available for each case.

`appTrackingTransparencyNotResolved` exposes a ready-to-render `BackupAdView` to be used when ATT has not been resolved yet.

```
```

```
case .failure(let error):
    switch error {
    case .appTrackingTransparencyNotResolved(let backupBanner, _):
        self.addBannerToView(backupBanner)
    case .noCMP:
        // re-display the consent page
        MobileADVOrchestrator.shared.showCookiePermissionPage(on: self)
    case .configurationNotCompleted, .configNotFound:
        // init not completed yet: retry later
        break
    default:
        print(error.localizedDescription)
    }
}
```

API reference

Main public types

Type	Description
`MobileADVOrchestrator`	Singleton (`.shared`), SDK entry point
`MobileADVOrchestrator.InitializationOptions`	Initialization options (ATT, CMP, analytics, yieldbird fallback)
`AnyBanner`	Wrapper around the result of `getAds`
`BannerAdView`	`UIView` for banners / brand header
`IOLNativeAdView`	`NativeAdView` for native ads
`BackupAdView`	View for autopromo/backup ads
`InterstitialAd`	Interstitial wrapper with `canPresent(on:)` / `present(on:)`
`AdType`	OptionSet of supported formats
`AdTrackingStatus`	ATT status
`PermissionStatus` / `PermissionSettings`	Iubenda CMP state
`UserProfilingLevelRequest`	Requested profiling level
`MAORError`	Errors
`PrefetchAds.Category`	`.consumable` / `.persistent`

`AnalyticsType` / `AnalyticsAd`	Analytics payload
`LogLevel` / `LogDomain`	Internal logger (optional use)
`Theme`	`.dark` / `.light` / `.unspecified`

`MobileADVOrchestrator` — public methods

```
// Init
public func initialize(fontName:options:isTest:storeID:configUrl:globalConfigUrl:l

// Profiling
@discardableResult
public func onProfileExpired(setNextProfile:completion:) -> MobileADVOrchestrator

// Permissions
public func requestPermissions(key:on:) -> (Bool, Bool)
public func showCookiePermissionPage(on:)
public func acceptCMP() -> Bool
public func clearCMP() -> Bool
public func resetPermission()
public func updatePurposeConsents(_:)
public func getIubendaID() -> String?
public func isConsentGivenForVendor(withId:) -> Bool?
public func isConsentGivenForPurpose(withId:) -> Bool?

// Ads
public func getAds(key:id:in:at:widgetIndex:completion:)
public func getAds(key:id:widgetIndex:in:completion:)         // position = 1
public func getBackupBanner(key:id:format:completion:)

// Prefetch
public func prefetchAds(key:id:in:at:widgetIndex:type:)
public func removePrefetchAds(key:id:at:widgetIndex:)
public func removeAllPrefetchAds()

// Loader
public func presentInterstitialLoader(in:autoDismiss:timeoutAction:timeout:) -> UI

// External identity
public func setExternalUserId(email:)
public func resetExternalUserId()
@discardableResult public func deleteEnvelope(email:) -> Bool

// IMA
public func getIMAAdDisplayContainer(adContainer:viewController:) -> IMAAdDisplayC
public func registerIMAFriendlyObstructionNotVisibile(container:friendView:detaile

// State
public var userExpressedPermission: Bool?
public var iubendaSettings: PermissionSettings?
```

```
public var iubendaStatus: PermissionStatus?
public var adTrackingStatus: AdTrackingStatus
public func getAvertisingIdentifier() -> String
```

Integration checklist

- [ ] Add the SPM package (this distribution repo) to your app target.
- [ ] Fill in `Info.plist`: `GADApplicationIdentifier`, `NSUserTrackingUsageDescription`, `SKAdNetworkItems`, `UIAppFonts`.
- [ ] Add `GoogleService-Info.plist` and call `FirebaseApp.configure()` from `application(_:didFinishLaunchingWithOptions:)`.
- [ ] Register the initial profile with `onProfileExpired` **before** `initialize`.
- [ ] Call `MobileADVOrchestrator.shared.initialize(...)` and store the returned `key`.
- [ ] From a visible view controller, call `requestPermissions(key:on:)` to trigger ATT/CMP.
- [ ] Request ads with `getAds(key:id:in:at:)` and handle all `AnyBanner` cases.
- [ ] Keep a strong reference to `InterstitialAd` while it is on screen.
- [ ] Handle `MAORError.appTrackingTransparencyNotResolved` by rendering the `BackupAdView` it provides.

Italiaonline