# Appstock SDK iOS - Overview

Appstock SDK is a native library that monetizes iOS applications. The latest SDK version is **1.0.1**.

The minimum deployment target is **iOS 12.0**.

Demo applications (Swift, ObjC): https://public-sdk.al-ad.com/ios/appstock-demo/demo-app-1.0.1/demo-app-1.0.1.zip

## Integration and configuration

Follow the integration instructions to add the SDK to your app. Once the SDK is integrated, you can provide configuration options that will help increase your revenue. Keep in mind that the SDK supports basic consent providers according to industry standards.

Appstock SDK supports the following ad formats:

- Banner (HTML or Video)

- Interstitial (HTML and Video)

- Native

The SDK can be integrated directly into your app or via supported Mediation Adapters:

- AppLovin MAX

- GMA SDK (AdMob, GAM)

# Appstock SDK iOS - Integration

Appstock SDK is available for integration via CocoaPods dependency manager and direct download of the compiled framework.

## Cocoapods

We assume the CocoaPods dependency manager has already been integrated into the project. If not, follow the "Get Started" instructions on cocoapods.org.

Add this line into your Podfile within the application target:

```
pod 'AppstockSDK', '1.0.1'
```

Then run `pod install --repo-update`.

## Direct download

The Appstock SDK is also available via a direct download link: https://public-sdk.al-ad.com/ios/appstock-sdk/1.0.1/AppstockSDK.xcframework.zip

## SDK Initialization

Import the Appstock SDK core class in the main application class:

```
import AppstockSDK
```

Initialize Appstock SDK in the `application:didFinishLaunchingWithOptions` method by calling `Appstock.initializeSDK()` method.

*Swift*

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?)
    // Initialize SDK SDK.
    Appstock.initializeSDK(with: PARTNER_KEY)
}
```

*Objective-C*

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary<UIApplicationLaunchOptionsKey, id> *)lau
    // Initialize SDK SDK.
    [Appstock initializeSDKWithPartnerKey:PARTNER_KEY];
    return YES;
}
```

The `Appstock.initializeSdk()` method has a parameter:

- **partnerKey** - determine the Appstock server URL. The Appstock account manager should provide you with this key.

It is recommended that contextual information be provided after initialization to enrich the ad requests. For this purpose, use SDK parametrization properties.

Once SDK is initialized and all needed parameters are provided, it is ready to request the ads.

# Appstock SDK iOS - Banner

To load a banner ad, create a `AppstockAdView` object, configure it, add it to the view hierarchy, and call its `loadAd()` method.

*Swift*

```swift
private var adView: AppstockAdView!

private func loadAd() {
    // 1. Create a AppstockAdView
    adView = AppstockAdView(
        frame: CGRect(origin: .zero, size: CGSize(width: 300, height: 250))
    )

    // 2. Configure the AppstockAdView
    adView.placementID = placementID
    adView.delegate = self

    // Add Appstock ad view to the app UI
    containerAdView.addSubview(adView)

    // 3. Load the ad
    adView.loadAd()
}
```

*Objective-C*

```objc
@property (nonatomic) AppstockAdView * adView;

- (void)loadAd {
    // 1. Create a AppstockAdView
    self.adView = [[AppstockAdView alloc] initWithFrame:CGRectMake(0, 0, 300, 250)];

    // 2. Configure the AppstockAdView
    self.adView.placementID = self.placementID;
    self.adView.delegate = self;

    // Add Appstock ad view to the app UI
    [self.containerAdView addSubview:self.adView];

    // 3. Load the ad
    [self.adView loadAd];
}
```

The `AppstockAdView` should be provided with one of the required configuration properties:

- **placementID** - unique placement identifier generated on the Appstock platform's UI;

- **endpointID** - unique endpoint identifier generated on the Appstock platform's UI.

Which one to use depends on your type of Appstock account.

You should also provide `CGRect` value for ad view to initialize `UIView`.

If you need to integrate video ads, you can also use the `AppstockAdView` object in the same way as for banner ads. The single required change is you should explicitly set the ad format via the respective property:

*Swift*

```swift
adView.adFormat = .video
```

*Objective-C*

```objc
self.adView.adFormat = AppstockAdFormat.video;
```

Once it is done, the TeqBlzae SDK will make ad requests for video placement and render the respective creatives.

You can optionally subscribe to the ad's lifecycle events by implementing the `AppstockAdViewDelegate` protocol:

*Swift*

```swift
extension BannerAdViewController: AppstockAdViewDelegate {

    func adViewPresentationController() -> UIViewController? {
        // View controller used by SDK for presenting modal.
        // Usual implementation may simply return self,
        // if it is view controller class.
        self
    }

    func adView(_ adView: AppstockAdView, didFailToReceiveAdWith error: any Error)
        // Called when SDK failed to load ad
        print("Did fail to receive ad with error: \(error.localizedDescription)")
    }

    func adView(_ adView: AppstockAdView, didReceiveAdWithAdSize adSize: CGSize) {
        // Called when ad is loaded
    }

    func adViewWillPresentModal(_ adView: AppstockAdView) {
        // Called when modal is about to be presented
    }

    func adViewDidDismissModal(_ adView: AppstockAdView) {
        // Called when modal is dismissed
    }

    func adViewWillLeaveApplication(_ adView: AppstockAdView) {
        // Called when the application is about to enter the background
    }
}
```

*Objective-C*

```
@interface AppstockBannerAdViewController : UIViewController <AppstockAdViewDelegat

@end

// ...

- (UIViewController *)adViewPresentationController {
    // View controller used by SDK for presenting modal.
    // Usual implementation may simply return self,
    // if it is view controller class.
    return self;
}

- (void)adView:(AppstockAdView *)adView didFailToReceiveAdWith:
(NSError *)error {
    // Called when Appstock SDK failed to load ad
    NSLog(@"Did fail to receive ad with error: %@", error.localizedDescription);
}

- (void)adView:(AppstockAdView *)adView
didReceiveAdWithAdSize:(CGSize)adSize {
    // Called when ad is loaded
}

- (void)adViewWillPresentModal:(AppstockAdView *)adView {
    // Called when modal is about to be presented
}

- (void)adViewDidDismissModal:(AppstockAdView *)adView {
    // Called when modal is dismissed
}

- (void)adViewWillLeaveApplication:(AppstockAdView *)adView {
    // Called when the application is about to enter the background
}
```

The `refreshInterval` property controls the frequency of automatic ad refreshes. This interval is set in seconds and dictates how often a new ad request is made after the current ad is displayed.

*Swift*

```
adView.refreshInterval = 30.0
```

*Objective-C*

```
adView.refreshInterval = 30.0;
```

You can stop auto refresh by calling respective method:

*Swift*

```
adView.stopAutoRefresh()
```

*Objective-C*

```
[adView stopAutoRefresh];
```

You can also set `adPosition` property to specify the position of the ad on the screen and

corresponding value will be sent in `bidRequest.imp[].banner.pos` ORTB field during bid request.

*Swift*

```
adView.adPosition = .footer
```

*Objective-C*

```
adView.adPostion = AppstockAdPositionFooter;
```

# Appstock SDK iOS - Interstitial

To load interstitial ads, you should create and configure the `AppstockInterstitialAdUnit` and call its `loadAd()` method.

*Swift*

```swift
private var interstitialAdUnit: AppstockInterstitialAdUnit!

private func loadAd() {
    // 1. Create a AppstockInterstitialAdUnit
    interstitialAdUnit = AppstockInterstitialAdUnit()

    // 2. Configure the AppstockInterstitialAdUnit
    interstitialAdUnit.placementID = placementID
    interstitialAdUnit.delegate = self

    // 3. Load the interstitial ad
    interstitialAdUnit.loadAd()
}
```

*Objective-C*

```objectivec
@property (nonatomic) AppstockInterstitialAdUnit * interstitialAdUnit;

- (void)loadAd {
    // 1. Create a AppstockInterstitialAdUnit
    self.interstitialAdUnit = [[AppstockInterstitialAdUnit alloc] init];

    // 2. Configure the AppstockInterstitialAdUnit
    self.interstitialAdUnit.placementID = self.placementID;
    self.interstitialAdUnit.delegate = self;

    // 3. Load the interstitial ad
    [self.interstitialAdUnit loadAd];
}
```

If you need to integrate **video** ads or **multiformat** ads, you should set the adFormats property to the respective value:

*Swift*

```
// Make ad request for video ad
interstitialAdUnit.adFormats = [.video]

// Make ad request for both video and banner ads
interstitialAdUnit.adFormats = [.video, .banner]

// Make ad request for banner ad (default behaviour)
interstitialAdUnit.adFormats = [.banner]
```

*Objective-C*

```
// Make ad request for video ad
interstitialAdUnit.adFormats = [NSSet setWithArray:@[AppstockAdFormat.video]];

// Make ad request for both video and banner ads
interstitialAdUnit.adFormats = [NSSet setWithArray:@[AppstockAdFormat.video, Appstoc

// Make ad request for banner ad (default behaviour)
interstitialAdUnit.adFormats = [NSSet setWithArray:@[AppstockAdFormat.banner]];
```

You can check if the ad is ready to be shown by calling respective property:

*Swift*

```
if interstitialAdUnit.isReady {
    // Show the ad...
}
```

*Objective-C*

```
if (interstitialAdUnit.isReady) {

}
```

Once the ad is loaded, you can invoke the `show()` method at any appropriate point of the app flow to present the fullscreen ad. To know when the ad is loaded, you should implement `AppstockInterstitialAdUnitDelegate` protocol and subscribe to the ad events in its methods.

When the delegate's method `interstitialDidReceiveAd` is called, it means that the SDK has successfully loaded the ad. Starting from this point, you can call the `show()` method to display the full-screen ad.

*Swift*

```
extension AppstockBannerInterstitialViewController:
AppstockInterstitialAdUnitDelegate {

    func interstitialDidReceiveAd(_ interstitial: AppstockInterstitialAdUnit) {
        // Called when ad is loaded

        // Show the full screen ad
        if interstitialAdUnit.isReady {
            interstitial.show(from: self)
        }
    }

    func interstitial(
        _ interstitial: AppstockInterstitialAdUnit,
        didFailToReceiveAdWithError error: (any Error)?
    ) {
        // Called when Appstock SDK failed to load ad
        print("Did fail to receive ad with error:
        \(String(describing: error?.localizedDescription))")
    }

    func interstitialWillPresentAd(_ interstitial: AppstockInterstitialAdUnit) {
        // Called when interstitial is about to be presented
    }

    func interstitialDidDismissAd(_ interstitial: AppstockInterstitialAdUnit)
    {
        // Called when interstitial is dismissed
    }

    func interstitialDidClickAd(_ interstitial: AppstockInterstitialAdUnit) {
        // Called when interstitial was clicked
    }

    func interstitialWillLeaveApplication(_ interstitial:
    AppstockInterstitialAdUnit) {
        // Called when the application is about to enter the background
    }
}
```

*Objective-C*

```
@interface AppstockBannerInterstitialViewController : UIViewController <AppstockInte

@end

// ...

- (void)interstitial:(AppstockInterstitialAdUnit *)interstitial didFailToReceiveAdW:
    // Called when Appstock SDK failed to load ad
    NSLog(@"Did fail to receive ad with error: %@", error.localizedDescription);
}

- (void)interstitialDidReceiveAd:(AppstockInterstitialAdUnit *)interstitial {
    // Called when ad is loaded
    [interstitial showFrom:self];
}

- (void)interstitialWillPresentAd:(AppstockInterstitialAdUnit *)interstitial {
    // Called when interstitial is about to be presented
}

- (void)interstitialDidDismissAd:(AppstockInterstitialAdUnit *)interstitial {
    // Called when interstitial is dismissed
}

- (void)interstitialDidClickAd:(AppstockInterstitialAdUnit *)interstitial {
    // Called when interstitial was clicked
}

- (void)interstitialWillLeaveApplication:(AppstockInterstitialAdUnit *)interstitial
    // Called when the application is about to enter the background
}
```

## Rendering Controls

The following properties enable rendering customization of video interstitial ads.

| Property | Description |
|---|---|
| isMuted | This option lets you switch the sound on or off during playback. Default is `false` . |
| closeButtonArea | This setting determines the percentage of the device screen that the close button should cover. Allowed range - `0...1` . Default value is `0.1` . |
| closeButtonPosition | This setting controls where the close button appears on the screen. Allowed values: `topLeft` , `topRight` . Other values will be ignored. Default is `topRight` . |
| skipButtonArea | This setting determines the percentage of the device screen that the skip button should cover. Allowed range - `0...1` . Default value is `0.1` . |
| skipButtonPosition | This control sets the position of the skip button. Allowed values: `topLeft` , `topRight` . Other values will be ignored. Default is `topLeft` . |
| skipDelay | This setting determines the number of seconds after the start of playback before the skip or close button should appear. Default value is `10.0` . |

| Property | Description |
|---|---|
| isSoundButtonVisible | This option switches on or off the visibility of the sound/mute button for users. Default value is `false` . |

Usage example:

*Swift*

```
interstitialAdUnit.isMuted = true
interstitialAdUnit.closeButtonArea = 0.2
interstitialAdUnit.closeButtonPosition = .topRight
interstitialAdUnit.skipButtonArea = 0.2
interstitialAdUnit.skipButtonPosition = .topLeft
interstitialAdUnit.skipDelay = 15.0
interstitialAdUnit.isSoundButtonVisible = true
```

*Objective-C*

```
interstitialAdUnit.isMuted = YES;
interstitialAdUnit.closeButtonArea = 0.2;
interstitialAdUnit.closeButtonPosition = AppstockPositionTopRight;
interstitialAdUnit.skipButtonArea = 0.2;
interstitialAdUnit.skipButtonPosition = AppstockPositionTopLeft;
interstitialAdUnit.skipDelay = 15.0;
interstitialAdUnit.isSoundButtonVisible = YES;
```

# Appstock SDK iOS - Native

To load a native ad, you should initialize and configure `AppstockNativeAdUnit` object and call the `loadAd()` method.

*Swift*

```
private var nativeAdUnit: AppstockNativeAdUnit!
private var nativeAd: AppstockNativeAd?

private func loadAd() {
    // 1. Create a AppstockNativeAdUnit
    nativeAdUnit = AppstockNativeAdUnit()

    // 2. Configure the AppstockNativeAdUnit
    nativeAdUnit.placementID = placementID
    let image = AppstockNativeAssetImage(minimumWidth: 200,
    minimumHeight: 50, required: true)
    image.type = .Main

    let icon = AppstockNativeAssetImage(minimumWidth: 20,
    minimumHeight: 20, required: true)
    icon.type = .Icon

    let title = AppstockNativeAssetTitle(length: 90, required: true)
    let body = AppstockNativeAssetData(type: .description, required: true)
    let cta = AppstockNativeAssetData(type: .ctatext, required: true)
    let sponsored = AppstockNativeAssetData(type: .sponsored, required: true)

    let parameters = AppstockNativeParameters()
    parameters.assets = [title, icon, image, sponsored, body, cta]
```

```
    let eventTracker = AppstockNativeEventTracker(
        event: .Impression,
        methods: [.Image, .js]
    )

    parameters.eventtrackers = [eventTracker]
    parameters.context = .Social
    parameters.placementType = .FeedContent
    parameters.contextSubType = .Social

    nativeAdUnit.parameters = parameters

    nativeAdUnit.loadAd { [weak self] ad, error in
        guard let self = self else {
            return
        }

        guard let ad = ad, error == nil else {
            return
        }

        self.nativeAd = ad
        self.nativeAd?.delegate = self

        // 3. Render the native ad
        self.titleLabel.text = ad.title
        self.bodyLabel.text = ad.text
        self.sponsoredLabel.text = ad.sponsoredBy

        self.mainImageView.setImage(from: ad.imageUrl, p
        laceholder: UIImage(systemName: "photo.artframe"))
        self.iconView.setImage(from: ad.iconUrl,
        placeholder: UIImage(systemName: "photo.artframe"))

        self.callToActionButton.setTitle(ad.callToAction, for: .normal)

        self.nativeAd?.registerView(view: self.view,
        clickableViews: [self.callToActionButton])
    }
}
```

*Objective-C*

```objc
@property (nonatomic) AppstockNativeAdUnit * nativeAdUnit;
@property (nonatomic, nullable) AppstockNativeAd * nativeAd;

- (void)loadAd {
    // 1. Create a AppstockNativeAdUnit
    self.nativeAdUnit = [[AppstockNativeAdUnit alloc] init];

    // 2. Configure the AppstockNativeAdUnit
    self.nativeAdUnit.placementID = self.placementID;

    AppstockNativeAssetImage *image = [
        [AppstockNativeAssetImage alloc]
        initWithMinimumWidth:200
        minimumHeight:200
        required:true
    ];

    image.type = AppstockImageAsset.Main;
```

```objc
AppstockNativeAssetImage *icon = [
    [AppstockNativeAssetImage alloc]
    initWithMinimumWidth:20
    minimumHeight:20
    required:true
];

icon.type = AppstockImageAsset.Icon;

AppstockNativeAssetTitle *title = [
    [AppstockNativeAssetTitle alloc]
    initWithLength:90
    required:true
];

AppstockNativeAssetData *body = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetDescription
    required:true
];

AppstockNativeAssetData *cta = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetCtatext
    required:true
];

AppstockNativeAssetData *sponsored = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetSponsored
    required:true
];

AppstockNativeParameters * parameters = [AppstockNativeParameters new];
parameters.assets = @[title, icon, image, sponsored, body, cta];

AppstockNativeEventTracker * eventTracker = [
    [AppstockNativeEventTracker alloc]
    initWithEvent:AppstockEventType.Impression
    methods:@[AppstockEventTracking.Image, AppstockEventTracking.js]
];

parameters.eventtrackers = @[eventTracker];
parameters.context = AppstockContextType.Social;
parameters.placementType = AppstockPlacementType.FeedContent;
parameters.contextSubType = AppstockContextSubType.Social;

self.nativeAdUnit.parameters = parameters;

__weak typeof(self) weakSelf = self;
[self.nativeAdUnit loadAdWithCompletion:^(AppstockNativeAd * _Nullable ad, NSErr
    if (error != nil || ad == nil) {
        return;
    }

    weakSelf.nativeAd = ad;
    weakSelf.nativeAd.delegate = self;

    weakSelf.titleLabel.text = ad.title;
    weakSelf.bodyLabel.text = ad.text;
    weakSelf.sponsoredLabel.text = ad.sponsoredBy;
```

```
        [weakSelf.mainImageView setImageFromURLString:ad.imageUrl
         placeholder:[UIImage systemImageNamed:@"photo.artframe"]];
        [weakSelf.iconView setImageFromURLString:ad.iconUrl
        placeholder:[UIImage systemImageNamed:@"photo.artframe"]];
        [weakSelf.callToActionButton
        setTitle:ad.callToAction  forState:UIControlStateNormal];
    }];
}
```

You can configure the native assets by setting up `parameters` property. Here is a brief description of
`AppstockNativeParameters` :

- `assets` - an array of assets associated with the native ad.

- `eventtrackers` - an array of event trackers used for tracking native ad events.

- `version` - the version of the native parameters. Default is `"1.2"` .

- `context` - the context type for the native ad (e.g., content, social).

- `contextSubType` - a more detailed context in which the ad appears.

- `placementType` - the design/format/layout of the ad unit being offered.

- `placementCount` - the number of identical placements in this layout. Default is `1` .

- `sequence` - the sequence number of the ad in a series. Default is `0` .

- `asseturlsupport` - whether the supply source / impression impression supports returning an
  assetsurl instead of an asset object. Default is `0` (unsupported).

- `durlsupport` - whether the supply source / impression supports returning a dco url instead of an
  asset object. Default is `0` (unsupported).

- `privacy` - set to 1 when the native ad support buyer-specific privacy notice. Default is `0` .

- `ext` - a dictionary to hold any additional data as key-value pairs.

Here is a brief description of available assets:

| Class/Enum | Type | Name | Description |
|---|---|---|---|
| `AppstockNativeAssetTitle` | Class | | A subclass representing a title asset in a native ad. |
| | Property | `ext` | An optional extension for additional data. |
| | Property | `length` | The length of the title. |
| `AppstockNativeAssetImage` | Class | | A subclass representing an image asset in a native ad. |
| | Property | `type` | The type of image asset (e.g., icon, main image). |
| | Property | `width` | The width of the image. |

| Class/Enum | Type | Name | Description |
|---|---|---|---|
| | Property | `widthMin` | The minimum width of the image. |
| | Property | `height` | The height of the image. |
| | Property | `heightMin` | The minimum height of the image. |
| | Property | `mimes` | An array of supported MIME types for the image. |
| | Property | `ext` | An optional extension for additional data. |
| `AppstockNativeAssetData` | Class | | A subclass representing a data asset in a native ad. |
| | Property | `length` | The length of the data string. |
| | Property | `ext` | An optional extension for additional data. |
| | Property | `type` | The type of data asset (e.g., sponsored, description). |
| `AppstockImageAsset` | Class | | A class representing different types of image assets in the Appstock SDK. |
| | Property | `Icon` | Represents an icon image asset. |
| | Property | `Main` | Represents a main image asset. |
| | Property | `Custom` | Represents a custom image asset. |
| `AppstockDataAsset` | Enum | | An enum representing different types of data assets in the Appstock SDK. |
| | Case | `sponsored` | Represents sponsored content. |
| | Case | `description` | Represents a description. |
| | Case | `rating` | Represents a rating. |
| | Case | `likes` | Represents likes. |
| | Case | `downloads` | Represents download count. |
| | Case | `price` | Represents the price. |

| Class/Enum | Type | Name | Description |
| --- | --- | --- | --- |
| | Case | `saleprice` | Represents a sale price. |
| | Case | `phone` | Represents a phone number. |
| | Case | `address` | Represents an address. |
| | Case | `description2` | Represents a secondary description. |
| | Case | `displayurl` | Represents a display URL. |
| | Case | `ctatext` | Represents call-to-action text. |
| | Case | `Custom` | Represents a custom data asset. |
| | Method | `exchangeID` | Returns or sets the exchange ID for the `Custom` case. |

You can also specify what type of event tracking is supported. For that you need to set `eventtrackers` property. Here is a brief description of available types:

| Class/Enum | Type | Name | Description |
| --- | --- | --- | --- |
| `AppstockNativeEventTracker` | Class | | A class representing event trackers for native ads. |
| | Property | `event` | The type of event to be tracked (e.g., impression, viewable impression). |
| | Property | `methods` | An array of tracking methods used for the event. |
| | Property | `ext` | An optional extension for additional data. |

| Class/Enum | Type | Name | Description |
|---|---|---|---|
| `AppstockEventType` | Class | | A class representing different event types that can be tracked. |
| | Property | `Impression` | Represents an impression event. |
| | Property | `ViewableImpression50` | Represents a 50% viewable impression event. |
| | Property | `ViewableImpression100` | Represents a 100% viewable impression event. |
| | Property | `ViewableVideoImpression50` | Represents a 50% viewable video impression event. |
| | Property | `Custom` | Represents a custom event type. |
| `AppstockEventTracking` | Class | | A class representing different methods of event tracking. |
| | Property | `Image` | Represents image-based event tracking. |
| | Property | `js` | Represents JavaScript-based event tracking. |

| Class/Enum | Type | Name | Description |
|---|---|---|---|
|  | Property | `Custom` | Represents a custom tracking method. |

Once the ad is loaded, the SDK provides you with a `AppstockNativeAd` object in the callback of the `loadAd()` method. This object contains ad assets that you should apply to the native ad layout.

If you need to manage stages of the ad lifecycle you should implement the `AppstockNativeAdDelegate` protocol.

*Swift*

```swift
extension AppstockNativeViewController: AppstockNativeAdDelegate {

    func adDidExpire(ad: AppstockNativeAd) {
        // Called when the ad expired
    }

    func adWasClicked(ad: AppstockNativeAd) {
        // Called when the ad was clicked
    }

    func adDidLogImpression(ad: AppstockNativeAd) {
        // Called when the impression was logged
    }
}
```

*Objective-C*

```objc
@interface AppstockNativeViewController : UIViewController<AppstockNativeAdDelegate>

@end

// ...

- (void)adDidExpireWithAd:(AppstockNativeAd *)ad {
    // Called when the ad expired
}

- (void)adWasClickedWithAd:(AppstockNativeAd *)ad {
    // Called when the ad was clicked
}

- (void)adDidLogImpressionWithAd:(AppstockNativeAd *)ad {
    // Called when the impression was logged
}
```

If you need ORTB native request object, you can use `getNativeRequestObject` method for that. It returns a dictionary with request configuration.

*Swift*

```swift
let request = adUnit.getNativeRequestObject()
```

*Objective-C*

```
NSDictionary * request = [self.nativeAdUnit getNativeRequestObject];
```

# Appstock SDK iOS - SDK Parametrization

## Configuration via `AppstockTargeting` class

The `AppstockTargeting` class provided a set of properties that allow to enrich the ad request.

| Method | Description | OpenRTB field |
| --- | --- | --- |
| AppstockTargeting.userExt | Placeholder for exchange-specific extensions to OpenRTB. | user.ext |
| AppstockTargeting.userCustomData | Set the specific user data | user.customdata |
| AppstockTargeting.subjectToCOPPA | Integer flag indicating if this request is subject to the COPPA regulations established by the USA FTC, where 0 = no, 1 = yes | regs.coppa |
| AppstockTargeting.storeURL | App store URL for an installed app. | app.storeurl |
| AppstockTargeting.sourceapp | ID of publisher app in Apple's App Store. | imp[].ext.skadn.sourceapp |
| AppstockTargeting.publisherName | App's publisher name | app.publisher.name |
| AppstockTargeting.itunesID | The app identifier in iTunes. | app.bundle |
| AppstockTargeting.eids | Placeholder for User Identity Links | usr.ext.eids |
| AppstockTargeting.externalUserIds | Defines the User Id Object from an External Thrid Party Source. | usr.ext.eids |
| AppstockTargeting.domain | Domain of the app (e.g., "mygame.foo.com"). | app.domain |
| AppstockTargeting.coordinate | Location of the user's home base. This is not necessarily their current location | user.geo.lat, user.geo.lon |
| AppstockTargeting.addAppKeyword | Comma-separated list of keywords about the app | app.keywords |

Usage examples:

*Swift*

```
// Set the userExt property
AppstockTargeting.shared.userExt = ["customField": "value"]

// Set the userCustomData property
AppstockTargeting.shared.userCustomData = "{\"key\":\"value\"}"

// Set the subjectToCOPPA property
AppstockTargeting.shared.subjectToCOPPA = true

// Set the storeURL property
AppstockTargeting.shared.storeURL = "https://appstore.com/app"

// Set the sourceapp property
AppstockTargeting.shared.sourceapp = "123456789"

// Set the publisherName property
AppstockTargeting.shared.publisherName = "MyPublisher"

// Set the itunesID property
AppstockTargeting.shared.itunesID = "com.example.app"

// Set the eids property
AppstockTargeting.shared.eids = [["uids":["id": "123"], "source": "idfa"]]

// Set the externalUserIds property
AppstockTargeting.shared.externalUserIds =
[AppstockExternalUserId(source: "adserver.org", identifier:
"111111111111", ext: ["partner" : "abs"])]

// Set the domain property
AppstockTargeting.shared.domain = "mygame.foo.com"

// Set the coordinate property
AppstockTargeting.shared.coordinate = NSValue(cgCoordinate:
CLLocationCoordinate2D(latitude: 37.7749, longitude: -122.4194))

// Add a keyword
AppstockTargeting.shared.addAppKeyword("gaming")
```

*Objective-C*

```
// Set the userExt property
AppstockTargeting.shared.userExt = @{@"customField": @"value"};

// Set the userCustomData property
AppstockTargeting.shared.userCustomData = @"{\"key\":\"value\"}";

// Set the subjectToCOPPA property
AppstockTargeting.shared.subjectToCOPPAObjc = @1;

// Set the storeURL property
AppstockTargeting.shared.storeURL = @"https://appstore.com/app";

// Set the sourceapp property
AppstockTargeting.shared.sourceapp = @"123456789";

// Set the publisherName property
AppstockTargeting.shared.publisherName = @"MyPublisher";

// Set the itunesID property
AppstockTargeting.shared.itunesID = @"com.example.app";

// Set the eids property
AppstockTargeting.shared.eids = @[@{@"uids": @{@"id": @"123"}, @"source": @"idfa"}],

// Set the externalUserIds property
AppstockTargeting.shared.externalUserIds = @[[[AppstockExternalUserId
alloc] initWithSource:@"adserver.org" identifier:@"111111111111" atype:@1
ext:@{@"partner": @"abs"}]];

// Set the domain property
AppstockTargeting.shared.domain = @"mygame.foo.com";

// Set the coordinate property
AppstockTargeting.shared.coordinate = [NSValue
valueWithMKCoordinate:CLLocationCoordinate2DMake(37.7749, -122.4194)];

// Add a keyword
[AppstockTargeting.shared addAppKeyword:@"gaming"];
```

## Configuration via `Appstock` class

`Appstock` class provides some properties to configure SDK and ad request. Here is a brief overview:

| Property/Method | Description |
| --- | --- |
| `timeoutUpdated` | Indicates whether the ad request timeout has been updated. |
| `debugRequests` | Enables or disables debug mode for requests. |
| `endpointID` | A unique identifier generated on the platform's UI. |
| `shouldAssignNativeAssetID` | Determines whether the asset ID for native ads should be manually assigned. |
| `shareGeoLocation` | Controls whether location data is shared for better ad targeting. |
| `logLevel` | Sets the desired verbosity of the logs. |

| Property/Method | Description |
| --- | --- |
| `externalUserIdArray` | An array containing objects that hold external user ID parameters. |
| `version` | Returns the SDK version. |
| `omsdkVersion` | Returns the OM SDK version used by the SDK. |
| `timeoutMillis` | The timeout in milliseconds for ad requests. |
| `timeoutMillisDynamic` | The dynamic timeout value set when `timeoutMillis` changes. |
| `adRequestTimeout` | The time interval allowed for a creative to load before it is considered a failure. |
| `adRequestTimeoutPreRenderContent` | The time interval allowed for video and interstitial creatives to load. |
| `initializeSDK(with partnerKey)` | Initializes the Appstock SDK with the provided partner key. |

Usage examples:

*Swift*

```
// Setting the timeoutUpdated flag
Appstock.shared.timeoutUpdated = true

// Enabling debug logging
Appstock.shared.debugRequests = true

// Setting the endpoint ID
Appstock.shared.endpointID = "12345"

// Managing the asset ID for native ads
Appstock.shared.shouldAssignNativeAssetID = true

// Sharing location data for targeted ads
Appstock.shared.shareGeoLocation = true

// Setting the log level to debug
Appstock.shared.logLevel = .debug

// Adding an external user ID
Appstock.shared.externalUserIdArray = [AppstockExternalUserId(
source: "adserver.org", identifier: "111111111111",
ext: ["partner" : "abs"])]

// Accessing the SDK version
let sdkVersion = Appstock.shared.version

// Accessing the OM SDK version
let omVersion = Appstock.shared.omsdkVersion

// Setting the timeout for ad requests
Appstock.shared.timeoutMillis = 5000

// Adjusting the creative load time
Appstock.shared.adRequestTimeout = 8.0

// Adjusting the pre-rendered content load time
Appstock.shared.adRequestTimeoutPreRenderContent = 20.0

// Initializing the SDK
Appstock.initializeSDK(with: "partner-key")
```

*Objective-C*

```
// Setting the timeoutUpdated flag
Appstock.shared.timeoutUpdated = YES;

// Enabling debug logging
Appstock.shared.debugRequests = YES;

// Setting the endpoint ID
Appstock.shared.endpointID = @"12345";

// Managing the asset ID for native ads
Appstock.shared.shouldAssignNativeAssetID = YES;

// Sharing location data for targeted ads
Appstock.shared.shareGeoLocation = YES;

// Setting the log level to debug
Appstock.shared.logLevel = APSLogLevel.debug;

// Adding an external user ID
Appstock.shared.externalUserIdArray = @[[[AppstockExternalUserId alloc]
initWithSource:@"adserver.org" identifier:@"111111111111" atype:@1
ext:@{@"partner": @"abs"}]];

// Accessing the SDK version
NSString *sdkVersion = Appstock.shared.version;

// Accessing the OM SDK version
NSString *omVersion = Appstock.shared.omsdkVersion;

// Setting the timeout for ad requests
Appstock.shared.timeoutMillis = 5000;

// Adjusting the creative load time
Appstock.shared.adRequestTimeout = 8.0;

// Adjusting the pre-rendered content load time
Appstock.shared.adRequestTimeoutPreRenderContent = 20.0;

// Initializing the SDK
[Appstock initializeSDKWith:@"partner-key"];
```

## Appstock SDK iOS - Consent Management

Appstock SDK reads consent data provided by CMPs from User Settings and sends it in the ad request.
You shouldn't do anything except to be sure that the CMP SDKs write data into particular place in the user
storage defined by the IAB standards. The following table describes which data is used by SDK and how
exactly:

| Storage Key | Description | |
|---|---|---|
| [TCF v2](#) | | |

| Storage Key | Description | |
|---|---|---|
| `IABTCF_gdprApplies` | Number:<br>1 GDPR applies in current context<br>0 - GDPR does not apply in current context<br>Unset - undetermined (default before initialization) | `regs.ext.gdpr` |
| `IABTCF_TCString` | String: Full encoded TC string | `user.ext.consent` |
| `IABTCF_PurposeConsents` | Binary String: The '0' or '1' at position n – where n's indexing begins at 0 – indicates the consent status for purpose ID n+1; false and true respectively. eg. '1' at index 0 is consent true for purpose ID 1 | Defines the ability of SDK to collect device info. |
| [CCPA](#) | | |
| `IABUSPrivacy_String` | String: Aligns with IAB OpenRTB CCPA Advisory. The String encodes all choices and information. | `regs.ext.us_privacy` |
| [GPP](#) | | |
| `IABGPP_HDR_GppString` | Full consent string in its encoded form | `regs.gpp` |
| `IABGPP_GppSID` | Section ID(s) considered to be in force. Multiple IDs are separated by underscore, e.g. "2_3" | `regs.gpp_sid` |

# Appstock SDK iOS - Mediation - AdMob

In order to integrate Appstock AdMob Adapter into your app, add the following lines to your Podfile:

```
pod 'AppstockSDK', '1.0.1'
pod 'AppstockGoogleMobileAdsAdapter', '1.0.1'
```

In order to add Appstock to the waterfall, you need to create a custom event in your AdMob account and then add this event to the respective mediation groups.

To create a Appstock custom event, follow the instructions:

1. Sign in to your AdMob account at https://apps.admob.com.

2. Click **Mediation** in the sidebar.
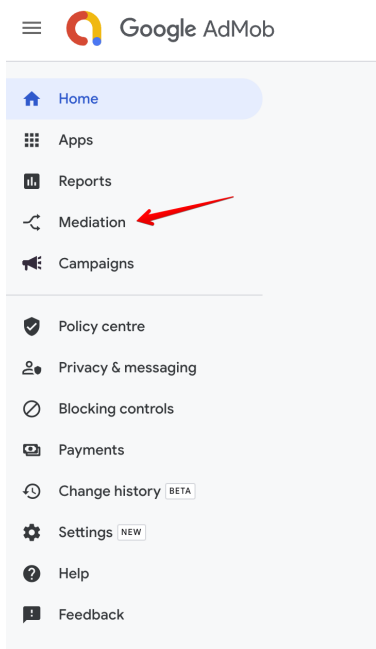
3. Click the **Waterfall** sources tab.
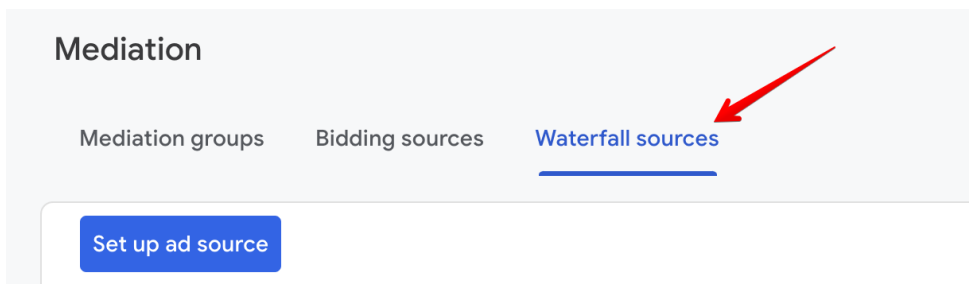


4. Click **Custom Event**.
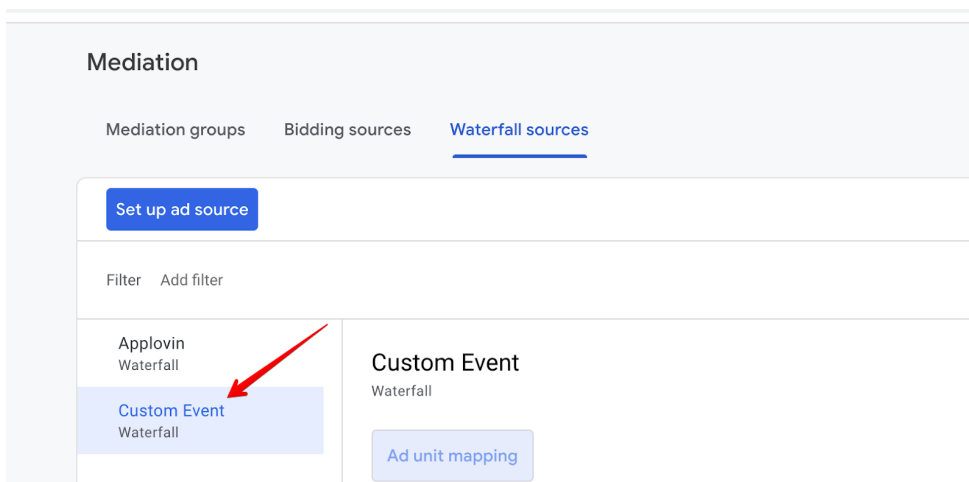


5. Find your app in the list and click on it to expand.

## Custom Event
Waterfall

Ad unit mapping

All visible apps ▾

| Apps ↑ | App ID | |
|---|---|---|
| [image] | View | ▾ |
| [image] | View | ▾ |
| [image] | View | ▾ |
| [image] | View | ▾ |
| [image] | View | ▾ |
| iOS Test App<br>Free | iOS | View | ▾ |

6. Click **Add mapping**.

| iOS Test App<br>Free | iOS | View | ⌃ |
|---|---|---|

No ad units have been mapped yet

Add mapping

7. Click **Add mapping**. To include multiple custom events, you'll need to set up additional mappings.

Edit ad unit mapping

Custom Event
Waterfall

iOS Test App
Free | iOS
ca-app-pub-2844566227051243~1997399649

| AdMob ad unit | Custom Event |
|---|---|
| 🗒 Test Banner<br>Banner | Add mapping |

Show rows  10 ▾   1 - 1 of 1   |< < > >|

Cancel   Save

8. Add the mapping details, including a mapping name. Enter a class name (required) and a parameter (optional) for each ad unit. Typically, the optional parameter contains a JSON that contains IDs (placement ID, endpoint ID) that will be used by the custom event to load ads.

Parameters:

- **placement_id** - unique identifier generated on the platform's UI;

- **endpoint_id** - unique identifier generated on the platform's UI.

```
{
   "placement_id": "4"
}
```



9. Click **Save**.



After you've finished setting up your custom event, you're ready to add it to a mediation group. To add your ad source to an existing mediation group:

1. Sign in to your AdMob account at https://apps.admob.com.

2. Click **Mediation** in the sidebar.
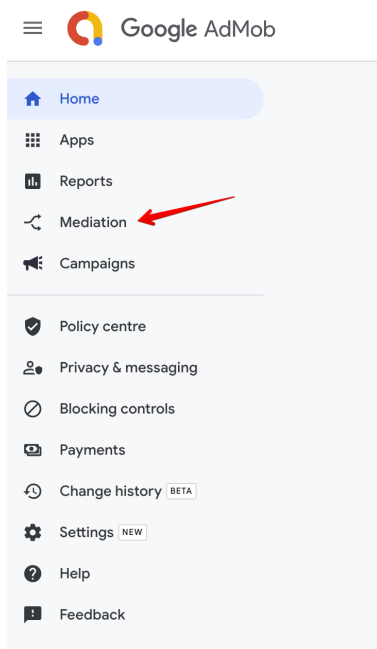
3.  In the **Mediation group** tab, click the name of the mediation group to which you're adding the ad source.



4.  In the Waterfall ad sources table, click **Add custom event**.
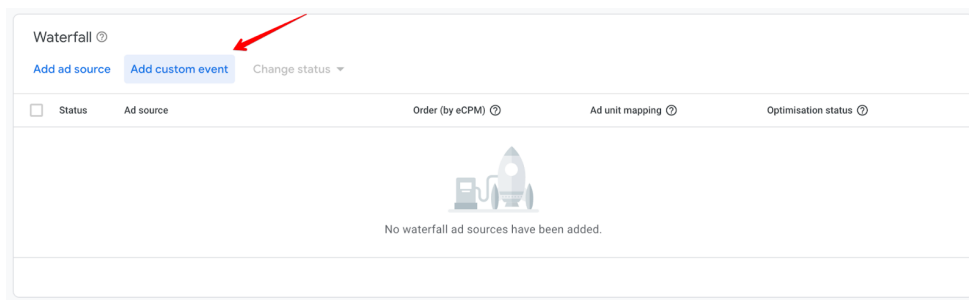
5. Enter a descriptive label for the event. Enter a manual eCPM to use for this custom event. The eCPM will be used to dynamically position the event in the mediation waterfall where it will compete with other ad sources to fill ad requests.



6. Click **Continue**.

# Add custom event

Label ⓘ

TestCustomEvent

15 / 255

Manual eCPM ($ USD) ⓘ

US$ 2

ⓘ    Enter a manual eCPM for this custom event. The eCPM determines the order of the ad source to serve ads.

Cancel    Continue

7. Select an existing mapping to use for this custom event or click Add mapping to set up a new mapping. To use multiple custom events, you'll have to create an additional mapping for each custom event.

## Map ad units: Test Custom Event

Map your ad units to this custom event. ⓘ

| AdMob | Test Custom Event |
|---|---|
| **iOS**   Appstock Internal Test App <br> Free \| iOS | 🔍 Search |
| Test Banner <br> ca-app-pub-2844566227051243/5501759296 | Test Banner <br> Label: Test Banner <br> Class Name: AppstockGADMediationAdapter <br> Parameter: {"placement_id":"4"} |
| | **Add additional mapping** |

8. Click **Done**.

9. Click **Save**. The mediation group will be saved.



# Native Ads

If you integrate native ads, you should pass the native assets through Google Mobile Ads SDK ( `GADAdLoader` ) to the Appstock Adapter using `AppstockGADExtras` class in your app code:

*Swift*

```swift
private func loadAd() {
    // 1. Create a GADAdLoader
    adLoader = GADAdLoader(
        adUnitID: adUnitId,
        rootViewController: self,
        adTypes: [.native],
        options: []
    )

    // 2. Configure the GADAdLoader
    adLoader?.delegate = self

    // 3. Configure the native parameters
    let image = AppstockNativeAssetImage(minimumWidth: 200,
    minimumHeight: 50, required: true)
    image.type = .Main

    let icon = AppstockNativeAssetImage(minimumWidth: 20,
    minimumHeight: 20, required: true)
    icon.type = .Icon

    let title = AppstockNativeAssetTitle(length: 90, required: true)
    let body = AppstockNativeAssetData(type: .description,
    required: true)
    let cta = AppstockNativeAssetData(type: .ctatext, required: true)
    let sponsored = AppstockNativeAssetData(type: .sponsored,
    required: true)

    let parameters = AppstockNativeParameters()
    parameters.assets = [title, icon, image, sponsored, body, cta]

    let eventTracker = AppstockNativeEventTracker(
        event: .Impression,
        methods: [.Image, .js]
    )

    parameters.eventtrackers = [eventTracker]
    parameters.context = .Social
    parameters.placementType = .FeedContent
    parameters.contextSubType = .Social

    // 4. Create a AppstockGADExtras
    let extras = AppstockGADExtras(nativeParameters: parameters)

    // 5. Create a GADRequest
    let request = GADRequest()

    // 6. Register the AppstockGADExtras
    request.register(extras)

    // 7. Load the ad
    adLoader?.load(request)
}
```

*Objective-C*

```objc
- (void)loadAd {
    // 1. Create a GADAdLoader
    self.adLoader = [[GADAdLoader alloc] initWithAdUnitID:self.adUnitId
    rootViewController:self adTypes:@[GADAdLoaderAdTypeNative]
    options:@[]];
```

```objc
// 2. Configure the GADAdLoader
self.adLoader.delegate = self;

// 3. Configure the native parameters
AppstockNativeAssetImage *image = [
    [AppstockNativeAssetImage alloc]
    initWithMinimumWidth:200
    minimumHeight:200
    required:true
];

image.type = AppstockImageAsset.Main;

AppstockNativeAssetImage *icon = [
    [AppstockNativeAssetImage alloc]
    initWithMinimumWidth:20
    minimumHeight:20
    required:true
];

icon.type = AppstockImageAsset.Icon;

AppstockNativeAssetTitle *title = [
    [AppstockNativeAssetTitle alloc]
    initWithLength:90
    required:true
];

AppstockNativeAssetData *body = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetDescription
    required:true
];

AppstockNativeAssetData *cta = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetCtatext
    required:true
];

AppstockNativeAssetData *sponsored = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetSponsored
    required:true
];

AppstockNativeParameters * parameters =
[AppstockNativeParameters new];
parameters.assets = @[title, icon, image, sponsored, body, cta];

AppstockNativeEventTracker * eventTracker = [
    [AppstockNativeEventTracker alloc]
    initWithEvent:AppstockEventType.Impression
    methods:@[AppstockEventTracking.Image, AppstockEventTracking.js]
];

parameters.eventtrackers = @[eventTracker];
parameters.context = AppstockContextType.Social;
parameters.placementType = AppstockPlacementType.FeedContent;
parameters.contextSubType = AppstockContextSubType.Social;

// 4. Create a AppstockGADExtras
```

```
    AppstockGADExtras * extras = [[AppstockGADExtras alloc]
    initWithNativeParameters:parameters];

    // 5. Create a GADRequest
    GADRequest * request = [GADRequest new];

    // 6. Register the AppstockGADExtras
    [request registerAdNetworkExtras:extras];

    // 7. Load the ad
    [self.adLoader loadRequest:request];
}
```

Display the ad as described in [AdMob docs](#):

*Swift*

```
func adLoader(_ adLoader: GADAdLoader, didReceive nativeAd: GADNativeAd) {
    // Set GADNativeAd in GADNativeAdView
    admobNativeView.nativeAd = nativeAd

    // 8. Render the ad
    titleLabel.text = nativeAd.headline
    bodyLabel.text = nativeAd.body

    mainImageView.setImage(
        from: nativeAd.images?.last?.imageURL?.absoluteString,
        placeholder: UIImage(systemName: "photo.artframe")
    )

    iconView.setImage(
        from: nativeAd.icon?.imageURL?.absoluteString,
        placeholder: UIImage(systemName: "photo.artframe")
    )

    callToActionButton.setTitle(nativeAd.callToAction, for: .normal)
    sponsoredLabel.text = nativeAd.advertiser
}
```

*Objective-C*

```
- (void)adLoader:(GADAdLoader *)adLoader didReceiveNativeAd:(GADNativeAd *)nativeAd
    // Set GADNativeAd in GADNativeAdView
    self.admobNativeView.nativeAd = nativeAd;

    self.titleLabel.text = nativeAd.headline;
    self.bodyLabel.text = nativeAd.body;
    self.sponsoredLabel.text = nativeAd.advertiser;

    [self.mainImageView setImageFromURLString:nativeAd.images.lastObject.imageURL.ak
                                  placeholder:[UIImage systemImageNamed:@"photo.
    [self.iconView setImageFromURLString:nativeAd.icon.imageURL.absoluteString
                             placeholder:[UIImage systemImageNamed:@"photo.
    [self.callToActionButton setTitle:nativeAd.callToAction forState:UIControlStateI
}
```

# Appstock SDK iOS - Mediation - AppLovin

In order to integrate Appstock AppLovin MAX Adapter into your app, add following lines to your Podfile:

```
pod 'AppstockSDK', '1.0.1'
pod 'AppstockAppLovinMAXAdapter', '1.0.1'
```

To integrate the Appstock SDK into your AppLovin monetization stack, you should enable a Appstock SDK ad network and add it to the respective ad units.

1. In the MAX Dashboard, select MAX > Mediation > Manage > Networks.
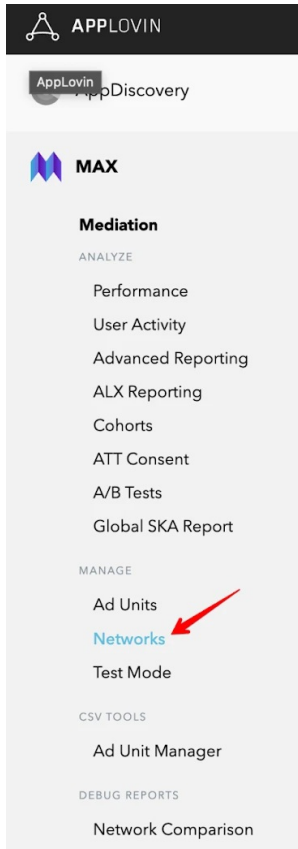


2. Click **Click here to add a Custom Network at the bottom of the page**. The **Create Custom Network** page appears.

3. Add the information about your custom network:

- **Network Type** : Choose **SDK**.

- **Name** : Appstock.

- **iOS Adapter Class Name** : AppstockAppLovinAdapter
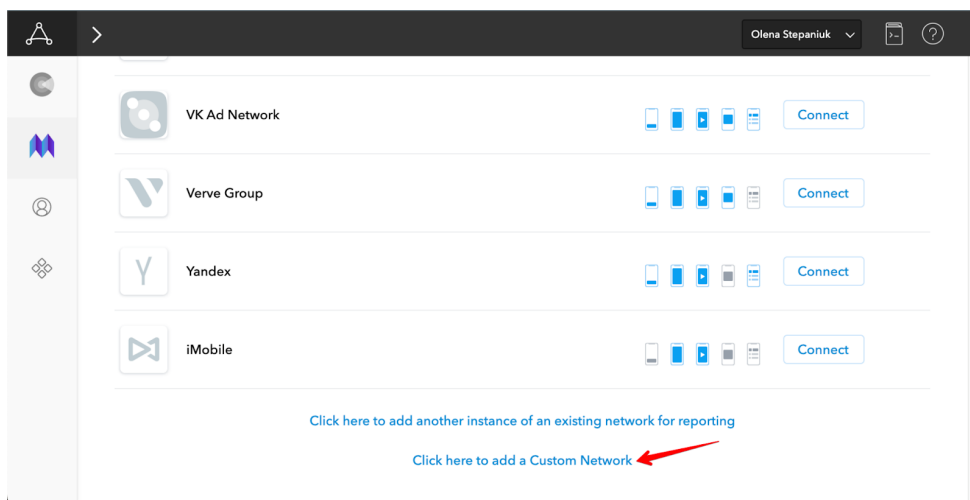
## Manage Network

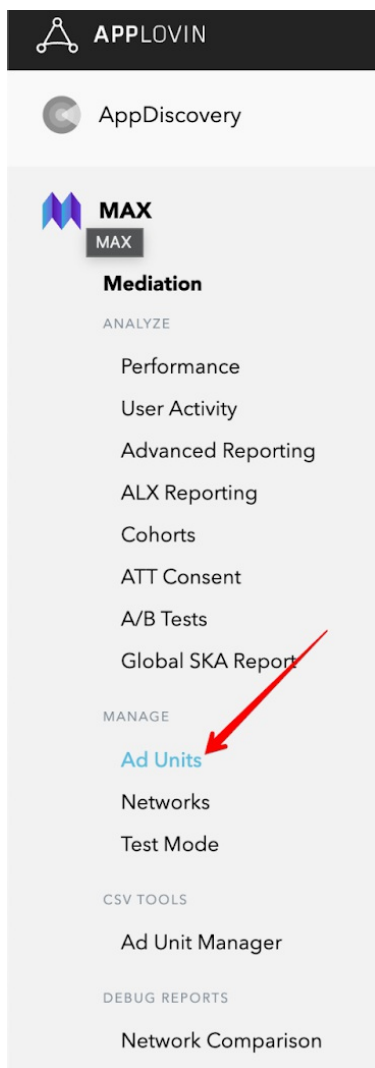**Network Type**

- ⦿ SDK

**Custom Network Name** ⓘ

TestNetwork

**iOS Adapter Class Name**

AppstockAppLovinAdapter

**Android / Fire OS Adapter Class Name**

4. Open [MAX > Mediation > Manage > Ad Units](#) in the MAX dashboard.

5. Search and select an ad unit for which you want to add the custom SDK network that you created in the previous step.



6. Select which custom network you want to enable and enter the information for each placement. Refer

to the network documentation to see what values you need to set for the **App ID**, **Placement ID**, and **Custom Parameters**.



Typically, the custom parameters field should contain a JSON that contains IDs (placement ID, endpoint ID) that will be used to load ads.

Parameters:

- **placement_id** - unique identifier generated on the platform's UI;

- **endpoint_id** - unique identifier generated on the platform's UI.

Example:

```
{
   "placement_id": "4"
}
```

7. Save ad unit.

# Native Ads

If you integrate native ads, you should pass the native assets through AppLovin MAX SDK ( `MANativeAdLoader` ) to the Appstock Adapter using `AppstockAppLovinExtras` class in your app code:

*Swift*

```swift
private func loadAd() {
    // 1. Create a MANativeAdLoader
    nativeAdLoader = MANativeAdLoader(adUnitIdentifier: adUnitId)

    // 2. Configure the MANativeAdLoader
    nativeAdLoader.nativeAdDelegate = self

    // 3. Configure the native parameters
    let image = AppstockNativeAssetImage(minimumWidth: 200,
    minimumHeight: 50, required: true)
    image.type = .Main

    let icon = AppstockNativeAssetImage(minimumWidth: 20,
    minimumHeight: 20, required: true)
    icon.type = .Icon

    let title = AppstockNativeAssetTitle(length: 90, required: true)
    let body = AppstockNativeAssetData(type: .description,
    required: true)
    let cta = AppstockNativeAssetData(type: .ctatext,
    required: true)
    let sponsored = AppstockNativeAssetData(type: .sponsored,
    required: true)

    let parameters = AppstockNativeParameters()
    parameters.assets = [title, icon, image, sponsored, body, cta]

    let eventTracker = AppstockNativeEventTracker(
        event: .Impression,
        methods: [.Image, .js]
    )

    parameters.eventtrackers = [eventTracker]
    parameters.context = .Social
    parameters.placementType = .FeedContent
    parameters.contextSubType = .Social

    // 4. Create a AppstockAppLovinExtras
    let extras = AppstockAppLovinExtras(nativeParameters: parameters)

    // 5. Set local extra parameter
    nativeAdLoader.setLocalExtraParameterForKey(
    AppstockAppLovinExtras.key, value: extras)

    // 6. Load the ad
    nativeAdLoader.loadAd(into: maNativeAdView)
}
```

*Objective-C*

```objc
- (void)loadAd {
    // 1. Create a MANativeAdLoader
    self.nativeAdLoader = [[MANativeAdLoader alloc]
    initWithAdUnitIdentifier:self.adUnitId];

    // 2. Configure the MANativeAdLoader
    self.nativeAdLoader.nativeAdDelegate = self;

    // 3. Configure the native parameters
    AppstockNativeAssetImage *image = [
        [AppstockNativeAssetImage alloc]
        initWithMinimumWidth:200
```

```
    initWithMinimumWidth:200
    minimumHeight:200
    required:true
];

image.type = AppstockImageAsset.Main;

AppstockNativeAssetImage *icon = [
    [AppstockNativeAssetImage alloc]
    initWithMinimumWidth:20
    minimumHeight:20
    required:true
];

icon.type = AppstockImageAsset.Icon;

AppstockNativeAssetTitle *title = [
    [AppstockNativeAssetTitle alloc]
    initWithLength:90
    required:true
];

AppstockNativeAssetData *body = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetDescription
    required:true
];

AppstockNativeAssetData *cta = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetCtatext
    required:true
];

AppstockNativeAssetData *sponsored = [
    [AppstockNativeAssetData alloc]
    initWithType:AppstockDataAssetSponsored
    required:true
];

AppstockNativeParameters * parameters =
[AppstockNativeParameters new];
parameters.assets = @[title, icon, image, sponsored, body, cta];

AppstockNativeEventTracker * eventTracker = [
    [AppstockNativeEventTracker alloc]
    initWithEvent:AppstockEventType.Impression
    methods:@[AppstockEventTracking.Image, AppstockEventTracking.js]
];

parameters.eventtrackers = @[eventTracker];
parameters.context = AppstockContextType.Social;
parameters.placementType = AppstockPlacementType.FeedContent;
parameters.contextSubType = AppstockContextSubType.Social;

// 4. Create a AppstockAppLovinExtras
AppstockAppLovinExtras * extras = [[AppstockAppLovinExtras alloc]
initWithNativeParameters: parameters];

// 5. Set local extra parameter
[self.nativeAdLoader
setLocalExtraParameterForKey:AppstockAppLovinExtras.key value:extras];
```

```
    // 6. Load the ad
    [self.nativeAdLoader loadAdIntoAdView:self.maNativeAdView];
}
```

Make sure you've bound the subviews using unique tag IDs with an instance of
`MANativeAdViewBinder` as described in [AppLovin MAX docs](#):

*Swift*

```
    // Bind the subviews using unique tag IDs with an instance of MANativeAdViewBinde
    let binder = MANativeAdViewBinder { builder in
        builder.iconImageViewTag = 1
        builder.titleLabelTag = 2
        builder.bodyLabelTag = 3
        builder.advertiserLabelTag = 4
        builder.callToActionButtonTag = 5
    }

    maNativeAdView.bindViews(with: binder)
```

*Objective-C*

```
MANativeAdViewBinder * binder = [
    [MANativeAdViewBinder alloc]
    initWithBuilderBlock:^(MANativeAdViewBinderBuilder * _Nonnull builder) {
        builder.iconImageViewTag = 1;
        builder.titleLabelTag = 2;
        builder.bodyLabelTag = 3;
        builder.advertiserLabelTag = 4;
        builder.callToActionButtonTag = 5;
    }
];

[self.maNativeAdView bindViewsWithAdViewBinder:binder];
```

# Appstock SDK iOS - Mediation - TopOn

In order to integrate Appstock TopOn Adapter into your app, add the following lines to your Podfile:

```
pod 'AppstockSDK', '1.0.1'
pod 'AppstockTopOnAdapter', '1.0.1'
```

To integrate the Appstock SDK into your TopOn monetization stack, you should create an ad network and add it to the respective ad units.

1.  Register an account at [toponad.com](#).

2.  Create an app in TopOn dashborad. Select [Application > Add app].
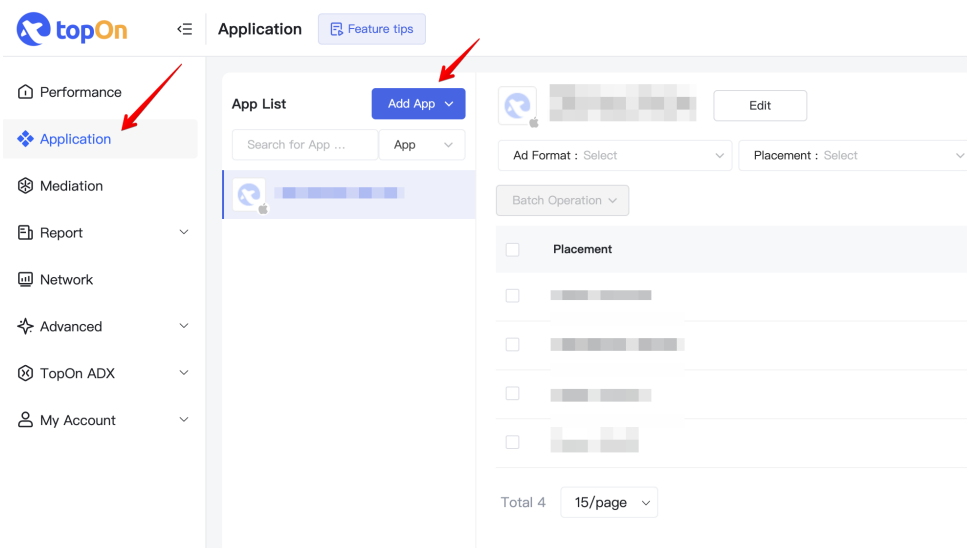
3. Fill the required information fields for your app.



**New App**                                                    ✕

| Platform | iOS | Android |

App Store — Yes / **No**

Download URL — optional

⚠ It is recommended that you fill in the Download URL of the app, otherwise it may affect the ad filling of the Bidding Ad Network

App Name — Test App                    8/100

Category — App

Sub–Category — Utilities

Bundle ID — com.test.app

Orientation — **Portrait** / Landscape / Auto

Weixin Open Platform ⓘ
App ID

Cancel    Confirm

4. Click `Confirm`.

## New App                                                            ✕

| Platform | iOS | Android |
| --- | --- | --- |

App Store   [ Yes ] [ **No** ]

Download URL   [ optional ]

⚠ It is recommended that you fill in the Download URL of the app, otherwise it may affect the ad filling of the Bidding Ad Network

App Name   [ Test App                                    8/100 ]

Category   [ App                                              ⌄ ]

Sub–Category   [ Utilities                                    ⌄ ]

Bundle ID   [ com.test.app                                     ]

Orientation   [ **Portrait** ] [ Landscape ] [ Auto ]

Weixin Open Platform
App ID  ⑦   [                                                 ]

[ Cancel ]   [ **Confirm** ]

5. Click  `Add placement` .



6. Select the app. Fill  `Placement name`  and  `Ad Format`  fields.

## New Placement                                               ✕

| App | 🍎 Test App ▾ |
|---|---|
| Placement Name | Test Banner                          11/100 |
| Ad Format | Banner ▾ |
| Status | Running ▾ |
| Note | optional |

Cancel          **Confirm**

7. Select `Network` and click `+ Custom Network Firm`.



◯ topOn    ⬅    Monetization    🗐 Feature tips

| | |
|---|---|
| ⌂ Performance | Filter by  2024–09–03 – 2024–09–03  📅     Network : – No selection – ▾ |
| ⬡ Application | |
| ⊕ Mediation | **+ Network Account**    **+ Custom Network Firm** |
| 🗐 Report ▾ | |
| 📊 Network | Network | Integration Type | Account | Operation |
| ✧ Advanced ▾ | TeqBlaze | AD Network SDK | Default | Export Config | AD Source Setting |
| ⊗ TopOn ADX ▾ | |
| 👤 My Account ▾ | Total 1    15/page ▾ |

8. Fill `Network Firm Name`. Fill the adapter's class name:

- Interstitial - `AppstockInterstitialATAdAdapter`;

- Banner - `AppstockBannerATAdAdapter`;

- Native - `AppstockNativeATAdAdapter`.

## Add Custom Network Firm                                              ✕

| Native | Example: com.customnetwork.android.native |
|--------|-------------------------------------------|
| Splash | Example: com.customnetwork.android.splash |

| iOS | |
|-----|---|
| RV | Example: CustomRewardedVideoAdapter |
| Interstitial | AppstockInterstitialATAdAdapter |
| Banner | AppstockBannerATAdAdapter |
| Native | AppstockNativeATAdAdapter |
| Splash | Example: CustomSplashAdapter |

Bidding For Each Load ⓘ    [ Opened ] [ Closed ]

Icon

Select Optional

How to find the parameters ?                    [ Cancel ]  [ Confirm ]

9. Click `Confirm` .

## Add Custom Network Firm                                                    ✕

| Native | Example: com.customnetwork.android.native |
|--------|-------------------------------------------|
| Splash | Example: com.customnetwork.android.splash |

**iOS**

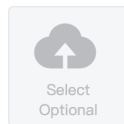| RV | Example: CustomRewardedVideoAdapter |
|----|-------------------------------------|
| Interstitial | AppstockInterstitialATAdAdapter |
| Banner | AppstockBannerATAdAdapter |
| Native | AppstockNativeATAdAdapter |
| Splash | Example: CustomSplashAdapter |

Bidding For Each Load ⑦    [ **Opened** | Closed ]

Icon    ⬆ Select Optional

How to find the parameters ?          Cancel    **Confirm**

---

10. Open the `Mediation` tab, select the app and placement, click `Add AD source`.



11. Find the needed network. Add `Ad source name` and `Price`. Fill the `Custom Parameters` Custom parameters should contain a valid JSON with IDs (placement ID, endpoint ID) values that will be used by the adapter to load ads. Click `Confirm`.

**Add Ad Source** 1

Test

+ Test Network

Segment    Default

User Name    Default

Ad Source Name    Test Network_banner_1    ← 2    21/50

Header Bidding ?    Yes    No

Price($) ?    1.00    ← 3

Other Parameters ?    {"placement_id":"4"}    ← 4

SDK Json ?    {"placement_id":"4"}

How to find the parameters ?

Bottom Ad ?    ⬭

5    More Config ▾

Cancel    Confirm

# Native Ads

If you integrate native ads, you should pass the native assets through extras to the Appstock Adapter using `kAppstockNativeAssets` key in your app code:

**Swift**

```swift
private func loadAd() {
    // 1. Configure the native parameters
    let image = AppstockNativeAssetImage(minimumWidth: 200, minimumHeight: 50, requi
    image.type = .Main

    let icon = AppstockNativeAssetImage(minimumWidth: 20, minimumHeight: 20, require
    icon.type = .Icon

    let title = AppstockNativeAssetTitle(length: 90, required: true)
    let body = AppstockNativeAssetData(type: .description, required: true)
    let cta = AppstockNativeAssetData(type: .ctatext, required: true)
    let sponsored = AppstockNativeAssetData(type: .sponsored, required: true)

    let parameters = AppstockNativeParameters()
    parameters.assets = [title, icon, image, sponsored, body, cta]

    let eventTracker = AppstockNativeEventTracker(
        event: .Impression,
        methods: [.Image, .js]
    )

    parameters.eventtrackers = [eventTracker]
    parameters.context = .Social
    parameters.placementType = .FeedContent
    parameters.contextSubType = .Social

    // 2. Set up the extras
    let extra = [
        kAppstockNativeAssets: parameters
    ]

    // 3. Load the ad
    ATAdManager.shared().loadAD(
        withPlacementID: placementID,
        extra: extra,
        delegate: self
    )
}
```

*Objective-C*

```objectivec
- (void)loadAd {
    // 1. Configure the native parameters
    AppstockNativeAssetImage *image = [
        [AppstockNativeAssetImage alloc]
        initWithMinimumWidth:200
        minimumHeight:200
        required:true
    ];

    image.type = AppstockImageAsset.Main;

    AppstockNativeAssetImage *icon = [
        [AppstockNativeAssetImage alloc]
        initWithMinimumWidth:20
        minimumHeight:20
        required:true
    ];

    icon.type = AppstockImageAsset.Icon;
```

```objectivec
    AppstockNativeAssetTitle *title = [
        [AppstockNativeAssetTitle alloc]
        initWithLength:90
        required:true
    ];

    AppstockNativeAssetData *body = [
        [AppstockNativeAssetData alloc]
        initWithType:AppstockDataAssetDescription
        required:true
    ];

    AppstockNativeAssetData *cta = [
        [AppstockNativeAssetData alloc]
        initWithType:AppstockDataAssetCtatext
        required:true
    ];

    AppstockNativeAssetData *sponsored = [
        [AppstockNativeAssetData alloc]
        initWithType:AppstockDataAssetSponsored
        required:true
    ];

    AppstockNativeParameters * parameters = [AppstockNativeParameters new];
    parameters.assets = @[title, icon, image, sponsored, body, cta];

    AppstockNativeEventTracker * eventTracker = [
        [AppstockNativeEventTracker alloc]
        initWithEvent:AppstockEventType.Impression
        methods:@[AppstockEventTracking.Image, AppstockEventTracking.js]
    ];

    parameters.eventtrackers = @[eventTracker];
    parameters.context = AppstockContextType.Social;
    parameters.placementType = AppstockPlacementType.FeedContent;
    parameters.contextSubType = AppstockContextSubType.Social;

    // 2. Set up the extras
    NSDictionary *extra = @{
        kAppstockNativeAssets : parameters
    };

    // 3. Load the ad
    [[ATAdManager sharedManager] loadADWithPlacementID:self.placementID
                                        extra:extra
                                        delegate:self];
}
```