Space Ad Mobile

Google AdMob

Unity Integration Guide



Space Ad Mobile unifies ad integration for your Unity mobile game with a no/low-code setup so you ship faster

OV [App Open ads]	Ban ner view [Banner ads]	InV [Interstitial ads]
RV [Rewarded ads]	Rew InV [Rewarded interstitial ads]	NO [Native overlay ads]

Contents

1. Prepare the Ad Platform 3
2. Creating Assets and Setting Up Scenes 6
3. Example Walkthrough and Ad Formats 9
   App Open Ads 9
   Banner Ads 10
   Interstitial Ads 11
   Rewarded Ads 11
   Rewarded Interstitial Ads 11
   Native Overlay Ads 12
   Mediation 12
   Ad Callbacks and Reward Delivery 13
   Rewarded Ad Example 15
   Preventing Banners from Obscuring the UI 17
4. API Reference and Settings 19
   Additional AdMob Features 20
   Useful Tips: 20

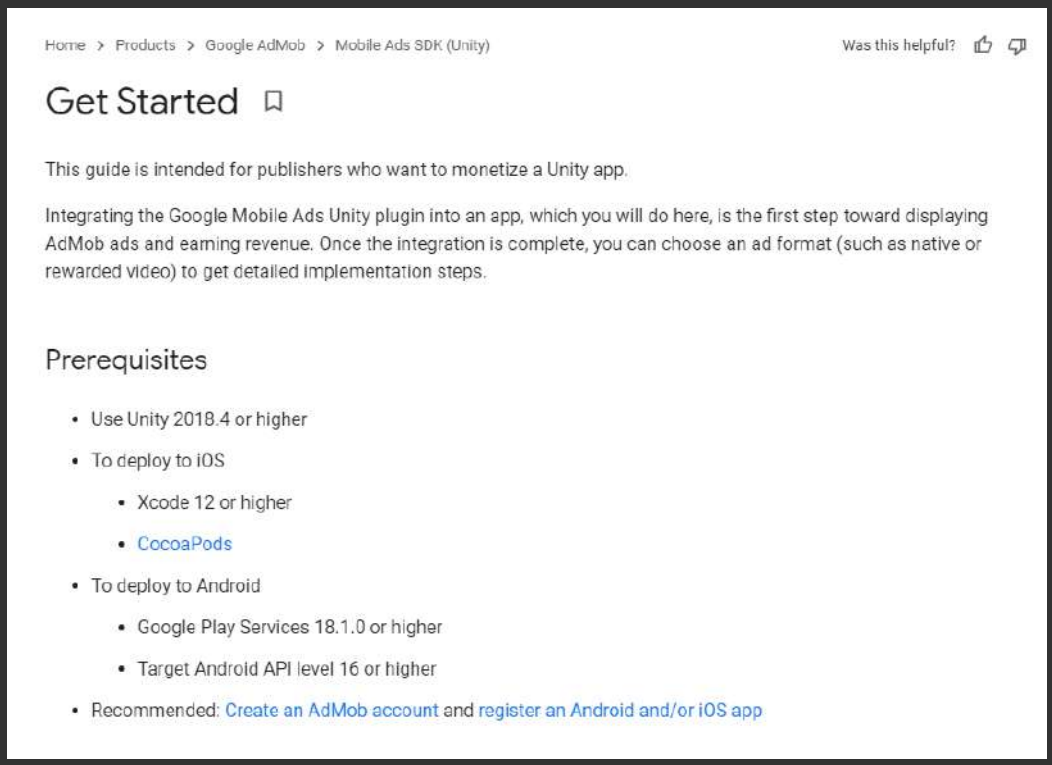
> *Space Ad Mobile* makes ads easy to add into your mobile game: one unified workflow for LevelPlay, AdMob, Yandex.Ads across banners, interstitials, rewarded. Ship faster with less setup. No code required.

1. Prepare the Ad Platform

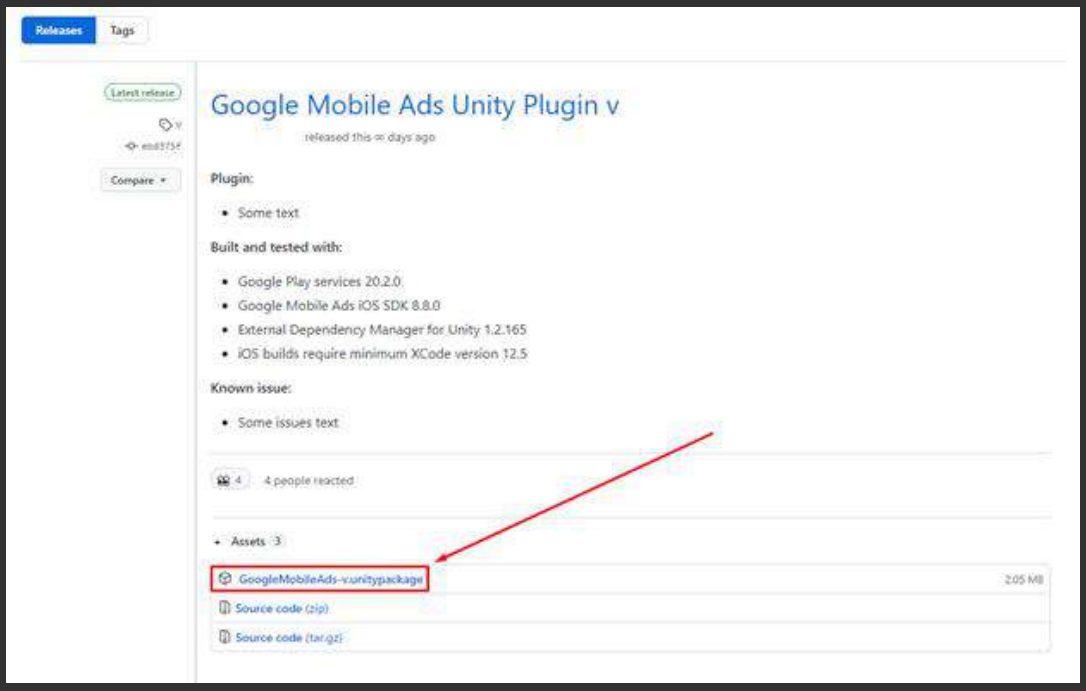
Follow these steps to set up AdMob for the first time:

(Skip this section if AdMob is already configured.)

1. Follow the official Google AdMob setup guide: https://developers.google.com/admob/unity/quick-start.



2. Download and install the Google Mobile Ads Unity plugin from: https://github.com/googleads/googleads-mobile-unity/releases/



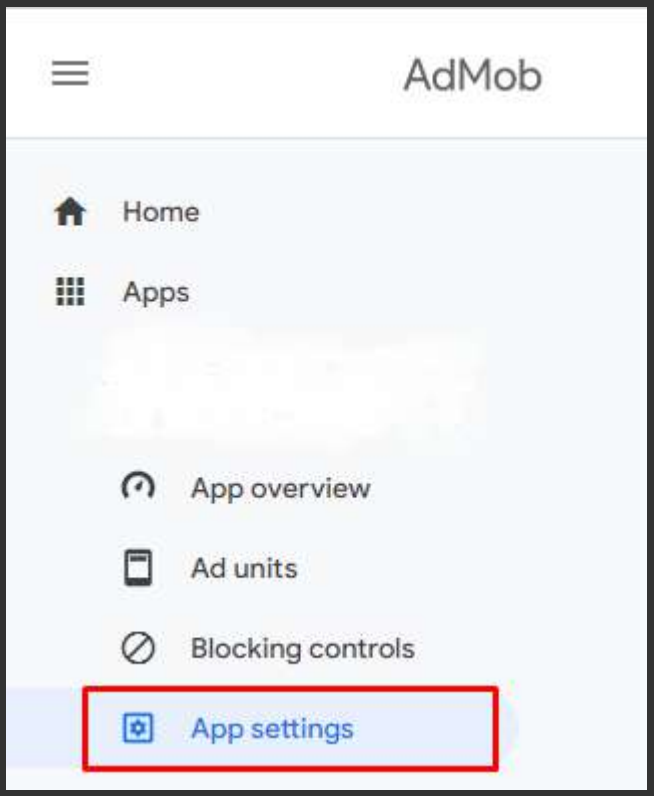
3. Create an AdMob account by following these instructions: https://support.google.com/admob/answer/7356219

4. Create an app in AdMob by following these instructions:
   https://support.google.com/admob/answer/9989980
5. Copy your app ID.

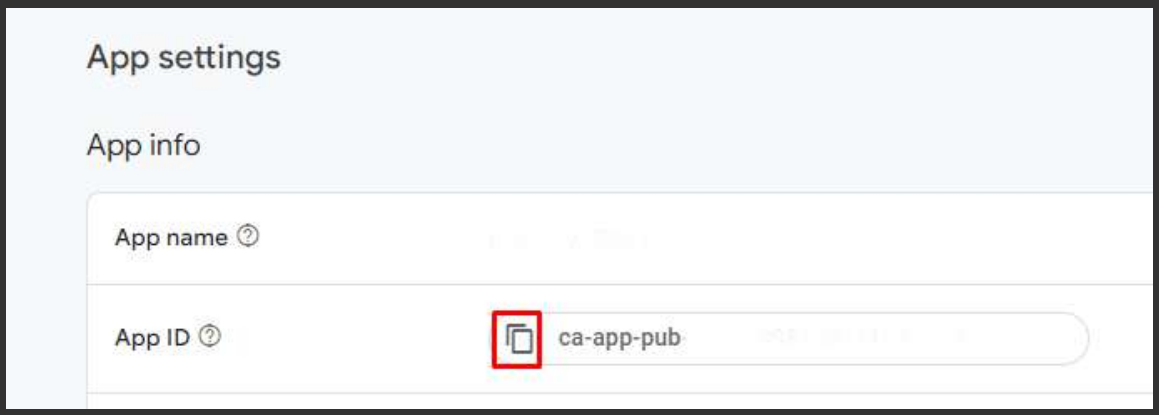
If your app already exists, open the AdMob apps page:

https://apps.admob.com/v2/apps

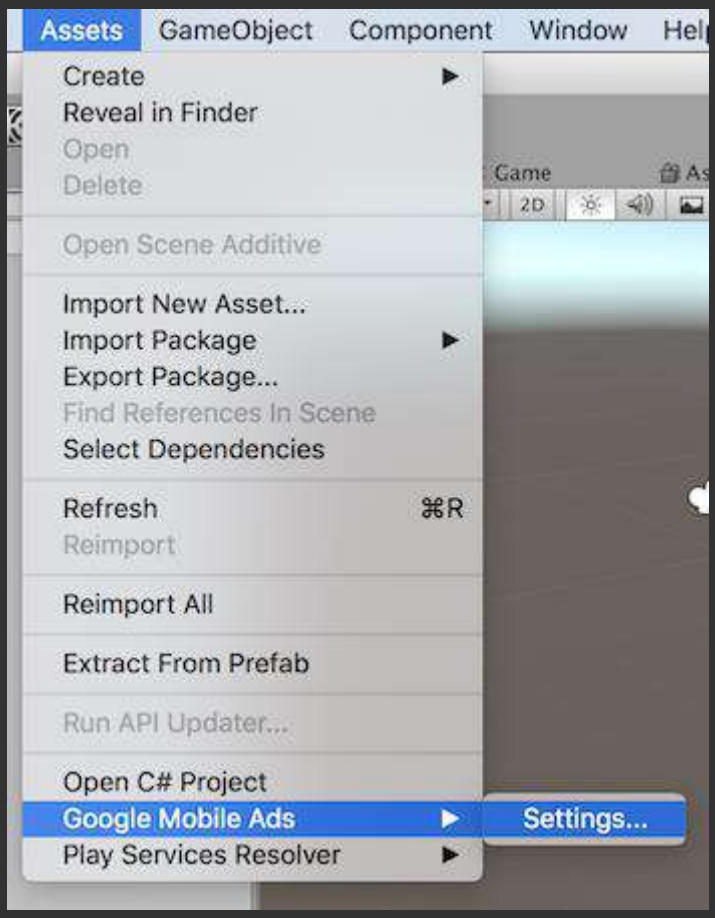
Select your app, open App settings, and copy the ID.

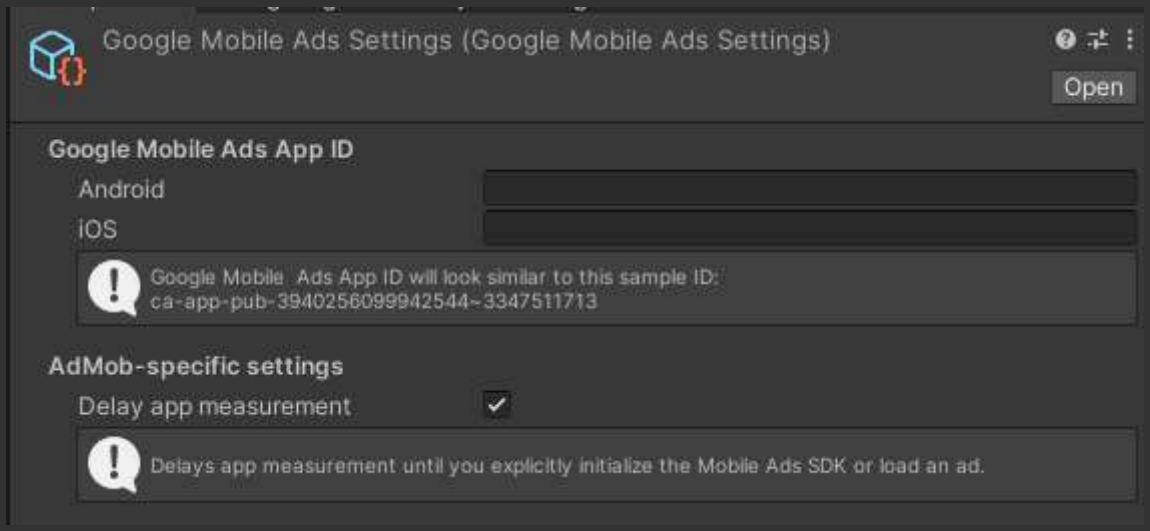


Then copy the app ID.



6. In the Unity Editor, go to Assets -> Google Mobile Ads -> Settings and enter your AdMob app ID.





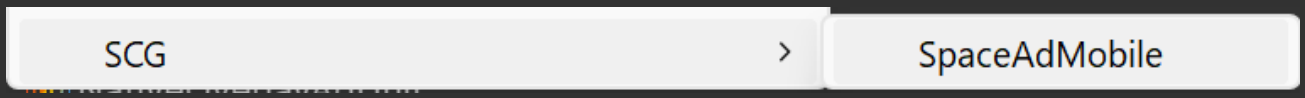
7. Create an ad unit and copy its ID.

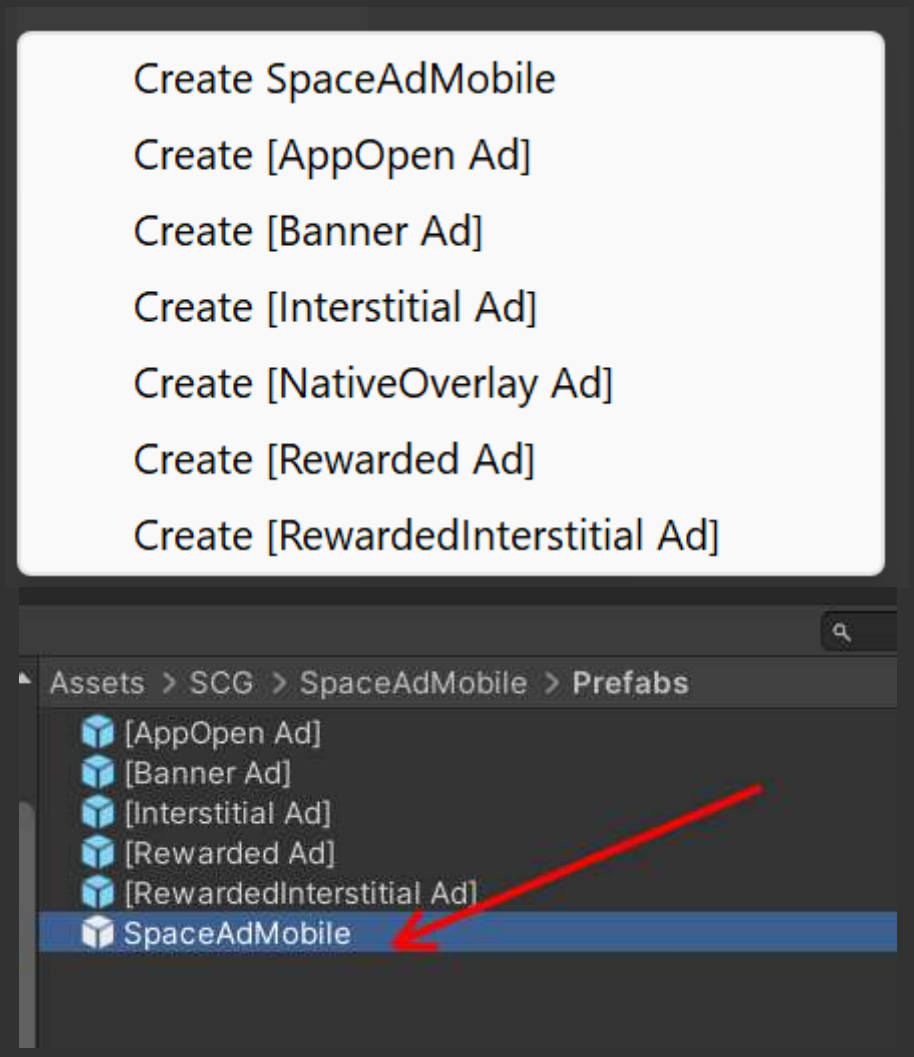


2. Creating Assets and Setting Up Scenes

Once your AdMob account is configured, you can set up the *SpaceAdMobile* asset.

1. Open the preload scene that runs before the rest of the game. Open the context menu and add the required components.

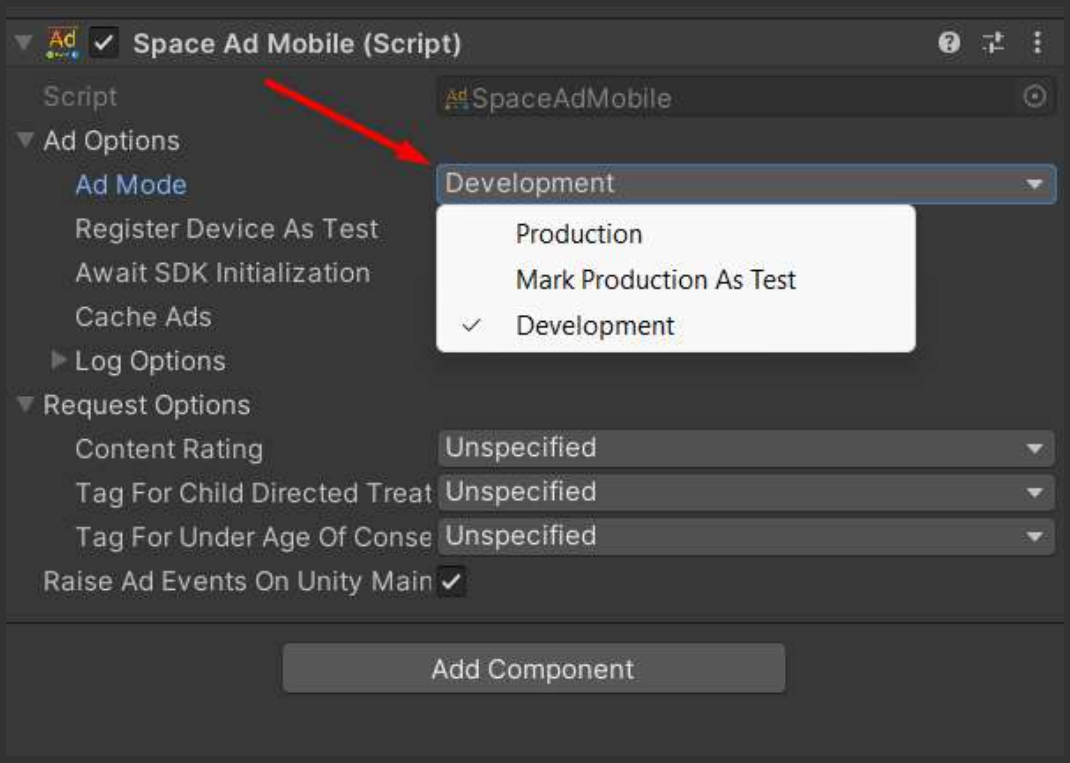




2. This prefab initializes the Google Mobile Ads SDK.

Choose whether to display test ads or live ads.

Learn more about test ads: https://developers.google.com/admob/unity/test-ads



3. Open the scene where the ad units will be loaded.

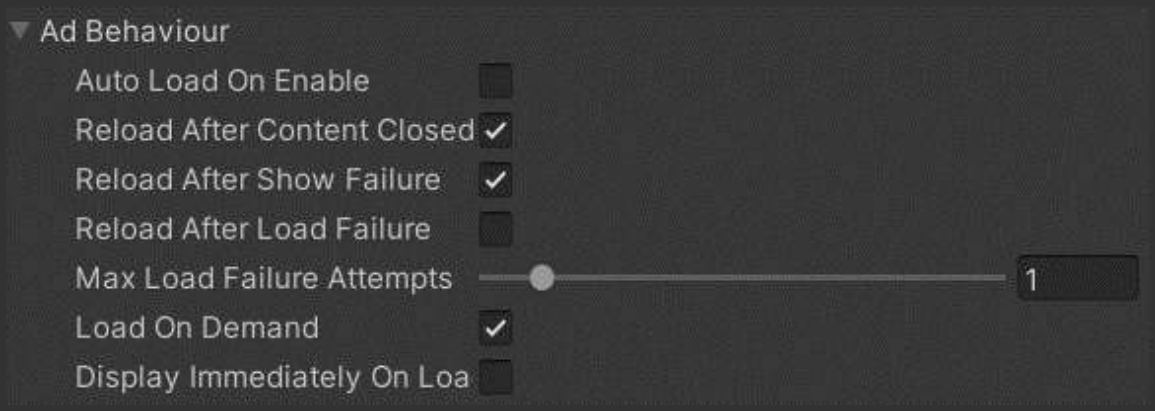
Drag the prefab for the required ad format into the scene.



4. Each ad unit needs the ID generated when you created it in AdMob. Enter the ID for the target platform.

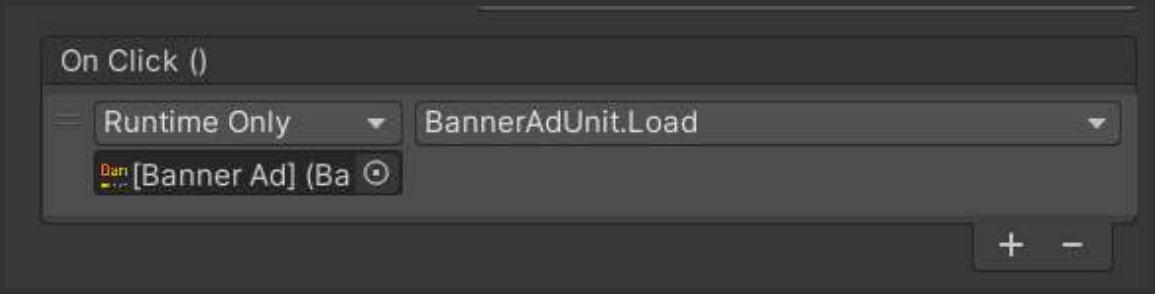


5. A key benefit of Space Ad Mobile is automatic ad loading and display. Take a look at the Ad Behaviour settings:



These settings are explained in more detail below.

6. Add a load event to a button (a banner ad in this example):



Note: Banners appear as soon as they load, without calling Show().

3. Example Walkthrough and Ad Formats

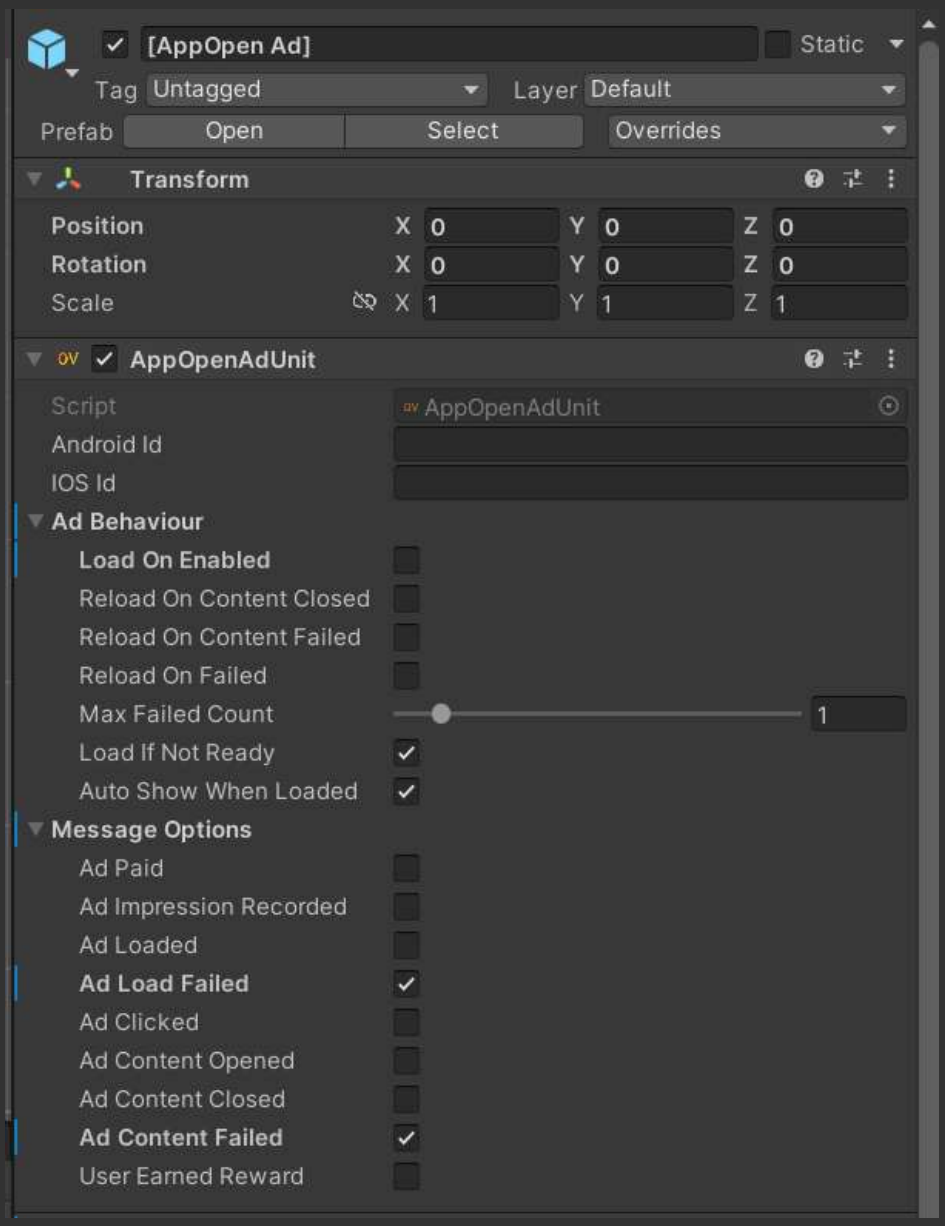
The example includes two scenes: Loading Scene, which contains the AdMob initializer, and SpaceAdMobile Demo, which loads the ads. Let us take a closer look at the [Ad Units] GameObject.

App Open Ads

This is a special ad format designed for publishers who want to monetize their app loading screens. App Open ads can be closed at any time and are designed to be shown when users bring your app to the foreground.

Use the [AppOpen Ad] prefab and the `AppOpenAdUnit` component.

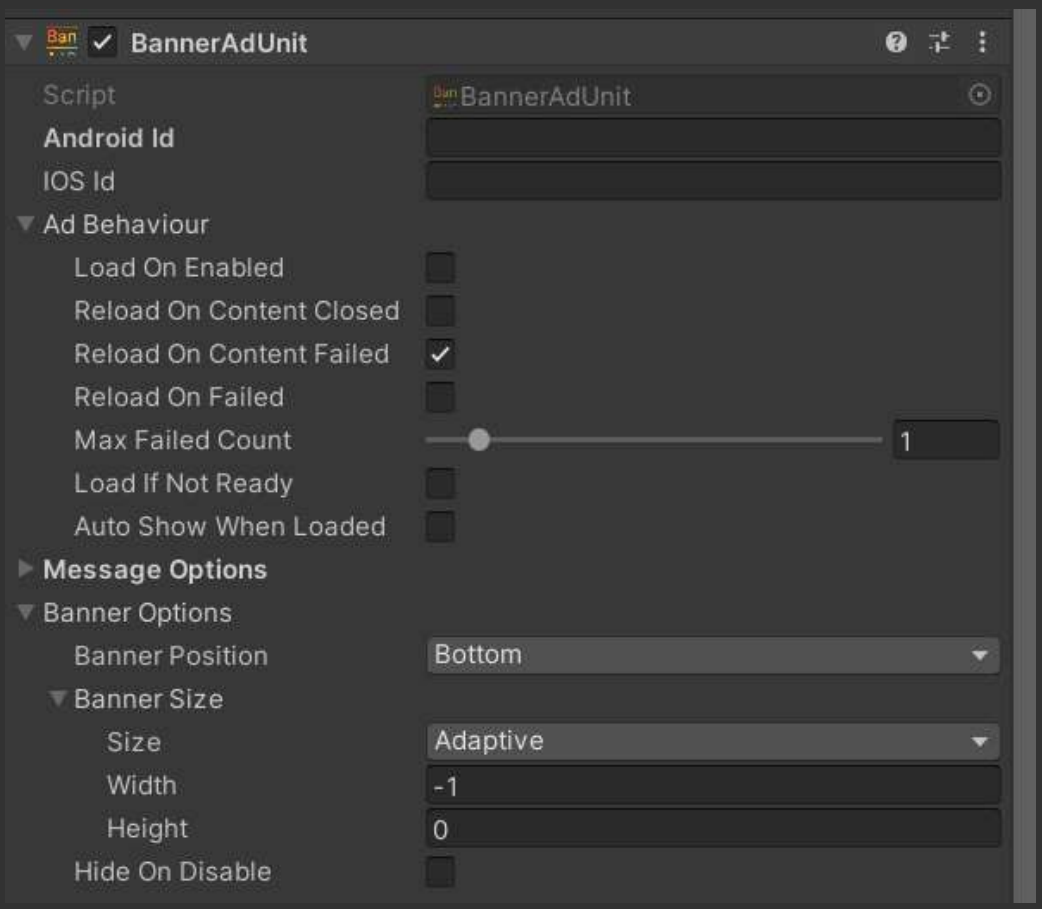
You can place it in the preload scene alongside the initialization prefab.



Banner Ads

These are rectangular image or text ads that occupy a certain part of the screen. They remain on the screen while users interact with the app and can be refreshed automatically after a set period of time.

Use the *[Banner Ad]* prefab and the *BannerAdUnit* component.



You can also configure the banner position. By default, the adaptive banner is placed at the bottom. Below you can see the available banner sizes.

Banner sizes

The table below lists the standard banner sizes.

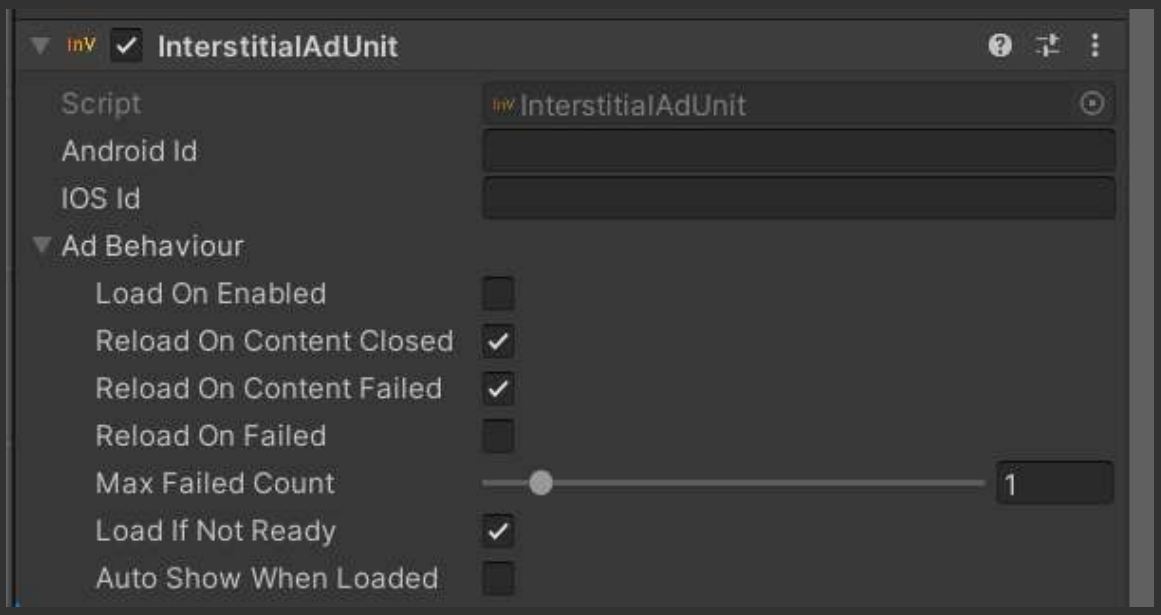
Size in dp (WxH)	Description	Availability	AdSize constant
320x50	Standard Banner	Phones and Tablets	BANNER
320x100	Large Banner	Phones and Tablets	LARGE_BANNER
300x250	IAB Medium Rectangle	Phones and Tablets	MEDIUM_RECTANGLE
468x60	IAB Full-Size Banner	Tablets	FULL_BANNER
728x90	IAB Leaderboard	Tablets	LEADERBOARD
Provided width x *Adaptive height*	Adaptive banner	Phones and Tablets	N/A
Screen width x 32|50|90	Smart banner	Phones and Tablets	SMART_BANNER

Learn more about Adaptive Banners, intended to replace Smart Banners.

Interstitial Ads

This is a full-screen ad that covers the app interface. It is usually displayed at natural transition points in the app flow, such as during a pause between levels in a game. When the app shows an interstitial ad, the user can either click the ad and go to the destination or close it and return to the app.

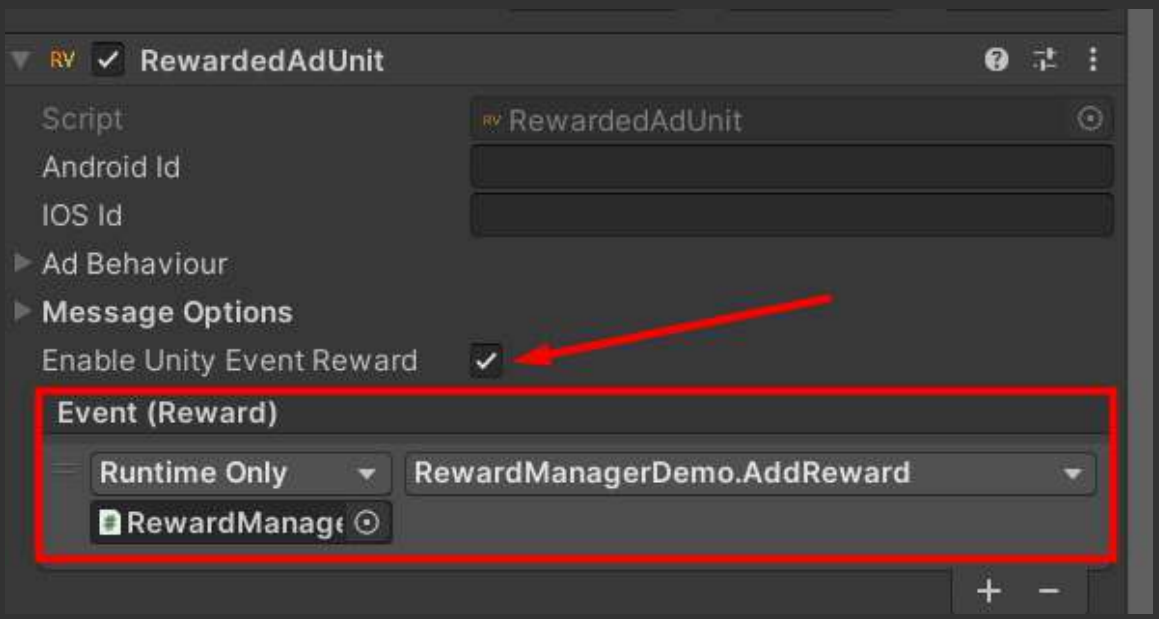
Use the *[Interstitial Ad]* prefab and the *InterstitialAdUnit* component.



Rewarded Ads

These are ads that users can interact with in exchange for in-app rewards.

Use the *[Rewarded Ad]* prefab and the *RewardedAdUnit* component.



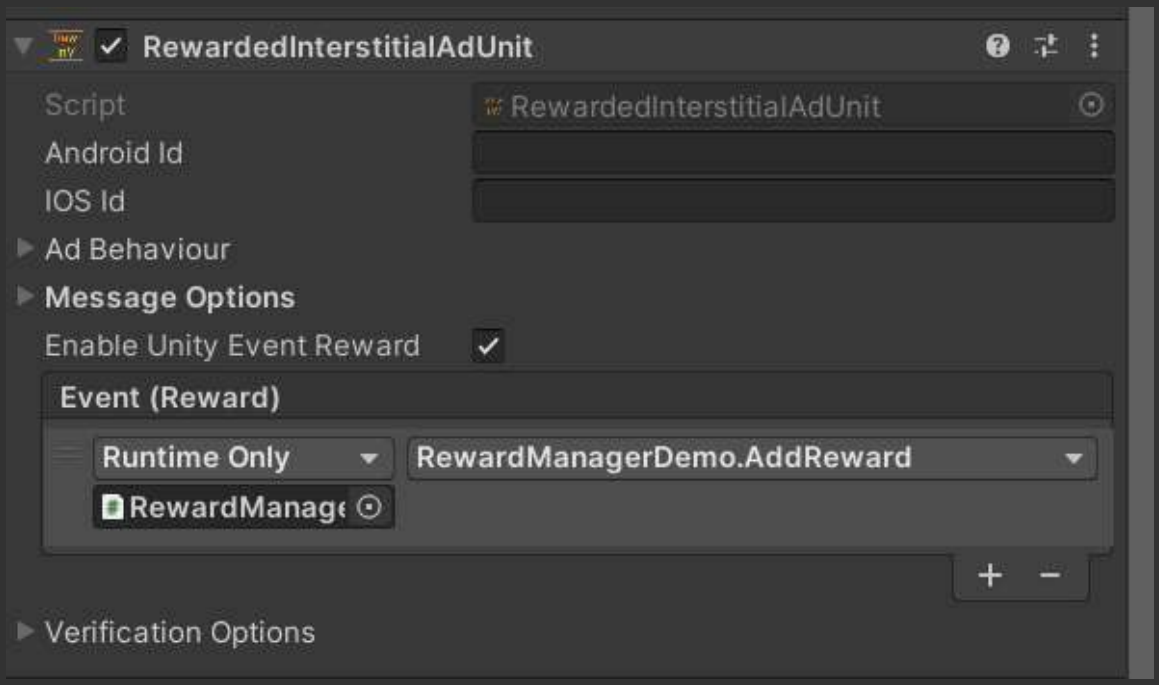
`Note`: Enable the Unity event flag before using this event.

Rewarded Interstitial Ads

Rewarded interstitial ads are full-screen ads shown at natural breaks in gameplay. AdMob confirms the reward when the ad is completed. Use the *[RewardedInterstitial Ad]* prefab and the `RewardedInterstitialAdUnit` component.

Tell players what reward they will receive before displaying the ad.

VerificationOptions for rewarded ads: If your app needs to pass additional data to server-side verification (SSV) callbacks, use the custom_data field. Any string set on the rewarded ad object is passed as the custom_data parameter in the SSV callback. If no value is provided, custom_data is omitted from the callback.



Native Overlay Ads

The AdMob adapter implements native ads through Native Overlay. It displays an ad over the application using native Android or iOS UI elements and a configurable template.

Use the [NativeOverlay Ad] prefab and the `NativeOverlayAdUnit` component. Configure the template, position, size, and appearance in Native Overlay Options. After loading, the component calls RenderTemplate(); Hide() hides the ad, and Show() displays it again.

Mediation

To enable mediation, install the required adapters:
https://developers.google.com/admob/unity/choose-networks

No additional code is required once the adapters are installed. Use the MediationAdaptersConfigured property of SpaceAdMobile to check whether the mediation adapters have finished loading.

Ad Callbacks and Reward Delivery

Override the relevant callback in a class derived from the ad component. Call base to preserve the internal state updates and configured events.

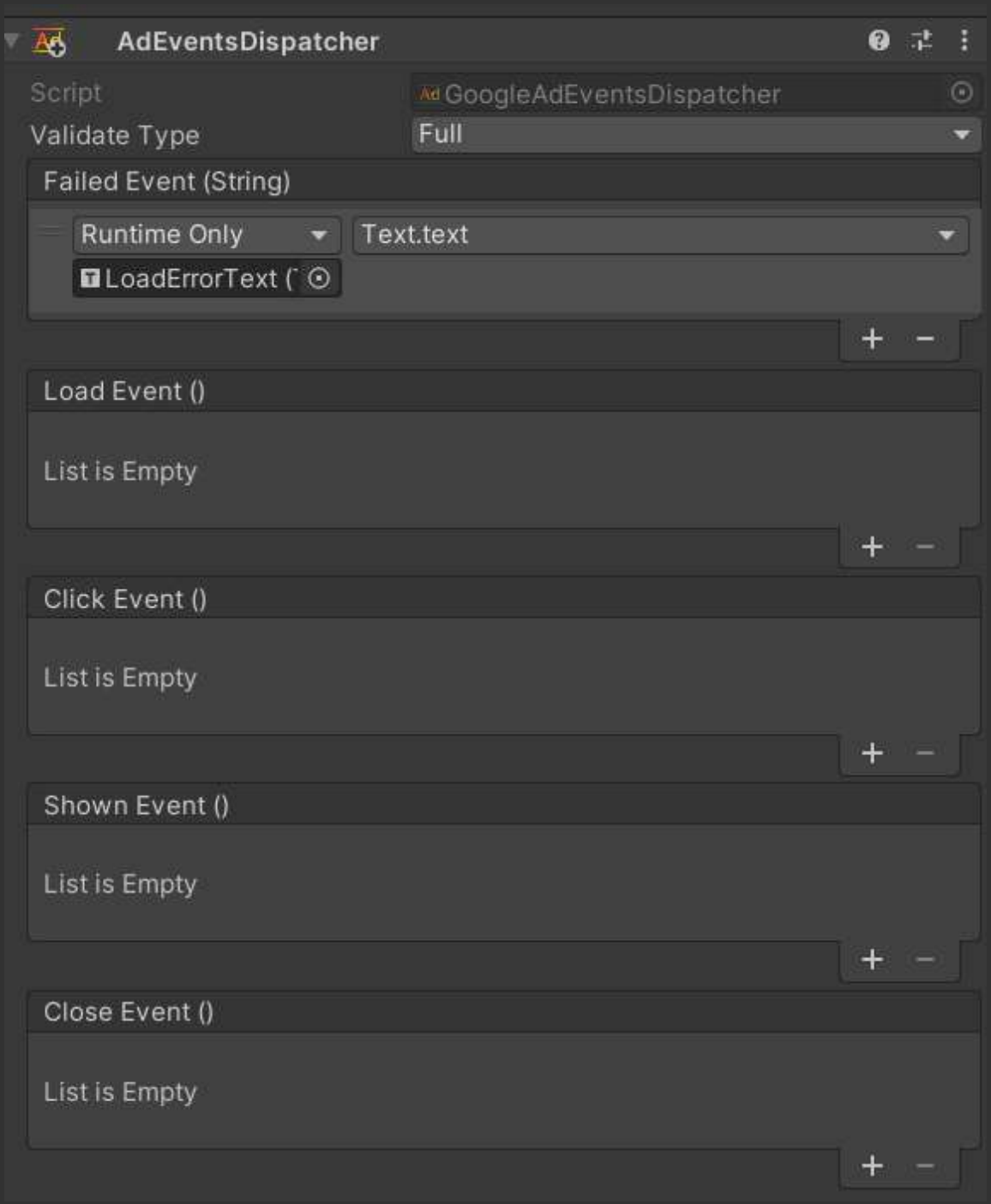
```
using UnityEngine;
using SCG.SpaceAdMobile;

public class RewardedAdExample : RewardedAdUnit
{
    protected override void OnAdLoaded()
    {
        base.OnAdLoaded();
        Debug.Log("Rewarded ad loaded");
    }

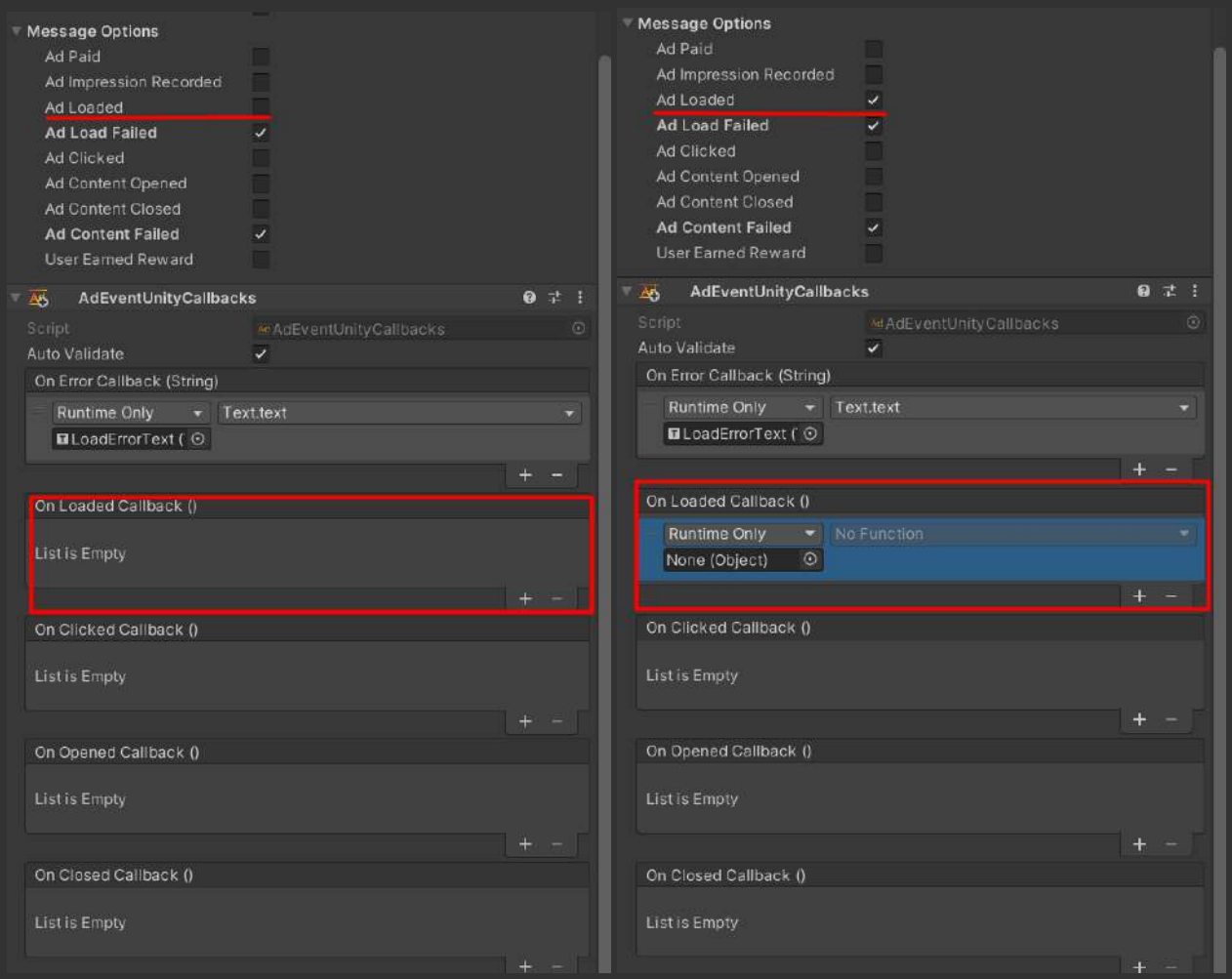
    protected override void OnAdShown()
    {
        base.OnAdShown();
        Debug.Log("Rewarded ad shown");
    }
}
```

Use the inherited Load() and Show() methods to load and display ads. `OnAdLoaded` is called after an ad loads, and `OnAdShown` when it appears on screen.

To configure events in the Inspector, add `AdEventsDispatcher` and connect your event handlers.



When AutoValidate is enabled, adding a Unity event automatically enables the matching interface callback flag. For example, adding an OnLoadedCallback handler enables the AdLoaded option in MessageOptions.



Rewarded Ad Example

Grant rewards only after confirmation from the ad platform. In this example, the player wallet reads the reward name and amount from `AdRewardBase` and adds `coins`.

```
using UnityEngine;
using SCG.SpaceAdMobile;

public class PlayerWallet : MonoBehaviour
{
    private int coins;

    public void GrantReward(AdRewardBase reward)
    {
        if (reward.Name == "coins")
            coins += reward.Amount;
    }
}
```

Add `PlayerWallet` to a GameObject. In `RewardedAdUnit`, enable Enable Unity Event Reward and connect Event (Reward) to `PlayerWallet.GrantReward` with a dynamic `AdRewardBase` argument.

Set the reward name to `coins` in your ad platform settings. In your game, also save the balance and update the UI.

Alternatively, subscribe the same handler to OnRewarded. Use only one reward path: using both the subscription and UnityEvent would grant the reward twice.

Available callbacks:

`OnAdPaid(AdValue)`
Called when ad revenue data is reported. AdValue contains the amount, currency and precision of the estimate.

`OnAdImpressionRecorded()`
Called when the SDK records an ad impression. It reports the impression, not reward confirmation.

`OnAdLoaded()`
Called after an ad loads successfully. Use it to update the UI and enable the show button.

`OnAdLoadFailed(LoadAdError)`
Called when an ad fails to load. The error argument contains details of the failure.

`OnAdClicked()`
Called when the user clicks the ad.

`OnAdShown()`
Called when the ad appears on screen. Use it for actions at the start of a display; reward confirmation arrives separately.

`OnAdClosed()`
Called after the ad closes. Use it to restore the game UI and resume gameplay.

`OnAdContentFailed(AdError)`
Called when a loaded ad cannot be displayed. The error argument describes the failure.

`OnUserEarnedReward(AdReward)`
Called when the user earns a reward.

Can I keep a banner visible throughout the game?

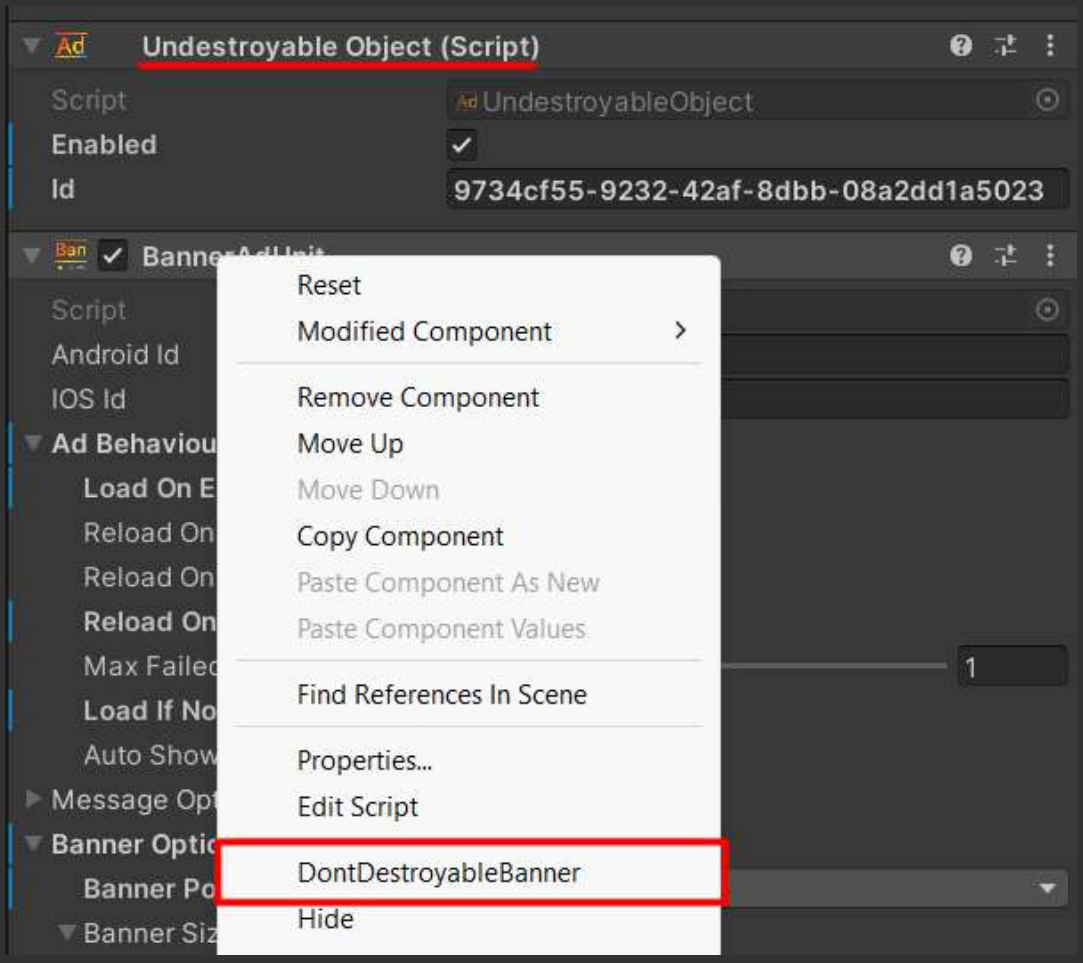
Absolutely. If your game switches between scenes, you do not have to lose the banner or control over it.

You can load the banner once and keep it visible across scene changes for the entire game session.

Keep in mind that other UI elements may need to appear in the same area as the banner. Plan when the ad should remain visible and when it should be hidden temporarily.

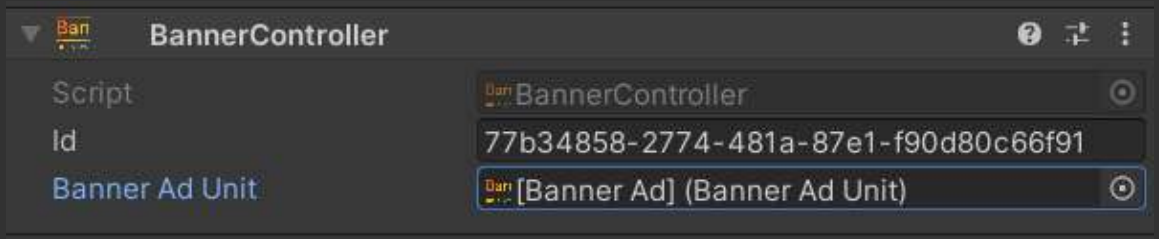
Use the *UndestroyableUtility* component to control whether the banner object is destroyed.

For quick setup, open the BannerAdUnit context menu and select DontDestroyableBanner. This adds a component with a unique ID.



To duplicate a banner, clear the ID field so a new ID can be generated.

Then add BannerController to the scene where the persistent ad is initialized.



Be sure to assign the banner reference!

To control a banner from another scene, create a controller, copy the ID from the controller in the first scene, and paste it into the new controller. Leave `BannerAdUnit` empty.

Preventing Banners from Obscuring the UI

When a banner remains on screen during gameplay, it can cover buttons, the HUD, bottom panels, and other UI elements. Use the offset helper components to move the UI automatically based on the banner anchor and its actual runtime size.

Which helper should you use?

Use [`BannerCanvasOffsetHelper`] for classic Unity UI built on Canvas and RectTransform. Use [`BannerToolkitOffsetHelper`] for UI Toolkit screens built on UIDocument and VisualElement. Both components solve the same problem, but each one targets a different UI system.

Recommended Setup

Place the helper on the object that owns the main UI root for the current screen. If the scene contains more than one Canvas or more than one UIDocument, assign the reference manually so the helper does not pick the first available object in the scene.

Set Banner Ad Component to a banner component that implements IBannerAd, usually `BannerAdUnit`. If the banner persists between scenes through `BannerController`, also assign Banner Ad Controller so the helper can access it even when `BannerAdUnit` is not present in the current scene. Only assign a controller if it is already connected to the persistent banner.

BannerCanvasOffsetHelper

This version is for Canvas-based UI. The Canvas field points to the root Canvas that should be adjusted. Explicit Container is optional and expects a RectTransform inside that Canvas. If you leave Explicit Container empty, or if the assigned RectTransform is outside the selected Canvas hierarchy, the helper creates or reuses a child container named sub-root, moves the current Canvas children into it, and offsets that generated container.

BannerToolkitOffsetHelper

This version is for UI Toolkit. The Ui Document field points to the UIDocument that hosts the screen. Explicit Container Name is optional and expects the name of a VisualElement inside the visual tree. If the name is empty or the element is not found, the helper creates a VisualElement named sub-root, moves the current root children into it, and offsets that generated container.

Inspector Fields

Margin Top / Bottom / Left / Right add extra pixels on top of the detected banner size. This is useful when you want to keep a small visual gap between the banner and the nearest UI element.

Shift Only Banner Side controls how the layout is moved. When it is enabled, only the side occupied by the banner receives extra inset. When it is disabled, the whole target container is shifted away from the banner instead. For most HUD layouts, the enabled state is the safer default.

Runtime Behavior

No manual method calls are required. The helper resolves the banner, subscribes to load and hide events, and also watches visibility changes so the offset is applied if the banner becomes visible later and removed when the banner is hidden or unloaded.

Offsets are calculated from the banner anchor. Top, Bottom, Left, and Right anchors use the corresponding banner size plus your extra margin. Center and Custom anchors do not add any offset. If several helpers target the same container, an internal side-lock prevents the same side from being offset twice.

In most projects, one helper on the main gameplay UI root is enough. If the banner is meant to survive scene changes, combine the helper with `BannerController` so the UI in every scene can stay clear of the persistent banner.

4. API Reference and Settings

SpaceAdMobile

- `Status` - AdMob initialization status; may be null.
- `MediationAdaptersConfigured` - whether all mediation adapters are loaded
- `Ready` - whether AdMob initialization is complete
- `AdvertisingOptions` - general ad behavior settings
  - `AdMode` - ad mode: test or live
  - `RegisterDeviceAsTest` - ads will still be shown as test ads even if the app was downloaded from an online store
  - `AwaitSDKInitialization` - wait for SDK initialization (recommended: true)
  - `LogOptions` - logging settings; disable unnecessary logs.
- `RequestOptions` - request parameters
- `RaiseAdEventsOnUnityMainThread` - dispatch ad events on the main Unity thread (recommended: true)

*AdUnit

- `androidId` - ad unit ID for Android devices
- `iOSId` - ad unit ID for Apple devices
- `adBehaviour` - ad loading and display behavior
  - `AutoLoadOnEnable` - load the ad automatically in `OnEnable`
  - `ReloadAfterContentClosed` - reload the ad after it is closed
  - `ReloadAfterShowFailure` - reload the ad after a content display error
  - `ReloadAfterLoadFailure` - reload the ad after a load error

(Google discourages repeated retries, so use this carefully.)

  - `MaxLoadFailureAttempts` - number of attempts to load after an error
  - `LoadOnDemand` - calling Show() also loads the ad. Convenient, right?
  - `DisplayImmediatelyOnLoad` - show the ad immediately after loading; do not combine with other preload settings
- `messageOptions` - flags that enable callbacks in I*Message interfaces. Avoid enabling everything; subclass callbacks are generally less resource-intensive.
- `Load` - method for loading the ad
- `Show` - method for displaying the ad

*AdUnit*Handlers

You can safely override these methods in your scripts:

- `AdPaid` - called when the ad is evaluated as revenue-generating
- `AdImpressionRecorded` - called when the ad impression is recorded
- `AdLoaded` - event fired when an ad is loaded (yes, even a banner)
- `AdLoadFailed` - event fired when ad loading fails
- `AdClicked` - triggered when an ad click is registered
- `AdShown` - triggered when an ad is opened
- `AdClosed` - triggered when the ad is closed
- `AdContentFailed` - triggered when ad display fails
- `AdReward` - called when the user earns a reward (rewarded ad formats). The best kind of callback!

AdEventsDispatcher

- ValidateType
  - `Full`: automatically enables and disables MessageOptions for each type listed above.
  - `Enable`: works like Full, but does not disable MessageOptions.
  - `None`: disables validation.
- You can also add missing events to this class.

Fields and properties may vary between Space Ad Mobile plugin versions.

Additional AdMob Features

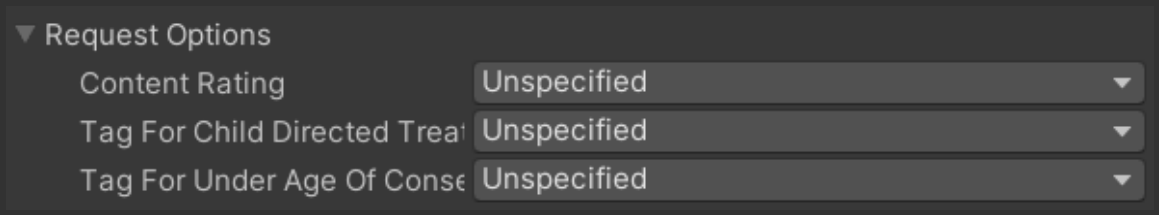
`RewardedInterstitialAdUnit`: Use the *[RewardedInterstitial Ad]* prefab for full-screen ads that can grant a reward. Inform the player about the reward before showing the ad, and grant it only after confirmation from the platform. Google documentation.

`NativeOverlayAdUnit`: Use the *[NativeOverlay Ad]* prefab when you need a native ad rendered as a customizable platform overlay. SpaceAdMobile renders the template after load; Hide() hides it and Show() displays it again. Google documentation.

`CollapsibleBanner.Enabled`: Turn this on in Banner Options to request a collapsible banner. The adapter supports top and bottom anchors; an eligible ad is not guaranteed on every request. IsCollapsible checks the loaded banner when available and otherwise reflects the configured option. Google documentation.

Useful Tips:

For the first launch, consider an age-selection screen to configure ad requests. You can then populate the following fields and save the settings in your app.
https://developers.google.com/admob/unity/targeting



This allows SpaceAdMobile to initialize with RequestOptions already configured:

Prefab initialization:

```
[SerializeField]
private SpaceAdMobile spaceAdMobilePrefab;
```

Then set the parameters:

```
public void InitiateMobileSDK()
{
    var spaceAdMobile = Instantiate(spaceAdMobilePrefab.gameObject)
        .GetComponent<SpaceAdMobile>();
    spaceAdMobile.RequestOptions = new RequestOptions
    {
        //PlayerPrefs here only for example
        ContentRating = (ContentRating)
            PlayerPrefs.GetInt("ContentRating", 3),
        TagForUnderAgeOfConsent = (TagForUnderAgeOfConsent)
            PlayerPrefs.GetInt("TagForUnderAgeOfConsent", -1),
        TagForChildDirectedTreatment = (TagForChildDirectedTreatment)
            PlayerPrefs.GetInt("TagForChildDirectedTreatment", -1)
    };
}
```

Good luck integrating ads into your project!

We hope this guide helps you set up ads smoothly.

If you have any questions, please feel free to contact us:

info@space-cat.games

Ad platform interfaces, APIs, and SDKs change over time. This guide may not reflect every update yet.