Space Ad Mobile

Yandex Ads

Unity Integration Guide



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

OV App Open ads	Ban ner view Banner ads
InV Interstitial ads	RV Rewarded ads

Contents

1. Prepare the Ad Platform 3
2. Creating Assets and Setting Up Scenes 5
3. Example Walkthrough and Ad Formats 8
   App Open ads 8
   Banner ads: adaptive inline and sticky banner 9
   Interstitial ads 9
   Rewarded ads 10
   Ad Callbacks and Reward Delivery 10
   Rewarded Ad Example 12
   Preventing Banners from Obscuring the UI 14
4. API reference and settings 16
   Asynchronous loading, targeting, and debugging 17
   Useful tips 17

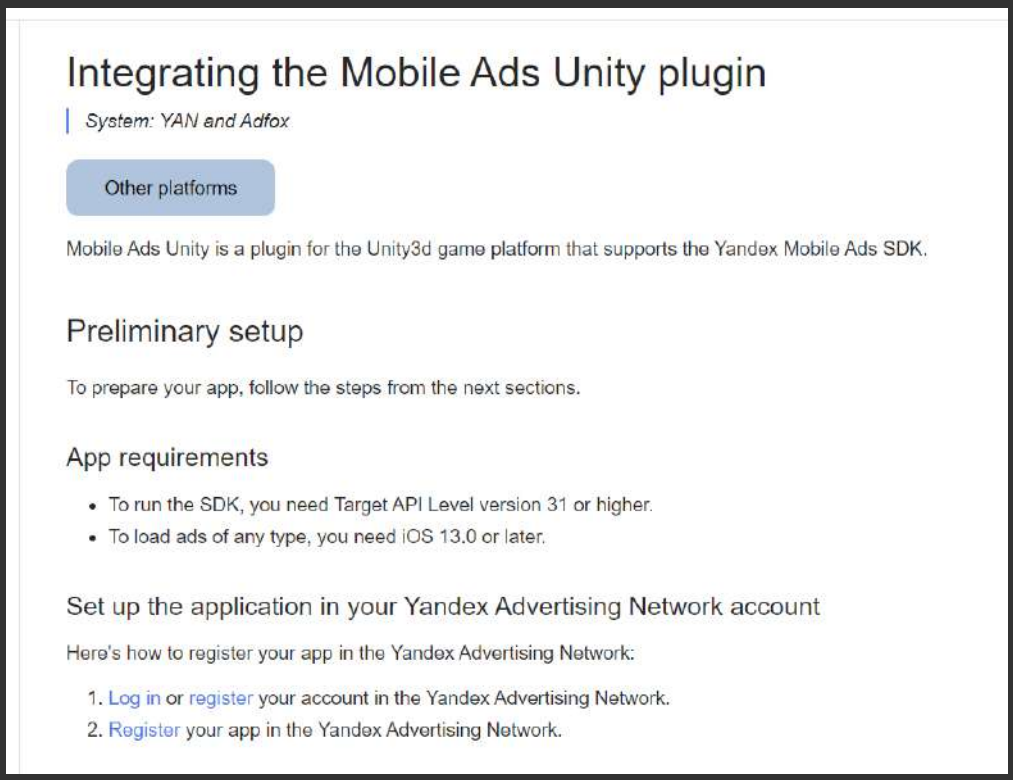
> *Space Ad Mobile* makes ads easy to add into your mobile game: one unified workflow for LevelPlay, AdMob, Yandex.Ads across banners, interstitials, rewarded. Ship faster with less setup. No code required.

1. Prepare the Ad Platform

Step-by-step installation for initial setup:

(Or you can skip this item if you have already set up Yandex Ads.)

1. Follow the Yandex Ads setup guide: https://yandex.ru/support2/mobile-ads/en/dev/unity/quick-start.



Integrating the Mobile Ads Unity plugin

System: YAN and Adfox

Other platforms

Mobile Ads Unity is a plugin for the Unity3d game platform that supports the Yandex Mobile Ads SDK.

Preliminary setup

To prepare your app, follow the steps from the next sections.

App requirements

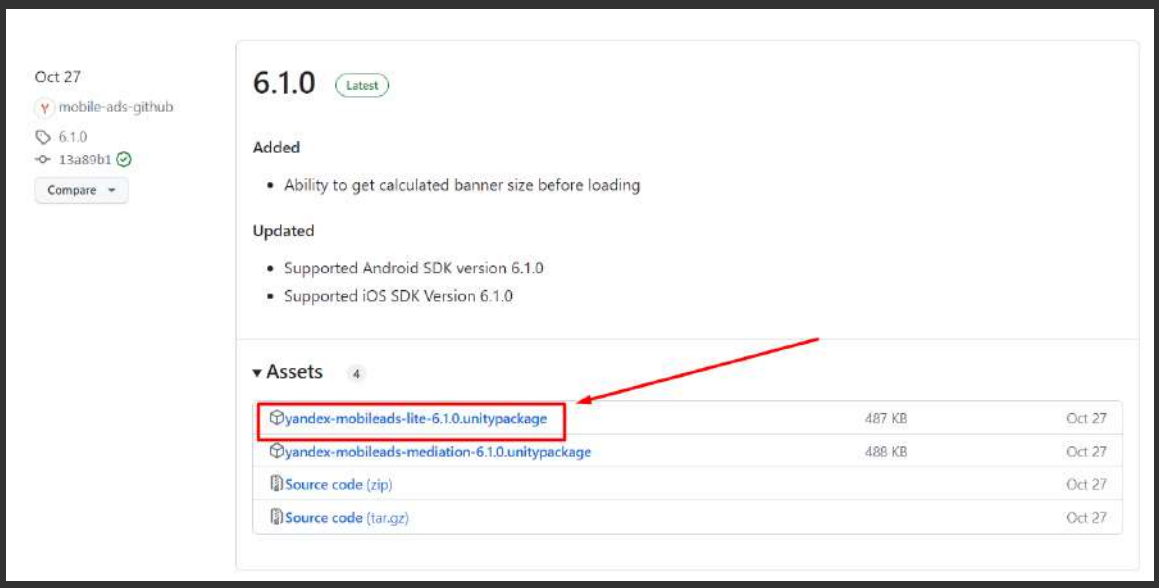
- To run the SDK, you need Target API Level version 31 or higher.
- To load ads of any type, you need iOS 13.0 or later.

Set up the application in your Yandex Advertising Network account

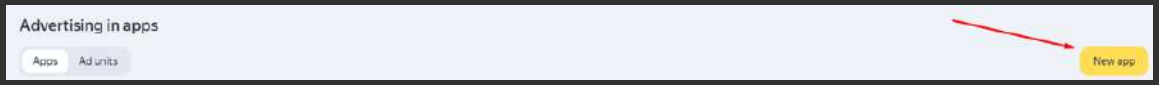
Here's how to register your app in the Yandex Advertising Network:

1. Log in or register your account in the Yandex Advertising Network.
2. Register your app in the Yandex Advertising Network.

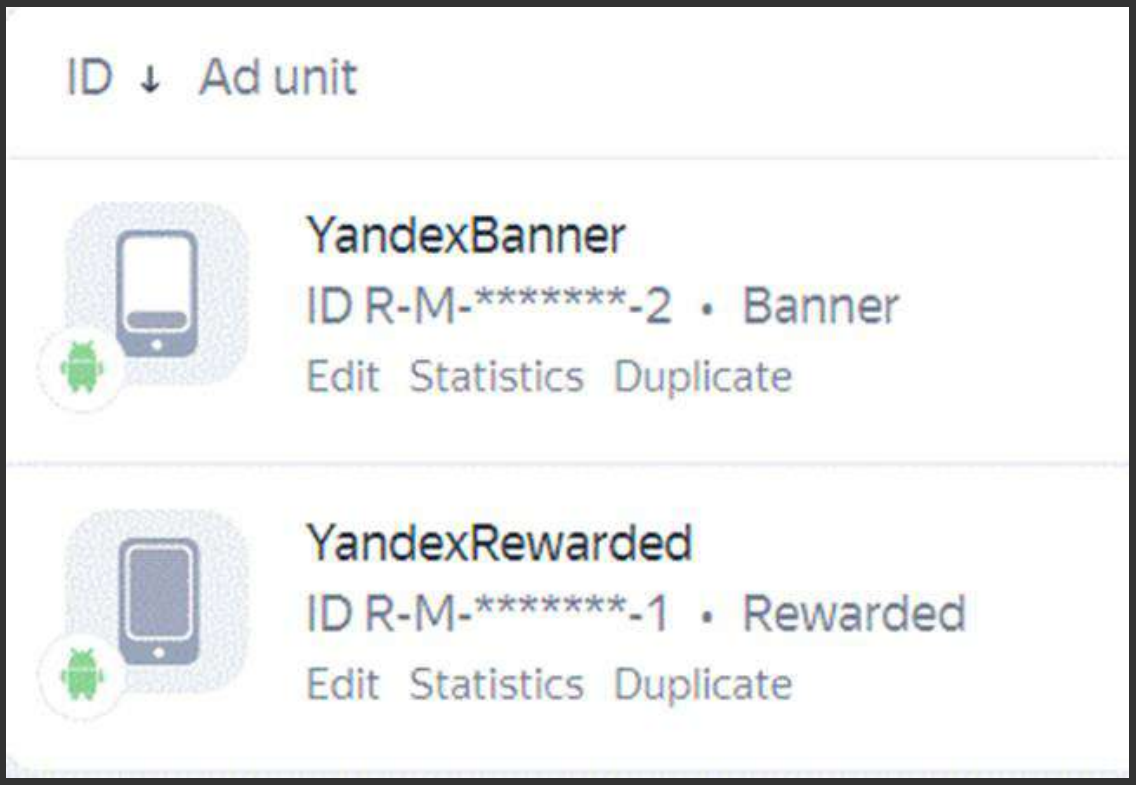
2. Download and install the Unity plugin in your project using the button: https://github.com/yandexmobile/yandex-ads-unity-plugin/releases.



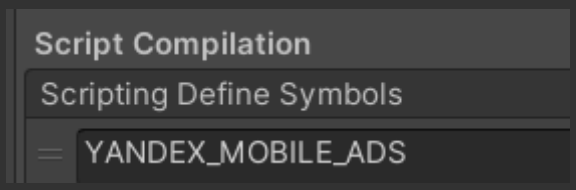
3. Register an account: https://partner.yandex.com/
4. Add an app.



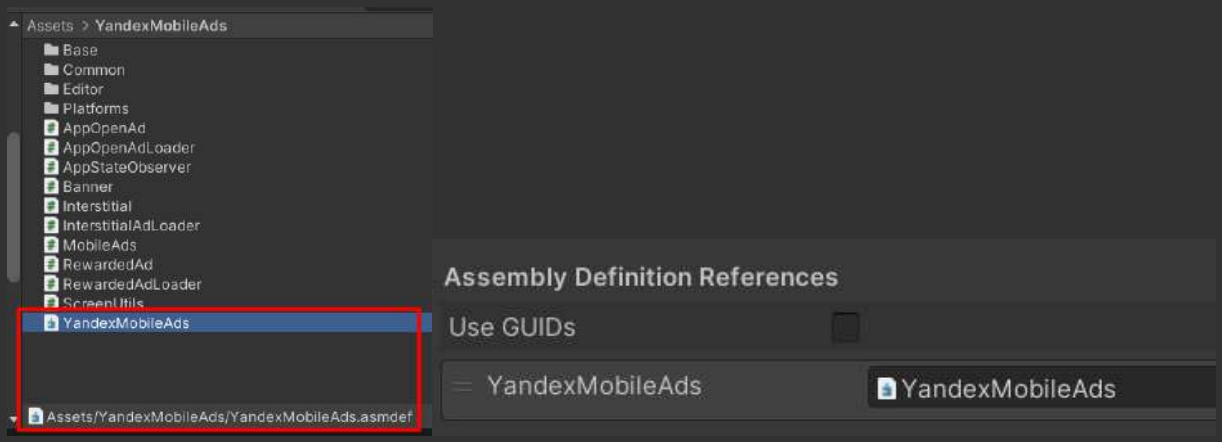
5. Add an ad block and copy its ID.



6. Go to the project settings and add the preprocessor define: Edit -> Project Settings -> Player -> Other Settings -> add YANDEX_MOBILE_ADS



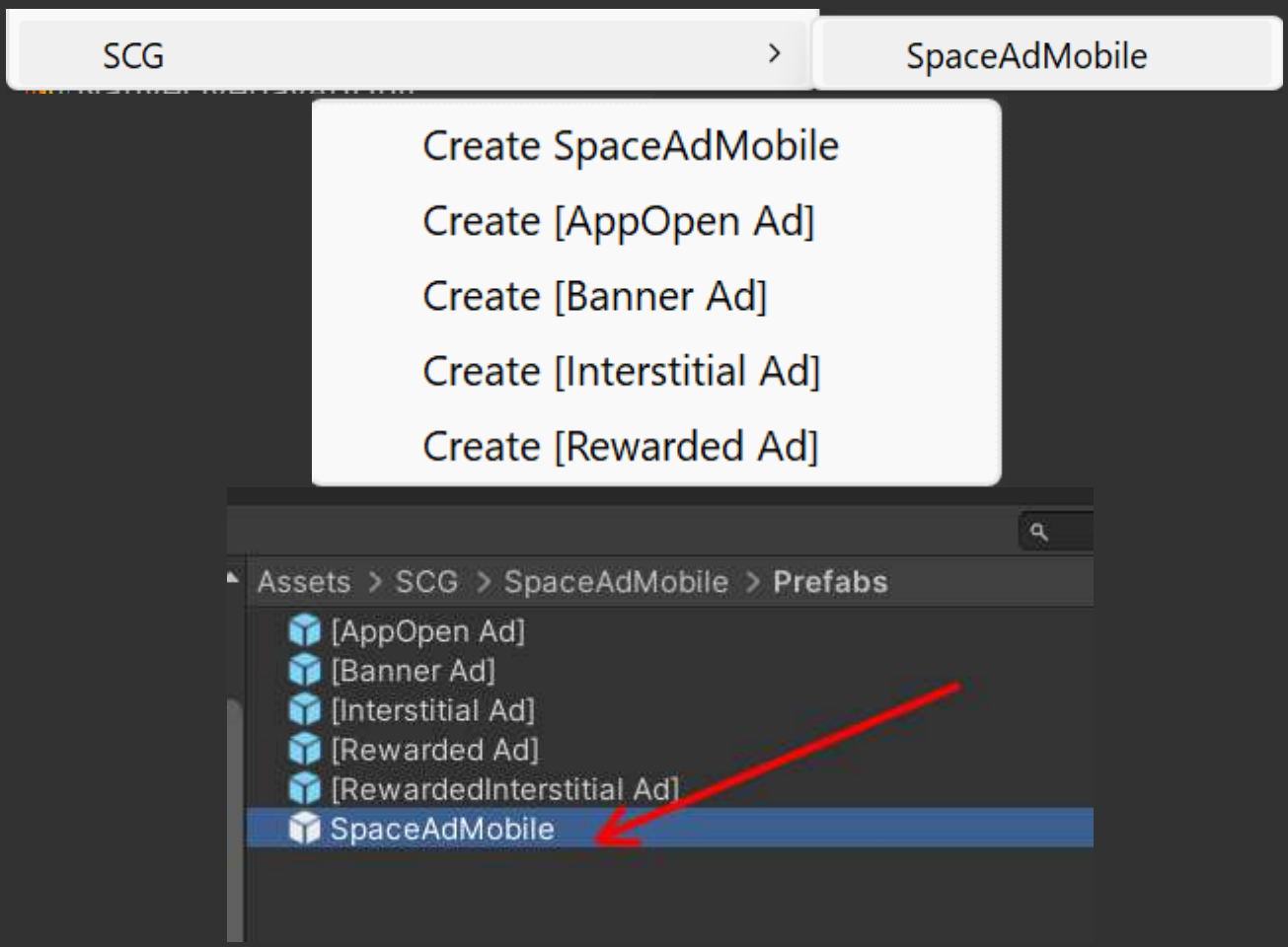
7. Make sure that the asmdef required for the asset to work has been created:



2. Creating Assets and Setting Up Scenes

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

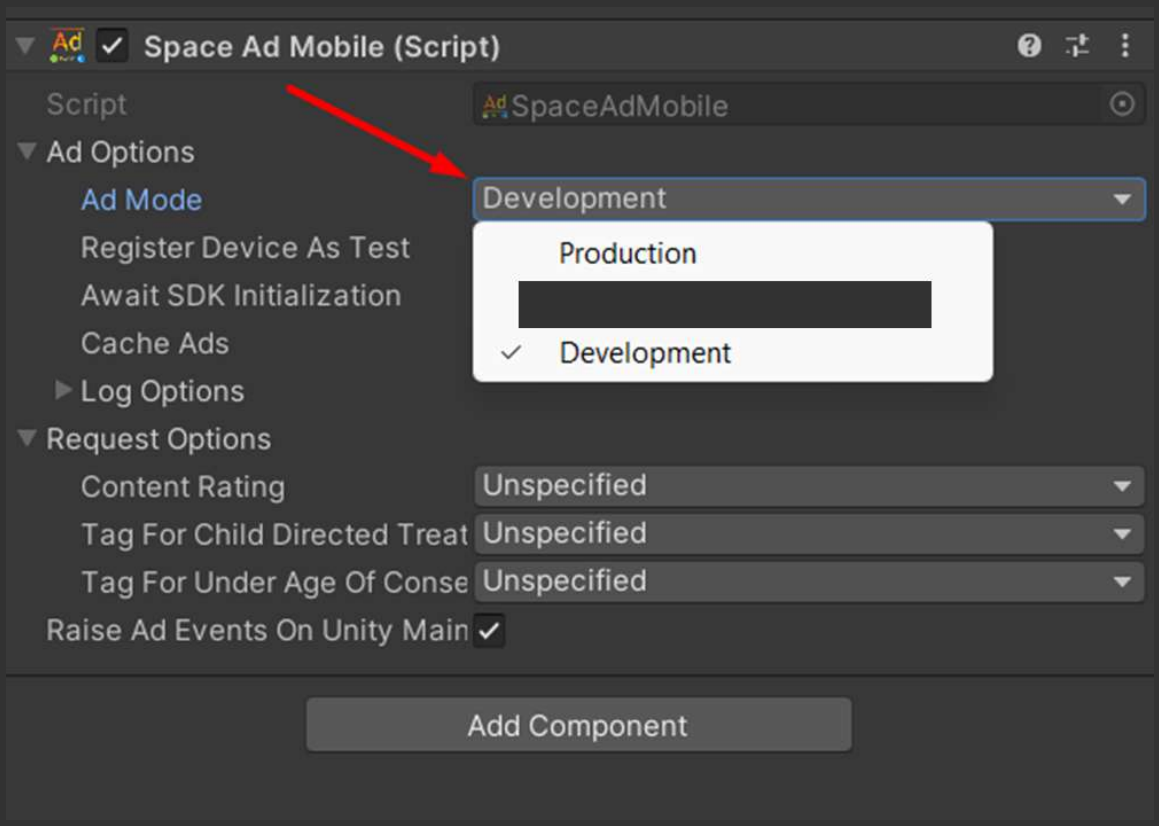
1. Open the preload scene that runs before the rest of your game. Open the context menu and add the required components.



2. This prefab is not required to initialize ads, but it stores the general settings required for the asset to work.

Choose whether you want to display test ads or live ads.

Read more about test ads: https://yandex.ru/support2/mobile-ads/en/dev/unity/formats



3. Open the scene from which the ad units will be loaded.

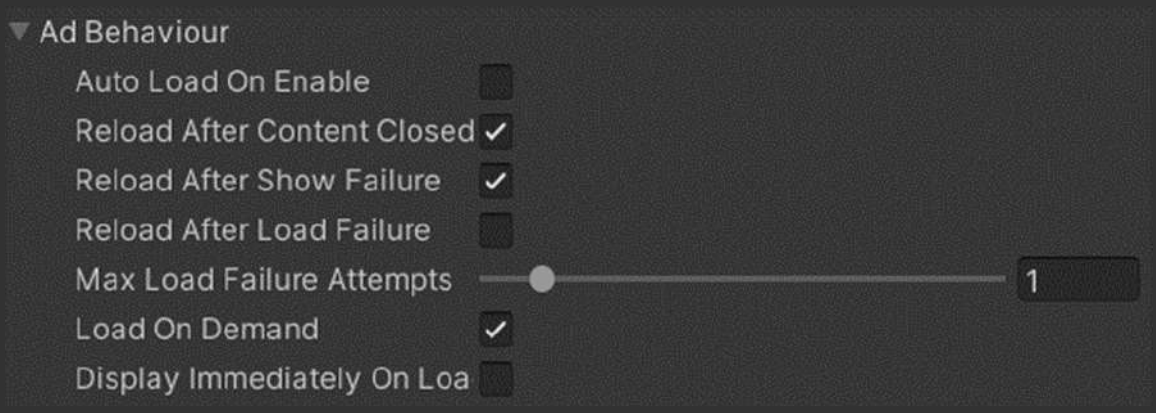
Drag the prefab with the required ad type into the scene.



4. The main part of each ad is the identifier you receive when you create the ad unit. Paste it for the target platform.

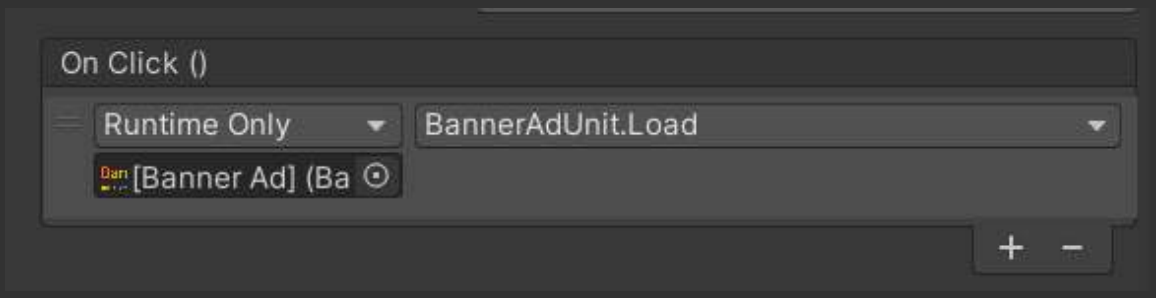


5. One of the main advantages of *Space Ad Mobile* is that it automates ad loading and display. We recommend exploring the options in the Ad Behaviour section:



The parameters will be discussed in more detail in the following paragraphs.

6. Then add a load event (in this case, for a banner) to the button:



3. Example Walkthrough and Ad Formats

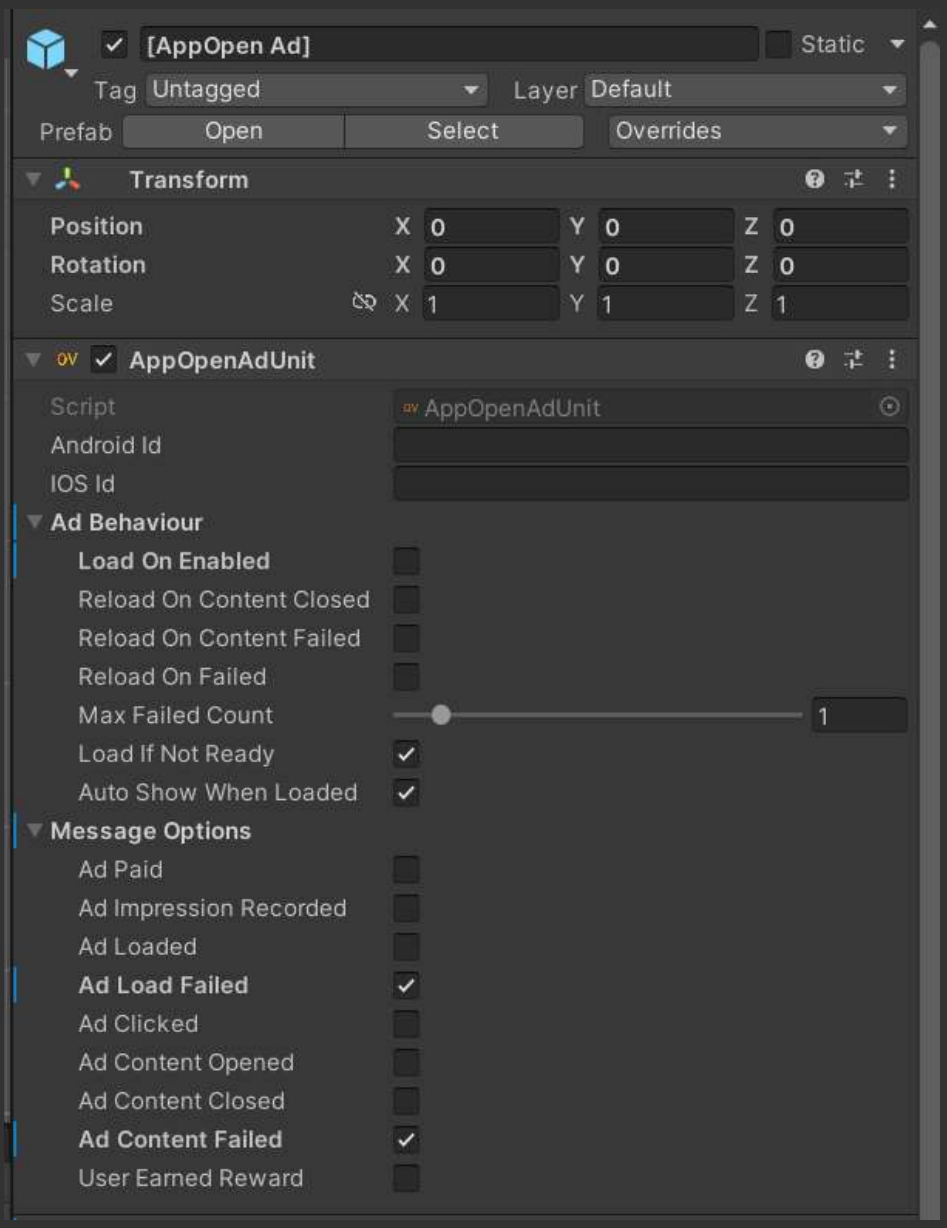
As an example, we created two scenes: Loading Scene and SpaceAdMobile Demo, where ads are loaded. Let's take a closer look at the [Ad Units] game object.

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.

We use the *[AppOpen Ad]* prefab and the `AppOpenAdUnit` component for this type.

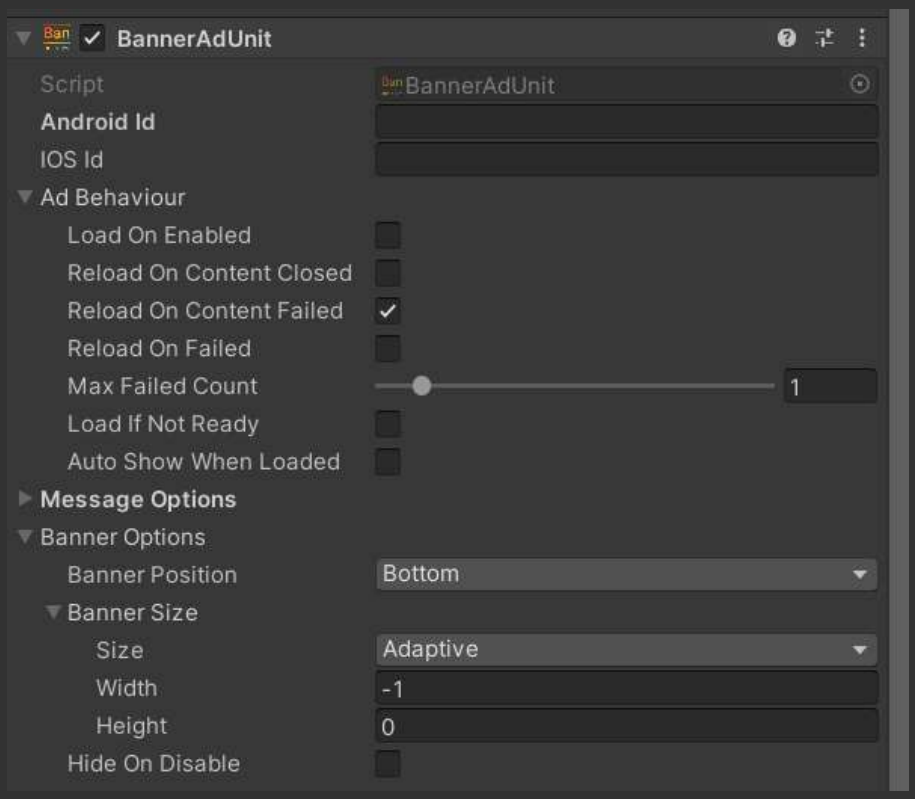
You can (and generally should) place it in the loading scene alongside the initialization prefab.



Banner ads: adaptive inline and sticky banner

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.

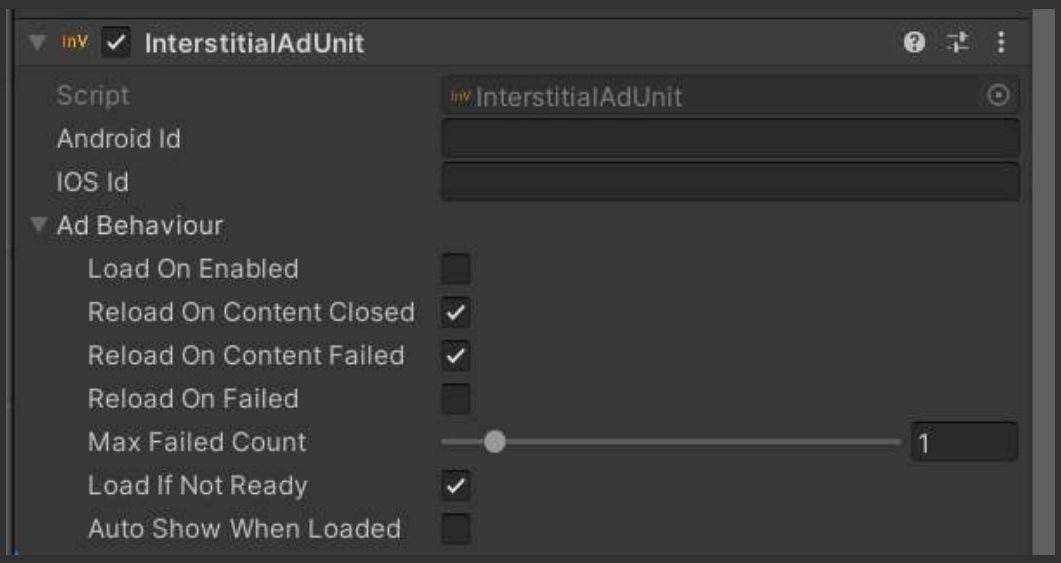
For this type, *use the [Ba*nner Ad] prefab *and the Bann*erAdUnit component.



You can also customize the location of the banner.

Interstitial ads

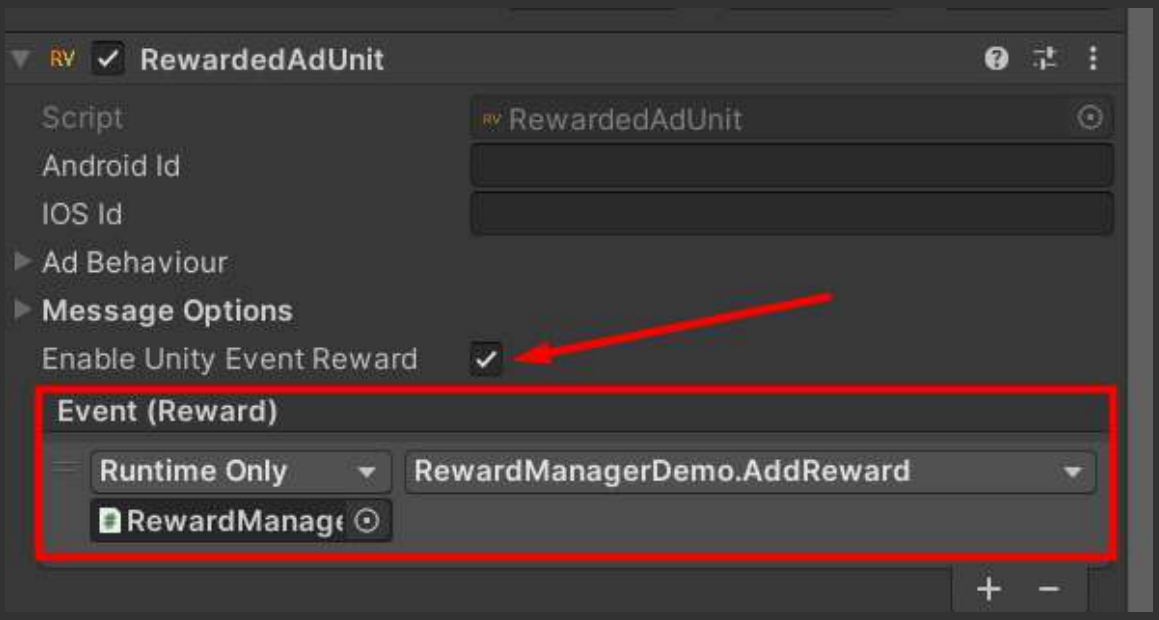
This is a full-screen ad that covers the app interface. It is usually displayed at natural transition points in the app flow, such as during a pause between levels in a game. When the app shows an interstitial ad, the user can either click the ad and go to the destination or close it and return to the app. For this t*ype, use the [Inter*stitial Ad] pre*fab and the Interst*itialAdUnit component.



Rewarded ads

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

For this type, use the [Rewarded Ad] prefab and the RewardedAdUnit component.



`Note`: To use the event in Unity, you must enable the event flag.

To connect mediation, download the adapters:
https://yandex.ru/support2/mobile-ads/en/dev/unity/third-mediation

Ad Callbacks and Reward Delivery

Override a callback in a component derived from the ad unit. Call base to keep the ad state and configured events up to date.

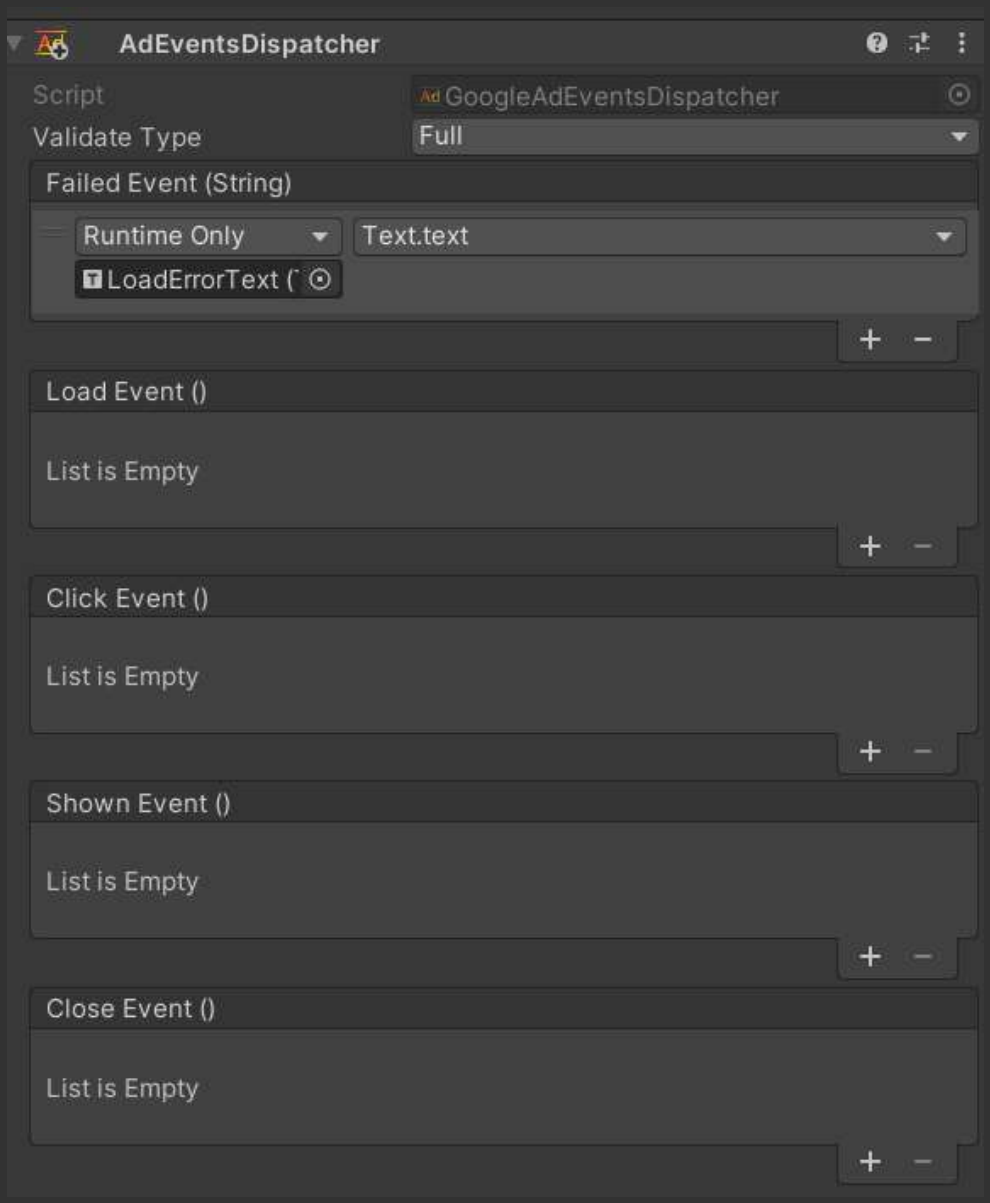
```
using UnityEngine;
using SCG.SpaceAdMobile;
using System;

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

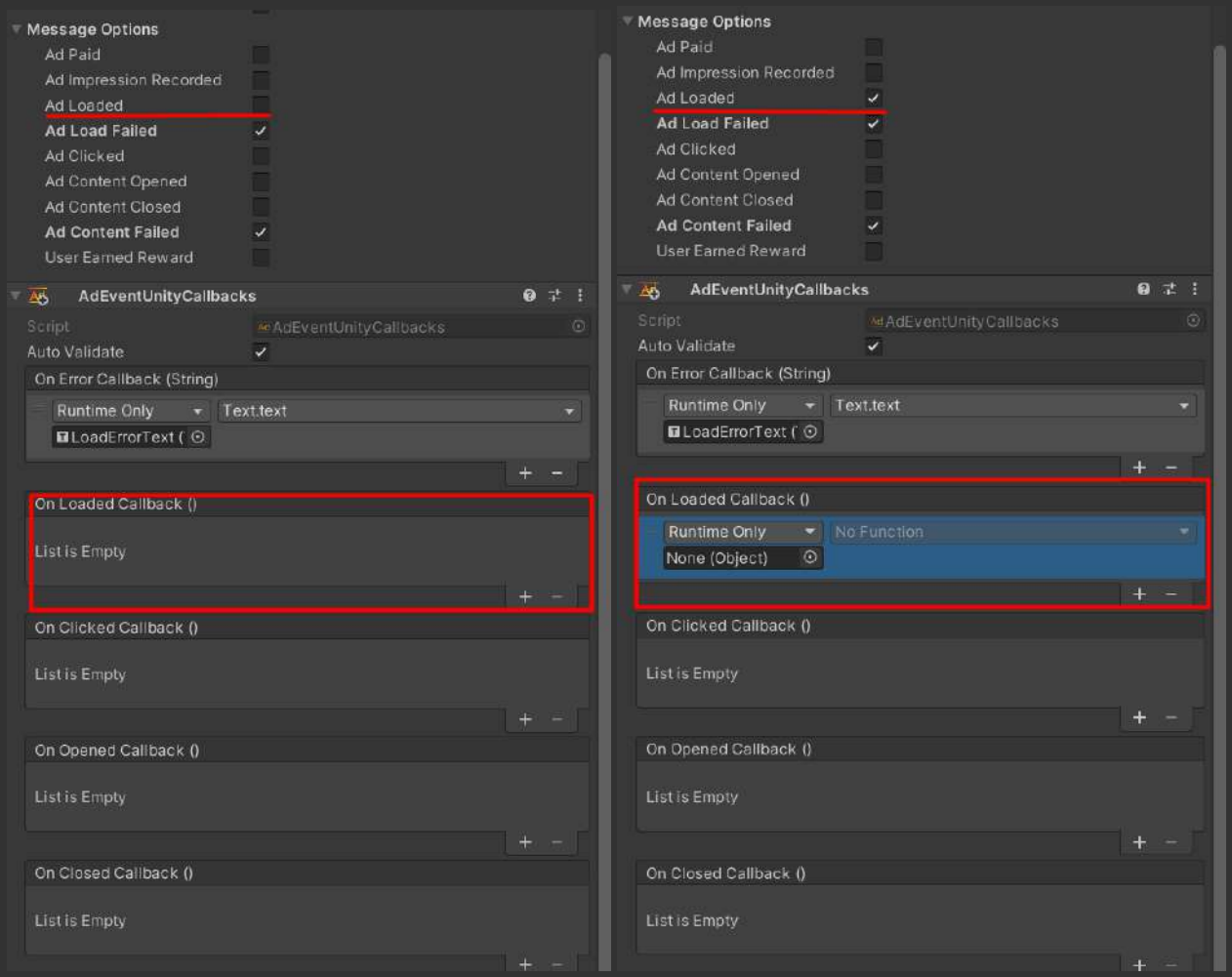
    protected override void OnAdShown(object sender, EventArgs args)
    {
        base.OnAdShown(sender, args);
        Debug.Log("Rewarded ad shown");
    }
}
```

Use the inherited Load() and Show() methods to load and display the ad. `OnAdLoaded` runs after loading; `OnAdShown` runs when the ad appears on screen.

For setup in the Inspector, add `AdEventsDispatcher` and connect your handlers to its events.



With *AutoValidate* enabled, adding an event listener automatically enables the corresponding interface callback in MessageOptions. For example, adding a listener to `OnLoadedCallback` automatically enables *MessageOptions -> AdLoaded*.



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 the ad platform settings. Add balance persistence and UI updates for your game.

Alternatively, subscribe the same handler to OnRewarded. Use one reward-granting route: using both the subscription and UnityEvent grants the reward twice.

Available callbacks:

`OnAdImpression(object, ImpressionData)`
Called when Yandex records an impression. sender is the event source; ImpressionData contains impression details, including rawData.

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

`OnAdLoadFailed(object, AdLoadingException) // current SDK`
Called when an ad fails to load. The error argument contains details of the failure. Use AdLoadingException with the current SDK, or AdFailedToLoadEventArgs with the legacy SDK.

`OnAdLoadFailed(object, AdFailedToLoadEventArgs) // legacy SDK`
Called when an ad fails to load. The error argument contains details of the failure. Use AdLoadingException with the current SDK, or AdFailedToLoadEventArgs with the legacy SDK.

`OnAdClicked(object, EventArgs)`
Called when the user clicks the ad. sender is the event source; EventArgs does not contain reward data.

`OnAdShown(object, EventArgs)`
Called when the ad appears on screen. Use it for actions at the start of a display; reward confirmation arrives separately. sender is the event source; EventArgs does not contain reward data.

`OnAdClosed(object, EventArgs)`
Called after the ad closes. Use it to restore the game UI and resume gameplay. sender is the event source; EventArgs does not contain reward data.

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

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

Yandex does not support `OnAdPaid`.

Can I keep a banner visible throughout the game?

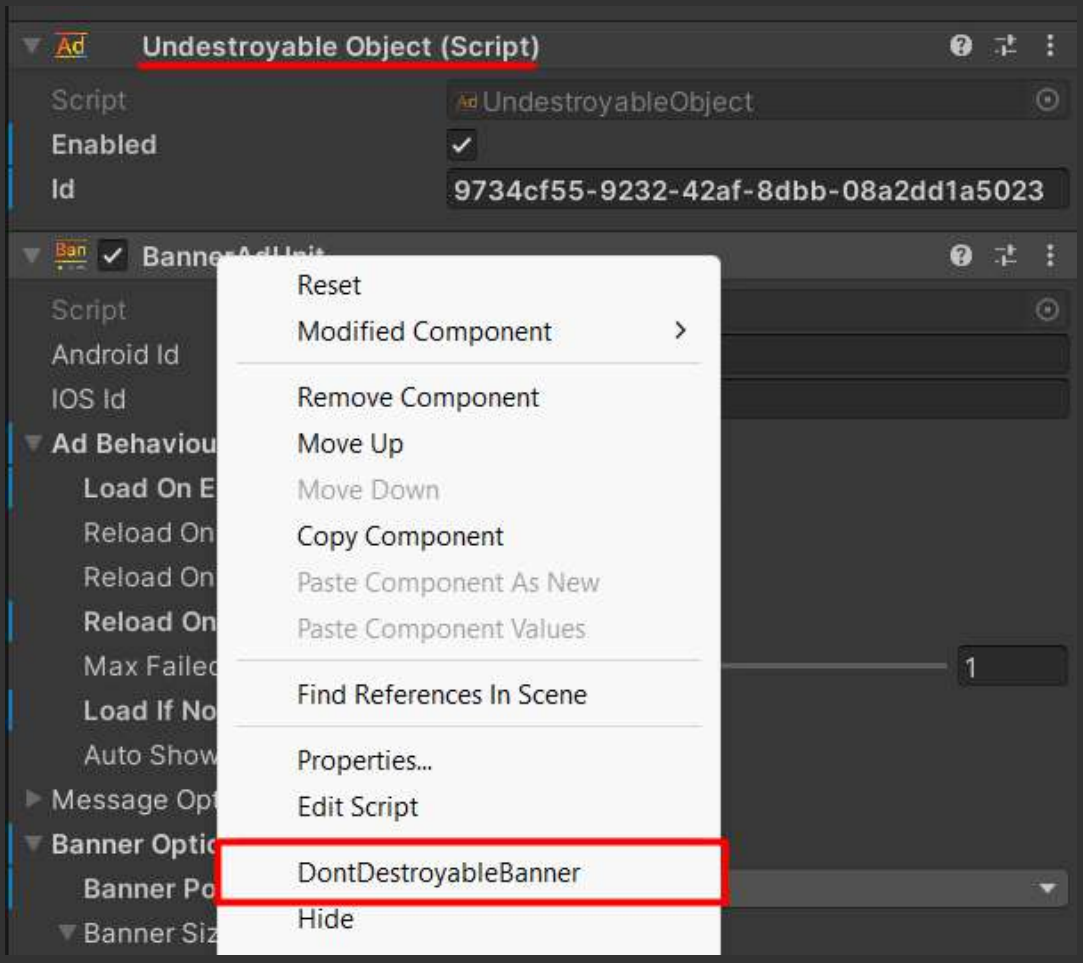
Absolutely. If your game switches between scenes, you do not have to lose the banner or control over it.

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.

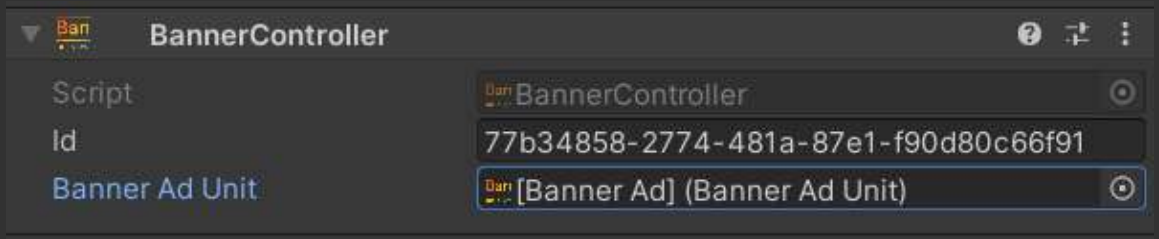
For such cases, use the *UndestroyableUtility* component, which allows you to control object destruction.

For quick creation, open *the BannerAdUnit* component and *use DontDestroyable*Banner. A component with a generated ID will be created.



If you need to copy the banner, delete the ID string to generate a new one.

Then place Bann*erController in t*he scene where the persistent ad will be 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 I 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.

Assign Banner Ad Component to the banner component that implements IBannerAd, usually `BannerAdUnit`. If the banner persists between scenes through a `BannerController`, also assign Banner Ad Controller so the helper can keep working even when the current scene does not own the banner object directly. Assign a controller only when it already resolves the persistent banner, because once a controller is present the helper relies on that controller instead of scene-wide banner discovery.

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

- `AdGeneralOptions` - general behavior settings
  - `AdMode` - ad mode: development or production
  - `IsTestDeviceInStore` - the ad will still be shown as a test ad even if the app was downloaded from an online store
  - `LogOptions` - logging settings. Disable noisy ones.

*AdUnit

- `androidId` - ad unit ID for Android devices
- `iOSId` - ad unit ID for Apple devices
- `adBehaviour` - ad loading and display behavior
  - `AutoLoadOnEnable` - whether the ad unit will be loaded 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

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

  - `MaxLoadFailureAttempts` - number of attempts to load after an error
  - `LoadOnDemand` - use the Show method to load ads too! Isn't it beautiful?
  - `DisplayImmediatelyOnLoad` - display ads immediately after they are loaded; do not use with the other preload settings
- `messageOptions` - parameters for inclusion of callbacks in methods of I*Message interface. It is not recommended to include all parameters; it is better to use a less resource-intensive way through inheritance.
- `Load` - method for loading the ad
- `Show` - method for displaying the ad

*AdUnit*Handlers

You can safely override these methods in your scripts:

- `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` - triggered when a user receives a reward (Rewarded ad types)

AdEventUnityCallbacks

- 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 extend this class with any events you need that are not included by default.

Fields and properties may vary between Space Ad Mobile versions.

Differences may also result from the Yandex Ads adapter.

Asynchronous loading, targeting, and debugging

`LoadAsyncByDefault`: In the non-legacy Yandex SDK 8 path, enabling this makes the usual Load() call use asynchronous loading. LoadAsync() returns Task<bool>; await it and check the result before showing the ad.

`RequestOptions`: Set Age, Gender, ContextQuery, ContextTags, Location and Parameters on SpaceAdMobile.`RequestOptions`. Location is included only when Location.Enabled is true and location tracking consent has been granted. Each Parameters item contains Key and Value; empty keys are skipped and later duplicate keys replace earlier ones.

`SpaceAdMobile.ShowDebugPanel()`: On Android in the non-legacy Yandex SDK 8 path, call this during development to open the Debug Panel and inspect SDK, adapter and privacy state.

`Async controller methods:` `BannerController`.LoadBannerAsync(), InterstitialController.LoadInterstitialAsync(), and RewardedController.LoadRewardedAsync() return Task<bool> for callers that prefer async/await.

Yandex ad targeting
Yandex SDK changelog
Yandex Debug Panel

Useful tips

For the first run and initial request configuration, you can create a menu with age selection. After that, fill in the following fields and save them in the application. https://yandex.ru/support2/mobile-ads/en/dev/unity/coppa



Initializing the prefab:

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

Then set the values.

```
public void InitiateMobileSDK()
{
    var spaceAdMobile = Instantiate(spaceAdMobilePrefab.gameObject)
        .GetComponent<SpaceAdMobile>();
    //PlayerPrefs here only for example
    spaceAdMobile.AgeRestrictedUser =
        PlayerPrefs.GetInt("AgeRestrictedUser", 0) == 1;
    spaceAdMobile.UserConsent = PlayerPrefs.GetInt("UserConsent", 0) == 1;
    spaceAdMobile.LocationConsent =
        PlayerPrefs.GetInt("LocationConsent", 0) == 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.