Space Ad Mobile

Unity LevelPlay

Unity Integration Guide



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

Banner view	InV	RV
Banner ads	**Interstitial ads**	**Rewarded ads**

Contents

1. Initial Ad Platform Setup 3
2. Asset Creation and Scene Setup 5
3. Example Scene and Ad Formats 7
   Banner ads 7
   Interstitial ads 8
   Handling Ad Events and Granting Rewards 9
   Rewarded Ad Example 12
   Keeping the UI Clear of Banner Ads 14
4. API Reference and Settings 16
   LevelPlay Validation and Banner Control 17
   Useful tips: 18

> *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. Initial Ad Platform Setup

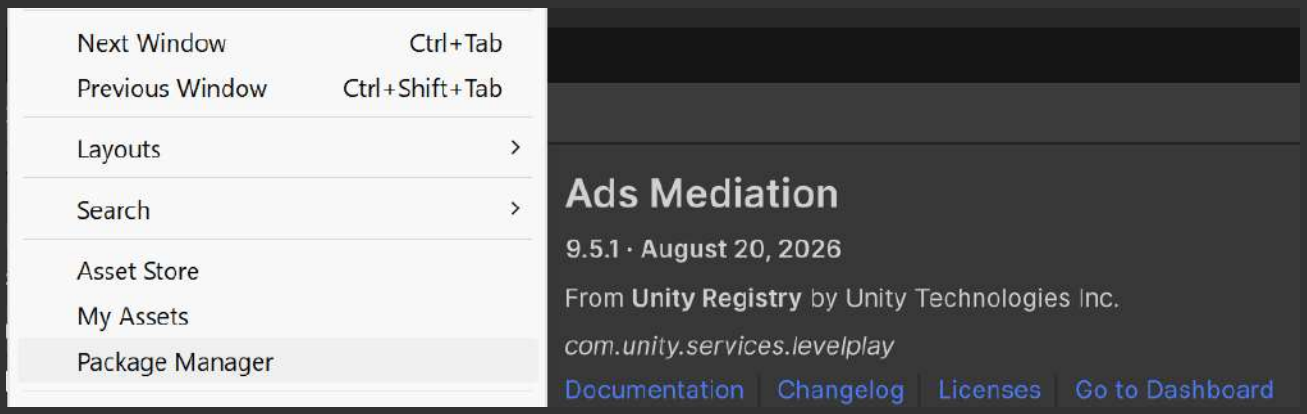
Step-by-step instructions for initial setup:

(You can skip this section if LevelPlay is already configured.)

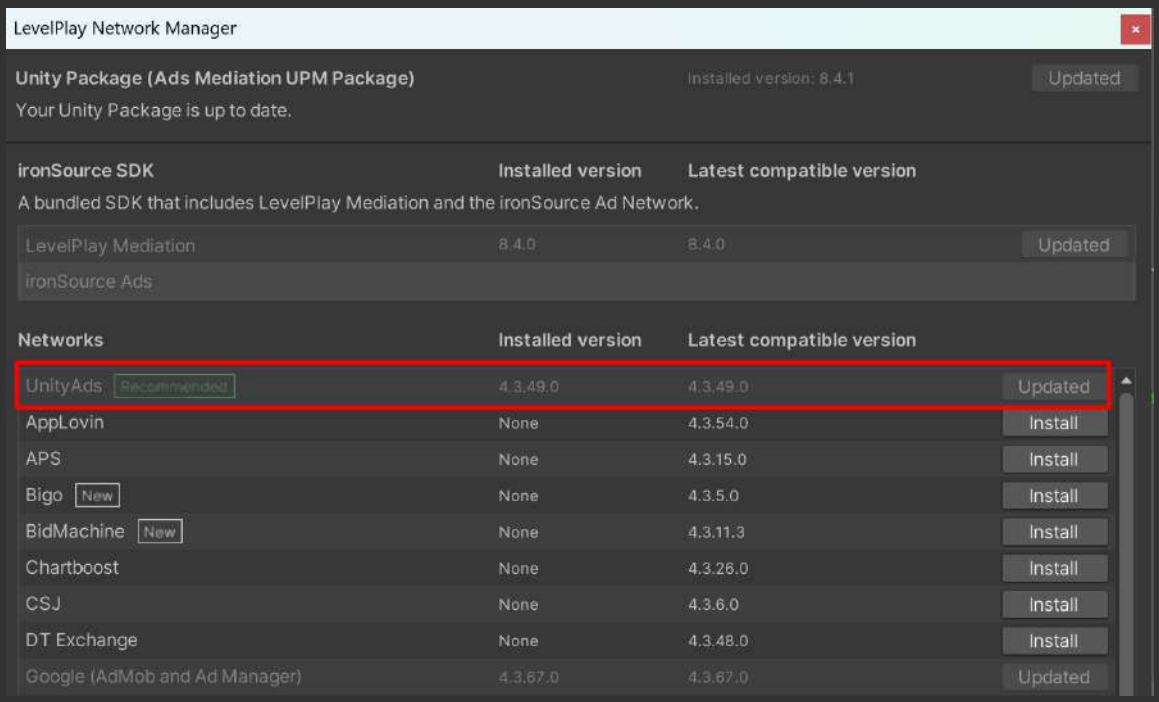
0. Set up your account by following the official guide:

https://docs.unity.com/en-us/monetization/getting-started/

1. Follow the official LevelPlay setup guide:
   https://docs.unity.com/en-us/grow/levelplay/sdk/unity/package-integration.
2. Install the Unity plugin through UPM (Window -> Package Management -> Package Manager).
   Search for Ads Mediation (*com.unity.services.levelplay*).

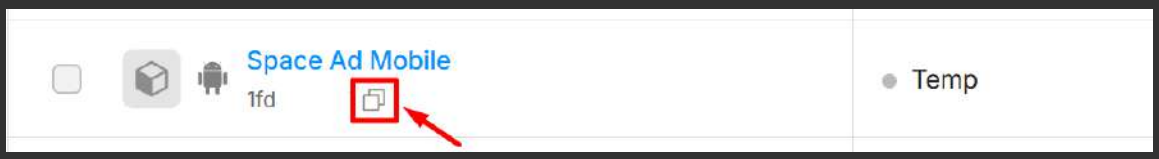


3. Install the Unity Ads mediation adapter and any other network adapters you need:

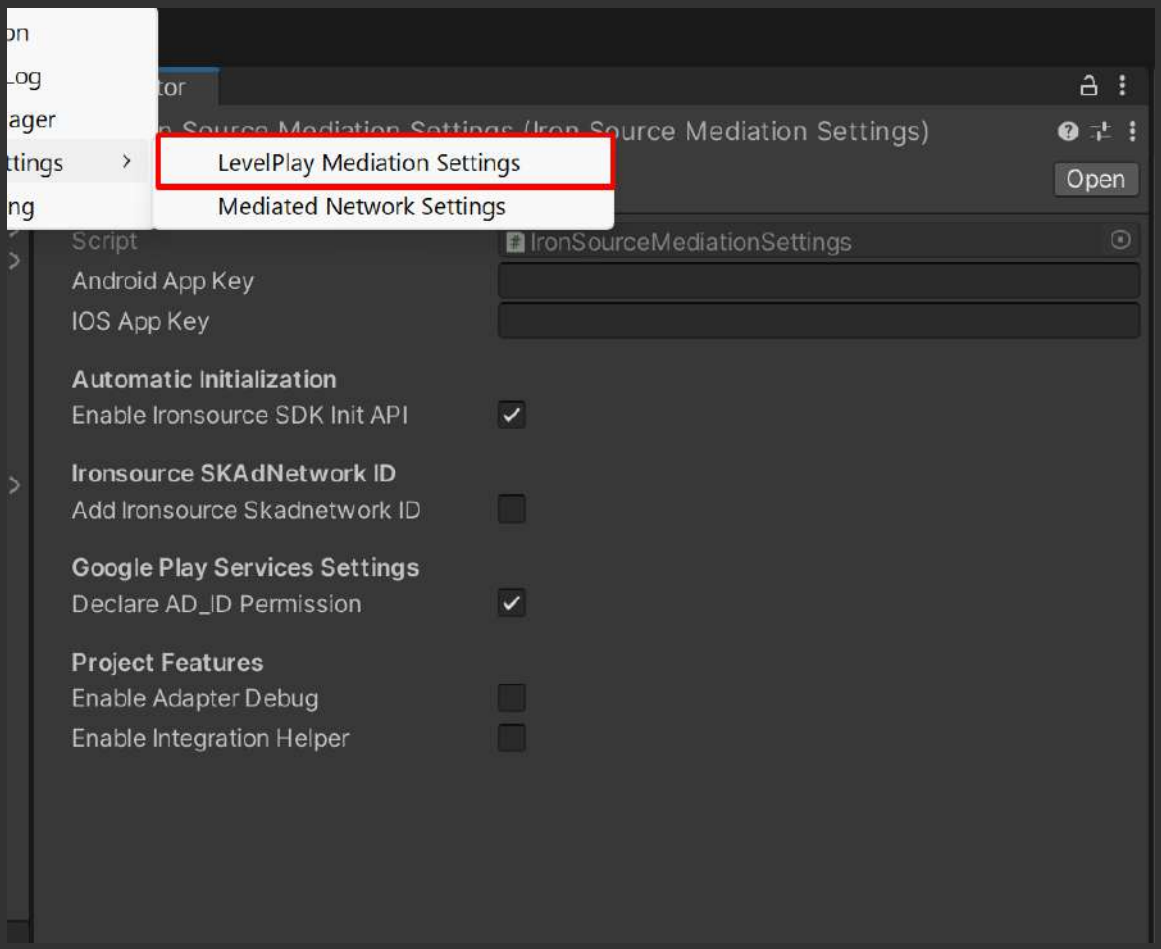


4. Copy your app ID.
   Open the following page:
https://platform.ironsrc.com/partners/next/mediation/apps

Then copy your app ID.



5. In the Unity Editor, open *Ad Mediation -> Developer Settings -> LevelPlay Mediation Settings* and enter your App Key.



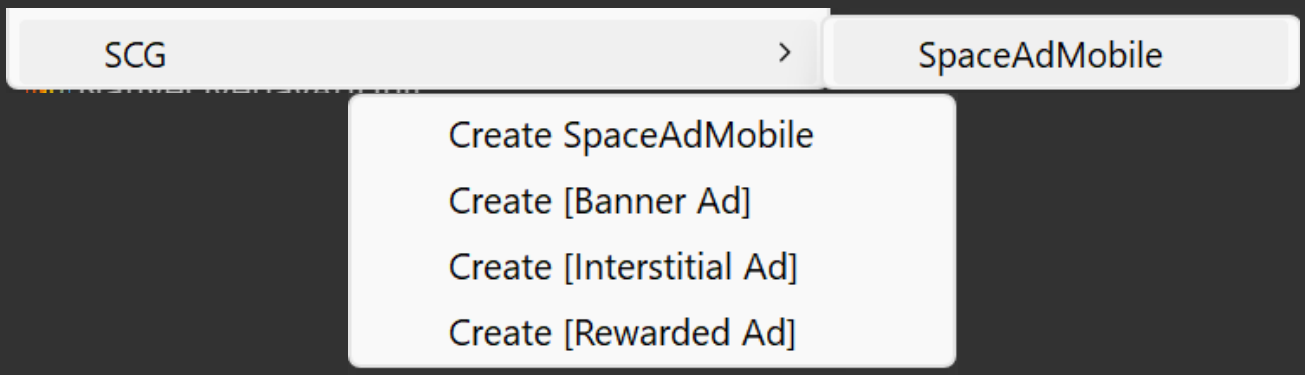
6. Create an ad unit and copy its ID.

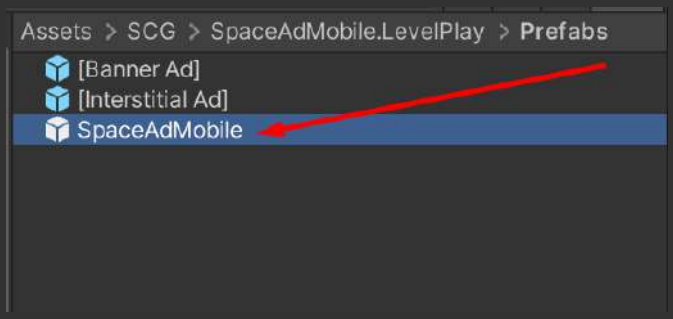


2. Asset Creation and Scene Setup

With your account configured and LevelPlay ready, you can now 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.

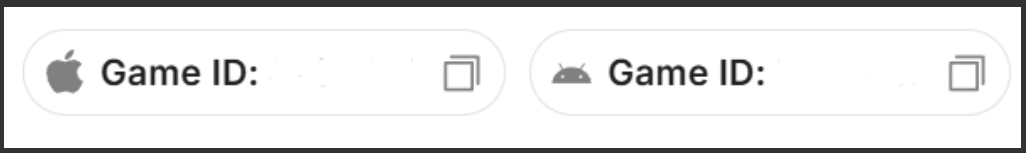




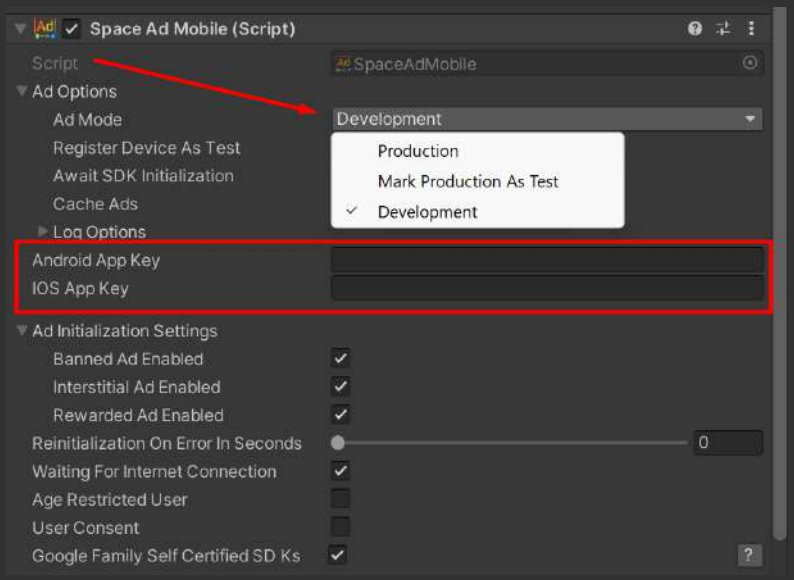
2. This prefab initializes the LevelPlay Mobile SDK.

Choose whether to use test ads or live ads.

Read more about test ads. Copy your App Key.

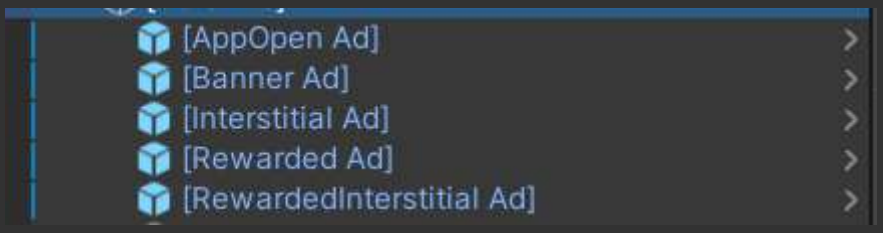


Enter the App Key to initialize Unity LevelPlay ads.



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

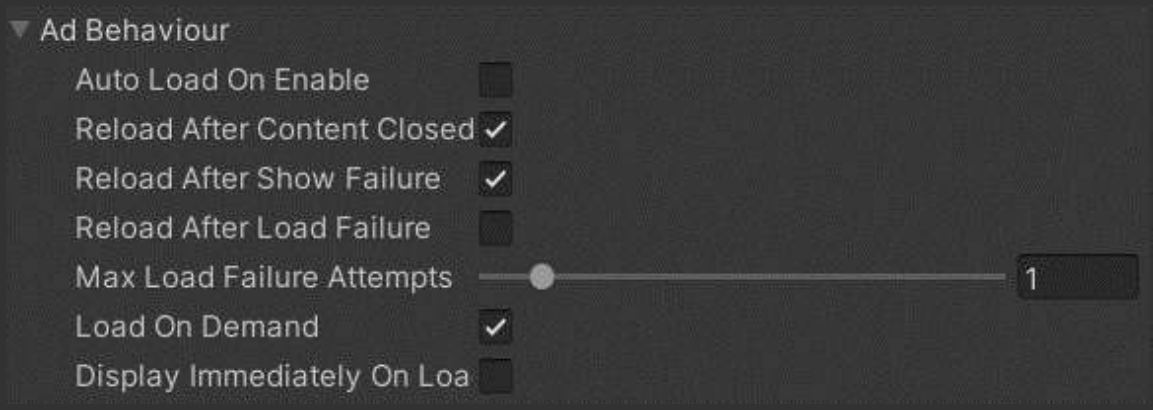
Drag the prefab for the required ad format into the scene.



4. Each ad unit needs the ID created in the LevelPlay dashboard. Enter the ID for the target platform.

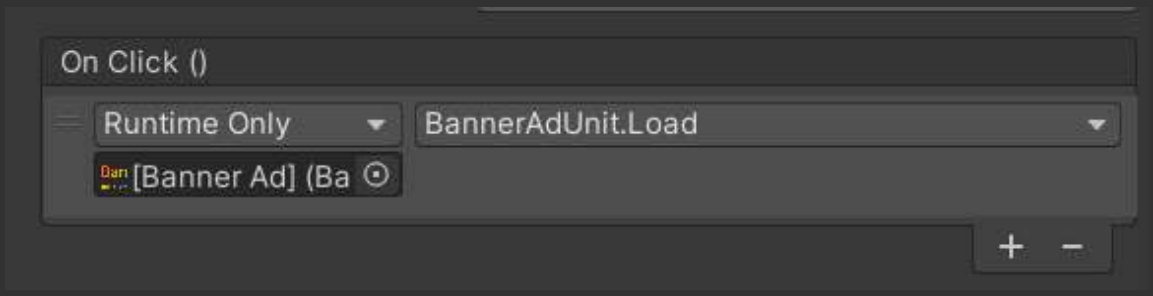


5. One of the main benefits of Space Ad Mobile is automated ad loading and display. We recommend exploring the Ad Behaviour settings:



These settings are explained in more detail in the following sections.

6. Then add an ad-loading event to your button (a banner in this example):



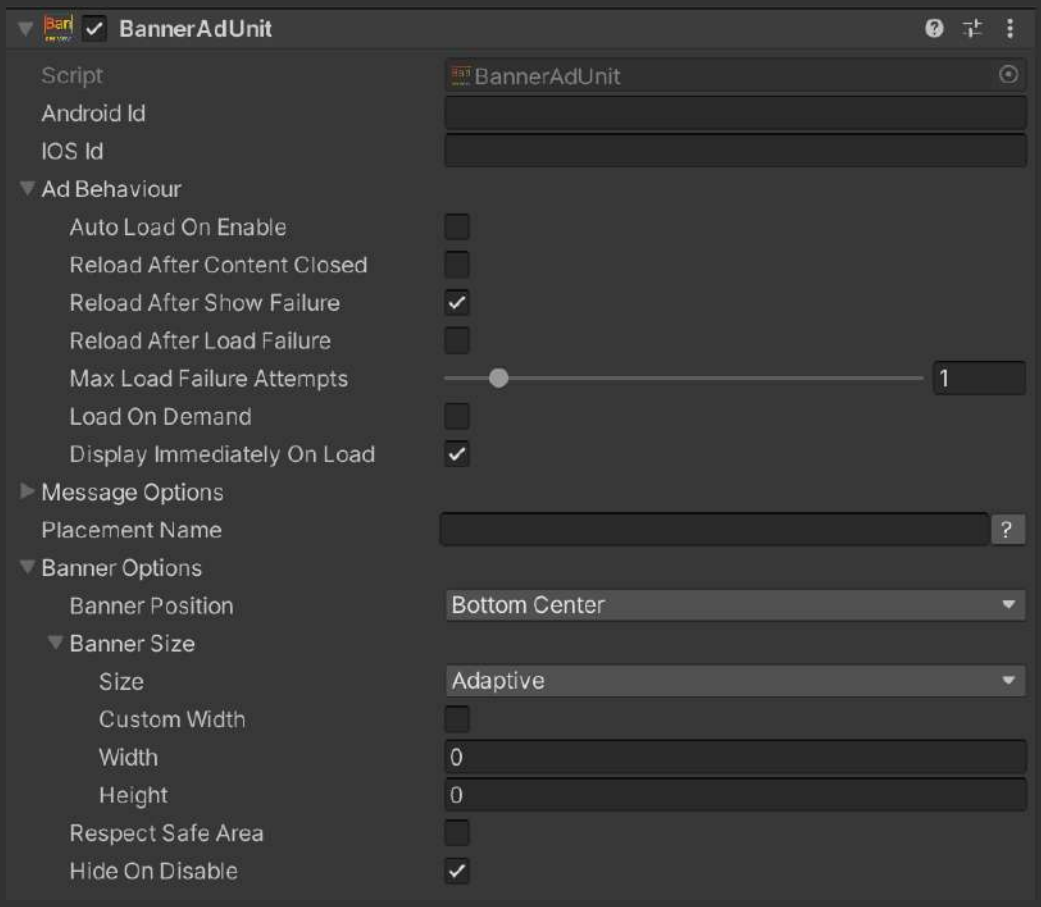
3. Example Scene and Ad Formats

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

Banner ads

Banner ads are rectangular image or text ads that occupy a fixed area of the screen. They remain visible as users interact with the app and may refresh automatically at set intervals.

Use the [Banner Ad] prefab and the `BannerAdUnit` component for this format.



You can also change the banner position. By default, an adaptive banner appears at the bottom of the screen. The supported banner sizes are listed below.

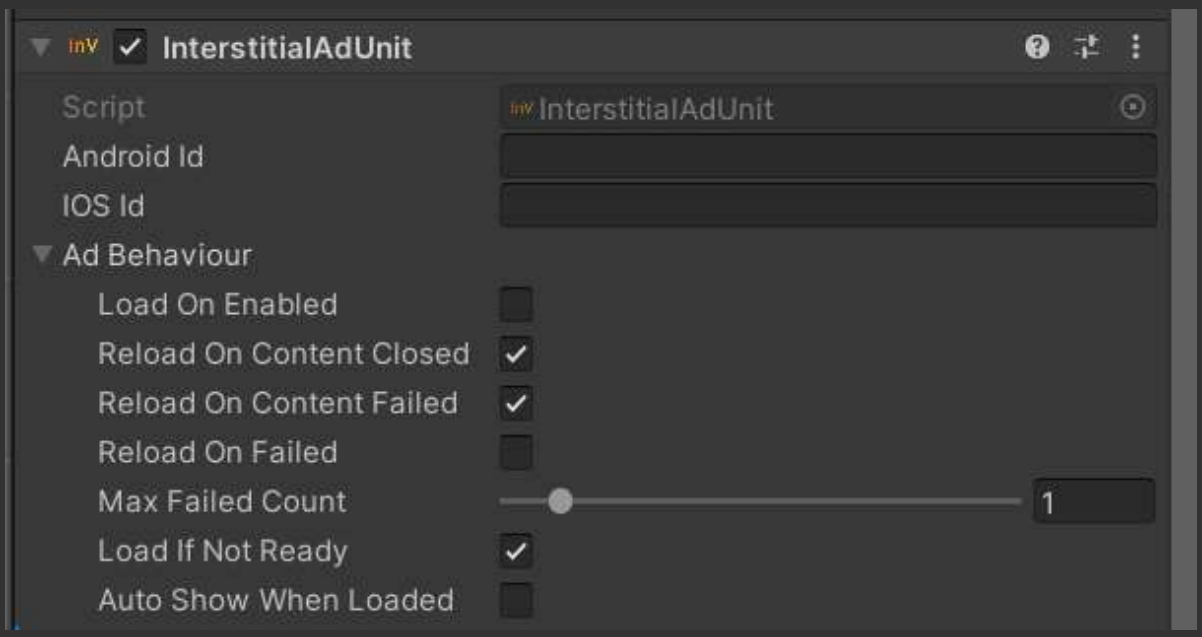
Banner sizes

LevelPlayAdSize	Description	Dimensions in dp (Width X Height)
BANNER	Standard banner	320 × 50
LARGE	Large banner	320 × 90
RECTANGLE	Medium Rectangular (MREC)	300 × 250
Adaptive	Automatically renders ads to adjust size and orientation for mobile & tablets (replaces "SMART" from previous indications)	device width X recommended height

Interstitial ads

Interstitial ads cover the app interface and are generally shown at natural breaks, such as between game levels. Users can tap the ad to open its destination or dismiss it and return to the app.

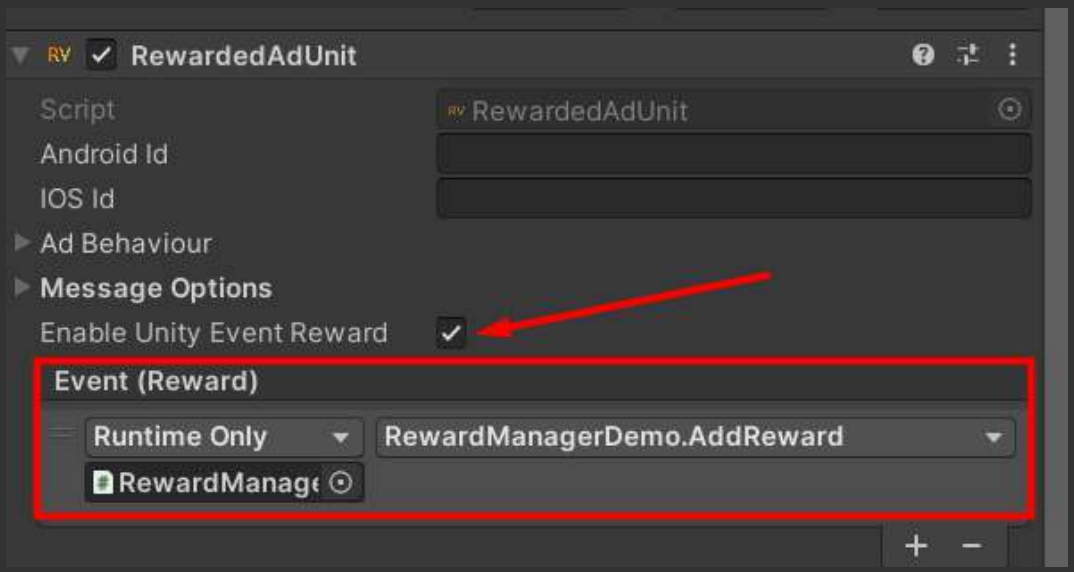
Use the [Interstitial Ad] prefab and the InterstitialAdUnit component for this format.



Rewarded ads

Rewarded ads let players receive in-app rewards in exchange for engaging with an ad.

Use the [Rewarded Ad] prefab and the RewardedAdUnit component for this format.



Important: Enable the corresponding event flag before using the event in Unity.

Handling Ad Events and Granting Rewards

Override the required callback in a subclass of the ad component. Call base to preserve the internal state updates and configured events.

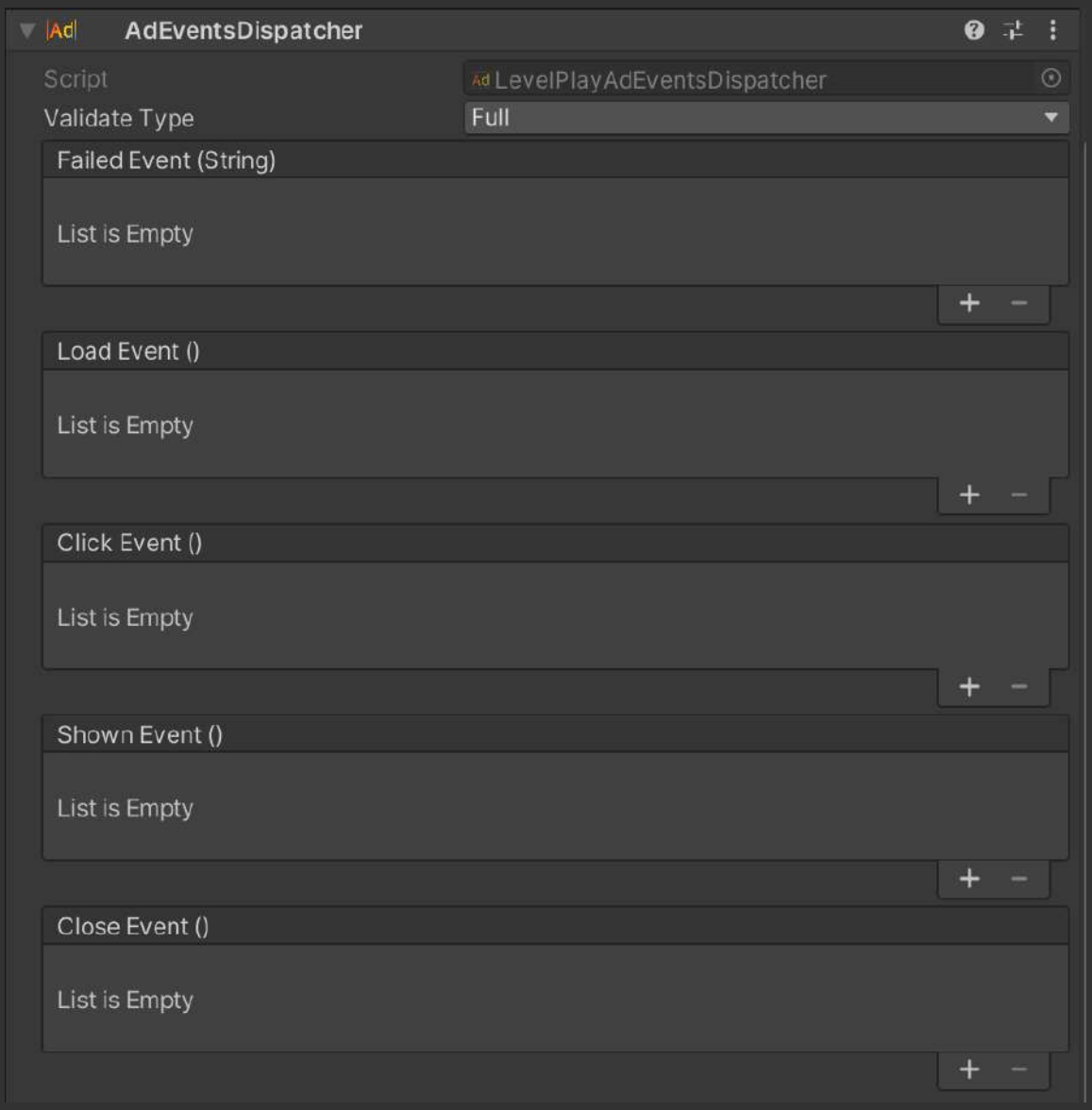
```
using UnityEngine;
using SCG.SpaceAdMobile;
using Unity.Services.LevelPlay;

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

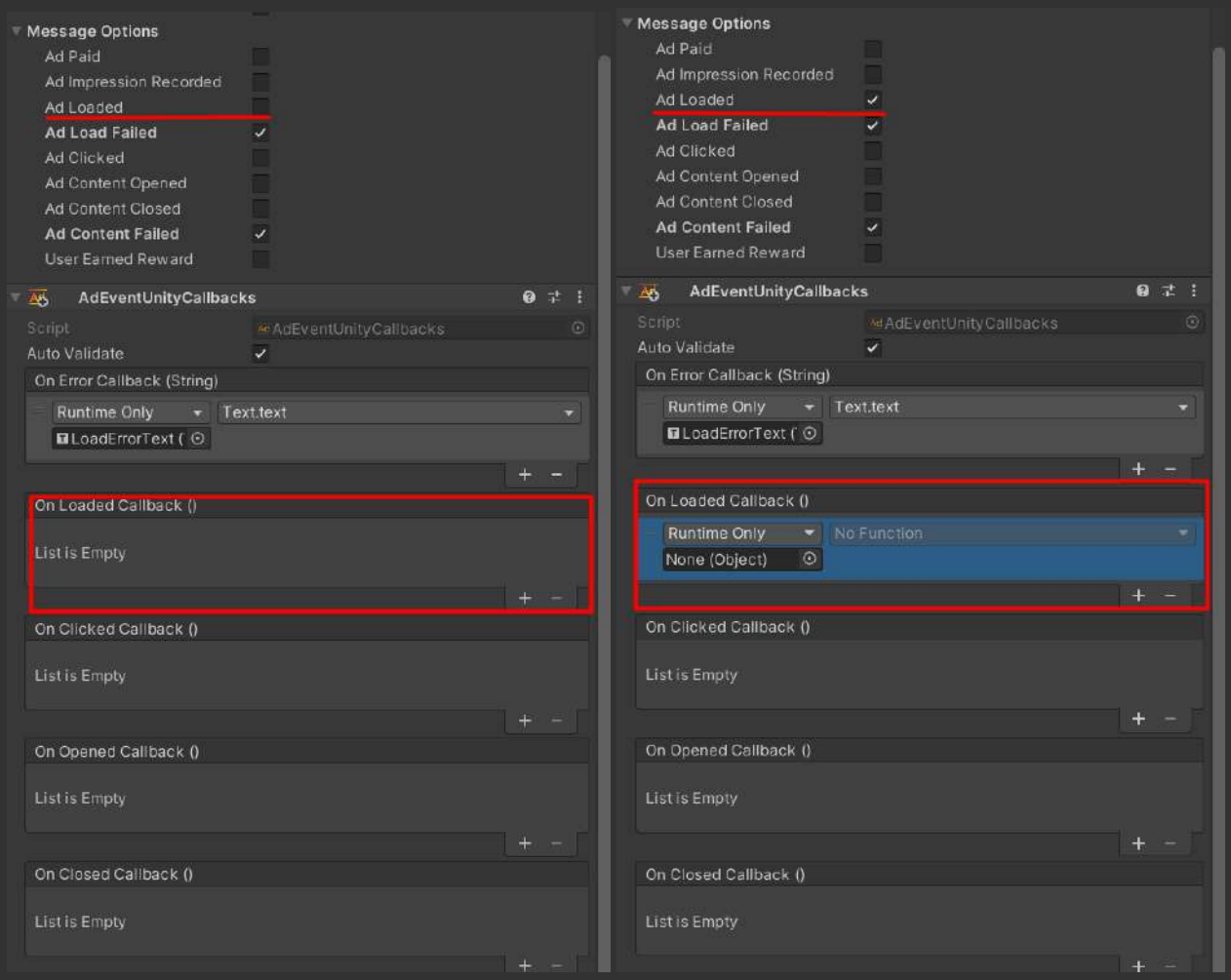
    protected override void OnAdShown(LevelPlayAdInfo adInfo)
    {
        base.OnAdShown(adInfo);
        Debug.Log("Rewarded ad shown");
    }
}
```

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

To configure callbacks in the Inspector, add `AdEventsDispatcher` and connect your handlers to its events.



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



Rewarded Ad Example

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

```
using UnityEngine;
using Unity.Services.LevelPlay;
using SCG.SpaceAdMobile;

public class GameRewardedAd : RewardedAdUnit
{
    private int coins;

    protected override void OnUserEarnedReward(
        LevelPlayAdInfo adInfo, LevelPlayReward reward)
    {
        base.OnUserEarnedReward(adInfo, reward);
        var adReward = new AdReward(reward);
        if (adReward.Name == "coins")
            coins += adReward.Amount;
    }
}
```

Set the reward name to `coins` in your LevelPlay settings. This method receives the confirmed reward and adds its amount to `coins`. Add balance saving and UI updates for your own game. When using this override, disable reward granting through `UnityEvent` to avoid granting the reward twice.

Available callbacks:

`OnAdInfoChanged(LevelPlayAdInfo)`
Called when ad information changes. `LevelPlayAdInfo` provides the latest details; the base method stores them in ResponseInfo.

`OnAdLoaded(LevelPlayAdInfo)`
Called when an ad loads successfully. You can update the UI and enable the Show button. `LevelPlayAdInfo` contains information about the ad.

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

`OnAdClicked(LevelPlayAdInfo)`
Called when the user clicks the ad. `LevelPlayAdInfo` contains information about the ad.

`OnAdShown(LevelPlayAdInfo)`
Called when the ad appears on screen. Use it for actions that should happen when display begins; reward confirmation arrives separately. `LevelPlayAdInfo` contains information about the ad.

`OnAdClosed(LevelPlayAdInfo)`
Called after the ad closes. You can restore the game UI and resume gameplay. `LevelPlayAdInfo` contains information about the ad.

`OnAdContentFailed(LevelPlayAdInfo, LevelPlayAdError)`
Called when a loaded ad cannot be displayed. The error parameter explains why. `LevelPlayAdInfo` contains information about the ad.

`OnAdImpressionDataReady(LevelPlayImpressionData)`
Called when impression revenue data becomes available. LevelPlayImpressionData includes impression and revenue details; calling base forwards the data to the shared package events.

`OnUserEarnedReward(LevelPlayAdInfo, LevelPlayReward)`
Called after LevelPlay confirms a reward. `LevelPlayAdInfo` contains ad details, while `LevelPlayReward` contains the reward name and amount. You can grant the reward to the player here.

What if I want to keep a banner visible throughout the game?

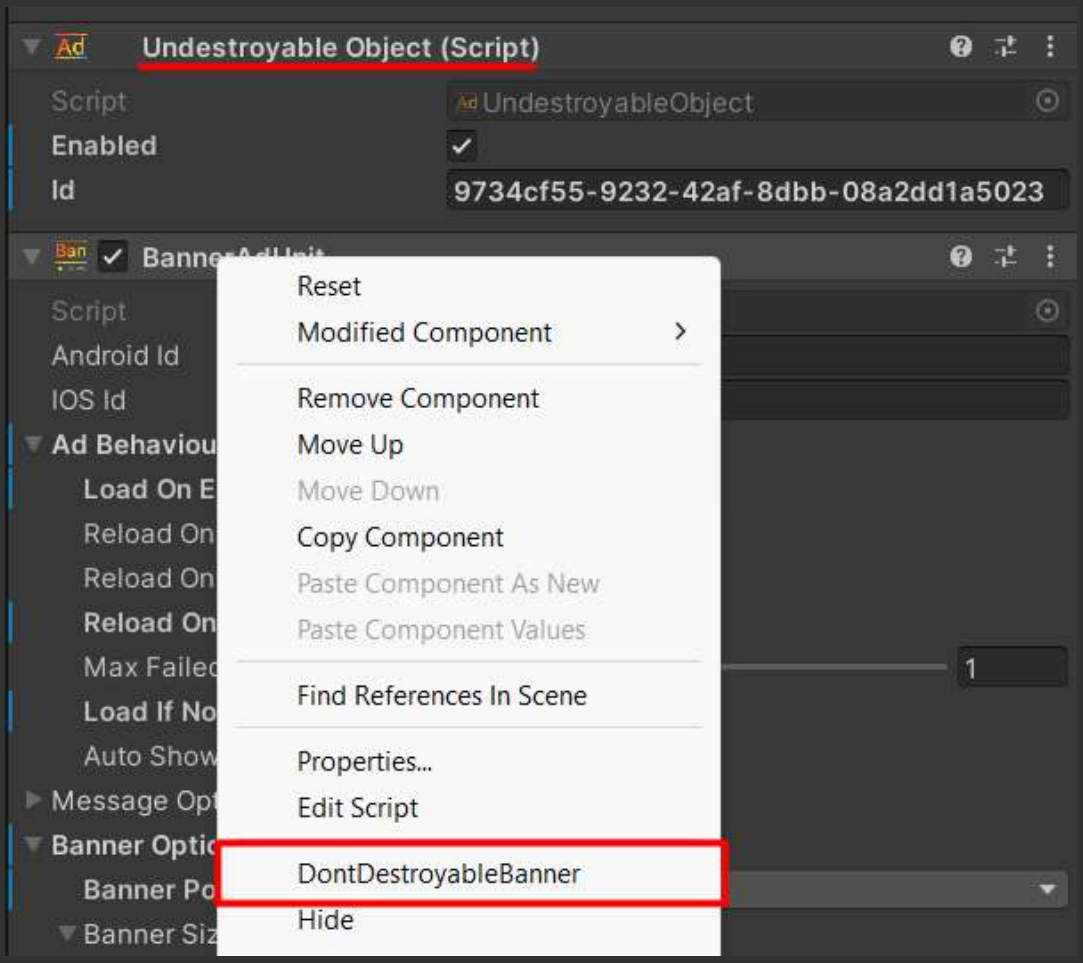
You can. Even if your game switches between multiple scenes, you can keep the banner active without losing control of it.

Load the banner once and keep it for the entire gameplay session, even as scenes change.

However, you may eventually need to display another UI element where the banner appears. Plan when the ad should be visible and when it should be temporarily hidden.

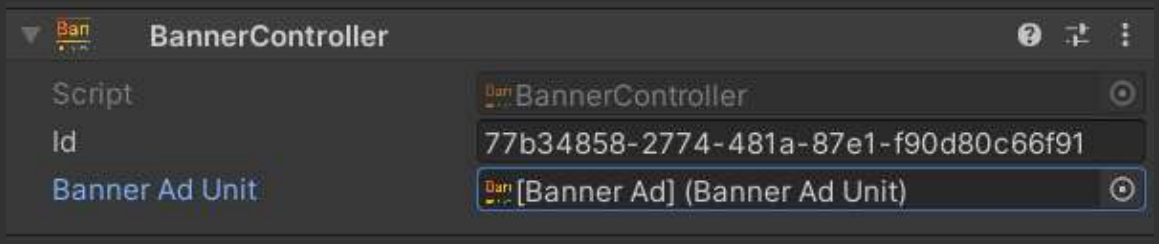
For this, use UndestroyableUtility to control whether the GameObject is destroyed.

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



If you duplicate the banner, clear the identifier field to generate a new ID.

Next, place `BannerController` in the scene where the persistent ad will be initialized.



Remember to assign a reference to the banner!

To control the banner from another scene, create a controller and **copy the ID** from the first scene's controller. **Paste the ID manually** and leave the `BannerAdUnit` field empty.

Keeping the UI Clear of Banner Ads

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

Which helper should I use?

Use [BannerCanvasOffsetHelper] for the built-in Unity UI based on Canvas and RectTransform. Use [`BannerToolkitOffsetHelper`] for UI Toolkit based on UIDocument and VisualElement. Both serve the same purpose, but support different UI systems.

Recommended setup

Attach the helper to the GameObject that owns the root UI for the current screen. If your scene has multiple Canvas or UIDocument objects, assign the reference manually to avoid automatically selecting the first match.

Set Banner Ad Component to the banner component implementing IBannerAd, usually `BannerAdUnit`. If the banner persists across scenes through `BannerController`, also set Banner Ad Controller. This lets the helper access the banner even when the current scene has no `BannerAdUnit`. Assign a controller only after it references the persistent banner: once a controller is assigned, the helper uses it instead of searching the scene.

BannerCanvasOffsetHelper

This version supports Canvas-based UI. Canvas specifies the root Canvas to offset. Explicit Container is optional and accepts a RectTransform within the selected Canvas. If the field is empty or references a RectTransform outside the Canvas hierarchy, the helper creates or reuses a child container named sub-root, moves the current Canvas children into it, and offsets that container.

BannerToolkitOffsetHelper

This version supports UI Toolkit. Ui Document specifies the UIDocument containing the screen. Explicit Container Name is optional and accepts the name of a VisualElement in the visual tree. If the field is empty or the element cannot be found, the helper creates a VisualElement named sub-root, moves the current root elements into it, and offsets that container.

Inspector fields

Margin Top / Bottom / Left / Right add a pixel margin on top of the actual banner size. Use these values to leave a visual gap between the banner and nearby UI elements.

Shift Only Banner Side controls the offset behavior. When enabled, only the side occupied by the banner gets additional inset. When disabled, the entire target container moves away from the banner. Keeping it enabled is generally safer for HUDs and menus.

Runtime behavior

No manual method calls are needed. The helper locates the banner, subscribes to load and hide events, and monitors visibility. It automatically applies the offset when the banner appears and removes it when the banner is hidden or unloaded.

The offset depends on the banner anchor. Top, Bottom, Left, and Right use the corresponding banner dimension plus your margin. Center and Custom add no offset. When multiple helpers target the same container, the built-in side-lock prevents the same side from receiving the offset twice.

In most cases, one helper per root UI is enough. For a banner that persists across scenes, use the helper with `BannerController` so each scene automatically leaves room for the persistent banner.

4. API Reference and Settings

SpaceAdMobile

- `androidAppKey` - app key for Android
- `iOSAppKey` - app key for iOS
- `adInitializationSettings` - Unity LevelPlay requires you to specify the ad formats you plan to display (for example: LevelPlayAdFormat[] legacyAdFormats = new[] { LevelPlayAdFormat.REWARDED };).
- `reinitializationOnErrorInSeconds` - if initialization fails, attempt to initialize the SDK again after the specified number of seconds
- `waitingForInternetConnection` - wait for an internet connection before attempting SDK initialization
- `AgeRestrictedUser` - setting used for COPPA compliance
- `UserConsent` - setting used for consent requirements
- `GoogleFamilySelfCertifiedSDKs` - Google self-certification for family-oriented apps (learn more)
- `UserId` - user identifier used for reporting or server-side identification; if empty, the device identifier is used
- `Ready` - indicates whether LevelPlay initialization has completed
- `LevelPlayConfiguration` - configuration information available after LevelPlay initialization
- `LevelPlayInitError` - details about initialization errors; normally null

*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 failure

(Use this option with caution.)

  - `MaxLoadFailureAttempts` - maximum number of load retry attempts after a failure
  - `LoadOnDemand` - Show() can load the ad for you if needed. Convenient, right?
  - `DisplayImmediatelyOnLoad` - show the ad immediately after it loads; do not combine this with other preloading options
- `messageOptions` - enables callbacks through the I*Message interface methods. Avoid enabling everything; subclassing is 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 revenue data is reported for an ad
- `AdImpressionRecorded` - called when an impression is recorded
- `AdLoaded` - called when an ad loads successfully (including banners)
- `AdLoadFailed` - called when an ad fails to load
- `AdClicked` - called when the user clicks an ad
- `AdShown` - called when an ad appears on screen
- `AdClosed` - called when an ad closes
- `AdContentFailed` - called when ad content cannot be displayed
- `AdReward` - called when the user receives a reward (rewarded ad formats). The best part!

`AdEventsDispatcher`

- `ValidateType`
  - `Full`: automatically enables and disables MessageOptions for each callback type listed above.
  - `Enable`: works like Full, but never disables MessageOptions.
  - `None`: disables validation.
- You can add any missing events to this class. Currently, only the most commonly used callbacks are included: FailedEvent, LoadEvent, ClickEvent, ShownEvent, and CloseEvent.

Fields and properties may vary between Space Ad Mobile versions.

LevelPlay Validation and Banner Control

`SpaceAdMobile.ValidateIntegration()`: Call ValidateIntegration() on the SpaceAdMobile component during development to check the SDK and adapters. MISSING may be expected for an unused network; investigate unexpected NOT VERIFIED results. Remove the validation call before release.

`BannerAdUnit.PauseAutoRefresh() / ResumeAutoRefresh()`: Pause automatic refresh when the banner is not needed, such as while a modal dialog is open. Call ResumeAutoRefresh() when it can resume. These methods control refreshing but do not hide the banner.

Useful tips:

For first-launch setup, you can add an age-selection screen like the one in the demo. Use it to configure the following fields and save the settings for your app.

https://docs.unity.com/en-us/grow/levelplay/sdk/unity/regulation-advanced-settings

https://docs.unity.com/en-us/grow/levelplay/sdk/unity/advanced-settings/



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.