Quick start

Integration

10.07.2024

Yandex

Quick start. Integration. Version 2.0

Document build date: 10.07.2024

This volume is a part of Yandex technical documentation.

Copyright Disclaimer

Contact information

Yandex LLC

https://www.yandex.com

Тел.: +7 495 739 7000

Email: pr@yandex-team.ru

16 L'va Tolstogo St., Moscow, Russia 119021

Contents

Enabling the Mobile Ads Unity plugin

Warning:

This is an archived version of the documentation. Actual documentation for all platforms can be found here.

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

Note:

1. To load ads of any type, you need iOS 12.0 or later.
2. To run the SDK, you need the Target API Level 31 or higher.

Integrating the plugin

Note: `yandex-ads-unity-plugin` runs only in Android and iOS environments. You can't use it in the Unity editor.

Lite version

1. Download the directory `yandex-ads-unity-plugin` and add the package `yandex-mobileads-lite-2.9.0.unitypackage`.

   How to add a package

   Select a plugin (**Assets** → **Import Package** → **Custom Package**), then click **Import**.
2. Add the Google resolver: download the directory `unity-jar-resolver` and add the package `external-dependency-manager-latest.unitypackage`.

   How to add a package

   Select a plugin (**Assets** → **Import Package** → **Custom Package**), then click **Import**.
3. Use the Google resolver to install dependencies: enable **auto-resolve** or select **Assets** → **External Dependency Manager** → **Android Resolver** → **Resolve** in the menu.
4. To test the Mobile Ads Unity plugin, use a sample script from the **samples** directory in the `yandex-ads-unity-plugin` repository. Copy the script to the project directory and add it as a **Component** to the main camera.

Downgrading the Target API Level

To downgrade the Target API Level to 30, add the explicit downgrade to `mainTemplate.gradle` and `launcherTemplate.gradle` (if you use launcherTemplate in the project):

```
configurations.all {
    resolutionStrategy {
        force 'androidx.core:core:1.6.0'
        force 'androidx.core:core-ktx:1.6.0'
    }
}
```

However, we recommend that you upgrade to the Target API Level 31, because Google has restrictions on releasing updates for applications running an outdated version of the Target API Level. Learn more in the article.

Ad formats

Banner ads

Warning:

This is an archived version of the documentation. Actual documentation for all platforms can be found here.

A *banner* is a configurable ad that covers part of the screen and reacts to clicks.

Adding Banner to the project

To display a banner in your app, create a Banner object in the script (in C#) that is attached to the GameObject.

```
...
using YandexMobileAds;
using YandexMobileAds.Base;
...

public class YandexMobileAdsBannerDemoScript : MonoBehaviour
{
    private Banner banner;
    ...
    private void RequestBanner()
    {
        string adUnitId = "YOUR_adUnitId";

        banner = new Banner(adUnitId, AdSize.BANNER_320x50, AdPosition.BottomCenter);
    }
    ...
}
```

The Banner constructor contains the following parameters:

- AdUnitId — A unique identifier that is issued in the Partner interface and looks like this: R-M-XXXXXX-Y.
- AdSize — The size of the banner you want to display.
- AdPosition — The position on the screen.

Loading ads

After creating and configuring the object of the Banner class, load ads. To load an ad, use the LoadAd method, which takes the AdRequest object as an argument.

```
banner.LoadAd(request);
```

About loading ads

Use the AdRequest object to transmit the code received in the Adfox interface (for more information, see Help for Adfox):

```
...
// Code from the Adfox interface for working with direct campaigns.
private Dictionary<string, string> CreateAdfoxParameters()
{
    Dictionary<string, string> parameters = new Dictionary<string, string>()
    {
        {"adf_ownerid", "<example>"},
        {"adf_p1", "<example>"},
        {"adf_p2", "<example>"},
        {"adf_pt", "<example>"},
        ...
    };

    return parameters;
}

...

private void RequestBanner()
{
    ...
    AdRequest request = new AdRequest.Builder()
        .WithParameters(CreateAdfoxParameters())
        .Build();
    banner.LoadAd(request);
    ...
}
```

Banner ad events

To track events that occur in banner ads, register a delegate for the appropriate EventHandler, as shown below:

```
...
private void RequestBanner()
{
    ...
        banner.OnAdLoaded += HandleAdLoaded;
        banner.OnAdFailedToLoad += HandleAdFailedToLoad;
        banner.OnReturnedToApplication += HandleReturnedToApplication;
        banner.OnLeftApplication += HandleLeftApplication;
        banner.OnAdClicked += HandleAdClicked;
```

```
        banner.OnImpression += HandleImpression;
    ...
}

public void HandleAdLoaded(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleAdLoaded event received");
    banner.Show();
}

public void HandleAdFailedToLoad(object sender, AdFailureEventArgs args)
{
    MonoBehaviour.print("HandleAdFailedToLoad event received with message: " + args.Message);
}

public void HandleLeftApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleLeftApplication event received");
}

public void HandleReturnedToApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleReturnedToApplication event received");
}

public void HandleAdLeftApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleAdLeftApplication event received");
}

public void HandleAdClicked(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleAdClicked event received");
}

public void HandleImpression(object sender, ImpressionData impressionData)
{
    var data = impressionData == null ? "null" : impressionData.rawData;
    MonoBehaviour.print("HandleImpression event received with data: " + data);
}
```

Clearing ads

When an ad object is no longer needed, you can delete it. To do this, call the `Destroy` method:

```
banner.Destroy();
```

Interstitial ads

Warning:

This is an archived version of the documentation. Actual documentation for all platforms can be found here.

An *interstitial ad* is a configurable ad that covers the entire screen and responds to clicks.

Adding `Interstitial` to the project

To enable advertising, create an `Interstitial` object in the script (in C#) that is attached to the `GameObject`.

```
...
using YandexMobileAds;
using YandexMobileAds.Base;
...

public class YandexMobileAdsInterstitialDemoScript : MonoBehaviour
{
    private Interstitial interstitial;
    ...
    private void RequestInterstitial()
    {
        string adUnitId = "YOUR_adUnitId";

        interstitial = new Interstitial(adUnitId);
    }
    ...
}
```

The `Interstitial` constructor contains the `adUnitId` parameter, which is a unique identifier that is assigned in the Partner interface and looks like this: R-M-XXXXXX-Y.

Loading ads

After creating and configuring the object of the `Interstitial` class, load ads. To load an ad, use the `LoadAd` method, which takes the `AdRequest` object as an argument.

```
interstitial.LoadAd(request);
```

About loading ads

Use the `AdRequest` object to transmit the code received in the Adfox interface (for more information, see Help for Adfox):

```
...
// Code from the Adfox interface for working with direct campaigns.
private Dictionary<string, string> CreateAdfoxParameters()
{
    Dictionary<string, string> parameters = new Dictionary<string, string>()
    {
        {"adf_ownerid", "<example>"},
        {"adf_p1", "<example>"},
        {"adf_p2", "<example>"},
        {"adf_pt", "<example>"},
        ...
    };

    return parameters;
}

...

private void RequestInterstitial()
{
    ...
    AdRequest request = new AdRequest.Builder()
        .WithParameters(CreateAdfoxParameters())
        .Build();
    interstitial.LoadAd(request);
    ...
}
```

Displaying ads

After the ad has loaded, you can display it:

```
...
private void ShowInterstitial()
{
    if (this.interstitial.IsLoaded())
    {
        interstitial.Show();
    }
    else
    {
        Debug.Log("Interstitial is not ready yet");
    }
}
...
```

Interstitial ad events

To track events that occur in interstitial ads, register a delegate for the appropriate `EventHandler`, as shown below:

```
...
private void RequestInterstitial()
{
    ...
    interstitial.OnInterstitialLoaded += HandleInterstitialLoaded;
    interstitial.OnInterstitialFailedToLoad += HandleInterstitialFailedToLoad;
    interstitial.OnReturnedToApplication += HandleReturnedToApplication;
    interstitial.OnLeftApplication += HandleLeftApplication;
    interstitial.OnAdClicked += HandleAdClicked;
    interstitial.OnInterstitialShown += HandleInterstitialShown;
    interstitial.OnInterstitialDismissed += HandleInterstitialDismissed;
    interstitial.OnImpression += HandleImpression;
    interstitial.OnInterstitialFailedToShow += HandleInterstitialFailedToShow;
    ...
}

public void HandleInterstitialLoaded(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleInterstitialLoaded event received");
}
```

```
public void HandleInterstitialFailedToLoad(object sender, AdFailureEventArgs args)
{
    MonoBehaviour.print(
        "HandleInterstitialFailedToLoad event received with message: " + args.Message);
}

public void HandleReturnedToApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleReturnedToApplication event received");
}

public void HandleLeftApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleLeftApplication event received");
}

public void HandleAdClicked(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleAdClicked event received");
}

public void HandleInterstitialShown(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleInterstitialShown event received");
}

public void HandleInterstitialDismissed(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleInterstitialDismissed event received");
}

public void HandleImpression(object sender, ImpressionData impressionData)
{
    var data = impressionData == null ? "null" : impressionData.rawData;
    MonoBehaviour.print("HandleImpression event received with data: " + data);
}

public void HandleInterstitialFailedToShow(object sender, AdFailureEventArgs args)
{
    MonoBehaviour.print(
        "HandleInterstitialFailedToShow event received with message: " + args.Message);
}
```

Clearing ads

When an ad object is no longer needed, you can delete it. To do this, call the `Destroy` method:

```
interstitial.Destroy();
```

Rewarded ads

Warning:

This is an archived version of the documentation. Actual documentation for all platforms can be found here.

A *rewarded ad* is a configurable full-screen ad. The user gets a reward for viewing the ad.

Adding a `rewarded ad` to the project

To enable advertising, create a `RewardedAd` object in the script (in C#) that is attached to the `GameObject`.

```
...
using YandexMobileAds;
using YandexMobileAds.Base;
...

public class YandexMobileAdsRewardedAdDemoScript : MonoBehaviour
{
    private RewardedAd rewardedAd;
    ...
    private void RequestRewardedAd()
    {
        string adUnitId = "YOUR_adUnitId";

        rewardedAd = new RewardedAd(adUnitId);
    }
    ...
}
```

The `RewardedAd` constructor contains the `adUnitId` parameter, a unique identifier that is assigned in the Partner interface and looks like this: R-M-XXXXXX-Y.

Loading ads

After creating and configuring the object of the `RewardedAd` class, you need to load ads. To load an ad, use the `LoadAd` method, which takes the `AdRequest` object as an argument.

```
rewardedAd.LoadAd(request);
```

About loading ads

Use the `AdRequest` object to transmit the code received in the Adfox interface (for more information, see Help for Adfox):

```
...
// Code from the Adfox interface for working with direct campaigns.
private Dictionary<string, string> CreateAdfoxParameters()
{
    Dictionary<string, string> parameters = new Dictionary<string, string>()
    {
        {"adf_ownerid", "<example>"},
        {"adf_p1", "<example>"},
        {"adf_p2", "<example>"},
        {"adf_pt", "<example>"},
        ...
    };

    return parameters;
}

...

private void RequestRewardedAd()
{
    ...
    AdRequest request = new AdRequest.Builder()
        .WithParameters(CreateAdfoxParameters())
        .Build();
    rewardedAd.LoadAd(request);
    ...
}
```

Displaying ads

After the ad has loaded, you can display it:

```
...
private void ShowRewardedAd()
{
    if (this.rewardedAd.IsLoaded())
    {
        rewardedAd.Show();
    }
    else
    {
        Debug.Log("Rewarded Ad is not ready yet");
    }
}
...
```

Rewarded ad events

To track events that occur in a rewarded ad, register a delegate for the appropriate `EventHandler`, as shown below:

```
...
private void RequestRewardedAd()
{
    ...
    rewardedAd.OnRewardedAdLoaded += HandleRewardedAdLoaded;
    rewardedAd.OnRewardedAdFailedToLoad += HandleRewardedAdFailedToLoad;
    rewardedAd.OnReturnedToApplication += HandleReturnedToApplication;
    rewardedAd.OnLeftApplication += HandleLeftApplication;
    rewardedAd.OnAdClicked += HandleAdClicked;
    rewardedAd.OnRewardedAdShown += HandleRewardedAdShown;
    rewardedAd.OnRewardedAdDismissed += HandleRewardedAdDismissed;
    rewardedAd.OnImpression += HandleImpression;
    rewardedAd.OnRewarded += HandleRewarded;
    rewardedAd.OnRewardedAdFailedToShow += HandleRewardedAdFailedToShow;
    ...
}
```

```
public void HandleRewardedAdLoaded(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleRewardedAdLoaded event received");
}

public void HandleRewardedAdFailedToLoad(object sender, AdFailureEventArgs args)
{
    MonoBehaviour.print(
        "HandleRewardedAdFailedToLoad event received with message: " + args.Message);
}

public void HandleReturnedToApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleReturnedToApplication event received");
}

public void HandleLeftApplication(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleLeftApplication event received");
}

public void HandleAdClicked(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleAdClicked event received");
}

public void HandleRewardedAdShown(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleRewardedAdShown event received");
}

public void HandleRewardedAdDismissed(object sender, EventArgs args)
{
    MonoBehaviour.print("HandleRewardedAdDismissed event received");
}

public void HandleImpression(object sender, ImpressionData impressionData)
{
    var data = impressionData == null ? "null" : impressionData.rawData;
    MonoBehaviour.print("HandleImpression event received with data: " + data);
}

public void HandleRewarded(object sender, Reward args)
{
    MonoBehaviour.print("HandleRewarded event received: amout = " + args.amount + ", type = " + args.type);
}

public void HandleRewardedAdFailedToShow(object sender, AdFailureEventArgs args)
{
    MonoBehaviour.print(
        "HandleRewardedAdFailedToShow event received with message: " + args.Message);
}
```

Clearing ads

When an ad object is no longer needed, you can delete it. To do this, call the `Destroy` method:

```
rewardedAd.Destroy();
```