Integration

Integration

10.07.2024

Yandex

Integration. Version 1.0

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

Adding 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This is an archived version of the documentation.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 gaming platform that includes support for the Yandex Mobile Ads SDK.

Note:

1. To run the SDK, you need the Target API Level version 31 or higher.
2. To upload ads of any type, you need iOS 12.0 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`. You will be prompted to install the Google resolver with the package. If you already added the Google resolver to your project, clear the checkbox.

   How to add the package

   Select your plugin (**Assets** → **Import Package** → **Custom Package**), then click **Import**.
2. Use the Google resolver to install dependencies: enable **auto-resolve** or select **Assets** → **External Dependency Manager** → **Android Resolver** → **Resolve** in the menu.
3. 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 version 30, add the explicit downgrade to `mainTemplate.gradle` and `launcherTemplate.gradle` (if you use launcherTemplate in the project):

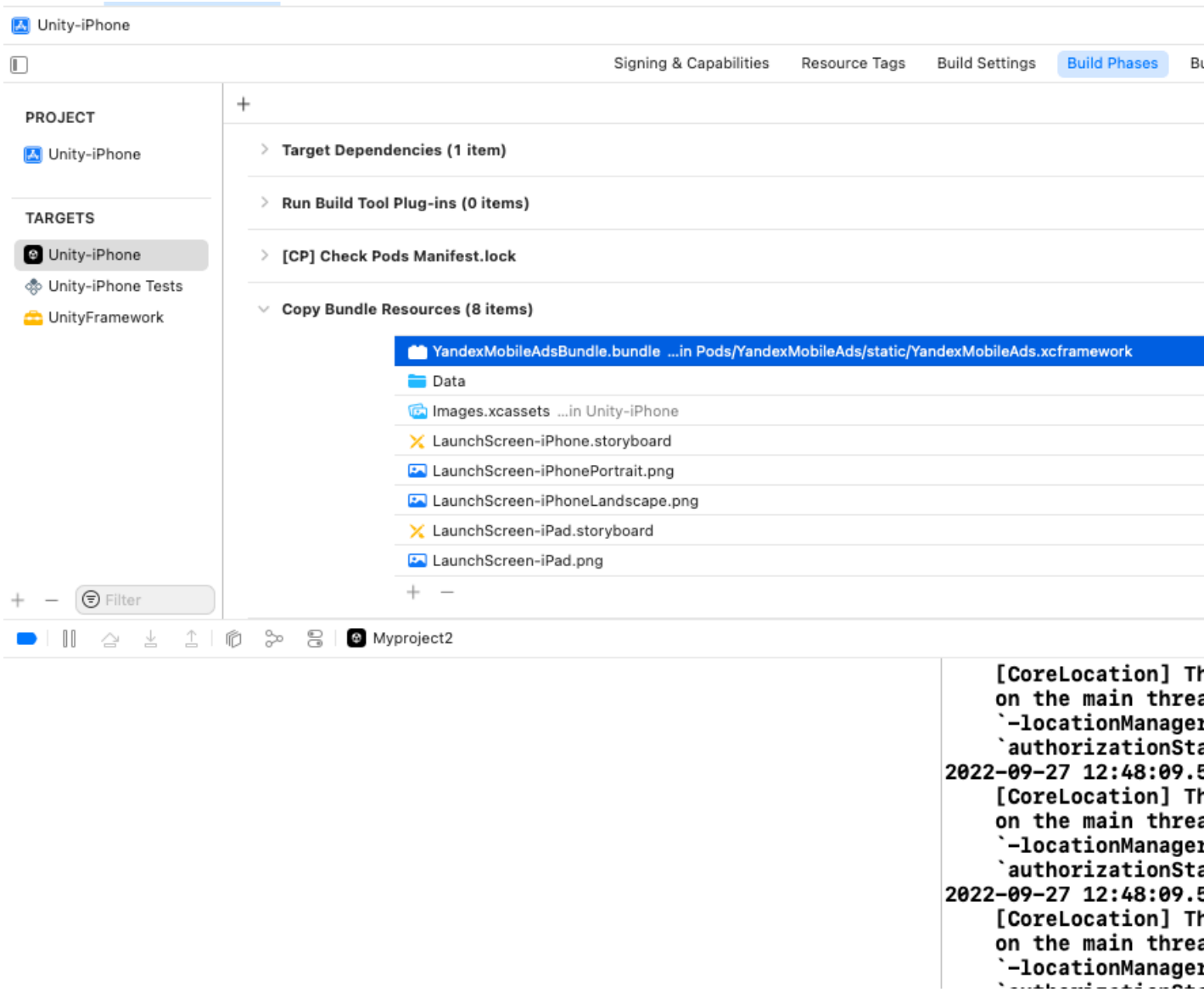
```
configurations.all {
    resolutionStrategy {
        force 'androidx.core:core:1.6.0'
        force 'androidx.core:core-ktx:1.6.0'
    }
}
```

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you upgrade to the Target API Level 31, because Google has restrictions on releasing updates for applications running outdated versions of the Target API Level.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

Error description

Interstitial Unity Ads aren't displayed, error “Incorrect fullscreen view”

The error “Incorrect fullscreen view” may arise when launching interstitial ads on iOS. If this issue arises, make sure that you added the value **YandexMobileAdsBundle.bundle** under **Copy Bundle Resources** in the **Build Phases** settings. If the value is missing, add it.



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 responds 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 = "demo-banner-yandex";

        // Set flexible banner maximum width and height
        AdSize bannerMaxSize = AdSize.FlexibleSize(GetScreenWidthDp(), 100);
        // Or set sticky banner maximum width
        //AdSize bannerMaxSize = AdSize.StickySize(GetScreenWidthDp());

        banner = new Banner(adUnitId, bannerMaxSize, AdPosition.BottomCenter);
    }

    // Example how to get screen width for request
    private int GetScreenWidthDp()
    {
        int screenWidth = (int)Screen.safeArea.width;
        return ScreenUtils.ConvertPixelsToDp(screenWidth);
    }
    ...
}
```

The `Banner` constructor contains the following parameters:

- `AdUnitId`: A unique identifier that is issued in the Partner interface and takes the format R-M-XXXXXX-Y.
- `AdSize`: The size of the banner you want to display.
- `AdPosition`: The banner's position on the screen.

The following banner sizes are supported:

- FlexibleSize(int width, int height): An adaptive banner with a given maximum width and height.
- StickySize(int width): An adaptive banner with a given maximum width.
- Fixed sizes. An outdated version of the requested size, we recommend Flexible or Sticky sizes.

Size	Device	AdSize constant
320 x 50	Phones and tablets	`BANNER_320x50`
320 x 100	Phones and tablets	`BANNER_320x100`
300 x 250	Phones and tablets	`BANNER_300x250`
300 x 300	Phones and tablets	`BANNER_300x300`
240 x 400	Phones and tablets	`BANNER_240x400`
400 x 240	Phones and tablets	`BANNER_400x240`
728 x 90	Tablets	`BANNER_728x90`

Loading ads

Once you've created and set up the `Banner` class object, you need to load your ad. To load an ad, use the `LoadAd` method, accepting the `AdRequest` object as a parameter.

```
...
private void RequestBanner()
{
    ...
    AdRequest request = new AdRequest.Builder().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();
```

Adaptive banners

Warning:

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

Adaptive banners are banners that fit seamlessly into user-defined unit sizes. Depending on how an adaptive banner is integrated, the optimal height is determined for the given width or the specified size of ad placement is used.

Note:

You can read about creating an ad unit for an adaptive banner in the Yandex Advertising Network Help.

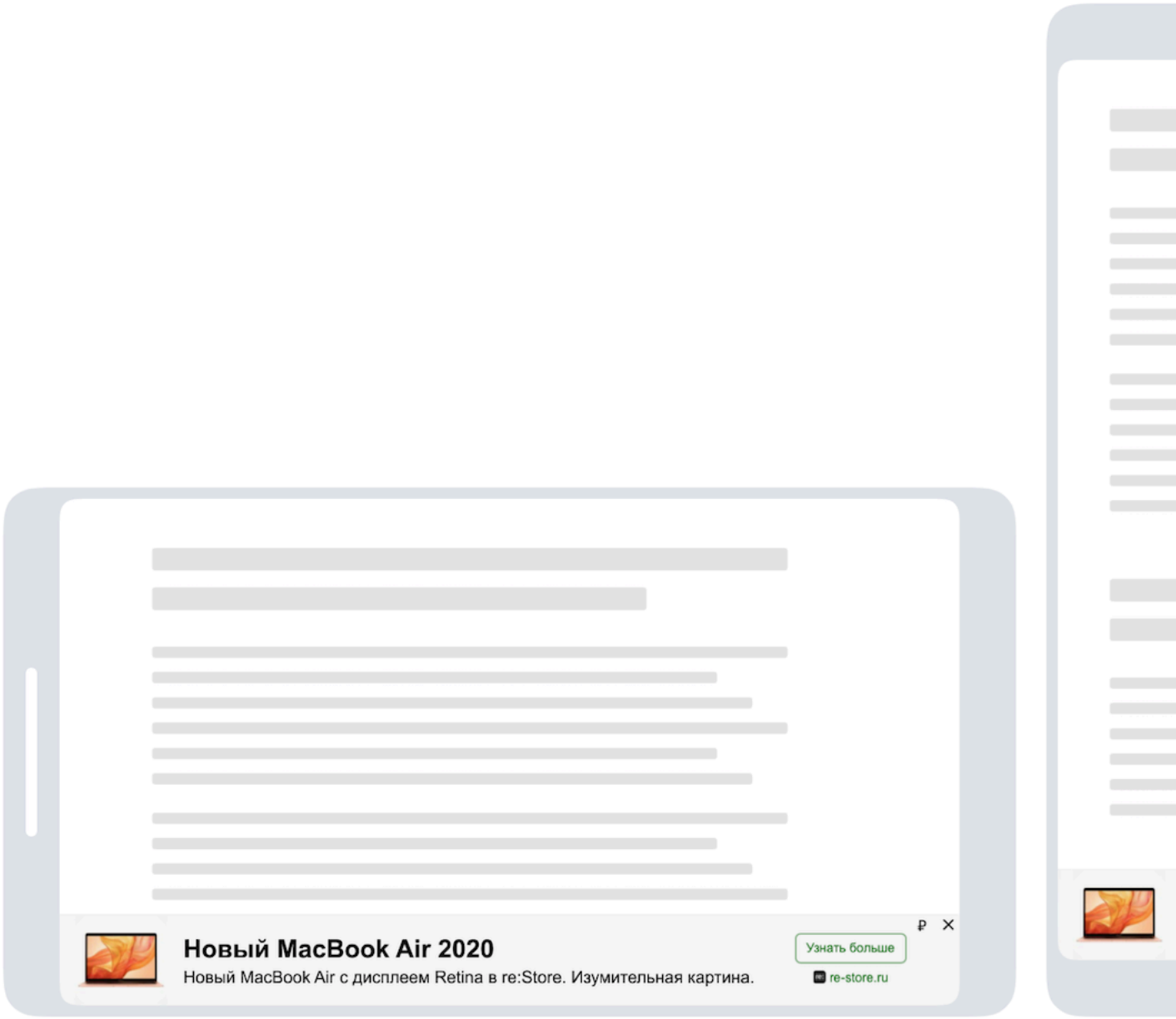
Types of adaptive banners

Banner with a set width

Features:

1. An alternative to 320x50 banners (when determining the banner height, the 320x50 aspect ratio is maintained).
2. The banner's position is fixed at the top or bottom of the screen (configured in the app).
3. The given banner width is used instead of the device screen width. This lets you take into account the display's features.
4. The width of an adaptive banner is set using the AdSize.StickySize(int width) method.

Examples of displaying adaptive banners:

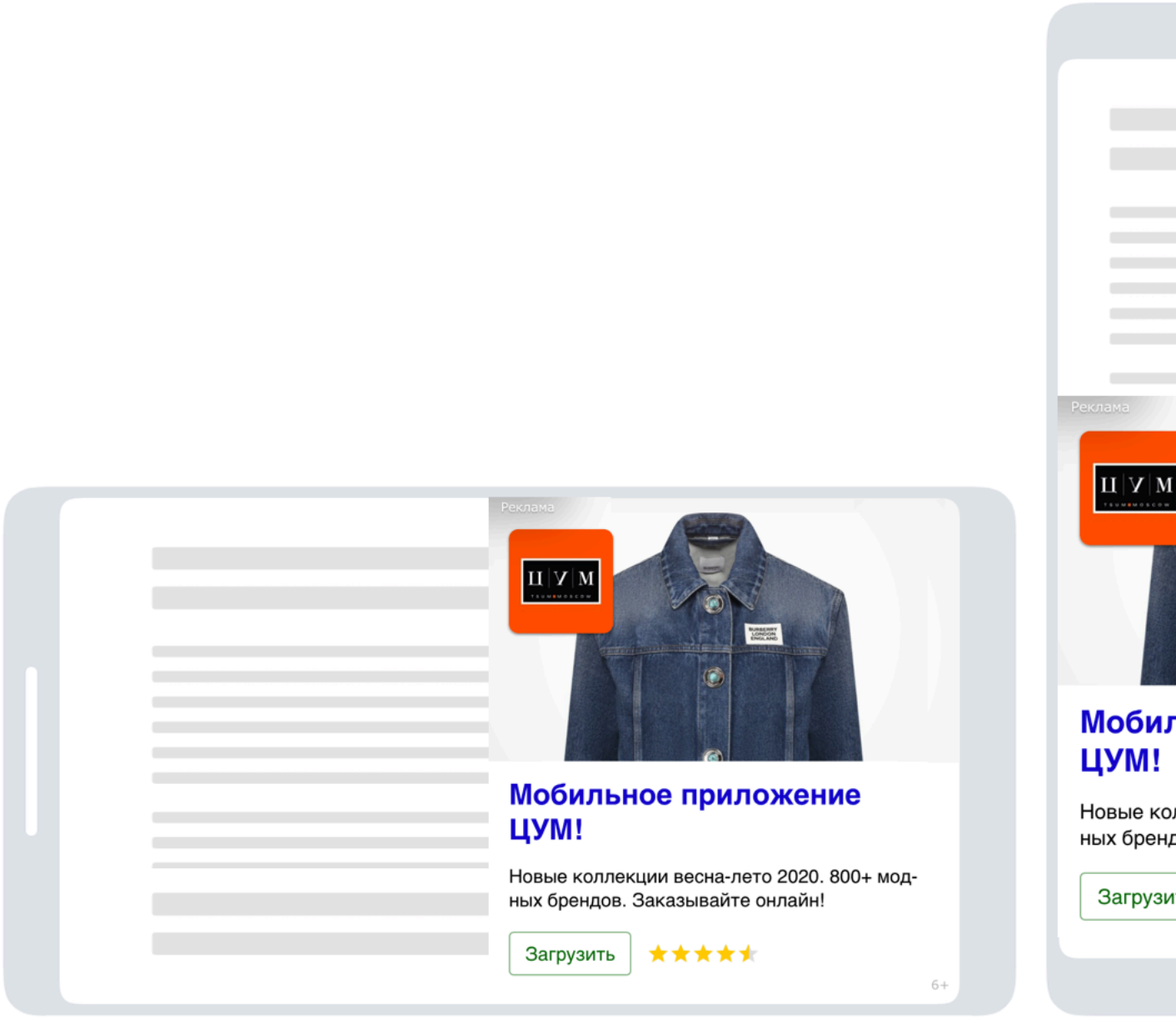


Banner with a set width and height

Features:

1. An adaptive banner fills up the entire unit using the given width and height.
2. The width and height of an adaptive banner is set using the AdSize.FlexibleSize(int width, int height) method.

Examples of displaying adaptive banners:



Adding Banner to the project

1. 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 = "demo-banner-yandex";
        banner = ...
    }
    ...
}
```

2. Specify the banner size and initialize the banner in the app. Use the ScreenUtils.ConvertPixelsToDp method to convert the width in pixels to density-independent pixels.

   Banner with a set width

   ```
   AdSize adSize = AdSize.StickySize(width);
   banner = new Banner(adUnitId, adSize, AdPosition.BottomCenter);
   ```

   Banner with a set width and height

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

Loading ads

Once you've created and set up the `Banner` class object, you need to load your ad. To load an ad, use the `LoadAd` method, accepting the `AdRequest` object as a parameter.

```
...
private void RequestBanner()
{
    ...
    AdRequest request = new AdRequest.Builder().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();
```

Example of working with adaptive banners

The following code demonstrates creating, configuring, and loading a `Banner` object:

```
...

public class YandexMobileAdsBannerDemoScript : MonoBehaviour
{
    private Banner banner;
    ...
    private void RequestBanner()
    {
        string adUnitId = "demo-banner-yandex";
        int screenWidth = (int)Screen.safeArea.width;
        int width = ScreenUtils.ConvertPixelsToDp(screenWidth);
        AdSize adSize = AdSize.StickySize(width);
        banner = new Banner(adUnitId, adSize, AdPosition.BottomCenter);
        AdRequest request = new AdRequest.Builder().Build();
        banner.LoadAd(request);
    }
    ...
}
```

Related information

Ad example

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 = "demo-interstitial-yandex";

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

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

Loading ads

Once you have created and configured the `Interstitial` class object, load your ads. To load an ad, use the `LoadAd` method, accepting the `AdRequest` object as a parameter.

```
...
private void RequestInterstitial()
{
    ...
    AdRequest request = new AdRequest.Builder().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 an ad, 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 = "demo-rewarded-yandex";

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

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

Loading ads

Once you have created and configured the `RewardedAd` class object, load your ads. To load an ad, use the `LoadAd` method, accepting the `AdRequest` object as a parameter.

```
...
private void RequestRewardedAd()
{
    ...
    AdRequest request = new AdRequest.Builder().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();
```

GDPR

Warning:

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

General information

In the spring of 2018, the General Data Protection Regulation (GDPR) came into effect. GDPR regulates how information about citizens of the European Economic Area and Switzerland can be collected and processed. This regulation is designed to protect the privacy of confidential data and to ensure the transparency of all processes related to the collection, storage, and processing of information on the internet.

The GDPR has an extraterritorial scope that applies to all companies that process the personal data of citizens of the European Economic Area and Switzerland, regardless of where the company is located.

Starting from version 0.3.0, Yandex Mobile Ads Unity allows you to restrict the collection of data for users located in the European Economic Area and Switzerland who do not consent to data collection.

Quick guide

The user's consent for processing personal data must be sent to the SDK each time the application is launched.

1. Follow the instructions for connecting Yandex Mobile Ads Unity.
2. Show a window where the user can accept the user agreement for personal data processing.
3. Use the setUserConsent method to pass the received value to the Yandex Mobile Ads SDK. Data of users located in the GDPR region will be processed only if the user consents to data processing.

```
MobileAds.SetUserConsent(true);
```