Yandex Mobile Ads

Integration

10.07.2024

Yandex Mobile Ads. Integration. Version 2.0

Document build date: 10.07.2024

This volume is a part of Yandex technical documentation.

© 2008—2024 Yandex LLC. All rights reserved.

Copyright Disclaimer

Yandex (and its applicable licensor) has exclusive rights for all results of intellectual activity and equated to them means of individualization, used for development, support, and usage of the service Yandex Mobile Ads. It may include, but not limited to, computer programs (software), databases, images, texts, other works and inventions, utility models, trademarks, service marks, and commercial denominations. The copyright is protected under provision of Part 4 of the Russian Civil Code and international laws.

You may use Yandex Mobile Ads or its components only within credentials granted by the Terms of Use of Yandex Mobile Ads or within an appropriate Agreement.

Any infringements of exclusive rights of the copyright owner are punishable under civil, administrative or criminal Russian laws.

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

Integrating the Mobile Ads SDK

Warning:

This is an archived version of the documentation.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

Note:

1. To load ads of any type, Android 4.1 or later is required.
2. Video ads are only selected for devices with Android 5.0 or later.

The Yandex Mobile Ads library is provided in AAR format. To enable the Mobile Ads SDK:

1. Add the Yandex Mobile Ads dependency to the `build.gradle` file of your app's module:

```
dependencies {
    ...
    implementation 'com.yandex.android:mobileads:5.10.0'
}
```

2. Update the kotlin-gradle-plugin dependency in your project's root `build.gradle` file:

```
dependencies {
    ...
    classpath("org.jetbrains.kotlin:kotlin-gradle-plugin:1.7.10")
}
```

3. Add Java 8 support to the `build.gradle` file in your app's module:

```
android {

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}
```

4. Set up permission to use the ad ID.

   The ad ID is a unique identifier provided by Google Play services for displaying ads to users who opt in to personalized ads. Users can opt out of ad personalization or reset their ID in the settings. In this case, advertising networks won't be able to use the ID to select relevant ads for the user.

   Mobile Ads SDK version 4.5.0 and later

   A new permission has been made available in the Yandex Mobile Ads SDK version 4.5.0 and higher: `com.google.android.gms.permission.AD_ID`. It's written in the library's `AndroidManifest.xml` file. Because of this, you don't have to specify it in the application's main manifest. The permission allows you to use an ad ID to select relevant ads from advertising networks.

   You can delete the permission if necessary. For example, if a policy does not allow the use of an ID for ad selection, such as the Families Policy.

   To prevent the permission from being added to the application's main manifest, add the following line to `AndroidManifest.xml`:

```
<uses-permission android:name="com.google.android.gms.permission.AD_ID" tools:node="remove"/>
```

   Mobile Ads SDK version below 4.5.0

   If your app uses a version of the Yandex Mobile Ads SDK below 4.5.0 and there are no restrictions on the use of an ad ID (for example, Families Policy), add the permission to the application's main manifest `AndroidManifest.xml`:

```
<uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
```

   Lack of this permission and access to the ID may reduce the relevance of ads and, as a result, your income.

Initializing the Mobile Ads SDK

Before loading ads, initialize the library using the initialize() method. Initialization makes ads load faster.

Note:

It's needed each time the app starts. That's why we recommend that you add the initialization code to the `onCreate` method of the `Application` class.

Initialization example:

```
public class YandexApplication extends Application {

    private static final String YANDEX_MOBILE_ADS_TAG = "YandexMobileAds";

    @Override
    public void onCreate() {
        super.onCreate();

        MobileAds.initialize(this, new InitializationListener() {
            @Override
            public void onInitializationCompleted() {
                Log.d(YANDEX_MOBILE_ADS_TAG, "SDK initialized");
            }
        });
    }
}
```

See the SDK usage examples.

Ad formats

Enabling 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.

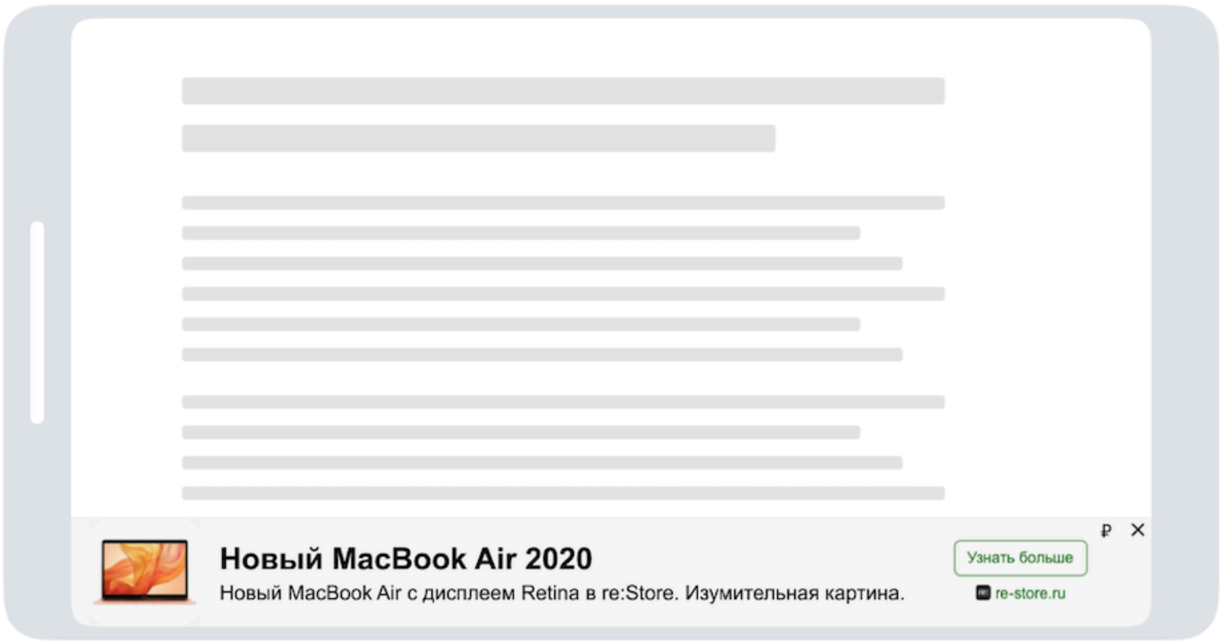
Banner types

Sticky banner

Features:

1. The specified banner width is used. The height is selected automatically.
2. The width of banners is set using the stickySize method.
3. The banner height shouldn't exceed 15% of the device height and should be at least 50 dp.

Examples of displaying banners:

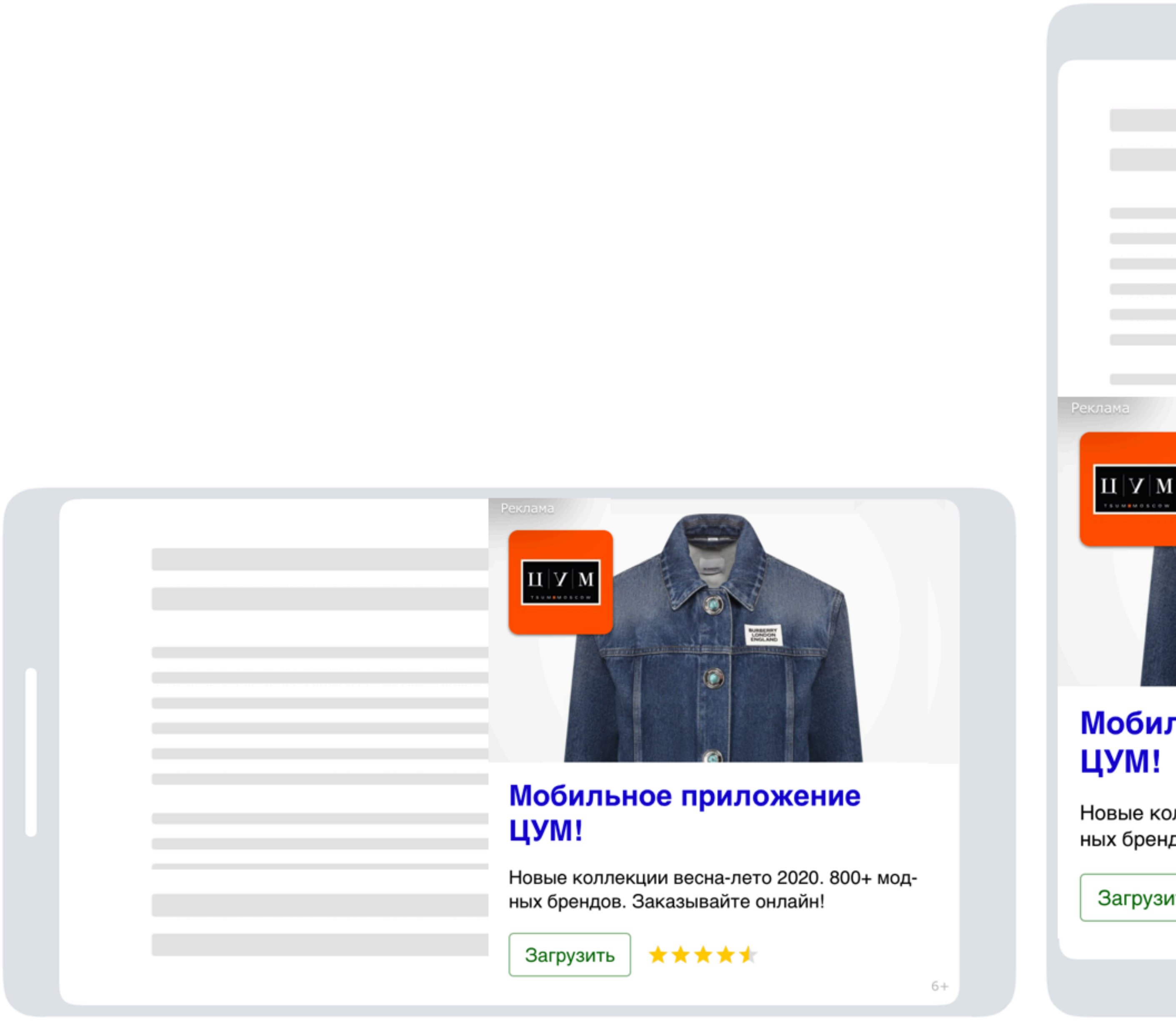

Новый MacBook Air 2020
Новый MacBook Air с дисплеем Retina в re:Store. Изумительная картина.
Узнать больше
re-store.ru

Flex banner

Features:

1. A banner fills up the entire unit using the set maximum sizes.
2. The width and height of a banner is set using the flexibleSize(int width, int height) method.

Examples of displaying banners:



Новые коллекции весна-лето 2020. 800+ модных брендов. Заказывайте онлайн!

Enabling a banner

Creating BannerAdview

1. Add an object of the `BannerAdView` class to the project using an XML file or programmatically.

```
// Creating an mBannerAdView instance using an XML file.
        mBannerAdView = (BannerAdView) findViewById(R.id.banner_view);

// Creating an mBannerAdView instance programmatically.
        mBannerAdView = new BannerAdView(this);
```

2. Set the `AdUnitId` using the setAdUnitId method.

```
mBannerAdView.setAdUnitId(<AdUnitId>)
```

`AdUnitId` is a unique identifier in R-M-XXXXXX-Y format, which is assigned in the Partner Interface.

3. Set the banner size using the setAdSize method.

   Sticky banner

   To set the width of a banner, call the stickySize(int width) method, where width is the maximum banner width.

   ```
   mBannerAdView.setAdSize(AdSize.stickySize(width));
   ```

   Flex banner

   To set the width and height of a banner, call the flexibleSize(int width, int height) method.

   ```
   mBannerAdView.setAdSize(AdSize.flexibleSize(width, height));
   ```

   Restriction: Banner size requirements when displaying video ads

   Minimum size of a banner that supports video playback is 300x160 or 160x300 dp (density-independent pixels).
4. After creating and configuring an instance of the `BannerAdView` class, you can set an AdEventListener on the ad object for tracking events (opening or closing the ad, exiting the app, and loading the ad successfully or unsuccessfully).

Loading ads

Note:

Any call of the Mobile Ads SDK should be made from the main thread.

After creating and configuring the `BannerAdView` class object, load the ads. To load an ad, use the loadAd method, which takes the AdRequest object as a parameter (or Builder, which optionally accepts ad targeting data).

Example of working with banner ads

The following code demonstrates creating and configuring the `AdView` object, registering a listener, and loading a banner:

```
...
<LinearLayout>
    ...
    <com.yandex.mobile.ads.banner.BannerAdView
        android:id="@+id/banner_ad_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

```
...
private BannerAdView mBannerAdView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // Creating an mAdView instance.
        mBannerAdView = (BannerAdView) findViewById(R.id.banner_ad_view);
        mBannerAdView.setAdUnitId(AdUnitId);
        mBannerAdView.setAdSize(AdSize.stickySize(maxBannerWidthDp));

        // Creating an ad targeting object.
        final AdRequest adRequest = new AdRequest.Builder().build();

        // Registering a listener for tracking events in the banner ad.
        mBannerAdView.setAdEventListener(new BannerAdEventListener() {
            @Override
            public void onAdLoaded() {
                ...
            }

            @Override
            public void onAdFailedToLoad(AdRequestError adRequestError) {
                ...
            }

            @Override
            public void onLeftApplication() {
                ...
            }

            @Override
            public void onReturnedToApplication() {
                ...
```

```
            }
        });

        // Loading ads.
        mBannerAdView.loadAd(adRequest);
    }
}
```

If an ad is integrated this way, the banner appears after the app starts:

10:45
◀ BannerExample
Yandex Mobile Ads

To see how the banner ad will be displayed in the app, use a demo AdUnitId:

- demo-banner-yandex

Related information

Ad example

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 Advertising Network Help.

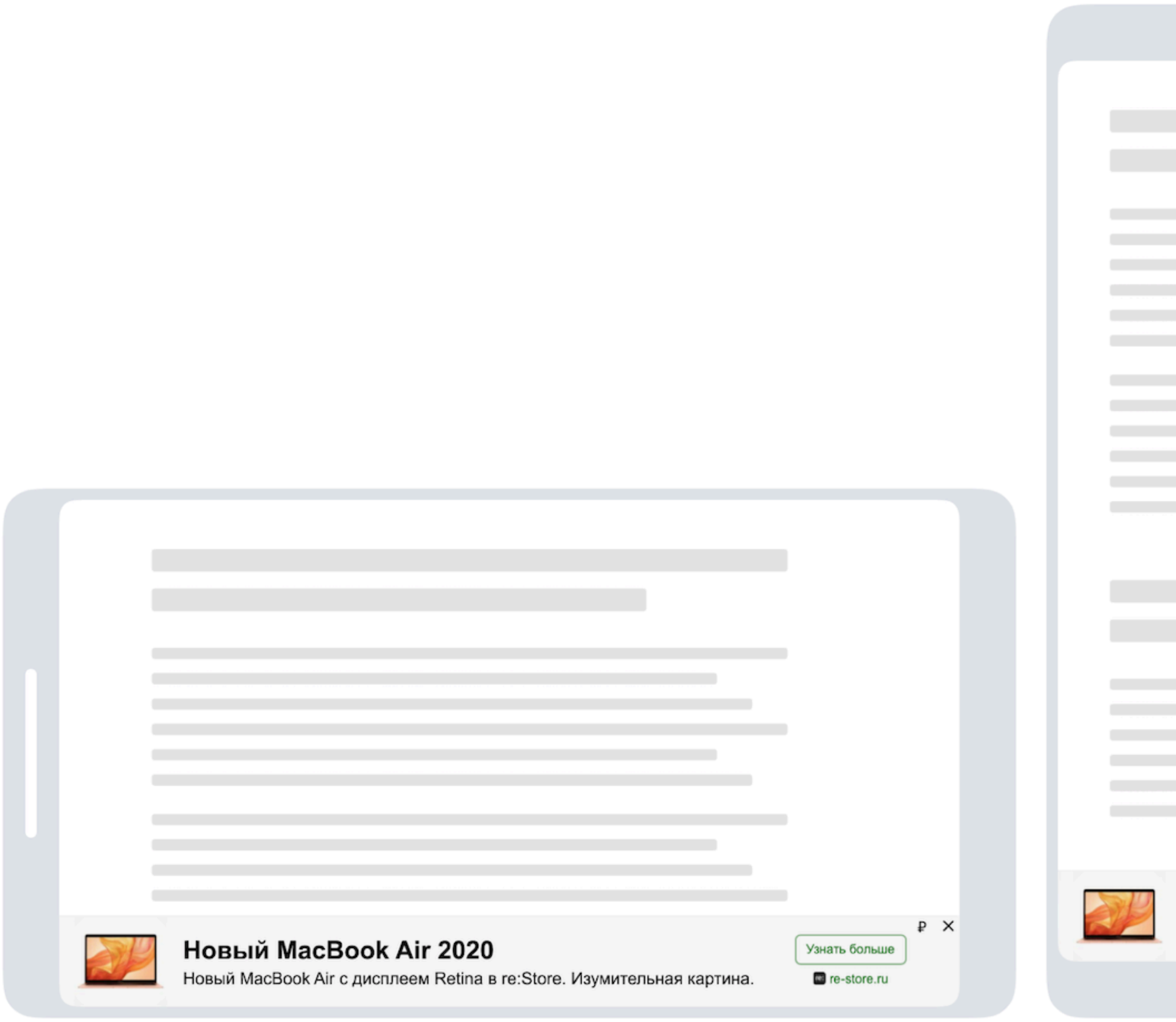
Types of adaptive banners

Banner with set width

Features:

1. An alternative to 320x50 banners (when determining the banner height, the aspect ratio of 320x50 is maintained).
2. The banner is fixed in place at the top or bottom of the screen (set up 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 adaptive banners is set using the stickySize method.

Examples of displaying adaptive banners:

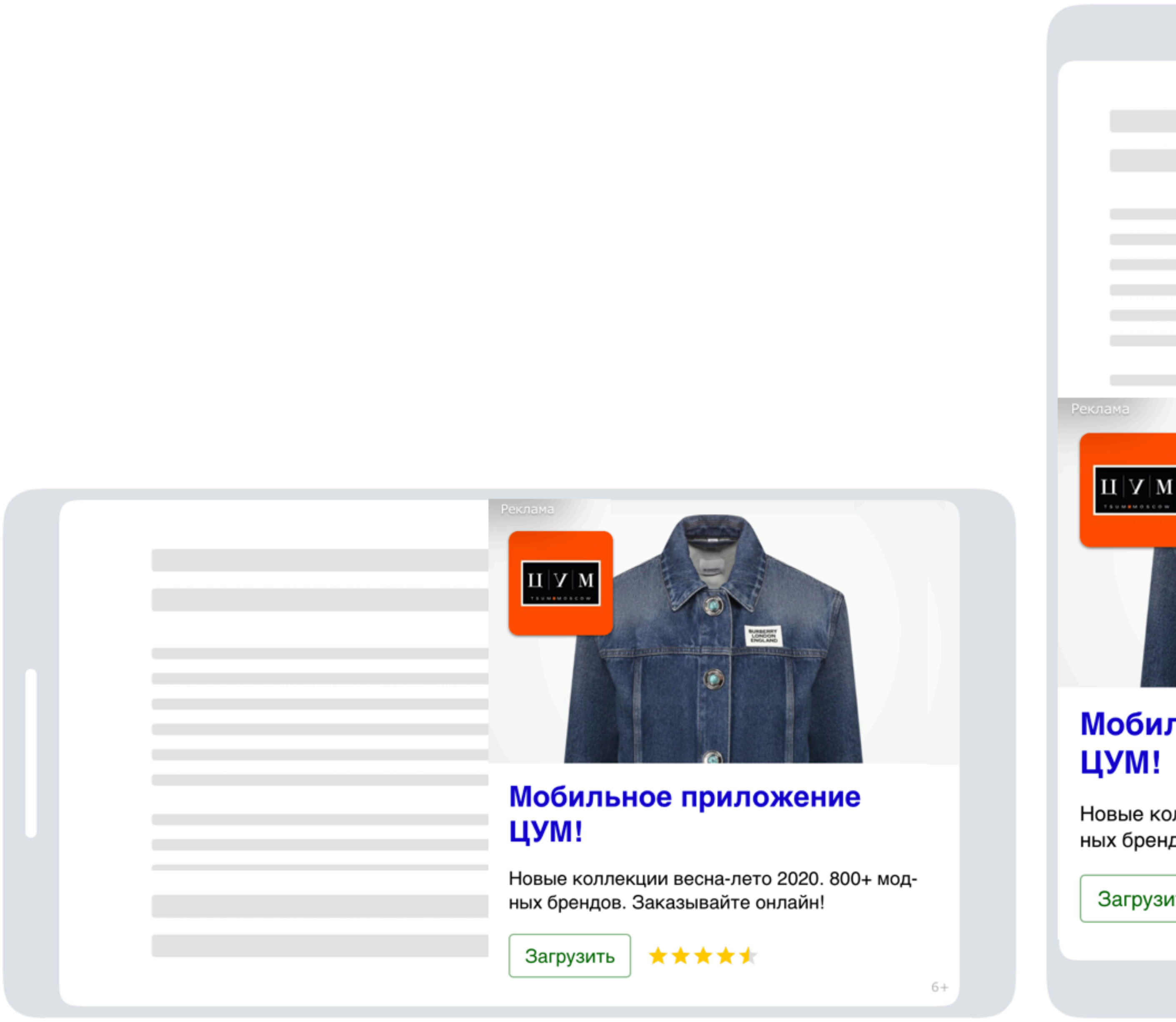

Новый MacBook Air 2020
Новый MacBook Air с дисплеем Retina в re:Store. Изумительная картина.
Узнать больше
re-store.ru

Banner with 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 flexibleSize(int width, int height) method.

Examples of displaying adaptive banners:


Реклама
ЦУМ
Мобильное приложение ЦУМ!
Новые коллекции весна-лето 2020. 800+ модных брендов. Заказывайте онлайн!
Загрузить
6+

Creating BannerAdview

1. Add an object of the `BannerAdView` class to the project using an XML file or programmatically.

```
// Creating an mBannerAdView instance using an XML file.
        mBannerAdView = (BannerAdView) findViewById(R.id.banner_view);

// Creating an mBannerAdView instance programmatically.
        mBannerAdView = new BannerAdView(this);
```

2. Set the `AdUnitId` using the setAdUnitId method.

```
mBannerAdView.setAdUnitId(<AdUnitId>)
```

`AdUnitId` is a unique identifier in R-M-XXXXXX-Y format, which is assigned in the Partner Interface.

3. Set the banner size using the setAdSize method.

   Banner with set width

   To set the width of an adaptive banner, call the stickySize(int width) method.

   ```
   mBannerAdView.setAdSize(AdSize.stickySize(AdSize.FULL_WIDTH));
   ```

   Banner with set width and height

   To set the width and height of an adaptive banner, call the flexibleSize(int width, int height) method.

   ```
   mBannerAdView.setAdSize(AdSize.flexibleSize(width, height));
   ```

4. After creating and configuring an instance of the `BannerAdView` class, you can set an AdEventListener on the ad object for tracking events (opening or closing the ad, exiting the app, and loading the ad successfully or unsuccessfully).

Loading ads

After creating and configuring the `BannerAdView` class object, load the ads. To load an ad, use the loadAd method, which takes the AdRequest object as a parameter (or Builder, which optionally accepts ad targeting data).

Example of working with adaptive banners

The following code demonstrates creating and configuring the `AdView` object, registering a listener, and loading an adaptive banner:

```
...
<LinearLayout>
    ...
    <com.yandex.mobile.ads.banner.BannerAdView
        android:id="@+id/banner_ad_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

```
...
private BannerAdView mBannerAdView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // Creating an mAdView instance.
        mBannerAdView = (BannerAdView) findViewById(R.id.banner_ad_view);
        mBannerAdView.setAdUnitId(AdUnitId);
        mBannerAdView.setAdSize(AdSize.stickySize(AdSize.FULL_WIDTH));

        // Creating an ad targeting object.
        final AdRequest adRequest = new AdRequest.Builder().build();

        // Registering a listener for tracking events in the banner ad.
        mBannerAdView.setAdEventListener(new BannerAdEventListener() {
            @Override
            public void onAdLoaded() {
                ...
            }

            @Override
            public void onAdFailedToLoad(AdRequestError adRequestError) {
                ...
            }

            @Override
            public void onLeftApplication() {
                ...
            }

            @Override
            public void onReturnedToApplication() {
                ...
            }
        });

        // Loading ads.
        mBannerAdView.loadAd(adRequest);
    }
}
```

Related information

Ad example

Classes and interfaces for working with banner ads

Warning:

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

Classes

- AdSize
- BannerAdView

Interfaces

- BannerAdEventListener

Enabling 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.

To enable advertising:

Create an InterstitialAd
Load the ad
Display the ad

Creating an InterstitialAd

1. Create an InterstitialAd class object. This object can only be created programmatically.

```
mInterstitialAd = new InterstitialAd(this);
```

2. Set the AdUnitId using the setAdUnitId method.

```
mInterstitialAd.setAdUnitId(AdUnitId);
```

`AdUnitId` is a unique identifier in R-M-XXXXXX-Y format, which is assigned in the Partner Interface.
3. After creating and configuring an instance of the InterstitialAd class, you can set an InterstitialEventListener on the ad object for tracking events (opening or closing the ad, exiting the app, and loading the ad successfully or unsuccessfully).

Loading ads

Note:

Any call of the Mobile Ads SDK should be made from the main thread.

After creating and configuring an instance of the AdView class, load ads. To load an ad, use the loadAd method, which takes the AdRequest object as a parameter (or Builder, which optionally accepts ad targeting data).

Displaying ads

An interstitial ad is loaded in the background immediately after the loadAd call. To display an interstitial ad, you must call the show method.

We recommend checking whether the ad has actually loaded. To do this, call the isLoaded method.

Note:

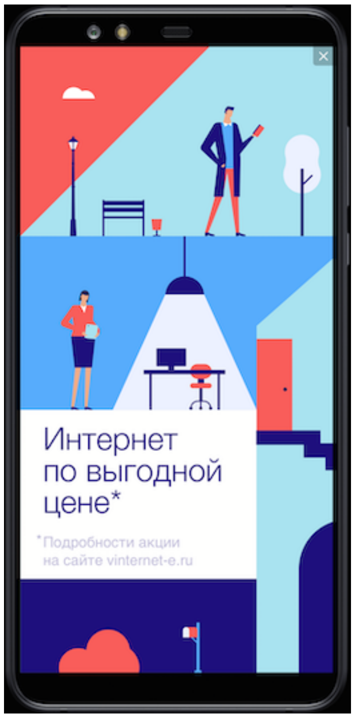
You don't need to check this if the show method is called after the onAdLoaded callback has been triggered.

Example of working with interstitial ads

The following code demonstrates creating and configuring the InterstitialAd object, registering a listener, and loading and displaying the interstitial ad:

```
...
public class InterstitialExample extends Activity {
    ...
    private static final String AdUnitId = "YOUR_AdUnitId";
    private InterstitialAd mInterstitialAd;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // Creating an InterstitialAd instance.
        mInterstitialAd = new InterstitialAd(this);
        mInterstitialAd.setAdUnitId(AdUnitId);

        // Creating an ad targeting object.
        final AdRequest adRequest = new AdRequest.Builder().build();

        // Registering a listener to track events in the ad.
        mInterstitialAd.setInterstitialAdEventListener(new InterstitialAdEventListener() {
            @Override
            public void onAdLoaded() {
                mInterstitialAd.show();
            }

            @Override
            public void onAdFailedToLoad(AdRequestError adRequestError) {
                ...
            }

            @Override
            public void onAdShown() {
                ...
            }

            @Override
            public void onAdDismissed() {
                ...
            }

            @Override
            public void onLeftApplication() {
                ...
            }

            @Override
            public void onReturnedToApplication() {
                ...
            }
        });

        // Loading ads.
        mInterstitialAd.loadAd(adRequest);
    }
}
```

If an ad is integrated this way, the ad unit appears after the app starts:



To see how the ad will be displayed in the app, use a demo AdUnitId:

- demo-interstitial-yandex

Related information

Ad example

Classes and interfaces for working with interstitial ads

Warning:

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

Classes

- InterstitialAd

Interfaces

- InterstitialAdEventListener

Enabling 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.

To enable advertising:

Create RewardedAd
Load the ad
Display the ad

Creating a rewarded ad

1. Add a RewardedAd class object.

   ```
   mRewardedAd = new RewardedAd(this);
   ```

2. Set the AdUnitId using the setAdUnitId method.

   ```
   mRewardedAd.setAdUnitId(AdUnitId);
   ```

   `AdUnitId` is a unique identifier in R-M-XXXXXX-Y format, which is assigned in the Partner interface.
3. If you are assigning rewards on the application side ("client-side reward"), implement the onRewarded interface method. It is called when the impression is registered and the user can be rewarded for viewing the ad. Use this chance to give the reward to the app user.

After creating and configuring a RewardedAd class object, you need to install a listener of the RewardedAdEventListener interface for the ad object in order to track events (like opening or closing the ad, exiting the app, or loading the app successfully or unsuccessfully).

Loading ads

Note:

Any call of the Mobile Ads SDK should be made from the main thread.

After creating and configuring the RewardedAd class object, load the ads. To load an ad, use the loadAd method, which takes the AdRequest object as a parameter (or Builder, which optionally accepts ad targeting data).

Displaying ads

Rewarded ads are loaded in the background immediately after the loadAd method is called. To display rewarded ads, call the show method.

We recommend checking whether the ad has actually loaded. To do this, call the isLoaded method.

Note:

You don't need to check this if the show method is called after the onAdLoaded callback has been triggered.

Example of working with rewarded ads

The following code demonstrates how to create and configure a RewardedAd object, register the listener, and load and display the rewarded ad:

```
...
public class RewardedAdExample extends Activity {
    ...
    private static final String AdUnitId = "YOUR_AdUnitId";
    private RewardedAd mRewardedAd;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
        // Creating a RewardedAd instance.
        mRewardedAd = new RewardedAd(this);
        mRewardedAd.setAdUnitId(AdUnitId);

        // Creating an ad targeting object.
        final AdRequest adRequest = new AdRequest.Builder().build();

        // Registering a listener to track events in the ad.
        mRewardedAd.setRewardedAdEventListener(new RewardedAdEventListener() {
            @Override
            public void onLoaded() {
                mRewardedAd.show();
            }

            @Override
            public void onRewarded(final Reward reward) {
                ...
            }

            @Override
            public void onAdFailedToLoad(final AdRequestError adRequestError) {
                ...
            }
```

```
            @Override
            public void onAdShown() {
                ...
            }

            @Override
            public void onAdDismissed() {
                ...
            }

            @Override
            public void onLeftApplication() {
                ...
            }

            @Override
            public void onReturnedToApplication() {
                ...
            }
        });

        // Loading ads.
        mRewardedAd.loadAd(adRequest);
    }
}
```

If advertising is integrated this way, the ad unit appears after the app starts.

To see how the ad will be displayed in the app, use a demo AdUnitId:

- `demo-rewarded-yandex`

Classes and interfaces for working with rewarded ads

Warning:

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

Classes

- RewardedAd

Interfaces

- Reward
- RewardedAdEventListener

Native ads

Warning:

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

Yandex Mobile Ads SDK lets you render ads using your own visual assets.

Native ads can adapt to the features and design of the app they are displayed in. The layout of a native ad matches the environment it is integrated into. This type of ad looks natural and contributes useful information to the app.

The SDK also provides a set of ready-made customizable visual assets (templates) that let you enjoy all the benefits of rendering from the platform's native tools without creating your own design.

There are three types of ways to load ads (ad loading):

- Loading one ad
- Loading multiple ads
- Ad slider

Loading and rendering ads

Warning:

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

Getting started

To ensure that the Yandex Mobile Ads SDK runs correctly, follow all the steps to attach an ad library.

Loading ads

Note:

Any call of the Mobile Ads SDK should be made from the main thread.

1. Create an instance of the NativeAdLoader class to get native ads.
2. Create a configuration for the `nativeAdRequestConfiguration` request using the NativeAdRequestConfiguration.Builder class. As the request parameters, you can use the ad unit ID, method for loading images, age, gender, and other data that might improve the quality of ad selection.
3. To get notifications (ad loaded successfully or failed with an error), create an instance of NativeAdLoadListener and set it as an event listener for the ad loader.
4. Start the ad loading process.

If display ad formats are enabled in the ad unit, add the container size to your ad request.

General ad request

```
final NativeAdLoader nativeAdLoader = new NativeAdLoader(this);
    nativeAdLoader.setNativeAdLoadListener(new NativeAdLoadListener() {
        @Override
        public void onAdLoaded(@NonNull final NativeAd nativeAd) {
            //bind nativeAd
        }

        @Override
        public void onAdFailedToLoad(@NonNull final AdRequestError error) {
            //log error
        }
    });

final NativeAdRequestConfiguration nativeAdRequestConfiguration =
    new NativeAdRequestConfiguration.Builder(AdUnitId).build();
nativeAdLoader.loadAd(nativeAdRequestConfiguration);
```

Tip:

We recommend that you keep a strong reference to the ad and its loader throughout the lifecycle of the screen hosting the assets.

Rendering ads

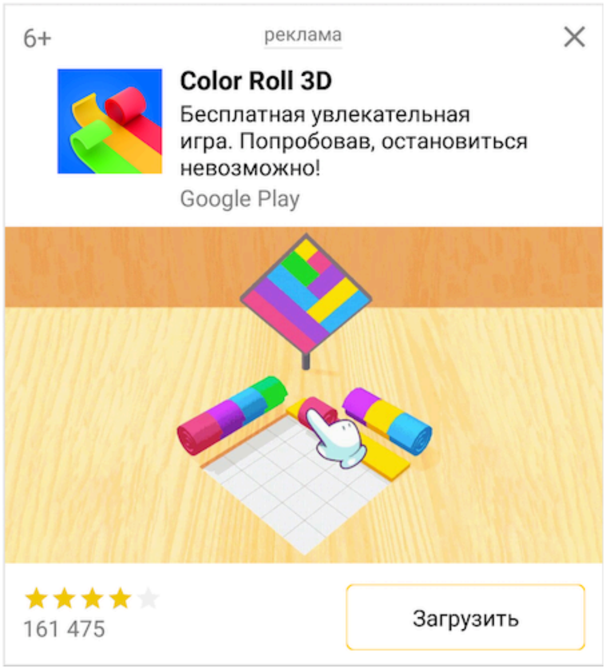
When the ad is loaded, you must render all of its assets. You can get the list of assets available in the ad from the NativeAd ad object.

There are two ways to configure the layout of an ad:

1. Layout using a template
2. Layout without a template

Layout using a template

Example of a native ad template


6+
реклама
Color Roll 3D
Бесплатная увлекательная игра. Попробовав, остановиться невозможно!
Google Play
161 475
Загрузить

The easiest way to work with native ads is to use a standard layout template: all you need is a few lines of code in the basic version.

The template already has the complete set of required assets and defines their arrangement relative to each other. The template works with any supported type of native ad.

```
final NativeBannerView nativeBannerView = new NativeBannerView(this);
nativeBannerView.setAd(nativeAd);
```

You can customize the native ad template. Learn more in Layout using a template.

Layout without a template

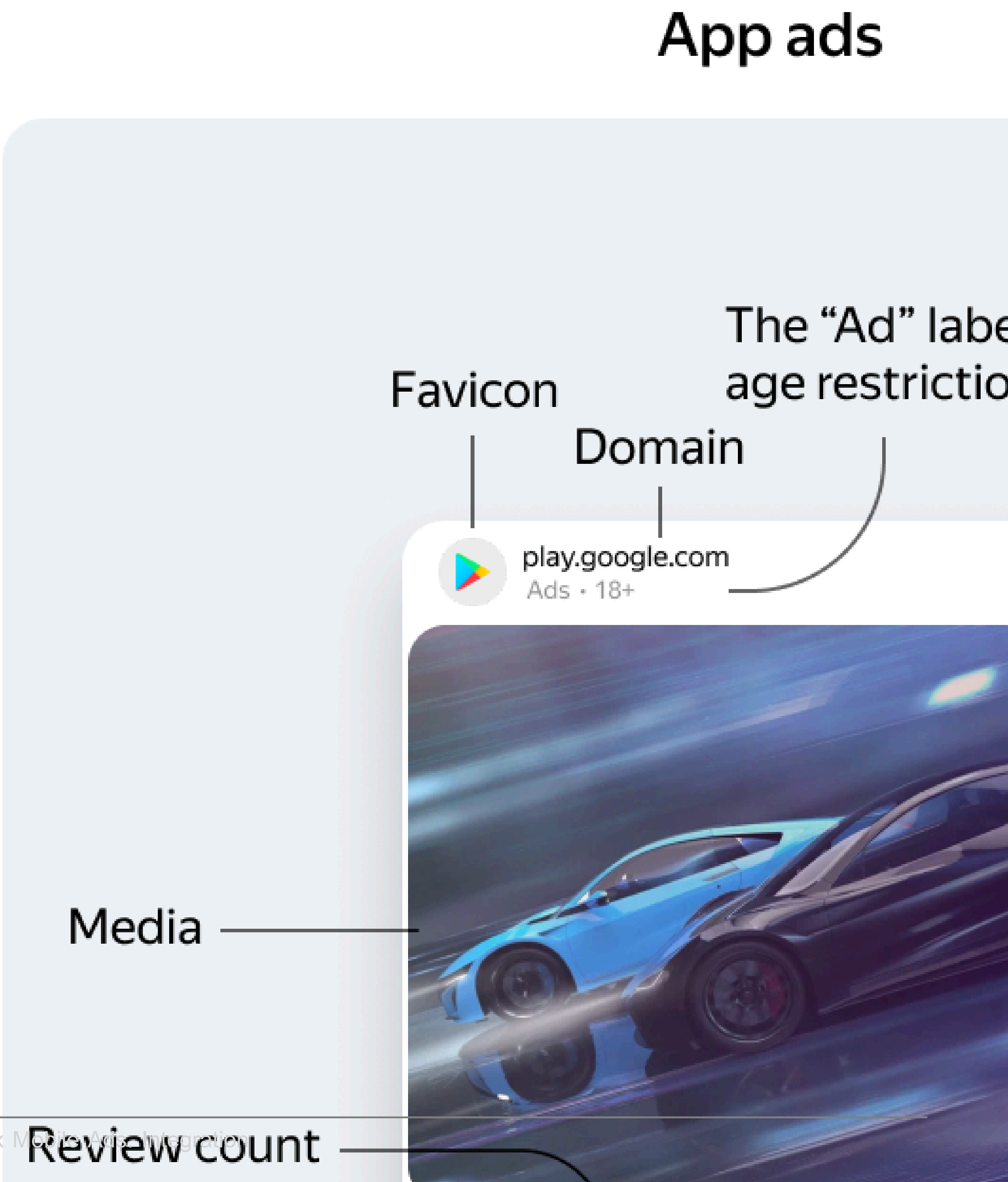
When the template settings aren't enough to get the desired effect, you can configure native ads manually.

With this method, you can lay out native ads yourself by positioning ad elements in respect of each other. Your ad may contain both mandatory and optional display assets. You can find their full list in Native ad assets.

Tip:

We recommend that you use a layout that includes the complete set of possible assets. Experience has shown that layouts with a complete set of assets are more clickable.

Example of a layout without using a template



Provide a View for each ad asset using the NativeAdViewBinder.Builder class instance. The class accepts the NativeAdView container as an argument. All the ad components must be defined as a subview of this container.

Link the generated ad layout with the NativeAd native ad object.

```
final NativeAdViewBinder nativeAdViewBinder = new NativeAdViewBinder.Builder(mNativeAdView)
                    .setAgeView((TextView) findViewById(R.id.age))
                    .setBodyView((TextView) findViewById(R.id.body))
                    .setCallToActionView((TextView) findViewById(R.id.call_to_action))
                    .setDomainView((TextView) findViewById(R.id.domain))
                    .setFaviconView((ImageView) findViewById(R.id.favicon))
                    .setFeedbackView((TextView) findViewById(R.id.feedback))
                    .setIconView((ImageView) findViewById(R.id.icon))
                    .setMediaView((MediaView) findViewById(R.id.media))
                    .setPriceView((TextView) findViewById(R.id.price))
                    .setRatingView((MyRatingView) findViewById(R.id.rating))
                    .setReviewCountView((TextView) findViewById(R.id.review_count))
                    .setSponsoredView((TextView) findViewById(R.id.sponsored))
                    .setTitleView((TextView) findViewById(R.id.title))
                    .setWarningView((TextView) findViewById(R.id.warning))
                    .build();

                    try {
                    nativeAd.bindNativeAd(nativeAdViewBinder);
                    mNativeAdView.setVisibility(View.VISIBLE);
                    } catch (final NativeAdException exception) {
                    //log exception
                    }
```

Note: mediaView size requirements when displaying video ads

Minimum size of an instance of the MediaView class that supports video playback: 300x160 or 160x300 dp (density-independent pixels).

To support video playback in native ad templates, we recommend setting the width for NativeBannerView to at least 300 dp. The correct height for mediaView will be calculated automatically based on the width to height ratio.

Loading multiple ads

Note:

Any call of the Mobile Ads SDK should be made from the main thread.

Yandex Mobile Ads SDK provides the option to load multiple ads in a single request (up to nine ads).

To make a bulk ad request, use the NativeBulkAdLoader class instance. The class instance provides the loadAds method with the COUNT argument where you can define the desired number of ads per request.

```
final NativeBulkAdLoader nativeBulkAdLoader = new NativeBulkAdLoader(this);
    nativeBulkAdLoader.setNativeBulkAdLoadListener(new NativeBulkAdLoadListener() {
        @Override
        public void onAdsLoaded(@NonNull final List<NativeAd> nativeAds) {
            for (final NativeAd nativeAd : nativeAds) {
                //bind native ad
            }
        }

        @Override
        public void onAdsFailedToLoad(@NonNull final AdRequestError error) {
            //log error
        }
    });

    final NativeAdRequestConfiguration nativeAdRequestConfiguration =
    new NativeAdRequestConfiguration.Builder(AdUnitId).build();
    nativeBulkAdLoader.loadAds(nativeAdRequestConfiguration, COUNT);
```

Note:

Yandex Mobile Ads SDK doesn't guarantee that the requested number of ads will be loaded. The resulting array will contain from 0 to COUNT NativeAd objects. You can render all the received ad objects separately from each other using the above methods for laying out native ads.

Ad slider

Warning:

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

Yandex Mobile Ads SDK lets you render a slider containing related ads. For more information about the slider, see this post.

The slider implements the native advertising paradigm. Publishers can adjust the ad layout based on its features and the design of their apps that render the slider.

Getting started

To ensure that the Yandex Mobile Ads SDK runs correctly, follow all the steps to attach an ad library.

Loading the slider

Note:

Any call of the Mobile Ads SDK should be made from the main thread.

1. Create an instance of the SliderAdLoader class to fetch ads into the slider.
2. Create a configuration for the `nativeAdRequestConfiguration` request using the NativeAdRequestConfiguration.Builder class. As the request parameters, you can use the ad unit ID, method for loading images, age, gender, and other data that might improve the quality of ad selection.
3. To get notifications (slider loaded successfully or failed with an error), create a SliderAdLoadListener instance and set it as an event listener for the ad loader.
4. Start the slider loading process.

Code example:

```
final SliderAdLoader sliderAdLoader = new SliderAdLoader(this);
sliderAdLoader.setSliderAdLoadListener(new SliderAdLoadListener() {
    @Override
    public void onSliderAdLoaded(@NonNull final SliderAd sliderAd) {
        //bind sliderAd
    }

    @Override
    public void onSliderAdFailedToLoad(@NonNull final AdRequestError error) {
        //log error
    }
});

final NativeAdRequestConfiguration nativeAdRequestConfiguration =
        new NativeAdRequestConfiguration.Builder(AdUnitId).build();
sliderAdLoader.loadSlider(nativeAdRequestConfiguration);
```

Tip:

We recommend that you keep a strong reference to the slider and its loader throughout the lifecycle of the screen hosting the assets.

Displaying the slider

When the slider is loaded, you must render all its assets.

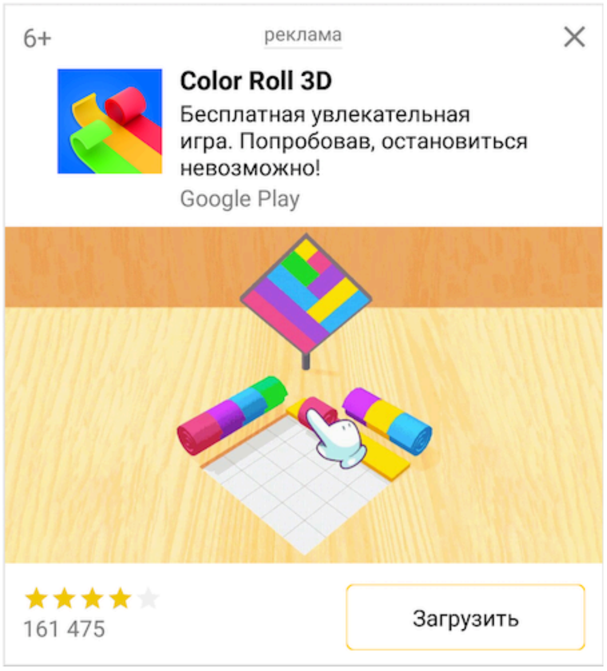
A successfully loaded slider contains one or more native ads with the same context. Ads in the slider must be displayed within the same shared container: otherwise, ad impressions won't count.

There are two ways to configure the slider layout:

1. Layout using a template
2. Layout without a template

Layout using a template

Example of a native ad template


6+
реклама
Color Roll 3D
Бесплатная увлекательная игра. Попробовав, остановиться невозможно!
Google Play
161 475
Загрузить

The easiest way to work with native ads is to use a standard layout template: all you need is a few lines of code in the basic version.

The template already has the complete set of required assets and defines their arrangement relative to each other. The template works with any supported type of native ad.

Making a slider using a template is easy. Link the SliderAd object to the root NativeAdView container and link all the ads in the slider to the template:

```
@Override
public void onSliderAdLoaded(@NonNull final SliderAd sliderAd) {
    mNativeAdView = (NativeAdView) findViewById(R.id.native_slider_ad_container);
    try {
        final NativeAdViewBinder sliderViewBinder =
            new NativeAdViewBinder.Builder(mNativeAdView).build();
sliderAd.bindSliderAd(sliderViewBinder);
        final List<NativeAd> nativeAds = sliderAd.getNativeAds();
        for (final NativeAd nativeAd : nativeAds) {
            final NativeBannerView nativeBannerView = new NativeBannerView(this);
            nativeBannerView.setAd(nativeAd);
            mNativeAdView.addView(nativeBannerView);
        }
    } catch (final NativeAdException exception) {
        //log error
    }
}
```

You can customize the native ad template. Learn more in Layout using a template.

Layout without a template

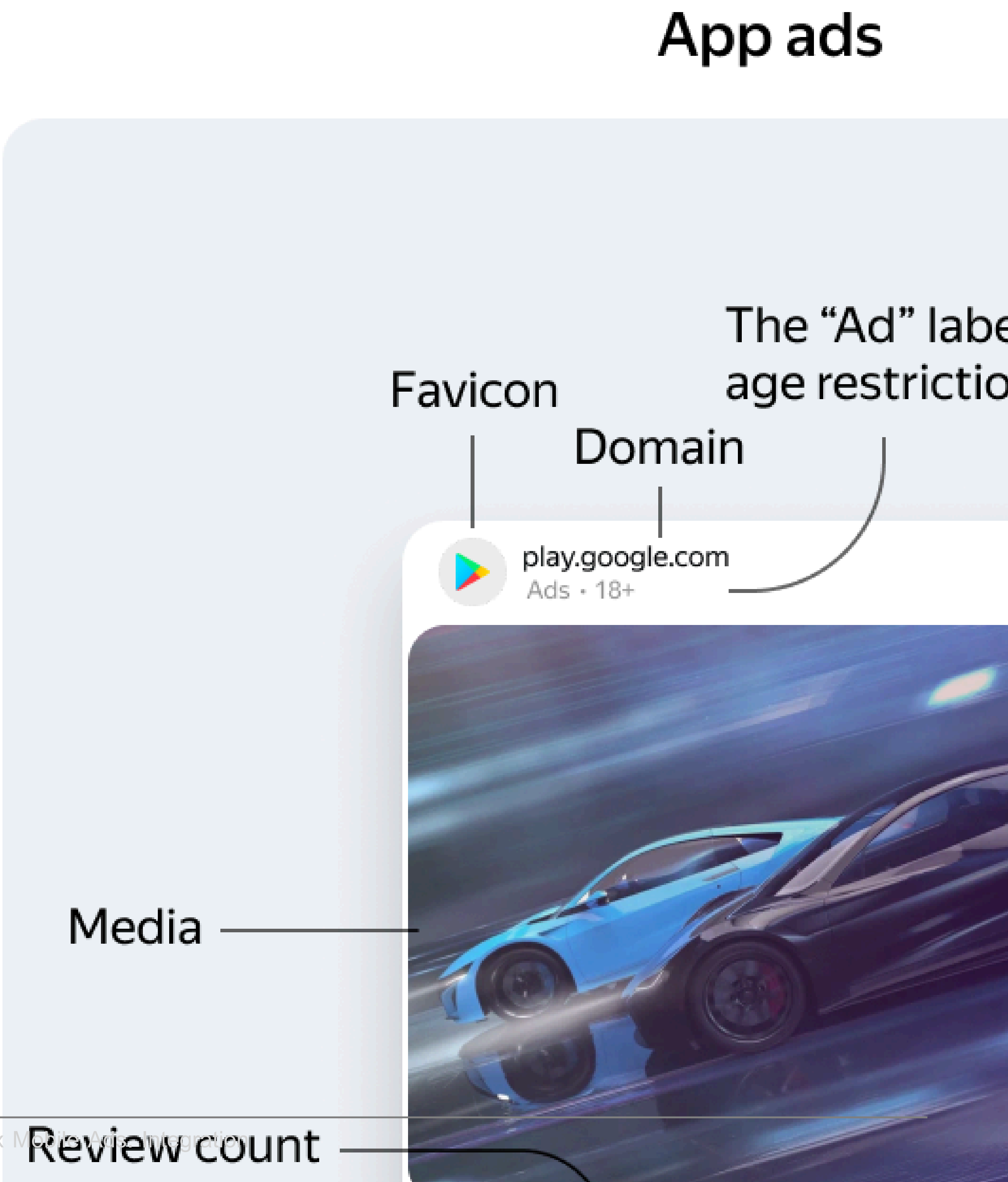
When the template settings aren't enough to get the desired effect, you can configure native ads manually.

With this method, you can lay out native ads yourself by positioning ad elements in respect of each other. Your ad may contain both mandatory and optional display assets. You can find their full list in Native ad assets.

Tip:

We recommend that you use a layout that includes the complete set of possible assets. Experience has shown that layouts with a complete set of assets are more clickable.

Example of a layout without using a template



Link the SliderAd object to the root NativeAdView object and link each ad in the slider to the child NativeAdView object.

Each ad in the slider is laid out using a standard native advertising layout method.

```
@Override
public void onSliderAdLoaded(@NonNull final SliderAd sliderAd) {
    mNativeAdView = (NativeAdView) findViewById(R.id.native_slider_ad_container);
    try {
        final NativeAdViewBinder sliderViewBinder =
            new NativeAdViewBinder.Builder(mNativeAdView).build();
sliderAd.bindSliderAd(sliderViewBinder);
        final List<NativeAd> nativeAds = sliderAd.getNativeAds();
        for (final NativeAd nativeAd : nativeAds) {
            final NativeAdView nativeAdView = mLayoutInflater.inflate(R.layout.widget_native_ad, mSliderAdView,
 false);
            final NativeAdViewBinder viewBinder = new NativeAdViewBinder.Builder(nativeAdView)
                    .setAgeView((TextView) nativeAdView.findViewById(R.id.age))
                    .setBodyView((TextView) nativeAdView.findViewById(R.id.body))
                    .setCallToActionView((Button) nativeAdView.findViewById(R.id.call_to_action))
                    .setDomainView((TextView) nativeAdView.findViewById(R.id.domain))
                    .setFaviconView((ImageView) nativeAdView.findViewById(R.id.favicon))
                    .setFeedbackView((Button) nativeAdView.findViewById(R.id.feedback))
                    .setIconView((ImageView) nativeAdView.findViewById(R.id.icon))
                    .setMediaView((MediaView) nativeAdView.findViewById(R.id.media))
                    .setPriceView((TextView) nativeAdView.findViewById(R.id.price))
                    .setRatingView((MyRatingView) nativeAdView.findViewById(R.id.rating))
                    .setReviewCountView((TextView) nativeAdView.findViewById(R.id.review_count))
                    .setSponsoredView((TextView) nativeAdView.findViewById(R.id.sponsored))
                    .setTitleView((TextView) nativeAdView.findViewById(R.id.title))
                    .setWarningView((TextView) nativeAdView.findViewById(R.id.warning))
                    .build();

            nativeAd.bindNativeAd(viewBinder);
            mNativeAdView.addView(nativeBannerView);
        }
    } catch (final NativeAdException exception) {
        //log error
    }
}
```

Note: mediaView size requirements when displaying video ads

Minimum size of an instance of the MediaView class that supports video playback: 300x160 or 160x300 dp (density-independent pixels).

To support video playback in native ad templates, we recommend setting the width for NativeBannerView to at least 300 dp. The correct height for mediaView will be calculated automatically based on the width to height ratio.

Advertising requirements

Warning:

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

Basic requirements

The ad impression won't be registered if even one of the following conditions isn't met:

1. The ad `View`, along with each of its `superviews`, must meet the following conditions:
    - The `View` must be visible.
    - The `View` must be non-transparent.
2. All required assets must meet the following conditions:
    - The asset, as well as its `superview`, must be fully visible.
    - The asset must be completely inside the ad `View`.
    - The asset must be within the hierarchy of the ad `View`.
    - The asset must be non-transparent.
    - The content of the asset must be truthful.
3. The app must be active (not running in the background).
4. At a single point in time, a loaded ad can only be served in one `View`. Simultaneous display of a single ad in multiple `Views` may result in a lost impression.

Note: If all the conditions are met but the impression was not registered, there will be another attempt to register the impression later.

Restrictions on changing assets

1. Don't edit the text content of assets.
2. Don't edit the content of images.
3. If you stretch an image, you must maintain the aspect ratio.
4. Don't crop an image by more than 20%. Permitted: masking, container resizing.

Native ad assets

Warning:

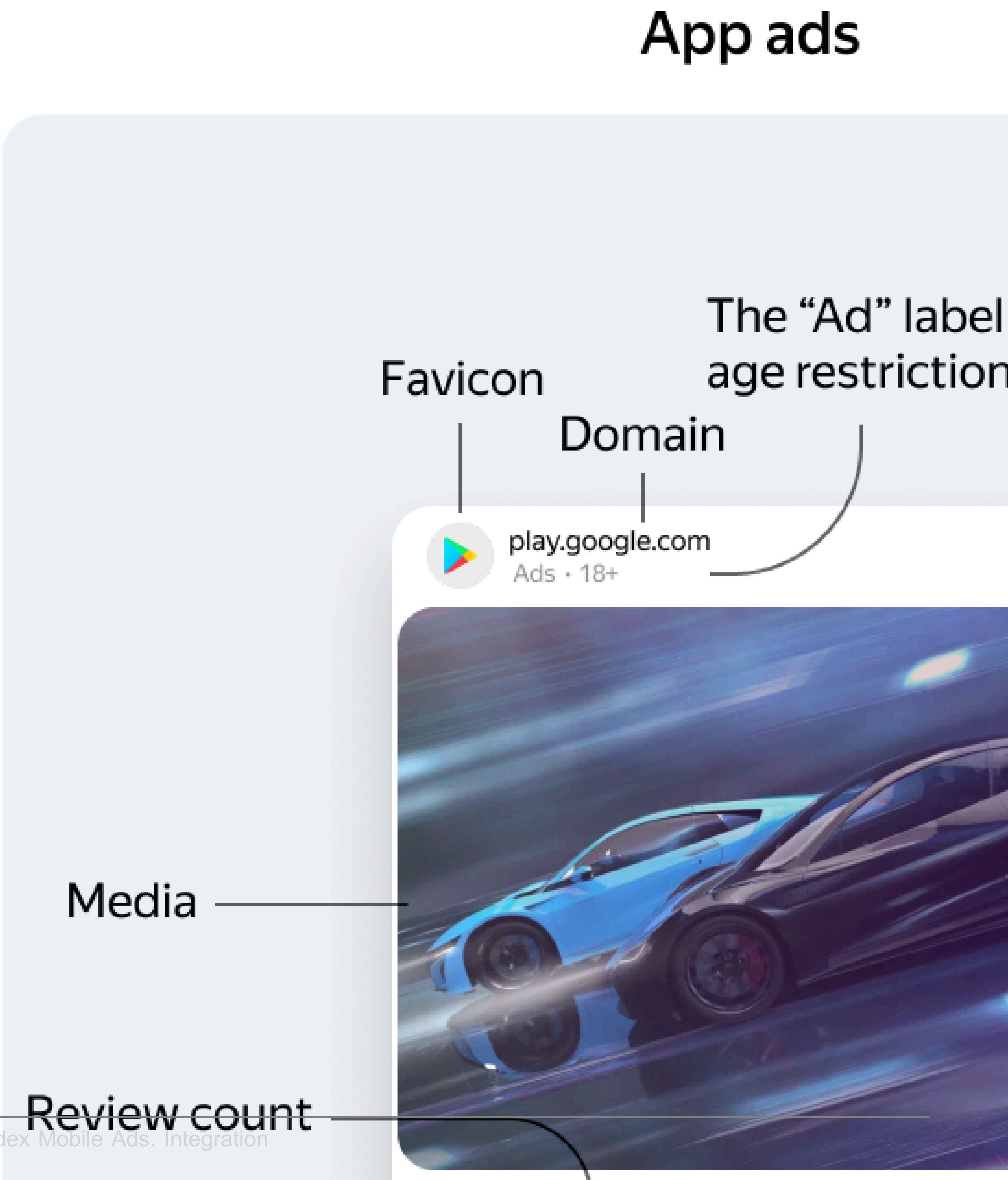
This is an archived version of the documentation. Actual documentation for all platforms can be found here.

Note:

Read the advertising requirements.

Asset list

The library includes both the required and optional assets. According to the advertising rules, it's enough that you render only the required assets. In practice, layouts that use a complete set of assets are more clickable. That's why we recommend that you use the design based on a complete set of supported assets.


App ads
The “Ad” label
age restriction
Favicon
Domain
play.google.com
Ads • 18+
Media
Review count

Ad element	Asset	Type	Required
Title	title	TextView	Yes
Domain	domain	TextView	Yes
Disclamer	warning	TextView	Yes
The “Ad” label and the age restriction label	sponsored	TextView	Yes
Menu icon	feedback	ImageView	Yes
Action button	callToAction	TextView	Yes
Media	media	MediaView	Yes
App icon	icon	ImageView	Yes, for Ads for Mobile Apps
Price	price	TextView	Yes, for Ads for Mobile Apps
Favicon	favicon	ImageView	No
Review count	reviewCount	TextView	No
Rating	rating	View implements Rating interface	No
Text	body	TextView	No

The list of required assets defines the set of data for which, if those assets are present, a View must be provided for rendering.

Tip:

We recommend that you use a layout that can render a complete set of both required and optional ad assets.

Asset layout

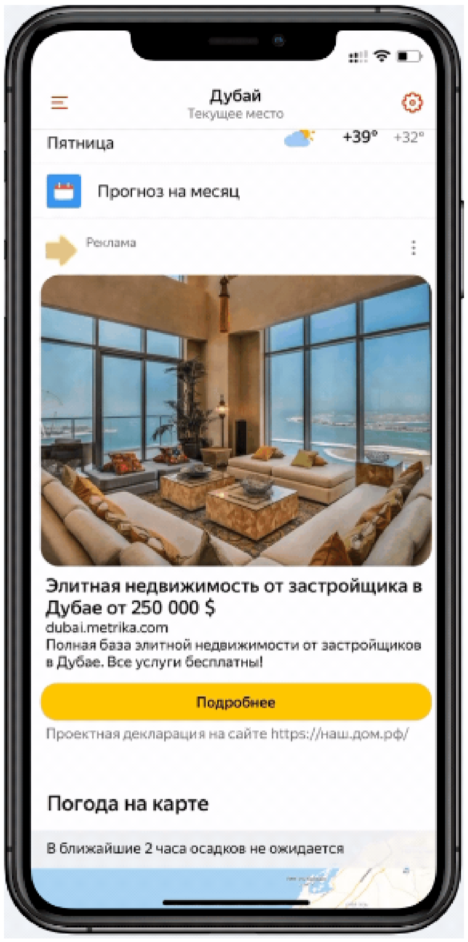
feedback

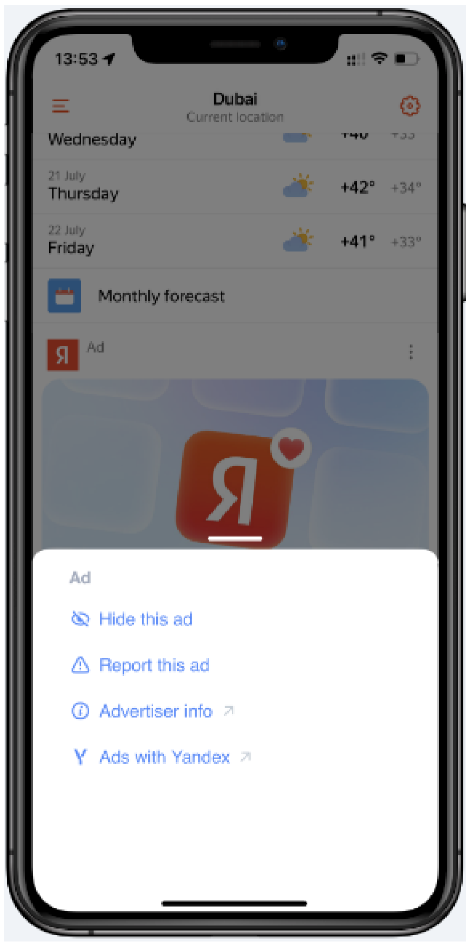
Menu icon. Required asset. The user can click the ⋮ menu icon to hide or report an ad.

The menu icon is added to the upper-right corner of the ad.

Note:

The developer must define what to do with the ad after the reason for closing it is chosen (for example, hide the ad or show a text). If there is no further action defined, the SDK will register the reason for closing, but the ad will not be hidden.


Дубай
Текущее место
Пятница
+39° +32°
Прогноз на месяц
Реклама
Элитная недвижимость от застройщика в Дубае от 250 000 $
dubai.metrika.com
Полная база элитной недвижимости от застройщиков в Дубае. Все услуги бесплатны!
Подробнее
Проектная декларация на сайте https://наш.дом.рф/
Погода на карте
В ближайшие 2 часа осадков не ожидается



Layout options:

feedback_dark_dots_with_background

White menu icon, with dark dots and a translucent background. Default value.

feedback_light_dots

Menu icon without a background, with light dots.

Code example:

```
final NativeAdLoader loader = new NativeAdLoader(this);
final HashMap<String, String> parameters = new HashMap<String, String>(){{
 put("feedback_image", "feedback_light_dots");
}};
final NativeAdRequestConfiguration nativeAdRequestConfiguration =
 new NativeAdRequestConfiguration.Builder("demo-native-app-yandex")
  .setParameters(parameters)
mNativeAdLoader.loadAd(nativeAdRequestConfiguration);
```

feedback_dark_dots

Menu icon without a background, with dark dots.

Code example:

```
final NativeAdLoader loader = new NativeAdLoader(this);
final HashMap<String, String> parameters = new HashMap<String, String>(){{
 put("feedback_image", "feedback_dark_dots");
}};
final NativeAdRequestConfiguration nativeAdRequestConfiguration =
 new NativeAdRequestConfiguration.Builder("demo-native-app-yandex")
  .setParameters(parameters)
mNativeAdLoader.loadAd(nativeAdRequestConfiguration);
```

Example of layout using a template

Warning:

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

To customize the layout and design, you can use a standard template or create your own design based on a standard template.

Creating your own template-based layout

1. Create an instance of the NativeBannerView class and make the preferred settings for it:

```
final NativeBannerView nativeBannerView =
      new NativeBannerView(getApplicationContext());
final NativeTemplateAppearance nativeTemplateAppearance =
      new NativeTemplateAppearance.Builder()
                 .withBannerAppearance(new BannerAppearance.Builder()
                         .setBackgroundColor(Color.GRAY).build())
                 .withTitleAppearance(new TextAppearance.Builder()
                         .setTextColor(Color.BLUE).build())
                 ...
                 .build();
nativeBannerView.applyAppearance(nativeTemplateAppearance);
```

Note:

A NativeBannerView instance can be created either programmatically or using an XML file.

Example of layout configuration

```
final NativeBannerView nativeBannerView = new NativeBannerView(getApplicationContext());
final NativeTemplateAppearance nativeTemplateAppearance =
      new NativeTemplateAppearance.Builder()
              // Setting the color for the ad frame.
              .withBannerAppearance(new BannerAppearance.Builder()
                      .setBorderColor(Color.YELLOW).build())

              // Setting the button parameters.
              .withCallToActionAppearance(new ButtonAppearance.Builder()
              // Setting the font color and size for the action button label.
                      .setTextAppearance(new TextAppearance.Builder()
                               .setTextColor(Color.BLUE)
                               .setTextSize(14f).build())

                      // Setting the button color for the normal state and the clicked state.
                      .setNormalColor(Color.TRANSPARENT)
                      .setPressedColor(Color.GRAY)
                      // Setting the color and thickness of the button border.
                      .setBorderColor(Color.BLUE)
                      .setBorderWidth(1f).build())

              // Setting the image width and the sizing constraint.
              .withImageAppearance(new ImageAppearance.Builder()
                      .setWidthConstraint(new SizeConstraint(SizeConstraint
                      .SizeConstraintType.FIXED, 60f)).build())

             //  Setting the font size and color for the age restriction label.
             .withAgeAppearance(new TextAppearance.Builder()
                      .setTextColor(Color.GRAY)
                      .setTextSize(12f).build())

             // Setting the font size and color for the main ad text.
             .withBodyAppearance(new TextAppearance.Builder()
                      .setTextColor(Color.GRAY)
                      .setTextSize(12f).build())

             // Setting the color for filled stars in the rating.
             .withRatingAppearance(new RatingAppearance.Builder()
                      .setProgressStarColor(Color.CYAN).build())

             // Setting the font size and color for the ad title.
             .withTitleAppearance(new TextAppearance.Builder()
                      .setTextColor(Color.BLACK)
                      .setTextSize(14f).build())

             .build();

// Applying the settings.
nativeBannerView.applyAppearance(nativeTemplateAppearance);
```

Note:

When creating your own template-based design, you don't have to make the preferred settings for all the visual elements. Any elements that don't have preferred settings are configured with the default values.

Classes and interfaces for working with native ads

Warning:

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

Classes

- BannerAppearance
- BannerAppearance.Builder
- ButtonAppearance
- ButtonAppearance.Builder
- HorizontalOffset
- ImageAppearance
- ImageAppearance.Builder
- NativeAdAssets
- NativeAdImage
- NativeAdLoader
- NativeAdRequestConfiguration
- NativeAdRequestConfiguration.Builder
- NativeAdMedia
- NativeAdViewBinder
- NativeAdViewBinder.Builder
- NativeBannerView
- NativeTemplateAppearance
- NativeTemplateAppearance.Builder
- MediaView
- RatingAppearance
- RatingAppearance.Builder
- SizeConstraint
- TextAppearance
- TextAppearance.Builder

Interfaces

- NativeAdEventListener
- NativeAdImageLoadingListener
- rating

InStream ads

Warning:

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

InStream is an ad format that lets you monetize your app by serving ads while video content is playing.

An InStream ad consists of a scenario with multiple video units. The type of video unit in an InStream scenario determines how the video ad should be played relative to the main video content.

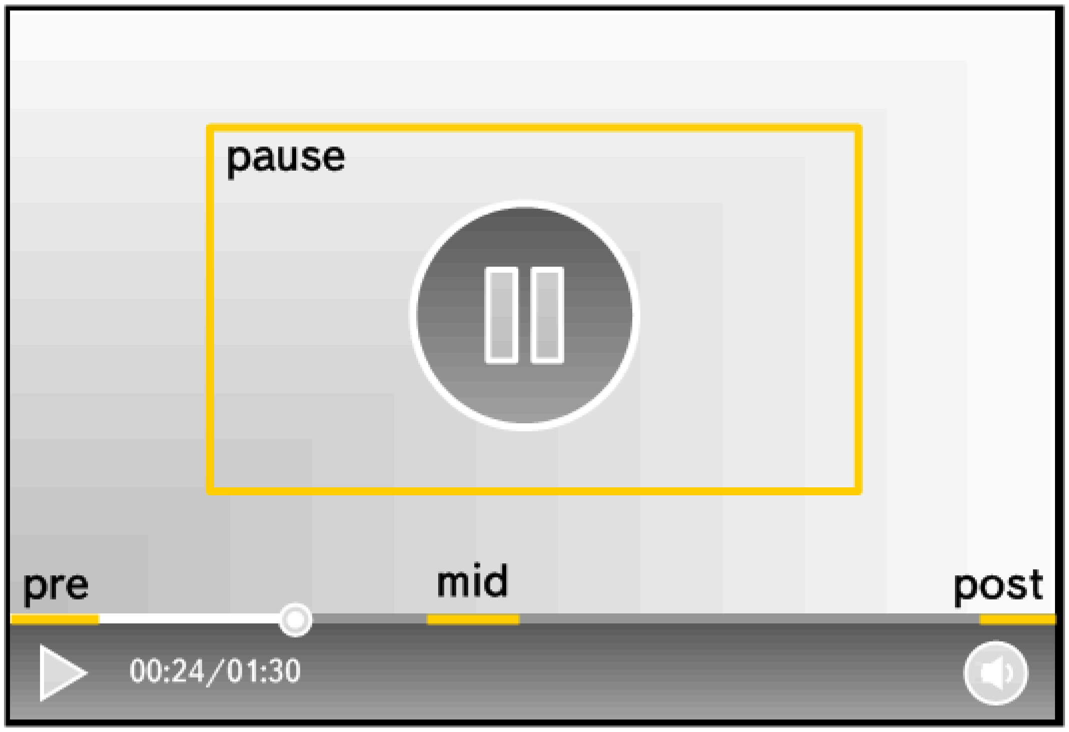
About InStream

To set up an InStream scenario, create a video resource in the Partner interface. Once created, the video resource is assigned a unique identifier (`Page ID`). This ID should be used in the Mobile Ads SDK.

To increase ad revenue, you can set up playing multiple ads within a single ad placement: an AdPod. You can set up the AdPod for an ad placement in the Partner interface.

Types of video units supported by the Mobile Ads SDK:

- Pre-Roll — A video ad is played before the main content.
- Mid-Roll — A video ad is played at a certain time within the main content.
- Post-Roll — A video ad is played after the main content.
- Pause-Roll — A video ad is played when the user clicks the pause button.
- In-Roll — A video ad is played anywhere in the video when a certain point is reached.



1. In-roll

API for working with InStream ads

There are multiple APIs for working with InStream ads:

ExoPlayer AdsLoader API

An API for basic integration of InStream ads into ExoPlayer. It lets you quickly integrate the display of InStream ads. The API supports Pre-Roll, Mid-Roll, and Post-Roll ad breaks.

Restriction:

1. A specific version of ExoPlayer is required.
2. It doesn't support In-Roll and Pause-Roll.

InStream API

An API for advanced integration of InStream ads. It lets you support playing any type of ad break and use your own implementation of a player. InStream ads consist of ad breaks that play automatically and manually.

Pre-Roll, Mid-Roll, and Post-Roll ad breaks are played automatically using the InstreamAdBinder API. To play In-Roll and Pause-Roll ad breaks manually, use the In-Roll API and Pause-Roll API, respectively.

Note:

You can also use the InstreamAdBinder API, In-Roll API, and Pause-Roll API concurrently if you:

1. Use different instances of the ad player.
2. Don't start the Pause-Roll and In-Roll APIs for playing ads if the main video was paused using the InStreamAdBinder API.

Basic integration (ExoPlayer AdsLoader API)

Warning:

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

YandexAdsLoader is a simplified API for integrating InStream ads using ExoPlayer. YandexAdsLoader supports playing Pre-Roll, Mid-Roll, and Post-Roll ad breaks.

This type of integration is suitable for apps that do not require advanced InStream API features.

Enable using Gradle

Add the following dependencies to the build.gradle file at the application level:

```
dependencies {
    ...
    implementation 'com.yandex.android:mobileads:5.10.0'
    implementation 'com.google.android.exoplayer:exoplayer-core:2.18.1'
}
```

Rendering ads

1. Add com.google.android.exoplayer2.ui.PlayerView to the app layout.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <com.google.android.exoplayer2.ui.PlayerView
        android:id="@+id/player_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

</FrameLayout>
```

2. Create a configuration for the `instreamAdRequestConfiguration` request using the InstreamAdRequestConfiguration.Builder class. Pass the page identifier (`Page ID`) from the Partner interface as a request parameter.
3. Create an instance of the YandexAdsLoader class to serve and upload InStream ads.
4. Create an instance of the `DefaultMediaSourceFactory` class and add the created YandexAdsLoader instance and `PlayerView` to it.

```
val userAgent = Util.getUserAgent(this, getString(R.string.app_name))
val dataSourceFactory = DefaultDataSourceFactory(this, userAgent)
val mediaSourceFactory = DefaultMediaSourceFactory(dataSourceFactory)
    .setAdsLoaderProvider { yandexAdsLoader }
    .setAdViewProvider(playerView)
```

5. Create an `ExoPlayer` instance and add the created `mediaSourceFactory` to it.
6. Add the created `ExoPlayer` instance to `PlayerView` and YandexAdsLoader.

```
val player = SimpleExoPlayer.Builder(this)
    .setMediaSourceFactory(mediaSourceFactory).build()
playerView.player = player
yandexAdsLoader.setPlayer(player)
```

7. Create a `MediaItem` instance and add a link to the video to be played and `YandexAdsLoader.AD_TAG_URI` to it.

```
val contentVideoUrl = getString(R.string.content_url_for_instream_ad)
val mediaItem = MediaItem.Builder()
    .setUri(contentVideoUrl)
    .setAdTagUri(YandexAdsLoader.AD_TAG_URI).build()
```

8. Start playing the video with the created `MediaItem` instance.

```
player.apply {
    setMediaItem(mediaItem)
    prepare()
    playWhenReady = true
}
```

Advanced integration (InStream API)

Warning:

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

The InStream API is an advanced API for setting up and managing ad loading and playing InStream ads. It lets you support playing any type of ad break and use your own implementation of a player. InStream ads consist of ad breaks that play automatically and manually.

Pre-Roll, Mid-Roll, and Post-Roll ad breaks are played automatically using the InstreamAdBinder API. To play In-Roll and Pause-Roll ad breaks manually, use the In-Roll API and Pause-Roll API, respectively.

Note:

You can also use the InstreamAdBinder API, In-Roll API, and Pause-Roll API concurrently if you:

1. Use different instances of the ad player.
2. Don't start the Pause-Roll and In-Roll APIs for playing ads if the main video was paused using the InStreamAdBinder API.

How it works

A loaded InStream ad object contains a schedule for playing ad breaks. Each ad break is described by an InstreamAdBreak object. An ad break may have one of the following types: Pre/Mid/Post/In/Pause-Roll. You can play Pre/MidPost-Roll ad breaks using the InstreamAdBinder API. To play In/Pause-Roll ad breaks, use the In-Roll API and Pause-Roll API, respectively.

The VideoPlayer interface is used to interact with the main video content. To play an ad break inside an ad placement, use the InstreamAdPlayer interface.

Playing Pre/Mid/Post-Roll video ads

InstreamAdBinder tracks the progress of playing the main video and shows ad breaks based on the video resource settings in the Partner interface.

InstreamAdBinder does not directly control the rendering of a video ad in PlayerView. Video ads must be played on the app side based on signals from player interfaces transmitted to InstreamAdBinder. InstreamAdBinder signals the start of playing an ad break by calling VideoPlayer.pauseVideo() and the end by calling VideoPlayer.resumeVideo().

When calling VideoPlayer.pauseVideo() on the app side, it's necessary to hide the main video controls, pause the main video, and start playing the video ad. On the ad SDK side, after calling the method, advertising controls are displayed inside the InstreamAdView container and the InstreamAdPlayer.playAd() method is called to start playing the video ad.

When calling VideoPlayer.resumeVideo() on the app side, it's necessary to return the main video controls and resume playing the main video. On the ad SDK side, ad controls inside the InstreamAdView container are removed before calling the method.

Playing In/Pause-Roll video ads

The In/Pause-Roll API doesn't directly control the rendering of a video ad in PlayerView. Video ads must be played on the app side based on signals from player interfaces transmitted to In/Pause-Roll. In/Pause-Roll signals the start of playing an ad break by calling InstreamAdBreakEventListener.onInstreamAdBreakStarted() and the end by calling InstreamAdBreakEventListener.onInstreamAdBreakCompleted() or InstreamAdBreakEventListener.onInstreamAdBreakError().

When calling InstreamAdBreakEventListener.onInstreamAdBreakStarted() on the app side, it's necessary to hide the main video controls and pause the main video. On the ad SDK side, after calling the method, advertising controls are displayed inside the InstreamAdView container and the InstreamAdPlayer.playAd() method is called to start playing the video ad.

When calling InstreamAdBreakEventListener.onInstreamAdBreakCompleted() or InstreamAdBreakEventListener.onInstreamAdBreakError() on the app side, return the main video controls and resume playing the main video. On the ad SDK side, ad controls are removed from the InstreamAdView container before calling the methods.

Loading ads

1. Create an instance of the InstreamAdLoader class to get InStream ads.
2. Set up notifications (ad loaded successfully or failed with an error): create an InstreamAdLoadListener instance and set it as an event listener for the ad loader.
3. Create a configuration for the `instreamAdRequestConfiguration` request using the InstreamAdRequestConfiguration.Builder class. Pass the page identifier (`Page ID`) from the Partner interface as a request parameter.
4. Load ads using the InstreamAdLoader.loadInstreamAd method and pass `Context` and `instreamAdRequestConfiguration` to it.

Code example:

To test the integration, use the demo Page ID: demo-instream-vmap-yandex.

```
final InstreamAdLoader instreamAdLoader = new InstreamAdLoader(context);
instreamAdLoader.setInstreamAdLoadListener(new InstreamAdLoadListener() {
        @override
        public void onInstreamAdLoaded(@NonNull final InstreamAd instreamAd) {
        ...
        }

        @override
        public void onInstreamAdFailedToLoad(@NonNull final String reason) {
        ...
        }
    });

final InstreamAdRequestConfiguration instreamAdRequestConfiguration =
    new InstreamAdRequestConfiguration.Builder(PAGE_ID).build();
instreamAdLoader.loadInstreamAd(this, instreamAdRequestConfiguration);
```

Rendering ads

Playing Pre/Mid/Post-Roll video ads

1. Implement the InstreamAdPlayer and VideoPlayer interfaces.

   For more information about using and implementing the appropriate methods, see the reference guide sections Package com.yandex.mobile.ads.instream.player.ad and Package com.yandex.mobile.ads.instream.player.content. Additionally, see a test implementation example.

   Warning:

   To make implementation easier, we recommend using different instances of players to play video ads and content.
2. Add InstreamAdView to the app layout. InstreamAdView must contain PlayerView to play video ads in.

   Code example:

   Restriction:

   A container must be at least 300dp x 160dp in size.

   ```
   <com.yandex.mobile.ads.instream.player.ad.InstreamAdView
       android:id="@+id/instream_ad_view"
       android:layout_width="match_parent"
       android:layout_height="wrap_content">

           <PlayerView
               android:id="@+id/player_view"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"/>

   </com.yandex.mobile.ads.instream.player.ad.InstreamAdView>
   ```
3. Create an InstreamAdBinder object: pass `Context`, the loaded InstreamAd object, and the InstreamAdPlayer and VideoPlayer implementations to the builder.

   Set up notifications about the ad playing progress (ready to play the video ad, the video ad played or failed to play), create an instance of InstreamAdListener and set it as an event listener for InstreamAdBinder.

   ```
   mInstreamAdBinder = new InstreamAdBinder(context, mInstreamAd, mYandexAdPlayer, mContentVideoPlayer);
   mInstreamAdBinder.setInstreamAdListener(...);
   ```

4. To start playing a Pre-Roll ad break faster, preload it in advance by calling the InstreamAdBinder.prepareAd() method.

   ```
   private void preparePrerollAd(@NonNull final InstreamAdBinder instreamAdBinder) {
       instreamAdBinder.setInstreamAdListener(new InstreamAdListener() {
           ...
           public void onInstreamAdPrepared() {
               addInstreamAdBinderToPreloadedAdQueue(instreamAdBinder);
           }
           ...
       });
       instreamAdBinder.prepareAd();
   }
   ```

5. Call the InstreamAdBinder.bind(instreamAdView) method for the created InstreamAdBinder object. Pass InstreamAdView as a parameter. After that, the InStream SDK starts to automatically track the progress of playing the main video and manage the way video ads are played.

   ```
   mInstreamAdBinder.bind(mInstreamAdView);
   ```

6. When playing InStream ads in the list, use the InStreamBinder.unbind() method when the cell with the ad is invalidated in the list. To implement a reused pool of players for scrolling, call InstreamAdbinder.invalidateAdPlayer() when reusing the ad player linked to InstreamAdBinder and InstreamAdBinder.invalidateVideoPlayer() when reusing the main content player.
7. When you stop using InStreamAdBinder, reset the state.

   ```
   public void onDestroy() {
       instreamAdBinder.unbind();
       instreamAdBinder.invalidateVideoPlayer();
       instreamAdBinder.invalidateAdPlayer();
       instreamAdBinder.setInstreamAdListener(null);
       instreamAdBinder.setVideoAdPlaybackListener(null);

       super.onDestroy();
   }
   ```

Playing In/Pause-Roll video ads

Note:

Setting up playing Pause-Roll ad breaks is similar to In-Roll. To do this, replace In-Roll classes/methods with Pause-Roll ones.

1. Implement the InstreamAdPlayer interface.

   For more information about the methods and their implementation, see the Package com.yandex.mobile.ads.instream.player.ad reference section. Additionally, see a test implementation example.

   Warning:

   To make implementation easier, we recommend using different instances of players to play video ads and content.
2. Add InstreamAdView to the app layout. InstreamAdView must contain PlayerView to play video ads in.

   Code example:

   Restriction:

   A container must be at least 300dp x 160dp in size.

   ```
   <com.yandex.mobile.ads.instream.player.ad.InstreamAdView
       android:id="@+id/instream_ad_view"
       android:layout_width="match_parent"
       android:layout_height="wrap_content">

           <PlayerView
               android:id="@+id/player_view"
               android:layout_width="match_parent"
               android:layout_height="wrap_content"/>

   </com.yandex.mobile.ads.instream.player.ad.InstreamAdView>
   ```

3. Use the InstreamAdLoader to load the InstreamAd object using the page ID (`Page ID`) from the Partner interface.
4. The InstreamAd object contains a set of different types of ad breaks. To get In-Roll ad breaks, use InrollQueueProvider. The InrollQueueProvider queue lets you receive In-Roll objects in the order required for display.

   ```
   public void onInstreamAdLoaded(@NonNull final InstreamAd instreamAd) {
       final InrollQueueProvider inrollQueueProvider = new InrollQueueProvider(context, instreamAd);
       mInstreamAdBreakQueue = inrollQueueProvider.getQueue();
   }
   ```

5. To launch the received In-Roll video ad, you need to prepare it. Unprepared In-Roll video ads won't start.

   To prepare an In-Roll video ad, call the interface Inroll.prepare(instreamAdPlayer) and pass an instance of the created InstreamAdPlayer implementation to it. To track the status of the ad break, set InstreamAdBreakEventListener

```
public void prepare(instreamAdPlayer) {
   currentInroll = mInstreamAdBreakQueue.poll();
   currentInroll.setListener(new InrollListener());
   currentInroll.prepare(instreamAdPlayer);
}
```

6. Once the In-Roll video ad is prepared, InstreamAdBreakEventListener.onInstreamAdBreakPrepared() is called. The prepared In-Roll video ad is ready to play.

   Tip:

   Play video ads in the order they're received from the InstreamAdBreakQueue. If the received In-Roll video ads are played in a different order, this may lower your app's monetization.
7. To play the prepared In-Roll video ad, call Inroll.play() and pass InstreamAdView as a parameter.

```
public void onInstreamAdBreakPrepared() {
    if (currentInroll != null) {
        currentInroll.play(instreamAdView);
    }
}
```

8. After the ad break starts playing, the InstreamAdBreakEventListener.onInstreamAdBreakStarted() method is called. After calling this method, pause the main video and hide its controls.

```
public void onInstreamAdBreakPrepared() {
    if (contentVideoPlayer != null) {
        contentVideoPlayer.pauseVideo();
    }
}
```

9. Once the ad break is played, resume playing the main video. A video ad may play successfully or fail. Both situations need to be handled.

```
@Override
public void onInstreamAdBreakCompleted() {
    handleAdBreakCompleted();
}

@Override
public void onInstreamAdBreakError(@NonNull final String reason) {
    handleAdBreakCompleted();
}

private void handleAdBreakCompleted() {
    currentInroll = null;
    if (contentVideoPlayer != null) {
        contentVideoPlayer.resumeVideo();
    }
}
```

10. After the current In-Roll video ad finishes playing, check the play queue for the next In-Roll video ad in the InstreamAdBreakQueue.

```
public void prepareNextAd() {
   currentInroll = mInstreamAdBreakQueue.poll();
   if (currentInroll != null) {
       prepareInroll(currentInroll);
   }
}
```

11. When you stop using an In-Roll video ad, reset its state.

```
public void onDestroy() {
    if (currentInroll != null) {
        currentInroll.invalidate();
        currentInroll.setListener(null);
    }
    super.onDestroy();
}
```

Classes and interfaces for working with InStream ads

Warning:

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

- Package com.yandex.mobile.ads.instream
- Package com.yandex.mobile.ads.instream.exoplayer
- Package com.yandex.mobile.ads.instream.inroll
- Package com.yandex.mobile.ads.instream.pauseroll
- Package com.yandex.mobile.ads.instream.player.ad
- Package com.yandex.mobile.ads.instream.player.ad.error
- Package com.yandex.mobile.ads.instream.player.content

GDPR

Warning:

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

General information

The General Data Protection Regulation (GDPR) came into effect in the spring of 2018. GDPR regulates how information about citizens of the European Economic Area and Switzerland can be collected and processed. Its intention is 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 2.80, the Yandex Mobile Ads SDK allows you to restrict the collection of data for users located in the European Economic Area and Switzerland who have not given their consent.

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 the Mobile Ads SDK.
2. Show a window where the user can accept the user agreement for personal data processing (for more information, see the example).

   Attention:

   This code is a sample, not a step-by-step guide to follow.

```
...
public class GdprDialogFragment extends DialogFragment {
  ...
  // The code demonstrates creating a dialog.
  public Dialog onCreateDialog(Bundle savedInstanceState) {
        final Context context = getContext();

        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setTitle(R.string.gdpr_title)
                .setMessage(R.string.gdpr_message)
                .setPositiveButton(R.string.accept, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        onButtonClicked(context, true);
                    }
                })
                .setNeutralButton(R.string.about, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(final DialogInterface dialog, final int which) {
                        openPrivacyPolicy();
                    }
                })
                .setNegativeButton(R.string.decline, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        onButtonClicked(context, false);
                    }
                });
        return builder.create();
    }
    private void openPrivacyPolicy() {
        final String url = getString(R.string.privacy_policy_url);
        final Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        startActivity(intent);
    }

    private void onButtonClicked(final Context context,
                                 final boolean userConsent) {
        final SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(context);
        preferences.edit()
                .putBoolean(SettingsFragment.USER_CONSENT_KEY, userConsent)
                .putBoolean(SettingsFragment.DIALOG_SHOWN_KEY, true)
                .apply();

        mNoticeDialogListener.onDialogClick();
    }
}
```

3. Use the setUserConsent method to pass the received value to the Mobile Ads SDK. For users from GDPR regions, the data will be processed only if they give their consent to it.

COPPA

Warning:

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

General information

On April 21, 2000, the Children's Online Privacy Protection Act (COPPA) came into effect. The act governs the collection of personal information from children under the age of 13 by individuals or entities within US jurisdiction. Pursuant to the Act, the application operator shall include in their privacy policy the methods for obtaining parental or caretaker's consent, as well as the operator's commitment to protecting the privacy and safety of children on the internet, including marketing restrictions.

Data about the user's age must be transmitted to the SDK every time the application starts.

Starting in version 5.4.0, the Yandex Mobile Ads SDK will enable you to restrict the collection of information from children under the age of 13.

Quick guide

Data about the user's age must be transmitted to the SDK every time the application starts.

1. Follow the instructions for connecting the Mobile Ads SDK.
2. Show a window where the user can accept the user agreement for personal data processing (for more information, see the example).
3. Use the setAgeRestrictedUser method to pass the received value to the Mobile Ads SDK. By default, it is assumed that the user isn't a child under the age of 13.

Accounting contextual data

Warning:

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

Note: To make the Mobile Ads SDK take the app context into account, enable the AppMetrica SDK version 3.14.3 and higher.

Contextual data accounting is available as of August 20, 2020.

To make monetization more effective, the Mobile Ads SDK automatically takes into account the app context: interface texts, their topics, and how the user interacts with content. This results in selecting more relevant ads.

At the same time:

- You can restrict contextual data tracking, for example, in places where users enter confidential information: on payment screens or in personal correspondences.
- You can completely disable contextual data accounting.
- The SDK only takes into account depersonalized data and corresponds to the ISO standard.

Requirements for using the function

1. The function is only available for Android.
2. The minimum supported version of the AppMetrica SDK is 3.14.3 and higher.
3. The minimum supported version of the Mobile Ads SDK is Android 2.160 and higher.

How to enable contextual data accounting

To enable automatic accounting of application context data, initialize the AppMetrica SDK library version 3.14.3 and higher.

How to disable contextual data accounting

You can disable automatic accounting for different entities: Application, Activity, or View:

Application

To disable automatic accounting of contextual data for the entire app, in the `AndroidManifest.xml` file at the application level, add the following code:

```
<meta-data
    android:name="@string/yandex_ads_context"
    android:value="@string/yandex_ads_context_do_not_parse"/>
```

Code example:

```
<application
    android:name="com.yandex.appmetrica.autotests.agent.AgentApplication"
    ...>
    <meta-data
        android:name="@string/yandex_ads_context"
        android:value="@string/yandex_ads_context_do_not_parse"/>
</application>
```

Activity

To disable automatic accounting of contextual data for a specific activity, in the `AndroidManifest.xml` file at the activity level, add the following code:

```
<meta-data
    android:name="@string/yandex_ads_context"
    android:value="@string/yandex_ads_context_do_not_parse"/>
```

Code example:

```
<activity
    android:name=".NoContextActivity"
    ...>
    <meta-data
        android:name="@string/yandex_ads_context"
        android:value="@string/yandex_ads_context_do_not_parse"/>
</activity>
```

View

There are two ways to disable automatic accounting of contextual data for a particular view:

In the Android resources of a project

```
<TextView ...>
```

```
    <tag android:id="@id/yandex_ads_context"
        android:value="@string/yandex_ads_context_do_not_parse"/>
</TextView>
```

Programmatically

```
view.setTag(R.id.yandex_ads_context, getString(R.string.yandex_ads_context_do_not_parse))
```

TCF v2.0 Consent

Warning:

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

If you implemented a consent management platform (CMP) that complies with the IAB Transparency and Consent Framework (TCF) v2.0 for obtaining consent from the end user to transfer their personal data (user consent flow), Yandex Mobile Ads SDK supports sending TCF v2.0 values.

Yandex Mobile Ads SDK retrieves TCF v2.0 consent strings from `SharedPreferences` using the following keys:

IABTCF_TCString	String	Full encoded TC string.
IABTCF_gdprApplies	Number	• 1: GDPR applies in the current context. • 0: GDPR doesn't apply in the current context. • Unset: Unknown (default before initialization).
IABTCF_CmpSdkID	Number	Unsigned integer ID of the CMP SDK used.
IABTCF_PurposeConsents	Binary string	“0” or “1” at position n, where n's indexing begins at 0. Indicates the consent status for purpose ID n+1. “0” corresponds to `false`, and “1” corresponds to `true`. For example, “1” at index 0 is consent true for purpose ID 1.
IABTCF_VendorConsents	Binary string	“0” or “1” at position n, where n's indexing begins at 0. Indicates the consent status for vendor ID n+1. “0” corresponds to `false`, and “1” corresponds to `true`. For example, “1” at index 0 is consent true for vendor ID 1.

For a detailed description, see the IAB documentation.

Passing values to `SharedPreferences`

Make sure to set required key values before making ad requests. There are two ways to do this:

Manually

Add the values for the keys from the table above to `SharedPreferences`. Example:

Kotlin

```
val sharedPref = activity?.getPreferences(Context.MODE_PRIVATE) ?: return
with (sharedPref.edit()) {
    putInt("IABTCF_gdprApplies", 1)
    apply()
}
```

Java

```
SharedPreferences sharedPref = getActivity().getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt("IABTCF_gdprApplies", newHighScore);
editor.apply();
```

Note:

For more information, see the documentation. The parameters depend on the country, user settings, and device.

Via a CMP SDK

TUse pre-configured CMP SDKs. These SDKs automatically pass the necessary values to `SharedPreferences` so that Yandex Mobile SDK can use them.

Ad targeting

Warning:

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

These classes can be used to configure the SDK and ad targeting settings.

- MobileAds
- AdRequest
- AdRequest.Builder
- Gender

Guide for migrating to version 5

Warning:

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

Additions

Package com.yandex.mobile.ads.banner

BannerAdEventListener interface

- Added the `void onImpression(ImpressionData impressionData)` method that is called when an impression is counted.
- Added the `void onAdClicked()` method that is called once the user clicks on the banner.

Package com.yandex.mobile.ads.instream.exoplayer

Added a new package for integrating InStream ads using ExoPlayer.

Package com.yandex.mobile.ads.instream.player.ad

InstreamAdPlayerListener interface

- Added a new method named `void onAdBufferingFinished(VideoAd videoAd)`. Called when the InstreamAdPlayer finishes buffering a video ad.
- Added a new method named `void onAdBufferingStarted(VideoAd videoAd)`. Called when the InstreamAdPlayer starts buffering a video ad.

Package com.yandex.mobile.ads.instream.player.ar.error

Added a new package to handle InstreamAdPlayer errors.

Package com.yandex.mobile.ads.interstitial

InterstitialAdEventListener interface

- Added the `void onImpression(ImpressionData impressionData)` method that is called when an impression is counted.
- Added the `void onAdClicked()` method that is called once the user clicks on an ad.

Package com.yandex.mobile.ads.nativeads

NativeAdEventListener interface

- Added the `void onImpression(ImpressionData impressionData)` method that is called when an impression is counted.
- Added the `void onAdClicked()` method that is called once the user clicks on an ad.

Package com.yandex.mobile.ads.rewarded

RewardedAdEventListener interface

- Added the `void onImpression(ImpressionData impressionData)` method that is called when an impression is counted.
- Added the `void onAdClicked()` method that is called once the user clicks on an ad.

Changes

Package com.yandex.mobile.ads.banner

BannerAdView class

- The `void setBlockId` method was renamed to `void setAdUnitId`.

Package com.yandex.mobile.ads.interstitial

InterstitialAd class

- The `void setBlockId` method was renamed to `void setAdUnitId`.

Package com.yandex.mobile.ads.nativeads

NativeAdRequestConfiguration.Builder class

- The `public Builder(@NonNull java.lang.String blockId)` method was renamed to `public Builder(@NonNull java.lang.String adUnitId)`.

Package com.yandex.mobile.ads.rewarded

RewardedAd class

- The `void setBlockId` method was renamed to `void setAdUnitId`.

Removals

Package com.yandex.mobile.ads.banner

AdSize class

Removed the following methods:

- `static AdSize flexibleSize()`
- `static AdSize flexibleSize(int width)`

Package com.yandex.mobile.ads.instream

- Removed the `InstreamAdSkipInfo` interface.

Package com.yandex.mobile.ads.instream.model

The package was removed.

Package com.yandex.mobile.ads.nativeads

- Removed the `SliderAdView` class.

Recommendations

Version 4.X.Y	Version 5

`final AdSize flexibleAdSize = AdSize.flexibleSize(width);`	Removed, alternative:
	`final AdSize flexibleAdSize =`
	` AdSize.flexibleSize(width, height);`

`final AdSize flexibleAdSize = AdSize.flexibleSize();`	Removed, alternative:
	`final AdSize flexibleAdSize =`
	` AdSize.flexibleSize(width, height);`

`mBannerAdView.setBlockId(<BlockId>);`	The `BlockId` parameter was renamed `AdUnitId`. `mBannerAdView.setAdUnitId(<AdUnitId>);`
`mInterstitialAd.setBlockId(<BlockId>);`	The `BlockId` parameter was renamed `AdUnitId`. `mInterstitialAd.setAdUnitId(<AdUnitId>);`
`mRewardedAd.setBlockId(<AdUnitID>);`	The `BlockId` parameter was renamed `AdUnitId`. `mRewardedAd.setAdUnitId(<AdUnitId>);`

`final NativeAdRequestConfiguration`
` nativeAdRequestConfiguration =`
`    new`
` NativeAdRequestConfiguration.Builder(BlockId).build();`
	The `BlockId` parameter was renamed to `AdUnitId`:
	`final NativeAdRequestConfiguration`
	` nativeAdRequestConfiguration =`
	`    new`
	` NativeAdRequestConfiguration.Builder(AdUnitId).build();`

`bindSliderAd(@NonNull final SliderAdView sliderAdView);`	Changed the View setting method:
	`bindSliderAd(@NonNull final SliderAdViewBinder`
	` viewBinder);`

`void onError(@NonNull final VideoAd videoAd);`	`void onError(@NonNull final VideoAd videoAd, @NonNull final InstreamAdPlayerError error);`

Integration debugging mode

Warning:

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

Use the integration debugging mode to find errors when enabling the ad SDK or to check integration success.

In debug mode, you can use:

1. Native ad integration indicator
2. Ad SDK latest version integration check

Native ad integration indicator

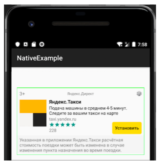
The indicator shows you if native ad integration was successful or provides debugging information so you can see the reason for the error.

To enable the indicator in debugging mode, call the enableDebugErrorIndicator method with the value `true`:

```
MobileAds.enableDebugErrorIndicator(true)
```

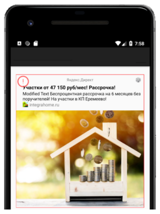
Successful integration

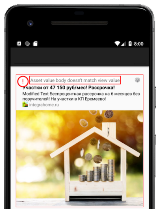
If the integration is successful, a light green border appears on top of the ad in debugging mode.



Failed integration

If an error was made when integrating native ads, the indicator appears on top of the ad in debugging mode. Click on the indicator to see a message with debugging information to help you understand the reason for the error. Click the indicator again to hide the message.





To disable the indicator in debugging mode, call the enableDebugErrorIndicator method with the value `false`:

```
MobileAds.enableDebugErrorIndicator(false)
```

Ad SDK up-to-date version indicator

We added an SDK latest version integration check to Yandex Mobile Ads SDK. Ad SDK latest version integration check includes:

- Outdated library version indicator.
- Lint check for the latest SDK version.

Outdated library version indicator

The indicator of an outdated ad SDK version is shown as a toast icon during the library initialization or ad loading.

More details about the problem are output to the app's logs.

Example of logs:

```
****************************************************************************************
* The integrated version of the Yandex Mobile Ads SDK is outdated.                     *
* Please update com.yandex.android:mobileads to the latest version.                    *
* Learn more about the latest version of the SDK here:                                 *
* https://yandex.ru/dev/mobile-ads/doc/android/quick-start/android-ads-component.html *
* Changelog: https://yandex.ru/dev/mobile-ads/doc/intro/changelog-android.html         *
****************************************************************************************
```

To disable the indicator in debugging mode, call the enableDebugErrorIndicator method with the value `false`:

```
MobileAds.enableDebugErrorIndicator(false)
```

Tip:

We don't recommend that you disable validation for the outdated SDK version indicator. Make sure to use the latest library version to maximize your ad revenue.

Lint check for the latest SDK version

Lint check for the latest SDK version is run at the app's release version build time.

Lint check is performed by calling the `lintVitalRelease` gradle task. The task crashes if the ad SDK version is outdated.

This check prevents building an app if a newer SDK version has already been released.

To disable the latest SDK version check, use the standard gradle code to disable lint checks (for more information, see the Android documentation).

```
android {
    lintOptions {
        disable 'MobileAdsSdkOutdatedVersion'
    }
}
```