Быстрый старт

Интеграция

10.07.2024

Яндекс

Быстрый старт. Интеграция. Версия 2.0

Дата подготовки документа: 10.07.2024

Этот документ является составной частью технической документации Яндекса.

Предупреждение об исключительных правах и конфиденциальной информации

Контактная информация

ООО «ЯНДЕКС»

https://www.yandex.ru

Тел.: +7 495 739 7000

Email: pr@yandex-team.ru

Главный офис: 119021, Россия, г. Москва, ул. Льва Толстого, д. 16

Содержание

Подключение Mobile Ads Unity плагина

Внимание:

Это архивная версия документации. Актуальная документация по всем платформам находится здесь.

Mobile Ads Unity — плагин для игровой платформы Unity3d, включающий поддержку Yandex Mobile Ads SDK.

Примечание:

1. Для работы SDK требуется Target API Level версии 31 и выше.
2. Для загрузки любого вида рекламы необходима версия iOS 12.0 и выше.

Интеграция плагина

Примечание: `yandex-ads-unity-plugin` работает только в окружениях Android и iOS. Работа в редакторе Unity невозможна.

Lite-версия

1. Скачайте каталог `yandex-ads-unity-plugin` и добавьте пакет `yandex-mobileads-lite-.unitypackage`. Вместе с ним будет предложено установить Google resolver. Если в ваш проект уже добавлен Google resolver, уберите галочку.

   Как добавить пакет

   Выберите необходимый плагин (**Assets** → **Import Package** → **Custom Package**) и нажмите кнопку **Import**.
2. С помощью Google resolver установите зависимости: включите **auto-resolve** или выберите в меню пункт **Assets** → **External Dependency Manager** → **Android Resolver** → **Resolve**.
3. Чтобы проверить работу Mobile Ads Unity плагина, воспользуйтесь одним из демонстрационных скриптов в каталоге **samples** репозитория `yandex-ads-unity-plugin`. Скопируйте скрипт в каталог с проектом и добавьте как **Component** в основную камеру.

Понижение Target API Level

Чтобы понизить Target API Level до версии 30, добавьте в `mainTemplate.gradle` и `launcherTemplate.gradle` (если launcherTemplate используется в проекте) явное понижение версии:

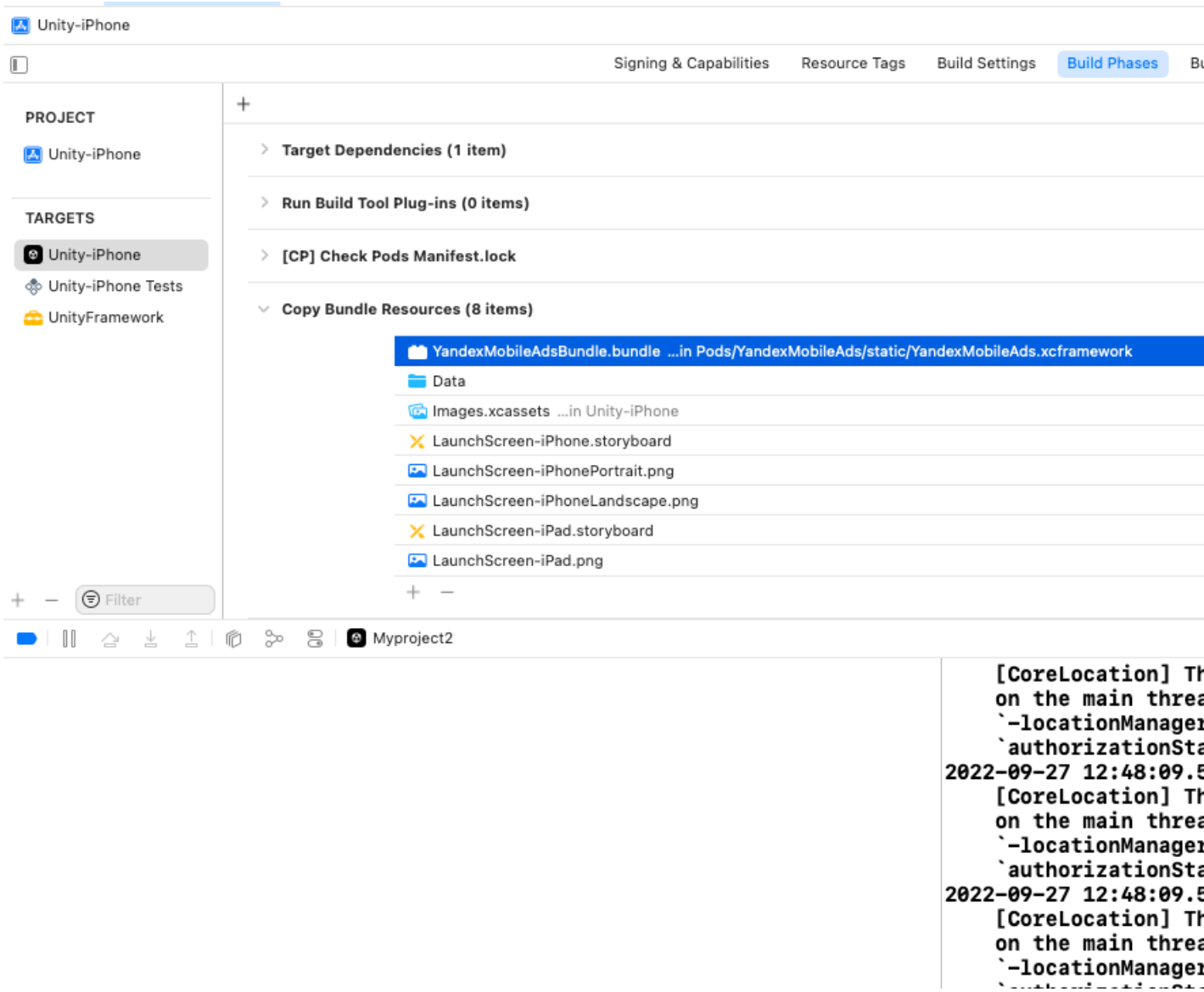
```
configurations.all {
    resolutionStrategy {
        force 'androidx.core:core:1.6.0'
        force 'androidx.core:core-ktx:1.6.0'
    }
}
```

Однако, рекомендуется обновление до Target API Level версии 31, так как у Google есть ограничения на выпуск обновлений для приложений с устаревшей версией Target API Level. Подробнее в статье.

Описание ошибок

Полноэкранная реклама Unity Ads не отображается, ошибка «Incorrect fullscreen view»

Во время запуска полноэкранной рекламы на iOS возможна ошибка «Incorrect fullscreen view». При возникновении данной проблемы проверьте, что в настройках **Build Phases**, секции **Copy Bundle Resources** добавлено значение **YandexMobileAdsBundle.bundle**. Если значение отсутствует, добавьте его.



Форматы рекламы

Баннерная реклама

Баннер — это настраиваемое объявление, которое занимает часть экрана и реагирует на нажатие.

Добавление `Banner` в проект

Чтобы отобразить баннер в вашем приложении, создайте объект `Banner` в скрипте (на C#), который прикреплен к `GameObject`.

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

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

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

Конструктор `Banner` содержит следующие параметры:

- `AdUnitId` — уникальный идентификатор, который выдается в Партнерском интерфейсе и имеет следующий вид: R-M-XXXXXX-Y;
- `AdSize` — размер баннера, который необходимо показать;
- `AdPosition` — позиция на экране.

Загрузка рекламы

После создания и настройки объекта класса `Banner` необходимо загрузить рекламу. Для загрузки рекламы используйте метод `LoadAd`, принимающий в качестве параметра объект `AdRequest`.

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

Особенности загрузки

С помощью объекта `AdRequest` передайте код, полученный в интерфейсе Adfox (подробнее смотрите в помощи по Adfox):

```
...
// Код из интерфейса Adfox для работы с прямыми кампаниями.
private Dictionary<string, string> CreateAdfoxParameters()
{
    Dictionary<string, string> parameters = new Dictionary<string, string>()
    {
        {"adf_ownerid", "<example>"},
        {"adf_p1", "<example>"},
        {"adf_p2", "<example>"},
        {"adf_pt", "<example>"},
        ...
    };

    return parameters;
}

...

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

События баннерной рекламы

Чтобы отслеживать события, происходящие в баннерной рекламе, зарегистрируйте делегата для соответствующего `EventHandler`, как показано ниже:

```
...
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);
}
```

Очистка рекламы

Когда объект рекламы больше не нужен, его можно удалить. Для этого вызовите метод `Destroy`:

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

Полноэкранная реклама

Полноэкранная реклама (Interstitial) — это настраиваемое объявление, отображаемое на весь экран и реагирующее на нажатие.

Добавление `Interstitial` в проект

Чтобы подключить рекламу, создайте объект `Interstitial` в скрипте (на C#), который прикреплен к `GameObject`.

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

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

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

Конструктор `Interstitial` содержит параметр `adUnitId` — уникальный идентификатор, который выдается в Партнерском интерфейсе и имеет следующий вид: R-M-XXXXXX-Y.

Загрузка рекламы

После создания и настройки объекта класса `Interstitial` необходимо загрузить рекламу. Для загрузки рекламы используйте метод `LoadAd`, принимающий в качестве параметра объект `AdRequest`.

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

Особенности загрузки

С помощью объекта `AdRequest` передайте код, полученный в интерфейсе Adfox (подробнее смотрите в помощи по Adfox):

```
...
// Код из интерфейса Adfox для работы с прямыми кампаниями.
private Dictionary<string, string> CreateAdfoxParameters()
{
    Dictionary<string, string> parameters = new Dictionary<string, string>()
    {
        {"adf_ownerid", "<example>"},
        {"adf_p1", "<example>"},
        {"adf_p2", "<example>"},
        {"adf_pt", "<example>"},
        ...
    };

    return parameters;
}

...

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

Показ рекламы

После того, как реклама загружена, ее можно показать:

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

События полноэкранной рекламы

Чтобы отслеживать события, происходящие в полноэкранной рекламе, зарегистрируйте делегата для соответствующего `EventHandler`, как показано ниже:

```
...
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);
}
```

Очистка рекламы

Когда объект рекламы больше не нужен, его можно удалить. Для этого вызовите метод `Destroy`:

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

Реклама с вознаграждением

Реклама с вознаграждением (Rewarded Ad) — настраиваемое объявление, отображаемое на весь экран. За просмотр такой рекламы пользователь получает вознаграждение.

Добавление `Rewarded Ad` в проект

Чтобы подключить рекламу, создайте объект `RewardedAd` в скрипте (на C#), который прикреплен к `GameObject`.

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

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

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

Конструктор `RewardedAd` содержит параметр `adUnitId` — уникальный идентификатор, который выдается в Партнерском интерфейсе и имеет следующий вид: R-M-XXXXXX-Y.

Загрузка рекламы

После создания и настройки объекта класса `RewardedAd` необходимо загрузить рекламу. Для загрузки рекламы используйте метод `LoadAd`, принимающий в качестве параметра объект `AdRequest`.

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

Особенности загрузки

С помощью объекта `AdRequest` передайте код, полученный в интерфейсе Adfox (подробнее смотрите в помощи по Adfox):

```
...
// Код из интерфейса Adfox для работы с прямыми кампаниями.
private Dictionary<string, string> CreateAdfoxParameters()
{
    Dictionary<string, string> parameters = new Dictionary<string, string>()
    {
        {"adf_ownerid", "<example>"},
        {"adf_p1", "<example>"},
        {"adf_p2", "<example>"},
        {"adf_pt", "<example>"},
        ...
    };

    return parameters;
}

...

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

Показ рекламы

После того, как реклама загружена, ее можно показать:

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

События рекламы с вознаграждением

Чтобы отслеживать события, происходящие в рекламе с вознаграждением, зарегистрируйте делегата для соответствующего `EventHandler`, как показано ниже:

```
...
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);
}
```

Очистка рекламы

Когда объект рекламы больше не нужен, его можно удалить. Для этого вызовите метод `Destroy`:

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