

YUMIMOB Unity3D SDK

Integration Guide for Android Developers (V3.2.2)

Contents

<u>1. OVERVIEW</u>	<u>1</u>
<u>1.1 INTRODUCTION</u>	<u>1</u>
1.2 DEVELOPMENT ENVIRONMENT	3
<u>2. DOWNLOADS REQUIRED</u>	<u>3</u>
2.1 SDK DOWNLOAD.....	3
2.2 THIRD-PARTY SDK DOWNLOAD.....	3
<u>3. INTEGRATION.....</u>	<u>4</u>
3.1 ADD RESOURCE FILES	4
3.2 CONFIGURING ACCESS TO ADS.....	6
3.4 LOG AND TOAST.....	9
3.5 TESTING PLATFORM INTEGRATIONS	10
3.6 OBFUSCATION	14
<u>5. COMPILATION.....</u>	<u>15</u>

1. OVERVIEW

1.1 Introduction

This document is designed to guide and assist developers in the integration of the YUMI Unity 3D SDK into their Unity3D Android products. Successful integration and

activation will make the full range of Yumi ad monetization services available to the product worldwide, or in specific geos such as China (if so required). You also can control which ad networks get installed for mediation, and have a variety of other features available to help your product earn more ad revenue quickly.

Special note: the Yumi approach to mediation is unique in that the ad networks chosen each use the Yumi account for, and not that of the developer (if any). This fact benefits developers in that it can shorten and simplify the integration requirements. As Yumi consolidates and markets its own ZPLAY inventory with that of its select working partners, this can also increase revenue performance for the developer. The benefit does not stop there though, as we consolidate more than just the reporting using this approach. All payments from these ad networks come together in one place, and get paid at one time, often paying you even faster than we get paid. Great revenue possibilities, simplified 😊

As the above approach implies, Yumi is a full service ad monetization solution. You will be assigned an operations rep and a technical support rep to help you get started. When the time comes to get started on integration, simply email onboarding@yumimobi.com and we will let you know who will be working with you to assist in getting your integration complete and functional. We can even help in making suggestions about what ads to use where, and how often. We are truly your partner for success!

1.2 Development Environment

OS: Windows, Mac, Linux

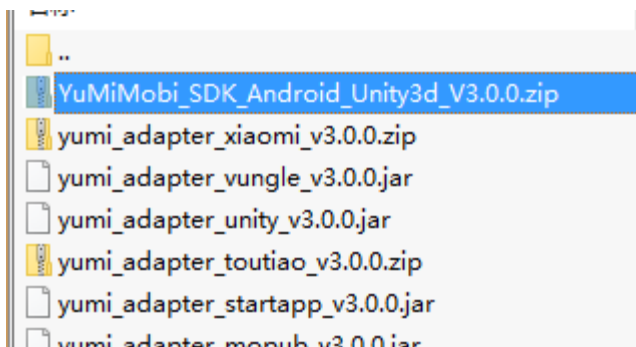
Android SDK: 2.1 and newer versions

IDE: Unity 5

2. DOWNLOADS REQUIRED

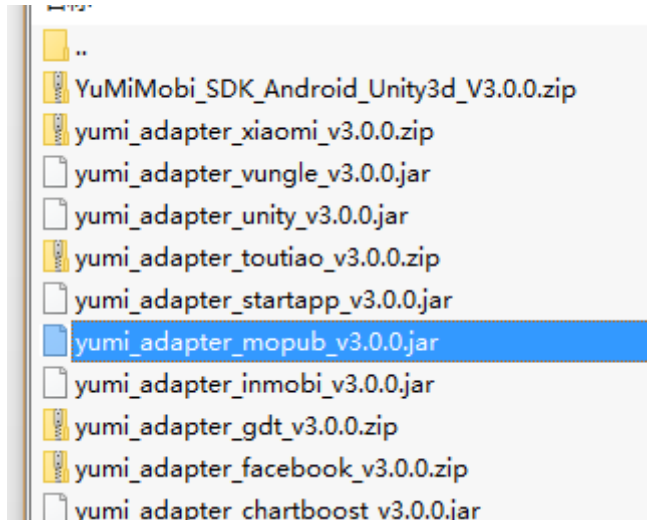
2.1 SDK Download

Our main SDK and all 3rd party SDKs are available through registering and logging in to our website <https://www.yumimobi.com>. Download the SDK package here or contact us at support@yumimobi.com and we can assist you in setting up your account to get started.



2.2 Third-Party SDK Download

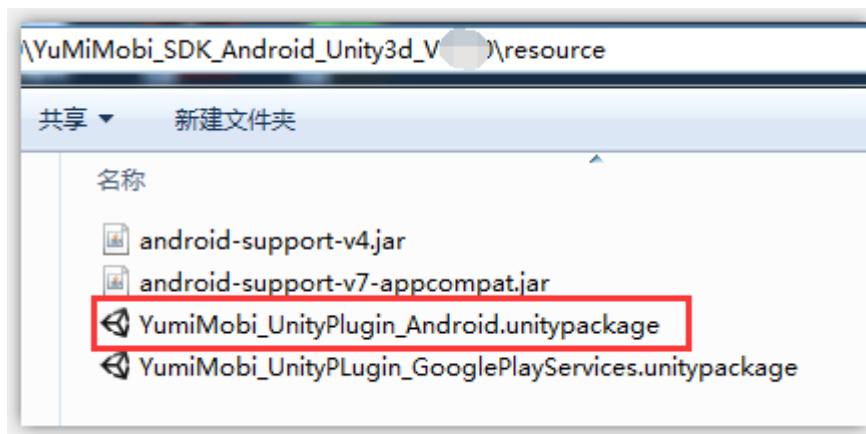
Your Yumi operation rep may suggest the most appropriate ad networks to integrate for your app or game, but the final decision is yours. Choose the third-party platforms which you would like to integrate into the Yumi SDK jar package (the various package names clearly identify each ad network available).



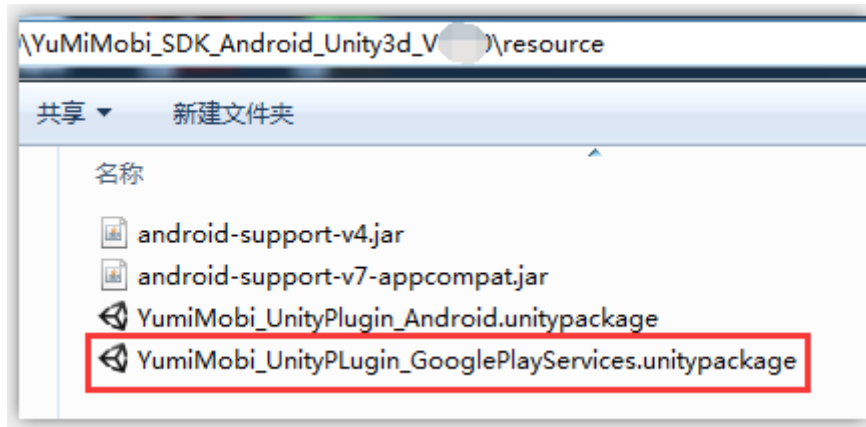
3. Integration

3.1 Add resource files

3.1.1 Everyone needs this first party Yumi resource file. Import `resource\YumiMobi_UnityPlugin_Android.unitypackage` to the project. For a manual import option, please see Exhibit A.



3.1.2 When the application needs to be published to the googlePlay platform , `Assets / plugins / Android` does not have information about googlePlayServices related to Jar or aar files , import `YumiMobi_UnityPlugin_GooglePlayServices.unitypackage`



3.1.3 When Assest / plugins / Android does not have support-v7 or support-v4 related jar or aar files , copy the following files to the ../Assest/plugins/Android folder

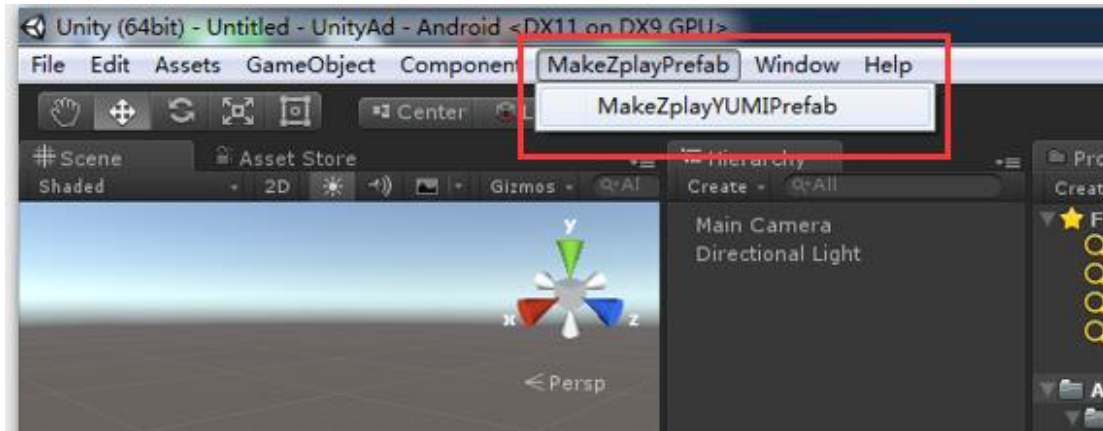


3.1.4 Add any third-party SDK adapter resources for the Ad Networks previously chosen: The third-party adapters (yumi_adapter_*****_v*.*.*.jar) within adapter folder should be added under \Assets\plugins\Android.

3.2 Configuring Access to Ads

3.2.1 Environment Configuration

1) Click MakeZplayPrefab / MakeZplayYUMIPrefab to generate ZplayYUMIHelper prefab to Hierarchy, This prefab will follow all the scenes

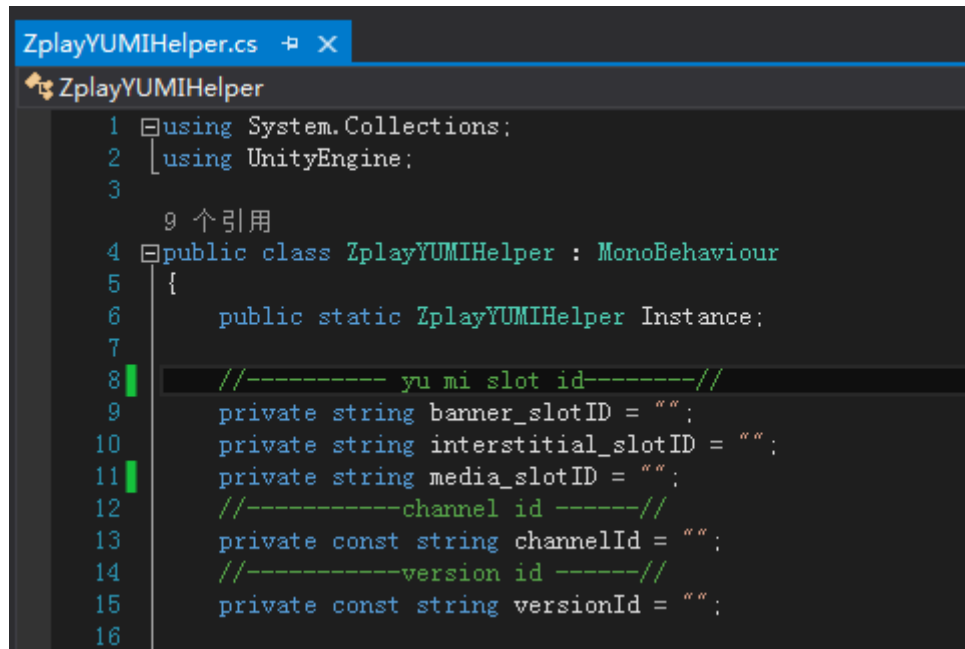


2) Once the above steps are complete, set your slot ID, channel number, version number (channel number and version number as non-mandatory) within the ZplayYUMIHelper class.

Note: ChannelID refers to the channel labeling of the application, and the YUMI platform can carry out data statistics and effect analysis according to the ChannelID. A Popstar! For example, when the game is released to the SamSung channel, setChannelID(channelStr) needs to be set to setChannelID(' SamSung ').

The channelID is labeled as the YUMI platform to generate the information and cannot be modified at will :

Channel name	ChannelID
SamSung	SamSung



```

ZplayYUMIHelper.cs
ZplayYUMIHelper
1 using System.Collections;
2 using UnityEngine;
3
4 public class ZplayYUMIHelper : MonoBehaviour
5 {
6     public static ZplayYUMIHelper Instance;
7
8     //----- yu mi slot id -----//
9     private string banner_slotID = "";
10    private string interstitial_slotID = "";
11    private string media_slotID = "";
12    //-----channel id -----//
13    private const string channelId = "";
14    //-----version id -----//
15    private const string versionId = "";
16

```

3) ZplayYUMIHelper class Start () method The body has the following code

```

void Start () {
    //Remind users to get permission
    yuMiUnityAd.CheckPermission();
    //Init video
    InitMedia();
    //Init Interstitial
    InitInterstitialAd();
    //Whether to open the log information (false:close, true: open)
    Logger.SetDebug(true);
    //After the initial video screen is completed, wait for 0.2 seconds to start requesting the video and the interstitial
    StartCoroutine(Reques(0.2f));
}

```

initMedia(); // Initialize video, note if you don't need video

InitlterstitialAD() ;// Initialize Initlterstitial , note if you don't need Initlterstitial

ZplayLogger.setDebug(false); // log, true to display the log, false does not display logs

3.2.2 Banner, Interstitial & Rewarded Video Ad Access

STANDARD PROCESS

These code calls generate and control ads. See notes below for additional details.

Action	Banner	Interstitial	Rewarded Video
Show	ZplayYUMIHelper.Instance.ShowBanner();	ZplayYUMIHelper.Instance.ShowInterstitialAD();	ZplayYUMIHelper.Instance.ShowMedia();

Hide	ZplayYUMIHelper.Instance.DismissBanner();		
Unhide	ZplayYUMIHelper.Instance.ResumeBanner();		

Confirm – Call to confirm whether rewarded video has completed loading or not.

ZplayYUMIHelper.Instance.IsMediaPrepared ();

// Load completed function call logic In the ZplayYUMIHelperupdate update , as follows :

```
void Update () {
    //Ask about video every 5 seconds is prepared
    if (!rotalsMediaPrepared)
    {
        timeNum += Time.fixedDeltaTime;
        if (timeNum > 5f)
        {
            timeNum = 0;
            IsMediaPrepared();
        }
    }
}
```

Call IsMediaPrepared() to determine whether video has been loaded. It is recommended to request every five seconds. If loaded successfully, return true; if not, return false. After loading completed, please set GetRotalsMediaPrepared to true and suspend calling IsMediaPrepared().


```

/// <summary>
/// Video Is Prepared
/// </summary>
1 ↑引用
public void IsMediaPrepared()
{
    bool isPrepared = yuMiUnityAD.IsMediaPrepared();
    if (isPrepared)
    {
        //The video load completes the shutdown of the rotation request video logic
        GetRotaIsMediaPrepared = true;
        Logger.LogError("yumiMobi SDK Media IsMediaPrepared : true");
    }
    else
    {
        Logger.LogError("yumiMobi SDK Media IsMediaPrepared : false");
        //The video screen does not have 5 seconds to request a screen for loading
    }
}

```

Turn on polling in the video ad closure callback to load the video again

```

public void onMediaClosed(string data)
{
    Logger.LogError("yumiMobi SDK Media Closed : " + data);
    //Once the video screen is successful, it will be retrained to see if the video screen is loaded
    ZplayYUMIHelper.Instance.GetRotaIsMediaPrepared = false;
}

```

3.3 Google Play Release

The following code will set the App as being released on Google Play. Setting this state helps determine what global servers we will use, and also how the App may get handled in China where Google has a very small market share.

```
ZplayYUMIHelper.Instance.SetAppIsGooglePlayVersions();
```

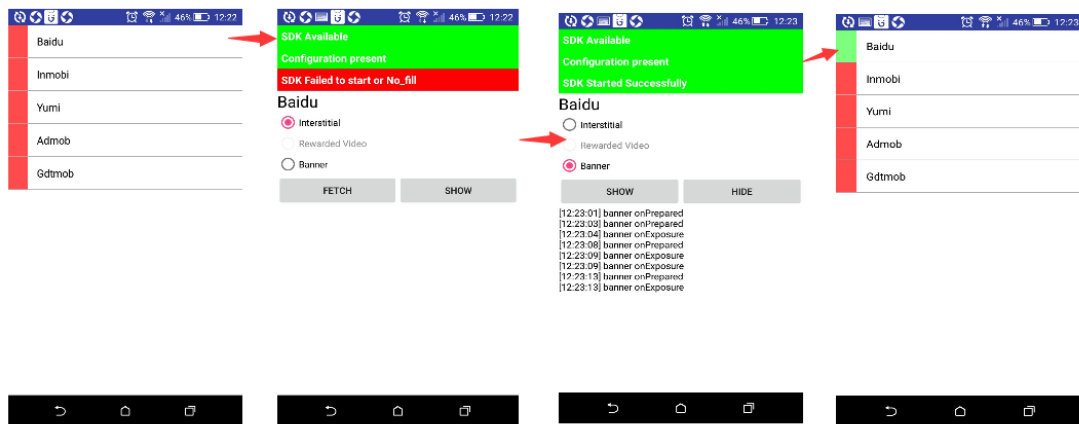
3.4 Log and Toast

The Android Log output method provided for the Unity plugin is as follows. Developers are strong encouraged to use it for efficient debugging.

```
ZplayLogger.Log("*****");
```

3.5 Testing Platform Integrations

Here, developers can test a) initialization of the various adapters, b) the configuration of the Yumi and other ad network account setups, and c) the availability and response to ad calls. Testing is accomplished in the Yumimobi SDK debug mode. The following image shows the process of drilling down from the network level to the ad level of testing. The debugging process is outlined in more detail afterwards.



Debugging Steps:

1) To initialize the debugging feature, call:

```
ZplayYUMIHelper.Instance.StartDebugging();
```

2) Debugging Home Screen – After basic account configuration on our website, and download of the SDK, all of the platforms should be listed on this first screen, initially in red. The red markers will turn green once a platform is properly integrated and (at least one ad type) is shown successfully within the debugging program. If none or only some are shown, please check with your operations rep for assistance.

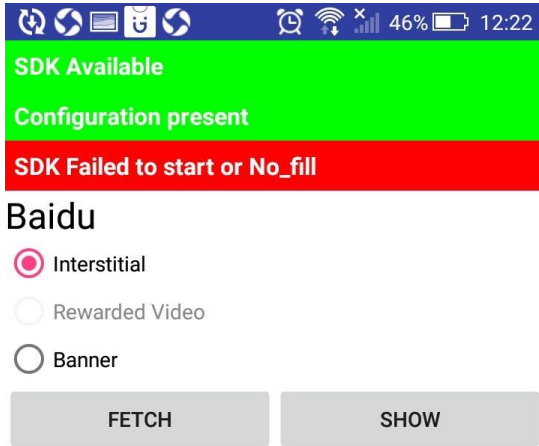


Each platform line can be clicked to enter and investigate the various layers of testing available. The following items can be tested within the second screen for each platform shown on the debugging home screen.

a) Adapter SDK Present? When this first line is green, it means that the platform adapter has been added; when in red, it means that the adapter has not yet been added. If red, please review section 3.1-2 and check for or add any missing adapter.

b) Adapter Configuration Correct? When green, it means the platform adapter has been properly registered in the Manifest; when in red, it means that the adapter has not been added, is incomplete, or contains errors. Please see section 3.1.3, Exhibit B (if needed) along with any custom integration content generated on our website when the account was first configured. The platform adapter component content is essential.

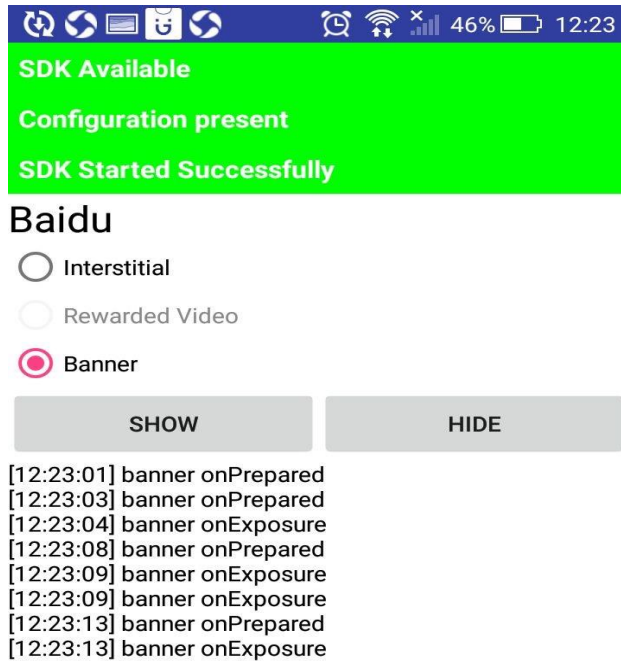
c) Ad Calls Initialized? You need to fetch an ad to test this line. When green, it means the ad platform was reached and an ad has been shown successfully; when in red, it means that the ad call was not successfully initialized or that there was no demand available from the platform for that request. Please email us if assistance required [here](#).



d) Demand Available? Click Fetch to request ad. A fetch will generate an ad download. You will get a text response confirming the ad download, or instead an error.

e) Ad Displayed? Click Show to display an ad. When ad shows properly, all test elements will turn green, which indicates this platform has been integrated successfully, at least for that ad type.

Below is a sample screen of some banner ads that were fetched from Baidu ad network and displayed. The HIDE button simple allows the developer to test the SDK's hide feature.



f) The debugging mode must be completed before releasing the App.

3.6 Obfuscation

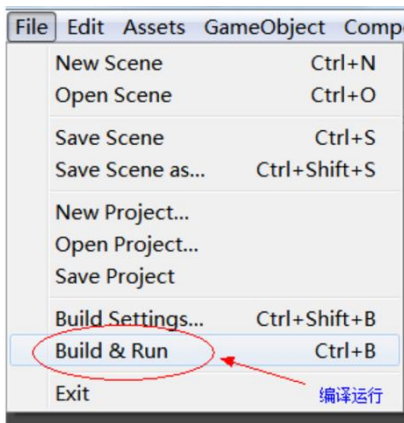
If an obfuscated compilation is required for your project, please add the following items..

```
-keepattributes Exceptions , InnerClasses , Signature , Deprecated , SourceFile ,  
LineNumberTable , *Annotation* , Synthetic , EnclosingMethod  
-keep class com.yumi.android.sdk.ads.** { *;}  
-keep class com.yumi.android.sdk.ads.self.**{*;}  
-keep class com.yumi.android.sdk.ads.selfmedia.**{*;}
```

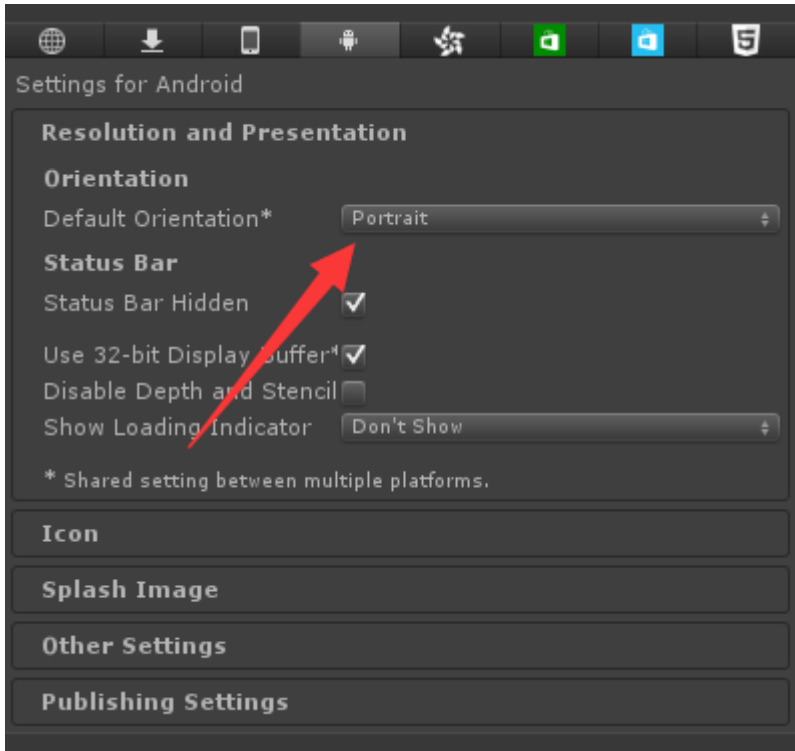
5. Compilation

To make a build of your game complete with the integrated Yumi SDK, please do the following steps:

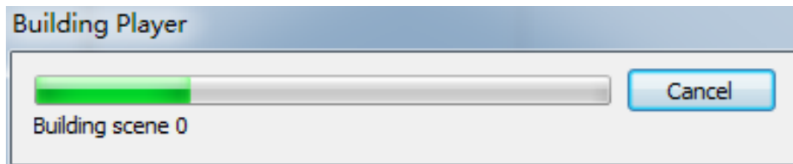
1) Click File, choose Build & Run to compile, and complete the build and run screen that follows as needed.



2) Screen Orientation. While many games force a specific orientation, some allow for either portrait or landscape. Yumi, however, requires you to pick a default. The screen orientation in Unity3D shall also need to match the settings of AndroidManifest.xml file, as a conflict may lead to crash after starting application. The following image shows where this is set in the build process.



3) Click **【Build And Run】** on the main screen and wait.



Conclusion

This concludes our integration document. We hope your project has gone smoothly, and we wish you well in monetizing your game with ads. Please don't hesitate to reach out to your operations or technical support contact if you need any assistance in solving integration problems. We also value and welcome your feedback on this document and for the integration process in general. Thank you!

Exhibit A

Using your own AndroidManifest.xml file

To use your own file, please do the following:

(1) Add Permission(s)

This first permission is necessary for our content to function properly.

```
<!-- yumi sdk start -->
<!-- Mandatory-->
<uses-permission android : name="android.permission.INTERNET"/>
<uses-permission android : name="android.permission.READ_PHONE_STATE"/>
<uses-permission android : name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android : name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android : name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android : name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android : name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- yumi sdk end -->
```

Adding this 2nd permission (optional) will generally improve the ad fill rate.

```
<!-- yumi sdk start -->
<uses-permission android : name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android : name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android : name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.WRITE_CALENDAR"/>
<!-- yumi sdk end -->
```

Caution: If app targetSdkVersion is 23 or above, you will need to call the following permission check code. If the user has not previously provided permission, we will need to prompt user authorization via a popup. This process needs to be called before instantiating an ad. The android-support-v4.jar also will need to be added prior.

```
YumiUnityAdUtils.CheckPermission();
```

(2) Add the following Component to **Assets\plugins\Android\AndroidManifest.xml**.

```
<receiver android:name="com.yumi.android.sdk.ads.self.module.receiver.ADReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.DOWNLOAD_COMPLETE" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.PACKAGE_ADDED" />
        <data android:scheme="package" />
    </intent-filter>
</receiver>
<service android:name="com.yumi.android.sdk.ads.service.YumiAdsEventService" />
<activity android:name="com.yumi.android.sdk.ads.self.activity.YumiFullScreenActivity"
    android:configChanges="keyboardHidden|orientation|screenSize"
    android:theme="@android:style/Theme.NoTitleBar.Fullscreen" />
<!--Debugging Activity -->
<activity android:name="com.yumi.android.sdk.ads.mediation.activity.MediationTestActivity" ></activity>
```

Note: The third-party platform adapter component registration details will be generated on our website automatically when you choose which Ad Networks you intend to work with.

Exhibit B

Options for Banner/Interstitial/Rewarded Video Access

1. Banner Options

a) Banner callback: The following callback messages are available to developers that would like to track or take certain game control actions (pause, unpause, etc.) conditional on these various ad status conditions. To use this option, inherit Assets.YumiUnityAdUtils.BannerAdCallbackListener **[check]** interface and use any of the following callback codes.

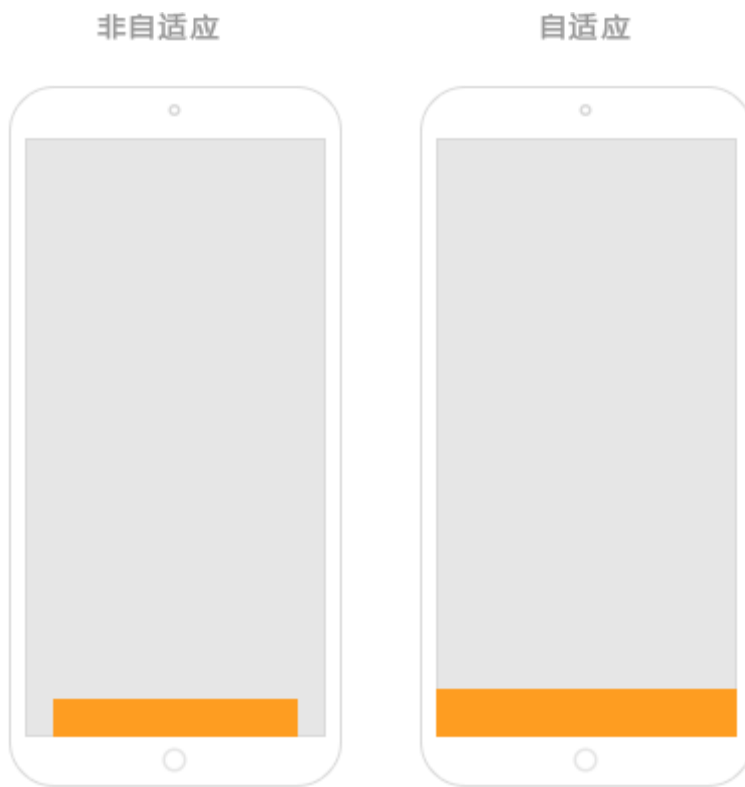
<code>onBannerPreparedFailed(string data)</code>	callback when banner loading failed, get failure reason via data.
<code>onBannerPrepared(string data)</code>	callback when banner loading success.
<code>onBannerExposure(string data)</code>	callback when banner show success.
<code>onBannerClosed(string data)</code>	callback when banner closed.
<code>onBannerClicked(string data)</code>	callback when banner clicked.

Sample:

```
#region Banner Callback
//callback method when banner loading failed
public void onBannerPreparedFailed(string data)
{
    ZplayLogger.Log("onBannerPreparedFailed:" + data);
}
//callback method when banner loading success
public void onBannerPrepared(string data)
{
    ZplayLogger.Log("onBannerPrepared:" + data);
}
//callback when banner show success
public void onBannerExposure(string data)
{
    ZplayLogger.Log("onBannerExposure:" + data);
}
//callback method of banner close event
public void onBannerClosed(string data)
{
    ZplayLogger.Log("onBannerClosed:" + data);
}
//callback of banner click event
public void onBannerClicked(string data)
{
    ZplayLogger.Log("onBannerClicked:" + data);
}
#endregion
```

b) Banner width controls: The parameter status of `isMatchWindowWidth` can be used to autoexpand the banner width to the screen width, or to leave it in its default size. We recommend using the default “false” status (do not expand) to improve ad display performance. See below for the code to utilize, and also to see examples of the two banner display possibilities.

```
YumiUnityAdUtils.AddBannerAd(gameObject.name, isMatchWindowWidth);
```



2. Interstitial Options

Callbacks: The following callback messages are available to developers that would like to track or take certain game control actions (pause, unpause, etc.) conditional on these various ad status conditions. To make use of this option, inherit Assets.YumiUnityAdUtils.InterstitialAdCallbackListener interface and use any of the following methods.

<code>onInterstitialPreparedFailed(string data)</code>	callback when interstitial loading failed, get failure reason via data
<code>onInterstitialPrepared(string data)</code>	callback when interstitial loading success
<code>onInterstitialExposure(string data)</code>	callback when interstitial show success
<code>onInterstitialClosed(string data)</code>	callback when interstitial closed
<code>onInterstitialClicked(string data)</code>	callback when interstitial clicked

Sample:

```
#region InterstitialAd Callback
//callback method when interstitial loading failed
public void onInterstitialPreparedFailed(string data)
{
    ZplayLogger.Log("onInterstitialPreparedFailed:" + data);
}
//callback method when interstitial loading success
public void onInterstitialPrepared(string data)
{
    ZplayLogger.Log("onInterstitialPrepared:" + data);
}
//method when interstitial show success
public void onInterstitialExposure(string data)
{
    ZplayLogger.Log("onInterstitialExposure:" + data);
}
//callback of interstitial close event
public void onInterstitialClosed(string data)
{
    ZplayLogger.Log("onInterstitialClosed:" + data);
}
//callback of interstitial click event
```

3. Rewarded Video Options

Callbacks: The following callback messages are available to developers that would like to

track or take certain game control actions (pause, unpause, etc.) conditional on these various ad status conditions. To make use of this option, inherit Assets.YumiUnityAdUtils. [MediaAdCallbackListener](#) interface and use any of the following methods.

<code>onMediaExposure(string data)</code>	when rewarded video starts successfully.
<code>onMediaClosed(string data)</code>	callback when rewarded video closed.
<code>onMediaClicked(string data)</code>	callback when rewarded video clicked. Note: this method does not guarantee 100% callback depending on the platform
<code>onMediaIncentived(string data)</code>	callback when rewarded video play completed and available to get bonus. Note: If video play not completed, this callback will not be initiated. Note: this callback will always be sequenced after an <code>onMediaClosed ()</code> callback.

Sample:

```
#region MediaAd Callback
// bonus callback method of rewarded video
public void onMediaIncentived(string data)
{
    ZplayLogger.Log("onMediaIncentived:" + data);
}
// callback method of rewarded video show completed
public void onMediaExposure(string data)
{
    ZplayLogger.Log("onMediaExposure:" + data);
}
// callback method of rewarded video close event
public void onMediaClosed(string data)
{
    ZplayLogger.Log("onMediaClosed:" + data);
}
// callback method of rewarded video click event
public void onMediaClicked (string data)
{
    ZplayLogger.Log("onMediaClicked:" + data);
}
#endregion
```