Contents

[Audience 3](#_Toc341272138)

[Authors 3](#_Toc341272139)

[Supported Platforms 3](#_Toc341272140)

[Introduction 3](#_Toc341272141)

[Import Package 3](#_Toc341272142)

[Imported Files 4](#_Toc341272143)

[References 4](#_Toc341272144)

[Example Scenes 5](#_Toc341272145)

[Scene ShowProducts 5](#_Toc341272146)

[Scene ShowController 5](#_Toc341272147)

[Android Setup 6](#_Toc341272148)

[Build Settings 8](#_Toc341272149)

[OUYA Panel 9](#_Toc341272150)

[OUYA Tab 10](#_Toc341272151)

[Unity Tab 11](#_Toc341272152)

[Java JDK Tab 11](#_Toc341272153)

[Android Tab 12](#_Toc341272154)

[NDK Tab 12](#_Toc341272155)

[Example Scenes 13](#_Toc341272156)

[Scene Controller Example 13](#_Toc341272157)

[Script 14](#_Toc341272158)

[Scene Products Example 15](#_Toc341272159)

[Script 16](#_Toc341272160)

[Scene NDK Example 17](#_Toc341272161)

[Script 18](#_Toc341272162)

[Asset Store Resources 19](#_Toc341272163)

[New to Unity 19](#_Toc341272164)

[NDK 19](#_Toc341272165)

[Single Development IDE 19](#_Toc341272166)

[Verbal Commands 19](#_Toc341272167)

[Deployment 19](#_Toc341272168)

# Audience

The OUYA SDK Unity Package is targeted towards Unity developers intending to publish to the OUYA platform and allows testing on Android devices.

# Authors

Tim Graupmann (tim@tagenigma.com)

# Supported Platforms

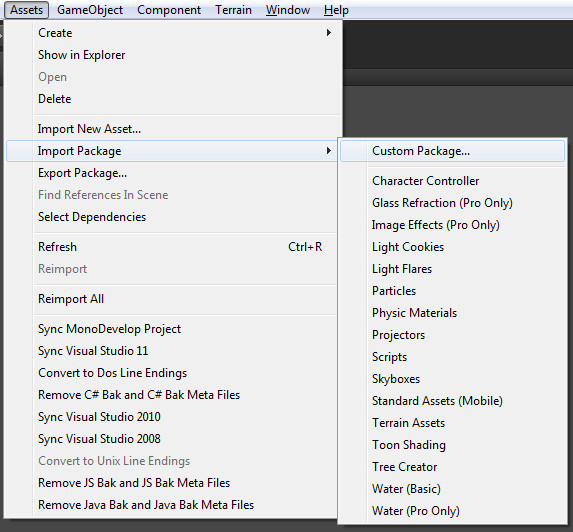
The OUYA SDK Unity Package supports publishing from Mac and Windows.

# Introduction

Welcome to the OUYA Unity developers club. This document will provide an overview for setting up the example Unity application to publish on the OUYA platform.

# Import Package

Part of receiving this document, you received OuyaSDK.unitypackage which can be imported into the Unity IDE on Mac and Windows. Start with a New Project, and Choose Assets->Import Package->Custom Package from the menu bar. Browse to the OuyaSDK.unitypackage and import all files.



# Imported Files

You will find the following structure imported into your Assets folder.

./LitJson – 3rd party JSON parsing library

./Ouya/Docs/Readme.doc – You are reading this doc

./Ouya/Examples/Scenes/SceneShowProducts.unity – Example Products Scene

./Ouya/Examples/Scenes/SceneShowController.unity – Example Controller Scene

*./Ouya/Examples/Scenes/SceneShowNDK.unity – Example NDK Scene*

./Ouya/Examples/Scripts/OuyaInputManager.cs – Example script interacting with OUYA input

./Ouya/Examples/Scripts/OuyaShowController.cs – Script for showing controller states

./Ouya/Examples/Scripts/OuyaShowProducts.cs – Script for showing products

*./Ouya/Examples/Scripts/OuyaShowNDK.cs – Script for showing C++ interaction*

./Ouya/SDK/Editor/OuyaPanel.cs – Custom editor extension for OUYA

./Ouya/SDK/Prefabs/OuyaGameObject.prefab – OUYA SDK setup prefab

./Ouya/SDK/Scripts/OuyaGameObject.cs – OUYA SDK java interface

./Ouya/SDK/Scripts/OuyaSDK.cs – OUYA SDK Unity API

./Plugins/Android – Special folder for customizing Android platform publishing

./Plugins/Android/AndroidManifest.xml – Custom manifest, overrides the default

./Plugins/Android/libs – Jar libraries

./Plugins/Android/libs/gson-2.2.2.jar – GSON jar for parsing JSON in Java

./Plugins/Android/libs/guava-r09.jar – Google jar for Google API

./Plugins/Android/libs/ouya-sdk.jar – OUYA SDK jar for Java API

./Plugins/Android/OuyaUnityApplication.jar – Example application Jar

./Plugins/Android/OuyaUnityPlugin.jar – OUYA SDK interface Jar

*./Plugins/Android/jni/jni.cpp – C++ JNI interface*

./Plugins/Android/res/drawable/app\_icon.png – Custom application icon

./Plugins/Android/res/layout/main.xml – Custom android layout

./Plugins/Android/res/values/strings.xml – Custom android strings

./Plugins/Android/src/OuyaUnityApplication.java – Example Android Application Activity

./Plugins/Android/src/tv/ouya/demo/OuyaUnityApplication/R.java – Auto generated from custom layout

# References

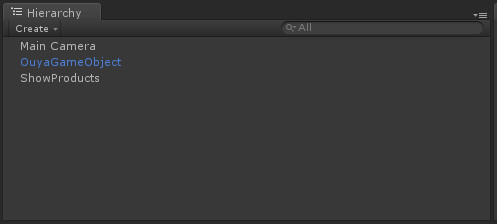
Litjson – third party JSON parsing library for parsing in C#. <http://litjson.sourceforge.net>

Google-GSON - third party JSON parsing library for Java. <http://code.google.com/p/google-gson/>

Google-Guava – third party library for Google core. <http://code.google.com/p/guava-libraries/>

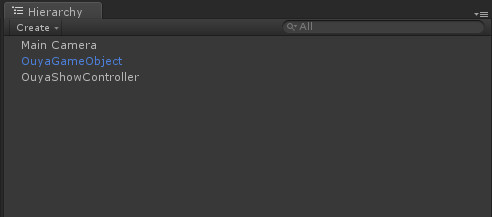
# Example Scenes

## Scene ShowProducts

Open the ./Ouya/Examples/Scenes/SceneShowProducts.unity example scene.

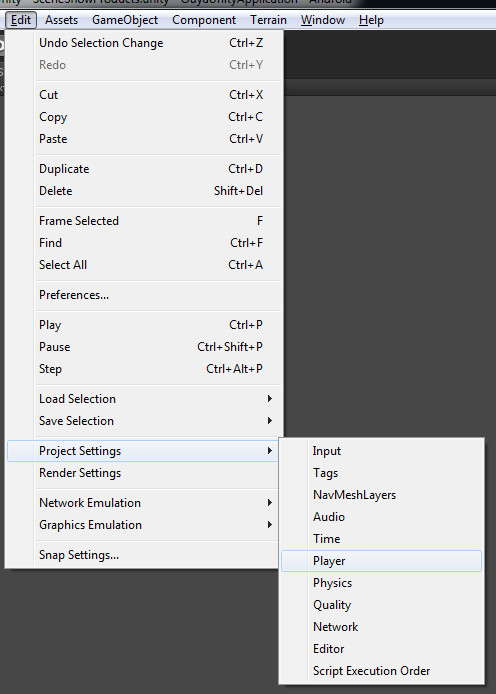
All scenes start with a Main Camera. There’s a custom GameObject added for OuyaGameObject which handles taking messages from the OUYA SDK from Java to C#. The “ShowProduct” GameObject is a simple display script for displaying retrieved products and invoking purchases

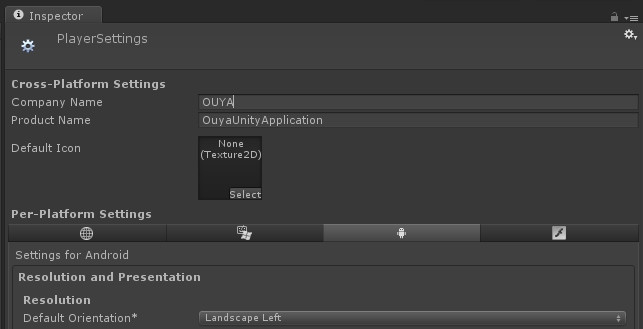
## Scene ShowController

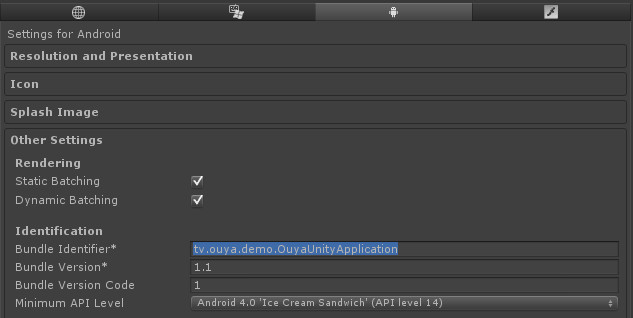
Open the ./Ouya/Examples/Scenes/SceneShowController.unity example scene.

Both scenes needed to add the OuyaGameObject to interface with the OUYA SDK to receive display messages. The OuyaShowController script handles the display code necessary for mapping OUYA SDK input to the Unity GUI.

# Android Setup

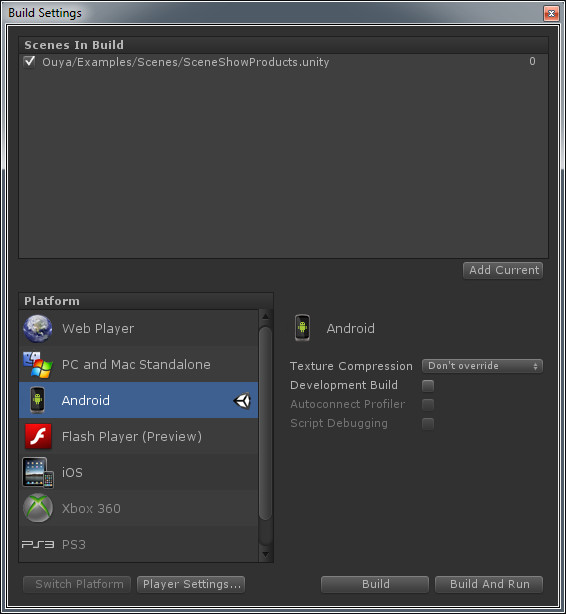
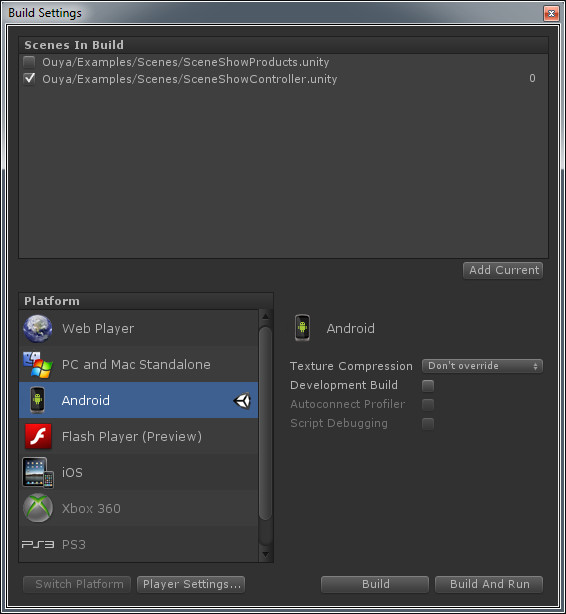
Some player settings must be customized to build on Android. Open the player settings by navigating the menu to the Edit->Project Settings->Player menu item. The PlayerSettings will appear in the inspector.

In the inspector, expand the Resolution and Presentation section. Be sure to set your product and company name. The OUYA will use a default orientation for landscape. Choose landscape left from the drop down. Unity calls drop downs, pop ups (intuitively opposite).

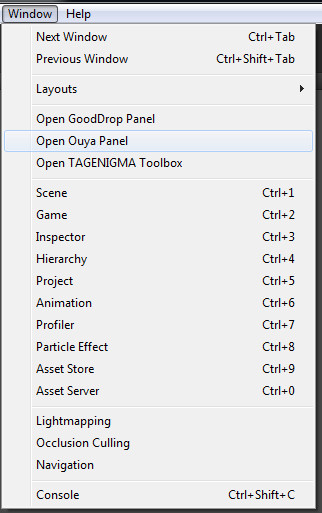
Before any Android application can build you must set the bundle identifier. Be careful that the Bundle Identifier matches what you’ve used in the Android manifest. Also set the Minimum API Level to 12 or better. The OUYA console will be version 15, but you may want to test on another Android device.

# Build Settings

Make sure you change the target platform to Android. Select Android and then click the button “Switch Platform”. Add a default scene by clicking the “Add Current”. Make sure at least one scene is checked. At this point you can build and enter the file path for the Android APK build.

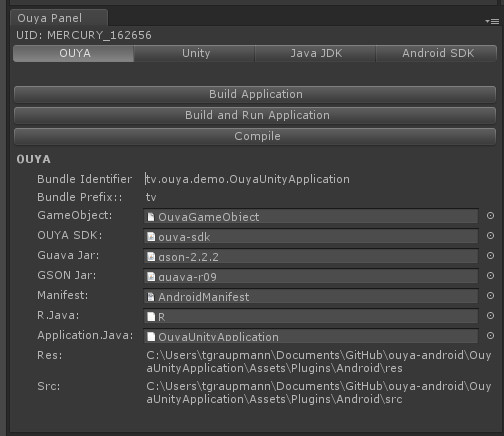
Using the build settings, if you have more than one scene just by marking the toggles, you can switch between the loading scenes. You can use Unity script for switching between scenes.

OUYA Panel

Immediately after importing the OUYA SDK Unity Package, the OUYA Panel becomes available in the Menu. Use Window->Open Ouya Panel to open the OUYA Panel. The OUYA Panel will open which can be docked. The top of the panel has a unique identifier (MachineName\_ProcessPID) to help identify the Unity process which is useful for debugging or task killing.

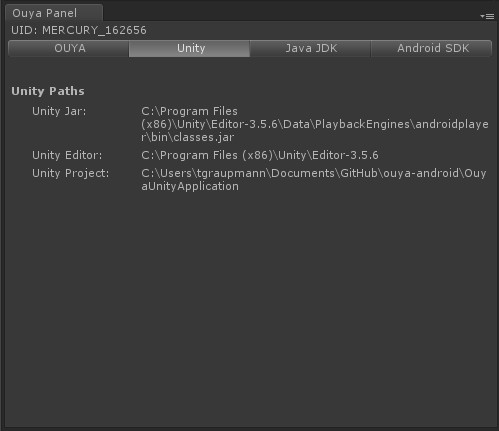
The OUYA panel has several tabs to show information for functional areas: “OUYA”, “Unity”, “Java JDK”, “Android SDK”, and “Android NDK”. Run through each tab to check for missing paths. Any missing path or file with grey out. You may need to browse to find JDK or SDK paths. There should be no greyed out tabs in order for the compile, build, and run automation to work.

# OUYA Tab

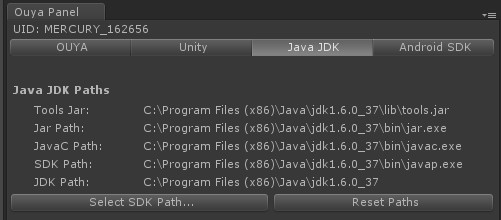
The OUYA tab has a button section and then an info section below. The button “Build Application” will compile the Java Application and immediately build the Application APK for Android. The button “Build and Run Application” compiles the Java Application, builds the APK, and executes the APK on a connected device. The button “Compile” will compile the Java Application. The different buttons are available for your convenience.

The information section has several meta references for quick navigation into the project. If any dependency, path, or file is missing it will be greyed out in the information panel. The bundle identifier that was set in player settings can be changed here. The bundle prefix is used when packaging the application jar and is based on the bundle identifier. The GameObject is a meta reference and clicking the object field will highlight that item in the scene view. New OUYA projects will need the OUYA GameObject added to the scene to accept communication from Java. There is an OUYA GameObject prefab in the project which can be dragged to the scene for easy setup. The OUYA SDK object field when clicked will navigate and highlight the Jar library. The same is true for both Guave and GSON jar libraries. The AndroidManifest is a custom manifest which is used in the Android build process. Make sure that the manifest matches the BundleIdentifier, if not your application will exit on start without detecting the main activity. R.java is autogenerated by the Android SDK when the compile button is pressed. The R.java uses the main layout for automatic generation. The Application.Java references the example OuyaUnityApplication.java which is an Activity which passes input to the OuyaPlugin.jar which can be subscribed to like in the example controller script. “Res” and “Src” folders are custom folders that can be customized during the APK building process. The custom icon is located in the “Res” folder. The Application java file is in the “Src” folder.

# Unity Tab

The Unity Tab has useful path information. The Unity JAR “classes.jar” contains the library necessary for Java to instantiate the Unity Web Player and communicate with the Unity Web Player. The path is detected from Unity and depends on where the editor was installed. The Unity project folder is the base folder for the current project. The OUYA SDK Unity Package should support Unity 3.5.0 or greater.

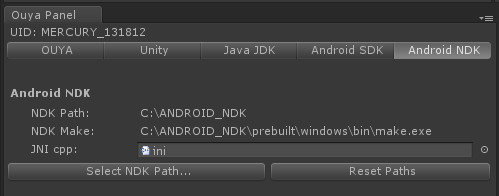
# Java JDK Tab

The Java JDK Tab lets the user specify their path to the Java JDK on Windows. On Mac, the JDK is part of the OS install. Some background, make sure you use a 32-bit install of the Java JDK. You’ll want to use 1.6 as your Java target for optimal compatibility. Only more recent Unity versions support JDK 1.7. The path information below shows the path to the “tools.jar” on Windows. On the Mac the tools jar is “classes.jar”. On Windows the path links to file executables for Jar packaging, Java compiler, and the Java disassembler for generating signatures. On Mac, the path links to file applications, similar to Windows. The button “Select SDK Path..” will open a folder browse dialog to find the JDK path. “Reset paths” will set to a Default. Any missing path will be grayed out.

# Android Tab

The Android Tab has info to point to the Android SDK download location. Windows has an installer for a more standard location; where Mac has an arbitrary unpacked zip. The info pane shows the Android min SDK version from the player settings. Android JAR is used in the classpath when compiling the Application jar. ADB path is the application file path for the ADB platform tool in the Android SDK. ADB is useful to get the list of connected devices, install, and launch APK on devices. APT is yet another Android SDK useful tool. And the SDK path is the path to the Android SDK. The “Select SDK Path…” button will open a folder browse dialog to find the Android SDK base folder. Reset path will use the default. Any missing links will grey out.

# NDK Tab

The NDK Tab shows info for the Android NDK. The NDK can be downloaded from <http://developer.android.com/tools/sdk/ndk/index.html> the android site. Windows and Mac are just an archive, so you’ll need to point the tab to where you unpack. NDK allows compiling the C++ NDK example which potentially can also wrap JNI. You can wrap the JNI from both C++ and C# to pick the best approach for your application. The JNI.cpp example has an easy reference from within the Android plugins folder. On Mac, there is a patch that needs to be applied to NDK in order to fix a compile issue which is explained here: <https://groups.google.com/forum/?fromgroups=#!topic/android-ndk/b4DSxE1NAS0> on Google groups.

# Example Scenes

The package includes several example scenes. Each example scene has a custom script to illustrate the functional area. At a minimum, an OUYA application will need the OuyaGameObject added to the scene. There is an OuyaGameObject prefab that you can drag to the scene to create the game object. This object is responsible for letting the OUYA java interface send messages to Unity. Unity can communicate with Java and C++ via the OuyaSDK class.

## Scene Controller Example

This example scene maps known controllers to a virtual OUYA controller. That said, you may have an unrecognized controller that you are testing while doing development. If your controller is not recognized, let us know by posting in the developer forums. In the example, as you press a controller button, the axis or button will highlight on the virtual control. As you move your physical controller axis, the virtual controller axis will move. The scene has 3 area lights, 3 camera positions, an instance of the controller model, and the example OuyaGameObject script. The 3 camera positions are used to transition the camera. The OuyaGameObject script randomly picks a new camera position every N seconds from the supplied camera transforms. The OuyaGameObject script has meta references to the specific controller parts to control the highlighting and movement. Each button and axis has a MeshRenderer component which is used to access the material and change the color. From the MeshRenderer component, the transform can be accessed to rotate the thumbsticks and triggers. The axis and button values are provided by the OUYA SDK. The text that displays is Unity GUI provided in the OnGUI event of the OuyaGameObject script.

### Script

To be able to attach a script to a GameObject, the example must extend MonoBehaviour.

public class OuyaShowController : MonoBehaviour

You need to assign the developer identifier which you can find in the OUYA dashboard.

private const string DEVELOPER\_ID = "YOUR DEVELOPER ID";

Attached GameObject scripts have an Awake() event that fires the first time the GameObject is activated. This is where we initialized the OUYA SDK by providing your developer identifier.

OuyaSDK.initialize(DEVELOPER\_ID);

To receive controller events, there are two types of events that are registered. Register an input button listener to receive button events.

OuyaSDK.registerInputButtonListener(new OuyaSDK.InputButtonListener<OuyaSDK.InputButtonEvent>()

The input button listener has an onSuccess event which receives the InputButtonEvent. The input button event has the controller details about the button that was pressed.

onSuccess = (OuyaSDK.InputButtonEvent inputEvent) =>

The onFailure event reports errors like controller has been disconnected or has a low battery.

onFailure = (int errorCode, string errorMessage) =>

Register an input axis listener to receive axis events.

OuyaSDK.registerInputAxisListener(new OuyaSDK.InputAxisListener<OuyaSDK.InputAxisEvent>()

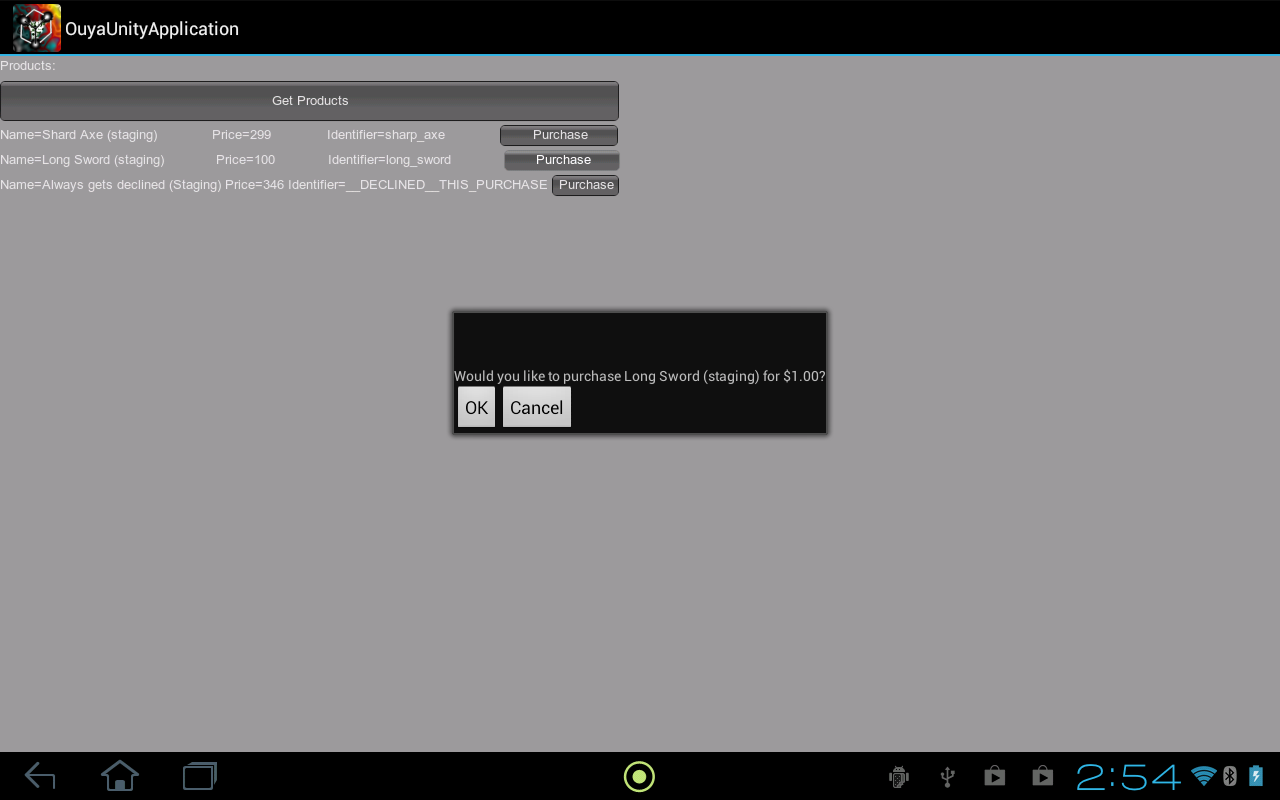
The input axis listener has an onSuccess event which receives the InputAccessEvent. The input axis event has the controller details about the axis that was changed.

onSuccess = (OuyaSDK.InputAxisEvent inputEvent) =>

The onFailure event reports errors similar to the button listener onFailure event.

onFailure = (int errorCode, string errorMessage) =>

## Scene Products Example

The products example shows how to get details about items that can be purchased in the OUYA store. For the example to run on your test device, make sure the OUYA Launcher is installed and running. The OUYA SDK provides methods for getting the details and invoking a purchase. When a purchase is invoked the OUYA Launcher will display a purchase layer above the Unity application. The result of the purchase is returned in the purchase success or failure event which you can handle in your Unity application. You can also get access to purchase receipts to verify the purchase. These methods allow you to create your own store presentation or in-app purchases while the transaction is happening in the OUYA Launcher. In the example, OuyaShowProducts script displays the UI using Unity GUI in the OnGUI event. The Get Products button will invoke getting the products, although this also happens in the Awake event. The purchase button will invoke a purchase event which will open layout above the Unity application in the OUYA Launcher.

### Script

To be able to attach a script to a GameObject, the example must extend MonoBehaviour.

public class OuyaShowProducts : MonoBehaviour

You need to assign the developer identifier which you can find in the OUYA dashboard.

private const string DEVELOPER\_ID = "YOUR DEVELOPER ID";

Attached GameObject scripts have an Awake() event that fires the first time the GameObject is activated. This is where we initialized the OUYA SDK by providing your developer identifier.

OuyaSDK.initialize(DEVELOPER\_ID);

To receive product details, purchase details, and receipts, there are several types of event listeners that are registered. Register a product list listener to receive product events.

OuyaSDK.registerProductListListener(new OuyaSDK.CancelIgnoringIapResponseListener<List<OuyaSDK.Product>>()

The product listener has an onSuccess event which receives the product list.

onSuccess = (List<OuyaSDK.Product> products) =>

The onFailure event reports errors like the server couldn’t be contacted.

onFailure = (int errorCode, string errorMessage) =>

Register a purchase listener to purchase events.

OuyaSDK.registerRequestPurchaseListener(new OuyaSDK.CancelIgnoringIapResponseListener<OuyaSDK.Product>()

The onFailure event reports errors like if the user lacked funds or cancelled the purchase in the OUYA launcher.

onFailure = (int errorCode, string errorMessage) =>

The receipt listener @todo

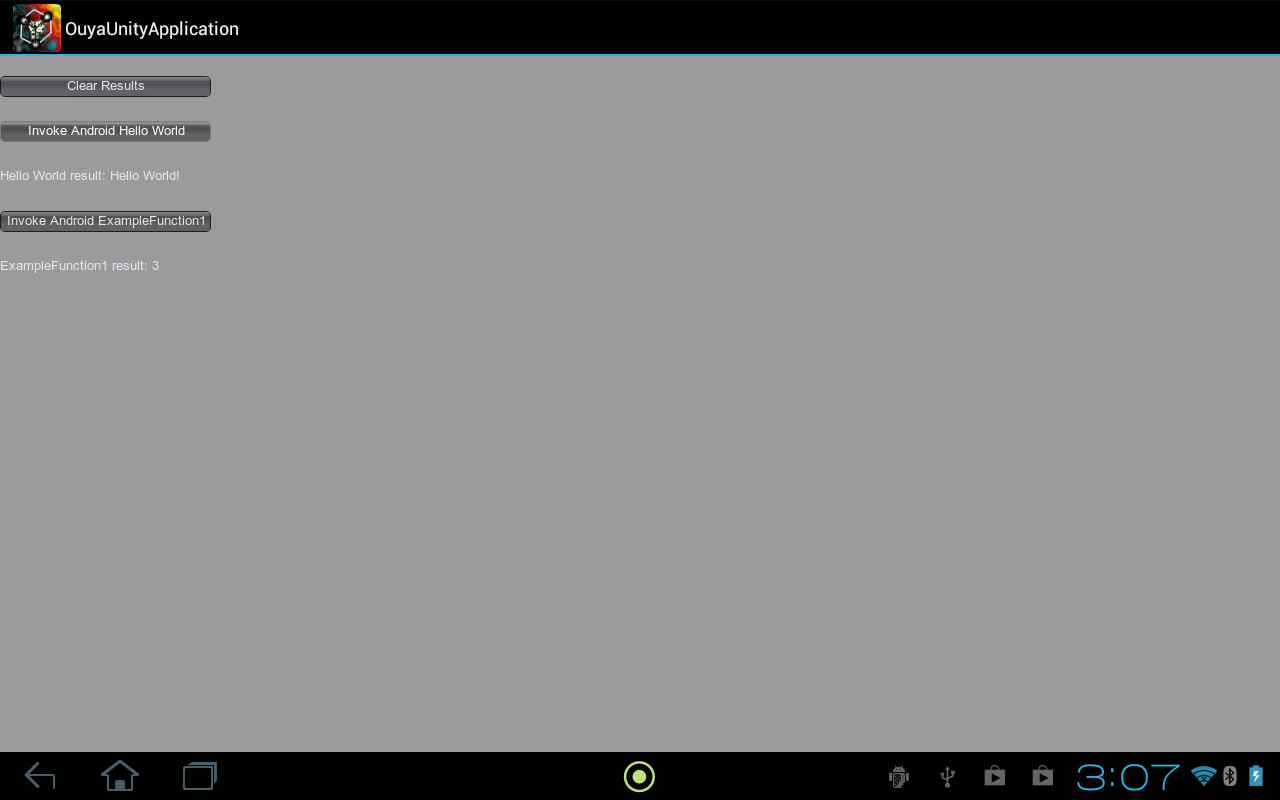
After registering all the listeners, request the product details using a list of purchasables. Your application will already have identifiers for the products you are trying to sell. Invoking the request will asynchronously hit the callbacks.

OuyaSDK.requestProductList(OuyaSDK.PRODUCT\_IDENTIFIER\_LIST, OuyaSDK.getProductListListener());

A purchase is requested by passing the purchasable identifier and the purchase listener.

OuyaSDK.requestPurchase(purchasable.getIdentifier(), OuyaSDK.getRequestPurchaseListener());

## Scene NDK Example

The NDK Example shows how to write C++ and interface that with Unity. The Unity GUI is simply buttons that are invoking C++ methods from Unity. The “Clear Results” button will set the local fields back to their defaults. “Invoke Android Hello World” will invoke the example method which will allocate a string in C++ and pass it back to Unity. Unity will then release the C++ memory string after it is received. The result is displayed in a GUI label. The button “Invoke Android ExampleFunction1” invokes the C++ example which passes a byte array an int parameter by out. These examples are useful when you want to pass something like PNG bytes from C# to C++ and/or get information back from the C++ side. This example uses C++ code which can also interface with the JNI and other C++ libraries. With C++ you can also write code to natively access custom hardware. The C++ is natively compiled in the OUYA panel.

### Script

First take a look at the native C++ code interface, which is being invoked from Unity. This source is located in a file called “jni.cpp” because that’s what the Android NDK build scripts look for. NDK compiles the C++ code and places into the target “Assets\Plugins\Android\libs\armeabi\lib-ouya-ndk.so” library.

extern "C"

{

char\* AndroidGetHelloWorld(long\* size);

void AndroidReleaseMemory(char\* buffer);

void AndroidExampleFunction1(unsigned char\* a, int b, int\* c);

}

Now back to the C# Unity side. To be able to attach a script to a GameObject, the example must extend MonoBehaviour.

public class OuyaShowNDK : MonoBehaviour

For organization, the C++ interface was placed into an AndroidPlugin structure, although this is not required.

private struct AndroidPlugin

To import the method from C++ first you need to name the C++ library.

[DllImport("lib-ouya-ndk")]

Next, the C# matching signature is added to correspond with the C++ interface.

private static extern IntPtr AndroidGetHelloWorld(out long size);

When you invoke AndroidGetHelloWorld if the DLL is missing, the call with throw an exception and likely kill the application on the target device. This method allocates a “Hello World” string in C++ and returns the pointer back to C# while setting the string length using the size out parameter. The memory is released by calling the AndroidReleaseMemory method.

private static extern void AndroidReleaseMemory(IntPtr buffer);

The AndroidExampleFunction1 is an example of passing various argument types. To be able to pass the byte array, a calling convention is specified.

[DllImport("lib-ouya-ndk", CallingConvention = CallingConvention.Cdecl)]

private static extern void AndroidExampleFunction1(byte[] a, int b, out int c);

You can find other marshalling examples and supported types by visiting the <http://pinvoke.net> site.

# Asset Store Resources

There are many packages available in the Unity Asset Store to improve the functionality of your game or even extend the functionality of the Unity editor like this package.

## New to Unity

If your team is new to Unity and they need to ramp up quickly, the “Paint Video Series 1” is an hour of video tutorials on how to build an application in Unity from start to finish. <http://u3d.as/1Dw>

## NDK

Creating matching C#/C++ interfaces that work in Unity for NDK and native plugins can be tedious where the “Standalone Plugin Maker” can assist with auto generating matching signatures at runtime directly in the Editor. <http://u3d.as/3kL>

## Single Development IDE

The OUYA SDK is a mixture of XML, C#, C++, Java, and XML which doesn’t natively export to a single MonoDevelop or Visual Studio project natively. The “Toolbox” allows you to work with all the aspects OUYA SDK in the same Visual Studio project. <http://u3d.as/2gR>

## Verbal Commands

Given that you’ve integrated a Bluetooth headset with your application, verbal commands can add a new dimension into your gameplay. <http://u3d.as/3pP>

## Deployment

With multiple Android devices on the same machine, Unity natively runs your application on the first device it finds. To extend Unity for simultaneous build/deployment/execution on multiple devices on multiple targets check out “GoodDrop” in the Unity Asset Store. You can extend your deployment target to your local machine, remote machines, devices on your machine, devices on your network, and devices over the Internet all from the Unity IDE.