OUYA Unity Plugin

Contents

2. Audience 4

3. Authors 4

4. Supported Platforms 4

5. Introduction 4

6. Forums 4

7. Import Package 4

8. Imported Files 5

9. References 5

10. Example Scenes 6

Scene ShowProducts 6

Scene ShowController 6

11. Android Setup 7

12. Build Settings 8

13. OUYA Tab 10

14. Unity Tab 11

15. Java JDK Tab 11

16. Android Tab 11

17. NDK Tab 12

18. Scene Setup 12

19. Starter Kit 12

20. Example Scenes 13

Scene Controller Example 14

Script 15

21. Scene Multiple Controller Example 17

Script 17

Scene Products Example 19

Script 20

Request product list 20

Request purchase 21

Get receipts 21

Scene NDK Example 22

Script 23

22. Asset Store Resources 24

New to Unity 24

NDK 24

Single Development IDE 24

Verbal Commands 24

Deployment 24

Version Control 24

# 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.

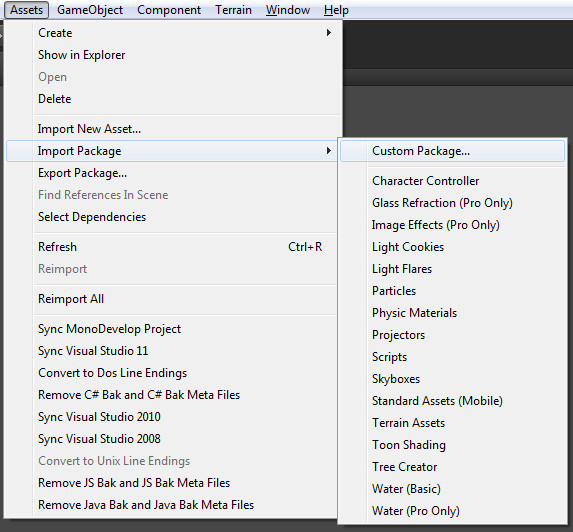
# Introduction

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.

# Forums

The forums are a great place for support and to get answers for common questions. There are several stickied posts in the [Unity section](http://forums.ouya.tv/categories/unity-on-ouya). Be sure to watch out for updates to the SDK in the [minor updates thread](http://forums.ouya.tv/discussion/197/sticky-ouya-unity-sdk-minor-updates#latest). You can [subscribe](https://hashbanggamesportal.ontimenow.com/) for email updates. There are [40 training videos](http://forums.ouya.tv/discussion/200/sticky-ouya-unity-package-videos/p1) and growing. There are also [weekly hangouts](http://forums.ouya.tv/discussion/710/g-hangout-scheduling) to field support questions.

# 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. The original package has been split into Core and Examples. Core contains the OUYA panel and all the important code that you need for building on the OUYA. The examples utilize NGUI and have an optional NGUI distribution package that you can import if you don’t already have NGUI.

# 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>

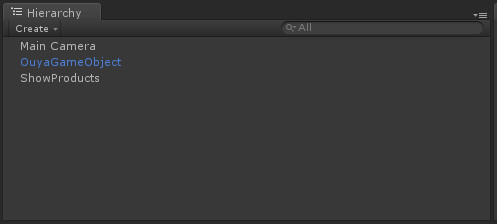
Google-GSON - third party JSON parsing library for Java. <http://code.google.com/p/google-gson/>

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

# Example Scenes

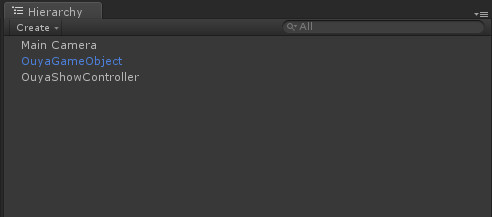
Note: Although some of the example scenes have OuyaGameObject, the preferred way is to put OuyaGameObject into an init scene; see the starter kit for details.

## Scene ShowProducts

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

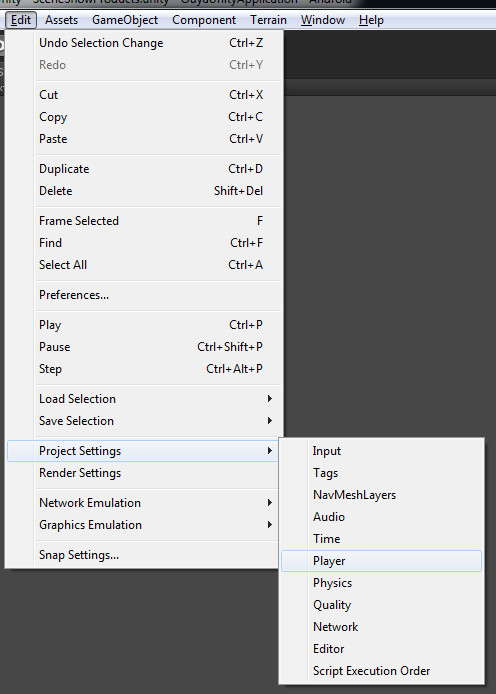
All scenes start with a Main Camera. There’s a custom GameObject added for OuyaGameObject which handles taking messages from the OUYA SDK from Java to C#. The “ShowProduct” GameObject is a simple display script for displaying retrieved products and invoking purchases. Since this example is a single scene that’s why it has the OuyaGameObject. Otherwise it would use the init scene setup.

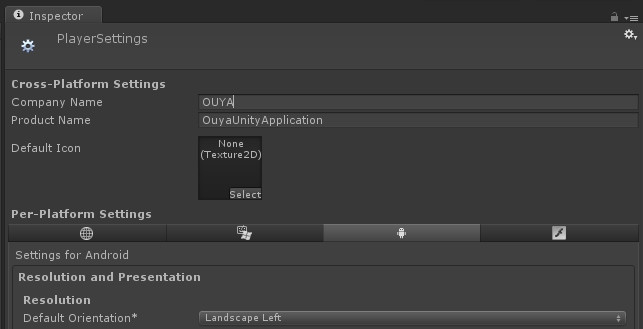
## Scene ShowController

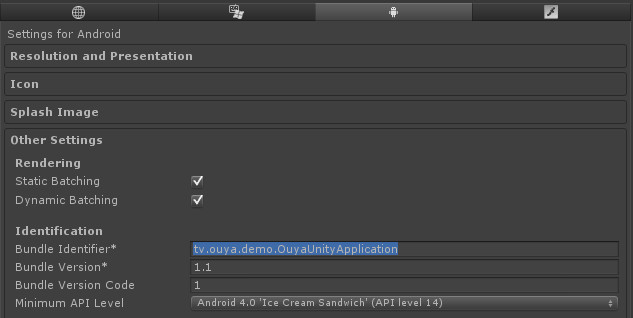
Open the ./Ouya/Examples/Scenes/SceneShowController.unity example scene.

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. Again single scene examples can have the OuyaGameObject as long as the scene is not reloaded. Otherwise the OuyaGameObject would move to an init scene that loads your main menu or game scene.

# Android Setup

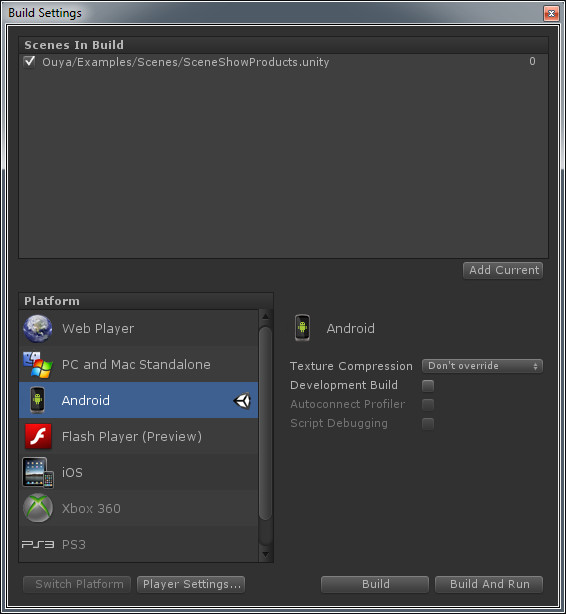
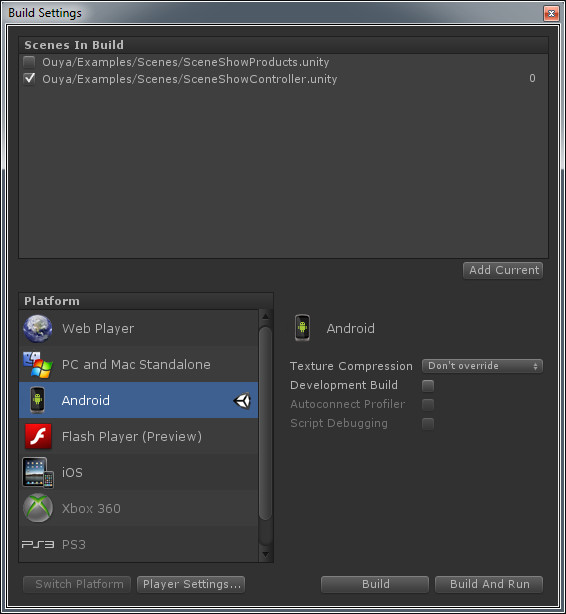
Some player settings must be customized to build on Android. Open the player settings by navigating the menu to the Edit->Project Settings->Player menu item. The PlayerSettings will appear in the inspector.

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).

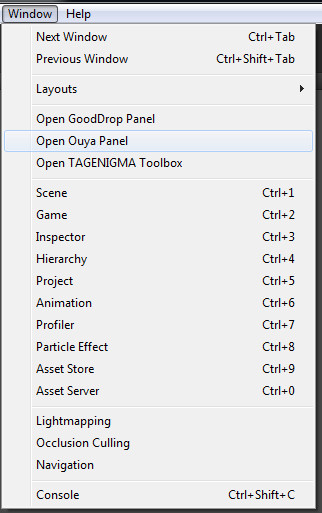
Before any Android application can build you must set the bundle identifier. Be careful that the Bundle Identifier matches what you’ve used in the Android manifest. Also set the Minimum API Level to 16 or better. The OUYA console will be version 16, and the SDK used Android API 16 calls in the application Java for detecting controller connect/disconnect events. The screenshots may show 14 but you want to use 16.

# 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.

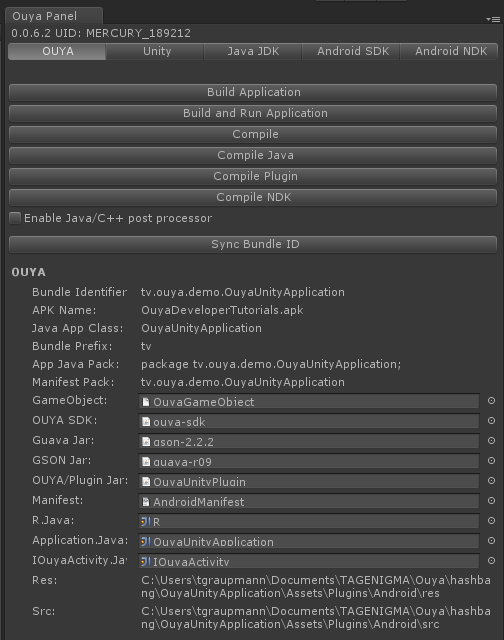
OUYA Panel

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.

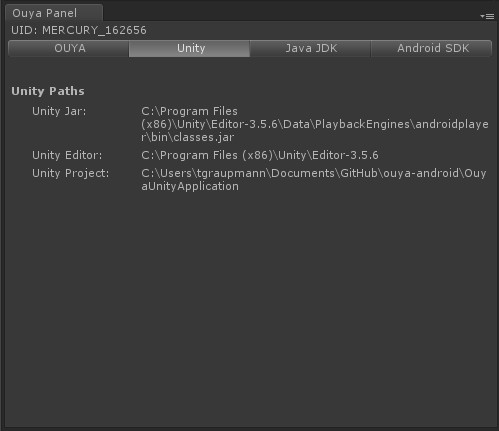
# OUYA Tab

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 compile buttons are available for your convenience which are helpful during customization. The post processor is off by default and will detect any Java/C++ change which will trigger a recompile. The “Sync Bundle ID” button will sync your Android manifest and Java Application package identifiers. The bundle identifier should be unique for every game. The APK name will change the name of the build output file. If you choose your Java Application name, you can specify an alternate name which is intended for hardcore tinkerers. The bundle prefix is just for debugging and is the first part of your bundle identifier. The App Java Pack is the package referenced by your Java Application. If the bundle identifier is mismatched and error will display here, in which case just click the sync button. The manifest pack refers to the bundle identifier specified in the Android manifest, and the sync button will fix a mismatch.

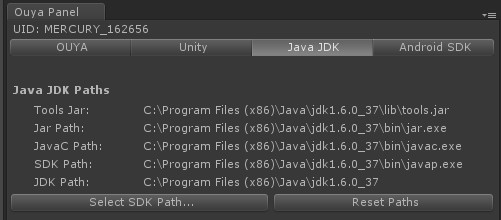


The information section has several meta references for quick navigation into the project. If any dependency, path, or file is missing it will be greyed out in the information panel. The bundle identifier that was set in player settings can be changed here. The bundle prefix is used when packaging the 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.

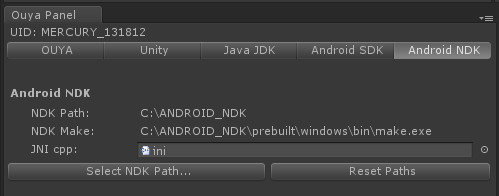
# Java JDK Tab

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.

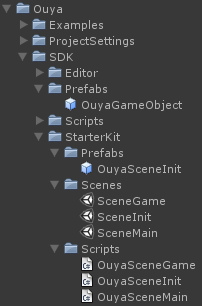
# Android Tab

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.

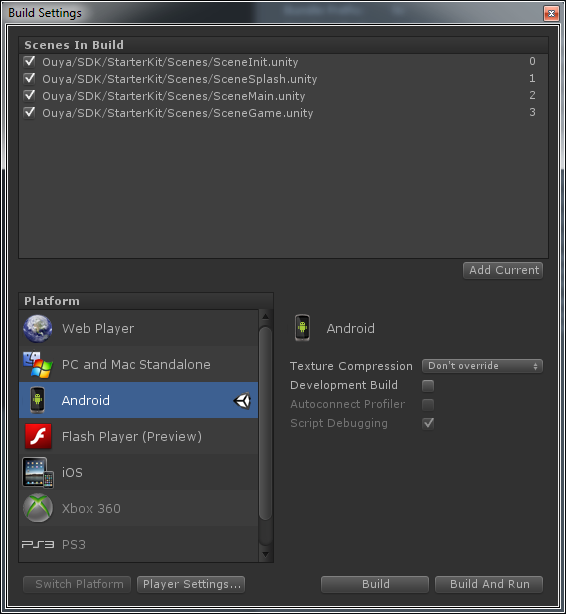
# NDK Tab

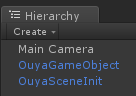
The NDK Tab shows info for the Android NDK. Windows and Mac are just an archive, so you’ll need to point the tab to where you unpack. NDK allows compiling the C++ NDK example which potentially can also wrap JNI. You can wrap the JNI from both C++ and C# to pick the best approach for your application. The JNI.cpp example has an easy reference from within the Android plugins folder. On Windows and 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. The NDK can be downloaded from <http://developer.android.com/tools/sdk/ndk/index.html> the android site.

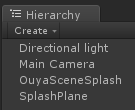
# Scene Setup

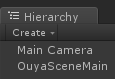
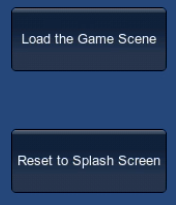
Since the initial writing of this document, controller input for games has been optimized. When examples are a single scene, they each include a single instance of the OuyaGameObject prefab. In other cases where multiple scenes are being used, you want to go with an init scene approach. Your game will usually want an initial scene, splash screen, a menu scene, and a game scene. The initial scene will contain an OuyaGameObject and is the only scene that will contain it. The initial scene will have a script that immediately loads your menu scene. And from your menu scene and game scene they will not contain a duplication OuyaGameObject. Everything other than the initial scene can load new scenes and switch between them freely. In your initial game scene, this is not the place where you enter your developer id that you’ll find on the OUYA developer portal. The OuyaGameObject also has other parameters for turning on debugging and raw debug logging. Debugging is useful for setting up an unrecognized controller.

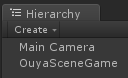
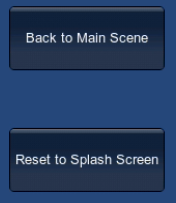
# Starter Kit

The SDK now includes a starter kit to jump start users into a proper scene and project setup. The starter kit has 4 scenes, SceneInit, SceneSplash, SceneMain, and SceneGame. Be sure to add all 4 scenes to the build settings when using the starter kit.

The starter kit begins with SceneInit which has the only OuyaGameObject instance. The OuyaGameObject has don’t destroy on load and should only be placed into the init scene. You should never have duplicate OuyaGameObjects in your scene. The OuyaSceneInit script powers the SceneInit scene. SceneInit immediately loads the next scene which transitions to the SceneSplash.

SceneSplash displays a splash screen that fades in, holds, and then fades out. To display the splash screen a plane was used. The OuyaScreenSplash has a public material reference and the timers affect the alpha channel of the material. The material uses a shader that uses an alpha channel to fade in the splash texture. The directional light brightens the splash image. After the timers elapse, the SceneMain scene is loaded.

SceneMain is a simple example of switching scenes powered by a GUI button. OuyaSceneMain drives the GUI event. The OuyaSceneMain has a scene parameter for “SceneMain” which tells it which scene to load next. The next scene after SceneMain changes to SceneGame when the button “Load the Game Scene” is clicked. The SceneMain is a placeholder for showing the main menu from your game.

SceneGame is just like SceneMain as it has a button to switch back to the main menu SceneMain scene. The button is labelled “Back to Main Scene” and when clicked goes back to the SceneMain scene. OuyaSceneGame is the script that powers the SceneGame scene. This scene is a placeholder for your game scene.

# Example Scenes

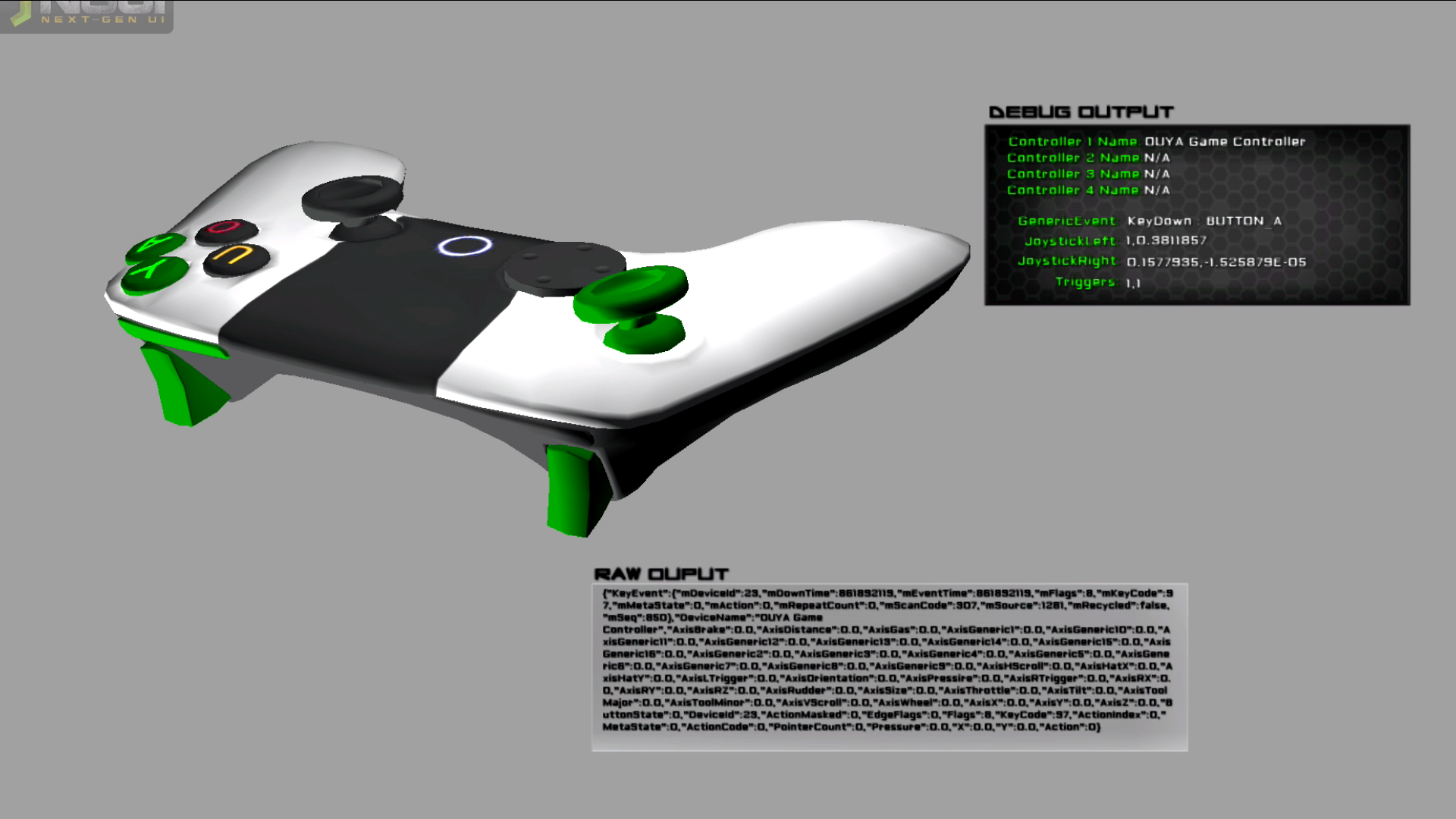
The package includes several example scenes. Each example scene has a custom script that highlights a functional area. Each OUYA example will need the OuyaGameObject added to only the initial scene. Refer to the starter kit for more info. There is an OuyaGameObject prefab that you can drag to the initial 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. The OuyaGameObject is where you enter your developer id from the developer portal. OuyaGameObject is how Java talks to Unity C#. And the OuyaSDK is how Unity C# communicates with Java.

## 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.

The OUYA Bluetooth controller, PS2/3 controllers, and XBOX wired/wireless controllers all work while testing in the Unity3d Editor. The same controllers will also work connected to the OUYA console.

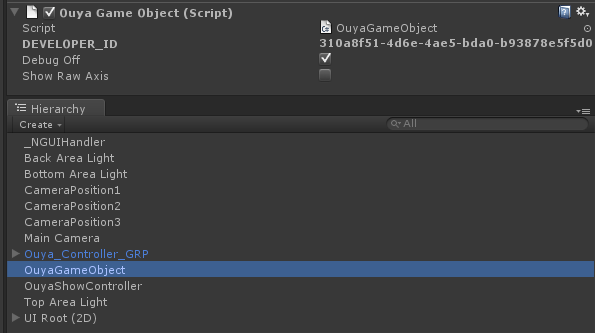
The OUYA SDK provides controller input using state-based and event-based input. In state-based input, the OUYA input API is similar to the Unity input API which takes an additional player parameter. State input can be accessed via the OuyaInputManager with GetAxis, GetButton, GetButtonDown, and GetButtonUp. Event-based input is accessed by a handler and is documented in the script section.



### Script

To be able to attach a script to a GameObject, the example must extend MonoBehaviour. The main logic script should implement the pause and resume interface to handle pause and resume events.

public class OuyaShowController : MonoBehaviour, OuyaSDK.IPauseListener, OuyaSDK.IResumeListener

The developer identifier is now entered in the init scene in the inspector on the OuyaGameObject scene object. The developer id is found in the developer portal.

Per the interface for pause and resume listeners, register in the Awake event.

Conversely, unregister in the OnDestroy event.

void Awake()

{

OuyaSDK.registerPauseListener(this);

OuyaSDK.registerResumeListener(this);

}

void OnDestroy()

{

OuyaSDK.unregisterPauseListener(this);

OuyaSDK.unregisterResumeListener(this);

}

With the pause/unpause listeners you need to need to implement OuyaOnPause and OuyaOnResume events. This is where you can shutdown your game, or reinitialize the game on resume, or potentially save your game state.

public void OuyaOnPause()

{

}

public void OuyaOnResume()

{

}

Register a handler for event-driven input in the Awake event.

void Awake()

{

OuyaInputManager.OuyaButtonEvent.addButtonEventListener(HandleButtonEvent);

}

Unregister the event-driven handler in the OnDestroy event. Clear the key states to avoid button presses getting stuck during a scene change.

void OnDestroy()

{

OuyaInputManager.OuyaButtonEvent.removeButtonEventListener(HandleButtonEvent);

OuyaInputManager.initKeyStates();

}

You can get button notifications for multiple controllers in the event handler.

private void HandleButtonEvent(OuyaSDK.OuyaPlayer p, OuyaSDK.KeyEnum b, OuyaSDK.InputAction bs)

{

}

The handler receives an event so you can trap player, button, and button state combinations. For example, you can detect when the O button has a button down for controller 1 event. The potential states are in an enum list for easy access.

if (b.Equals(OuyaSDK.KeyEnum.BUTTON\_O) && bs.Equals(OuyaSDK.InputAction.KeyDown))

The KeyEnum has all the OUYA controller buttons. InputActions map to KeyDown and KeyUp.

State-based input can be accessed via the OuyaInputManager in Update or FixedUpdate events. Axis states can be accessed in GetAxis.

float lx = OuyaInputManager.GetAxis("LX", OuyaSDK.OuyaPlayer.player1);

float ly = OuyaInputManager.GetAxis("LY", OuyaSDK.OuyaPlayer.player1);

State-based input for buttons can be accessed via the OuyaInputManager in Update or FixedUpdate events.

if (OuyaInputManager.GetButtonDown("RB", OuyaSDK.OuyaPlayer.player1))

State-based input for GetButtonUp can be used in Update or FixedUpdate.

if (OuyaInputManager.GetButtonUp("LB", OuyaSDK.OuyaPlayer.player1))

# Scene Multiple Controller Example

The multiple controller example shows use of multiple controllers independently controlling characters.

### C:\Users\tgraupmann\Documents\GitHub\ouya-android\OuyaUnityApplication\Docs\29_MultipleControllers.pngScript

The OuyaInputHandlerExample script drives the skeleton characters and determines which controller corresponds to each character. In the inspector, OuyaInputHandlerExample has a Player field that determines the controller player that controls each skeleton. OuyaInputHandlerExample has the same setup as the previous example.

public class OuyaInputHandlerExample : MonoBehaviour, OuyaSDK.IPauseListener, OuyaSDK.IResumeListener

The OuyaInputHandlerExample has a Player enum that is used in the input to control a specific player.

public OuyaSDK.OuyaPlayer player;

The player field is used in the OuyaInputManager GetAxis so that the specific player controls the attached GameObject.

Vector2 point = convertRawJoystickCoordinates(OuyaInputManager.GetAxis("LX", player), OuyaInputManager.GetAxis("LY", player), .25f);

The button handler has a player filter and any code after the filter condition is targeted at the correct GameObject.

void HandleButtonEvent(OuyaSDK.OuyaPlayer p, OuyaSDK.KeyEnum b, OuyaSDK.InputAction bs)

{

if (!player.Equals(p)) { return; }

if (b.Equals(OuyaSDK.KeyEnum.BUTTON\_O) && bs.Equals(OuyaSDK.InputAction.KeyDown))

{

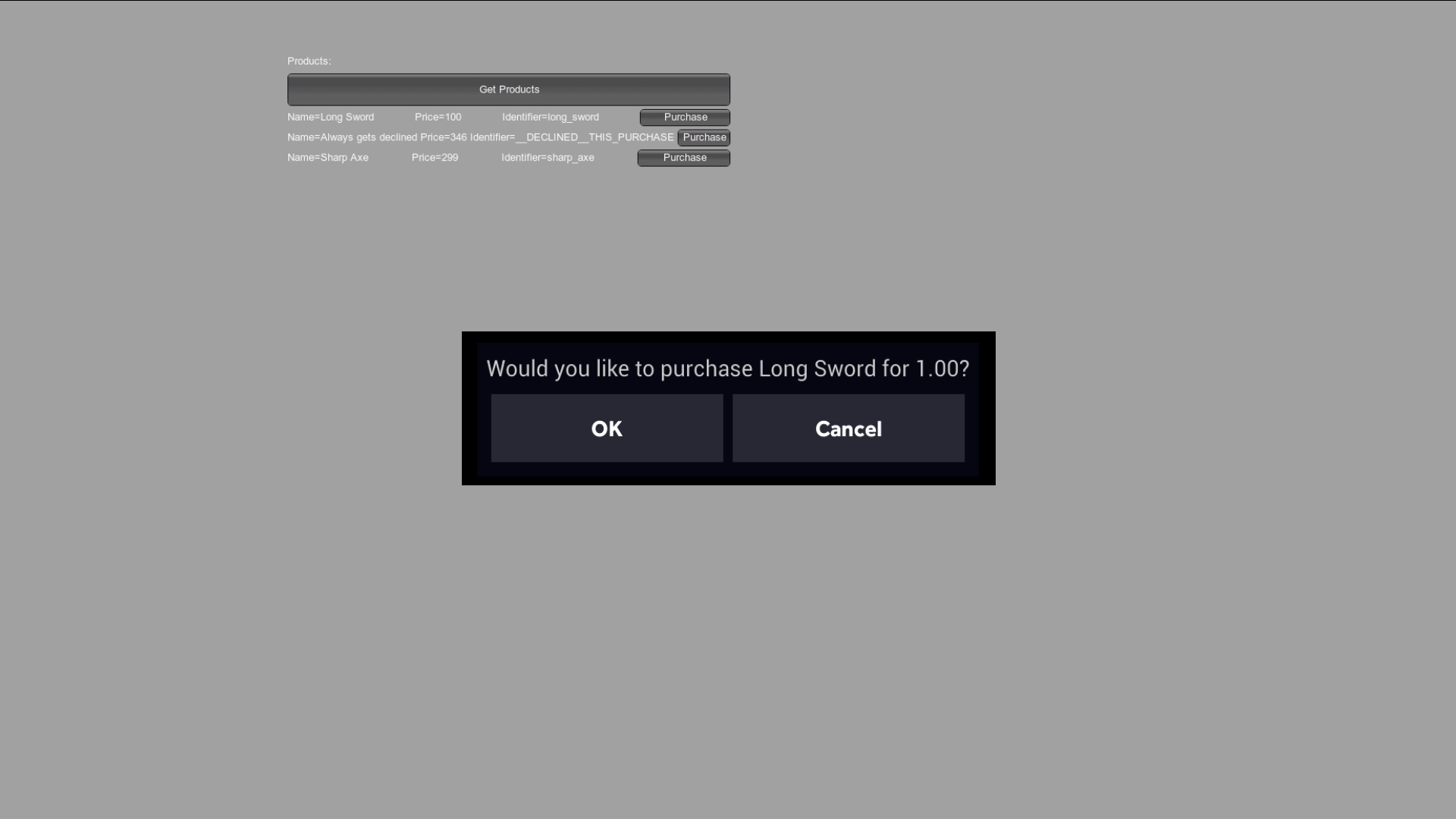
this.animation.Play("attack");

}

}

## Scene Products Example

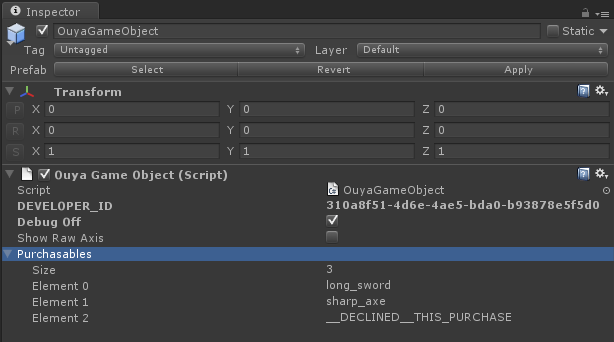
The products example shows how to get details about items that can be purchased in the OUYA store. The example comes with a list of example products. You will need to register your own products and product ids in the developer portal. For the example to run on your test device, make sure the OUYA Launcher 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

Add listeners for the purchase-iap-system areas for GetProducts, Purchase, and GetReceipts.

public class OuyaShowProducts : MonoBehaviour, OuyaSDK.IPauseListener, OuyaSDK.IResumeListener, OuyaSDK.IGetProductsListener, OuyaSDK.IPurchaseListener, OuyaSDK.IGetReceiptsListener

Enter your purchasables into the OuyaGameObject. The developer creates purchasables in the developer portal. And then the OuyaGameObject has a ProductKey list where you enter the product app ids from the developer portal.

Register your GetProductsListener, PurchaseListener, and GetReceiptsLister in the Awake and clear in the OnDestroy events.

void Awake()

{

OuyaSDK.registerGetProductsListener(this);

OuyaSDK.registerPurchaseListener(this);

OuyaSDK.registerGetReceiptsListener(this);

}

void OnDestroy()

{

OuyaSDK.unregisterGetProductsListener(this);

OuyaSDK.unregisterPurchaseListener(this);

OuyaSDK.unregisterGetReceiptsListener(this);

}

It’s recommended that you avoid switching scenes when events are pending. As in, don’t request receipts and immediately switch scenes before the callback happens. Also be sure to invoke a single request and wait for the response before invoking the iap service again.

The listeners implement an OnSuccess, OnFailure, and OnCancel callback.

### Request product list

To get a list of products (which includes the price information), invoke OuyaSDK.requestProductList and pass a list of purchasables. You likely want to obtain prices from the server to avoid hardcoding prices and showing an inaccurate price.

public void OuyaGetProductsOnSuccess(List<OuyaSDK.Product> products)

{

}

public void OuyaGetProductsOnFailure(int errorCode, string errorMessage)

{

}

public void OuyaGetProductsOnCancel()

{

}

### Request purchase

Start a purchase by invoking OuyaSDK.requestPurchase and pass the product identifier that you obtained from the list of purchasables. Be sure to wait for the callback result before invoking another purchase.

public void OuyaPurchaseOnSuccess(OuyaSDK.Product product)

{

}

public void OuyaPurchaseOnFailure(int errorCode, string errorMessage)

{

}

public void OuyaPurchaseOnCancel()

{

}

### Get receipts

Games need to check whether the game has been unlocked or whether to show a BUY NOW button. Check the user’s receipt list for the purchasable identifier to see if the content has been unlocked. Call OuyaSDK.requestReceiptList to get the list of receipts and check the identifier.

public void OuyaGetReceiptsOnSuccess(List<OuyaSDK.Receipt> receipts)

{

}

public void OuyaGetReceiptsOnFailure(int errorCode, string errorMessage)

{

}

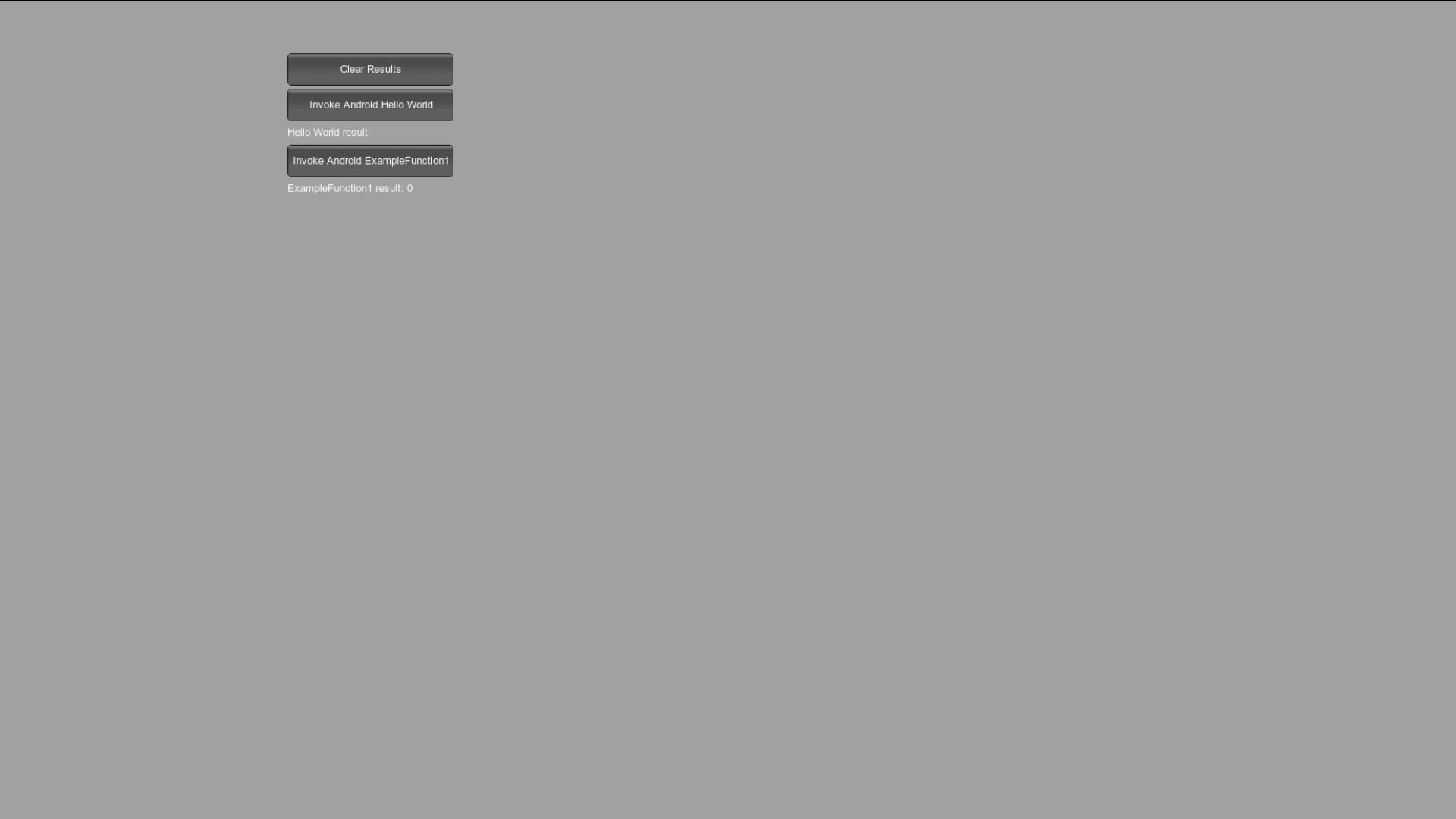
public void OuyaGetReceiptsOnCancel()

{

}

## 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.

## Version Control

If your project uses SVN for version control, check out “Good SVN” for integrated version control in Unity which works on Windows and Mac. <https://www.assetstore.unity3d.com/#/content/7242>