Android System for Unity

Version 1.3

Unity is a very extensible game engine, but lacks of good integration with native functionalities of the systems running on.

Specifically for Android, there's no integration with in-app purchase or ads, and even access to system information is very poor.

Even worse, the only way to call C# methods from Java code requires developer to:

- Create and manage a library on Android side;
- Compress the data in a string to call UnityPlugin.UnitySendMessage;
- Parse the data received in Mono side.

This makes the developer to lose focus on your game and get worried about system functionalities.

Android System allows developers to interact with native Android functionalities without any Java code. You can listen events with delegates, parse the data as it comes and even start and interact with Android services directly.

This document is separated in two parts: the first explains the services exposed by this plugin, and the second presents the low level features that allows deeper interaction with Android OS.

Services

Telephony

The **Telephony** class is used to initiate phone calls from application, send SMS messages and pick any contact from phone's contacts.

To make a phone call, you must use the **MakeCall** method. The first parameter is the number to call, and the second is used to call immediatly (if true) or to just open the caller application (if false):

```
Telephony.MakeCall("55550009", true);
```

To send SMS messages, use the **SendSMS** method:

```
GUIText messageStatus; // This is showed in GUI
Telephony.SendSMS("55550009",  "This is a SMS message",
                  (sentOK) => {
                     messageStatus.text = sentOK
                                          ? "SMS sent successfully"
                                          : "Failed to send SMS";
                               },
                  (deliveredOK) => {
                     messageStatus.text = deliveredOK
                                         ? "SMS delivered successfully"
                                         : "Failed to deliver SMS";
                               } );
```

The first argument is the phone number to send the SMS, the second is the text message to send.

The third and fourth parameters are optional. The third parameters is a delegate to receive the status when the message was sent, and the last parameter is another delegate to receive the status when the message was delivered to the recipient.

Picker

The **Picker** class allows easy access to images, videos and other media types. It opens the gallery, allowing the user to choose the media object, and returns it through a callback. Also, you can take a picture from the native camera app, and return it to the application in the same way.

To access gallery, you need to add **READ_EXTERNAL_STORAGE** permission to AndroidManifest.xml. Below is and example that pick and image from the gallery and is assigned to some material as the main texture:

```
Picker.PickImageFromGallery( (Texture2D tex) => {
                                obj.material.mainTexture = tex;
                               } );
```

IMPORTANT: To use **Picker** service, you need to use the custom Android activities from the **unityandroidsystem.jar** in the plugin's package. To more information, follow the configuration advice from **OnActivityResult** section later in this document.

NFC

This feature allows immediate sharing of small data only by aproaching two phones. It can be used to easily exchange connection info between two devices, and even exchange game itens like power-ups.

Before using NFC, the application need to check if the feature is supported with **NFC.Supported**.

The service can publish messages as string or raw binary. To publish a string message, you must simply call:

```
NFC.Publish( "NFC test message" );
```

And to receive messages, just register a listener calling:

```
GUIText txt;
NFC.Subscribe( ( string msg ) => { txt.text = msg; } );
```

IMPORTANT: To use **NFC** service, you need to use the custom Android activities from the **unityandroidsystem.jar** in the plugin's package. To more information, follow the configuration advice from **OnNewIntent** section later in this document.

Low Level APIs

Broadcast Receiver

Many system events in Android, like device boot or a headset connection, are delivered through broadcast messages. Any application can register a receiver and handle the information passed, and so trigger an event based on the message. A list with some events that Android broadcasts is presented below:

- Device boot and shutdown;
- Battery level;
- Headset plugged and unplugged;
- Wi-fi networks found;
- Phone call or SMS received.

Following is a code to allow the application to receive the event of battery low:

```
public BroadcastReceiver receiver;
public void RegisterForBatteryEvent() {
    receiver = new BroadcastReceiver();
    receiver.OnReceive +=
                (context, intent) => {
                   string action = intent.Call<string>("getAction");
                   if (BATTERY_LOW == action) {
                      // Quit application
                      Application.Quit();
                   }
                };

    receiver.Register(BroadcastActions.ACTION_BATTERY_LOW);
}
```

As broadcast receiver action runs in the application's main thread, it is important to do not run long operations on it. The ability to run broadcast actions in a background thread will be available in a future release.

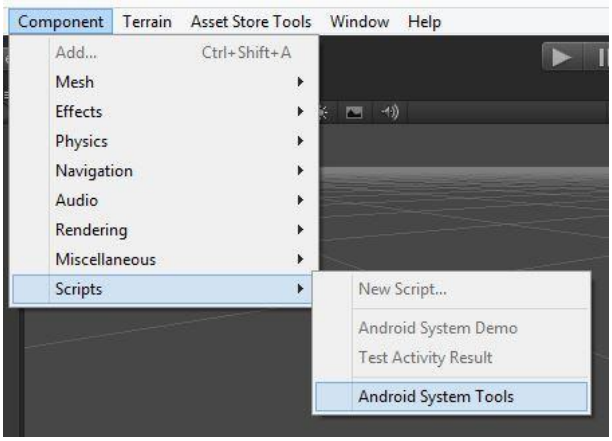
Many broadcast actions return some data, for example the battery level or name of connected headset. To get a list of available broadcast actions and data received in each action, see the list in Android Intent class documentation here. If you need more information, you can consult the documentation for BroadcastReceiver in this link.

Service Connection and Binder wrappers

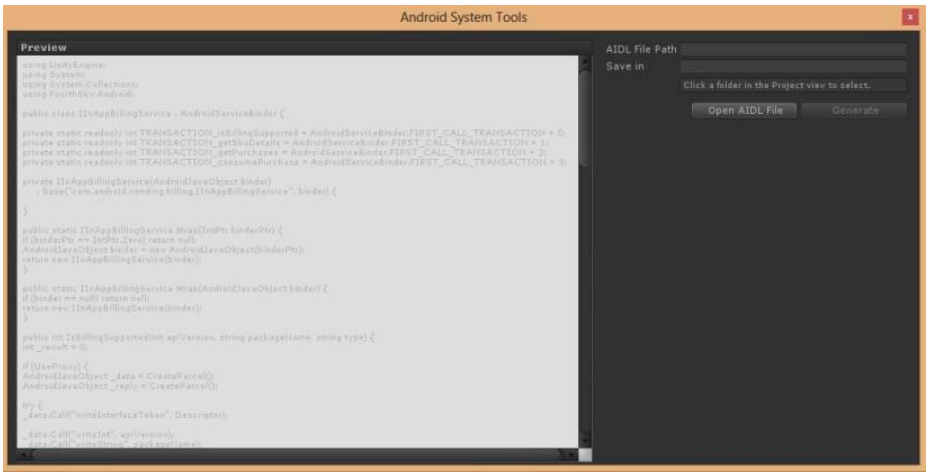
To interact with some Android services, first you have to establish a "connection" with it, and then use the object representing the connection, which is called Binder. The Binder is used to call operations on service, and receive responses from it.

To use services, first we have to create c# wrappers for binders, using the AIDL file that represents the interface to interact with the service.

In Unity, select Component -> Scripts -> Android System Tools:



Opening Android System Tools window, first you have to choose and AIDL file. Click in "Open AIDL file" button, and choose the IInAppBillingService.aidl file in extras/google/play_billing/in-app-billing-v03 directory, inside Android SDK path. The C# wrapper will be generated and showed in Preview pane:



Finally, choose a directory in Project view and click in **Generate**. The wrapper class will be generated, and can be edited if needed.

In case of any of the situations below, the **Generate** button will be disabled:

- The wrapper is already generated;
- Another class with the same name of the wrapper exists in the project;
- Name of the wrapper is invalid;

- The target path is empty or invalid.

Following is an example that uses the wrapper generated from IInAppBillingService.aidl to check if the service is supported in the device:

```
// Action string
public static readonly string BILLING_ACTION = "com.android.vending.bill
ing.InAppBillingService.BIND";
IInAppBillingService service = null;

public void BindBillingService() {
    // Create service connection
    ServiceConnection connection = new ServiceConnection();

    // Bind delegate methods
    connection.OnServiceConnected +=
           (AndroidJavaObject namePtr, AndroidJavaObject binder) {
                if (binderPtr == IntPtr.Zero) {
                     Debug.Log("Something's wrong");
                }

                // Wrap binder pointer with generated wrapper
                service = IInAppBillingService.Wrap(binder);
                Debug.Log("Billing service connected");

                // Check if service is supported
                int responseCode = service.IsBillingSupported(3,
                                                    packageName,
                                                    "inapp");
                if (responseCode == 0)
                     Debug.Log("Billing service is supported");
                else
                     Debug.Log("Billing service is unsupported");
           };

    connection.OnServiceDisconnected +=
           (AndroidJavaObject name) {
                // Clear wrapper
                service.Dispose();
                service = null;

                Debug.Log("Billing service disconnected");
           } ;

    // Connect to service
    connection.Bind(BILLING_ACTION);
}
```

OnActivityResult

Sometimes you want to interact with another Android app, like camera, contacts or image gallery, pick an item from them, and return the chosen item to the calling application.

In Android, this is done by calling **startActivityForResult** from the Activity, specifying the application we want to get data. The return values are received in the implementation of **onActivityResult** method in the calling application.

The Android System plugin implements the same behaviour, ported to Unity C#. You can see and example in **Picker** service. The service calls the gallery (or contacts application, if picking an contact), and returns the choosen image as a **Texture2D** object.

To enable application to receive **OnActivityResult** calls, the developer needs to configure **AndroidManifest.xml** (or use the one from plugin package) to use the custom Unity Activities created for **Android System**.

OnNewIntent

This is a very particular callback, and their uses is better explained in Android documentation. Currently, the only use for this callback is receiving messages using the **NFC** service.

To enable application to receive **OnNewIntent** calls, the developer needs to include in **AndroidManifest.xml** (or use the one from plugin package) the **UnityNFCActivityReceiver** activity. The reason for this is that the flow for NFC and onNewIntent activity method doesn't works with **Unity3D** lifecycle, so the need to handle this callback in another activity, and then returns fast to application.

Current Limitations

- You cannot register some custom C# **BroadcastReceiver** in AndroidManifest.xml. We will provide a solution in the next release.

Roadmap

We have some great ideas for the future releases, some of them include:

- A tool to create customized AndroidManifest.xml, to add permissions and other configurations;
- A generator to wrap any Android class or interface and call their methods in C#;
- Creating and handling status bar notifications in Mono C#.

Changes in document

1.3

- All callbacks now returns instances of **AndroidJavaObject** instead of **IntPtr** pointers;
- Also, the callbacks can be lambdas or instance methods.
- Added services: **Picker**, **NFC**, and **Telephony**;
- Added explanation about **OnNewIntent** callback

1.2

- Editor classes now wrapped in DLL library, to allow execution even with some compilation errors
- **AndroidSystem.ConstructJavaObjectFromPtr** – due to changes in AndroidJavaObject class since Unity 4.2, it's recommended (mandatory in 4.2) to use this method to create instances of AndroidJavaObject from IntPtr values