Android System for Unity

Version 1.2

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.

Basics

There are some ways to ensure that the communication with Java works as expected:

- Every delegate method SHALL be static, because the method reference will be passed between many layers of the system;
- For the same reason, do not use lambda expressions because they behave as instance variables;
- In the current version of the plugin, all Android objects is passed to delegates as C# System.IntPtr objects (future versions will use AndroidJavaObject instances).

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:

```
1.  public static readonly string BATTERY_LOW = "android.intent.action.BATTERY_LOW";
2.  public BroadcastReceiver receiver;
3.
4.  public void RegisterForBatteryEvent() {
5.      receiver = new BroadcastReceiver();
6.      receiver.OnReceive += OnReceive;
7.      receiver.Register(BATTERY_LOW);
8.  }
9.
10. private static void OnReceive(IntPtr contextPtr, IntPtr intentPtr) {
11.     AndroidJavaObject intent =  AndroidSystem.ConstructJavaObjectFromPtr(intentPtr);
12.     string action = intent.Call<string>("getAction");
13.
14.     if (BATTERY_LOW == action) {
15.         Debug.Log("Battery is low, close app or connect power");
16.     }
17. }
```

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.

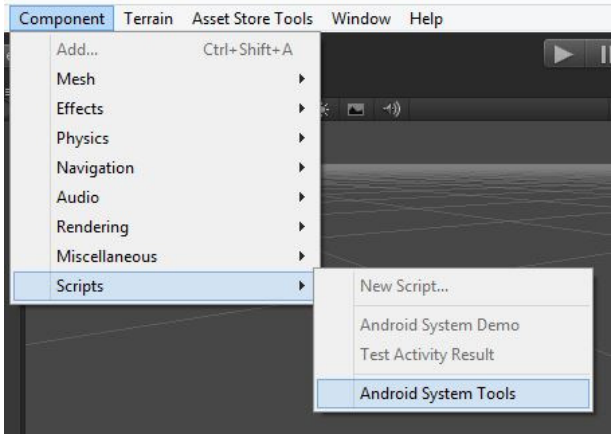
Many broadcast actions return some data, for example the battery level or name of connected headset. To get a list of available broadcast actions and data received in each action, see the list in Android Intent class documentation here. If you need more information, you can consult the documentation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

Service Connection and Binder wrappers

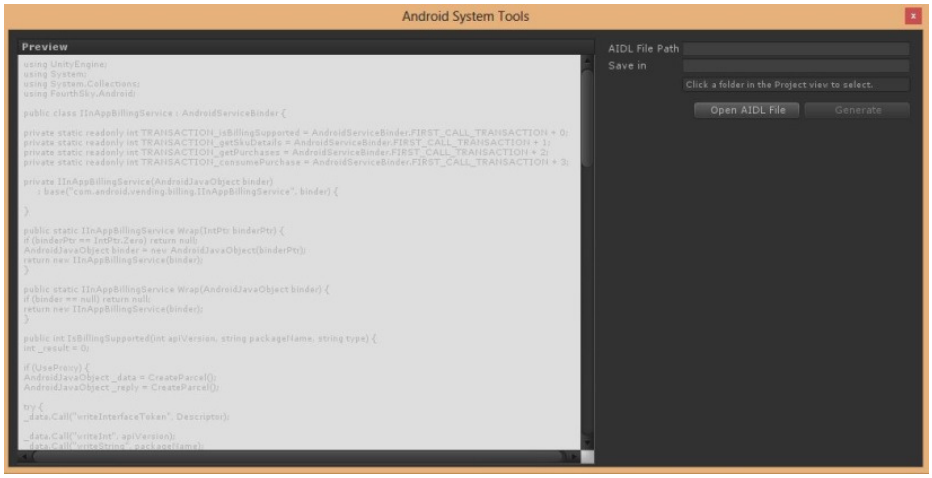
To interact with some Android services, first you have to establish a "connection" with it, and then use the object representing the connection, which is called Binder. The Binder is used to call operations on service, and receive responses from it.

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:

```
1.  // Action string
2.  public static readonly string BILLING_ACTION = "com.android.vending.billing.InAppBillingService.BIND";
3.  private IInAppBillingService service = null;
4.
5.  public void BindBillingService() {
6.      // Create service connection
7.      ServiceConnection connection = new ServiceConnection();
8.
9.      // Bind delegate methods
10.     connection.OnServiceConnected += OnServiceConnected;
11.     connection.OnServiceDisconnected += OnServiceDisconnected;
12.
13.     // Connect to service
14.     connection.Bind(BILLING_ACTION);
15. }
16.
17. private static void OnServiceConnected(IntPtr namePtr, IntPtr binderPtr) {
18.     if (binderPtr == IntPtr.Zero) {
19.      Debug.Log("Something's wrong");
20.     }
21.
22.     // Wrap binder pointer with generated wrapper
23.     service = IInAppBillingService.Wrap(binderPtr);
24.     Debug.Log("Billing service connected");
25.
26.     // Check if service is supported
27.     int responseCode = service.IsBillingSupported(3, packageName, "inapp");
28.     if (responseCode == 0)
29.         Debug.Log("Billing service is supported");
30.     else
31.         Debug.Log("Billing service is unsupported");
32. }
33.
34. private static void OnServiceDisconnected(IntPtr name) {
35.     // Clear wrapper
36.     service.Dispose();
37.     service = null;
38.
39.     Debug.Log("Billing service disconnected");
40. }
```

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#. The sample code below is used to start the gallery app in the device, using **AndroidSystem.StartActivityForResult**, to choose an image from external or internal storage. The URI of the image is returned in **OnActivityResult** delegate implementation, and so we can obtain the real path (using some Android magic).

```
1.  // Arbitrary request code
2.  private static readonly int CAMERA_PICK = 1423;
3.
4.  // Android intent action for pick object from another app
5.  private static readonly string ACTION_PICK = "android.intent.action.PICK";
6.
7.  // Variable used to signal Update method to create texture
8.  private bool foundTexture = false;
9.
10. public void PickImageFromGallery(bool externalStorage) {
11.
12.     // Alert: "$" character denotes inner classes
13.     // From MediaStore.Images class (there are also MediaStore.Video, MediaStore.Audio and MediaStore.Files)
14.     using (AndroidJavaClass mediaClass = new AndroidJavaClass("android.provider.MediaStore$Images$Media")) {
15.         AndroidJavaObject pickImageURI = null;
16.         if (externalStorage) {
17.             // ... pick URI for external content (SD card) ...
18.             pickImageURI = mediaClass.GetStatic<AndroidJavaObject>("EXTERNAL_CONTENT_URI");
19.
20.         } else {
21.             // ... or internal content (internal memory) ...
22.             pickImageURI = mediaClass.GetStatic<AndroidJavaObject>("INTERNAL_CONTENT_URI");
23.
24.         }
25.
26.         // ... and start gallery app to pick an image.
27.         AndroidSystem.StartActivityForResult(ACTION_PICK, pickImageURI, PICK_IMAGE, OnActivityResult);
28.     }
29. }
30.
31. private static void OnActivityResult(int requestCode, int resultCode, IntPtr intentPtr) {
32.     try {
33.         if (requestCode == PICK_IMAGE) {
34.             switch(resultCode) {
35.                 case AndroidSystem.RESULT_OK:
36.                     AndroidJavaObject intent = AndroidSystem.ConstructJavaObjectFromPtr(intentPtr);
37.
38.                     // Get URI path (for example: content://media/external/images/media/712)
39.                     string contentUri = intent.Call<AndroidJavaObject>("getData").Call<string>("toString");
40.                     Debug.Log("URI from picked image: " + contentUri);
41.
42.                     // Get real path of file (for example: mnt/images/IMG357.jpg )
43.                     imagePickPath = GetFilePathFromUri(contentUri);
44.                     Debug.Log("File path of picked image: " + imagePickPath);
45.
46.                     // Load bytes and signal main thread create texture
47.                     foundTexture = true;
48.                 break;
49.
50.                 case AndroidSystem.RESULT_CANCELED:
51.                     Debug.Log("Pick image operation cancelled");
52.                     break;
53.
54.                 default:
55.                     Debug.LogError("Error occurred picking image from gallery");
56.                     break;
57.                 }
58.         }
59.
60.     } catch (Exception ex) {
61.         Debug.LogException(ex);
62.     }
63. }
```

Loading the image bytes to a Unity Texture2D should be done in the **Update** method of the MonoBehavior, due to restrictions imposed by Unity. The full code is in the sample scene.

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

Current Limitations

- All implementation for delegate methods used in plugin SHALL be static methods in C# classes. Do not use lambda expressions, because they are objects that can be collected and so became invalid for plugin calls;
- The delegate methods returns pointers (instances of System.IntPtr) to Android classes, instead of instances of **AndroidJavaObject**;
- You cannot register some custom C# **BroadcastReceiver** in AndroidManifest.xml;
- Due to changes in AndroidJavaObject in Unity 4.2, is recommended to use **AndroidSystem.ConstructJavaObjectFromPtr** to create instances of AndroidJavaObject from IntPtr values.

All limitations above are under investigation, and the enhancements will be available in a future 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

1.2

- **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