

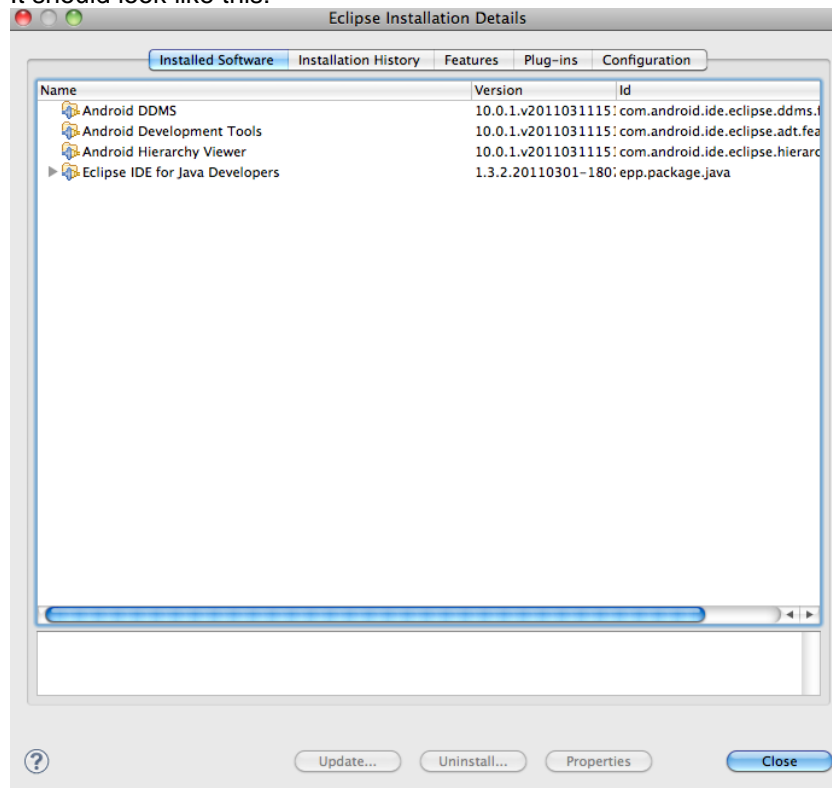
Ratrod Studio Unity Android Storekit v1.0

Getting Your Public Key (Base 64-encoded RSA)

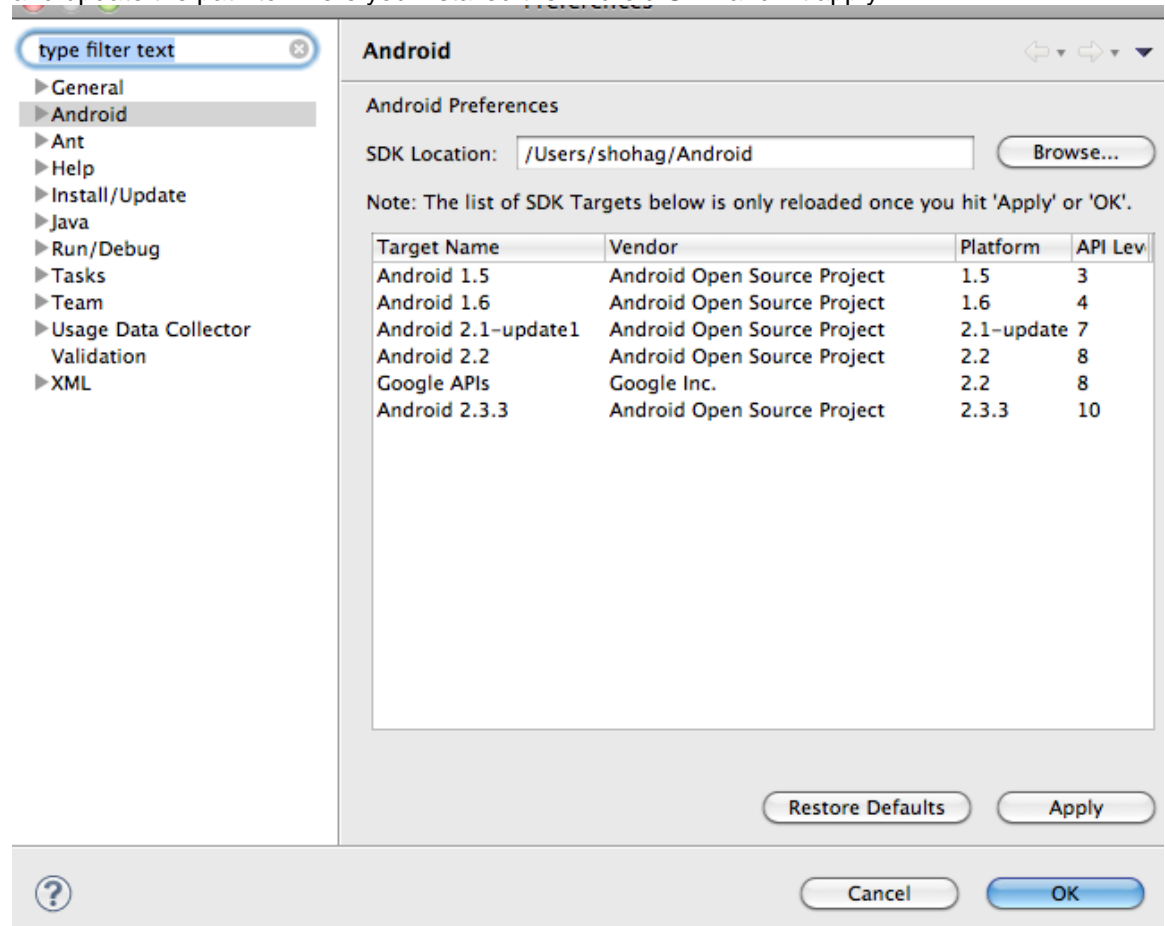
1. Log into your Android Development panel.
2. On the top left, under your account name, click on "Edit Profile".
3. You will see under "Licensing & In-app Billing" your Public Key.
4. Cut and paste the content of the entire box. You will need it during the following steps.

Setting Dev Environment

1. Download Eclipse
 - a. Download Eclipse from this [website](#)
 - b. Choose your platform for it.
 - c. You can choose [Helios](#).
2. Download the Android SDK
 - a. [Android SDK](#) is available here.
 - b. All the installation steps are available [here](#).
 - c. Make sure to also get the [ADT plugin](#).
 - d. After setting the ADT plugin in Eclipse, go to Eclipse -> About Eclipse -> Installation Details
 - e. It should look like this:



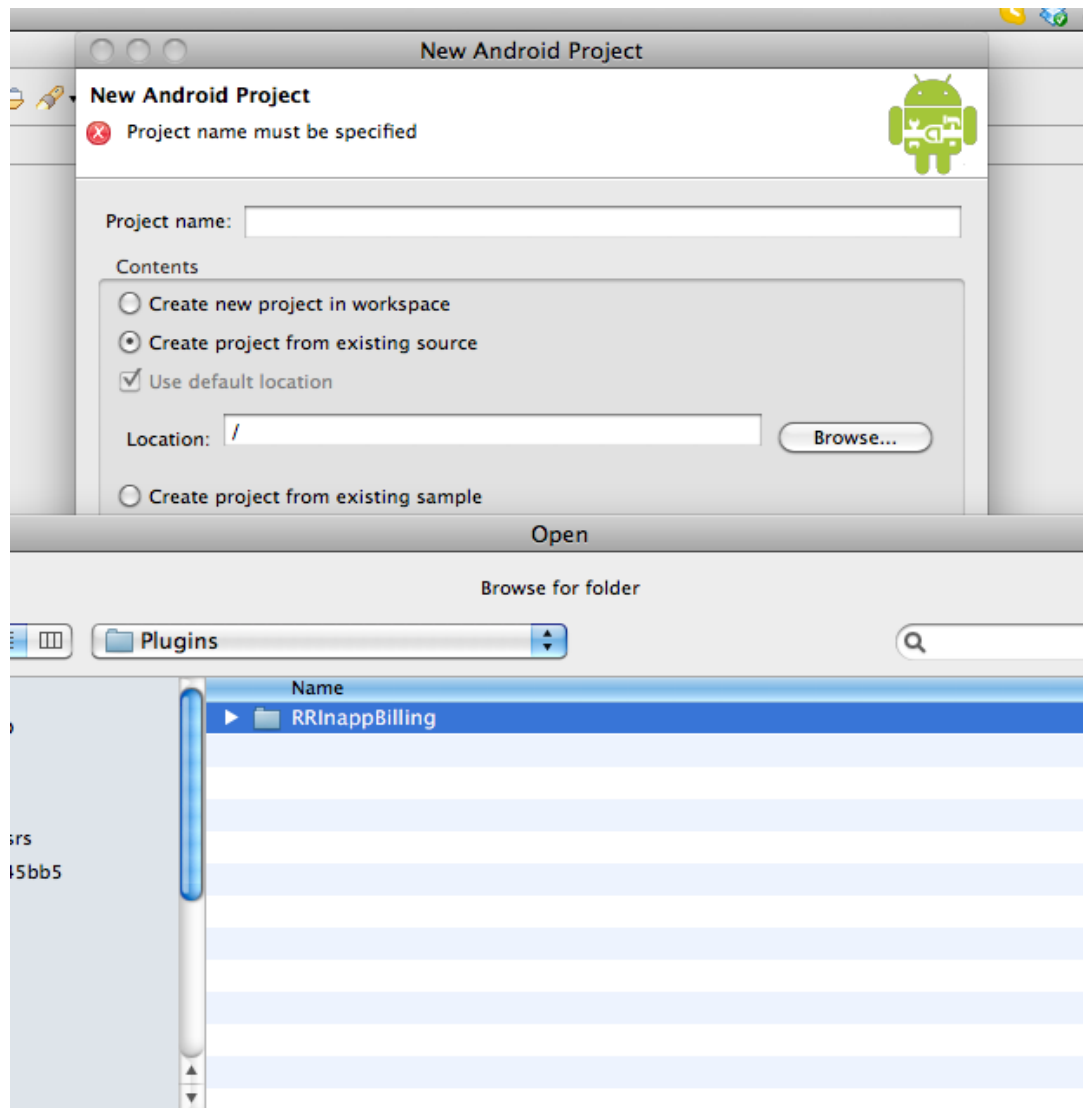
- f. You should have the corresponding SDK package previously installed from [here](#).
- g. Show the Android SDK path in Eclipse by going to Windows/Preference -> Android and update the path to where you installed the Android SDK and hit apply.



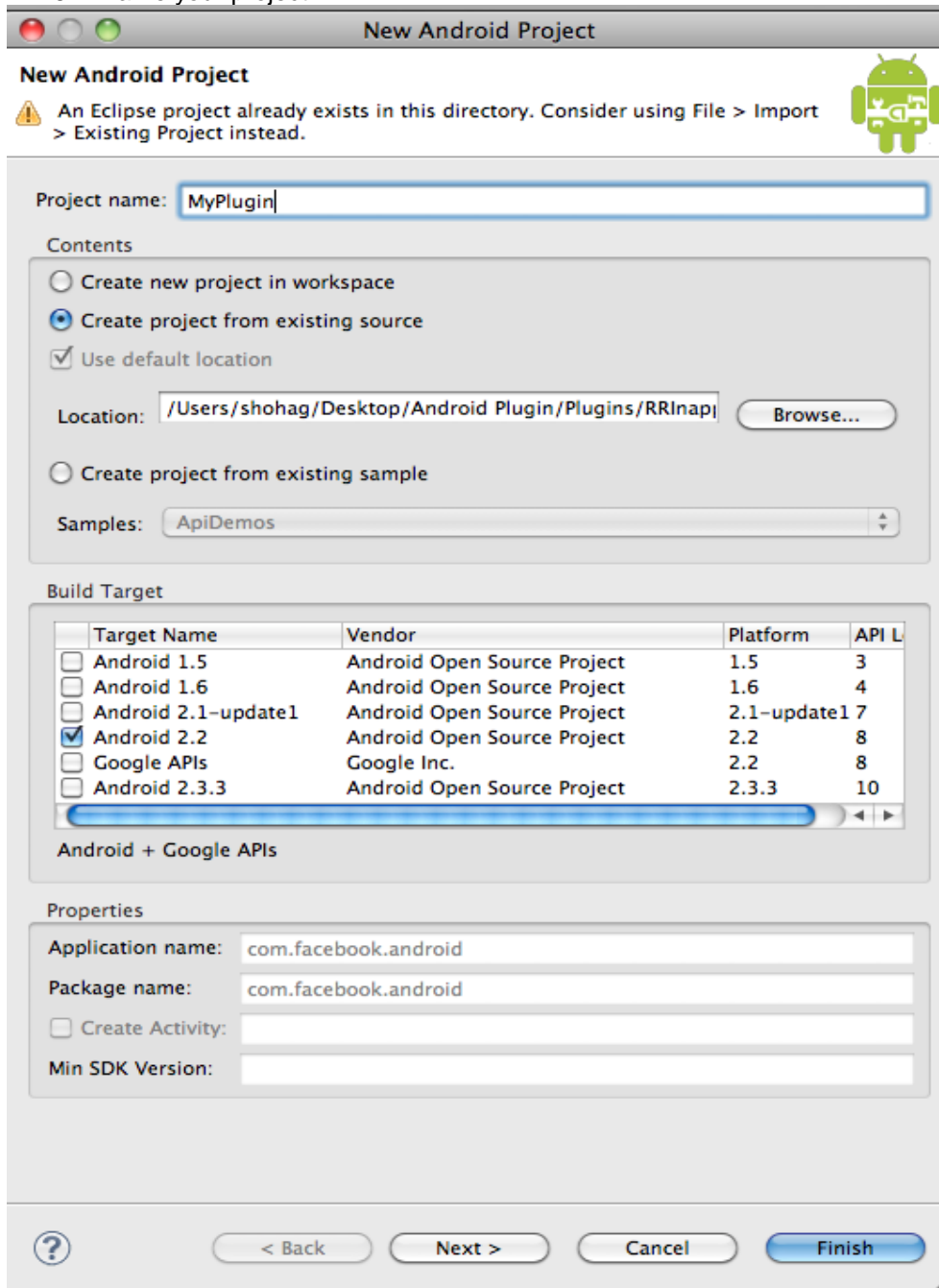
3. You need a Google Android developer account in order to develop for this platform. If you have not created an account yet, follow the steps here to create a [publisher account](#).

Setting Up the Plugin

1. After you imported the project from the Asset Store, you will notice a file called "RRInappBillingPlugin_Source.zip" located in the Source folder.
2. RRInappBillingPlugin_Source.zip is the Java plugin source code.
3. Start by unzipping the Java project "RRInappBillingPlugin_Source.zip" and place the output folder somewhere in your system (remove it from the Unity Project).
4. Open Eclipse, select File->New->Android Project.
5. Select "Create project from existing sources" option, browse and show the source path to the Java project folder "RRInappBilling" (that you just unzipped).



6. Name your project.



New Android Project

⚠ An Eclipse project already exists in this directory. Consider using File > Import > Existing Project instead.

Project name:

Contents

☐ Create new project in workspace

☒ Create project from existing source

☒ Use default location

Location:

☐ Create project from existing sample

Samples:

Build Target

Target Name	Vendor	Platform	API L
<input type="checkbox"/> Android 1.5	Android Open Source Project	1.5	3
<input type="checkbox"/> Android 1.6	Android Open Source Project	1.6	4
<input type="checkbox"/> Android 2.1-update1	Android Open Source Project	2.1-update1	7
<input checked="" type="checkbox"/> Android 2.2	Android Open Source Project	2.2	8
<input type="checkbox"/> Google APIs	Google Inc.	2.2	8
<input type="checkbox"/> Android 2.3.3	Android Open Source Project	2.3.3	10

Android + Google APIs

Properties

Application name:

Package name:

☐ Create Activity:

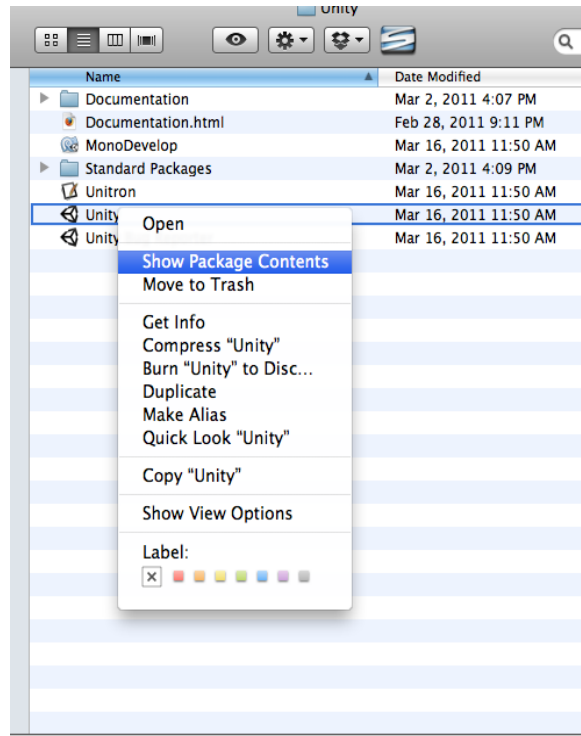
Min SDK Version:

7. Press Finish.

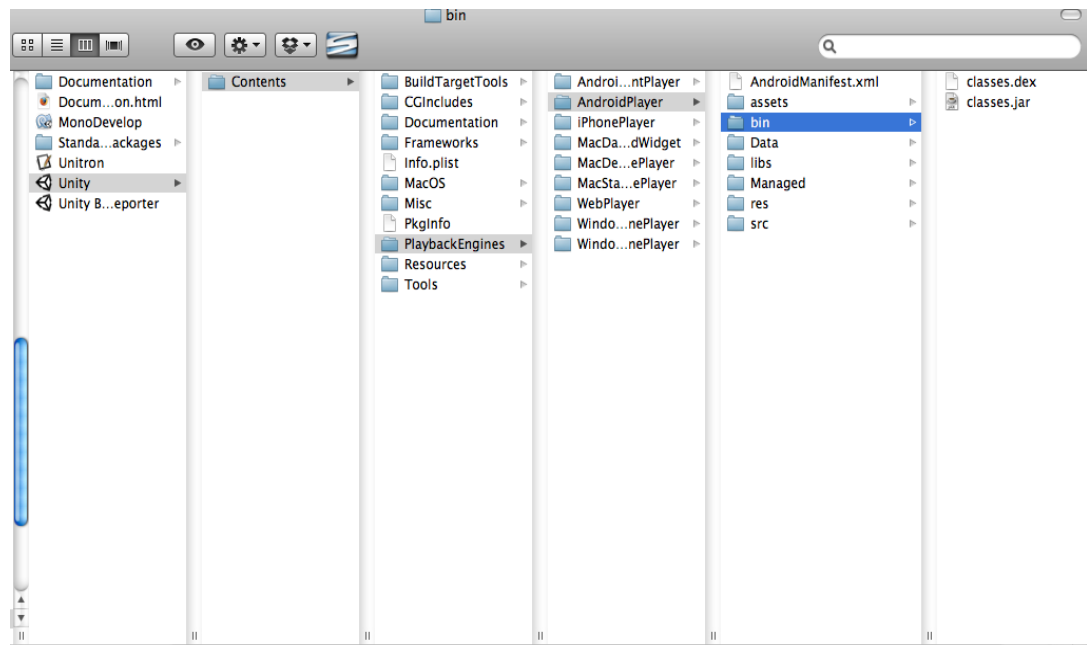
8. You may face an error message due to a Unity package missing. To resolve it, you need to add the Unity class JAR file in the library.

9. Check out the [Unity manual](#) for detail. But for an easy solution, follow these steps:

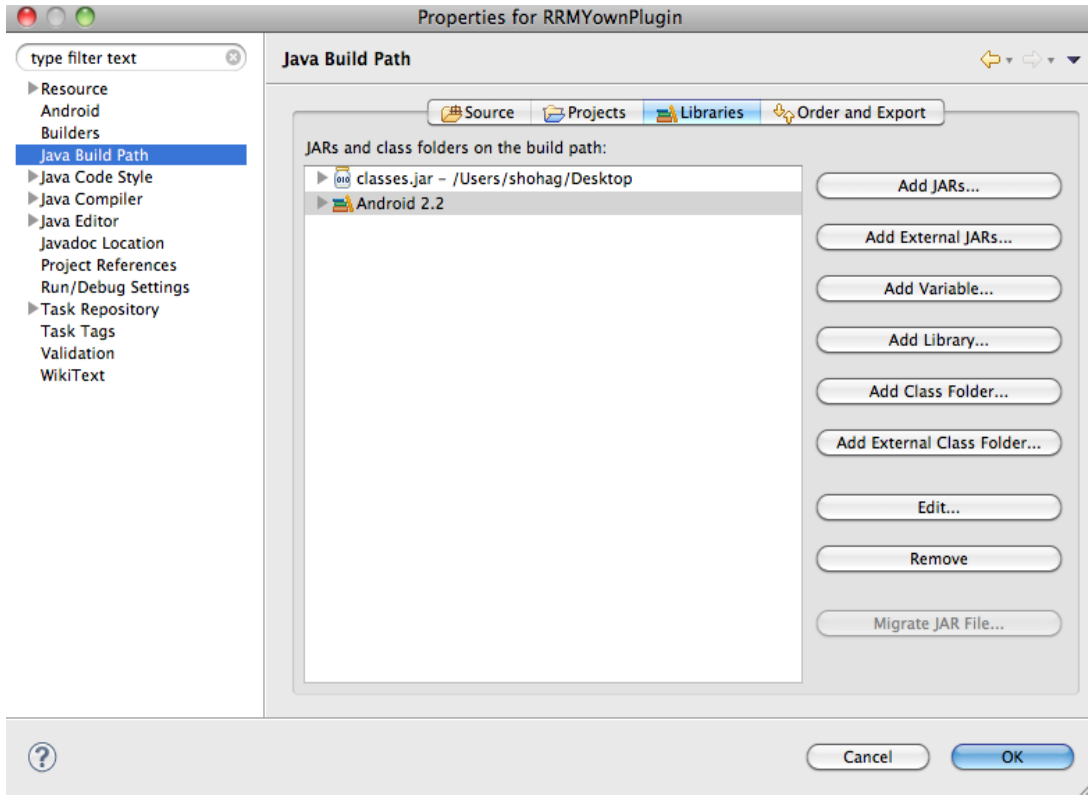
- a. Go to the directory where you installed Unity
 - Mac: "Application/Unity"
 - Windows: "C:\Program Files\Unity"
- b. If you are on a Mac, open the content of the package.



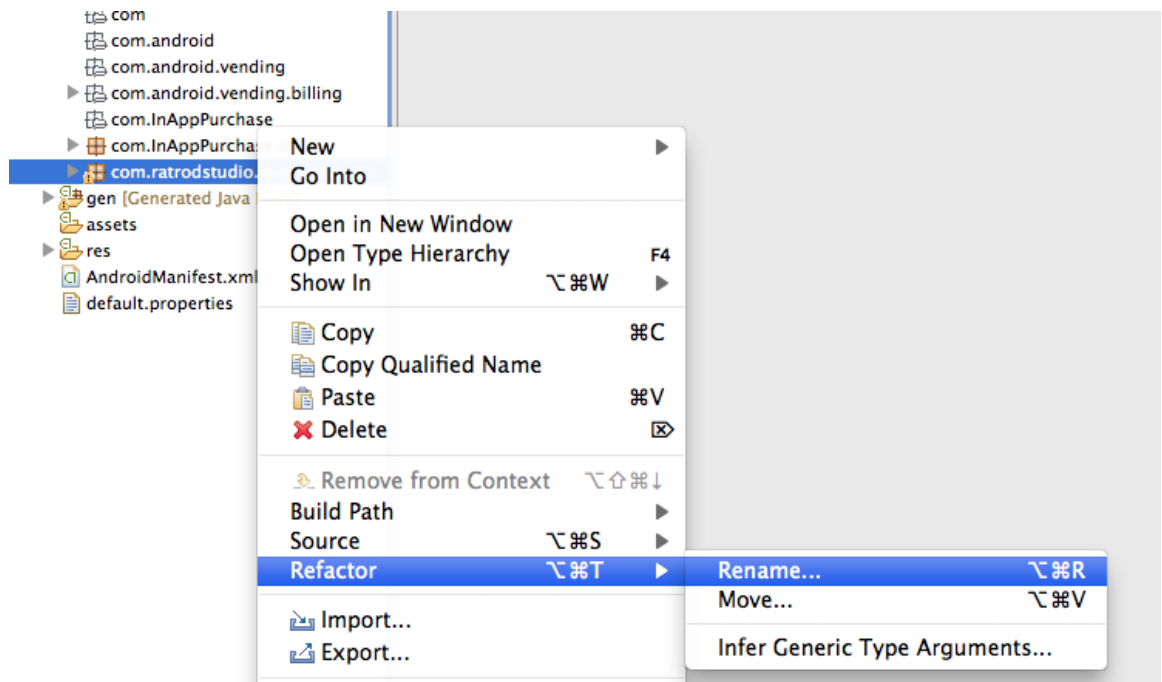
- c. If you are on Windows, go to this folder:
C:\Program files\Unity\Editor\Data\PlaybackEngines\androidplayer\bin\
- d. Copy the class.jar and place it to your desktop.

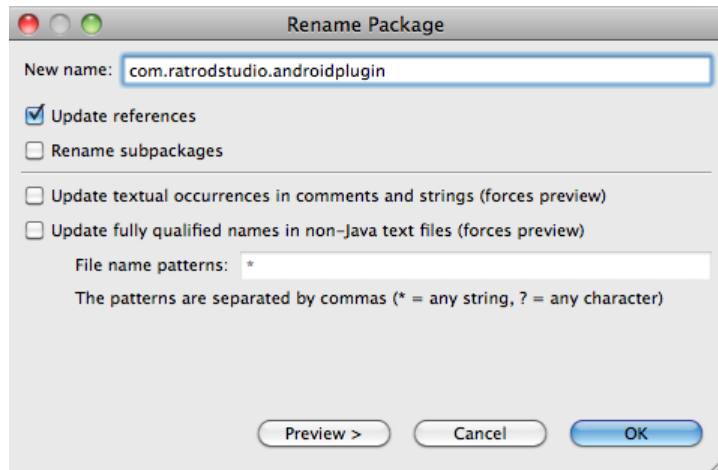


- e. Go to files/properties, then in “Java Build Path” add the class.jar by clicking “Add External JARs” from the Eclipse library.

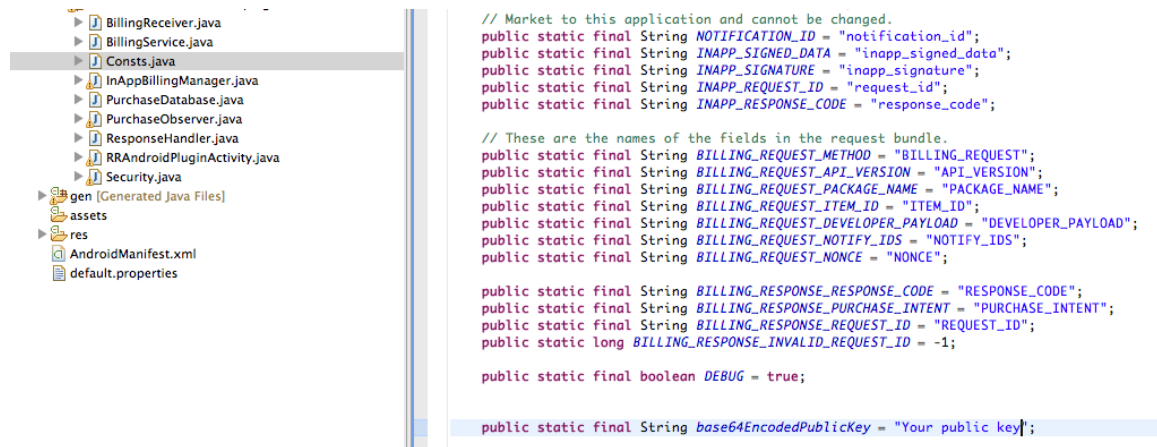


10. Rename your package for the proper name of your Unity application package.



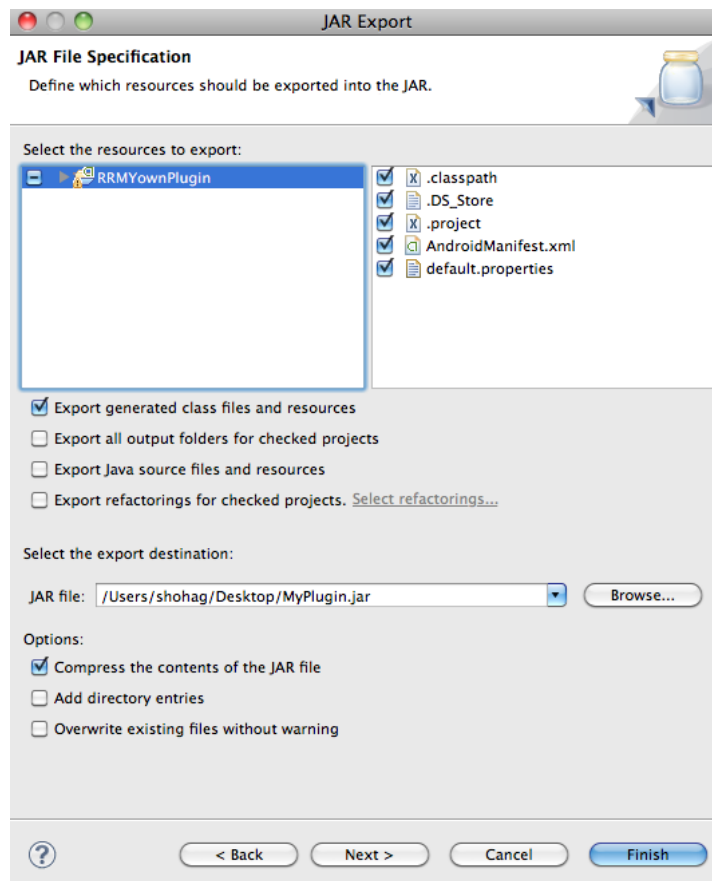


11. Place your *base64EncodedPublicKey* in the Consts.java file (found in the first step).

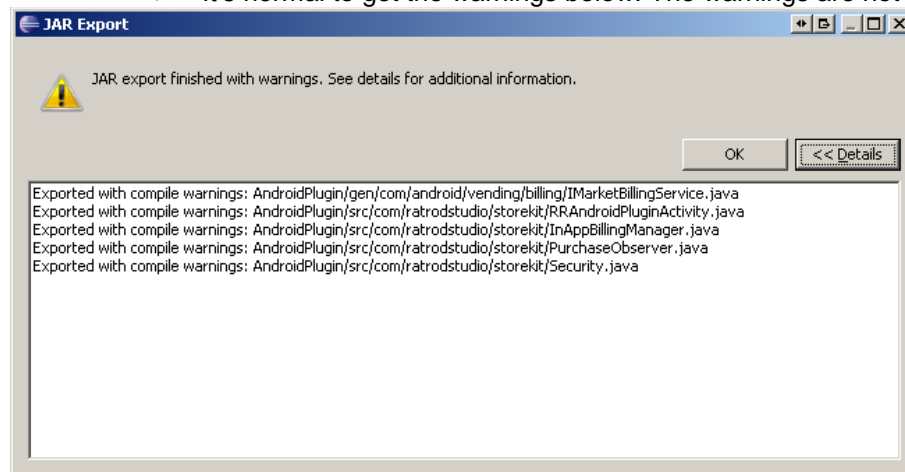


12. You are now done with configuring the plugin. To export the plugin, right click on the project and select Export -> Java -> JAR

- If you get an error when exporting, simply uncheck “.DS_Store” and “default.properties” and try again.
- Additional help can be found on this website:
<http://www.eclipse.org/forums/index.php?t=msg&goto=654757&S=16e78a5d653bc353cf3da659792b01d2>



- It's normal to get the warnings below. The warnings are not harmful.



13. Now using the Unity project you downloaded from the Asset Store, please your *.JAR plugin that you just created in your Unity project directory: Plugins/Android.
14. You will also need to include your "AndroidManifest.xml" in the same directory.
15. You only have to change the package name for the one that you used in Eclipse.
Example: "com.ratrodstudio.androidplugin"
16. If you are not sure of what you are doing, do not edit any other options of the AndroidManifest file.
17. Open the plugin -> Ratrodstudio -> BillingCONST.cs and replace the name of the root package name with the package name that you used while creating the JAR plugin with Eclipse.
18. Drag the "RRInappBillingCallback" prefab from RatRodStudio -> Prefab in your scene. By default, it will not destroy when another scene loads since this is important to keep.
19. To get the purchase call back you need to register to the listener. Always register the callback in OnEnable and unregister at OnDisable. Example:

```

void OnEnable(){
    RRInappBillingCallback.purchaseSuccessful += purchaseSuccessful;
    RRInappBillingCallback.purchaseFailed    += purchaseFailed;
    RRInappBillingCallback.purchaseCancelled += purchaseCancelled;
}

void OnDisable (){
    RRInappBillingCallback.purchaseSuccessful -= purchaseSuccessful;
    RRInappBillingCallback.purchaseFailed    -= purchaseFailed;
    RRInappBillingCallback.purchaseCancelled -= purchaseCancelled;
}

void purchaseSuccessful( string productIdentifier )
{
    Debug.Log( "purchased product: " + productIdentifier );
}

void purchaseFailed( string error )
{
    Debug.Log( "purchaseFailed: " + error );
}

void purchaseCancelled( string error )
{
    Debug.Log( "purchaseCancelled: " + error );
}

```

20. Check the example code in the sample Unity project.
21. For more details, visit the Android Developer website and read the full documentation available [here](#).

AndroidManifest Explanation

In-app billing relies on the Android Market application, which handles all communication between your application and the Android Market server. To use the Android Market application, your application must request the proper permission. You can do this by adding the **com.android.vending.BILLING** permission to your **AndroidManifest.xml** file. If your application does not declare the in-app billing permission, but attempts to send billing requests, Android Market will refuse the requests and respond with a **RESULT_DEVELOPER_ERROR** response code.

In addition to the billing permission, you need to declare the **BroadcastReceiver** that you will use to receive asynchronous response messages (broadcast intents) from Android Market, and you need to declare the Service that you will use to bind with the **IMarketBillingService** and send messages to Android Market. You must also declare intent filters for the **BroadcastReceiver** so that the Android system knows how to handle the broadcast intents that are sent from the Android Market application.

We have provided a complete manifest file with an example. It will look like:

//For billing permission

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.android.vending.BILLING" />
<uses-permission android:name="com.android.vending.CHECK_LICENSE" />
```

//For Service & response

```
<service android:name="BillingService" />

<receiver android:name="BillingReceiver">
    <intent-filter>
        <action android:name="com.android.vending.billing.IN_APP_NOTIFY" />
        <action android:name="com.android.vending.billing.RESPONSE_CODE" />
        <action android:name="com.android.vending.billing.PURCHASE_STATE_CHANGED" />
    </intent-filter>
</receiver>
```

Unity Example Project

There are two scenes in the Unity example project.

1. **InappBillingTestScene**: This is to demonstrate the In App Billing without setting any bundle. It's simply to test the functionality of [In App Billing](#).
2. **InappBillingExampleScene**: This scene has more details about how to setup your In App Billing for your project. Generally you will give user points (monetization) or let them unlock game content after they completed the transaction. In this sample application we considered that you have 3 bundles that already have been created in your admin panel and their identifiers are as followed:
 - a. pts_100
 - b. pts_200
 - c. pts_300

When the user is buying the item "pts_100", you will then be allocating them 100 Points that are added to his profile. We used that functionality and call back function for this example.

We are going to explain how you can create your own bundle and test them.

1. Open the **InappBillingExampleScene** scene.

2. The scene includes the **SampleInApp** script linked to the Main Camera GameObject. Open the "SampleInApp" script.
3. On Start, we placed a code segment **RRInappBillingPluginKit.InitInAppBillingSupport()**; which is important. When you do your **In App Billing** in your specific scene, you have to call this function at the start of your In App Billing controller script.
4. We have registered and unregistered all call back at OnEnable and OnDisable.
5. There are 3 buttons OnGUI to do 3 bundles purchase action.
6. If you press "pts_100", then it will do the purchase and will give a call back to ***purchaseSuccessful***. If it's successful, we are then allocating the corresponding points to the user profile.
7. Using a similar method, you can do anything including giving points, unlocking features or giving additional content to your user.
8. To test the sample scene, you will need to create those 3 bundle IDs using the developer panel. Follow the next section of this tutorial to learn how to create them.
9. You may need to change the scene level index from File -> Build Settings to set the test scene that you would like to start.
10. Also in order to test in-app billing, you have to make sure that your application is signed. In your Player settings, under "Publishing settings". Create a Keystore key with a password. Then also create an alias and password. Visit the Unity forums for more details: <http://forum.unity3d.com/threads/62137-Android-Signing>
11. Then copy your Public key in the "Android Market Licensing (LVL) field.

Creating a Bundle Item (In-App)

1. Log in your Android Development panel.
2. First you need to create an application in order to setup your In-app bundle. Press on the "Upload Application" button.
3. You will need to provide an APK file from your project. Make sure to use the manifest included with the sample Unity project or add those specific permission to your own manifest:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="com.android.vending.BILLING" />
<uses-permission android:name="com.android.vending.CHECK_LICENSE" />
```

4. Within Unity, go to "Files", then Build Settings, and then press on "Build".
5. Select where you want to save your APK file.
6. Select this APK to be uploaded as your new application.
7. Include at least 2 screenshots, they can simply be placeholders.
8. You will need to include a 512x512 icon (required).
9. Then fill up the details for your game title and description. You will also need to select the application type and a category.
10. Set the proper Publishing options and contact information. Then checkmark the Consent boxes and press "save".
11. From the development panel home page, you will then see your application. Right under, click on "In-app Products".
12. You will then see a button on the bottom right "Add in-app product", click on it and a new form will appear.

Create New In-app Product

Hockey Fight Pro



Version: 1.20

Application State: ✔ Published

Questions about in-app products? Click [here](#) to learn more!

In-app Product ID

0 characters (100 max)

Purchase Type

☒ Managed per user account ☐ Unmanaged

Please note that purchase type cannot be modified in the future.

Publishing State

☐ Unpublished

Language

[add language](#)

| *English (en) |

The default language is inherited from the owning application. Click [here](#) to change default language!

Title

0 characters (best viewed under 25, max 55)

Description

0 characters (80 max)

Price

CAD

Auto Fill

Automatically populate the fields with a one-time conversion to local currencies from the default price with the current exchange rate

13. Name your in-app Product ID. This will be your bundle ID name that you will be using in Unity.
14. Select the purchase type: managed is for a single purchase while unmanaged can be purchased multiple times by the user. Make sure to select the right type as you cannot change this option later.
15. Add a title, description and a price. Then you can save your item.
16. Once all your bundle items are created, you will need to mark them as "Published" in order to test the in-app purchase.
*More details can be found on the Android developer website:
<http://developer.android.com/guide/market/billing/index.html>
17. When you are ready to test your in-app items, follow the instructions located on the Android developer website:
http://developer.android.com/guide/market/billing/billing_testing.html

SUPPORT

You are now ready to add in-app billing within your Unity game or application.

If you have any questions or need support contact us at support@ratrodstudio.com. We will also create a thread on the Unity forums in the Android section.

Enjoy!