Android In-App Billing (Google Play) Adobe AIR Native Extension

http://www.milkmangames.com

For support, contact support@milkmangames.com

Before you begin:

The Android Billing Extension requires Adobe AIR 3.5 or higher. You can download the latest version of AIR at http://www.adobe.com/devnet/air/air-sdk-download.html.

To View full AS3 documentation, see 'docs/as3docs/index.html'.
Review 'example/IABExample.as' for a sample application.

Note that IABExample.as is a Document Class. If you're a Flash Professional user and don't know how to use a document class, see the appendix "Using the Example Class in Flash CS6" at the end of this guide.

If you are migrating from v1.0.3 of the extension, please review this guide and see releasenotes.txt for details of changes.

1. Include the Library

Start by creating a new AIR for Android project and adding the native extension.

In Flash Professional CS6 or Higher:

1. Create a new Android project
2. Choose File>Publish Settings...
3. Select the wrench icon next to 'Script' for 'ActionScript Settings'
4. Select the Library Path tab.
5. Click 'Browse for Native Extension (ANE) File' and select the com.milkmangames.extensions.AndroidIAB.ane file. Press OK.
6. Select the wrench icon next to 'Target' for Player Settings
7. Select the 'Permissoins' tab, and enable 'INTERNET'.
8. Check the 'Manually manage permissions and manifest additions for this app' box.
9. Press OK

In Flash Builder 4.6 or Higher:

1. Go to *Project Properties*
2. Select *Native Extensions* under *Actionscript Build Path*
3. Choose *Add ANE...* and navigate to the com.milkmangames.extensions.AndroidIAB.ane file
4. Select *Actionscript Build Packaging > Google Android*
5. Select the *Native Extensions* tab, and click the '*Package*' checkbox next to the extension

2. Update Your Application Descriptor

For Android Billing to work, changes are required to the application XML file for your app. Modify the xml file created by your IDE with the following changes (if you're a Flash Professional user, make sure you've followed the steps above for 'Include the Library in Flash Professional', otherwise Flash might undo your changes as you make

them):

1. Set your AIR SDK to 3.5 in the app descriptor file:

```
<application xmlns="http://ns.adobe.com/air/application/3.5">
```

2. Include a link to the extension in the descriptor:

```
<extensions>
<extensionID>com.milkmangames.extensions.AndroidIAB</extensionID>
</extensions>
```

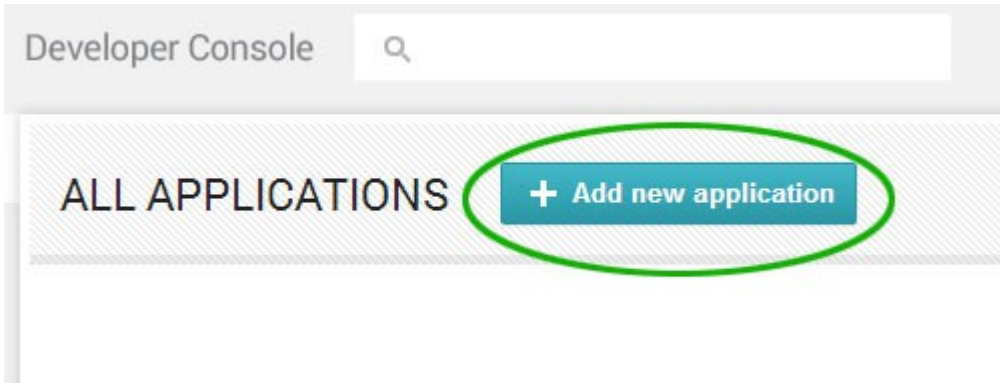
3. Update your Android Manifest Additions in the <android> xml element, by setting the targetSdkVersionand minSdkVersion, and adding the com.android.vending.BILLING permission:

```
<android>
 <manifestAdditions><![CDATA[
  <manifest android:installLocation="auto">
      <uses-sdk android:targetSdkVersion="12"/>
      <uses-sdk android:minSdkVersion="8"/>

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


</manifest>
]]></manifestAdditions>
</android>
```

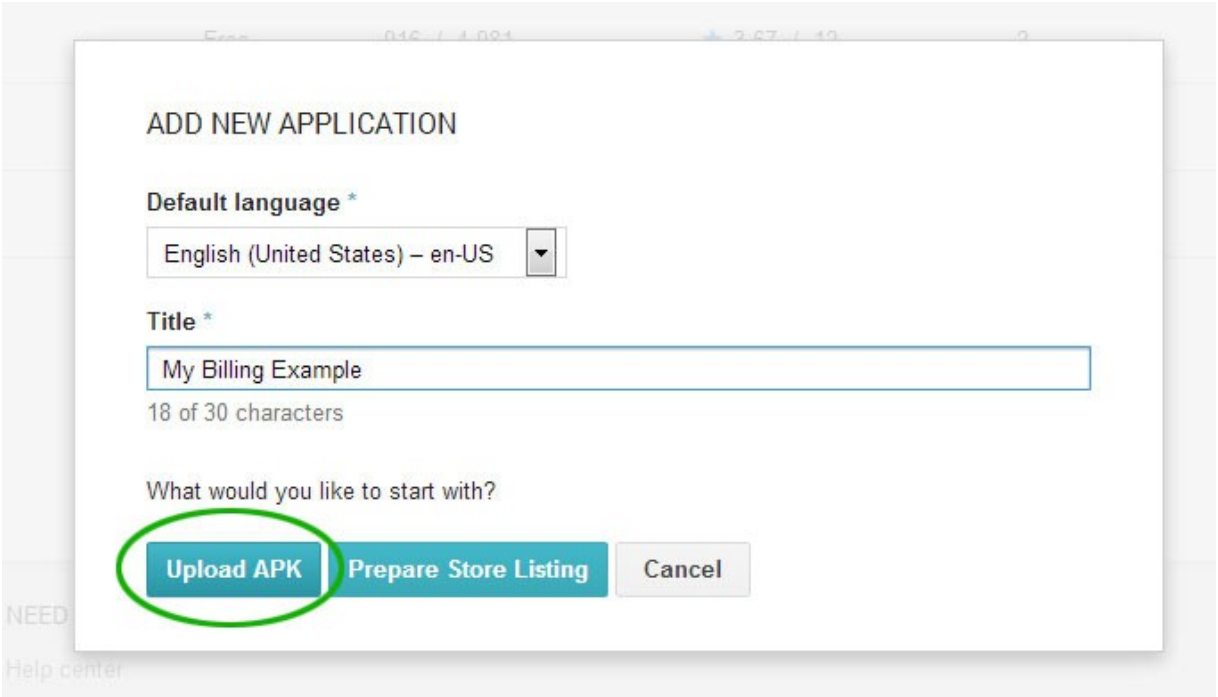
3. Build the Application and Upload the Draft to Google Play

1. Save your work, and build an APK file from your application. Although there is no code in the application yet, it's a good idea to build it now, as Google Play requires you to upload an APK before In-App Billing will work. Make sure you are happy with the application ID, title, p12 certificate, and version number- if these values change, you will need to upload a new draft – and it can take anywhere from minutes to hours before Google processes the APK file!

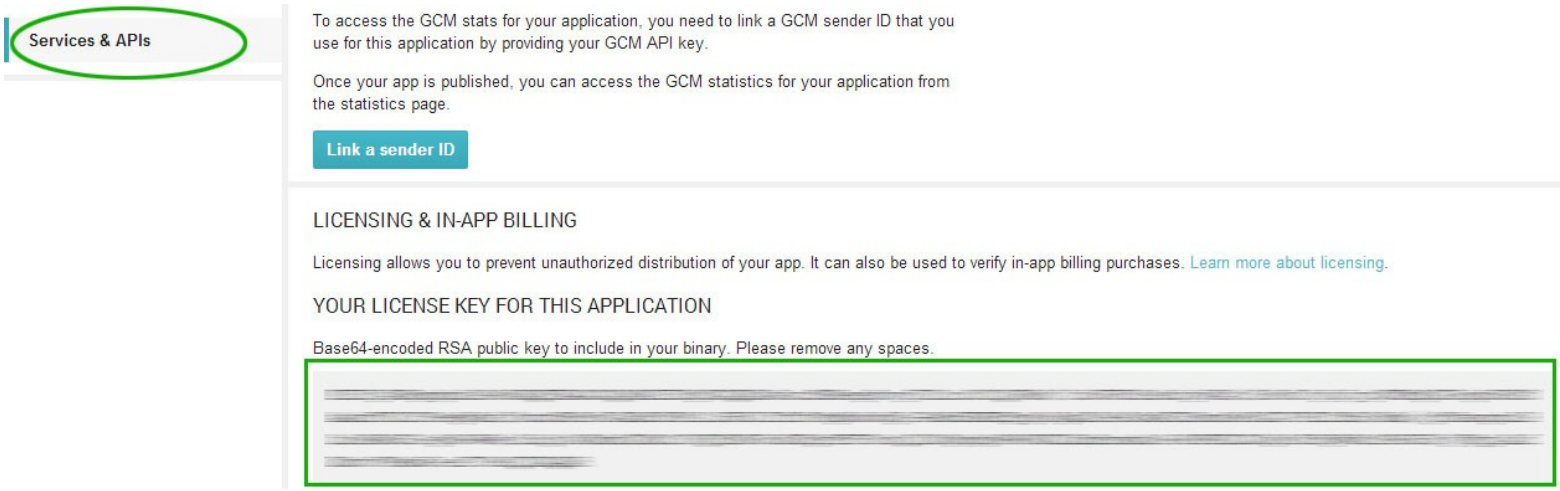
   Important Note for Flash Builder Users: Flash Builder 4.6 will change your application ID without your permission every time you switch between a 'release' and 'debug' build. Because the IDs of the uploaded APK and the APK you test with must match, you will either need to commit to using only one type of build during testing, or uploading a new build and waiting a few hours every time you need to change targets. One solution to mitigate this is to build only for debug until you're ready to publish app, then rebuild and reupload a new draft at that time. In Flash Builder 4.7, this behavior can be overriden by removing the '.debug' from the 'AppId' box under Run/Debug Configurations Dialog Box (Project Properties > Build Packaging > Run / Debug Settings > Edit.)
2. Go to http://play.google.com/apps/publish/ to access the Google Play Developer Panel.
3. Choose '+Add new application' to define your new app.



4. Enter a title and language for your app, then select 'Upload APK'.

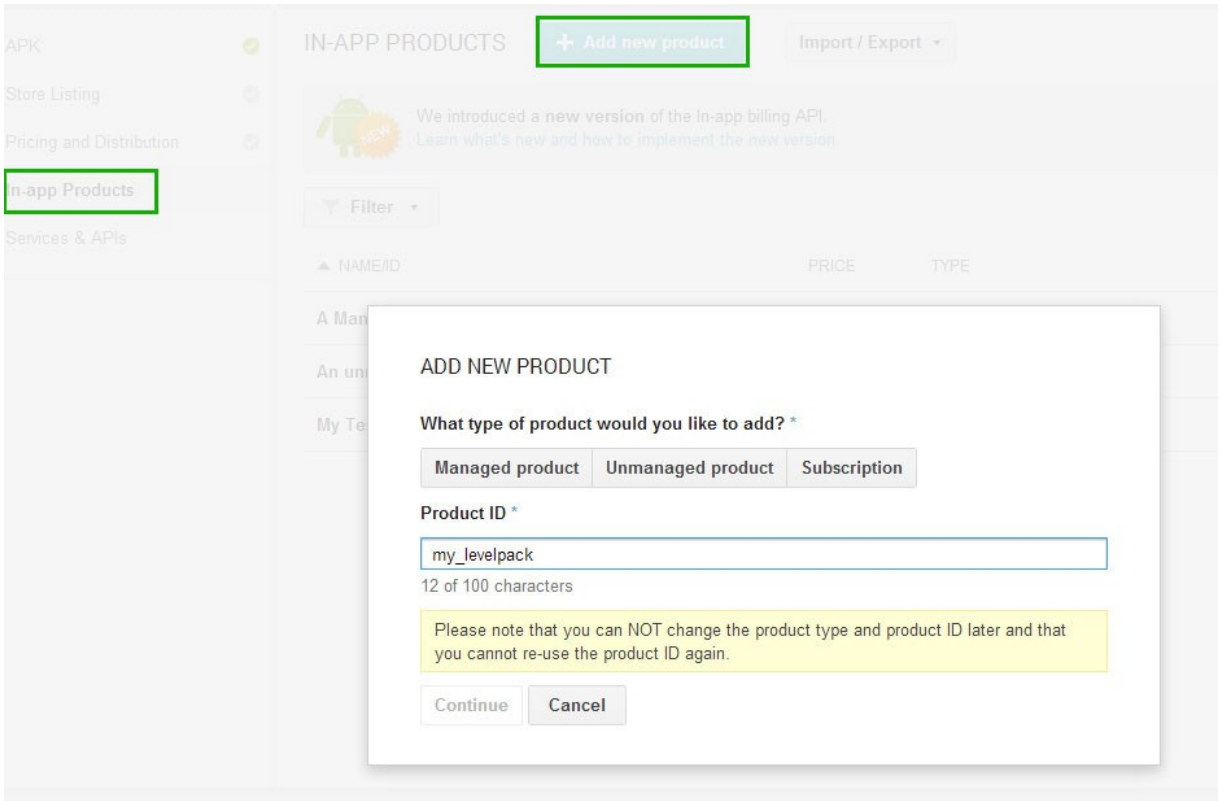


5. Select 'Upload Your First APK', and select the APK file you created in step 1.
6. From the 'Services & APIs' tab, copy and paste your License Key, and save it for future reference.

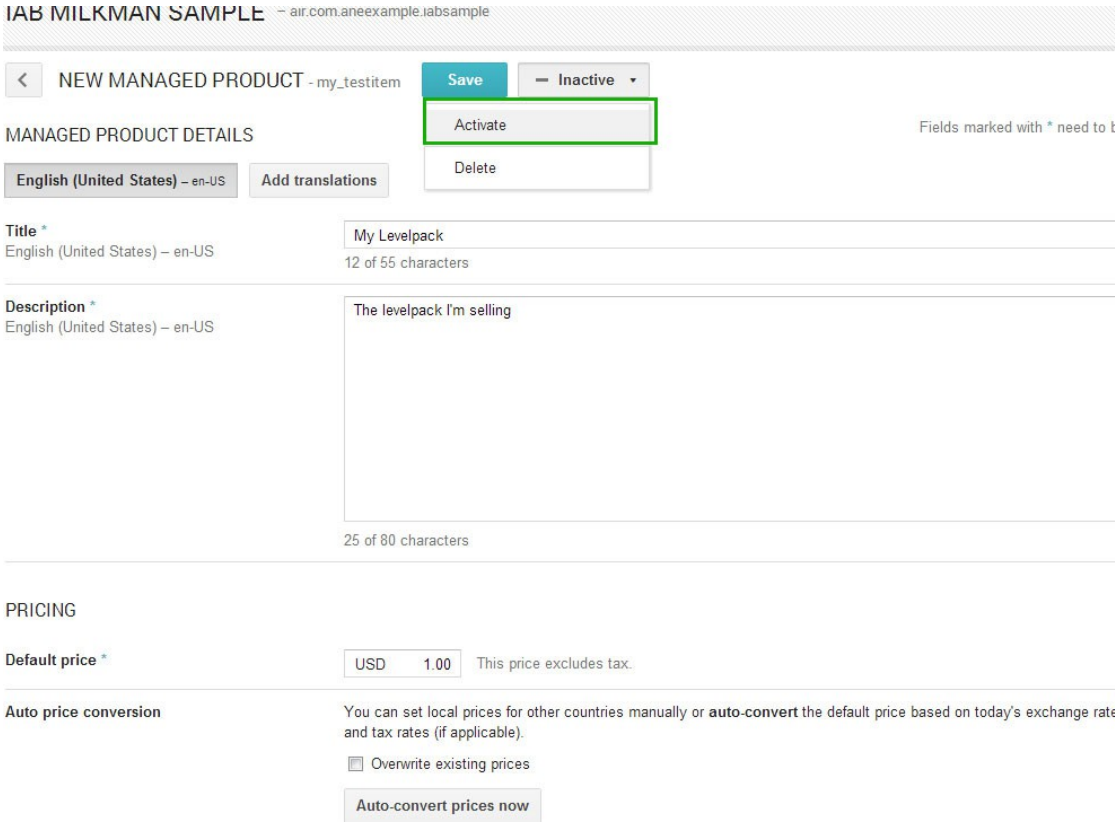


7. From the 'In-App Products' tab, select 'Add New Product', and enter a name for the first in-app item you'd like to sell.

   If the Item is a Subscription, choose the Subscription button. Otherwise, choose 'Managed'. (Although you may still select 'Unmanaged', the option is deprecated as all Google Play Billing items are now managed.)



8. Enter the Product ID Title, Description, pricing and language information for your product. Then choose Save, and change the status to Active by choosing the 'Activate' option.



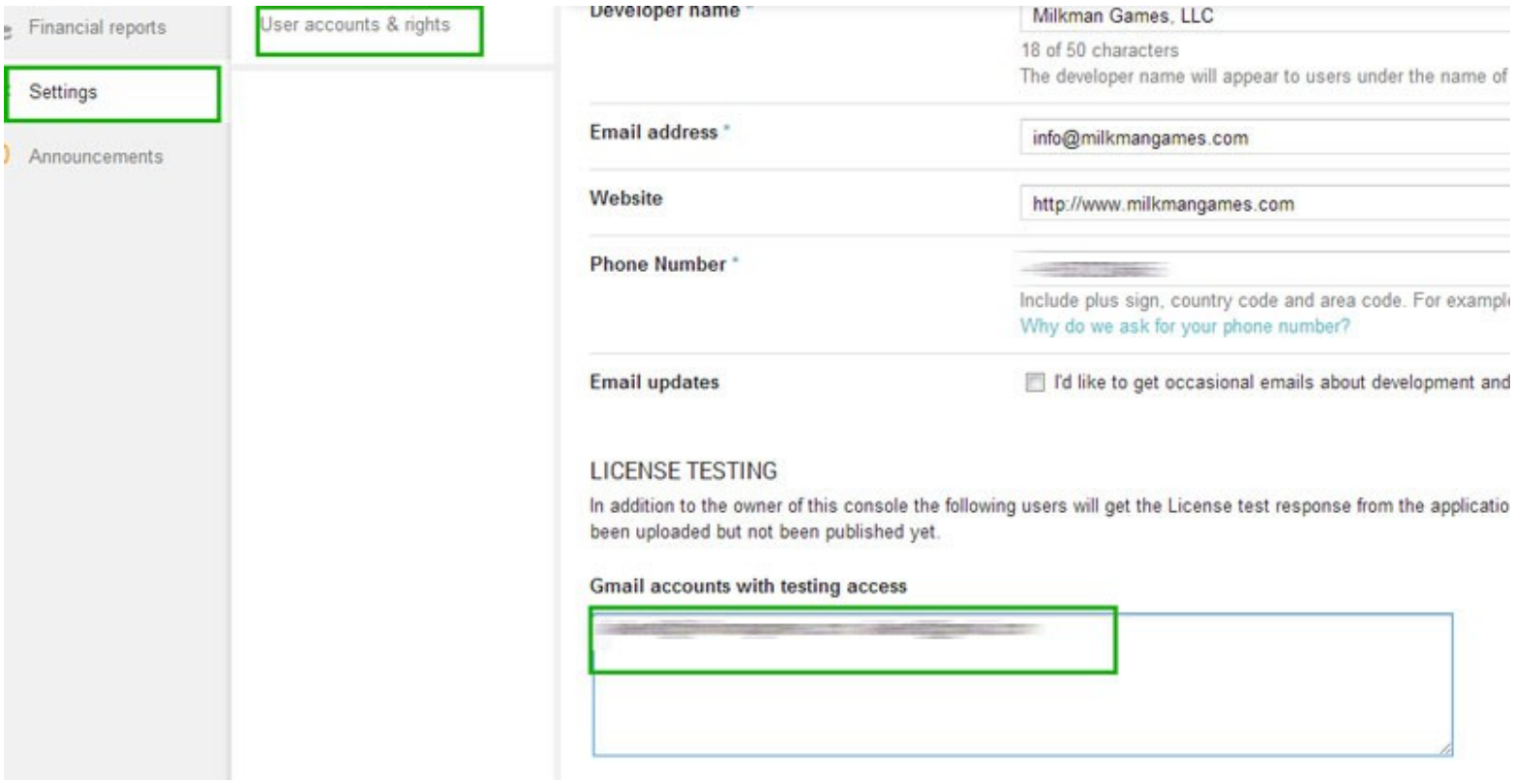
9. Repeat the process for each product you'd like to sell. If you'd like to create products that match with the example code, create managed items called 'my_levelpack' and 'my_spell', and a subscription called 'my_subscription'.

4. Prepare for Device Testing

In order to test In-App Billing, you'll need to update Google Play with your test account address, and possibly make changs to the registered account on your phone.

When you first setup your Android device, you entered a Google/ Gmail Account to use with Google Play – the 'primary account' on your phone. This gmail address needs to be added to the Google Play developre panel in order to test In-app Billing. To add the test email to Google Play:

1. Select the 'Settings' tab from the left.
2. Select 'Account Details'
3. Under 'Gmail accounts with testing access', enter the gmail address that is the primary account on the device. You may enter multiple email addresses separated by commas if you wish.



4. As part of a continuing effort to make Android development as annoying as possible, Google does not let you 'buy things from yourself'. In other words, if the primary gmail account is the same as the one you used to sign up for Google Play, you'll need to change the primary account on the device to something else.

   Although many Android devices will let you remove and add accounts, this isn't good enough for Google Play testing- you'll have to factory reset the device. See the documentation for your device's make and model if you need to factory reset the phone.

5. Set the API and Connect to the Billing Service

While you wait for Google to process your APK, you can start adding code to your application.

1. Import the API Classes:

```
import com.milkmangames.nativeextensions.android.*;
import com.milkmangames.nativeextensions.android.events.*;
```

2. Initialize the API by calling AndroidIAB.create(). You can check the AndroidIAB.isSupported() method first, to ensure the current platform is Android and not an unsupported platform (like iOS or Windows.).

   IMPORTANT: This initialization must occur in the entry point of your application. If you are using Flex/MXML, this means you must call create() in the initialize() event of the main class (not a View constructor or createComplete callback.) If you are using pure ActionScript, you must call create() inside the Constructor function of the Document Class. If you are using timeline code in Flash, call this once at the beginning of Frame 1.

```
// check if the current platform supports android extensions
if (AndroidIAB.isSupported())
```

```
{
	AndroidIAB.create();
}
```

3.

4. Call the AndroidIAB.androidIAB.startBillingService("YOUR_PUBLIC_KEY_HERE") method to begin interactions with Android's In-App Billing server. Use the public key you retrieved earlier in the section "Application and Upload the Draft to Google Play", Step 6.

   If the current device will not support In-App Billing (because it doesn't have Google Play, for instance), you'll receive the AndroidBillingEvent.SERVICE_NOT_SUPPORTED event. Otherwise, when the service is ready, you'll receive the AndroidBillingEvent.SERVICE_READY event; once that happens, you may begin calling the other API methods to make purchases.

```
// listeners for billing service startup
AndroidIAB.androidIAB.addEventListener(AndroidBillingEvent.SERVICE_READY,onReady);
AndroidIAB.androidIAB.addEventListener(AndroidBillingEvent.SERVICE_NOT_SUPPORTED, onUnsupported);

// start the service
AndroidIAB.androidIAB.startBillingService("YOUR_PUBLIC_KEY_HERE");

private function onReady(e:AndroidBillingEvent):void
{
	trace("service now ready- you can now make purchases.");
}

private function onUnsupported(e:AndroidBillingEvent):void
{
	 trace("sorry, in app billing won't work on this phone!");
}
```

6. Loading the Player's Inventory

Once you've connected to Google Play, you can request information about the items that the user has previously purchased. It's a good idea to do this as soon as your app starts, in case the user has installed the app on a new device, had an item refunded, or their inventory has otherwise changed.

1. Add event listeners for AndroidBillingEvent.INVENTORY_LOADED and AndroidBillingErrorEvent.INVENTORY_LOAD_FAILED. When the INVENTORY_LOADED event is dispatched, its 'purchases' property will be populated with a vector array of AndroidPurchase objects, representing the purchases the user has previously made.

   The LOAD_INVENTORY_FAILED is dispatched when an error occurs attempting to load the inventory. Because it is an error event, you should listen for it even if you don't plan to do anything about the error, to avoid an uncaught exception in your code:

```
// listen for inventory events
AndroidIAB.androidIAB.addEventListener(AndroidBillingEvent.INVENTORY_LOADED, onInventoryLoaded);
AndroidIAB.androidIAB.addEventListener(AndroidBillingErrorEvent.LOAD_INVENTORY_FAILED,
onInventoryFailed);

function onInventoryLoaded(e:AndroidBillingEvent):void
{
	for each(var purchase:AndroidPurchase in e.purchases)
	{
	trace("You own the item:"+purchase.itemId);
	// this is where you'd update the state of your app to reflect ownership of the item
	}
}
```

```
function onInventoryFailed(e:AndroidBillingErrorEvent):void
{
	trace("Something went wrong loading inventory: "+e.text);
}
```

2. To start the inventory request, call AndroidIAB.androidIAB.loadPlayerInventory():

```
// load the player's current inventory
AndroidIAB.androidIAB.loadPlayerInventory();
```

7. Loading Details About Purchaseable Items

In addition to loading the list of items the player already owns, you may wish to load details about items your app offers for sale, such as title, description, and price.

1. Add event listeners for AndroidBillingEvent.ITEM_DETAILS_LOADED and AndroidBillingErrorEvent.ITEM_DETAILS_FAILED. When the ITEM_DETAILS_LOADED event is dispatched, its 'itemDetails' property will be populated with a vector array of AndroidItemDetails objects, representing the details about items available in your app's store.

   The ITEM_DETAILS_FAILED is dispatched when an error occurs attempting to load the item details. Because it is an error event, you should listen for it even if you don't plan to do anything about the error, to avoid an uncaught exception in your code:

```
// listen for inventory events
AndroidIAB.androidIAB.addEventListener(AndroidBillingEvent.ITEM_DETAILS_LOADED, onItemDetails);
AndroidIAB.androidIAB.addEventListener(AndroidBillingErrorEvent.ITEM_DETAILS_FAILED,
onDetailsFailed);

function onItemDetails(e:AndroidBillingEvent):void
{
	for each(var item:AndroidItemDetails in e.itemDetails)
	{
		trace("item id:"+item.itemId);
		trace("title:"+item.title);
		trace("description:"+item.description);
		trace("price:"+item.price);
	}
}

function onDetailsFailed(e:AndroidBillingErrorEvent):void
{
	trace("Something went wrong loading details: "+e.text);
}
```

2. To start the details request, pass a Vector array of item ids to the loadItemDetails() function. (The IDs you choose will need to be defined later in the Google Play control panel after you've uploaded your APK):

```
// load info about the items for sale
var allItemIds:Vector.<String>=new <String>["my_spell", "my_subscription", "my_levelpack"];
AndroidIAB.androidIAB.loadItemDetails(allItemIds);
```

8. Making Purchases

1. First, add event listeners for AndroidBillingEvent.PURCHASE_SUCCEEDED and AndroidBillingErrorEvent.PURCHASE_FAILED:

```
// listen for purchase events
AndroidIAB.androidIAB.addEventListener(AndroidBillingEvent.PURCHASE_SUCCEEDED, onPurchaseSuccess);
AndroidIAB.androidIAB.addEventListener(AndroidBillingErrorEvent.PURCHASE_FAILED, onPurchaseFailed);
```

2. When PURCHASE_SUCCEEDED is dispatched, its 'purchases' property will be a vector array containing an AndroidPurchase object for the item that was bought. When this event fires, you can call loadPlayerInventory() again to update the list of items the player now owns. If something goes wrong, PURCHASE_FAILED will be dispatched instead:

```
function onPurchaseSuccess(e:AndroidBillingEvent):void
{
        var purchase:AndroidPurchase=e.purchases[0];
        trace("you purchased the item "+purchase.itemId);
       AndroidIAB.androidIAB.loadPlayerInventory();
}

function onPurchaseFailed(e:AndroidBillingErrorEvent):void
{
        trace("Something went wrong with the purchase of "+e.itemId+": "+e.text);
}
```

3. The extension provides a number of functions for testing purchase event responses with 'fake' items that you have not registered in Google Play. The following example illustrates how to initiate such a 'fake' purchase:

```
// this will start a purchase dialog and on success callback the PURCHASE_SUCCEEDED event
AndroidIAB.androidIAB.testPurchaseItemSuccess();

// this will start a purchase dialog then cause a PURCHASE_FAILED event
AndroidIAB.androidIAB.testPurchaseItemCancelled();
```

4. To start the purchase process for a real item, call AndroidIAB.androidIAB.purchaseItem(itemId), where itemId is one of the products you registered on the Google Play Developer panel.

   Note that for the purchase to work on an unpublished app, the test account must be set up properly as described above in Section 4, “Prepare for Device Testing.” As long as you are using a a test account, your credit card will not be charged- make sure that the purchase window includes the phrase “You will not be charged.”

   To initiate the purchase of a real item:

```
AndroidIAB.androidIAB.purchaseItem("your_itemid");
```

5. If your item is a Subscription Item, use the purchaseSubscriptionItem(itemId) call instead. Because not all phones will support subscriptions, you should check the areSubscritpionsSupported() method first:

```
if(AndroidIAB.androidIAB.areSubscriptionsSupported())
{
```

```
        AndroidIAB.androidIAB.purchaseSubscriptionItem("your_itemid");
    }
    else
    {
        trace("this device needs a Google Play Update to support subscriptions.");
    }
```

9. Consuming Items

Some in-app purchases may be treated as permanent upgrades, such as pack of new levels added to the game. For these types of items, the user purchases them one time, and they are then treated as permanently owned.

Some games or apps may benefit from a 'consumable' type of item: something that can be purchased, used up, and purchased again. To implement such an item, implement the consumption flow as described below:

1. Add event listeners for AndroidBillingEvent.CONSUME_SUCCEEDED and AndroidBillingErrorEvent.CONSUME_FAILED. When the CONSUME_SUCCEEDED event is dispatched, the item has been removed from the player's inventory and can be purchased again, and you should call loadPlayerInventory() again to update the state of the player's inventory.

   The CONSUME_FAILED event is dispatched when an error occurs attempting to consume an item. Because it is an error event, you should listen for it even if you don't plan to do anything about the error, to avoid an uncaught exception in your code:

```
// listen for consumption events
AndroidIAB.androidIAB.addEventListener(AndroidBillingEvent.CONSUME_SUCCEEDED, onItemConsumed);
AndroidIAB.androidIAB.addEventListener(AndroidBillingErrorEvent.CONSUME_FAILED, onConsumeFailed);

function onItemConsumed(e:AndroidBillingEvent):void
{
        trace("you consumed the item "+e.itemId);
        AndroidIAB.androidIAB.loadPlayerInventory();
}

function onConsumeFailed(e:AndroidBillingErrorEvent):void
{
        trace("Something went wrong consuming "+e.itemId+": "+e.text);
}
```

2. To start the consume request, pass the item's id to the consumeItem() method. (Note that you cannot consume item unless you've previously called loadPlayerInventory() and the item is present):

```
// consume an item
AndroidIAB.androidIAB.consumeItem("my_spell");
```

10. Troubleshooting and FAQ

"Why I am not receiving events / why am I getting error events?"

- Make sure you've saved a draft of the application to the Google Play Developer Website.
- Make sure you've updated your application descriptor with the appropriate manifestAdditions. If you're using Flash, make sure that you've ticked the box to Mange Permissions and Manifest Additions Manually.
- Make sure the Google email tied to your device as described in Prepare for Device Testing.
- If you're using the IABSample application as is, make sure that you've created the in app products examples it uses: a managed product called 'my_levelpack' and an unmanaged one called 'my_spell'. Make sure you've put your own public key into the code.

- If you're using a subscription product, make sure you're using the purchaseSubscriptionItem() method.
- Google can take anywhere from an hour to a day to update its servers after you upload a new draft APK or change a product; wait one day and try again.

"I refunded an item in my merchant account, but it's still showing up in the player inventory."

- Make sure you've called loadPlayerInventory() to update the inventory state locally.
- Recently, we've seen refunds take between an hour and a few days to be reflected in the inventory. A Google search on the topic indicates that it's taken over a week for some people. Use refunds sparingly or add a secondary tracking layer on the server side.

"I want to implement purchase verification on the server side. How do I go about this?"

- The AndroidPurchase object contains the properties itemId, developerPayload, orderId, jsonData, signature, purchaseTime, and purchaseToken. You can pass all or a subset of this data back to your server for verification. Refer to Google's documentation for information on how to validate a purchase using your key.

"What happens when a subscription expires?"

- When a subscription expires, it will no longer be present in the inventory response.

"How do I use the IABExample.as file in Flash Professional CS6?"

1. First, create the application, add the extension, and update Google Play by following this guide, Sections 1-4.
2. Copy and paste IABExample.as into the same folder as your .fla. Do not copy and paste its contents on to the timeline. That will not work.
3. In Flash properties, under 'Document Class', type 'IABExample' (no quotes) and press OK.
4. Edit the IABExample.as file, and change the PUBLIC_KEY value to your own public key.
5. Build and install the application.

"Why doesn't the extension work when I run my swf on my Mac / iPhone / Windows Computer?"

- Android Billing is part of Google Play, and the Android Operating system. The extension will only work when you run it on an Android phone or tablet.

"Google Play used to have unmanaged items that could be bought repeatedly. How do I do that with the new API?"

- All Google Play items are now managed. To replicate the old behavior, purchase the item, then immediately call consumeItem() on it.

"Why does my application stop responding after I call a function?"

- If you don't have a debugger attached, and you're doing something that's causing an ErrorEvent, execution will halt but you may not notice it. Always add event listeners of the type AndroidBillingErrorEvent.
- If you've done that, look carefully at your function signature, and make sure the type is e:AndroidBillingErrorEvent and not e:AndroidBillingEvent.

"I still need help! How can I get technical support?"

- Your purchase comes with free technical support by email – just send us a message at support@milkmangames.com . We're here to help! Please remember that...:

- We're open during United States business hours, excluding U.S. Holidays, Monday-Friday, Pacific Standard Time. We strive to answer all support email within 24 hours (not including weekends and holidays) but usually do so much faster. Remember that we may not be in the same time zone.
- Please remember to mention: which extension you're having a problem with, what IDE you're using (such as 'Flash CS6' or 'Flash Builder 4.6'), and what device you're targeting. If you're experiencing an error message, please specify what that message is.
- We don't provide tech support through blog comments, Facebook, or Twitter. Please email us and we'll be happy to help you out!