StoreKit In-App Purchase Native Extension for iOS

For support, contact support@milkmangames.com

Before you begin:

To View full AS3 documentation, see 'docs/as3docs/index.html'.

This extension requires the AIR 15.0 SDK or higher, which you can get at http://www.adobe.com/devnet/air/air-sdk-download.html ***.***

Review 'example/StoreKitExample.as' for a sample application.

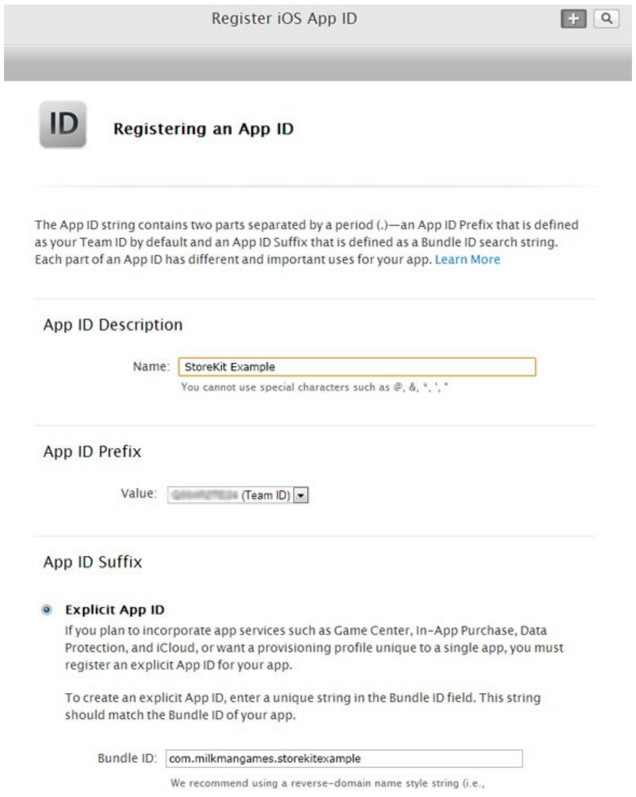
Note that StoreKitExample.as is a Document Class. If you're a Flash Professional user and don't know how to use a document class, see the appendix "Using the Example Class in Flash CS6" at the end of this guide.

1. Set Up In-App Purchase in iTunes Connect

The StoreKit extension makes adding in-app purchases to your iOS Application easy; however, setting up the program on Apple's developer website takes a few steps:

Create a New App ID

1. Go to http://developer.apple.com , and select the iOS Developer area.
2. From the menu on the right, select 'Certificates, Identifiers, & Profiles'.
3. You need to create a custom App ID for your purchase-enabled application. Choose 'Identifiers' from the menu, then the Plus (+) to Add a new App ID.



Registering an App ID

The App ID string contains two parts separated by a period (.)—an App ID Prefix that is defined as your Team ID by default and an App ID Suffix that is defined as a Bundle ID search string. Each part of an App ID has different and important uses for your app. Learn More

App ID Description

Name: StoreKit Example

You cannot use special characters such as @, &, *, ', "

App ID Prefix

Value: [redacted] (Team ID)

App ID Suffix

● **Explicit App ID**

If you plan to incorporate app services such as Game Center, In-App Purchase, Data Protection, and iCloud, or want a provisioning profile unique to a single app, you must register an explicit App ID for your app.

To create an explicit App ID, enter a unique string in the Bundle ID field. This string should match the Bundle ID of your app.

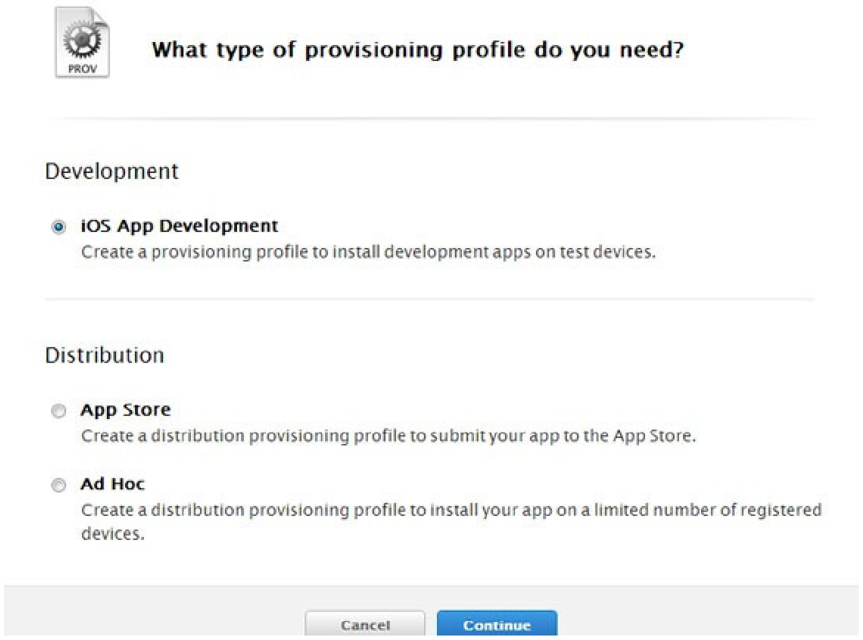
Bundle ID: com.milkmangames.storekitexample

We recommend using a reverse-domain name style string (i.e.,

4. For 'Description', enter a description for your app. This will be used to easily identify the App within the Provisioning Portal, but won't be shown to users. For 'App ID Prefix', select 'Use Team ID'. For Explicit App ID, enter the ID for your app, such as 'com.yourcompany.game'. Take note of this ID as you will have to use later in your AIR Application Descriptor file.

Create a New Provisioning Profile for Your Application

5. Now, return to Certificates, Identifiers, & Profiles window, and select Provisioning Profiles from the left hand side.


What type of provisioning profile do you need?

Development

[✓ iOS App Development]
Create a provisioning profile to install development apps on test devices.

Distribution

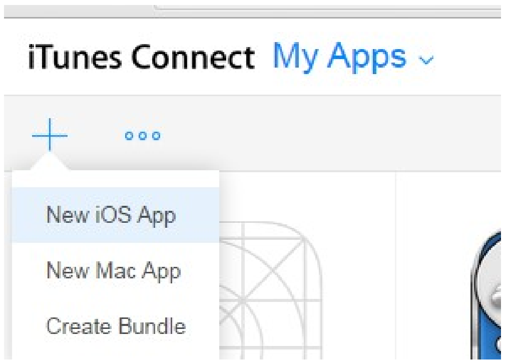
App Store
Create a distribution provisioning profile to submit your app to the App Store.

Ad Hoc
Create a distribution provisioning profile to install your app on a limited number of registered devices.

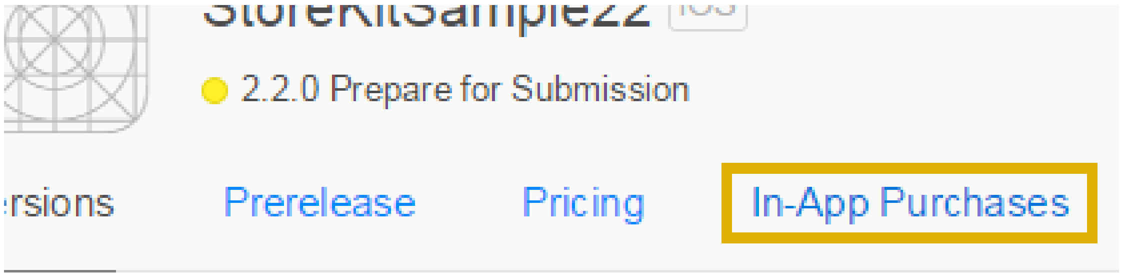
6. Choose the Plus (+) button in the top right. Choose 'iOS App Development' to create your development profile and press Continue.
7. On the next screen, select the App ID you created for your app earlier.
8. Complete the on screen prompts to generate and download the .mobileprovision. You will use this .mobileprovision for signing your apps with AIR.
9. **You may repeat these steps to create a Distribution Profile when you're ready to submit your app to the App Store.**
10. Return to the main developer page, and choose **iTunes Connect.** At this time, if you haven't already accepted the Paid Application contract under 'Contracts, Tax, and Banking', do so now.

Add In-App Purchases to Your Application

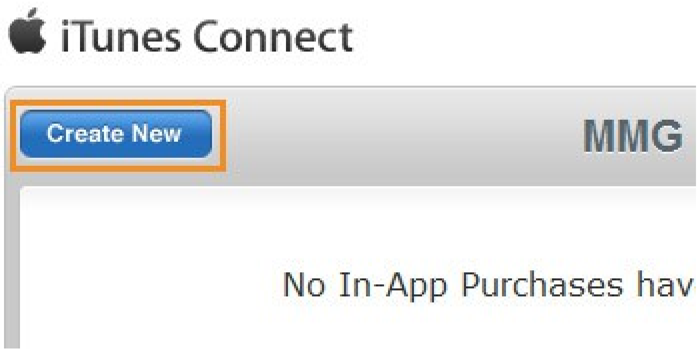
11. Now you need to register your Application with Apple. From iTunes Connect, choose 'My Apps', then '+'; 'New iOS App'



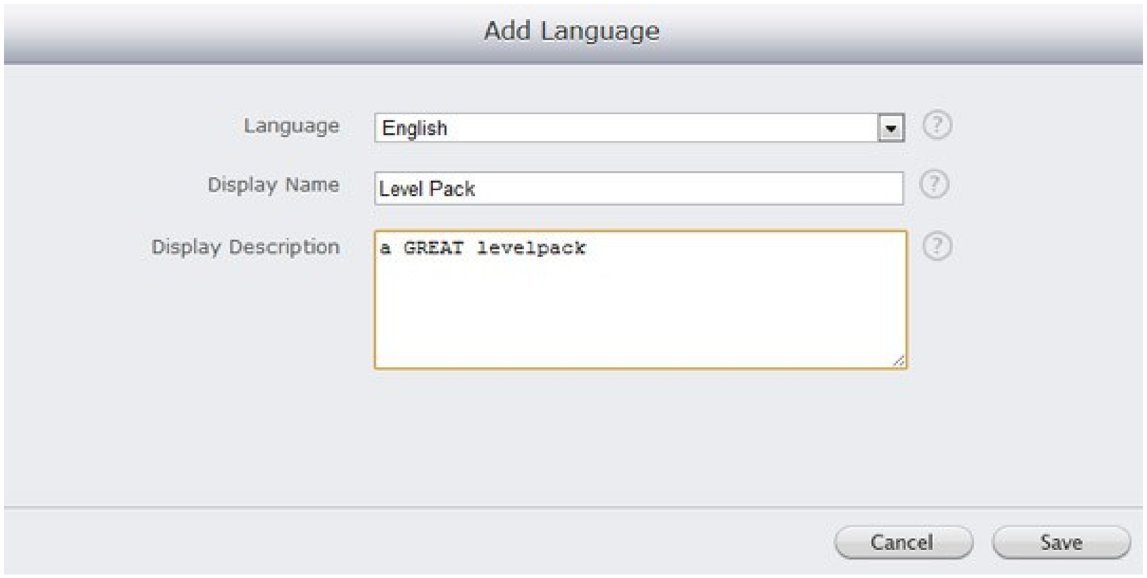
12. Enter a name and SKU for your application. ***For Bundle ID, you must now select the Bundle ID you created in Step 4***. Choose 'Continue'. Finish setting up the rest of the basic information for your new App.
13. When you're done, select your new Application, and choose 'In-App Purchases'.



14. On the next screen, choose 'Create New' to add an In-App Purchase product.



15. You now can choose between **Consumable, Non-Consumable** and **Subscriptions** product types. Follow the descriptions on this page for more information. Essentially, *consumable* items can be 'used up'- things like magic spells- and **non-consumable** items are permanent features – such as a level pack.
16. Enter the information for your Product on the next screen. For **Product ID,** use Apple's reccommended scheme of BundleID.productID- for instance, if you named your bundle 'com.yourcompanysite.yourappname' in Step 4, you might give an ID of 'com.yourcompanyname.yourappname.LevelPack' to your Level Pack product.
17. You must choose 'Add Language' and add at least one language definition. Fill in the *Display Name* (what users will see) and *description* and choose 'Save'.



Language: English

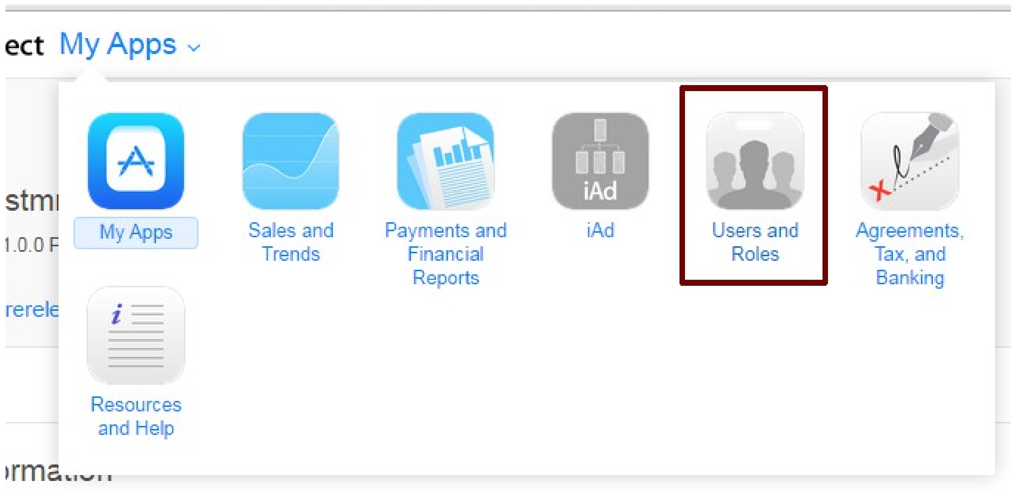
Display Name: Level Pack

Display Description: a GREAT levelpack

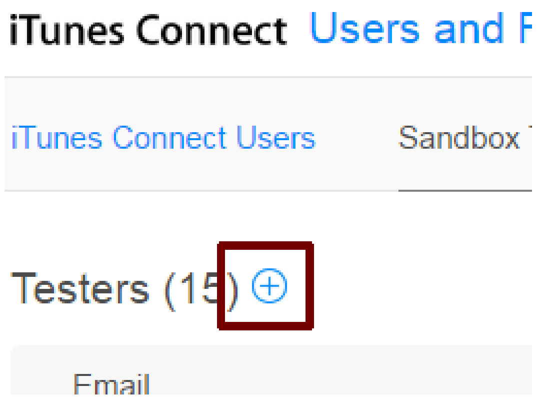
18. For 'Pricing and Availability', set **Cleared for Sale** to **Yes.** For 'Price Tier', choose the cost of the item. For the Screenshot, upload an image of your product in action. This is not shown to users, but is meant for Apple's review team to get an idea of what the product is supposed to do.
19. Repeat the process for each Product you want to add. When you're finished, select 'Done'. Be sure you've noted the exact Ids of all your products because you will need them in your application.

Create a Test User in the Apple Sandbox

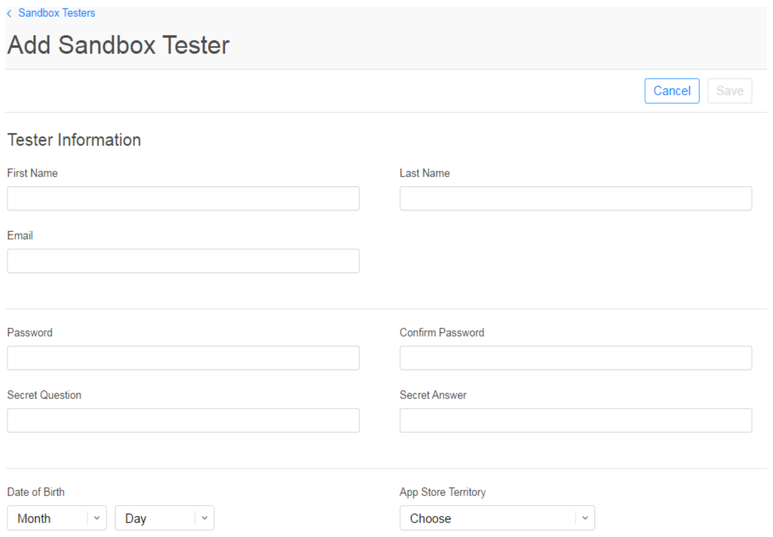
20. Almost done! The final step is to add a Test User to the Sandbox. A Test User is a 'fake' iTunes account, that you can use for testing in-app purchases. It will behave just like a real account, except no money is actually be deducted when the test purchases happen. Return to the iTunes Connect screen, pull down 'My Apps' and choose 'Users and Roles'.



21. Choose 'Sandbox Tetsters' then the '+' to add a new tester account.



22. This is a fake user for testing purposes. Later, you will be able to login to the store when testing your application, and complete the purchase flow without actually having any money charged to your account. For **First Name** and **Last Name**, enter whatever you wish. For **email address**, you do not need to enter a real address- but it does need to be unique. You should choose an email address at your own domain and remember it (even if the address doesn't actually exist), for instance 'tester1@yourcompany.com'. Make sure you mark down the password you choose as well. Fill in the rest of the relevant information and press 'Save'.



2. Install the AIR 15.0 SDK in your IDE

The StoreKit extension requires the AIR 15.0 SDK or higher. You can download the latest AIR SDK from http://www.adobe.com/devnet/air/air-sdk-download.html. If you haven't already installed the AIR 15.0 SDK (or higher) for your Flash CS6 or Flash Builder IDE, follow the instructions below:

Enabling the AIR 15.0 SDK (or higher) in Flash Professional CS6+:

1. Unzip the AIR 15.0 SDK (or higher) package to a location on your hard drive.
2. Launch Flash Professional.
3. Select Help > Manage AIR SDK...
4. Press the Plus (+) Button and navigate to the location of the unzipped AIR SDK
5. Press OK
6. Select File > Publish Settings
7. Select the AIR 15.0 (or higher) SDK for iOS from the 'Target' Dropdown menu

Enabling the AIR 15.0 SDK (or higher) in Flash Builder 4.6+ on Windows:

1. Unzip the AIR 15.0 SDK package to a location on your hard drive.
2. Close Flash Builder.
3. Locate the Flash Builder SDK directory. On the PC, this is usually c:\Program Files\Adobe\Adobe Flash Builder 4.6\sdks .
4. Make a copy of the current Flex SDK directory, and give it a descriptive name. For instance, copy the "4.6.0" SDK folder inside /sdks and name the copy "4.6.0_AIR15".
5. Copy and paste the contents of the AIR 15 SDK on top of the 4.6.0_AIR15 directory. Accept all changes.
6. Edit the flex-sdk-description.xml file inside the new directory, and change the value of the <name> tag to 'Flex 4.6.0 (AIR 15)'.
7. Open Flash Builder and choose Project > Properties > Flex Compiler > Configure Flex SDKs.
8. Press 'Add' and navigate to the new folder location.

Enabling the AIR 15 SDK (or higher) in Flash Builder 4.6+ on a Mac:

1. Copy the contents AIR 15 SDK package to a location on your hard drive.
2. Close Flash Builder.
3. Locate the Flash Builder SDK directory. On the Mac, it is usually /Applications/Adobe Flash Builder 4.6/sdks/. On the PC, c:\Program Files\Adobe\Adobe Flash Builder 4.6\sdks .
4. Create a new folder inside the SDK folder, called AIR15SDK and copy the contents of the SDK package into it.
5. Open the Terminal, and merge the AIR 15 SDK files into your current SDK directory:

   ```
   sudo cp -Rp /Applications/Adobe\ Flash\ Builder\ 4.6/sdks/AIR35SDK/ /Applications/Adobe\ Flash\
   Builder\ 4.6/sdks/4.6.0/
   ```

6. Edit the flex-sdk-description.xml file inside the new directory, and change the value of the <name> tag to 'Flex 4.6.0 (AIR 15)'.
7. Open Flash Builder and choose Project > Properties > Flex Compiler > Configure Flex SDKs.

8. Press 'Add' and navigate to the new folder location.

3. Include the StoreKit Extension Library

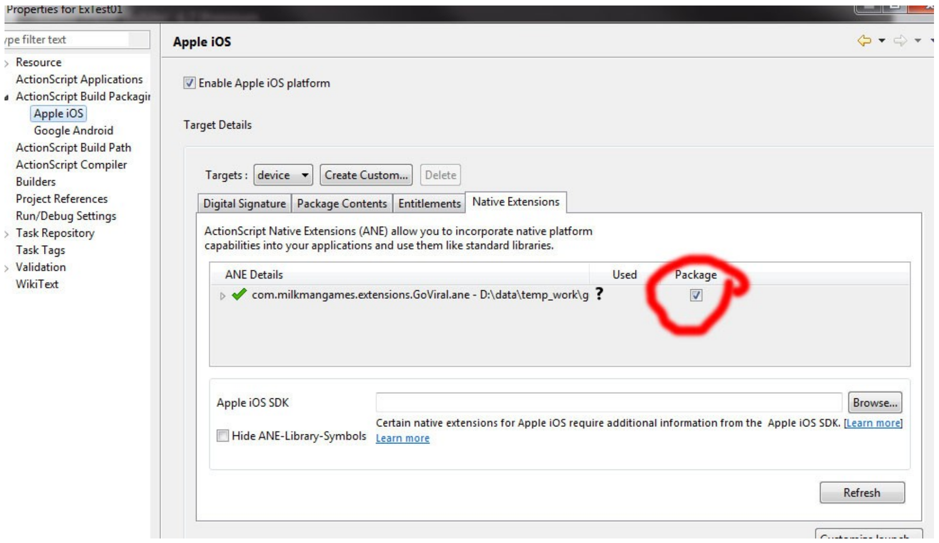
Add the com.milkmangames.extensions.StoreKit.ane to your project. These are in the /extension folder of the extension package zip file.

In Flash Professional CS6+:

1. Create a new project of the type 'AIR for iOS'
2. Select File > Publish Settings...
3. Select the wrench icon next to 'Script' for 'Actionscript Settings'
4. Select the Library Path tab
5. Press the 'Browse for Native Extension (ANE) File' button and select the com.milkmangames.extensions.StoreKit.ane file. Press OK.
6. Make sure you're using the AIR 15 (or higher) SDK per the instructions in Section 2.
7. Select File>Publish Settings...
8. For 'Target' Select 'AIR 15 for iOS' (or higher version)

In Flash Builder 4.6+:

1. Go to *Project Properties*
2. Select *Native Extensions*
3. Choose *Add ANE...* and navigate to the com.milkmangames.extensions.StoreKit.ane file
4. Make sure the 'package' box is checked for your target, under Actionscript Build Packaging:



Apple iOS

[✓] Enable Apple iOS platform

Target Details

Targets : device

ActionScript Native Extensions (ANE) allow you to incorporate native platform capabilities into your applications and use them like standard libraries.

ANE Details	Used	Package
com.milkmangames.extensions.GoViral.ane - D:\data\temp_work\g	?	✓

Apple iOS SDK

Certain native extensions for Apple iOS require additional information from the Apple iOS SDK. [Learn more]

[ ] Hide ANE-Library-Symbols

Make sure you're using the AIR 15 or higher SDK per the instructions in Section 2.

4. API Quick Start

The StoreKit extension can be up and running in a few simple calls. See '**example/StoreKitExample.as**' for a full example, including making purchases, restoring transactions, saving state, and more. 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.

Initialize the Extension

1. Import the API Classes:

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

2. First, initialize the API and create an instance of the StoreKit object. Make sure the current platform will support iOS Purchases (as in, it's not a PC or Android, etc.):

```
if(StoreKit.isSupported())
{
       StoreKit.create();
}
else {
       trace("StoreKit only works on iOS!");
       return;
}
```

3. Check if the specific Device will support In-App Purchases. It is possible for parental controls or other settings to prevent purchases from being made at all, so you need to check this case first before using the other methods.

```
if(!StoreKit.storeKit.isStoreKitAvailable())
{
       trace("this device has purchases disabled.");
       return;
}
```

Load Details about In-App Products

4. Using the StoreKit.storeKit.loadProductDetails method, you can retrieve information about available purchases, such as title, description, price, and so on. To use these method, pass in a Vector array of product IDs you created in “Section 1: Add In-App Purchases to Your Application”:

```
// the first thing to do is to supply a list of product ids you want to display,
// and Apple's server will respond with a list of their details (titles, price, etc)
// assuming the ids you pass in are valid.  Even if you don't need to use this
// information, you must make the details request before doing a purchase.

// the list of ids is passed in as an as3 vector (typed Array.)

var productIdList:Vector.<String>=new Vector.<String>();
productIdList.push("com.yourcompany.yourapp.LevelPack");
productIdList.push("com.yourcompany.yourapp.MagicSpell");

// when this is done, we'll get a PRODUCT_DETAILS_LOADED or PRODUCT_DETAILS_FAILED
event and go on from there...
StoreKit.storeKit.loadProductDetails(productIdList);
```

5. In response to this request, a `StoreKitEvent.PRODUCT_DETAILS_LOADED` event will be dispatched. Its `.validProducts` property will contain an array of `StoreKitProduct` objects, with additional information about the products. If any of the IDs you requested are not valid, they will be listed in the array

property `.invalidProductIds`:

```
// listen for a response from loadProductDetails():
StoreKit.storeKit.addEventListener(StoreKitEvent.PRODUCT_DETAILS_LOADED,onProducts);

function onProducts(e:StoreKitEvent):void
{
            for each(var product:StoreKitProduct in e.validProducts)
            {
                  trace("ID: "+product.productId);
                  trace("Title: "+product.title);
                  trace("Description: "+product.description);
                  trace("String Price: "+product.localizedPrice);
                  trace("Price: "+product.price);
            }
            trace("Loaded "+e.validProducts.length+" Products.");
            if (e.invalidProductIds.length>0)
            {
                  trace("[ERR]: invalid product ids:"+e.invalidProductIds.join(","));
            }
}
```

6. You must also add an event listener for `StoreKitErrorEvent.PRODUCT_DETAILS_FAILED`, which will be dispatched if an error occurs with the product request. You should be sure to include this listener to avoid errors halting your program:

```
// listen for ERROR response from loadProductDetails():
StoreKit.storeKit.addEventListener(StoreKitErrorEvent.PRODUCT_DETAILS_FAILED,
onProductsFailed);

function onProductsFailed(e:StoreKitErrorEvent):void
{
       trace("error loading products: "+e.text);
}
```

Purchase an In-App Product

7. When your user wants to purchase a product, you can use the `StoreKit.storeKit.purchaseProduct()` function. Pass the product ID you created in iTunes Connect as the first parameter, and (optionally), the quantity of items to buy as the second parameter:

```
// this call purchases a product.  The second parameter is an
// optional quantity- if you want to purchase more than one item at a time.
StoreKit.storeKit.purchaseProduct("com.yourcompany.yourapp.MagicSpell",1);
```

8. If the purchase succeeds, a `StoreKitEvent.PURCHASE_SUCCEEDED` event will be dispatched. At this time, you should run code to “give” the purchased item to the user within your application:

```
// this event is fired when a purchase goes through ok
StoreKit.storeKit.addEventListener(StoreKitEvent.PURCHASE_SUCCEEDED,onPurchaseSuccess);

function onPurchaseSuccess(e:StoreKitEvent):void
```

```
{
      // your app is now responsible for 'giving' the user whatever they bought!
      giveUserItem(e.productId);
}
```

9. If the user cancels the purchase, `StoreKitEvent.PURCHASE_CANCELLED` will be dispatched instead:

```
// this event is fired when a purchase gets cancelled
StoreKit.storeKit.addEventListener(StoreKitEvent.PURCHASE_CANCELLED,onPurchaseCancel);

function onPurchaseCancel(e:StoreKitEvent):void
{
      trace("the user decided not to buy: "+e.productId);
}
```

10. In some cases, parental controls may require the user to get a parent's permission to complete the purchase, or it might otherwise be temporarily on hold. In this case `StoreKitEvent.PURCHASE_DEFERRED` will be dispatched instead. At a later time, you will receive the usual success or fail events after mom or dad has weighed in on this important purchasing decision.

```
// this event is fired when a purchase gets deferred
StoreKit.storeKit.addEventListener(StoreKitEvent.PURCHASE_DEFERRED,onPurchaseDeferred);

function onPurchaseCancel(e:StoreKitEvent):void
{
      trace("Waiting for permission to buy: "+e.productId);
}
```

11. You must also add an event listener for `StoreKitErrorEvent.PURCHASE_FAILED`, which will be dispatched if an error occurs with the purchase request. You should be sure to include this listener to avoid errors halting your program:

```
// listen for ERROR response from purchaseProduct():
StoreKit.storeKit.addEventListener(StoreKitErrorEvent.PURCHASE_FAILED,
onPurchaseFailed);

function onPurchaseFailed(e:StoreKitErrorEvent):void
{
       trace("error purchasing product: "+e.text);
}
```

Restore Previous Transactions

12. Apple requires that your app includes a button to "Restore Transactions." If the user uninstalls your app, or gets a new device, they will use this button to restore any non-consumable items they had previously purchased. To initiate a transaction restore, call `StoreKit.storeKit.restoreTransactions()`:

```
// this will make a PURCHASE_SUCCEEDED event happen again for
// anything non-consumable the user has bought previously for your app.
// When it's all done, the TRANSACTIONS_RESTORED event will occur.
StoreKit.storeKit.restoreTransactions();
```

13. After calling `restoreTransactions()`, **a `StoreKitEvent.PURCHASE_SUCCEEDED` event for all previous transactions is "replayed".** If you've implemented the `StoreKitEvent.PURCHASE_SUCCEEDED` correctly in step 8, no additional action should be required to handle the restore.

When there are no more previous purchase events to be replayed, the `StoreKitEvent.TRANSACTIONS_RESTORED` event will be dispatched:

```
// this event is fired when all the old PURCHASE_SUCCEEDED events were replayed
StoreKit.storeKit.addEventListener(StoreKitEvent.TRANSACTIONS_RESTORED,
onTransactionsRestoreComplete);

function onTransactionsRestoreComplete(e:StoreKitEvent):void
{
      // your app got PURCHASE_SUCCEEDED for each old purchase, and should've
      // give the items to the user.
      trace("restore complete!");
}
```

14. You must also add an event listener for `StoreKitErrorEvent.TRANSACTION_RESTORE_FAILED`, which will be dispatched if an error occurs with the restore request. You should be sure to include this listener to avoid errors halting your program:

```
// listen for ERROR response from purchaseProduct():
StoreKit.storeKit.addEventListener(StoreKitErrorEvent.TRANSACTION_RESTORE_FAILED,
onRestoreFailed);

function onRestoreFailed(e:StoreKitErrorEvent):void
{
       trace("something went wrong restoring transactions: "+e.text);
}
```

(Optional) Displaying In-App Product Views

15. Optionally, the extension can display a window inside your app for any product on the iTunes Store- (apps, songs, movies, etc.). This can be useful for promoting your other apps, or other related products, from within your own app.
16. Displaying a product view requires the iTunes Product ID, which can be obtained by searching for the product in iTunes, right-clicking on its image, and copying the image URL. Then copy the numeric ID value from the URL (for instance: the '343200656' from 'https://itunes.apple.com/us/app/angry-birds/id343200656' is the iTunes ID for Angry Birds).
17. Displaying a product view requires iOS 6 or higher. Determine whether the functionality is available by checking `StoreKit.storeKit.isProductViewAvailable()`, and if so, calling `displayProductView()` with the iTunes ID:

```
if(StoreKit.storeKit.isProductViewAvailable())
{
      StoreKit.storeKit.displayProductView("343200656");
}
```

18. A product view window will be displayed and start loading the product information. When this view appears, `StoreKitEvent.PRODUCT_VIEW_DISPLAYED` is dispatched:

```
// listen for when the product view is shown
```

```
StoreKit.storeKit.addEventListener(StoreKitEvent.PRODUCT_VIEW_DISPLAYED,
onProductDisplayed);

function onProductDisplayed(e:StoreKitEvent):void
{
       trace("showed product view for itunes id: "+e.productId);
}
```

19. When this information has finished loading and is shown to the user, `StoreKitEvent.PRODUCT_VIEW_LOADED` is dispatched:

```
// listen for when the product view's inner content is loaded
StoreKit.storeKit.addEventListener(StoreKitEvent.PRODUCT_VIEW_LOADED,
onProductViewLoaded);

function onProductViewLoaded(e:StoreKitEvent):void
{
       trace("content loaded in view for itunes id: "+e.productId);
}
```

20. You must also add an event listener for `StoreKitErrorEvent.PRODUCT_VIEW_FAILED`, which will be dispatched if an error occurs loading the iTunes content. You should be sure to include this listener to avoid errors halting your program:

```
// listen for ERROR response loading the product view
StoreKit.storeKit.addEventListener(StoreKitErrorEvent.PRODUCT_VIEW_FAILED,
onProductViewFailed);

function onProductViewFailed(e:StoreKitErrorEvent):void
{
      trace("something went wrong with the product view: "+e.text);
}
```

21. When the user closes the product view window, `StoreKitEvent.PRODUCT_VIEW_DISMISSED` is dispatched:

```
// listen for when the product view is dismissed
StoreKit.storeKit.addEventListener(StoreKitEvent.PRODUCT_VIEW_DISMISSED,
onProductDisplayed);

function onProductDismissed(e:StoreKitEvent):void
{
       trace("the user closed the product view for "+e.productId);
}
```

(Optional) Handling Apple-Hosted Downloadable Content

This is an an advanced user feature. *Before implementing downloads of Apple-hosted content*, note that:

- iOS 6 or higher is required. If you support iOS 4 or 5, you'll need an alternate strategy for hosting.
- You must create and upload the downloadable content using Xcode 4+, with the "In-App Purchase Content" product template. The Bundle Identifier needs to match that of the corresponding In-App Product in iTunes Connect. Upload it by choosing Product>Archive from the XCode menu.

- You should be familiar with the flash.filesystem.File API and the SecurityDomain restrictions that apply to iOS file management in Adobe AIR.

22. Optionally, Apple will host downloadable files for non-consumable items for you, if your app runs on iOS 6 or higher. You can determine if the application is capable of downloading Apple-Hosted content at runtime by using the `StoreKit.storeKit.isHostedContentAvailable()` method. If it returns false, the device is running an older version of iOS and no downloading will occur.

    If it returns true, you can add listeners for the three download events dispatched by the extension:

```
if(!StoreKit.storeKit.isHostedContentAvailable())
{
      trace("this device is not capable of downloading apple-hosted content.");
}
else
{
      StoreKit.storeKit.addEventListener(StoreKitEvent.DOWNLOAD_UPDATED,
onDownloadUpdated);
      StoreKit.storeKit.addEventListener(StoreKitEvent.DOWNLOAD_FINISHED,
onDownloadFinished);
      StoreKit.storeKit.addEventListener(StoreKitErrorEvent.DOWNLOAD_FAILED,
onDownloadFailed);
}
```

23. After a successful purchase of a non-consumable item that includes Apple-hosted downloadable content, the extension will automatically begin the download of the package from Apple's server. As the download progresses, `StoreKitEvent.DOWNLOAD_UPDATED` events will be periodically dispatched, indicating the progress of the download:

```
function onDownloadUpdated(e:StoreKitEvent):void
{
      trace("downloading item: "+e.productId);
      // e.downloadProgress is a 0-1.0 scale
      trace("percent complete:"+(e.downloadProgress*100));
      trace("seconds remaining:"+e.downloadTimeRemaining);
      trace("content version:"+e.downloadVersion);
}
```

24. When the download is completed, `StoreKitEvent.DOWNLOAD_FINISHED` is dispatched. The extension will automatically unpack the downloaded content into a subdirectory of your application's local storage path, and provide the location of the downloaded content in the `downloadPath` property of the event. You may then use the `flash.filesystem.File` object to access the files and process them in your app, or move them to a new location. In the example below, we look through the download directory and load any PNG files found on to the stage:

```
import flash.display.Loader;
import flash.filesystem.File;
import flash.system.ApplicationDomain;
import flash.system.LoaderContext;
import flash.system.SecurityDomain;

// dispatched when a download has finished
function onDownloadUpdated(e:StoreKitEvent):void
{
      trace("finished downloading:"+e.productId);
      var downloadDirectory:File=new File(e.downloadPath);
```

```
        if (downloadDirectory.exists)
        {
              var contents:Array=theStuff.getDirectoryListing();
              for each(var file:File in contents)
              {
                    if (file.extension=="png")
                    {
                          var context:LoaderContext=new LoaderContext(true,
                           ApplicationDomain.currentDomain);

                          var loader:Loader=new Loader();

                          loader.load(new URLRequest(file.url),context);

                          stage.addChild(loader);
                    }
              }
        }
  }
```

25. You must also add an event listener for `StoreKitErrorEvent.DOWNLOAD_FAILED`, which will be dispatched if an error occurs with the download. You should be sure to include this listener to avoid errors halting your program:

```
// listen for ERROR response during a download
StoreKit.storeKit.addEventListener(StoreKitErrorEvent.DOWNLOAD_FAILED,
onDownloadFailed);

function onDownloadFailed(e:StoreKitErrorEvent):void
{
       trace("something went wrong with the download: "+e.text);
}
```

(Optional) Getting the App Receipt and Server Side Validation

26. **This is an Advanced User Feature.**

 On iOS 7 devices and above, you can access the app's App Receipt. This is a binary bit of data that represents purchase information for the app itself, as well as any in-app purchases that have been made by the current user for the app. This data is useful for advanced users who wish to validate receipt information on their own server. (See also **"I'm running my own server back-end to deliver products and verify purchases with Apple's validation server. How do I implement this on the client side?"** in the appendix on page 17.)

27. Getting the App Receipt requires iOS 7 or higher. Determine whether the functionality is available by checking `StoreKit.storeKit.isAppReceiptAvailable()`, and if so, getting the receipt with `getAppReceipt()` :

```
if(StoreKit.storeKit.isAppReceiptAvailable())
{
      // get the base64-encoded app receipt
      var receipt:String=StoreKit.storeKit.getAppReceipt();
}
```

28. If the App Receipt needs to refreshed, call `StoreKit.storeKit.refreshAppReceipt()`. You may want to do this when testing in the SandBox, because in the SandBox the receipt will be null until that sandbox user has purchased an item. (In Production, the receipt is populated at download time from the App Store.) `StoreKitEvent.APP_RECEIPT_REFRESHED` is dispatched, and its .receipt property contains a Base64 Encoded String representing the receipt's data:

```
StoreKit.storeKit.refreshAppReceipt();
// listen for when the app receipt is refreshed
StoreKit.storeKit.addEventListener(StoreKitEvent.APP_RECEIPT_REFRESHED,
onReceiptRefresh);

function onReceiptRefresh(e:StoreKitEvent):void
{
       trace("base64 app receipt is "+e.receipt);
}
```

29. When refreshing the App Receipt, You must also add an event listener for `StoreKitErrorEvent.APP_RECEIPT_REFRESH_FAILED`, which will be dispatched if an error occurs loading the App Receipt. You should be sure to include this listener to avoid errors halting your program:

```
// listen for ERROR response loading the app receipt
StoreKit.storeKit.addEventListener(StoreKitErrorEvent.APP_RECEIPT_REFRESH_FAILED,
onReceiptFailed);

function onReceiptFailed(e:StoreKitErrorEvent):void
{
      trace("something went wrong with the app receipt: "+e.text);
}
```

30. For Guidelines on implementing a receipt validation scheme on your own server, See **"I'm running my own server back-end to deliver products and verify purchases with Apple's validation server. How do I implement this on the client side?"** in the appendix on page 17.

5. Update Your Application Descriptor

You'll need to be using the AIR 15 or higher SDK, include the extension in your Application Descriptor XML. For an example, see **'example/app.xml'**. (Remember, the <id> section must exactly match your iTunes Connect id, so you can't copy that part of the example xml verbatim.)

1. Set your AIR SDK to 15 (or higher) in the app descriptor file:

```
<application xmlns="http://ns.adobe.com/air/application/15.0">
```

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

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

3. Make sure that your <id> property *exactly* matches the App ID you created in iTunes Connect.

6. Building for Test Vs Building for Release

1. **When testing:** Be sure the .mobileprovision file you built the ipa with was your **development provision created in Section 1, under *"Create a New Provisioning Profile for Your Application"***. (but you can and should use the distribution mobileprovision when building your final app to upload to the App Store.) **You'll also need to make sure the provision is installed on the Test phone – you can just drag the file to iTunes to do this.**
2. **When releasing to the store:** Be sure the .mobileprovision file you built the ipa with was your **release provision created in Seciton 1, under *"Create a New Provisioning Profile for Your Application"***.

7. Testing Purchases with your Test User Account

In Section 1, you created a Test User in iTunes Connect for testing the purchase flow without actually spending money. You should use this account when testing your implementation. Follow these steps exactly to run with the Test User:

1. Make sure your application descriptor xml is using your exact app id created in Section 1
2. Make sure when building your application you are using the ipa-debug or ipa-debug-interpreter target
3. Make sure when building your application you are using the development mobileprovision file you created in Section 1
4. Before starting your app on the device, go to **Settings>Store** (or Settings > iTunes & App Store), select your user, and **Sign Out**.
5. Start your test app. When you try to make a purchase, it will prompt you to sign in,use the test user email and password you created in iTunes Connect, in Section 1, *"Create a Test User in the Apple Sandbox"*.

8. Troubleshooting Common Problems

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

- First, create the application, add the extension, and perform iTunes Connect Setup, by following this guide, Sections 1-3.
- Copy and paste StoreKitExample.as into the same folder as your .fla. Do not copy and paste its contents on to the timeline. That will not work.
- Change the values "your_product_id" at the top of StoreKitExample.as to match the Product IDs you created in iTunes Connect.
- In Flash properties, under 'Document Class', type 'StoreKitExample' (no quotes) and press OK.
- Build and install the application. Be sure to use your test user account as described in Section 7, "Testing Purchases with your Test User Account".

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

- *The extension uses features that are built into the iOS operating system. The extension will only work when you run it on an iOS phone or tablet.*

"Why won't my app Connect to the iTunes Store?"

- Verify that the 'bundle id' in iTunes Connect EXACTLY matches the application id in your AIR manifest .xml file.
- Flash Builder will sometimes rename your App ID in the application XML from 'com.yourcompany.app' to 'com.yourcompany.app-debug' when compiling to a debug target. Build with a release target or manually fix the .xml file if this is the case.

- Make sure you've agreed to the Paid Applications contract with Apple in the Contracts section of iTunes Connect.
- Make sure that your app id in your application.xml exactly matches the App ID you created provision files for in Step 1
- Make sure you are using the custom provision files created in Section 1 for your App ID
- Make sure the product Ids you pass to loadProductDetails() match exactly the ones you created in iTunes Connect
- Be sure you are using a test user account as described in Section 7, "Testing Purchases with your Test User Account".
- You may need to wait 24 hours after creating products in iTunes Connect before they are available on the Apple test servers.
- **In-App Purchase will not work on Jailbroken devices!**

"I'm 'stuck' in sandbox mode and want to go back to my regular iTunes Store account!"

- Log out of the App Store, close and uninstall your AIR app, and force close the App Store application. Restart the App Store and log in with your actual personal account.
- Every time your test app with a mobileprovision for development is run, you will re-enter sandbox mode. You may have to repeat the process above to exit.

"Why does my app stop responding after I make a request?"

- Run in debug mode to see if there are any errors. Make sure you are capturing all the StoreKitErrorEvent's for the calls you are making. You can check the StoreKitErrorEvent.text property to see what went wrong.

"I'm running my own server back-end to deliver products and verify purchases with Apple's validation server. How do I implement this on the client side?"

- The example application shows tracking in-app purchases with a simple client based model. However, advanced users may wish to use their own server to track, verify, and provide in-app purchase content. For more information on how this might be implemented, see Apple's most recent documentation.

  When running on iOS 7 and above, `storeKit.isAppReceiptAvailable()` returns true, and you can call `storeKit.getAppReceipt()` to load the app's Store Receipt; see 'Getting the App Receipt' above for more information. Pass this receipt to your server, and then to Apple's server to validate all transactions in the app. Carefully review Apple's explanation of how to send and validate a receipt remotely here.

  When running on iOS 6 and below, `storeKit.isAppReceiptAvailable()` returns false, and the App Receipt is not available. In these cases, `StoreKitEvent.PURCHASE_SUCCEEDED`'s .receipt event will contain the 'old' format of Apple receipt, specific to that purchase, with you can pass to your server for validation. (This value will be null on iOS 7 and above, as you should be using the App Receipt for those cases.)

  You can force the extension to set `StoreKitEvent.PURCHASE_SUCCEEDED`'s .receipt property with the old, one string receipt-per-purchase format value, even on iOS 7 and 8, by setting the `forcceUseOfOldReceiptFormats` of `StoreKit.create()` to true.

- *Milkman Games cannot provide specific support on programming the server-side of a receipt validation protocol for your particular app in your particular server programming language. However, if you wish to do so, these are the general guidelines you should follow:*
  - *If storeKit.isAppReceiptAvailable() return TRUE, you can use the app receipt and you should:*
    1. After a purchase is successfully completed, get the App Receipt as described above in

'Getting the App Receipt'.

2. Send the receipt to your server along with any extra data such as the product ID. Construct a JSON string, according to Apple's spec here, that contains the app receipt as a key. Send the JSON to the appropriate URL (`https://sandbox.itunes.apple.com/verifyReceipt` in the sandbox, or in production, use `https://buy.itunes.apple.com/verifyReceipt`).
3. Parse Apple's response as described here. Take note of Apple's description of the meaning of the status key. Review the meaning of the receipt response key's fields here. When using the app receipt, you will need to iterate through the array of returned receipt descriptors and look for a match with the expected product ID.
4. If the receipt was found to be valid for your product, send a response to your app confirming the validity and providing the content the user purchased. (If it's not valid, you probably don't want to send the user the purchase content.)
5. When restoring transactions, wait for the transactions restored complete event, then refresh the receipt and send it to your server, to validate and provide content for any purchase items it includes.

- I*f storeKit.isAppReceiptAvailable() return FALSE, you can't use the app receipt and you should:*
  1. Each time you get the PURCHASE_SUCCEEDED event, send the .receipt property to your server along with any extra data such as the product ID.
  2. On the server, Construct a JSON string, according to Apple's spec here, that contains the individual transaction receipt, base64 encoded, as a key. Send the JSON to the appropriate URL (`https://sandbox.itunes.apple.com/verifyReceipt` in the sandbox, or in production, use `https://buy.itunes.apple.com/verifyReceipt`).
  3. Parse Apple's response as described here. Take note of Apple's description of the meaning of the status key. Because this receipt is specific to the item for the PURCHASE_SUCCEEDED event, if the status is valid you can assume the particular purchase is valid.
  4. If the receipt was found to be valid for your product, send a response to your app confirming the validity and providing the content the user purchased. (If it's not valid, you probably don't want to send the user the purchase content.)
  5. When restoring transactions, no additional work is needed, because PURCHASE_SUCCEEDED will have been dispatched for all previous .

- *Also, if you are using server-side verification, you may wish to delay the completion of the transaction on the client until your server has verified the purchase. To do this, you may call `StoreKit.storeKit.setManualMode(true)` to enable manual verification, then subsequently call `StoreKit.storeKit.manualFinishTransaction(transactionId)` to finish the transaction on the iOS side after your server has verified it, in the PURCHASE_`SUCCEEDED` or `PURCHASE_FAILED` listeners.* ***If you are also using Apple-Hosted downloadable content, you must NOT*** *call `manualFinishTransaction(transactionId)`* ***until*** *the downloaded has failed or completed; otherwise the downloaded content will not be available on the filesystem.*

"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!*