

Easy StoreKit for iOS

An easy to integrate, extensible plugin for StoreKit (In-App Purchase) integration in Unity3D

Document History

Date	Author	Version
23 August 2013	Preet Kamal Singh Minhas	Initial Release
04 March 2015	Preet Kamal Singh Minhas	1.1, Updated for Unity 5

- Document History 1**
- Introduction 3**
- Integrating Easy StoreKit with your Unity3D project..... 3**
- Using Easy StoreKit 3**
- Easy StoreKit exposed methods 4**
- Easy StoreKit events..... 5**
 - Sample method for configuring events 5
- Advanced 6**
 - Plugin Architecture 6
 - Implementing server side receipt verification..... 6
- Suggestions/Queries? 7**

Introduction

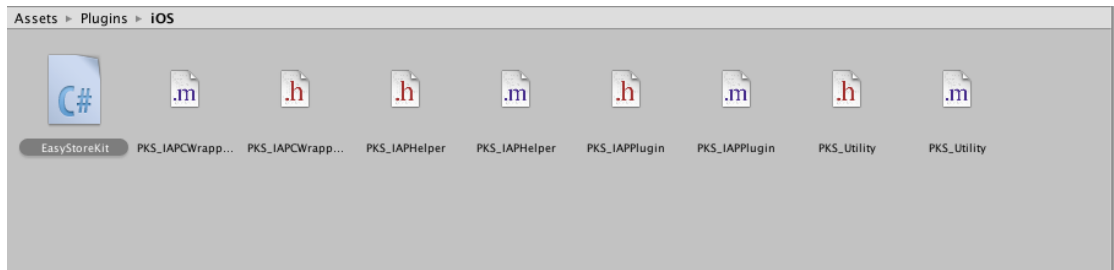
Integrating In-App Purchases in Unity3D can be a complicated process. Easy StoreKit aims to simplify this process by providing an easy to use API and an easy to understand plugin design. Easy StoreKit comes with full source code so that advanced users can easily modify the Objective-C/C code and tailor the plugin to work as per their needs.

Easy StoreKit handles all of the heavy lifting associated with In-App Purchases. However, this plugin cannot handle the required iTunesConnect setup process. Before using this plugin, you must set up a bundle identifier for your game and configure your app with In App Purchase identifiers. Please find below links to some wonderful guides and resources prepared by Apple:

1. [Apple Technical Note TN2259](#)
2. [In-App Purchase Programming Guide](#)

Integrating Easy StoreKit with your Unity3D project

1. Once you have imported the Easy StoreKit package into your project, you must ensure that Assets->Plugins->iOS folder contains the files listed in the screenshot below:



2. It is mandatory that your scene has a GameObject with the EasyStoreKit.cs script attached to it. This script receives all the callbacks from the native code.
3. You are now ready to use Easy StoreKit!

Using Easy StoreKit

1. The EasyStoreKit.cs script must be attached to any one GameObject in the scene.
2. Initialize StoreKit by calling `EasyStoreKit.AssignIdentifiers(string[] productIdentifiers);` This method must be called only once in your scene.
3. Configure the event handlers. StoreKit is an asynchronous API. Hence, the various callbacks are raised as events by the Easy StoreKit plugin. Refer [Easy StoreKit events](#).
4. Verify whether the user is allowed to make payments by calling

EasyStoreKit.CanMakePayments(); This method returns a boolean value to indicate whether the user is allowed to make payments or not.

5. Load the products using
EasyStoreKit.LoadProducts();
Once the products have been loaded, the *productsLoadedEvent* will be raised.
6. Buy a product by calling
EasyStoreKit.BuyProductWithIdentifier(string productId, int quantity);
This method will return true/false to indicate whether a valid product identifier had been supplied.
Based on the transaction status of the purchase, any of the following events might be called:
 - a) *transactionPurchasedEvent*
 - b) *transactionFailedEvent*
 - c) *transactionCancelledEvent*You must provide the relevant content to the user when the *transactionPurchasedEvent* is called. Also, in case of non-consumable products, it is recommended to save the purchased state of this identifier for later use.
7. To restore any previously purchased non-consumable products, you can call
EasyStoreKit.RestoreProducts();
The *transactionRestoredEvent* is called for every restored transaction.
If the restore process completes successfully, *restoreCompletedEvent* is called. Otherwise, the *restoreFailedEvent* is called.

Easy StoreKit exposed methods

```
//Initializes the StoreKit using the provided identifiers  
public static void AssignIdentifiers(string[] identifiers)
```

```
//Checks whether the user is allowed to make payments  
public static bool CanMakePayments();
```

```
//Loads the products from the App Store  
public static void LoadProducts();
```

```
//Buy the product identified by the identifier and in the quantity specified.  
Returns true if a valid product identifier is supplied as an argument and false  
otherwise.  
public static bool BuyProductWithIdentifier(string identifier, int quantity);
```

```
//Restores already purchased non-consumable items  
public static void RestoreProducts();
```

Easy StoreKit events

```
//The delegate definitions
public delegate void ProductsLoadedEventHandler(StoreKitProduct[] products);
public delegate void TransactionPurchasedEventHandler(string
productIdentifier);
public delegate void TransactionFailedEventHandler(string
productIdentifier,string errorMessage);
public delegate void TransactionRestoredEventHandler(string
productIdentifier);
public delegate void TransactionCancelledEventHandler(string
productIdentifier);
public delegate void RestoreFailedEventHandler(string errorMessage);
public delegate void RestoreCompletedEventHandler();
```

```
//Events
//Called when the products have been loaded
public static event ProductsLoadedEventHandler productsLoadedEvent;
//Called when the transaction has been successfully purchased
public static event TransactionPurchasedEventHandler
transactionPurchasedEvent;
//Called when the transaction has failed
public static event TransactionFailedEventHandler
transactionFailedEvent;
//Called when the transaction has been cancelled by the user
public static event TransactionCancelledEventHandler
transactionCancelledEvent;
//Called when any product is restored
public static event TransactionRestoredEventHandler
transactionRestoredEvent;
//Called when the restore process is complete
public static event RestoreCompletedEventHandler
restoreCompletedEvent;
//Called when the restore process has failed
public static event RestoreFailedEventHandler restoreFailedEvent;
```

Sample method for configuring events

```
private function ConfigureStoreKitEvents() {
    EasyStoreKit.productsLoadedEvent += ProductsLoaded;
    EasyStoreKit.transactionPurchasedEvent += TransactionPurchased;
    EasyStoreKit.transactionFailedEvent += TransactionFailed;
    EasyStoreKit.transactionRestoredEvent += TransactionRestored;
    EasyStoreKit.transactionCancelledEvent += TransactionCancelled;
    EasyStoreKit.restoreCompletedEvent += RestoreCompleted;
    EasyStoreKit.restoreFailedEvent += RestoreFailed;
}
```

```
function ProductsLoaded(products : StoreKitProduct[]) {
    //refresh the UI
}

function TransactionPurchased(productId : String) {
    //Unlock feature based on the identifier
}

function TransactionFailed(productId : String, errorMessage : String) {
    //display the error message to the user
}

function TransactionRestored(productId : String) {
    //Unlock feature based on the identifier restored.
}

function TransactionCancelled(productId : String) {
    //Remove any activity indicators as the user has cancelled the transaction
    //Do not display any message to the user
}

function RestoreCompleted() {
    //change ui state
}

function RestoreFailed(errorMessage : String) {
    //change ui state and display error message
}
```

Advanced

Plugin Architecture

PKS_IAPCWrapper : C Wrapper over the Objective-C methods

PKS_IAPPlugin : Methods to expose StoreKit functionality as a plugin

PKS_IAPHelper : Contains the implementation for iOS StoreKit

PKS_Utility : Utility methods to convert char* to NSString* and vice versa.

Implementing server side receipt verification

Provide an implementation for the method

-(BOOL) verifyTransaction:(SKPaymentTransaction*)transaction in

PKS_IAPHelper.m

Suggestions/Queries?

Please write in to contact@marchingbytes.com