

# In App Purchases 구현



작성자 : 왕수용([wangsy@wangsy.com](mailto:wangsy@wangsy.com))

소속 : (주)민트기술 (mintech.kr)

2011. 11. 17

revision #1

## 개요 3

### 기본 개념 익히기 3

IAP (In App Purchase) 3

Product 3

iOS & Mac 4

### “Product” 정보 만들고, 얻어오기 4

App ID 생성하기 4

Provisioning Profile 생성하기 5

bundle ID 업데이트 및 Xcode 설정 6

iTunes Connect 에 앱 정보 올리기 7

iTunes Connect 에 앱 바이너리 올리기 8

In-App Purchase 의 Product 새로 만들기 8

Product 관련 정보 받아오는 코드 구현하기 10

기다리기 13

### “Product” 구매 구현하기 13

Transactions 을 지원하는 코드 작성 14

테스트 사용자 생성 19

디바이스에서 테스트 20

구매 실험 20

### 아직 못다한 이야기들 20

# 개요

이 문서는 다음 문서들을 참조하여 작성되었다. 블로그 내용은 조금 오래됐기 때문에, 현재 시점에 맞게 수정하였다.

- WWDC 2011 - 510 In App Purchase for iOS and Mac OS X
- <http://troybrant.net/blog/2010/01/in-app-purchases-a-full-walkthrough/>

In App Purchases (이후, IAP) 를 iOS 에서 구현하는 것을 애플은 매우 단순하게 설명하고 있지만, 실제로 쉽게 따라해 볼 수 있는 예제가 없는 것이 사실이다.

아래 정보는 Xcode 4.2 를 기준으로 한다.

## 기본 개념 익히기

### IAP (In App Purchase)

1. 무료 앱에서 새로운 기능을 사용하기 위해서 지불하는 것
2. 게임에서 새로운 레벨을 얻기 위해서 지불하는 것
3. 가상의 상품을 구매하는 것
4. 잡지 구독을 신청하는 것

### Product

IAP 에서 추가로 구매할 수 있는 가상의 상품을 Product 라고 부른다. 예를 들어, 게임의 경우 추가 레벨을 플레이 할 수 있도록 하던지, 추가 무기를 구입 하던지, 새로운 잡지 콘텐츠를 구입하던지 이렇게 IAP 를 통해서 판매하는 것을 Product 라고 한다.

## iOS & Mac

IAP 는 iOS 3.0 이상 Mac OS X Lion 10.7 이상부터 지원된다.

## “Product” 정보 만들고, 얻어오기

아래 순서대로 따라하면서 만들어야 한다. 차근 차근히 따라해 보자.

1. App ID 생성하기
2. Provisioning Profile 생성하기
3. bundle ID 업데이트 및 Xcode 설정
4. iTunes Connect 에 앱 정보 올리기
5. iTunes Connect 에 앱 바이너리 올리기
6. In-App Purchase 의 Product 새로 만들기
7. Product 관련 정보 받아오는 코드 구현하기
8. 기다리기

### App ID 생성하기

IAP 를 지원하기 위해서는, 해당 앱의 App ID 가 “\*” 가 포함되어서는 안된다. 먼저 iOS Provisioning Portal 에 접속을 한다. 그리고, 왼쪽 메뉴에서 “App IDs” 메뉴를 선택해 준다.

아래와 같은 형태면 “고유(Unique) App ID” 형태이다.

M2US52D6AA.com.yourcompany.yourapp

하지만, 아래와 같은 형태면 “고유 App ID” 형태가 아니다.

M2US52D6AA.com.yourcompany.\*

만일 “고유 App ID” 가 없다면, 다음과 같이 새로 만들어야 한다. App ID 는 한번 생성하고 나면, 편집하거나 삭제할 수 없기 때문에, 매우 신중하게 만들어야 한다.

1. “App IDs” 가 선택된 상태에서, 오른쪽의 “New App ID” 버튼을 클릭해 준다.
2. 아래 정보를 기입해 준다.
  1. **Description** : 나중에 Provisioning Profile 만들때, 새로 만든 App ID 를 쉽게 확인할 수 있게 해준다. 따라서, 그때 알아 볼 수 있는 이름을 만들어 주면 된다.
  2. **Bundle Seed ID (App ID Prefix)** : 같은 Seed ID 를 쓰는 앱끼리는 서로 동일한 KeyChain 에 접근할 수 있다. KeyChain 을 통해서, 비밀번호 같은 것을 공유할 수 있는데, IAP 와는 무관한 이야기 이니, 무슨 뜻인지 모르면 그냥 기본 “Use Team ID” 를 선택해도 무방하다.
  3. **Bundle Identifier (App ID Suffix)** : com.yourcompany.appname 의 형식으로 써 주는데, 중요한 것은 이 이름과 나중에 Xcode 의 Info 에서 CFBundleIdentifier 에 쓰일 값과 동일해야 한다.

3. “Submit” 버튼 클릭

“고유 App ID” 형태로 쓰여 졌다면, In App Purchase 부분에 Enabled 라고 표시가 될 것이다.

## Provisioning Profile 생성하기

“고유 App ID” 를 만들었으니, 이제 이것과 연결된 Provisioning Profile 을 만들 차례다.

1. Provisioning Portal 화면에서, 좌측 “Provisioning” 메뉴를 선택한다.

2. “Development” 탭을 선택한 상태에서, 좌측 “New Profile” 버튼을 누른다.
3. 내용을 채운다.
  1. **Profile Name** : Xcode 에서 식별할 수 있는 적절한 이름을 작성해 준다.
  2. **Certificates** : 이 Profile 을 사용할 개발자의 인증서를 선택해 준다. 별다른 이유가 없다면 “Select All” 을 클릭해서 전체 선택을 해 준다.
  3. **App ID** : 방금 만들어 준, “고유 App ID”를 선택해 준다.
  4. **Devices** : 이 Profile 을 가지고, 테스트에서 사용할 기기를 선택해 준다. 별다른 이유가 없다면, “Select All”을 클릭해서 전체 선택을 해 준다.
4. “Submit” 버튼을 누르고 나면, 다시 목록 화면으로 돌아오는데, Status 란 이 Pending 으로 나온다. 이때 다시 한번 새로고침을 해 주면, Active 상태로 바뀐다.
5. “Download” 버튼을 눌러준다.
6. 다운받은 Profile 파일을 Xcode 에 드래그 앤 드롭으로 올려다 놓는다.
7. 혹은 다른 방법으로, 다운로드 대신, Xcode 에서 Window > Organizer 를 통해, Organizer 창을 띄우고, 좌측에 Library > Provisioning Profile 을 선택한 다음, 아래측 “Refresh” 버튼을 누르면, iOS Developer 계정을 묻고, 확인해 주면, 자동으로 새롭게 만든 Profile 을 가져와서 보여준다.

## bundle ID 업데이트 및 Xcode 설정

1. Xcode 에 가서, 해당 프로젝트 파일을 연 다음, View > Navigators > Show Project Navigator ( Cmd+1 ) 상단 프로젝트 아이콘을 클릭. 오른쪽에 나타나는 화면에서 좌측 TARGETS 아래 앱을 클릭하고, 오른쪽 상

단 탭에서 “Info” 를 선택해 준다. 여기서 “Bundle Identifier” 라고 된 부분의 오른쪽에, “고유 App ID” 와 동일한 값을 기입해 준다. 여기서 앞부분 (Bundle Seed ID) 부분은 빼고, 뒷부분 (Bundle Identifier) 부분만 쓰면 된다. 예를들어, “고유 App ID”가 `M2US52D6AA.com.yourcompany.yourapp` 와 같았다면, `com.yourcompany.yourapp` 부분만 써 주면 된다.

2. 상단의 Build Setting 탭으로 이동하여, Code Signing Identity 에서 Debug 와 Release 에 대해서 방금 새롭게 만든 Profile 을 선택해 주면 된다. (모든 라인에 동일하게 선택해 준다.)

## iTunes Connect 에 앱 정보 올리기

만일 앱스토어에 이미 앱을 등록해 놓은 상태라면, 이번 단계는 건너뛰어도 좋다.

“Product” 를 등록하기 전에, 먼저 앱 자체가 앱스토어에 등록 되어 있어야 한다. 아직 완전히 완성된 상태가 아니라도 걱정할 필요는 없다. 일단 먼저 등록만 해 놓아도 된다. 심사는 나중에 받아도 문제가 안된다.

SKU 와 Version 은 나중에 변경할 수 없으니, 신중하게 선택해야 한다.

1. 먼저 iOS Developer 사이트로 이동 한다.
2. 오른쪽에 있는 iTunes Connect 사이트로 들어간다.
3. 오른쪽에 보면, “Manage Your Applications” 메뉴가 나오는데, 그쪽으로 이동한다.
4. 왼쪽 상단에, “Add New App” 버튼을 눌러준다.
5. 관련된 항목을 모두 기입해 주고, “Save” 버튼을 눌러준다. 자세한 내용은 iTunes Connect 관련 가이드 문서를 참고한다.

## iTunes Connect 에 앱 바이너리 올리기

여기에 대해서는 애플 문서 어디에도 나와 있지 않지만, 사실 “Product” 를 만들기 위해서는 반드시 앱 바이너리를 올려야 한다. 앱 심사에 대해서는 걱정하지 않아도 된다. 올리고 나서 바로 Reject Binary 를 해 주면 된다.

자세한 내용은 iTunes Connect 관련 문서를 참고하기 바란다.

## In-App Purchase 의 Product 새로 만들기

1. iOS Developer 사이트로 이동한다.
2. 오른쪽 iTunes Connect 사이트에 접속한다.
3. 오른쪽 상단 “Manage Your Applications” 버튼을 누르고 들어가면, 올려진 앱들의 목록이 나온다. 여기서, IAP 를 지원하고 싶은 앱을 선택한다. 들어가면 오른쪽 상단에 “Manage In-App Purchases” 버튼이 나온다. 누르고 들어간다.
4. 왼쪽 상단의 “Create New” 버튼을 눌러준다.
5. Type 을 선택해 준다.
  1. **Consumable** : 이 타입의 경우 매번 다운로드 할 때, 해당 구매한 내역이 사라지는 경우이다. 일회용 서비스와 같은 것일때 사용한다. 예를들어 낚시 게임에서 물고기 밥 같은 종류는 이 타입을 선택해 준다.
  2. **Non-Consumable** : 단 한번만 구매해도 될 때 사용한다. 시간이 지나도 소멸하지 않는 경우 예를 들어 게임에서 새로운 레벨이라던지 이런 경우 이 타입을 선택해 준다.
  3. **Auto-Renewable Subscription** : 특정 기간동안 서비스를 할 경우 이 타입을 선택해 준다. 특정 기간이 지난 경우, 사용자가 특별히 조치해 두지 않으면, 자동으로 갱신이 된다. 잡지라던지, 신문의 경우 이 타입이 해당된다.

이 타입의 경우, 사용자가 등록한 모든 기기에 콘텐츠가 전달이 된다. 만일 사용자가 취소를 하게 되면, 지정된 기간이 지나면, 자동으로 구독이 취소가 된다.

4. **Free Subscription** : Newsstand 에 무료 구독 기능을 구현할 수 있다. 사용자가 한번 등록하면, 사용자의 모든 디바이스에 적용이 된다. 이 타입은 만료되지 않으며, 반드시 Newsstand 기능이 구현된 앱에 만 해당된다.

5. **Non-Renewing Subscription** : 과거에는 이 타입의 경우 제한된 시간동안만 기능을 제공하기 위해서 많이 사용되었다. 여전히 이 타입을 사용할 수 있지만, Auto-Renewable Subscription 기능을 사용하는 것을 추천한다. 이유는

1. auto-renewable subscription 을 생성할 때, 다양한 옵션으로 기간 설정을 할 수 있다.

2. non-renewable subscription 의 경우, 사용자가 매번 갱신해야 하기 때문에, 앱의 코드상에서 만료사점을 파악하는 부분이 필요하다. 그리고, 그때마다 매번 사용자에게 갱신하도록 해야 하는데, 차라리 auto-renewable subscription 을 쓰면, 이런 불편이 없다.

3. auto-renewable subscription 의 경우 사용자의 등록된 모든 기기에 자동으로 등록이 된다. non-renewable subscription 의 경우 이렇게 구현하려면, 앱에서 직접 잘 만들어야 하는데, 힘들 것이다.

6. 상세 항목을 써 준다.

1. **Reference Name** : AppStore 에서는 보이지 않는다. iTunes Connect 에서 Sales and Trend 에서 항목으로 표시될 것이다. 이후 수정할 수 없다.

2. **Product ID** : 고유 식별자.  
com.yourcompany.yourapp.product 이런 형태로 써 주는 것이 좋다.
3. **Add Language** : 반드시 하나 이상의 언어를 설정해 주어야 한다.  
“Add Language” 버튼을 클릭하면 새로운 창이 뜬다. 아래 항목을 기입해 준다.
  1. **Language** : 표시할 언어를 선택해 준다.
  2. **Display Name** : 사용자에게 보여줄 이름이다. Auto-Renewable Subscription 의 경우 기간을 표시 안하는 것이 좋다.
  3. **Display Description** : 사용자에게 표시되는 설명 텍스트이다.
4. **Cleared for Sale** : “Yes”로 선택해 준다. 아니면, 테스트 할 때, Invalid Product 의 결과가 나올 수 있다.
5. **Price Tier** : 받고 싶은 금액을 골라 준다.
6. **Screenshot for Review** : IAP 가 동작하는 화면을 캡처해서 올려 준다. 리뷰의 목적으로만 사용하고, 앱스토어에서 사용자에게 보여지지는 않는다.
7. “Save” 버튼을 누르면 생성이 된다.

## Product 관련 정보 받아오는 코드 구현하기

```
// InAppPurchaseManager.h
#import <StoreKit/StoreKit.h>

#define
kInAppPurchaseManagerProductsFetchedNotification
@"kInAppPurchaseManagerProductsFetchedNotification"
```

```

@interface InAppPurchaseManager : NSObject
<SKProductsRequestDelegate>
{
    SKProduct *proUpgradeProduct;
    SKProductsRequest *productsRequest;
}

// InAppPurchaseManager.m

- (void)requestProUpgradeProductData
{
    NSMutableSet *productIdentifiers = [NSMutableSet
setWithObject:@"com.runmonster.runmonsterfree.upgrade
topro" ];
    productsRequest = [[SKProductsRequest alloc]
initWithProductIdentifiers:productIdentifiers];
    productsRequest.delegate = self;
    [productsRequest start];

    // we will release the request object in the
delegate callback
}
#pragma mark -
#pragma mark SKProductsRequestDelegate methods

- (void)productsRequest:(SKProductsRequest *)request
didReceiveResponse:(SKProductsResponse *)response
{
    NSArray *products = response.products;
    proUpgradeProduct = [products count] == 1 ?
[[products firstObject] retain] : nil;
    if (proUpgradeProduct)
    {
        NSLog(@"Product title: %@",
proUpgradeProduct.localizedTitle);
        NSLog(@"Product description: %@",
proUpgradeProduct.localizedDescription);
        NSLog(@"Product price: %@",
proUpgradeProduct.price);
        NSLog(@"Product id: %@",
proUpgradeProduct.productIdentifier);
    }
}

```

```

    }

    for (NSString *invalidProductId in
response.invalidProductIdentifiers)
    {
        NSLog(@"Invalid product id: %@",
invalidProductId);
    }

    // finally release the request we alloc/init'ed
in requestProUpgradeProductData
    [productsRequest release];

    [[NSNotificationCenter defaultCenter]
postNotificationName:kInAppPurchaseManagerProductsF
etchedNotification object:self userInfo:nil];
}

```

```
// SKProduct+LocalizedPrice.h
```

```
#import <Foundation/Foundation.h>
#import <StoreKit/StoreKit.h>
```

```
@interface SKProduct (LocalizedPrice)
```

```
@property (nonatomic, readonly) NSString
*localizedPrice;
```

```
@end
```

```
// SKProduct+LocalizedPrice.m
```

```
#import "SKProduct+LocalizedPrice.h"
```

```
@implementation SKProduct (LocalizedPrice)
```

```
- (NSString *)localizedPrice
{
```

```

    NSNumberFormatter *numberFormatter =
[[NSNumberFormatter alloc] init];
    [numberFormatter
setFormatterBehavior:NSNumberFormatterBehavior10_4];
    [numberFormatter
setNumberStyle:NSNumberFormatterCurrencyStyle];
    [numberFormatter setLocale:self.priceLocale];
    NSString *formattedString = [numberFormatter
stringFromNumber:self.price];
    [numberFormatter release];
    return formattedString;
}

@end

```

## 기다리기

실제 애플의 앱스토어에서 테스트 가능한 상태가 되기까지는 시간을 두고 조금 기다려야 한다. 길게는 24시간까지 걸릴 수도 있다.

## “Product” 구매 구현하기

SKProduct 를 통해서 정보를 가져오는 것까지 성공했다. 이제 이 정보를 통해서 구매하기 기능만 구현하면 된다. 사실 구매하기 기능을 구현하는 것은, 여태까지 한 것에 비하면 쉬운 편이다.

아래의 과정을 거치면 된다.

1. Transactions 을 지원하는 코드 작성
2. 테스트 사용자 생성
3. 디바이스에서 테스트
4. 구매 실험

## Transactions 을 지원하는 코드 작성

실제 구매 관련한 UI 는 제공되는 것이 아니라, 직접 만들어야 한다.

storeKit 에는 UI 관련한 부분은 전혀 없다. 애플에서 제공하는 UI 처럼 보이게 할려면, 스스로 만드는 수 밖에 없다.

```
// InAppPurchaseManager.h

// add a couple notifications sent out when the
transaction completes
#define KIAPTransactionFailedNotification
@"KIAPTransactionFailedNotification"
#define KIAPTransactionSucceededNotification
@"KIAPTransactionSucceededNotification"

...

@interface InAppPurchaseManager : NSObject
<SKProductsRequestDelegate,
SKPaymentTransactionObserver>
{
    ...
}

// public methods
- (void)loadStore;
- (BOOL)canMakePurchases;
- (void)purchaseProUpgrade;

@end

// InAppPurchaseManager.m

#define KIAPProUpgradeProductId
@"com.yourcompany.yourapp.product"

...
```

```

#pragma -
#pragma Public methods

//
// call this method once on startup
//
- (void)loadStore
{
    // restarts any purchases if they were
    interrupted last time the app was open
    [[SKPaymentQueue defaultQueue]
    addTransactionObserver:self];

    // get the product description (defined in early
    sections)
    [self requestProUpgradeProductData];
}

//
// call this before making a purchase
//
- (BOOL)canMakePurchases
{
    return [SKPaymentQueue canMakePayments];
}

//
// kick off the upgrade transaction
//
- (void)purchaseProUpgrade
{
    SKPayment *payment = [SKPayment
    paymentWithProductIdentifier:kIAPProUpgradeProductId]
    ;
    [[SKPaymentQueue defaultQueue]
    addPayment:payment];
}

#pragma -
#pragma Purchase helpers

```

```

//
// saves a record of the transaction by storing the
receipt to disk
//
- (void)recordTransaction:(SKPaymentTransaction
*)transaction
{
    if ([transaction.payment.productId
isEqualToString:kIAPPProUpgradeProductId])
    {
        // save the transaction receipt to disk
        [[NSUserDefaults standardUserDefaults]
setValue:transaction.transactionReceipt
forKey:@"proUpgradeTransactionReceipt" ];
        [[NSUserDefaults standardUserDefaults]
synchronize];
    }
}

//
// enable pro features
//
- (void)provideContent:(NSString *)productId
{
    if ([productId
isEqualToString:kIAPPProUpgradeProductId])
    {
        // enable the pro features
        [[NSUserDefaults standardUserDefaults]
setBool:YES forKey:@"isProUpgradePurchased" ];
        [[NSUserDefaults standardUserDefaults]
synchronize];
    }
}

//
// removes the transaction from the queue and posts a
notification with the transaction result
//
- (void)finishTransaction:(SKPaymentTransaction
*)transaction wasSuccessful:(BOOL)wasSuccessful

```

```

{
    // remove the transaction from the payment queue.
    [[SKPaymentQueue defaultQueue]
finishTransaction:transaction];

    NSDictionary *userInfo = [NSDictionary
dictionaryWithObjectsAndKeys:transaction,
@"transaction" , nil];
    if (wasSuccessful)
    {
        // send out a notification that we've
finished the transaction
        [[NSNotificationCenter defaultCenter]
postNotificationName:kIAPTransactionSucceededNotifica
tion object:self userInfo:userInfo];
    }
    else
    {
        // send out a notification for the failed
transaction
        [[NSNotificationCenter defaultCenter]
postNotificationName:kIAPTransactionFailedNotificatio
n object:self userInfo:userInfo];
    }
}

//
// called when the transaction was successful
//
- (void)completeTransaction:(SKPaymentTransaction
*)transaction
{
    [self recordTransaction:transaction];
    [self
provideContent:transaction.payment.productIdentifier]
;
    [self finishTransaction:transaction
wasSuccessful:YES];
}

```

```

//
// called when a transaction has been restored and
// and successfully completed
//
- (void)restoreTransaction:(SKPaymentTransaction
*)transaction
{
    [self
recordTransaction:transaction.originalTransaction];
    [self
provideContent:transaction.originalTransaction.paymen
t.productId];
    [self finishTransaction:transaction
wasSuccessful:YES];
}

//
// called when a transaction has failed
//
- (void)failedTransaction:(SKPaymentTransaction
*)transaction
{
    if (transaction.error.code !=
SKErrorPaymentCancelled)
    {
        // error!
        [self finishTransaction:transaction
wasSuccessful:NO];
    }
    else
    {
        // this is fine, the user just cancelled, so
don't notify
        [[SKPaymentQueue defaultQueue]
finishTransaction:transaction];
    }
}

#pragma mark -
#pragma mark SKPaymentTransactionObserver methods

```

```

//
// called when the transaction status is updated
//
- (void)paymentQueue:(SKPaymentQueue *)queue
updatedTransactions:(NSArray *)transactions
{
    for (SKPaymentTransaction *transaction in
transactions)
    {
        switch (transaction.transactionState)
        {
            case SKPaymentTransactionStatePurchased:
                [self
completeTransaction:transaction];
                break;
            case SKPaymentTransactionStateFailed:
                [self failedTransaction:transaction];
                break;
            case SKPaymentTransactionStateRestored:
                [self
restoreTransaction:transaction];
                break;
            default:
                break;
        }
    }
}

```

## 테스트 사용자 생성

실제 구매 테스트를 해 보기 위해서는, 앱스토어 테스트 계정을 만들어야 한다.

1. iOS Developer 사이트로 이동한다.
2. 오른쪽 iTunes Connect 사이트에 들어간다.
3. 왼쪽 아래편에 Manager Users 선택해 준다.
4. Select User Type 에서 Test User 를 선택해 준다.

5. 왼쪽 상단, Add New User 선택해 준다.
6. 각 항목을 기입해 준다. iTunes Store 계정을 생성할 때와 거의 동일한 정보이다. 기본적으로 Email 주소를 확인하지 않기 때문에, 아무 계정을 해주어도 무관하다.

## 디바이스에서 테스트

실제 기기에서 테스트 할 때, 이미 로그인 되어있는 스토어 계정에서 로그아웃 해 준다.

1. 아이폰의 “설정” 앱에 들어간다.
2. Store 탭을 선택한다.
3. 맨 아래 Apple ID 버튼을 누르고, 나타나는 팝업 창에서 “로그아웃” 버튼을 클릭해 준다.

## 구매 실험

1. 디바이스에서 앱을 실행한다. (시뮬레이터에서는 테스트 할 수 없다.)
2. 구매 버튼을 눌러본다.
3. 테스트용 계정으로 로그인 해 본다.

만일 동일 계정으로 반복하면, 이미 구매했다고 나온다.

## 아직 못다한 이야기들

1. Auto-Renewable Subscription 에 대한 자세한 이야기
2. IAP for Mac App

### 3. Receipt 사용하여, 확인하기