
# iOS SDK v2.0

## SDK Integration Guide

Prepared by the
**Mobile App Tracking Engineering Team**

*Introduction & Table of Contents*

# iOS SDK v2.0

The Mobile App Tracking SDK for Apple iOS provides basic application install and event tracking functionality. The iOS SDK is provided in the form of a framework that you may simply include in your iOS project. Our SDK is compatible with iPhone®, iPad® and iPod Touch® devices. Developers may also add and track additional events beyond an app install (such as purchases, game levels, and any other user engagement) as well as app opens and closes to determine user engagement.

This document outlines the iOS SDK install and integration.

*Table of Contents*

## 1. What's new in v2.0

- Simplified set methods for passing in data and setting options
- Simplified start method, just pass in advertiser id and key
- Control auto generation of data like mac address and other identifiers
- Improved and simplified track action methods
- Create custom track action events with event items to track in-app purchases
- New support for Re-Engagement
- Consolidation to a single download for the framework static library file, support for arc and non-arc projects in one framework file
- Sample Xcode project
- Support for iOS® 4.3 and above
- Support for iPhone 5.0 and iOS 6
- Uses new iOS 6 Advertiser Identifier and Vendor Identifier
- App to App Tracking, track an install from your app to other apps you've integrated with MAT SDK

## 2. Compatibility

The MAT SDK is compatible with iOS 4.3 and above using Xcode 4.5. The SDK is compatible with ARC and non-ARC projects.
There is 1 SDK framework file available for download:

**MobileAppTracker.framework - iOS 4.3+**

## 3. Download iOS SDK

There is one download option for developers using the iOS SDK:

**Projects with target of iOS 4.3+ may use the MobileAppTracker.framework**

Once you've downloaded the iOS SDK that is compatible with your mobile apps, decompress the .zip file and copy the files to your development computer. The iOS SDK is comprised of a framework folder making it easy to integrate into your mobile app and contains the necessary header file as well as appledoc style documentation.

## 4. Quick Start

***Note: the starter project is available as a complete Xcode project***

**1.** Create a new project in Xcode.

**2.** In Xcode, Build Phases -->Link Binary with Libraries, link the following files:

**A. MobileAppTracker.framework**. Tip: be sure to choose the plus to "add" the file to your project. This will create the necessary linking for a static library in Xcode. Additionally, you can drag and drop the framework file into the list of libraries you want to link.

**B.** CoreTelephony.framework

**C.** SystemConfiguration.framework

**D.** MobileCoreServices.framework

**E.** AdSupport.framework

**3.** Open your **AppDelegate.m file.**

**4. Add the following to the top of your project**

```
#import <MobileAppTracker/MobileAppTracker.h>
```

**5. Initialize the MobileAppTracker.m class** by pasting the following inside your **"didFinishLaunchingWithOptions"** method.

```
[[MobileAppTracker sharedManager] startTrackerWithAdvertiserId:advertiser_id
                                                  advertiserKey:advertiser_key
                                                  withError:&error];
```

- Use your Advertiser Id and Advertiser Key provided to you when you created the Mobile App on Mobile App Tracker's web site
- &error provides an NSError object to evaluate to make sure the SDK initializes correctly. This is optional.

**6.** Pass a unique identifier such as a device id or UUID using the setDeviceId method

**7.** Your AppDelegate.m file should look similar to below:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Override point for customization after application launch.

    // MAT Notes:
    // The minimum data needed to initialize the MobileAppTracker is advertiser_id
    // and advertiser_key.
    // An optional NSError object can be passed to the start method
    // Other keys are defined in the header file under Data Parameters
    [[MobileAppTracker sharedManager] startTrackerWithAdvertiserId:MAT_Advertiser_Id
                                                     advertiserKey:MAT_Advertiser_Key
                                                         withError:nil];

    // MAT Notes:
    // Turn this on to see debug messages and to allow duplicate events
    // NOTE: don't release with this set on
    [[MobileAppTracker sharedManager] setShouldDebugResponseFromServer:YES];

    // MAT Notes:
    // This should be a unique generated value so that each install
    // will be attributed to the correct user
    // This will help further identify the app to the Tracking Engine
    NSString *userId = @"UserId12345";
    [[MobileAppTracker sharedManager] setUserId:userId];

    // MAT Notes:
    // Create a unique identifier to pass to the SDK via either of these 2 methods
    // Version 1: use the deprecated device unique identifier
    [[MobileAppTracker sharedManager]setDeviceId:[[UIDevice currentDevice]uniqueIdentifier]];

    // Version 2: create a UUID specific to the app
    //[[MobileAppTracker sharedManager]setDeviceId:[MATAppDelegate createUniqueDeviceId]];

    // !!!: Always call track update to make sure your app gets an install/update event
    // MAT Notes:
    // Always call trackInstallWithUpdateOnly when the app starts
    // When set to NO, the SDK will always record an install and then any
    // version updates.
    // When set to YES (update only), the SDK will never record an install
    // and will record version updates only.
    [[MobileAppTracker sharedManager] trackInstallWithUpdateOnly:NO];

    return YES;
}
```

**8.** Build and run your application.

## 5. SDK Data

The SDK provides several methods as a way of passing data independently of the initialization method. These settings can be called using provided header methods.

***advertiser_key*** – your advertiser key when you setup an app in MAT on the web

***advertiser_id*** – your advertiser id when you setup an app on MAT on the web

Other Parameters

| *name* | *description* |
|---|---|
| currency_code | the default currency_code for the sdk is USD, pass in currency_code to override this. Please format the currency code in the currency format as referenced by:<br>http://www.thefinancials.com/Default.aspx?SubSectionID=curformat |
| redirect_url | to be used with cookie based tracking. this is the url scheme name to be used in your app. |
| device_id | typically, this is the UDID passed in from your ios app. Or, create a unique identifier specific to your app, you can call the **CFUUIDCreate** function to create a UUID, and write it to the defaults database using the **NSUserDefaults** class.<br>*MobileApp Tracking tracks app installs from other apps (app-to-app) and mobile web. When UDID is available from the publisher, the following code can be added by your team in our SDK to collect UDID upon install. Collecting UDID as device ID on click and then install allows device ID matching to be used for attribution.<br><br>To set device_id use the following deprecated code in iOS:<br><br>NSString *device_id = [[UIDevice currentDevice] uniqueIdentifier] ;<br>// then pass it to the MobileAppTracker SDK<br>[[MobileAppTracker sharedManager]setDeviceId:device_id]; |

| *name* | *description* |
|---|---|
| user_id | unique user ID of user defined by advertiser<br>Example value: tom12345 |
| open_udid | the OpenUDID of the device. To be generated or retrieved using the official implementation according to: **http://OpenUDID.org** |
| truste_tpid | a Truste TPID to be used by the sdk. see **http://truste.com/** |
| use cookie tracking | set to YES if you want the sdk to use cookie tracking. make sure to set the redirect url in the app settings online on the Mobile App Tracking web site for your app. default=NO |
| should auto generate advertiser identifier | default = YES, the SDK will gather an advertiser identifier in iOS 6 if the user has not turned it off. Set to NO to pass in your own<br>Advertiser Id is a non-permanent, non-personal device identifier and can be turned off by a user |
| should auto generate vendor identifier | default = YES, the SDK will gather a vendor identifier in iOS 6. Set to No to pass in your own<br>a UUID that may be used to uniquely identify the device, same across apps from a single vendor |
| should auto generate mac address | set to YES if you want the sdk to auto generate a mac address<br>default=YES |
| should auto generate odin1 key | set to YES if you want the sdk to auto generate an ODIN1 key.<br>the ODIN of the device. To be generated or retrieved using the official implementation according to: **http://code.google.com/p/odinmobile/wiki/ODIN1**<br>default=YES |

| *name* | *description* |
|---|---|
| should auto generate open udid key | set to YES if you want the sdk to generate a Open UDID key<br>default=YES |
| should use https | set to YES if you want the sdk to use https, otherwise NO uses http<br>default=YES |
| should debug response from server | set to yes when debugging the sdk. This will show server response data and allow duplicate event's to be logged. ** Set to no for app releases ** |

**The MobileAppTracker SDK will track updates and installs based on the following rules:**

When your app runs, the first event to be called is:

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions

To Track installs and/or updates, you should call:

trackInstallWithUpdateOnly:YES/NO

**What to do if you have an app in the Apple store and now want to integrate the MAT sdk:**

Call **trackInstallWithUpdateOnly:YES**, only updates will be tracked and no installs will be recorded. So, when a new version of your app is installed by a user, an update will be recorded to the MAT servers. This options exists so that if you have an app in the Apple store and you are just now using the MAT sdk, no install events need to be recorded, only update events will be recorded.

**What to do if you have a new app about to be released to the Apple Store:**

Call **trackInstallWithUpdateOnly:NO**. This will record any new installs of your app and subsequently record updates to your app. So, when the user first downloads your app from the Apple store, an ***install*** will be recorded. Then, when a user updates to a new version of your app, an ***update*** will be recorded.

Optionally, you can call **trackInstallWithUpdateOnly:YES/NO** and pass in a reference id.

## 6. Track In-App Events

You can track events in your application by calling trackAction. This will record an event name or id and optional reference ids or event items.

All Track Action methods in the sdk begin with this method name:

**[[MobileAppTracker sharedManager] trackActionForEventIdOrName:@"event name or id" eventIsId:YES/NO];**

You can pass in an event name or event id. If you pass in an event name and eventIsId:NO, then you are indicating to the sdk that you want your own event name passed in. If you pass in an event id and eventIsId:YES, then you are indicating that you have a pre-defined event id that you want to track against.

### Track Custom Events

The Track Action method can record an event with revenue and a currency code. Likewise the Track Action method can also record events that contain sub items, much like a shopping cart. The keys we look for are item, value, and quantity.

```
[[MobileAppTracker sharedManager] trackActionForEventIdOrName:@"purchase"
                                                    eventIsId:NO
                                                  referenceId:@"ref1234567890"];

NSDictionary * item1 = [[NSDictionary alloc]initWithObjectsAndKeys:@"cheese", @"item",
                                                    @"10.00", @"unit_price",
                                                    @"2", @"quantity",
                                                    @"20.00", @"revenue", nil];
```

```
NSDictionary * item2 = [[NSDictionary alloc]initWithObjectsAndKeys:@"banana", @"item",
                                                                   @"1.50", @"unit_price",
                                                                   @"3", @"quantity",
                                                                   @"4.50", @"revenue", nil];

NSMutableArray * eventItems = [[NSMutableArray alloc]init];
[eventItems addObject:item1];
[eventItems addObject:item2];

[[MobileAppTracker sharedManager] trackActionForEventIdOrName:@"checkout"
                                                    eventIsId:NO
                                                   eventItems:eventItems
                                                  referenceId:@"ref12345678"
                                               revenueAmount:@"27.50"
                                                 currencyCode:@"USD"];
```

## 7. Track Opens and Closes

You can track opens and closes of your mobile app. This data allows you to analyze user engagement, usage trends and time spent in app. The “open” and “close” events are created by default in the tracking engine.

Below are examples to track app on open and close events within your AppDelegate.m. To track open and closes, you must pass in “open” and “close” respectively in the **trackAction** method.

```
- (void)applicationDidBecomeActive:(UIApplication *)application
{
[[MobileAppTracker sharedManager] trackActionForEventIdOrName:@"open" eventIsId:NO];
}
- (void)applicationWillResignActive:(UIApplication *)application
{
  [[MobileAppTracker sharedManager] trackActionForEventIdOrName:@"close" eventIsId:NO];
}
```

### 8. Track Events in WebView

Sometimes, your app contains HTML content which you would like to track events on. A way to do this is find the url request that corresponds with your event and call a trackAction before it loads.

The example below checks if the url ends with “skip” and calls a skip action before completing the request, although you should adapt this to your own requests.

```
- (BOOL)webView:(UIWebView*)webView
shouldStartLoadWithRequest:(NSURLRequest*)request
navigationType:(UIWebViewNavigationType)navigationType {

        NSURL *url = request.URL;

        if ([[url lastPathComponent] isEqual:@”skip”]) {

                [[MobileAppTracker sharedManager] trackActionForEventIdOrName:@"skip"
                eventIsId:NO];

}

        return YES;
```

See the NSURL class documentation for accessible fields:

https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSURL_Class/Reference/Reference.html

### 9. Collecting User ID, OpenUDID and TRUSTe TPID

You can set the UserID,, OpenUDID and TRUSTe TPDID by calling the set<> methods for the sdk.

```
- (void)setOpenUDID:(NSString *)open_udid;

- (void)setTrusteTPID:(NSString *)truste_tpid;

- (void)setUserId:(NSString *)user_id;
```

### 10. Collecting ODIN

SDK generates and collects ODIN for you. If your application uses ODIN for any other purpose be sure it is generated using mac address bytes array representation and not mac address string. See specification for details: (http://code.google.com/p/odinmobile/wiki/ODIN1 ).

### 11. Collecting currency code

The SDK set's the currency to 'USD' by default,  You can override the currency by setting the currency using the setter method.  Please see **ISO Currency Codes** for a list of acceptable currency codes.

For example, to set currency to Canadian Dollar use the following code :

```
- (void)setCurrencyCode:(NSString *)currency_code
[[MobileAppTracker sharedManager] setCurrencyCode:@"CAD"];
```

### 12. Test iOS App Tracking

The SDK is designed to track conversions through the Apple iTunes App Store as well as from outside market places or third parties, so you can test it without updating your production mobile app listing.  This allows you to update your mobile app for Apple after testing that tracking is fully operational, saving you time and resources.

The easiest way to test MobileAppTracking is to send an email with a tracking link and the updated file for the iOS app. From an iOS device (Apple iPhone or iPad), conduct the test with the information from the email.

- Compose an email to yourself or the engineer testing the iOS app.
- Include this test tracking link in the email to start a tracking session. Use any Tracking Link for a publisher from the Tracking panel. We also provide a test link in the integration instructions on the Tracking Code page.
- Apple devices must have a provisional profile setup before you can install a developer app. **Learn more here.**

  https://developer.apple.com/library/mac/#recipes/xcode_help-devices_organizer/articles/provision_device_for_development-generic.html
- Click on the tracking link you included
- Install the iOS app with MobileAppTrackeriOS SDK integration on your Apple device.
- Continue opening the mobile app and proceed to the point where the trackInstall and/or trackAction methods should be called.
- Check reporting in Mobile App Tracking interface to see if an install and/or event was recorded.
- If conversion appears in reporting, then proceed to update production with your updated iOS mobile app.

## 13. App to App Tracking

App to App tracking provides the ability for one app (the refering app) to download another app (the target app). Also, the target app will then record an install event that contains data from the referring app. The data can contain a publisher id, campaign id and tracking id.

To setup app to app tracking:

AppA - the referring app

AppB - the target app

AppA calls one of the setTracking methods provided in the SDK. Each of the set tracking methods require a target app id parameter (the bundle identifier of the target app). There are optional parameters such as publisher id and campaign id. Also, the developer can specify that their app redirect to the link where AppB can be downloaded (typically this is iTunes).

Example:

pkg=”com.hasoffers.AppB”
aid=”877”t
cid=”some campaign id”
redirect=YES

```
[[MobileAppTracker sharedManager] setTracking:pkg
                                  advertiserId:aid
                                   campaignId:cid
                                       redirect:redirect];
```

AppA will get a tracking id from the HasOffers tracking engine and then redirect to the link pointing to AppB’s install.

Once installed, AppB containing the SDK will execute a track action install with the containing the data stored by AppA so that the install track action will be credited to AppA.

## 14. In-App Purchase Tracking

To track in-app purchase events, add one of the MobileAppTracker trackAction: methods to the paymentQueue:updatedTransactions: delegate callback method provided by SKPaymentTransactionObserver in the StoreKit framework. The following code snippet shows a sample implementation.

```
#pragma mark SKPaymentTransactionObserver Methods

- (void)paymentQueue:(SKPaymentQueue *)queue updatedTransactions:(NSArray *)transactions
{
  for (SKPaymentTransaction *transaction in transactions)
  {
     NSString *eventName = nil;
     NSString *idItemPurchased = transaction.payment.productIdentifier;
     NSArray *arrEventItem = [NSArray arrayWithObject:idItemPurchased];

     switch (transaction.transactionState)
     {
        case SKPaymentTransactionStatePurchased:
        {
           // self.products is the dictionary (NString, SKProduct) to be created by the user
           NSDictionary *dictProducts = self.products;

           SKProduct *product = (SKProduct *)([dictProducts
objectForKey:transaction.payment.productIdentifier]);

           // purchase successful
           eventName = @"purchaseSuccessful";
           int quantity = transaction.payment.quantity;
           NSDecimalNumber *price = product.price;
           NSNumber *revenue = [NSNumber numberWithDouble:quantity * [price doubleValue]];

           NSArray *keys = [NSArray arrayWithObjects:@"item", @"unit_price", @"quantity",
@"revenue", nil];
           NSArray *item = [NSArray arrayWithObjects:product.localizedTitle, price, [NSNumber
numberWithInt:quantity], revenue, nil];
```

```
            NSDictionary *dictItem = [NSDictionary dictionaryWithObjects:item forKeys:keys];

            arrEventItem = [NSArray arrayWithObject:dictItem];

            [[SKPaymentQueue defaultQueue] finishTransaction:transaction];

            break;
        }
        case SKPaymentTransactionStateFailed:

            // purchase failed
            eventName = @"purchaseFailed";

            break;
        case SKPaymentTransactionStateRestored:

            // purchase restored
            eventName = @"purchaseRestored";

        default:
            break;
    }

    if(nil != eventName)
    {
        // track MAT event
        [[MobileAppTracker sharedManager] trackActionForEventIdOrName:eventName
                                                            eventIsId:NO
                                                           eventItems:arrEventItem];
    }
  }
}
```