Unity Plugin

SDK Integration Guide

Version 1.1 | May 2013

©2013 HasOffers, Inc. | All rights reserved

Table of Contents

Introduction....3
Downloading the Unity Plugin....3
Implementation....3
Platform-Specific Settings....6
Installs and Updates....8
  Track Installs....8
  Handling Installs Prior to SDK Implementation - Track as Updates....8
Events....9
  Registration....10
  Purchases....10
  Opens....11
  Other Events....11
Testing Plugin Integration with SDK....12
Debug Mode and Duplicates....12
Additional Resources....13
  Custom Settings....13
  Event Items....14
  App to App Tracking....16
  List of all Supported Methods....17

Introduction

The MobileAppTracking (MAT) Plugin for Unity provides basic application install and event tracking functionality via the MobileAppTracking SDKs. To track installs, you must integrate the Unity plugin with your Unity app. Once the plugin is integrated and set to track installs, you can add and track additional events beyond an app install (such as purchases, game levels, and any other user engagement).

This document outlines the Unity Plugin integration and use cases.

Downloading the Unity Plugin

To download the MobileAppTracking Unity Plugin click here.

Implementation

To integrate the MobileAppTracking plugin in your Unity app, you will be creating a C# file that will attach to a Unity GameObject as a script. Depending on the platform (UNITY_ANDROID or UNITY_IPHONE), you will import the corresponding MobileAppTracking Unity library and make tracking calls with it.

We have included sample code in the plugin repository inside folder “sample” that shows how you might set up your Unity scene to include a script calling the MobileAppTracking library.

1.Create a new Unity project. Create a C# script file.

2.In the C# scripts, DllImport attribute needs to be set for each method to be imported from the plugin code.

Android:

```
[DllImport ("mobileapptracker")]
```

iOS:

```
 [DllImport ("__Internal")]
```

3. If you want to target both Android and iOS platforms, then you can use conditional compilation.

For Example:

```
#if UNITY_ANDROID

[DllImport ("mobileapptracker")]

private static extern void initNativeCode(string advertiserId, string conversionKey);

#endif

#if UNITY_IPHONE

[DllImport ("__Internal")]

private static extern void initNativeCode(string advertiserId, string conversionKey);

#endif
```

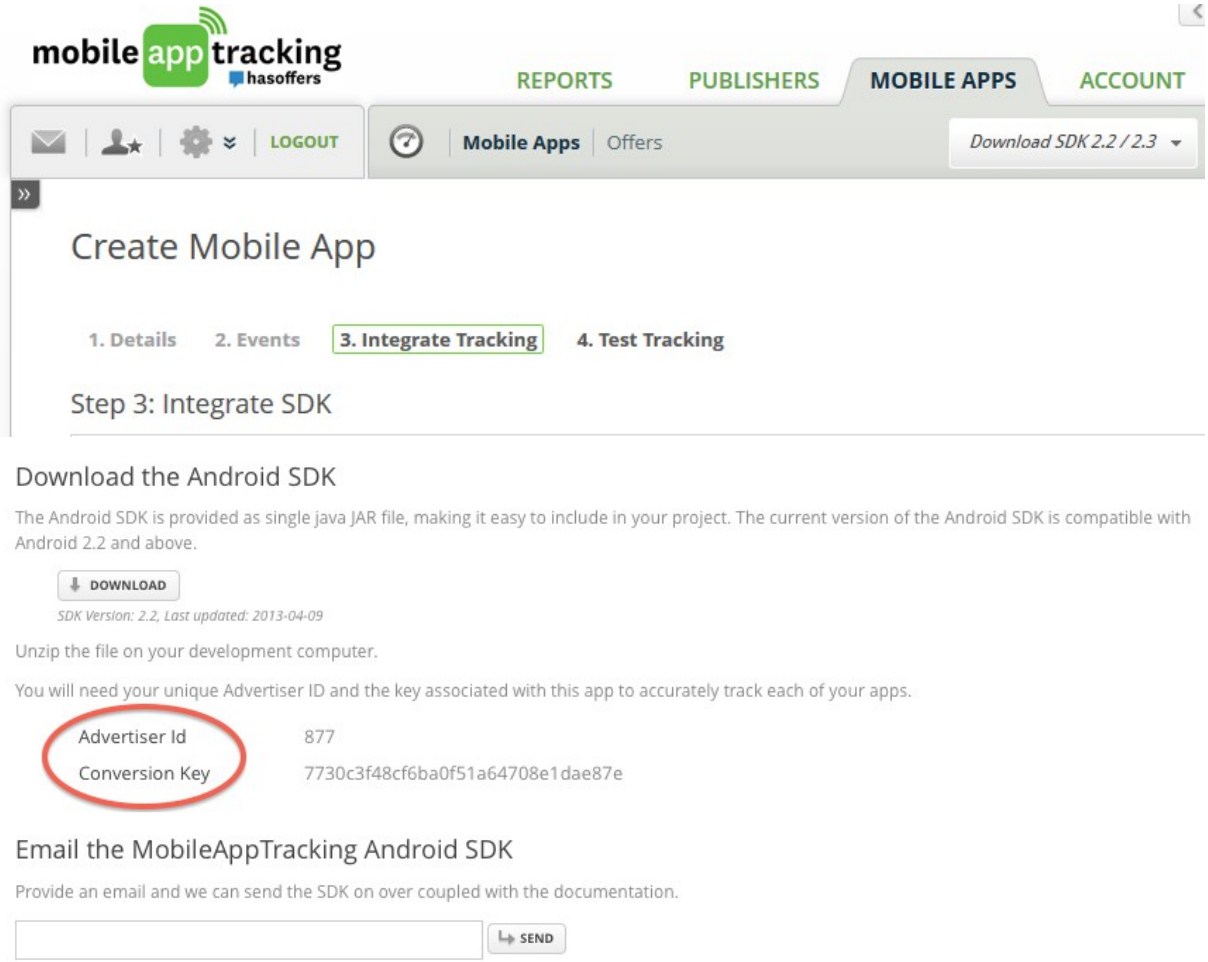
4. Import the constructor, initNativeCode, as indicated above.

5. You'll then need to call the initNativeCode function to instantiate a MobileAppTracker class. Choosing where to instantiate a new class is a decision that is unique to your application/code design, but one example (which we use in the sample code) is to call it in the Awake() function of an empty GameObject in our startup scene:

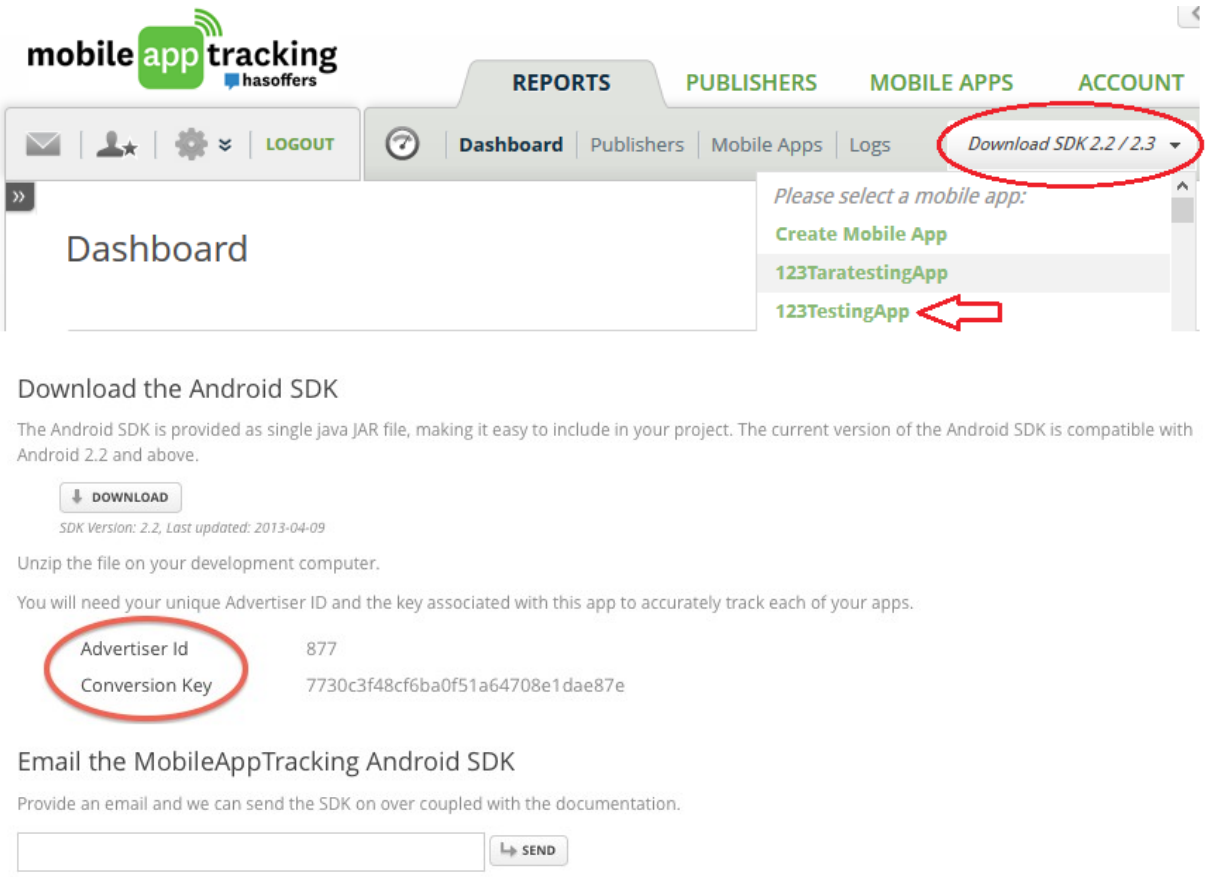
```
void Awake () {

  initNativeCode("your_advertiser_id", "your_advertiser_key");

  return;

    }
```

You will need to pass in the advertiser ID and key associated with your app in MobileAppTracking to the initNativeCode constructor.

The “your_advertiser_id” and the “your_key” values correlate to the **Advertiser Id** and **Conversion Key** provided to you when you created the Mobile App (Step 3 in “Create Mobile App”) in platform. See screenshots below for reference.



These values may also be found on the “Tracking Code for XXXApp” page by clicking on the “Download SDK 2.2/2.3” button and clicking on the appropriate mobile app. See screenshots below for reference.



Platform-Specific Settings

Android

Place libmobileapptracker.so and the Android SDK MobileAppTracker.jar in an Assets/Plugins/Android folder of your Unity project.

Configure AndroidManifest.xml:

To set the Android manifest for your Unity app on Android, create an AndroidManifest.xml file in the Assets->Plugins->Android folder of your project. If you're not sure what this should look like, the default manifest can be found at:

"Unity3/Editor/Data/PlaybackEngines/androidplayer/AndroidManifest.xml"

The SDK requires setting up a MobileAppTracker receiver in your Android manifest. Put this receiver inside your application tags.

Install Referral (Required):

```
<receiver android:name="com.mobileapptracker.Tracker" android:exported="true">
   <intent-filter>
       <action android:name="com.android.vending.INSTALL_REFERRER" />
   </intent-filter>
</receiver>
```

For more information on how MobileAppTracking uses the INSTALL_REFERRER, please see How Google Play Install Referrer Works .

If your app has multiple receivers for INSTALL_REFERRER, you will need to write a custom receiver that will call these receivers instead. Learn how to setup multiple Android install referrers.

Before closing the manifest tag, add the following permissions in as the SDK uses them.

a. **Internet Permission** (Required):

Internet permission is required to connect to tracking servers.

```
<uses-permission android:name="android.permission.INTERNET" />
```

b. **Offline Tracking Permission** (Required):

These permissions enable the SDK to queue tracking events while the user is not connected to the Internet. Once the user is online, the SDK will process all queued events.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

c. **Wifi State Permission** (Optional):

These permissions enable the SDK to access information about whether you are connected to a Wi-Fi network and obtain the device's MAC address. If not used, initialize MobileAppTracker with "collectMacAddres" set to false.

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

d. **Phone State Permission** (Optional):

Allows the user's device ID to be recorded. If not used, initialize MobleAppTracker with "collectDeviceId" set to false.

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Installs and Updates

As the success of attributing app events after the initial install is dependent upon first tracking that install, we require that the install is the first event tracked. To track install of your iOS mobile app, use the “trackInstall” method. If users have already installed your app prior to SDK implementation, then these users should be tracked as updates.

Track Installs

To track installs of your mobile app, use the Track Install method. Track Install is used to track when users install your mobile app on their device and will only record one conversion per install in reports. We recommend calling trackInstall() in the first scene’s Awake method after instantiating a MobileAppTracker class with initNativeCode.

```
[DllImport ("mobileapptracker")]

private static extern void trackInstall();



            trackInstall();
```

The “trackInstall” method automatically tracks updates of your app if the app version differs from the last app version it saw.

Handling Installs Prior to SDK Implementation - Track as Updates

What if your app already has thousands or millions of users prior to SDK implementation? What happens when these users update the app to the new version that contains the MAT SDK?

MAT provides you two ways to make sure that the existing users do not count towards new app installs.

1. Call SDK method "trackUpdate" instead of "trackInstall".

If you are integrating MAT into an existing app where you have users you've seen before, you can track an update yourself with the trackUpdate() method.

```
[DllImport ("mobileapptracker")]

private static extern void trackUpdate();



            trackUpdate();
```

2. Import prior installs to the platform.

These methods are useful if you already have an app in the Apple App Store and plan to add the MAT SDK in a new version. Learn how to handle installs prior to SDK implementation here.

If the code used to differentiate installs versus app updates is not properly implemented, then you will notice a spike of total installs on the first day of the SDK implementation.

Events

After the install has been tracked, the "trackAction" method is intended to be used to track user actions such as reaching a certain level in a game or making an in-app purchase. The "trackAction" method allows you to define the event name dynamically.

All events have the same import code:

```
#if UNITY_ANDROID
    [DllImport ("mobileapptracker")]
    private static extern void trackAction(string eventName, bool   isId, double revenue,
string currencyCode);
#endif
```

```
#if UNITY_IPHONE
    [DllImport ("__Internal")]
    private static extern void trackAction(string eventName, bool isId,  double revenue,
string  currencyCode);
#endif
```

Registration

If you have a registration process, its recommended to track it by calling trackAction set to “registration”.

Using the aforementioned import code, you would call the following:

```
trackAction("registration", false, 0, “USD”);
```

You can find these events in the platform by viewing Reports > Event Logs. Then filter the report by the “registration” event.

While our platform always blocks the tracking of duplicate installs, by deafult it does not block duplicate event requests. However, a registration event may be an event that you only want tracked once per device/user. Please see block duplicate requests setting for events for further information.

Purchases

The best way to analyze the value of your publishers and marketing campaigns is to track revenue from in-app purchases. By tracking in-app purchases for a user, the data can be correlated back to the install and analyzed on a cohort basis to determine revenue per install and lifetime value.

Using the aforementioned import code, you would call the following:

```
trackAction("purchase", false, 0.99, “USD”);
```

Track In-App Purchases

The basic way to track purchases is to track an event with a name of purchase and then define the revenue (sale amount) and currency code.

Note: Pass the revenue in as a Double and the currency of the amount if necessary. Currency is set to "USD" by default. See Setting Currency Code for currencies we support.

You can find these events in platform by viewing Reports > Logs > Events. Then filter the report by the “purchase” event.

Track App Store Purchase State

The SDK also allows you to track purchase events that occur inside your app by tying in your events to iTunes in-App Purchase system. Learn about tracking purchase events with Apple iTunes in-App Purchases here.

Opens

The SDK allows you to analyze user engagement by tracking unique opens. The SDK has built in functionality to only track one “open” event per user on any given day to minimize footprint. All subsequent “open” events fired on the same day are ignored and will not show up on the platform.

Using the aforementioned import code, you would call the following:

```
trackAction("open", false, 0, “USD”);
```

You can find counts of Opens by viewing Reports > Mobile Apps. Include the parameter of Opens to see the aggregated count. The platform does not provide logs of Opens. If you track Opens using a name other than "open" then these tracked events will cost the same price as all other events to track.

Other Events

You can track other events in your app dynamically by calling “trackAction”. The “trackAction” method is intended for tracking any user actions. This method allows you to define the event name.

To dynamically track an event, replace “event name or action” with the name of the event you want to track. The tracking engine will then look up the event by the name. If an event with the defined name doesn’t exist, the tracking engine will automatically create an event for you with that name. An Event Name has to be alphanumeric.

```
trackAction(string eventName, bool   isId, double revenue, string currencyCode);
```

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 in the platform that you associate the action with.

You can find these events in platform by viewing Reports Logs Event Logs.

The max event limit per site is 100. Learn more about the max limit of events.

While our platform always blocks the tracking of duplicate installs, by deafult it does not block duplicate event requests. However, there may be other types of events that you only want tracked once per device/user. Please see block duplicate requests setting for events for further information.

Testing Plugin Integration with SDK

These pages contain instructions on how to test whether the SDKs were successfully implemented for the various platforms:

Testing Android SDK Integration

Testing iOS SDK Integration

Debug Mode and Duplicates

Debugging

When the Debug mode is enabled in the SDK, the server responds with debug information about the success or failure of the tracking requests. Note: For Android, debug mode log output can be found in LogCat under the tag "MobileAppTracker".

To debug log messages that show the event status and server response, call the "setDebugMode" method with Boolean true:

```
[DllImport ("mobileapptracker")]
private static extern void setDebugMode(bool debugMode);



...



setDebugMode(true);
```

Allow Duplicates

The platform rejects installs from devices it has seen before. For testing purposes, you may want to bypass this behavior and fire multiple installs from the same testing device.

There are two methods you can employ to do so: (1) calling the "setAllowDuplicates" method, and (2) set up a test profile.

(1) Call the “setAllowDuplicates” after initializing MobileAppTracker, with Boolean true:

```
[DllImport ("__Internal")]
private static extern void setAllowDuplicateRequests(bool allowDuplicates);


...


setAllowDuplicateRequests(true);
```

(2) Set up a test profile. A Test Profile should be used when you want to allow duplicate installs and/or events from a device you are using from testing and don't want to implement setAllowDuplicateRequests in the code and instead allow duplicate requests from the platform.

*****The setDebugMode and setAllowDuplicates calls are meant for use only during debugging and testing. Please be sure to disable these for release builds.*****

Additional Resources

Custom Settings

The SDK supports several custom identifiers that you can use as alternate means to identify your installs or events. Call these setters before calling the corresponding trackInstall or trackAction code.

OpenUDID (iOS only)

This overwrites the automatically generated OpenUDID of the device with your own value. Calling this will do nothing on Android apps. The official implementation according to: http://OpenUDID.org.

```
setOpenUDID("your_open_udid");
```

TRUSTe ID

If you are integrating with the TRUSTe SDK, you can pass in your TRUSTe ID with setTRUSTeId, to populate the “TPID” field.

```
setTRUSTeId("your_truste_id");
```

User ID

If you have a user ID of your own that you wish to track, pass it in as a string with setUserId. This populates the “User ID” field in our reporting, and also as a postback variable {user_id}.

```
setUserId("custom_user_id");
```

The SDK supports several custom identifiers that you can use as alternate means to identify your installs or events. Please navigate to the Custom SDK Settings page.

Event Items

While an event is like your receipt for a purchase, the event items are the individual items you purchased. Event items allow you to define multiple items for a single event. The “trackAction” method can include this event item data.

When using the function to track events with event items, a public struct has to be defined in the C# script:

```
public struct MATEventItem

                {
                        public string    item;
                        public double     unitPrice;
                        public int        quantity;
                        public double    revenue;
                }
```

```
[DllImport ("__Internal")]

                private static extern void trackActionWithEventItem(string eventName, bool
isId, MATEventItem[] items, int eventItemCount, string refId, double revenue, string
currency, int transactionState);
```

Sample tracking code:

```
MATEventItem item1 = new MATEventItem();

                item1.item = "subitem1";
                item1.unitPrice = 5;
                item1.quantity = 5;
                item1.revenue = 3;

                MATEventItem item2 = new MATEventItem();
                item2.item = "subitem2";
                item2.unitPrice = 1;
                item2.quantity = 3;
                item2.revenue = 1.5;

                MATEventItem[] arr = { item1, item2 };


// Any extra revenue that might be generated over and above the revenues generated from
event items.

                // Total event revenue = sum of even item revenues in arr + extraRevenue
                float extraRevenue = 0; // default to zero

// Transaction state may be set to the value received from iOS/Android app store when a
purchase transaction is finished.

                int transactionState = 1;

                trackActionWithEventItem("eventName", false, arr, arr.Length, null,
extraRevenue, "USD", transactionState);
```

App to App Tracking

App to App tracking provides the ability for one app (the referring app) to download another app (the target app). The target app will then record an install event that contains data from the referring app. Also, you can specify that your app (AppA - referring app) redirect to the link where AppB (target app) can be downloaded (typically this is Google Play or iTunes).

If your app has a referral to another app, upon click of that link you should call "startAppToAppTracking" and pass in the referred app's package name.

With "doRedirect" set to true, the download url will immediately be opened.

```
startAppToAppTracking("com.referred.app", "877", "123", "456", true);
```

If you want to handle this yourself, you can set "doRedirect" to false.

```
void startAppToAppTracking(string targetAppId, string advertiserId, string offerId, string publisherId, bool shouldRedirect)
```

Parameters:

- targetAppId **-** the target package name or bundle ID of the app being referred to
- advertiserId - the advertiser ID of the publisher app in our system
- offerId - the offer ID for referral
- publisherId - the publisher ID for referral
- shouldRedirect - if "true", this method will automatically open the destination URL for the target package name

If supporting Android, you will also need to add a MATProvider to your original app's AndroidManifest.xml file. Place the provider inside thetags with the package names of the apps accessing referral information:

```
<provider android:name="com.mobileapptracker.MATProvider"
          android:authorities="com.referred.app" />
```

List of all Supported Methods

Reference: MATNativeBridge.mm

```
private static extern void initNativeCode(string matAdvertiserId, string
matConversionKey);
private static extern string getSDKDataParameters();
private static extern void setAllowDuplicates(bool allowDuplicates);
private static extern void setCurrencyCode(string currency_code);
private static extern void setDebugMode(bool enable);
private static extern void setDelegate(bool enable);
private static extern void setOpenUDID(string open_udid);
private static extern void setPackageName(string package_name);
private static extern void setRedirectUrl(string redirectUrl);
private static extern void setShouldAutoGenerateMacAddress(bool
shouldAutoGenerate);
private static extern void setShouldAutoGenerateODIN1Key(bool shouldAutoGenerate);
private static extern void setShouldAutoGenerateOpenUDIDKey(bool
shouldAutoGenerate);
private static extern void setShouldAutoGenerateAppleVendorIdentifier(bool
shouldAutoGenerate);
private static extern void setShouldAutoGenerateAppleAdvertisingIdentifier(bool
shouldAutoGenerate);
private static extern void setSiteId(string site_id);
private static extern void setTRUSTeId(string truste_tpid);
private static extern void setUserId(string user_id);
private static extern void setUseCookieTracking(bool useCookieTracking);
private static extern void setUseHTTPS(bool useHTTPS);
private static extern void setAppleAdvertiserIdentifier(string
appleAdvertisingIdentifier);
private static extern void setAppleVendorIdentifier(string appleVendorIdentifier);
private static extern void startAppToAppTracking(string targetAppId, string
advertiserId, string offerId, string publisherId, bool shouldRedirect); private
static extern void setJailbroken(bool isJailbroken);
private static extern void setShouldAutoDetectJailbroken(bool shouldAutoDetect);
private static extern void trackAction(string action, bool isId, double revenue,
string  currencyCode);
private static extern void trackActionWithEventItem(string action, bool isId,
MATEventItem[] items, int eventItemCount, string refId, double revenue, string
currency, int transactionState);
private static extern void trackInstall();
private static extern void trackUpdate();
private static extern void trackInstallWithReferenceId(string refId);
private static extern void trackUpdateWithReferenceId(string refId);
```

```
Reference: Unity application C# Script file
Delegate callback methods:
public void trackerDidSucceed(string data);
private void trackerDidFail(string error);
```