Unity Plugin v1.0

Unity Integration Whitepaper

Ver 1.0 | Jan 23, 2013

Prepared by the **MobileAppTracking Engineering Team**

Unity Plugin v1.0

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.

CONTENTS

1. Getting Started

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.

2. Setup

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

2a. Importing

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")]`

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

E.g.
```
#if UNITY_ANDROID

[DllImport ("mobileapptracker")]
private static extern void initNativeCode(string advertiserId, string advertiserKey);

#endif

#if UNITY_IPHONE

[DllImport ("__Internal")]
private static extern void initNativeCode(string advertiserId, string advertiserKey);

#endif
```

2b. Initialization

Start off by importing the constructor, initNativeCode, as indicated above.

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.

3. Platform-Specific Settings

3a. 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 an INSTALL_REFERRER receiver in your Android manifest. Put this receiver inside your application tags.

Install Referral (Required):
Gives the SDK access to the install referrer value from Google Play.

```
<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 this article:

How Google Play Install Referrer Works

The SDK uses the following permissions, put these in before closing the manifest tag.

Internet Permission (Required):
The Internet permission is required to connect to our platform.
```
        <uses-permission android:name="android.permission.INTERNET" />
```

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" />
```

Wifi State Permission (Required):
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.
```
        <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

Phone State Permission (Required):
Allows the user's device ID to be recorded.
```
        <uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

3b. iOS

Build the Unity app for iOS platform.
Note that a new iPhone Xcode project is generated.

Configure the iPhone project:
Open the iPhone project in Xcode.
In the Project Navigator panel, go to the Classes folder and add the following classes:

```
        MATNativeBridge.h
        MATNativeBridge.mm
        NSData+MATBase64.h
        NSData+MATBase64.m
```

In the Project Navigator panel, select the project. Under the Build Phases tab, open the Link Binary With Libraries drop-down and add the following frameworks:

```
        MobileAppTracker.framework
        AdSupport.framework
        CoreTelephony.framework
        MobileCoreServices.framework
```

The collect UDID setting may be required, depending on the network you are integrating with.

```
        [DllImport ("__Internal")]
        private static extern void setDeviceId(bool enable);

        ...

        mobileAppTracker.setDeviceId(true);
```

When setDeviceId is set to true, the ANE will collect the Apple device UDID. By default the MAT ANE and SDK do not collect the device UDID.

The uniqueIdentifier property of UIDevice class has been deprecated by Apple. If you do not wish to access the device UDID then you can remove the code from the ANE. Since a lot of advertisers still use the device UDID for attribution, we recommend that this setting be enabled to allow you to work with such advertisers. Collecting UDID as device ID on click and then install allows device ID matching to be used for attribution.

4. Debug Mode

To see detailed server responses during debugging, the setDebugMode function can be used.
Our platform typically 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. To enable this, import and call the setAllowDuplicates method after initializing MobileAppTracker, with boolean true as the parameter:

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


        ...

        setDebugMode(true);
        setAllowDuplicates(true);
```

Note:
For Android setAllowDuplicates is auto-enabled along with setDebugMode.
Debug mode also turns on logging messages, available in LogCat under tag “MobileAppTracker”.

These functions should only be used during debugging. In order to capture accurate install numbers, make sure that these function calls are removed before sending the app live.

You can also set up Test Profiles in the platform which will allow a device to create multiple installs.

5. Track Installs and Updates

If your mobile app already has thousands or millions of users prior to implementing the SDK, the platform will record each new user as a new app install when those users updates their app unless you call track update instead of track install.

Without the app developer letting the SDK or platform know that the user already installed the app prior to implementing the SDK, the SDK will track the update as an install. Tracking the update as a new install means potentially attributing the install to a publisher even though they are already user of your app, impacting reporting accuracy and possibly requiring you to pay the publisher for an install you already acquired. It will also prevent you from reconciling with iTunes or Google play reports.

Handling Installs prior to SDK implementation

Developer Setting Updating Flag

The app developer should know if the user has actually installed the app for the first time or if the user is updating the app. Generally, the app developer has set a preference on the user's device upon the initial install that would be present when the user updated their app.

This way, when a user updates their app and it's the first time the MAT SDK interacts with the user, you can forcefully flag installs as updates. After we've recorded the user once, our platform will be able to correctly distinguish future updates as updates.

If this functionality is not implemented and actual app updates are tracked as new installs, then these updates as installs will be attributed to publishers. This will make your company possibly pay for these installs that should be updates so it's important to implement your own logic as described below to tell the SDK if the action should be tracked as an install or an update.

trackInstall can always be called if your app has not been uploaded yet, since all user installs are new users.
For existing apps, check if the user has been seen before and accordingly fire an install or update:

```
if (newUser) {
        trackInstall();
} else {
        trackUpdate();
}
```

Using trackInstall

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();
```

Track Install automatically tracks updates of your app if the app version differs from the last app version it saw.

Using trackUpdate

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();
```

6. Track Events

Track Action is used to track events (other than the install event). The *Track Action* method is intended for tracking user actions like reaching a certain level in a game or making an in-app purchase. This method allows you to define the event name.

Place the code below to track an event. You must replace "**eventName"** with the appropriate value for the event. If the event does not exist, it will be dynamically created in our site and incremented. Pass the revenue in as a double, and the currency code of the amount.

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

trackAction("eventName", false, 0.99, "USD");
```

Parameters:
eventName - the event name associated with the event

isId - whether an event id or event name is being passed in - in most cases, this will be false
revenue - the revenue amount associated with the event
currencyCode - the ISO 4217 currency code for the revenue

If you do not have any revenue data to track, it's acceptable to pass in 0 and "USD" as default values.

7. Track Events With Event Items

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 action, 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);
```

8. Set Custom IDs

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.

Open UDID (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 is 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");
```

9. App to App Tracking

The SDK supports tracking installs between apps that both contain the SDK. When you have one app that refers the install of another app, you can pass in the referrer app’s information to us upon install of the referred app.

If your app has a referral to another app of yours, upon click of that link you should call startAppToAppTracking and passing 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 it 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 the <application> tags with the package names of the apps accessing referral information:

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

10. Test Tracking

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

11. Complete list of supported methods

Reference: MATNativeBridge.mm

```
private static extern void initNativeCode(string advertiserId, string advertiserKey);
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 setDeviceId(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 setShouldAutoGenerateVendorIdentifier(bool
shouldAutoGenerate);
private static extern void setShouldAutoGenerateAdvertiserIdentifier(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 setAdvertiserIdentifier(string advertiserIdentifier);
private static extern void setVendorIdentifier(string vendorIdentifier);
private static extern void startAppToAppTracking(string targetAppId, string
advertiserId, string offerId, string publisherId, bool shouldRedirect);
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);
```