AIR Native Extension v1.0

ANE Integration Whitepaper

Ver 1.0 | Jan 23, 2013

Prepared by the **MobileAppTracking Engineering Team**

Introduction & Table of Contents

AIR Native Extension v1.0

The MobileAppTracking (MAT) ANE for Adobe AIR provides basic application install and event tracking functionality. To track installs, you must integrate the AIR Native Extension with your AIR app. Once the ANE 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 AIR Native Extension integration and use cases.

CONTENTS

1. Build Steps

The following tools are required in order to build the MobileAppTracking ANE:

Adobe AIR
http://get.adobe.com/air/

Adobe Flex SDK
http://www.adobe.com/devnet/flex/flex-sdk-download.html

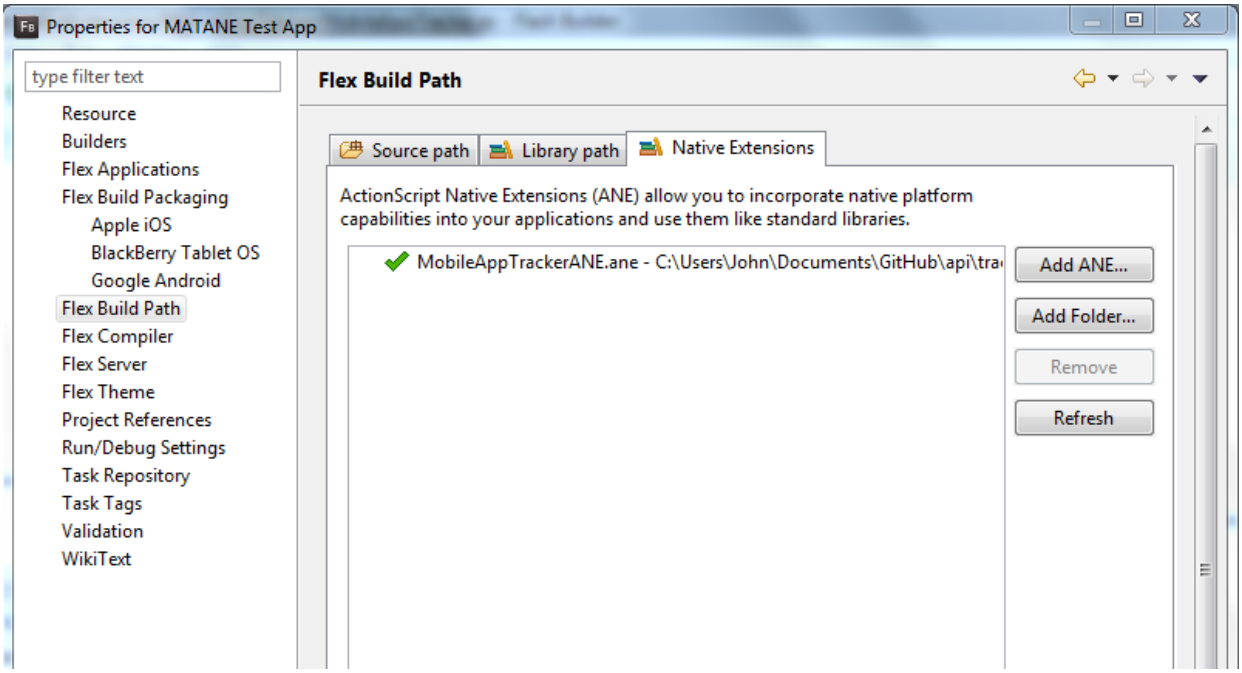
Android SDK
http://developer.android.com/sdk/index.html

In the /build/ folder, open the build.config file and set your system paths for flex.sdk, android.sdk and android.platformtools.

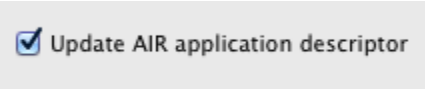
Then from /build/, run 'ant' which will create a MobileAppTrackerANE.ane file in the /bin/ folder.

2. Setup

To integrate the MobileAppTracking ANE in your AIR app, use the file MobileAppTrackerANE.ane located in the /bin/ folder. Add this file as a Native Extension to your AIR app by right clicking your project and selecting Properties->Flex Build Path->Native Extensions->Add ANE.



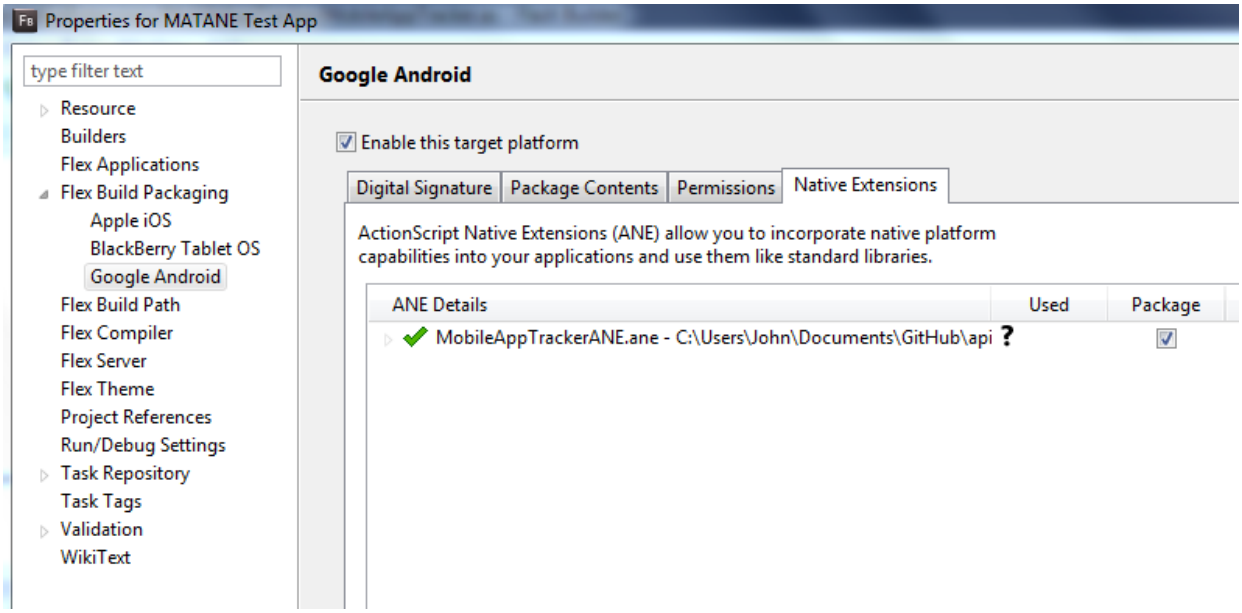
When adding the ANE, if available then you can select the Update AIR application descriptor checkbox.



If not then you can manually add the extension to your project's .xml file like so:

```
<extensions>
	<extensionID>com.mobileapptracker.MobileAppTracker</extensionID>
</extensions>
```

Now enable your platforms from Properties->Flex Build Packaging->Apple iOS/Google Android and select "Enable this target platform". Then go to the Native Extensions tab and check the "Package" box next to the MobileAppTrackerANE ANE.



Now import the MobileAppTracker native extension into your AIR app's Flex script. On app startup, get the singleton instance of MobileAppTracker and then call the init method.

```
import
com.mobileapptracker.nativeExtensions.MobileAppTracker.MobileAppTracker;

public var mobileAppTracker:MobileAppTracker;

…

mobileAppTracker = MobileAppTracker.instance;
mobileAppTracker.init("your_advertiser_id", "your_advertiser_key");
```

You will need to pass in the advertiser ID and key associated with your app in MobileAppTracking to the MobileAppTracker init method.

Android Permissions:

If your app will support Android, add the following permissions to your AIR project's .xml file. It should look like this:

```
<android>
  <manifestAdditions>
    <![CDATA[
      <manifest android:installLocation="auto">
        <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
        <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
        <uses-permission android:name="android.permission.INTERNET"/>
        <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
      </manifest>
    ]]>
  </manifestAdditions>
</android>
```

You will also need to place our INSTALL_REFERRER receiver inside the application tag of your .xml manifest:

```
<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, see this article:

How Google Play Install Referrer Works

iOS Permissions:

If your app will support iOS, this setting may be required, depending on the network you are integrating with. Calling this will do nothing on Android apps, as the Android SDK collects the device IMEI by default.

```
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.

3. 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, call the setAllowDuplicates method after initializing MobileAppTracker, with boolean true as the parameter:

```
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, as an alternative to using setAllowDuplicates.

4. Track Installs and Updates

If your mobile app already has thousands or millions of users prior to implementing the ANE, 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 ANE or platform know that the user already installed the app prior to implementing the ANE , the ANE 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 sets 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 ANE 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 fire an install or update accordingly:

```
if (newUser) {
	mobileAppTracker.trackInstall();
} else {
	mobileAppTracker.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.

```
mobileAppTracker.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.

```
mobileAppTracker.trackUpdate();
```

5. 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.

The trackAction method is used in the following format:

```
trackAction(event:String, revenue:Number=0, currency:String="USD",
refId:String=null, isEventId:Boolean=false):void
```

Notice there are default values if you do not supply the parameters.

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. You may pass a revenue value, currency code, reference id, or whether you are using an event ID or event name, as optional fields.

The reference id is an optional parameter that you supply to use for reconciliation - on a purchase event, it could be their order ID or something else you track. This is called "Advertiser Ref ID" in our reporting, and accessed as {advertiser_ref_id} as a postback variable.

```
mobileAppTracker.trackAction("eventName");
mobileAppTracker.trackAction("eventName", 0.99, "EUR", "123", false);
```

Track Action With Event Item can be used to store optional data you'd like to pass in with the event. The method *trackActionWithEventItem* allows you to pass in an array of mappings of item, unit price, quantity and revenue strings, which we call an "event item" that is associated with the event.

For example, here we add an event item of two bananas for a total of 1.00:

```
var eventItems:Array = new Array();

var dict:Dictionary = new Dictionary();
dict["item"] = "banana";
dict["unit_price"] = "0.50";
dict["quantity"] = "2";
dict["revenue"] = "1.00";

eventItems.push(dict); // you may add any additional event items to the array
                       // formatted like above

// 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;

        mobileAppTracker.trackActionWithEventItem("eventName", eventItems,
extraRevenue, "CAD", null, false, transactionState);
```

6. 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

```
        mobileAppTracker.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.

```
        mobileAppTracker.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}.

```
        mobileAppTracker.setUserId("custom_user_id");
```

7. 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.

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

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

```
startAppToAppTracking(targetAppId:String, advertiserId:String, offerId:String, publisherId:String, shouldRedirect:Boolean):void
```

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 (add as a ManifestAddition in your AIR project's .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" />
```

8. 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

9. Complete list of supported functions

Reference: MobileAppTracker.mm

```
public static function get instance():MobileAppTracker
public function init(advertiserId:String, key:String):void
public function trackInstall(refId:String=null):void
public function trackAction(event:String, revenue:Number=0, currency:String="USD",
refId:String=null, isEventId:Boolean=false):void
public function trackActionWithEventItem(event:String, eventItems:Array,
revenue:Number=0, currency:String="USD", refId:String=null, isEventId:Boolean=false,
transactionState:int=0):void
public function trackUpdate(refId:String=null):void
public function startAppToAppTracking(targetAppId:String, advertiserId:String,
offerId:String, publisherId:String, shouldRedirect:Boolean):void
public function setAdvertiserIdentifier(advertiserIdentifier:String):void
public function setCurrencyCode(currencyCode:String):void
public function setDebugMode(enable:Boolean):void
public function setPackageName(packageName:String):void
public function setRedirectUrl(redirectUrl:String):void
public function setUseCookieTracking(useCookieTracking:Boolean):void
public function setUseHTTPS(useHTTPS:Boolean):void
public function setSiteId(siteId:String):void
public function setTRUSTeId(tpid:String):void
public function setVendorIdentifier(vendorId:String):void
public function setUserId(userId:String):void
public function setDeviceId(enable:Boolean):void
public function setOpenUDID(openUDID:String):void
public function setAllowDuplicates(allowDuplicates:Boolean):void
public function
setShouldAutoGenerateAdvertiserIdentifier(shouldAutoGenerate:Boolean):void
public function setShouldAutoGenerateMacAddress(shouldAutoGenerate:Boolean):void
public function setShouldAutoGenerateODIN1Key(shouldAutoGenerate:Boolean):void
public function setShouldAutoGenerateOpenUDIDKey(shouldAutoGenerate:Boolean):void
public function
setShouldAutoGenerateVendorIdentifier(shouldAutoGenerate:Boolean):void
public function setDelegate(enable:Boolean):void
public function getSDKDataParameters():String
```

Delegate callback methods:

```
public static function onStatusEvent(event:StatusEvent):void
public static function trackerDidSucceed(data:String):void
public static function trackerDidFail(error:String):void
```