MAT Android SDK v2.7

SDK Integration Guide

Version 2.7 | December 2013

Table of Contents

Introduction

The MobileAppTracking (MAT) SDK for Google Android provides basic application install and event tracking functionality. The Android SDK is provided in the form of a single java JAR file that you simply include in your Android project. Our SDK is compatible with all devices running Android 2.1 and above . To track installs, you must first integrate the Android SDK with your Android app. You may also add and track additional events beyond an app install (such as purchases, game levels, and any other user engagement).

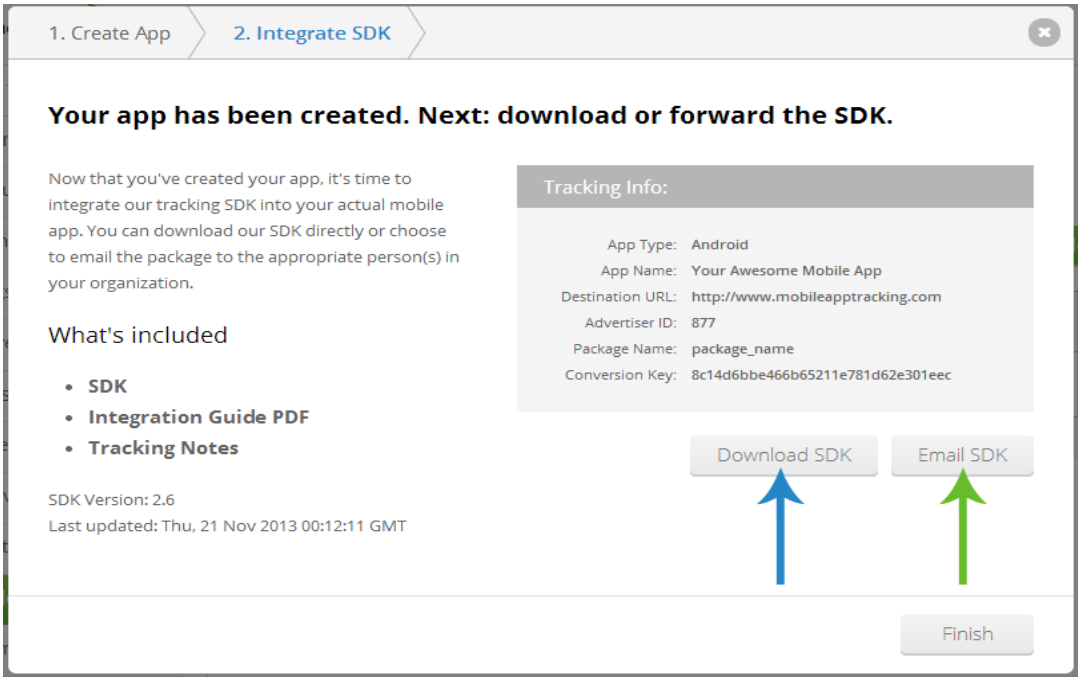
This document outlines the MAT Android SDK integration and use cases.

Compatibility

The current version of the Android SDK is compatible with Android 2.1 and above.

Downloading the Android SDK

There are several methods you can use to download the MAT iOS SDK. The main method of obtaining the MAT iOS SDK is directly downloading it during the second step ("Integrate SDK") of the "Add Mobile App" once you have logged into the MobileAppTracker platform. On this same page, you also have the ability to have the SDK emailed to any email address you provide. See the screenshot below for reference.



Once you've downloaded the Android SDK, decompress the zip file and extract the files to your development computer. The Android SDK is provided as a single java JAR file.

Implementation

1. Configure the AndroidManifest.xml. The Android SDK requires setting up a MobileAppTracker receiver in your Android manifest. Once you have set up the MAT receiver, you will need to place it inside your application tags.

2. Install Referral (required) in your manifest file. Doing so gives the SDK access to the install referrer value from Google Play. Learn how the Google Play Install Referrer works and can be used for attribution.

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

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.

3. 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 omitted, MAC address will not be collected.

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

d. **Phone State Permission** (Optional):
Allows the user's device ID to be recorded. If omitted, IMEI will not be collected.

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

4. Add **MobileAppTracker.jar** to your Android project's build path.

If you are using Eclipse, right click on your project folder:

→ **Properties** → **Java Build Path** → **Libraries** → **Add JARs**.

As of ADT 17.0, you can simply place MobileAppTracker.jar in a folder called "libs" in your project directory and Eclipse will automatically include the jar as part of the build under the classpath container "Android Dependencies" for export.

5. Import the package into your Java class.

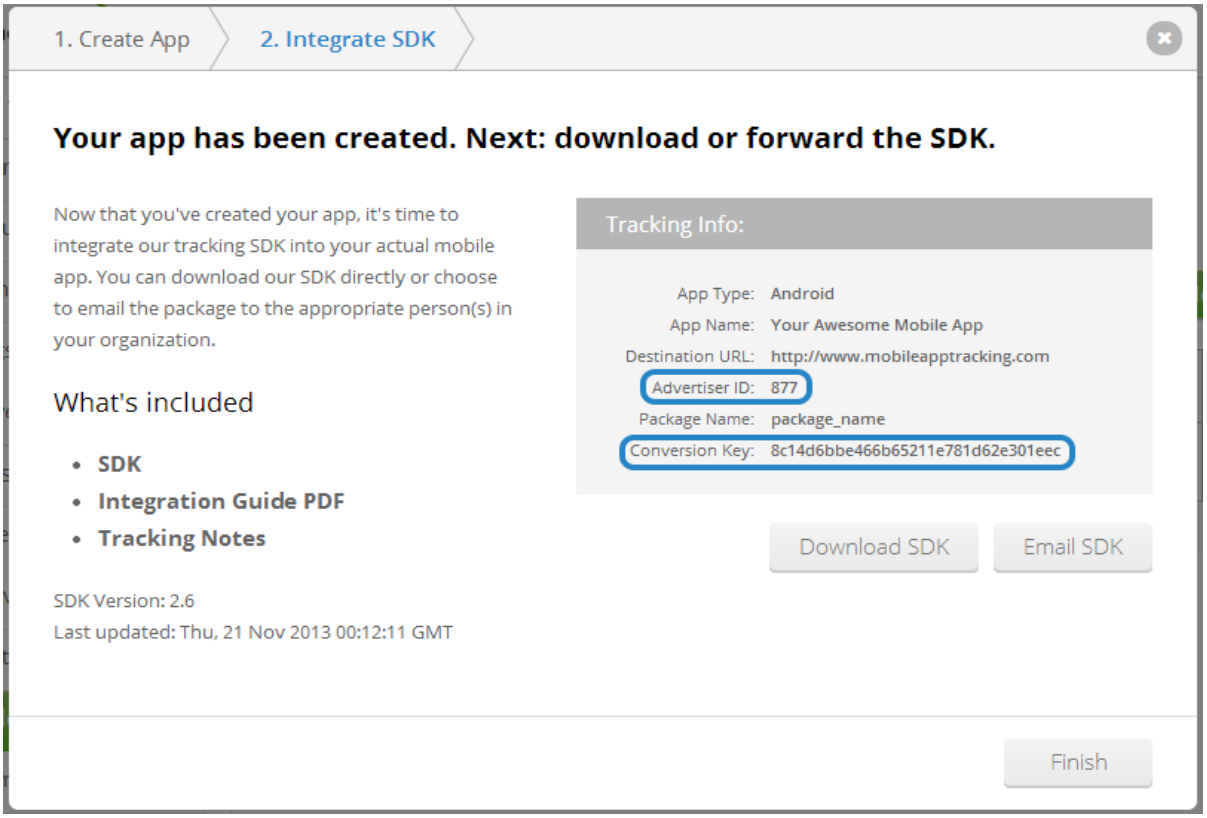
```
    import com.mobileapptracker.*;
```

6. You will then need to instantiate the "MobileAppTracker" class to allow you to call the functions from the Android SDK. Choosing where to instantiate a new class is a decision that is unique to your application/code design. Generally, the code is placed inside your main activity's "onCreate(Bundle)" method.

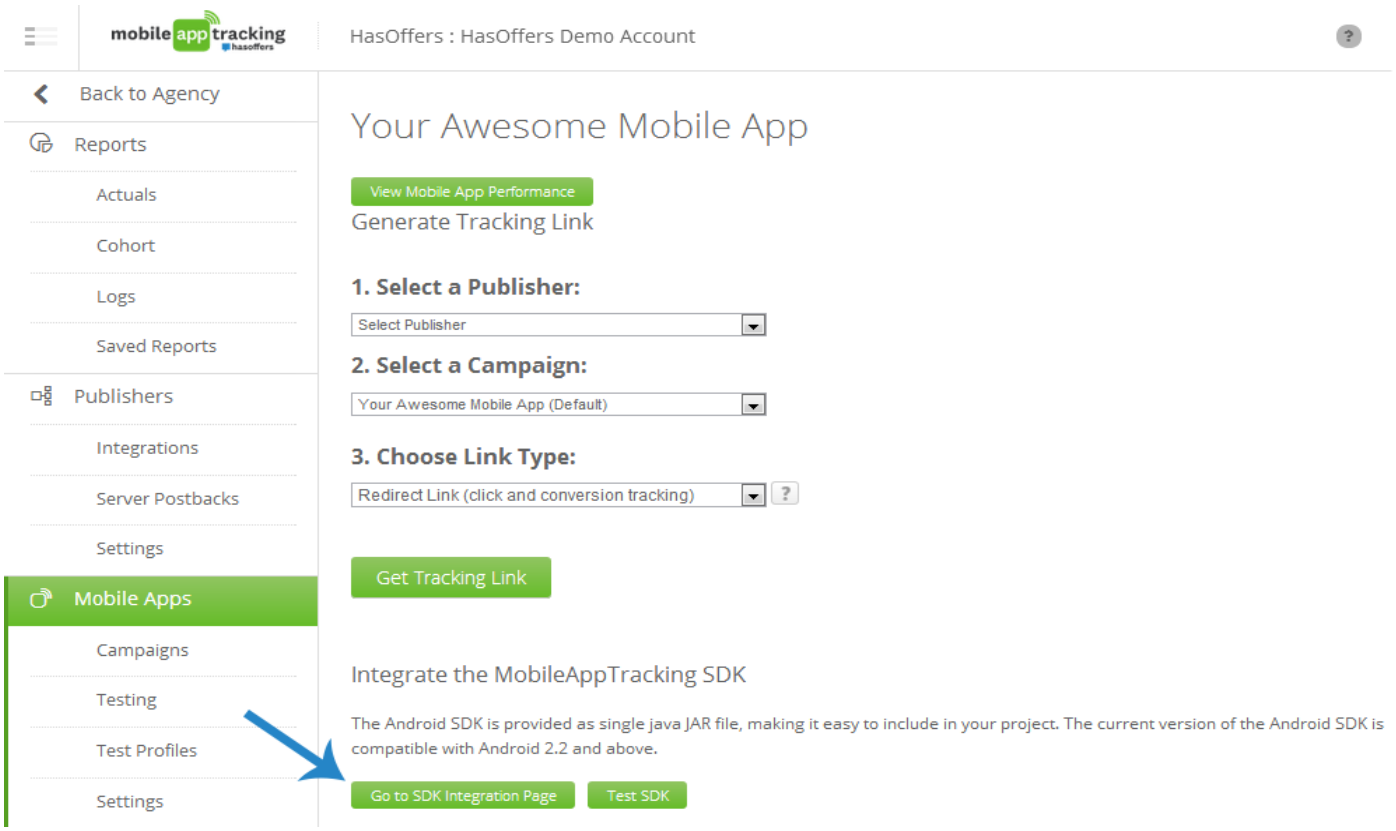
```
public class MyClass extends Activity {
    public MobileAppTracker mobileAppTracker = null;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        mobileAppTracker = new MobileAppTracker(
            getApplicationContext(),
            "your_advertiser_id",
            "your_key");
    }
    //...
}
```

The “MAT_ADVERTISER_ID” and the “MAT_CONVERSION KEY” string constants correlate with the **Advertiser Id** and **Conversion Key** provided to you when you created the Mobile App (Step 2 in “Add Mobile App”) in the platform. See screenshots below for reference.



These values may also be found on the “SDK Integration: XYZ App” page by navigating to your mobile app (Mobile Apps > Your Mobile App > Go to SDK Integration Page) . See screenshots below for reference.



SDK Integration: Your Awesome Mobile App

Back

Download or Forward the SDK

The MobileAppTracking SDK for Google Android provides basic application install and event tracking functionality. To track installs, you must integrate the Android SDK with your Android app. Once the Android SDK 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).

Directions:

Click the download SDK button. Unzip the file on your development computer. Alternatively, you can enter an email address and have the SDK forwarded along with the documentation.

Please Note

You will need your unique Advertiser ID and the key associated with this app to accurately track each of your apps.

Advertiser ID:	**877**
App ID:	**44598**
Package Name:	**app_package_name**
Conversion Key:	**8c14d6bbe466b65211e781d62e301eec**

7. You are now ready to call the functions to track installs and other events. ***IMPORTANT***: Always call "trackInstall" or "trackUpdate" if the user is either installing the app for the first time or updating their app after a previous install.

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 the install of your Android 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 Install

The "trackInstall" method is used to track when users install your mobile app on their device and will only record one conversion per install in reports. You should call trackInstall() in the main activity's onCreate method after instantiating a MobileAppTracker class.

```
mobileAppTracker.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

What if your app already has thousands or millions of users prior to implementing the SDK? After integrating with the MAT SDK, the platform would record each user as a new app install when those users updated their app.

The MAT SDK provides you two methods to force an update event to be tracked instead of an install event:

1. Call SDK method ‘‘trackUpdate" instead of "trackInstall"
2. Import prior installs to the platform

These methods are useful if you already have an app in the Android Market 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.

Sample Install Code

Now that you know how “trackInstall” vs. “trackUpdate” works, the main Activity of your app should now look similar to below.

The sample install code below also includes debugging. For more details on how debugging works, please refer to Debug Mode and Duplicates.

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Instantiate MAT object with your advertiser ID and key
        mobileAppTracker = new MobileAppTracker(getApplicationContext(), "your_advertiser_id",
"your_advertiser_key");

        // Enable these options for debugging only
        if (DEBUG) {
            mobileAppTracker.setAllowDuplicates(true);
            mobileAppTracker.setDebugMode(true);
        }

        // Track install on app open
        mobileAppTracker.trackInstall();
   }
```

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.

Registration

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

```
mobileAppTracker.trackAction("registration");
```

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 default 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.

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.

```
trackAction(java.lang.String eventname, java.lang.Double revenue, java.lang.String currency)
```

Here is an example of tracking an event for purchase with revenue and currency code.

```
mobileAppTracker.trackAction("purchase",1.99, 'USD");
```

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 the platform by viewing Reports > Event Logs. Then filter the report by the “purchase” event.

Purchases with Google's Billing system

The SDK also allows you to track purchase events that occur inside your app by tieing in your events to Google Play's In-App Billing system. To do so, implement a service that is bound to Google's billing system via their instructions. Passing us the IAP data and signature allows us to verify the purchase events.

```
trackAction(java.lang.String eventname, java.lang.Double revenue, java.lang.String currency,
java.lang.String purchaseData, java.lang.String dataSignature)
```

Learn more about tracking purchase events with Google Play's In-App Billing system 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.

To track an "open" event you need to override the "onStart()" inside your main activity. You must pass in "open" as the parameter in the "trackAction" call:

```
    @Override
    protected void onStart() {
        super.onStart();
        mobileAppTracker.trackAction("open");
    }
```

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" 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.

```
mobileAppTracker.trackAction("Event Name");
```

You can find these events in platform by viewing Reports > 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 default 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 SDK

The SDK is designed to track conversions through the Google Play Store as well as from outside marketplaces like the Amazon App Store or third parties. This allows you to test it without updating your production app listing. Subsequently, you will be able to update your mobile app for Google Play after testing that tracking is fully operational - saving you time and resources. Learn how to test the Android SDK integration here.

After you are done testing, you should review the Android SDK Support Checklist before submitting your app with the SDK.

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. 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:

```
mobileAppTracker.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:

```
mobileAppTracker.setAllowDuplicates(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. Its required to disable these for production builds. ******

Reading Platform Responses

You can implement the MATResponse interface to receive and read the platform responses from the SDK calls. Implementing the MATResponse can also be very helpful when debugging the application as it allows you to see platform responses and thus help you with pinpointing the problem.

The following sample code shows a class that implements the MATResponse. Override the "didSucceedWithData" method with your own that does what you would like with the server response. Here, we just output the response to LogCat:

```
public class MyMATResponse implements MATResponse {
    @Override
    public void didSucceedWithData(JSONObject data) {
        Log.d("my tag here", data.toString());
    }
}
```

Instantiate an instance of this class and set it with the MobileAppTracker "setMATResponse" method:

```
MyMATResponse response = new MyMATResponse();
mobileAppTracker.setMATResponse(response);
```

Now when a platform request completes, the code in your custom MATResponse "didSucceedWithData" method will be run - outputting to LogCat in this example.

Version History

v2.7 changes

- Add Facebook, Google, Twitter user id setters
- Add getSDKVersion
- setAppAdTracking renamed to setLimitAdTrackingEnabled
- Recommend passing Application Context over Activity Context

v2.6 changes

- Auto-get referral source package and url
- Make ref id thread-safe, add ref id as overloaded trackAction param
- Fix for in-app purchase validation with IAP data
- Remove collectDeviceId/collectMacAddress options, only collect if required permissions are detected

v2.5 changes

- Add Google Play In-App purchase parameters for verification in trackAction

v2.4.1 changes

- Add setAppAdTracking setter for app-level ad opt-out

v2.4 changes

- Better exception handling and added more informative debug output
- Improve offline event queue synchronization

v2.3 changes

- Add age, gender, latitude, longitude, altitude setters
- Add MATEventItem class for trackAction use

v2.2 changes

- Separate debug and allow duplicates settings
- Add MATResponse interface for reading platform responses
- Show alert dialog warning if debug mode is on
- Add event_referrer setter for app that caused open

v2.1 changes

- Add server response to debug log output
- Make revenue input a Double
- Add site_id setter
- Add "trackPurchase" method for store purchase events

Additional Resources

Custom SDK Settings

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 per a single event. The “trackAction” method can include this event item data. Learn how to track event items.

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). Learn about App to App Tracking here.

Track Referrer Application (Re-Engagement)

Some app ads will open your app if already installed on the device. When your app is launched from some other app - e.g. Safari - you receive a callback in the "UIApplicationDelegate". To track "Re-engagement", you can track the referrer app package name and URL.

Learn how to track Re-Engagement here.

Configure as Shared Instance

Sometimes, it becomes necessary to maintain a shared instance of the MobileAppTracker object throughout multiple activities. The recommended way of doing this is by creating a new shared instance that extends Application. Learn how to Configure the Android SDK as a Shared Instance.

Track Events in WebView

If your app contains HTML content and you would like to track events in the WebView, you can do this by overriding your WebViewClient. Then handle the URL loading to find the URL request that corresponds with your event and call a trackAction before it loads. Learn how to Track Events with SDK in WebView.

Unique Identifiers for Attribution

By default the Android SDK collects Device ID, Android ID, Odin and Mac Address. These Unique Identifiers for Attribution are collected to maximize your ability to work with integrated ad networks and publisher partners.

Methodologies for Attributing Installs

When the platform receives a request to track an app install, it uses three methodologies for

attributing installs, conducted in the following order:

1. Android Install Referrer
2. Unique Identifier Matching
3. Device Fingerprinting

Learn more about Methodologies for Attributing Installs.

App Level Ad Tracking Opt-Out

MAT allows apps to let their users opt-out of app level ad tracking. If a user chooses to opt-out, then the install or conversion event will be used only for ad reporting and not for ad optimization. By default ad tracking is enabled (opt-in).

To opt-out, the following setter method may be called:

```
setLimitAdTrackingEnabled(true);
```