# Oplytic Attribution

*V 2.2*
*September 26, 2018*

Oplytic provides attribution for app-to-app and mobile-web-to-app mobile marketing. Oplytic leverages the tracking provided by Universal Links (iOS) and App Links (Android) to deliver the precise attribution and re-attribution needed for mobile marketers.

# Two-Step Set-Up

## Step 1: Set-Up with Oplytic

Use the API or contact customer service to provide us with the following:

| General | iOS | Android |
|---|---|---|
| • App Name (friendly one-word name) | • Bundle ID<br>• Team ID<br>• iTunes App-Store Link | • Package Name<br>• SHA256 Cert Fingerprints |

After Oplytic receives the above information we will send you your app tracking link. For example, with an App Name of "awesomeapp", your tracking link would be:

- https://awesomeapp.oplct.com/

Append any parameters you want to be stored for attribution. If you are tracking campaign and affiliate, for example, your link might look like:

- https://awesomeapp.oplct.com/?cid={Campaign ID}&affid={Affiliate ID}

## Step 2: Add Oplytic to Your iOS and Android Apps

1) Enable App Links: In XCode or Android Studio.

2) Include the Oplytic Library: Add the project / source files, or reference the binaries (SDK).

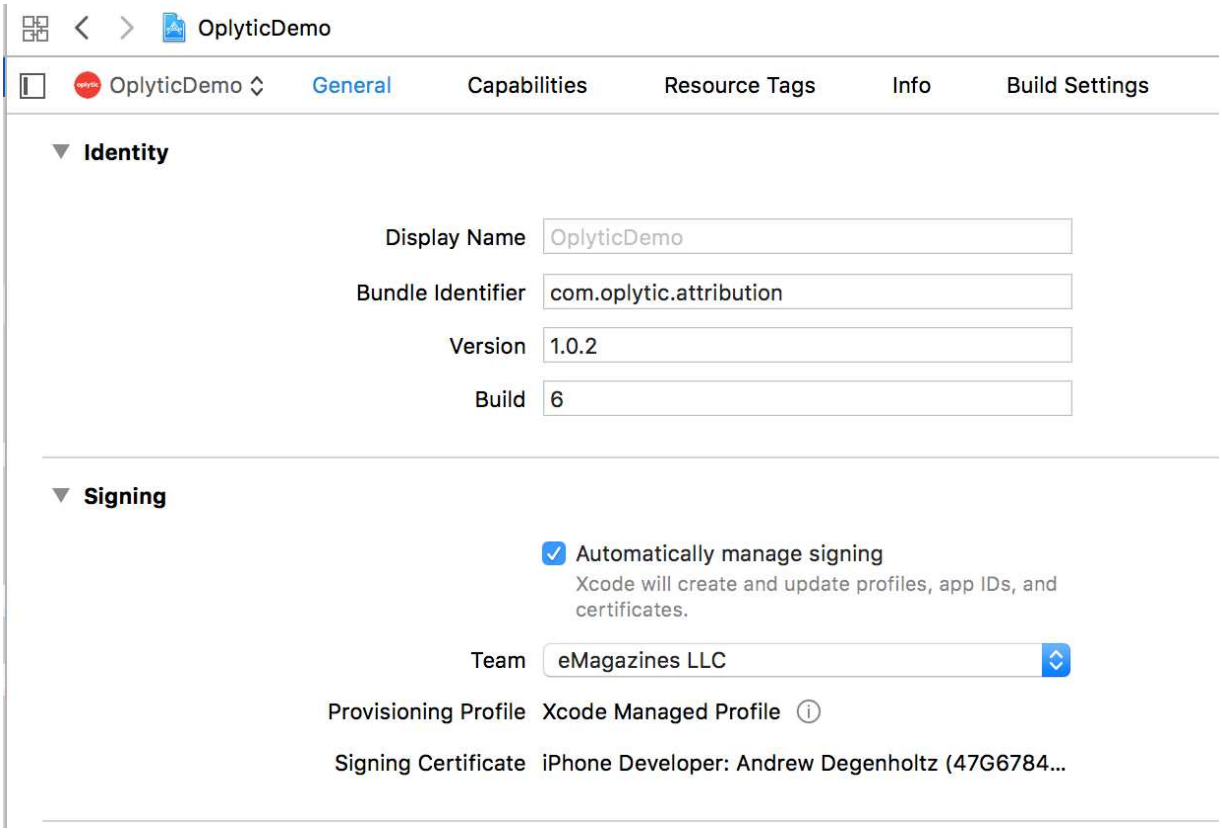
3) Use the Oplytic Library: Attribute app activity by adding just a few lines of code to your app.

# iOS Walk-Through

## Set-Up with Oplytic

Register your app with the Oplytic API or customer service. Simply provide a friendly one-word app name, the Bundle ID and the Team ID.

Grab the Bundle ID and Team ID from the XCode development environment. Look for these settings in the “General” tab, “Identity” and “Signing” sections.



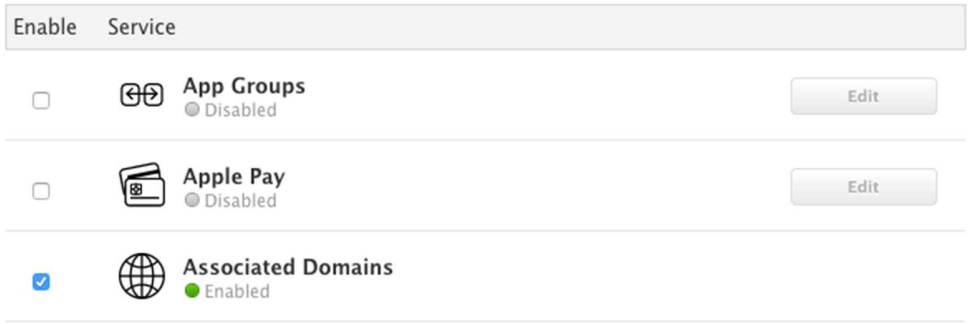

https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html

## Enable App Links

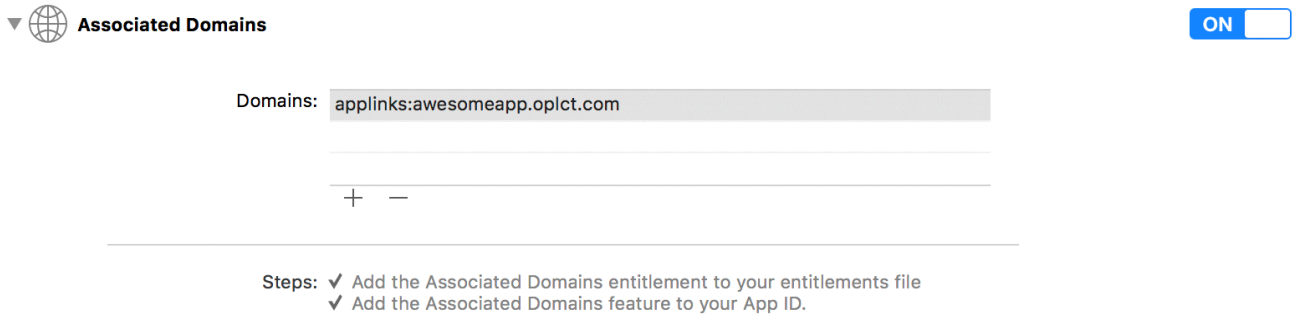
App links enable your app to be launched directly from clicks on safari and other apps. Follow the standard Apple Universal Link scenario:

https://developer.apple.com/ios/universal-links/

In your Apple iTunes developer account, under App IDs, select your application and make sure Associated Domains are enabled.




In XCode, select the project target, then click on the “Capabilities” tab. Scroll down to the “Associated Domains” option. Click on the button to turn it On, and then click on the “+” button to add the following item:




Make sure you specify the App-Name you provided to Oplytic instead of “awesomeapp” above.

NOTE: If you want to access the deep-link-path and URL data yourself, you can access it via the userActivity.webPageUrl attribute.

NOTE: Due to an issue with iOS browser security you cannot enter or copy/paste the above links directly into the Safari URL bar. However, you can embed the link in apps, web-pages, emails, or other forms of social media.

## Include the Oplytic Library

Include the Oplytic SDK Cocoa-Pod found here:

https://cocoapods.org/pods/OplyticSDK
https://github.com/oplytic/OplyticSDK

## Use the Oplytic Library

Be sure to include Oplytic in any files that use the library:

```
Import Oplytic
```

### Start the Oplytic SDK

Initialize the Oplytic SDK in your app delegate class. When the app is launched, start the SDK, passing in the API-Key provided by Oplytic. Handle app-link events as well.

Add the following code to your AppDelegate class:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        Oplytic.start(apiKey: [your_api_key])
        return true
    }

func application(_ application: UIApplication,
                 continue userActivity: NSUserActivity,
                 restorationHandler: @escaping ([Any]?) -> Void) -> Bool {
      Oplytic.handleUniversalLink(userActivity: userActivity)
      return true
}

func application(_ application: UIApplication,
                 willContinueUserActivityWithType userActivityType: String) -> Bool
{
     return userActivityType == NSUserActivityTypeBrowsingWeb
}
```

## 1) Track App-Events

There are just two methods for adding events. Make these calls whenever you want to track a purchase, registration, or other important app event.

The Oplytic SDK also tracks app installs and attribution events automatically. Each time the app is reached via an app-link, the SDK will register a new "last-click" attribution and all subsequent events and purchases will be credited to that link.

**AddEvent** is a general-purpose method:

```
public func addEvent(eventAction: String? = nil,
                     eventObject: String? = nil,
                     eventId : String? = nil,
                     str1: String? = nil, str2: String? = nil, str3: String? = nil,
                     num1 : Double? = nil, num2: Double? = nil)
```

1) *eventAction*: a string associated with the event action, for example, "view" or "shop."
2) *eventObject*: a string associated with the target of the event action, for example "map" or an object SKU.
3) *eventId*: a unique string that you can pass along to associate with the event.
4) *str1, str2, str3*: arbitrary strings associated with the event. You can use these to pass any sort of associated data for that event.
5) *num1, num2*: arbitrary Double numeric values associated with the event. You can use these to pass any sort of associated data for that event.

**AddPurchaseEvent** is used specifically to track in-app purchases:

```
public func addPurchaseEvent(item : String,
                             item_id: String,
                             quantity: Double,
                             price: Double,
                             currency_unit : String)
```

1) *item*: Name of item being purchased.
2) *Item_id*: SKU or other ID associated with the item being purchased
3) *quantity*: Quantity of items being purchased.
4) *price*: Price of item being purchased.
5) *currency_unit*: String value representing the currency, for example: "USD"

## Handle Click Attribution Data (optional)

If your app needs to know about the attributed click, assign an OplyticAttributionHandler protocol, like the simple ViewController example does below:

```
class ViewController: UIViewController, OplyticAttributionHandler {

    override func viewDidLoad() {
        Oplytic.OplyticAttributionHandler = self
        super.viewDidLoad()
    }

    func onAttribution(data: [String:String]){
        //handle Attributed click query params
    }
}
```

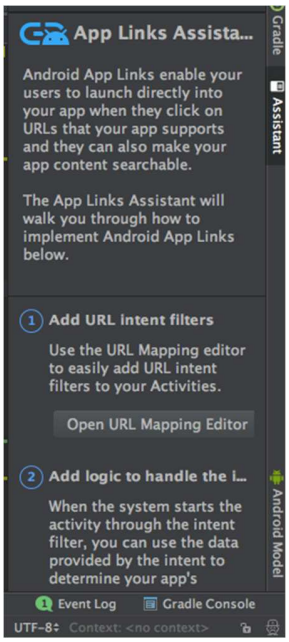
# Android Walk-Through

## Set-up with Oplytic

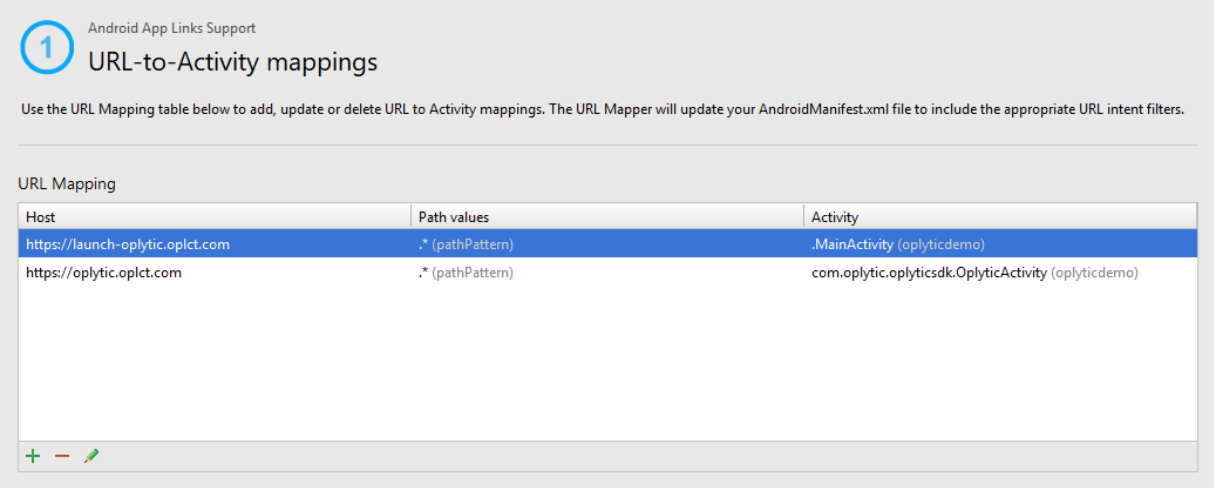
NOTE: The following assumes use of ***Google's Android Studio IDE***.

The Android SDK platform implements deep links via custom Intents.

After receiving your Oplytic link, launch the app-linking wizard via menu *Tools > App Links Assistant*.



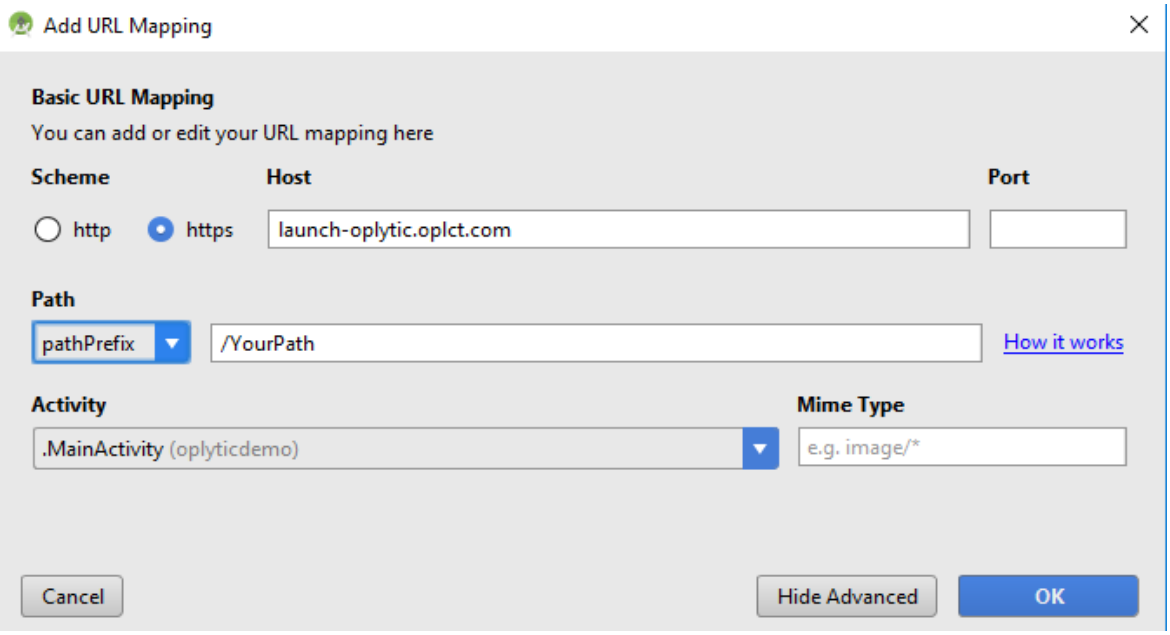

Click on the "Open URL Mapping Editor" button.




Click + > Show Advanced

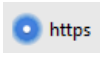
Create deep link paths for your links.
***Be sure to prepend "launch-" In front of the paths for your intent filter.**

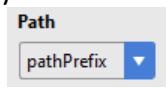
Feel free to customize your deep links with Path Patterns, Path Prefixes or just use a catch all ".*" path pattern to point to all links to your landing page or main activity.




For example, if you want the link to go to your main activity you could setup an intent filter with a pathPrefix. You could enter "/main" as the URL path and select the correct activity for it to point to.

Be sure to:

1. Select 

2. Enter your link host
3. Select 

& enter "/YourPath"
4. Select the correct activity

This will associate the URL with the app and activity. Once you click on "OK" you will see the selected path in the URL mapping screen.

Find the intent filters generated by the URL mappings above in the AndroidManifest.xml file and add the **android:autoVerify="true"** attribute. This will turn your Deep Link into an Android App Link which is a special type of deep link that allows your URLs to immediately open the corresponding content in your Android app without requiring the user to select the app through a disambiguation dialog box for phones with an API > 22.

Clicking different app links from Gmail can spawn multiple instances of your app. This can result in odd behavior. As a solution Oplytic will take the intent and close the new instance and send the intent to the correct deep link location in the existing instance of the app.

Include Oplytic's Activity in the Manifest and replace the host with your host.
***No "launch-" Needed here.***

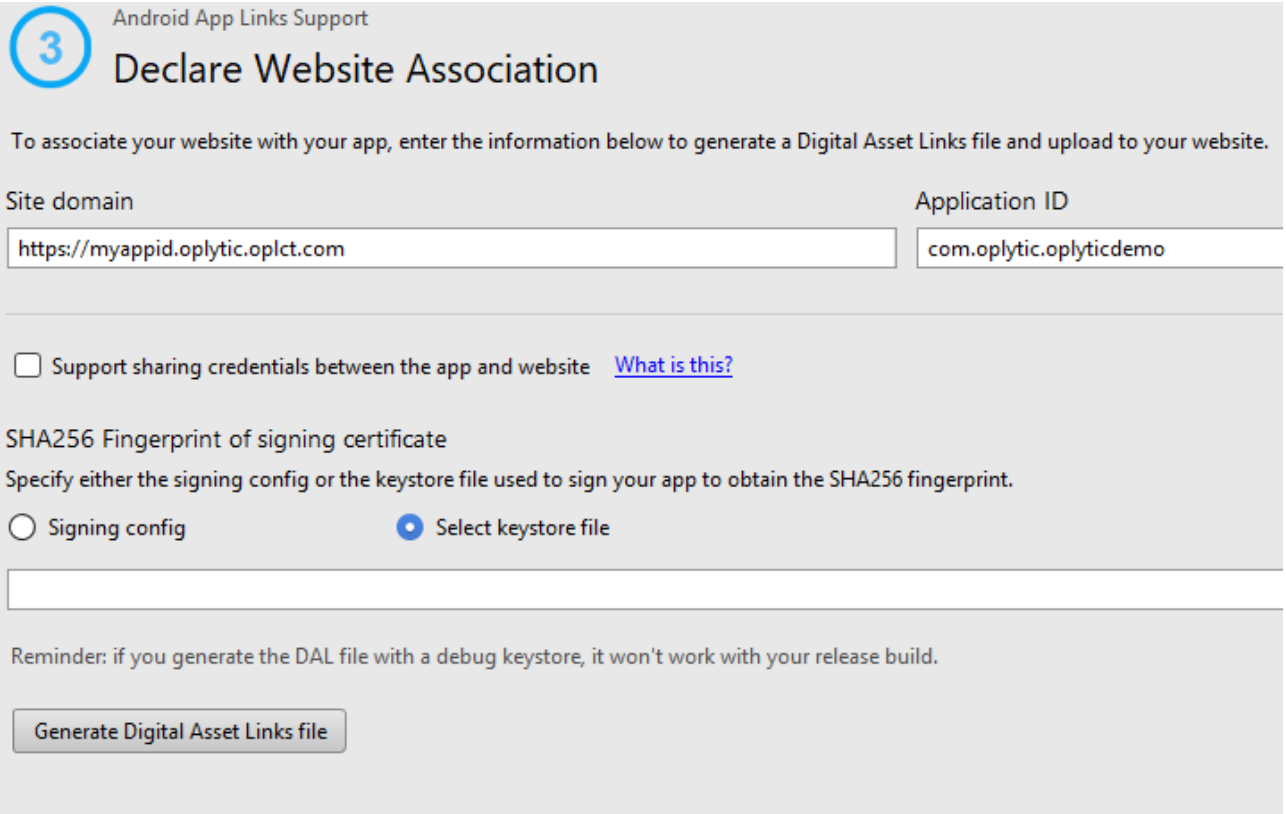
```
<activity
    android:name="com.oplytic.oplyticsdk.OplyticActivity"
    android:launchMode="singleTop">
    <!-- This is for our app links to work -->
    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW" />

        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />

        <data
            android:host="{replace}.oplct.com"
            android:pathPattern=".*"
            android:scheme="https" />

    </intent-filter>
</activity>
```

To do further parsing with the URI use the code below. This can help to direct the user the proper item or location in the activity.

```
Uri uri = getIntent().getData();
```

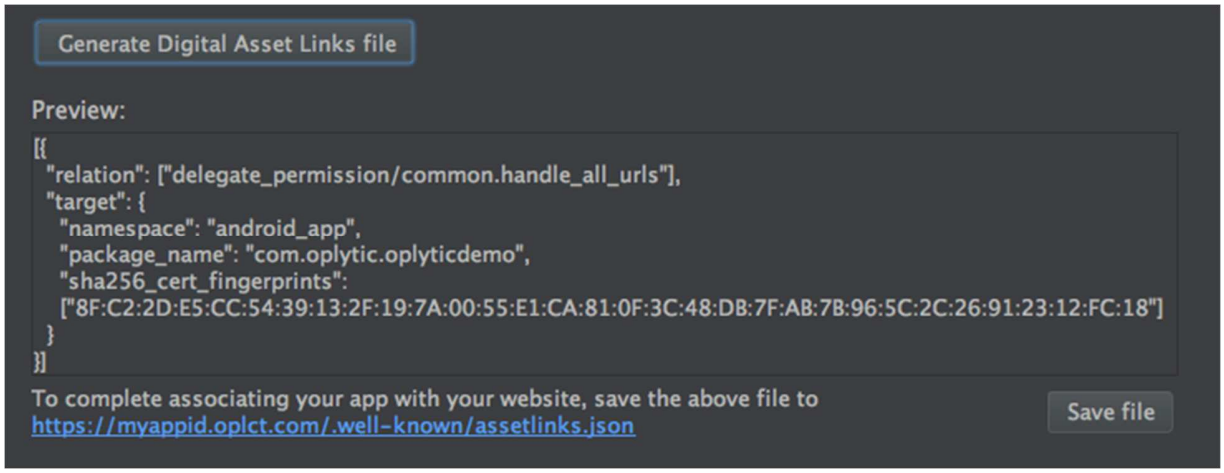
NOTE: Intents and their associated data are cached. If you change the deep-link URLs during development and notice the newer URLs not being used, make sure you indicate that the app intents should not be backed up. In the *AndroidManifest.xml* file, add the following to the *application* tag: <application android:allowBackup="false"…

Next, scroll down to the "Associate Website" section and tap on the "Open Digital Asset Links File Generator" button. In the Site Domain enter the full URL that links to your app. Enter the Application ID for the app if it isn't already specified.




Click on the "Generate Digital Asset Links File" button with your production keystore. This will generate SHA256_cert_fingerprints string to be sent to Oplytic. If you are using Google's app signing you will need to provide that app fingerprint instead which is located in the Google Play Console.

Google Play Console > Select App > Release Management > App Signing




NOTE: Attributed links manually entered into the browser URL may not work. If the default browser is set to Chrome there may be issues with launching deep links. This article provides further information on Android deep-links and Chrome: [https://paul.kinlan.me/deep-app-linking-on-android-and-chrome/](https://paul.kinlan.me/deep-app-linking-on-android-and-chrome/)

To receive App Install events add this code to your BroadcastReceiver. This will allow the SDK to have access to the installation intent for attribution.

```
OplyticSDK.onReceiveInstall(intent);
```

If your app does not use a BroadcastReceiver please include Oplytic's to receive the install broadcast. This can also be used if your broadcast receiver rebroadcasts the intent.

```
<!-- Handling install for attribution -->
        <receiver
            android:name="com.oplytic.oplyticsdk.AppInstallListener"
            android:exported="true"
            android:enabled="true">
            <intent-filter>
                <action android:name="com.android.vending.INSTALL_REFERRER" />
            </intent-filter>
        </receiver>
```

## Include the Oplytic Library

To access the Oplytic SDK import the whole project, source files or SDK jar into your application project and add it to your module build gradle settings:

```
dependencies {
  …
   compile project(":oplyticsdk")
  …
}
```

Be sure to import the Oplytic package in all your app's class files that use it, such as Broadcast Receiver, application class and anywhere you push events.

**import com.oplytic.oplyticsdk.OplyticSDK;**

## Initialize the SDK

**In Your Application Class**:

```
package com.valuemags.offers;

import android.app.Application;
import com.oplytic.oplyticsdk.OplyticSDK;

public class OffersApplication extends Application {
    private static OffersApplication application;

    public static OffersApplication getInstance() {
        return application;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        application = this;

        OplyticSDK.start(application, null);
    }
}
```

**Or Oplytic's Application Class in the android manifest (If you are not using your own)**

```
<application
    android:name="com.oplytic.oplyticsdk.OplyticApplication"
…
</application>
```

**NOTE:** When creating the OplyticSDK you may pass in an advertisingId (optionally can be null): to be used as a unique device token. If none are sent, a random UUID will be generated by the SDK for the user. The ADID can be accessed asynchronously from the Google Play services package.

```
OplyticSDK.start(singleton, advertId);
```

# Events

Wherever you need to track app activity, make an event call to Oplytic. The general-purpose method for tracking an event is:

**public static void *addEvent***(String eventAction, String eventObject, String eventId, String str1, String str2, String str3, Double num1, Double num2)

1. ***eventAction***: a string associated with the event action, for example, "view", "shop."
2. ***eventObject*****:** a string associated with the target of the event action, for example "map" or an object SKU.
3. ***eventId*****:** a unique string that you can pass along to associate with the event.
4. ***str1, str2, str3*****:** arbitrary strings associated with the event. You can use these to pass any sort of associated data for that event.
5. ***num1, num2*****:** arbitrary Double numeric values associated with the event. You can use these to pass any sort of associated data for that event.

For purchase events another method is provided. This method ensures that the server can properly track and aggregate purchase events consistently.

**public static void *addPurchaseEvent***(String item, String itemId, **double** quantity, **double** price, String currency_unit)

1. ***item*****:** Name of item being purchased.
2. **itemId:** ID associated with the item being purchased (ex. SKU)
3. ***quantity*****:** Quantity of items being purchased.
4. ***price*****:** Price of item being purchased.
5. ***currency_unit*****:** String value representing the currency of item, for example: "USD"

# Receive Attribution Events (Optional)

If you would like to be notified of attribution events your Application class needs to implement the IAttributionHandler interface. We will pass a map of the parameters on the link that can be accessed in a key-value fashion. We will also pass in the raw URL.

```
public interface IAttributionHandler {

    void onAttribution(Map<String, String> parameters, String deepLinkURL);

}
```

Below is a sample implementation. This must be done in your application class to ensure that you receive install attribution events and the handler is initialized. You can pass in “this” to the: `OplyticSDK.start(singleton, null, this);` as the handler.

```
package com.oplytic.oplyticdemo;

import com.oplytic.oplyticsdk.IAttributionHandler;
import com.oplytic.oplyticsdk.OplyticSDK;

import android.app.Application;

import java.util.Map;


public class DemoApplication extends Application implements IAttributionHandler
{
    private static DemoApplication singleton;

    public static DemoApplication getInstance() {
        return singleton;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        singleton = this;

        OplyticSDK.start(singleton, null, this);
    }

    public void onAttribution(Map<String, String> parameters, String
DeepLinkURL){
        //Access the parameters here
        String cid = parameters.get("cid");
    }

}
```