Android SDK

You can send data from your Android app and Android TV to Treasure Data, using our Android SDK library.

GDPR Compliance

To support compliance with national and global data privacy requirements such as the European General Data Privacy Regulation, our SDK provides methods that control the collection and tracking of personal data and metadata in applications and websites. When your company defines data privacy policies around personal data, you can use these methods in your code to implement default data collection behaviors, and add controls for individuals to use to manage data collection and privacy themselves.

Ensure that your usage of the SDK, including its use that collects personal data, complies with the legal agreement that governs access to and use of the Treasure Data service, including specifically Treasure Data's current Terms of Service, privacy policy, and Privacy Statement for Customer Data.

Prerequisites

- Basic knowledge of Android Development
- Android 2.3, or later. Auto tracking supports only Android devices that run on API 14 (Android 4.0) and higher
- Basic knowledge of Treasure Data
- Access to the Android SDK Releases

Treasure Data recommends that you implement any new features or functionality at your site using the Treasure Data JavaScript SDK version 3 Beta. It manages cookies differently. Be aware when referring to most of these articles that you need to define the suggested event collectors and Treasure Data JavaScript SDK version 3 calls in your solutions.

For example, change //cdn.treasuredata.com/sdk/2.5/td.min.js to //cdn.treasuredata.com/sdk/3.0.0-beta/td.min.js.

Installing Android SDK

The following video demonstrates how to install Android SDK.

Install the Library

Gradle

If you use Gradle, specify Treasure Data in the ~~dependencies~~ directive in the build.gradle.

```
dependencies {
  implementation 'com.treasuredata:td-android-sdk:0.6.0'
}
```

Maven

If you use maven, specify Treasure Data in the ~~dependencies~~ directive in your pom.xml.

```
<dependency>
  <groupId>com.treasuredata</groupId>
  <artifactId>td-android-sdk</artifactId>
  <version>0.6.0</version>
</dependency>
```

This SDK has an example Android application project. The pom.xml would be a good reference.

Jar File

If you don't use Maven, put the td-android-sdk-x.x.x-shaded.jar (get the most recent version) into (YOUR_ANDROID_PROJECT)/libs.

Enable Required Android Permissions

If it's not already present, add the INTERNET permission to your AndroidManifest.xml file. The following entry should appear between the <manifest> .. < /manifest> tags. Refer to the example here.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.treasuredata.android.demo"
  android:versionCode="1"
  android:versionName="1.0" >
  ...

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

Optionally Enable Tracking of Personal Information

To comply with data privacy regulations in various domains, and specifically the EU's GDPR, the Treasure Data Android SDK does not collect certain event metadata that is personally identifiable. Specifically, the following information is not collected by default:

- td_uuid - client's identifier, unique to this installation of this application on this device

The td_uuid is needed if you want to track individual users and analyze their data within and across user sessions, associate the tracked behavior with a real-world individual, and more.

You must review your data collection policy with your company's data privacy officer and legal counsel to determine what if any personal information you should collect. If you decide to enable tracking of individuals, we recommend that you also integrate a consent management system to track individual user opt-ins to tracking.

When you have determined the user consent, you can enable the collection of personal data. For example:

```
td.enableAutoAppendUniqId();
```

In testing, review the information you are collecting to ensure that you have the personal information you intended and nothing more.

Learn more about UUID for Android SDK.

Usage

Initialize the Library at onCreate in Your Application Subclass

For efficient API calls, we highly recommend initializing a ~~TreasureData~~ shared instance at the ~~onCreate()~~ method of your Application subclass.

```
public class ExampleApp extends Application {

  @Override
  public void onCreate() {

   // Initialize Treasure Data Android SDK
   TreasureData.initializeEncryptionKey("RANDOM_STRING_TO_ENCRYPT_DATA");
   TreasureData.disableLogging();
   TreasureData.initializeSharedInstance(this, "YOUR_WRITE_ONLY_API_KEY");
   TreasureData.sharedInstance.setDefaultDatabase("your_application_name");
   TreasureData.sharedInstance.setDefaultTable("your_event_name");
   TreasureData.sharedInstance.enableAutoAppendUniqId();
   TreasureData.sharedInstance.enableAutoAppendModelInformation();
   TreasureData.sharedInstance.enableAutoAppendAppInformation();
   TreasureData.sharedInstance.enableAutoAppendLocaleInformation();
  }
}
```

We recommend that you use a write-only API key for SDK. Follow these steps to get the key:

1. Login to the Treasure Data Console.
2. Visit your Profile page.
3. Insert your password under the 'API Keys' panel.
4. Under 'Write-Only API keys', copy the API key or select **Generate New** and copy the new API key.

Then, you can use a shared instance from anywhere with the ~~TreasureData.sharedInstance()~~ method.

Use the Shared Instance

```
public class ExampleActivity extends Activity {

  public void onDataLoadSomethingFinished(long elapsedTime) {
    Map<String, Object> event = new HashMap<String, Object>();
    event.put("data_type", "something");
    event.put("elapsed_time", elapsedTime);
    TreasureData.sharedInstance().addEvent("events", event);
  }
}
```

Add an Event to Local Buffer

To add an event to a local buffer, you can call ~~TreasureData#addEvent~~ or ~~TreasureData#addEventWithCallback~~ API.

```
View v = findViewById(R.id.button);
v.setOnClickListener(new OnClickListener() {
  @Override
  public void onClick(View v) {

    final Map event = new HashMap<String, Object>();
    event.put("id", v.getId());
    event.put("left", v.getLeft());
    event.put("right", v.getRight());
    event.put("top", v.getTop());
    event.put("bottom", v.getBottom());

    td.addEventWithCallback("testdb", "demotbl", event, new TDCallback() {
      @Override
      public void onSuccess() {
        Log.i("ExampleApp", "success!");
      }

      @Override
      public void onError(String errorCode, Exception e) {
        Log.w("ExampleApp", "errorCode: " + errorCode + ", detail: " + e.toString());
      }
    });

    // Or, simply...
    // td.addEvent("testdb", "demotbl", event);

  }
});
```

Specify the database and table to which you want to import the events. The total length of the database and table must be shorter than 129 chars.

Upload Buffered Events to Treasure Data

To upload events buffered events to Treasure Data, you can call ~~TreasureData#uploadEvents~~ or ~~TreasureData#uploadEventsWithCallback~~ API.

```
findViewById(R.id.upload).setOnTouchListener(new OnTouchListener() {
  @Override
  public boolean onTouch(View view, MotionEvent motionEvent) {

    // You can call this API to upload buffered events whenever you want.
    td.uploadEventsWithCallback(new TDCallback() {
      @Override
      public void onSuccess() {
        Log.i("ExampleApp", "success!");
      }

      @Override
      public void onError(String errorCode, Exception e) {
        Log.w("ExampleApp", "errorCode: " + errorCode + ", detail: " + e.toString());
      }
    });

    // Or, simply...
    // td.uploadEvents();

    return false;
  }
});
```

It depends on the characteristics of your application when to upload and how often to upload buffered events. We recommend the following:

- When the current screen is closing or moving to the background
- When closing the application

The sent events are buffered for a few minutes before they get imported into Treasure Data storage.

Retry Uploading and Deduplication

The SDK imports events in one style with the combination of these features:

- This SDK keeps buffered events by adding unique keys and retries to upload them until confirming the events are uploaded and stored on the server-side (at least once)
- The server side remembers the unique keys of all events within the past 1 hour by default and can prevent duplicate imports.

Deduplication is a best effort system that identifies a duplicate record if a record with the same identifier is seen in the same dataset, within the last hour at most or within the last 4096 records, whichever comes first.

Default Values

Set a default value if you want an event added to a table, a database, or any table or database to automatically set a value for a key. If you have multiple default values set to the same key, the newly added event will have the default value applied and override in the following order:

1. Default value targeting all tables and databases is applied first.
2. Default value targeting all tables in a database are applied.
3. Default value targeting the table to which the event is added will then be applied.
4. Default value targeting the table and database to which the event is added will then be applied.
5. Finally, if the event has a value for the key, that value will override all default values.

Set a Default Value

```
TreasureData.sharedInstance().setDefaultValue(null, null, "key", "Value"); // Targeting all databases and tables
TreasureData.sharedInstance().setDefaultValue("database_name", null, "key", "Value"); // Targeting all tables
of database "database_name"
TreasureData.sharedInstance().setDefaultValue(null, "table_name", "key", "Value"); // Targeting all tables with
"table_name"
TreasureData.sharedInstance().setDefaultValue("database_name", "table_name", "key", "Value"); // Targeting
table "table_name" of database "database_name"
```

Get the Default Value

```
String defaultValue = (String) TreasureData.sharedInstance().getDefaultValue("database_name", "table_name",
"key"); // Get default value for key targeting database "database_name" and table "table_name"
```

Remove the Default Value

```
TreasureData.sharedInstance().removeDefaultValue("database_name", "table_name", "key"); // Only remove default
value targeting database "database_name" and table "table_name"
```

Start/End Session

When you call the ~~TreasureData#startSession~~ method, the SDK generates a session ID that's kept until ~~TreasureData#endSession~~ is called. The session id outputs as a column name "td_session_id". Also, ~~TreasureData#startSession~~ and ~~TreasureData#endSession~~ methods add an event that includes ~~{"td_session_event":"start" or "end"}~~ .

```
@Override
protected void onStart(Bundle savedInstanceState) {
        :
    TreasureData.sharedInstance().startSession("demotbl");
        :
}

@Override
protected void onStop() {
        :
    TreasureData.sharedInstance().endSession("demotbl");
    TreasureData.sharedInstance().uploadEvents();
    // Outputs =>>
    // [{"td_session_id":"cad88260-67b4-0242-1329-2650772a66b1",
    // "td_session_event":"start", "time":1418880000},
    //
    // {"td_session_id":"cad88260-67b4-0242-1329-2650772a66b1",
    // "td_session_event":"end", "time":1418880123}
    // ]
        :
}
```

If you want to handle the following case, use a pair of class methods ~~TreasureData.startSession~~ and ~~TreasureData.endSession~~ for global session tracking:

- The user opens the application and starts session tracking. For example, session~~#0.~~
- The user moves to the home screen and destroys the Activity
- The user reopens the application and restarts session tracking within the default 10 seconds. But you want to deal with this new session as the same session as ~~session#0~~

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
        :
    TreasureData.setSessionTimeoutMilli(30 * 1000); // Default is 10 seconds
}

@Override
protected void onStart() {
        :
TreasureData.startSession(this);
    :
}

@Override
protected void onStop() {
:
TreasureData.endSession(this);
TreasureData.sharedInstance().uploadEvents();
:
}
```

In this case, you can get the current session ID using ~~TreasureData.getSessionId~~

```
@Override
protected void onStart() {
        :
    TreasureData.startSession(this);
    Log.i(TAG, "onStart(): Session ID=" + TreasureData.getSessionId(this));
        :
}
```

Automatically Track App Lifecycle Events

App lifecycle event tracking is optional and not enabled by default. You can track app lifecycle events automatically using : ~~TreasureData#enableAppLifecycleEvent()~~

App lifecycle events include Application Install, Application Open, Application Update. You can disable the individual core events as the following:

- Disable Application Install: ~~disableAppInstalledEvent()~~
- Disable Application Open: ~~disableAppOpenEvent()~~
- Disable Application Update: ~~disableAppUpdatedEvent()~~

Automatically Track In-App Purchase Events

In-app purchase event tracking is optional and not enabled by default. To track in-app purchase events automatically, add this code: ~~TreasureData#enableInAppPurchaseEvent()~~

It outputs the following columns:

- ~~td_android_event~~ : TD_ANDROID_IN_APP_PURCHASE
- ~~td_iap_product_id~~ : productId (Purchase)
- ~~td_iap_order_id~~ : orderId (Purchase)
- ~~td_iap_product_price~~ : price (SKU detail)
- ~~td_iap_quantity~~ : 1
- ~~td_iap_product_price_amount_micros~~ : price_amount_micros (SKU detail)
- ~~td_iap_product_currency~~ : price_currency_code (SKU detail)
- ~~td_iap_purchase_time~~ : purchaseTime (Purchase)
- ~~td_iap_purchase_token~~ : purchaseToken (Purchase)
- ~~td_iap_purchase_state~~ : purchaseState (Purchase)
- ~~td_iap_purchase_developer_payload~~ : developerPayload (Purchase)
- ~~td_iap_product_type~~ : type (SKU detail), inapp for one-time product and subs for subscription
- ~~td_iap_product_title~~ : title (SKU detail)
- ~~td_iap_product_description~~ : description (SKU detail)
- ~~td_iap_package_name~~ : packageName (Purchase)
- ~~td_iap_subs_auto_renewing~~ : autoRenewing (Purchase)
- ~~td_iap_subs_status~~ : Auto detection for subscription (New|Cancelled|Restored|Expired)
- ~~td_iap_subs_period~~ : subscriptionPeriod (SKU detail for subscription)
- ~~td_iap_free_trial_period~~ : freeTrialPeriod (SKU detail for subscription)
- ~~td_iap_intro_price_period~~ : introductoryPricePeriod (SKU detail for subscription)
- ~~td_iap_intro_price_cycless~~ : introductoryPriceCycles (SKU detail for subscription)
- ~~td_iap_intro_price_amount_micros~~ : introductoryPriceAmountMicro (SKU detail for subscription)

This SDK can track in-app purchase events for Android applications using both Google Play Billing Library and In-app Billing with AIDL. You must add the following ProGuard rule to keep AIDL classes using by the SDK to your project if your application is developed using In-app Billing with AIDL api.

```
-keep class com.android.vending.billing.** { *; }
```

Opt Out

Depending on the countries where you sell your app (e.g. the EU), you may need to offer the ability for users to opt-out of tracking data inside your app.

- To turn off auto-tracking application lifecycle events (when application lifecycle event tracking is enabled) : ~~TreasureData#disableAppLifecycleEvent()~~.
- To turn off auto-tracking in-app purchase events(when in-app purchase event is enabled) : ~~TreasureData#disableInAppPurchaseEvent()~~.
- To turn off custom events (the events you are tracking by ~~TreasureData#addEvent~~, ~~TreasureData#addEventWithCallback~~) : ~~TreasureData#disableCustomEvent~~. To turn on it again : ~~TreasureData#enableCustomEvent~~

You can query the state of tracking events by using: ~~TreasureData#isAppLifecycleEventEnabled()~~, ~~TreasureData#isInAppPurchaseEventEnabled()~~ and ~~TreasureData#isCustomEventEnabled()~~. The states have effects across device reboots, app updates, so you can simply call this once during your application.

About Error Codes

~~TreasureData#addEventWithCallback~~ and ~~TreasureData#uploadEventsWithCallback~~ call back ~~TDCallback#onError~~ method with ~~errorCode~~ argument. This argument is useful to know the cause type of error. There are the following error codes.

- ~~init_error~~: The initialization failed.
- ~~invalid_param~~: The parameter passed to the API was invalid

- ~~invalid_event~~: The event was invalid
- ~~data_conversion~~: Failed to convert the data to/from JSON
- ~~storage_error~~: Failed to read/write data in the storage
- ~~network_error~~: Failed to communicate with the server due to a network problem
- ~~server_response~~: The server returned an error response

Additional Configuration

Endpoint

The API endpoint (default: https://in.treasuredata.com/) can be modified using ~~TreasureData.initializeApiEndpoint~~. For example:

```
TreasureData.initializeApiEndpoint("https://specifying-another-endpoint.com");
td = new TreasureData(this, "your_api_key");
```

Encryption Key

If you've set an encryption key via ~~TreasureData.initializeEncryptionKey~~ , our SDK saves the event data as encrypted when called ~~TreasureData#addEvent~~ or ~~TreasureData.addEventWithCallback~~ .

```
TreasureData.initializeEncryptionKey("hello world");
    :
td.addEventWithCallback(...)
```

Default Database

```
TreasureData.sharedInstance().setDefaultDatabase("default_db");
        :
TreasureData.sharedInstance().addEvent("demotbl", …);
```

Automatically Add UUID of the Device to Each Event

UUID of the device is added to each event automatically if you call ~~TreasureData#enableAutoAppendUniqId()~~. This value won't change until the application is uninstalled.

```
td.enableAutoAppendUniqId();
        :
td.addEvent(...);
```

It outputs the value as a column name ~~td_uuid~~.

You can reset the UUID and send ~~forget_device_uuid~~ event with the old UUID using ~~TreasureData#resetUniqId()~~.

Automatically Add UUID to Each Event Record

UUID will be added to each event record automatically if you call ~~enableAutoAppendRecordUUID~~ . Each event has a different UUID.

```
td.enableAutoAppendRecordUUID();
// If you want to customize the column name, pass it to the API
// td.enableAutoAppendRecordUUID("my_record_uuid");
                :
td.addEvent(...);
```

It outputs the value as a column name ~~record_uuid~~ by default.

Automatically Add Advertising Id to Each Event Record

Advertising Id will be added to each event record automatically if you call ~~enableAutoAppendAdvertisingIdentifier~~. You must install Google Play Service Ads (Gradle ~~com.google.android.gms:play-services-ads~~) as a dependency for this feature to work. Users must also not turn on the Limit Ad Tracking feature in their device, otherwise, Treasure Data will not attach Advertising Id to the record. Because of the asynchronous nature of getting Advertising Id, after the enableAutoAppendAdvertisingIdentifier method called, it may take some time for the Advertising Id to be available to be added to the record. However, Treasure Data does cache the Advertising Id in order to add to the next event without having to wait for the fetch Advertising Id task to complete.

```
td.enableAutoAppendAdvertisingIdentifier();
// If you want to customize the column name, pass it to the API
// td.enableAutoAppendAdvertisingIdentifier("my_advertising_id_column");
:
td.addEvent(...);
```

It outputs the value as a column name ~~td_maid~~ by default.

Automatically Add Device Model Information to Each Event

Device model information will be added to each event automatically if you call ~~TreasureData#enableAutoAppendModelInformation~~ .

```
td.enableAutoAppendModelInformation();
                :
td.addEvent(...);
```

It outputs the following column names and values:

- ~~td_board~~ : android.os.Build#BOARD
- ~~td_brand~~ : android.os.Build#BRAND
- ~~td_device~~ : android.os.Build#DEVICE
- ~~td_display~~ : android.os.Build#DISPLAY
- ~~td_model~~ : android.os.Build#MODEL
- ~~td_os_ver~~ : android.os.Build.VERSION#SDK_INT
- ~~td_os_type~~ : "Android"

Automatically Add Application Package Version Information to Each Event

Application package version information will be added to each event automatically if you call ~~TreasureData#enableAutoAppendAppInformation~~ .

```
td.enableAutoAppendAppInformation();
                :
td.addEvent(...);
```

It outputs the following column names and values:

- ~~td_app_ver~~ : android.content.pm.PackageInfo.versionName (from Context.getPackageManager().getPackageInfo())
- ~~td_app_ver_num~~ : android.content.pm.PackageInfo.versionCode (from Context.getPackageManager().getPackageInfo())

Automatically Add Locale Configuration Information to Each Event

Locale configuration information will be added to each event automatically if you call ~~TreasureData#enableAutoAppendLocaleInformation~~.

```
td.enableAutoAppendLocaleInformation();
        :
td.addEvent(...);
```

It outputs the following column names and values:

- ~~td_locale_country~~ : java.util.Locale.getCountry() (from Context.getResources().getConfiguration().locale)
- ~~td_locale_lang~~ : java.util.Locale.getLanguage() (from Context.getResources().getConfiguration().locale)

Use Server-Side Upload Timestamp

If you want to use server side upload timestamp not only client device time that is recorded when your application calls ~~addEvent~~ , use ~~enableServerSide UploadTimestamp~~.

```
        // Use server side upload time as `time` column
        td.enableServerSideUploadTimestamp(true);

        // Add server side upload time as a customized column name
        td.enableServerSideUploadTimestamp("server_upload_time");
```

Profile API

Lookup for profiles via Profile API.

```
// Set your CDP endpoint to:
// https://cdp.in.treasuredata.com (US)
// https://cdp-tokyo.in.treasuredata.com (Tokyo)
// https://cdp-eu01.in.treasuredata.com (EU)
// https://cdp-ap02.in.treasuredata.com (Seoul)
TreasureData.sharedInstance().setCDPEndpoint("<your_cdp_endpoint>");

TreasureData.sharedInstance().fetchUserSegments(Arrays.asList("<your_profile_api_tokens>"),
                                                                                    Collections.
singletonMap("<your_key_column>", "<value>"),
                                                                                    new
FetchUserSegmentsCallback() {

@Override
                                                                                            public
void onSuccess(List<Profile> profiles) {

System.out.println(profiles);
                                                                                            }

@Override
                                                                                            public
void onError(Exception e) {

System.err.println(e);
                                                                                            }
                                                                                    });
```

Enable and Disable Debug Log

```
                TreasureData.enableLogging();
```

```
                TreasureData.disableLogging();
```

Android Version Support

Android SDK for Treasure Data only supports any Android device running API 15 (Android 4.0) and higher.

Codename	Version	API	Tested
Android 11	11.0	30	Yes
Android 10	10.0	29	Yes
Pie	9.0	28	Yes
Oreo	8.1	27	Yes
Oreo	8.0	26	Yes
Nougat	7.1	25	Yes
Nougat	7.0	24	Yes

Marshmallow	7.1	23	Yes
Lollipop	5.1	22	Yes
Lollipop	5.0	21	Yes
KitKat	4.4	19	Yes
Jelly Bean	4.3	18	Yes
Jelly Bean	4.2	17	No
Jelly Bean	4.2	16	No
Ice Cream Sandwich	4.0	15	No