
Deprecation notice

On November 13th 2018 Google issued the following statement:

We want to let you know that in October 2019 we will begin to sunset our Google Analytics for mobile apps reporting and the Google Analytics Services SDK.

Data collection and processing for such properties will stop on October 31, 2019.

The message is quite clear, and therefore I am officially deprecating this library. If you want to continue using Google's solutions for analytics, I recommend you move to Google Analytics for Firebase instead.

For React Native, there is a great library called react-native-firebase which implements Analytics (and other Firebase solutions).

I will continue to support this library for minor fixes, but no major changes will occur. The repository itself will be archived sometime in 2019.

Thanks to everyone who have used or contributed to this library!

- Christian (@cbrevik)

GoogleAnalyticsBridge build unknown build unknown

Google Analytics Bridge is built to provide an easy interface to the native Google Analytics libraries on both **iOS** and **Android**.

Why a native bridge? Why not use just JavaScript?

The key difference with the native bridge is that you get a lot of the metadata handled automatically by the Google Analytics native library. This will include the device UUID, device model, viewport size, OS version etc.

You will only have to send in a few parameteres when tracking, e.g:

```
1 import { GoogleAnalyticsTracker } from "react-native-google-analytics-bridge";
2 let tracker = new GoogleAnalyticsTracker("UA-12345-1");
3
4 tracker.trackScreenView("Home");
5 tracker.trackEvent("testcategory", "testaction");
```

Version 6 breaking changes!

If you are upgrading to version 6 from an older version, read this wiki post for important details.

The newest version of this library has a new API surface. The API changes are in most cases backwards-compatible.

Important: If you are using ecommerce or custom dimensions, you probably have to migrate to new API if you upgrade!

Content

- Installation
- Manual installation
- Usage
- JavaScript API
- Problems with the library?
- See wiki for more helpful topics

Installation and linking libraries

- For React Native ≥ 0.40 use version 5.0.0 (and up) of this module.
- For React Native < 0.40 use version 4.0.3.

Install with npm: `npm install --save react-native-google-analytics-bridge`

Or, install with yarn: `yarn add react-native-google-analytics-bridge`

Either way, then link with: `react-native link react-native-google-analytics-bridge`

If it doesn't work immediately after this, consult the manual installation guide. Both Android and iOS has a couple of prerequisite SDKs linked and installed.

Important: Does this library work with Expo? We have to sort of invert the question a bit, because it should be: does Expo work with other libraries? And the answer is no:

The most limiting thing about Expo is that you can't add in your own native modules without `detaching` and using ExpoKit.

This includes using `create-react-native-app` which also makes use of Expo.

Usage

```
1 // You have access to three classes in this module:
2 import {
3   GoogleAnalyticsTracker,
4   GoogleTagManager,
5   GoogleAnalyticsSettings
6 } from "react-native-google-analytics-bridge";
7
8 // The tracker must be constructed, and you can have multiple:
9 let tracker1 = new GoogleAnalyticsTracker("UA-12345-1");
10 let tracker2 = new GoogleAnalyticsTracker("UA-12345-2");
11
12 tracker1.trackScreenView("Home");
13 tracker1.trackEvent("Customer", "New");
14
15 // The GoogleAnalyticsSettings is static, and settings are applied
16 // across all trackers:
17 GoogleAnalyticsSettings.setDispatchInterval(30);
18 // Setting `dryRun` to `true` lets you test tracking without sending
19 // data to GA
20 GoogleAnalyticsSettings.setDryRun(true);
21
22 // GoogleTagManager is also static, and works only with one container.
23 // All functions here are Promises:
24 GoogleTagManager.openContainerWithId("GT-NZT48")
25   .then(() => {
26     return GoogleTagManager.stringForKey("pack");
27   })
28   .then(pack => {
29     console.log("Pack: ", pack);
30   })
31   .catch(err => {
32     console.log(err);
33   });
34
35 // You can also register Function Call tag handlers when the container
36 // is open.
37 GoogleTagManager.registerFunctionCallTagHandler(
38   "some_function", // Must be equal to Function Name field when the tag
39   // was configured.
40   (functionName, tagArguments) => {
41     // functionName is passed for convenience. In this example it will
42     // be equal to "some_function".
43     // tagArguments is an object and is populated based on Tag
44     // configuration in TagManager interface.
45     console.log("Handling Function Call tag:", functionName);
46   }
47 )
```

JavaScript API

Table of Contents

- GoogleAnalyticsSettings
 - setOptOut
 - * Parameters
 - * Examples
 - setDispatchInterval
 - * Parameters
 - * Examples
 - setDryRun
 - * Parameters
 - * Examples
- GoogleAnalyticsTracker
 - Examples
 - trackScreenView
 - * Parameters
 - * Examples
 - trackEvent
 - * Parameters
 - * Examples
 - trackTiming
 - * Parameters
 - * Examples
 - trackException
 - * Parameters
 - * Examples
 - trackSocialInteraction
 - * Parameters
 - * Examples
 - setUser
 - * Parameters
 - * Examples
 - setClient

-
- * Parameters
 - * Examples
 - getClientId
 - * Examples
 - allowIDFA
 - * Parameters
 - * Examples
 - setAppName
 - * Parameters
 - * Examples
 - setAppVersion
 - * Parameters
 - * Examples
 - setAnonymizeIp
 - * Parameters
 - * Examples
 - setSamplingRate
 - * Parameters
 - * Examples
 - setCurrency
 - * Parameters
 - * Examples
 - setTrackUncaughtExceptions
 - * Parameters
 - dispatch
 - * Examples
 - dispatchWithTimeout
 - * Parameters
 - * Examples
 - GoogleTagManager
 - Examples
 - openContainerWithId
 - * Parameters
 - * Examples
-

-
- refreshContainer
 - * Examples
 - boolForKey
 - * Parameters
 - * Examples
 - stringForKey
 - * Parameters
 - * Examples
 - doubleForKey
 - * Parameters
 - * Examples
 - pushDataLayerEvent
 - * Parameters
 - * Examples
 - registerFunctionCallTagHandler
 - * Parameters
 - setVerboseLoggingEnabled
 - * Parameters
 - TimingMetadata
 - Parameters
 - Examples
 - EventMetadata
 - Parameters
 - Examples
 - HitPayload
 - Parameters
 - Examples
 - CustomDimensionsByField
 - Examples
 - CustomDimensionsByIndex
 - Examples
-

-
- CustomDimensionsFieldIndexMap
 - Examples
 - CustomMetrics
 - Examples
 - DataLayerEvent
 - Parameters
 - Examples
 - ProductActionEnum
 - Product
 - Parameters
 - Examples
 - ProductAction
 - Parameters
 - Examples
 - Transaction
 - Parameters
 - Examples

GoogleAnalyticsSettings

Settings which are applied across all trackers.

setOptOut Sets if OptOut is active and disables Google Analytics. This is disabled by default. Note: This has to be set each time the App starts.

Parameters

- `enabled` **boolean**

Examples

```
1 GoogleAnalyticsSettings.setOptOut(true);
```

setDispatchInterval Sets the trackers dispatch interval. Events, screen views, etc, are sent in batches to your tracker. This function allows you to configure how often (in seconds) the batches are sent to your tracker. Recommended to keep this around 20-120 seconds to preserve battery and network traffic. This is set to 20 seconds by default.

Parameters

- `intervalInSeconds` **number**

Examples

```
1 GoogleAnalyticsSettings.setDispatchInterval(30);
```

setDryRun When enabled the native library prevents any data from being sent to Google Analytics. This allows you to test or debug the implementation, without your test data appearing in your Google Analytics reports.

Parameters

- `enabled` **boolean**

Examples

```
1 GoogleAnalyticsSettings.setDryRun(true);
```

GoogleAnalyticsTracker

Examples

```
1 // Constructing a tracker is simple:
2 import { GoogleAnalyticsTracker } from "react-native-google-analytics-bridge";
3 const tracker = new GoogleAnalyticsTracker("UA-12345-1");
4 tracker.trackScreenView("Home");
5
6 // You can have multiple trackers if you have several tracking ids
7 const tracker2 = new GoogleAnalyticsTracker("UA-12345-2");
```

```
1 // One optional feature as well is constructing a tracker with a
   CustomDimensionsFieldIndexMap, to map custom dimension field names
   to index keys:
2 const fieldIndexMap = { customerType: 1 };
3 const tracker3 = new GoogleAnalyticsTracker("UA-12345-3", fieldIndexMap);
4
```

```
5 // This is because the Google Analytics API expects custom dimensions
  to be tracked by index keys, and not field names.
6 // Here the underlying logic will transform the custom dimension, so
  what ends up being sent to GA is { 1: 'Premium' }:
7 tracker3.trackScreenView("Home", { customDimensions: { customerType: "
  Premium" } }));
8
9 // If you do not use a CustomDimensionsFieldIndexMap, you will have to
  use index as keys instead for custom dimensions:
10 tracker.trackScreenView("Home", { customDimensions: { 1: "Premium" } })
  ;
```

trackScreenView Track the current screen/view. Calling this will also set the “current view” for other calls. So events tracked will be tagged as having occurred on the current view, `Home` in this example. This means it is important to track navigation, especially if events can fire on different views.

Parameters

- `screenName` **string** (Required) The name of the current screen
- `payload` **HitPayload** (Optional) An object containing the hit payload (optional, default `null`)

Examples

```
1 tracker.trackScreenView('Home');
```

```
1 // With payload:
2 const payload = { impressionList: "Sale", impressionProducts: [ { id: "
  PW928", name: "Premium bundle" } ] };
3 tracker.trackScreenView("SplashModal", payload);
```

trackEvent Track an event that has occurred

Parameters

- `category` **string** (Required) The event category
- `action` **string** (Required) The event action
- `eventMetadata` **EventMetadata** (Optional) An object containing event metadata
- `payload` **HitPayload** (Optional) An object containing the hit payload (optional, default `null`)

Examples

```
1 tracker.trackEvent("DetailsButton", "Click");
```

```
1 // Track event with label and value
2 tracker.trackEvent("AppVersionButton", "Click", { label: "v1.0.3",
    value: 22 });
```

```
1 // Track with a payload (ecommerce in this case):
2 const product = {
3   id: "P12345",
4   name: "Android Warhol T-Shirt",
5   category: "Apparel/T-Shirts",
6   brand: "Google",
7   variant: "Black",
8   price: 29.2,
9   quantity: 1,
10  couponCode: "APPARELSALE"
11 };
12 const transaction = {
13   id: "T12345",
14   affiliation: "Google Store - Online",
15   revenue: 37.39,
16   tax: 2.85,
17   shipping: 5.34,
18   couponCode: "SUMMER2013"
19 };
20 const productAction = {
21   transaction,
22   action: 7 // Purchase action, see ProductActionEnum
23 }
24 const payload = { products: [ product ], productAction: productAction }
25 tracker.trackEvent("FinalizeOrderButton", "Click", null, payload);
```

trackTiming Track a timing measurement

Parameters

- **category** **string** (Required) The event category
- **interval** **number** (Required) The timing measurement in milliseconds
- **timingMetadata** **TimingMetadata** (Required) An object containing timing metadata
- **payload** **HitPayload** (Optional) An object containing the hit payload (optional, default **null**)

Examples

```
1 tracker.trackTiming("testcategory", 2000, { name: "LoadList" }); //
    name metadata is required
```

```
1 // With optional label:
2 tracker.trackTiming("testcategory", 2000, { name: "LoadList", label: "
    v1.0.3" });
```

trackException Track an exception

Parameters

- **error string** (Required) The description of the error
- **fatal boolean** (Optional) A value indicating if the error was fatal, defaults to false (optional, default **false**)
- **payload HitPayload** (Optional) An object containing the hit payload (optional, default **null**)

Examples

```
1 try {  
2     ...  
3 } catch(error) {  
4     tracker.trackException(error.message, false);  
5 }
```

trackSocialInteraction Track a social interaction, Facebook, Twitter, etc.

Parameters

- **network string**
- **action string**
- **targetUrl string**
- **payload HitPayload** (Optional) An object containing the hit payload

Examples

```
1 tracker.trackSocialInteraction("Twitter", "Post");
```

setUser Sets the current userId for tracking.

Parameters

- **userId string** An anonymous identifier that complies with Google Analytic's user ID policy

Examples

```
1 tracker.setUser("12345678");
```

setClient Sets the current clientId for tracking.

Parameters

- **clientId** **string** A anonymous identifier that complies with Google Analytic's client ID policy

Examples

```
1 tracker.setClient("35009a79-1a05-49d7-b876-2b884d0f825b");
```

getClientId Get the client id to be used for purpose of logging etc.

Examples

```
1 tracker.getClientId().then(clientId => console.log("Client id is: ",  
    clientId));
```

Returns **Promise<string>**

allowIDFA Also called advertising identifier collection, and is used for advertising features.

Important: For iOS you can only use this method if you have done the optional step 6 from the installation guide. Only enable this (and link the appropriate libraries) if you plan to use advertising features in your app, or else your app may get rejected from the AppStore.

Parameters

- **enabled** **boolean** (Optional) Defaults to true (optional, default **true**)

Examples

```
1 tracker.allowIDFA(true);
```

setAppName Overrides the app name logged in Google Analytics. The Bundle name is used by default. Note: This has to be set each time the App starts.

Parameters

- **appName** **string** (Required)

Examples

```
1 tracker.setAppName("YourAwesomeApp");
```

setAppVersion Sets the trackers appVersion

Parameters

- `appVersion` **string** (Required)

Examples

```
1 tracker.setAppVersion("1.3.2");
```

setAnonymizeIp Sets if AnonymizeIp is enabled If enabled the last octet of the IP address will be removed

Parameters

- `enabled` **boolean** (Required)

Examples

```
1 tracker.setAnonymizeIp(true);
```

setSamplingRate Sets tracker sampling rate.

Parameters

- `sampleRatio` **number** (Required) Percentage 0 - 100

Examples

```
1 tracker.setSamplingRate(50);
```

setCurrency Sets the currency for tracking.

Parameters

- `currencyCode` **string** (Required) The currency ISO 4217 code

Examples

```
1 tracker.setCurrency("EUR");
```

setTrackUncaughtExceptions Sets if uncaught exceptions should be tracked Important to note: On iOS this option is set on all trackers. On Android it is set per tracker. If you are using multiple trackers on iOS, this will enable & disable on all trackers.

Parameters

- `enabled` **boolean**

dispatch This function lets you manually dispatch all hits which are queued. Use this function sparingly, as it will normally happen automatically as a batch. This function will also dispatch for all trackers.

Examples

```
1 tracker.dispatch().then(done => console.log("Dispatch is done: ", done));
```

Returns **Promise<boolean>** Returns when done

dispatchWithTimeout The same as `dispatch`, but also gives you the ability to time out the Promise in case dispatch takes too long.

Parameters

- `timeout` **number** The timeout. Default value is 15 sec. (optional, default -1)

Examples

```
1 tracker
2   .dispatchWithTimeout(10000)
3   .then(done => console.log("Dispatch is done: ", done));
```

Returns **Promise<boolean>** Returns when done or timed out

GoogleTagManager

Can only be used with one container. All functions returns a Promise.

Examples

```
1 import { GoogleTagManager } from "react-native-google-analytics-bridge"
2   ;
3 GoogleTagManager.openContainerWithId("GT-NZT48")
4   .then(() => GoogleTagManager.stringForKey("pack"))
5   .then(str => console.log("Pack: ", str));
```

openContainerWithId Call once to open the container for all subsequent static calls.

Parameters

- `containerId` **string**

Examples

```
1 GoogleTagManager.openContainerWithId('GT-NZT48').then((..) => ..)
```

Returns **Promise<boolean>**

refreshContainer Refreshes the GTM container. According to Tag Manager documentations for Android can be called once every 15 minutes. No such limitations has been mentioned for iOS containers, though.

Examples

```
1 GoogleTagManager.refreshContainer().then((..) => ..)
```

Returns **Promise<boolean>**

boolForKey Retrieves a boolean value with the given key from the opened container.

Parameters

- `key` **string**

Examples

```
1 GoogleTagManager.boolForKey("key").then(val => console.log(val));
```

Returns **Promise<boolean>**

stringForKey Retrieves a string with the given key from the opened container.

Parameters

- `key` **string**

Examples

```
1 GoogleTagManager.stringForKey("key").then(val => console.log(val));
```

Returns **Promise<string>**

doubleForKey Retrieves a number with the given key from the opened container.

Parameters

- `key` **string**

Examples

```
1 GoogleTagManager.doubleForKey("key").then(val => console.log(val));
```

Returns **Promise<number>**

pushDataLayerEvent Push a datalayer event for Google Analytics through Google Tag Manager. The event must have at least one key “event” with event name.

Parameters

- `event` **DataLayerEvent** An Map<String, Object> containing key and value pairs. It must have at least one key “event” with event name

Examples

```
1 GoogleTagManager.pushDataLayerEvent({
2   event: "eventName",
3   pageId: "/home"
4 }).then(success => console.log(success));
```

Returns **Promise<boolean>**

registerFunctionCallTagHandler Register Function Call tag handler

Parameters

- `functionName` **String**
- `handler` **Function**

setVerboseLoggingEnabled Sets logger to verbose, default is warning

Parameters

- `enabled` **boolean**

TimingMetadata

Used when tracking time measurements

Parameters

- **name** **string** (Required)
- **label** **string** (Optional)

Examples

```
1 const timingMetadata = { name: "LoadList" } // name is a required value
  when tracking timing
2 tracker.trackTiming("testcategory", 13000, timingMetadata);
```

EventMetadata

Used when tracking event

Parameters

- **label** **string** (Optional)
- **value** **number** (Optional)

Examples

```
1 const eventMetadata = { label: "v1.0.3", value: 22 }
2 tracker.trackEvent("FinalizeOrderButton", "Click", eventMetadata);
```

HitPayload

The HitPayload object and possible values

Used by the different tracking methods for adding metadata to the hit.

Parameters

- **products** **Array<Product>** (Optional) Used for ecommerce
- **impressionProducts** **Array<Product>** (Optional) Used for ecommerce
- **impressionList** **string** (Optional) Used for ecommerce
- **impressionSource** **string** (Optional) Used for ecommerce
- **productAction** **ProductAction** (Optional) Used for ecommerce
- **customDimensions** (**CustomDimensionsByIndex** | **CustomDimensionsByField**) (Optional)
- **customMetrics** **CustomMetrics** (Optional)
- **utmCampaignUrl** **string** (Optional) Used for campaigns

-
- **session string** (Optional) Only two possible values, “start” or “end”. This will either start or end a session.

Examples

```
1 // If you want to do send a purchase payload with an event:
2 const product = {
3   id: "P12345",
4   name: "Android Warhol T-Shirt",
5   category: "Apparel/T-Shirts",
6   brand: "Google",
7   variant: "Black",
8   price: 29.2,
9   quantity: 1,
10  couponCode: "APPARELSALE"
11 };
12 const transaction = {
13   id: "T12345",
14   affiliation: "Google Store - Online",
15   revenue: 37.39,
16   tax: 2.85,
17   shipping: 5.34,
18   couponCode: "SUMMER2013"
19 };
20 const productAction = {
21   transaction,
22   action: 7 // Purchase action, see ProductActionEnum
23 }
24 const payload = { products: [ product ], productAction: productAction }
25 tracker.trackEvent("FinalizeOrderButton", "Click", null, payload);
```

```
1 // If you want to send custom dimensions with a screen view:
2 const customDimensions = {
3   1: "Beta",
4   3: "Premium"
5 };
6 const payload = { customDimensions };
7 tracker.trackScreenView("SaleScreen", payload);
```

CustomDimensionsByField

- **See: CustomDimensionsFieldIndexMap**
- **See: CustomDimensionsByIndex**

A dictionary with custom dimensions values and their (mapped) field name keys. In order to use this and send in custom dimensions by field name, you must have provided a [CustomDimensionsFieldIndexMap](#) when constructing the tracker.

Examples

```
1 const customDimensions = { customerType: "Premium", appType: "Beta",  
    credit: 1200 }  
2 tracker.trackScreenView("Home", { customDimensions });
```

CustomDimensionsByIndex

- **See: CustomDimensionsFieldIndexMap**
- **See: CustomDimensionsByField**

A dictionary with custom dimensions values and their index keys.

Examples

```
1 const customDimensions = { 1: "Premium", 3: "Beta", 5: 1200 }  
2 tracker.trackScreenView("Home", { customDimensions });
```

CustomDimensionsFieldIndexMap

- **See: CustomDimensionsFieldIndexMap**
- **See: CustomDimensionsByField**

A dictionary describing mapping of field names to indices for custom dimensions. This is an optional object used by the tracker.

Examples

```
1 // Create something like:  
2 const fieldIndexMap = { customerType: 1 };  
3 // Construct tracker with it:  
4 const tracker = new GoogleAnalyticsTracker("UA-12345-3", fieldIndexMap)  
    ;  
5 // This allows you to send in customDimensions in the`HitPayload` by  
    field name instead of index:  
6 tracker.trackScreenView("Home", { customDimensions: { customerType: "  
    Premium" } });  
7 // If you do not provide a map, you instead have to send in by index:  
8 tracker.trackScreenView("Home", { customDimensions: { 1: "Premium" } })  
    ;
```

CustomMetrics

A dictionary with custom metric values and their index keys.

Examples

```
1 const customMetrics = { 1: 2389, 4: 15000 }  
2 tracker.trackScreenView("Home", { customMetrics });
```

DataLayerEvent

The Google Tag Manager DataLayerEvent dictionary.

Populate this event-object with values to push to the DataLayer. The only required property is `event`.

Parameters

- `event` **string**

Examples

```
1 const dataLayerEvent = {  
2   event: "eventName",  
3   pageId: "/home"  
4 };  
5 GoogleTagManager.pushDataLayerEvent(dataLayerEvent);
```

ProductActionEnum

Enhanced Ecommerce ProductActionEnum

Used by `ProductAction` when describing the type of product action. The possible values (numbers) are:

- Detail = 1,
- Click = 2,
- Add = 3,
- Remove = 4,
- Checkout = 5,
- CheckoutOption = 6,
- Purchase = 7,
- Refund = 8

Product

Enhanced Ecommerce Product

Used by `HitPayload` when populating product actions or impressions

Parameters

- `id` **string**
- `name` **string**
- `category` **string** (Optional)
- `brand` **string** (Optional)
- `variant` **string** (Optional)
- `price` **number** (Optional)
- `couponCode` **string** (Optional)
- `quantity` **number** (Optional)

Examples

```
1 const product = {  
2   id: "P12345",  
3   name: "Android Warhol T-Shirt",  
4   category: "Apparel/T-Shirts",  
5   brand: "Google",  
6   variant: "Black",  
7   price: 29.2,  
8   quantity: 1,  
9   couponCode: "APPARELSALE"  
10 };
```

ProductAction

Enhanced Ecommerce Product Action

Used by `HitPayload` when describing a product action

Parameters

- `action` **ProductActionEnum**
- `transaction` **Transaction** (Optional)
- `checkoutStep` **number** (Optional)
- `checkoutOption` **string** (Optional)
- `productActionList` **string** (Optional)
- `productListSource` **string** (Optional)

Examples

```
1 const productAction = {  
2   transaction,  
3   action: 7 // Purchase action, see ProductActionEnum  
4 }
```

```
1 const productAction = {  
2   action: 3 // Add action, see ProductActionEnum  
3 }
```

Transaction

Enhanced Ecommerce Transaction

Used by [ProductAction](#) when populating describing a purchase/transaction

Parameters

- **id** **string**
- **affiliation** **string** (Optional)
- **revenue** **number** (Optional - but not really)
- **tax** **number** (Optional)
- **shipping** **number** (Optional)
- **couponCode** **string** (Optional)

Examples

```
1 const transaction = {  
2   id: "T12345",  
3   affiliation: "Google Store - Online",  
4   revenue: 37.39,  
5   tax: 2.85,  
6   shipping: 5.34,  
7   couponCode: "SUMMER2013"  
8 };
```