AppMetrica

AppMetrica

10.06.2024

Yandex

AppMetrica. AppMetrica. Version 1.0

Document build date: 10.06.2024

This volume is a part of Yandex technical documentation.

Copyright Disclaimer

Contact information

Yandex LLC

https://www.yandex.com

Тел.: +7 495 739 7000

Email: pr@yandex-team.ru

16 L'va Tolstogo St., Moscow, Russia 119021

Contents

Installation and initialization

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

The SDK for Android is an AAR library. The library is available in the Maven repository.

This section describes the steps to enable and initialize AppMetrica SDK:

Step 1. Add the library to your project

If you use Gradle for building the app, add this dependency to the `build.gradle` file:

```
dependencies {
    // AppMetrica SDK.
    implementation 'com.yandex.android:mobmetricalib:5.3.0'
}
```

Step 2. Initialize the library

Attention: You should take into account some features of the AppMetrica library during initialization. For more information, see Features of the AppMetrica library.

Initialize the library in the app and set up user activity tracking. Extend the `Application` class and override the `onCreate()` method as follows:

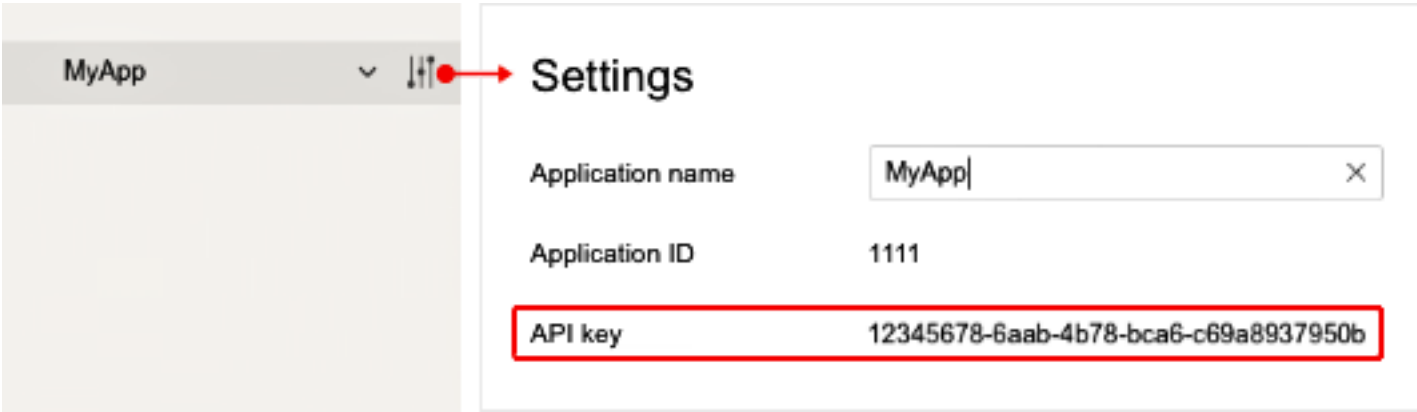
```
public class MyApp extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        // Creating an extended library configuration.
        YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key).build();
        // Initializing the AppMetrica SDK.
        YandexMetrica.activate(getApplicationContext(), config);
        // Automatic tracking of user activity.
        YandexMetrica.enableActivityAutoTracking(this);
        }
}
```

What is the API key?

The *API key* is a unique application identifier that is issued in the AppMetrica web interface during app registration.

Make sure you have entered it correctly.



AppMetrica allows tracking pre-installed apps. To use this feature, you should initialize the library with the extended configuration.

Optional

When using Firebase Performance Monitoring in Firebase version 31.0.0+, activate FirebaseApp for all processes, including the AppMetrica SDK process. In Application#onCreate(), call FirebaseApp.initializeApp(this) **before activating the AppMetrica SDK**. Otherwise, the AppMetrica SDK won't be activated.

Example:

```
class MainApplication : Application() {

    override fun onCreate() {
        super.onCreate()

        // Init FirebaseApp for all processes
        FirebaseApp.initializeApp(this)

        // Then activate AppMetrica SDK
        YandexMetrica.activate(
            this,
            YandexMetricaConfig.newConfigBuilder(API_KEY)
                .build()
        )
    }
}
```

Step 3. *(Optional)* Configure location detection

Location allows you to estimate the geographical distribution of users. By default, AppMetrica determines the device location by the IP address with accuracy to the country.

To determine the device location down to the city level, open the `AndroidManifest.xml` file and add the `uses-permission` element before the `application` element:

```
<manifest>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <application>...</application>
</manifest>
```

`ACCESS_COARSE_LOCATION` allows you to track the device's location. For more information, see Android documentation.

Step 4. *(Optional)* Configure sending events, profile attributes, and Revenue

To collect information on user actions in the app, set up sending your own events. For more information, see Sending your own events.

To collect information about users, set up sending profile attributes. For more information, see Profiles.

Note: Unlike events, a profile attribute can take only one value. When you send a new attribute value, the old value is overwritten.

To track in-app purchases, set up Revenue sending. For more information, see In-App purchases.

Step 5. Test the library operation

Attention: Before testing the library, ensure that the SDK is initialized in compliance with recommendations.

To test how the library works:

1. Start the app with the AppMetrica SDK and use it for a while.
2. Make sure your device is connected to the internet.
3. In the AppMetrica interface, make sure that:
   - There is a new user in the Audience report.
   - The number of sessions in the **Engagement** → **Sessions** report has increased.
   - There are events and profile attributes in the Events and Profiles reports.

Troubleshooting

The number of sessions does not increase

Check your session tracking settings. For more information, see Tracking user activity.

There are no events in the report

1. Perform a minimum of 10 app actions that trigger the event sending.

   It's necessary because AppMetrica accumulates events in the buffer and sends to the server in several parts.
2. Wait for 10 minutes and check the report. Reports don't display events immediately.

Error when adding the library to a project

```
UNEXPECTED TOP-LEVEL EXCEPTION:
com.android.dex.DexIndexOverflowException: method ID not in [0, 0xffff]: 65536
at com.android.dx.merge.DexMerger$6.updateIndex(DexMerger.java:502)
at com.android.dx.merge.DexMerger.mergeMethodIds(DexMerger.java:491)
at com.android.dx.merge.DexMerger$IdMerger.mergeSorted(DexMerger.java:277)
at com.android.dx.command.dexer.Main.runMonoDex(Main.java:303)
at com.android.dx.command.dexer.Main.mergeLibraryDexBuffers(Main.java:454)
at com.android.dx.merge.DexMerger.mergeDexes(DexMerger.java:168)
at com.android.dx.merge.DexMerger.merge(DexMerger.java:189)
at com.android.dx.command.dexer.Main.run(Main.java:246)
at com.android.dx.command.dexer.Main.main(Main.java:215)
at com.android.dx.command.Main.main(Main.java:106)
```

This error indicates that the method limit was exceeded at the DexIndexOverflowException stage of processing. We recommend reviewing the libraries used — perhaps they are very heavy. If they can't be replaced with lightweight alternatives, you can use multiple DEX files. This might increase the app loading time.

Error initializing AppMetrica with third-party libraries

The code in the `Application.onCreate()` method runs for all processes. If you encounter an initialization error after integrating a third-party library (such as Firebase Cloud Messaging), make sure that the third-party library is initialized only in the main process.

Error examples:

- ```
  Unable to create application your.package.name.YourApp: java.lang.IllegalStateException: Default FirebaseApp is
   not initialized in this process your.package.name:Metrica. Make sure to call FirebaseApp.initializeApp(Context)
   first.
  ```
- ```
  android.database.sqlite.SQLiteException: table httpauth already exists (code 1)
  ```
- ```
  Fatal Exception: java.lang.RuntimeException: Using WebView from more than one process at once with the same data
   directory is not supported.
  ```
- ```
  ANR at com.google.android.gms.ads.*
  ```

To fix the error, implement the main process check before initializing third-party libraries:

```
class MyApplication extends Application {
    @Override
    public void onCreate() {
        if (isMainProcess()) {
            // Initializing third-party libraries after verification.
        }
    }
}
```

Note: Implement verification of the main process on your own. See an example in the Stack Overflow answer.

Incorrect duration of user session during manual tracking

If the manual tracking of the user session is incorrectly implemented, it may lead to an inaccurate determination of its duration.

If the user session duration data doesn't look correct, make sure that when the user session ends, the YandexMetrica.pauseSession() method is always called. If this method is not called, the library considers that the session is active and commits regular data exchange with the server part of AppMetrica.

It is recommended to check the duration of the session timeout. It is specified with the `withSessionTimeout()` method. The timeout specifies the time interval during which the session will be considered active even after the application is closed.

High power consumption of the AppMetrica library

If there is the increased power consumption of the library, make sure that when the user session ends, the `YandexMetrica.pauseSession()` method is always called. If this method is not called, the library considers that the session is active and commits regular data exchange with the server part of AppMetrica.

It is recommended to check the duration of the session timeout. It is specified with the `withSessionTimeout()` method. The timeout specifies the time interval during which the session will be considered active even after the application is closed.

My problem is not listed

If your problem is not listed, contact support service. Specify the following:

1. The source code snippet that shows the SDK integration to your app.
2. Application ID in the AppMetrica web interface.
3. Device ID.

   How to get the Google AID

   a. Install the AppMetrica app on the test device.
   b. Log in and select your app from the list.
   c. In the upper-left corner, click ☰ → **Device Management**.
   d. The Google AID is shown in the **AID** field. Enter it in the AppMetrica web interface.

   Note: You can enable attribution testing in the AppMetrica app. To do this, turn on **Attribution testing**.

4. Device model and manufacturer, platform and OS version, AppMetrica SDK version.

See also

How to enable user location sending

How can I make sure that I have the latest versions of the Android libraries installed?

Related information

Example of library integration

Error initializing AppMetrica with third-party libraries on page 37

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

List of main dependencies

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

AppMetrica uses various dependencies to work correctly.

com.android.installreferrer:installreferrer

Supported version: 2.2.

Used to receive a referrer from Google Play: https://developer.android.com/google/play/installreferrer/library.

com.google.android.gms:play-services-appset

Supported version: 16.0.0.

Used to receive the AppSetID: https://developer.android.com/training/articles/app-set-id.

org.jetbrains.kotlin:kotlin-stdlib

Supported version: 1.4.32.

Used for AppMetrica development on Kotlin.

How the AppMetrica library works

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

The AppMetrica library consists of two parts: the client and the service. You should take their features into account during initialization:

Client side

Pre-validates data and transmits it to the service part.

An instance of the client part is created for each process in the application.

Service side

It is responsible for storing data and sending it to the server.

The service runs in a separate process. Therefore:

- AppMetrica has a minimal effect on performance and the amount of memory consumed.
- AppMetrica does not depend on the stability of the application. This allows you to save data and send it to the server in case of crashes and unstable operation of the application.

Recommendations on initialization

- If the application has multiple processes, initialize the library with the same configuration for each process. Otherwise, the configuration may depend on which of the processes starts first.

  Note: You can use reporters for sending data. They can be initialized with different configurations. For more information, see Sending statistics to an additional API key.
- Initialize the AppMetrica library in the Application.onCreate() method. This way the library will be initialized in each process.

  Attention: The code in the `Application.onCreate()` method runs for all processes. If you encounter an initialization error after integrating a third-party library (such as Firebase Cloud Messaging), make sure that the third-party library is initialized only in the main process. For more information, see Error descriptions.
- Keep in mind that the ContentProvider instance is created before the Application instance. If you initialize the library in the `Application.onCreate()` method, you can't send data from the `ContentProvider.onCreate()` method.
- Don't add the code to the `Application.onCreate()` method that should not be executed more than once. Run this code when creating other components, or run it in a separate Service.
- Avoid long-term operations in the `Application.onCreate()` method and do not initialize other libraries in it. The execution time of this method directly affects the opening speed of the first Activity, Service, or Receiver.

See also

How to enable user location sending

How can I make sure that I have the latest versions of the Android libraries installed?

Why is the new installation not shown in the reports?

Related information

Example of library integration

Error initializing AppMetrica with third-party libraries on page 37

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Tracking user activity

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

A single AppMetrica session is a certain period of the user interacting with your app.

In a standard scenario, a new session begins if the user returns to your app when a significant amount of time has passed since the app switched to background mode (the user hid the app or opened system settings).

Setting the length of the session timeout

By default, the session timeout is 10 seconds. The minimum acceptable value for the `sessionTimeout` parameter is 10 seconds.

To change the length of the timeout, pass the value in seconds to the withSessionTimeout(int sessionTimeout) method when creating the extended library configuration.

```
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Setting the length of the session timeout.
        .withSessionTimeout(15)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

Monitoring the app lifecycle

If the app has the minimum Android version set to 4.0 or later, the AppMetrica library can track the app lifecycle automatically. After library initialization, call the YandexMetrica.enableActivityAutoTracking(Application application) method:

```
public void YourApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        // Initializing the AppMetrica SDK.
        ...
        // Automatic tracking user activity.
        YandexMetrica.enableActivityAutoTracking(this);
        ...
    }
}
```

Note: Automatic activity tracking only works for the main API key. When sending statistics to an additional API key, use the IReporter interface methods.

If the minimum Android version in the app is earlier than 4.0, use the following methods: YandexMetrica.resumeSessoin(activity) and YandexMetrica.pauseSession(activity) in the corresponding methods for your Activity: onResume() and onPause(). When using these methods you should track that the active session

is always terminated by calling the YandexMetrica.pauseSession(activity) method. If this method is not called, the library considers that the session is active and commits regular data exchange with the server part of AppMetrica. This can lead to incorrect tracking of the session duration and energy consumption.

Example:

```
public class YourActivity extends Activity {
      ...
      @Override
      protected void onResume() {
          super.onResume();
          YandexMetrica.resumeSession(this);
      }

      @Override
      protected void onPause() {
          YandexMetrica.pauseSession(this);
          super.onPause();
      }
      ...
}
```

Working with an additional API key

By sending data to an additional API key, you can control the access to statistics. Use reporters to transmit events. For more information, see Usage examples.

For correct tracking, manually configure the reporters to send events about the start and the pause of the session for each reporter. Use the resumeSession() and pauseSession() methods in the IReporter interface when implementing onResume() and onPause() for your Activity:

```
public class YourActivity extends Activity {
    ...
    @Override
    protected void onResume() {
        super.onResume();
        YandexMetrica.getReporter(getApplicationContext(), API_key).resumeSession();
    }

    @Override
    protected void onPause() {
        YandexMetrica.getReporter(getApplicationContext(), API_key).pauseSession();
        super.onPause();
    }
    ...
}
```

This helps the library accurately monitor:

- The number of active users.
- Session length.
- Frequency of app use.

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Usage examples

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

Library initialization with the extended configuration

To initialize the library with the extended startup configuration, create an instance of the YandexMetricaConfig class with appropriate settings and activate the library using the YandexMetrica.activate(Context context, YandexMetricaConfig config) method. With the extended configuration allows you to enable/disable logging, set session timeout, pass parameters for tracking pre-installed apps, and so on.

The parameters of the extended configuration are applied from the time of library initialization.

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        // Creating an extended library configuration.
        YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
                // Setting up the configuration. For example, to enable logging.
                .withLogs()
                ...
                .build();
        // Initializing the AppMetrica SDK.
        YandexMetrica.activate(getApplicationContext(), config);
        // Automatic tracking of user activity.
        YandexMetrica.enableActivityAutoTracking(this);
    }
}
```

To configure the library while the application is running, use the methods of the YandexMetrica class.

Sending statistics to an additional API key

Sending data to an additional API key allows you to collect own statistics for these API keys. You can use this to control access to information for other users. For example, to provide access to statistics for analysts you can duplicate sending marketing data for the additional API key. Thus they will only have access to the information they need.

To send data to an additional API key, you must use reporters. Just like for the main API key, you can set up an extended startup configuration for a reporter, send events, profile information, and data about in-app purchases. The reporter can work without the AppMetrica SDK initialization.

Step 1. *(Optional)* Initialize a reporter with an extended configuration

To initialize a reporter with the extended configuration, create an object of the ReporterConfig class with the necessary settings and activate the reporter using the YandexMetrica.activateReporter(Context context, ReporterConfig config) method. This configuration is used for a reporter with the specified API key. You can set up your own configuration for each additional API key.

Attention: The reporter with the extended configuration should be initialized before the first call to the reporter. Otherwise, the reporter will be initialized without a configuration.

```
// Creating extended configuration of the reporter.
// To create it, pass an API_key that is different from the app's API_key.
ReporterConfig reporterConfig = ReporterConfig.newConfigBuilder(API_key)
        // Setting up the configuration. For example, to enable logging.
        .withLogs()
        ...
        .build();
// Initializing a reporter.
YandexMetrica.activateReporter(getApplicationContext(), reporterConfig);
```

Step 2. Configure sending data using a reporter

To send data using a reporter, you must get an object that implements the IReporter interface by the YandexMetrica.getReporter(Context context, String apiKey) method and use the interface methods to send reports. If the reporter was not initialized with the extended configuration, calling this method will initialize the reporter for the specified API key.

Example of sending an event:

```
YandexMetrica.getReporter(getApplicationContext(), API_key).reportEvent("Updates installed");
```

For correct tracking, manually configure the reporters to send events about the start and the pause of the session for each reporter. Use the resumeSession() and pauseSession() methods in the IReporter interface when implementing onResume() and onPause() for your Activity:

```
public class YourActivity extends Activity {
    ...
    @Override
    protected void onResume() {
        super.onResume();
        YandexMetrica.getReporter(getApplicationContext(), API_key).resumeSession();
    }

    @Override
    protected void onPause() {
        YandexMetrica.getReporter(getApplicationContext(), API_key).pauseSession();
        super.onPause();
    }
    ...
}
```

Tracking app crashes

Reports on app crashes are sent by default.

To disable automatic monitoring, initialize the library with the configuration in which sending crashes is disabled. To do this, pass the `false` value to the withLocationTracking(boolean enabled) method when creating the extended library configuration.

```
// Creating an extended library configuration.
        YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Disabling the data sending about the app crashes.
        .withCrashReporting(false)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

If automatic monitoring is disabled, you can manually submit information about crashes.

Manually monitoring app crashes

Reports on app crashes are sent by default. To avoid exception events duplication, disable the automatic monitoring of the app crashes.

To send crash information manually, use the YandexMetrica.reportUnhandledException(Throwable exception) method.

```
final Thread.UncaughtExceptionHandler previousUncaughtExceptionHandler =
 Thread.getDefaultUncaughtExceptionHandler();
// Creating a new handler.
Thread.UncaughtExceptionHandler uncaughtExceptionHandler = new Thread.UncaughtExceptionHandler() {
    @Override
    public void uncaughtException(Thread thread, Throwable exception) {
        try {
            // Sending a message about an app crash to the AppMetrica server.
            YandexMetrica.reportUnhandledException(exception);
        } finally {
            // Sending a message about an app crash to the system handler.
            if (previousUncaughtExceptionHandler != null) {
                previousUncaughtExceptionHandler.uncaughtException(thread, exception);
            }
        }
    }
};
// Setting the default handler.
Thread.setDefaultUncaughtExceptionHandler(uncaughtExceptionHandler);
```

Monitoring native app crashes

Reports are sent automatically on native app crashes if the library's SO files are added to the project. By default, SO files are omitted. To add them, enable the library for tracking native crashes.

To enable the library, add the following dependency to the `build.gradle` file in the `dependencies` block:

```
dependencies {
    ...
    implementation 'com.yandex.android:mobmetricalib-ndk-crashes:1.1.0'
}
```

If you don't use Gradle

Download the library and add it to the project.

It contains libraries for all the platforms (arm, armv7, mips, x86). Use abi splits to exclude unnecessary libraries.

To disable automatic monitoring, initialize the library with the configuration in which sending crashes is disabled. To do this, pass the `false` value to the withNativeCrashReporting(boolean enabled) method when creating the extended library configuration.

```
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Disabling the data sending about the native app crashes.
        .withNativeCrashReporting(false)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

If automatic monitoring is disabled, you can manually submit information about native crashes.

Related information

More information on the native code.

Monitoring native app crashes manually

Reports are sent automatically on native app crashes if the library's SO files are added to the project. If files have been added, you should disable the automatic monitoring of the app crashes to avoid exception events duplication.

To manually send information about native app crashes, use the YandexMetrica.reportNativeCrash(String nativeCrash) method.

```
String nativeCrashContent =
"...\n" +
"400ae000-400b7000 r-xp 00000000 103:0c 835       /system/lib/libcutils.so\n" +
"400b7000-400b8000 r--p 00008000 103:0c 835       /system/lib/libcutils.so\n" +
"400b8000-400b9000 rw-p 00009000 103:0c 835       /system/lib/libcutils.so\n" +
"400b9000-400bc000 r-xp 00000000 103:0c 1454      /system/lib/liblog.so\n" +
"...";
// Sending native app crash information.
YandexMetrica.reportNativeCrash(nativeCrashContent);
```

Sending the device location by the library

Note:

Starting with Android AppMetrica SDK 5.0.0, sending device location data is disabled by default.

To enable sending it, initialize the library with the configuration where sending device location data is enabled. To do this, pass the `true` value to the withLocationTracking(boolean enabled) method when creating an extended library configuration.

```
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Enabling the data sending about the device location.
        .withLocationTracking(true)
        .build();
```

```
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

To enable sending while the app is running, use the YandexMetrica.setLocationTracking(boolean enabled) method:

```
YandexMetrica.setLocationTracking(true);
```

To determine the location more precisely, add to the `AndroidManifest.xml` file one of the following permissions:

- android.permission.ACCESS_COARSE_LOCATION — For approximate determination.
- android.permission.ACCESS_FINE_LOCATION — For accurate determination.

Setting location manually

Before sending custom information about the device location, make sure that reporting is enabled.

The library determines the device location on its own. To send your own device location information, pass an instance of the android.location.Location class into the YandexMetrica.setLocation(Location Location) method.

```
// Determining the location.
Location currentLocation = ...;
// Setting your own location information of the device.
YandexMetrica.setLocation(currentLocation);
```

To send your own device location information using the startup configuration, pass the android.location.Location instance to the withLocation(Location Location) method when creating the extended library configuration.

```
// Determining the location.
Location currentLocation = ...;

// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Set your own location information of the device.
        .withLocation(currentLocation)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

To resume location detection by the library, pass the `null` value to the YandexMetrica.setLocation(Location location) method and configure sending the device location data.

```
YandexMetrica.setLocation(null);
```

Sending a custom event

To send your custom event without nested parameters, pass a short name or description of the event to the YandexMetrica.reportEvent(String eventName) method:

```
YandexMetrica.reportEvent("Updates installed");
```

Sending a custom event with nested parameters

The AppMetrica SDK allows you to send your own events with nested parameters that can be set in the JSON format or as a set of attributes (Map).

JSON

To pass nested event parameters in the JSON format, use the YandexMetrica.reportEvent(String eventName, String jsonValue) method:

```
String eventParameters = "{\"name\":\"Alice\", \"age\":\"18\"}";

YandexMetrica.reportEvent("New person", eventParameters);
```

Map

To send a custom event message as a Map, use the YandexMetrica.reportEvent(String eventName, Map<String, Object> attributes) method:

```
Map<String, Object> eventParameters = new HashMap<String, Object>();
eventParameters.put("name", "Alice");
eventParameters.put("age", 18);

YandexMetrica.reportEvent("New person", eventParameters);
```

The AppMetrica web interface displays up to five nesting levels for events. So if an event has six or more levels, only the top five are shown in the report. You can use the Reporting API to get up to ten levels.

For more information about events, see Events.

Sending an event from the WebView's JavaScript code

The AppMetrica SDK lets you send client events from JavaScript code. Initialize the sending by calling the initWebViewReporting method:

Java

```
WebView webView = (WebView) findViewById(R.id.myWebView);
// do your initialization here
webView.getSettings().setJavaScriptEnabled(true);
YandexMetrica.initWebViewReporting(webView);
webView.loadUrl(myURL);
```

Kotlin

```
val webView = findViewById<WebView>(R.id.myWebView)
// do your initialization here
webView.settings.javaScriptEnabled = true
YandexMetrica.initWebViewReporting(webView)
webView.loadUrl(myURL)
```

Call the `initWebViewReporting` method after calling the `setJavaScriptEnabled` method but before calling `loadUrl`.

To send an event from JavaScript code, use the `reportEvent(name, value)` method in the AppMetrica interface:

```
function buttonClicked() {
  AppMetrica.reportEvent('Button clicked!', '{}');
})
```

Arguments of the `reportEvent` method:

- `name` — A string. Can't be null or empty.
- `value` — A JSON string. Can be null.

Sending an error message

To send your own error message, use the following methods:

- YandexMetrica.reportError(String message, Throwable error)
- YandexMetrica.reportError(String groupIdentifier, String message)
- YandexMetrica.reportError(String groupIdentifier, String message, Throwable error)

Methods with `groupIdentifier` are supported since AppMetrica SDK version 3.14.2.

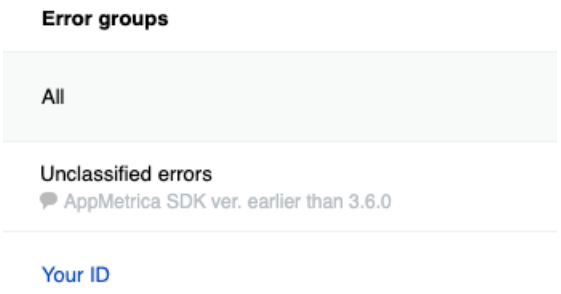
Example with groupIdentifier

If you send errors using methods with `groupIdentifier`, they are grouped by ID.

```
try {
    Integer.valueOf("00xffWr0ng");
} catch (Throwable error) {
    YandexMetrica.reportError("Your ID", "Error while parsing some integer number", error);
```

```
}
```

Don't use variable values as grouping IDs. Otherwise, the number of groups increases and it becomes difficult to analyze them.

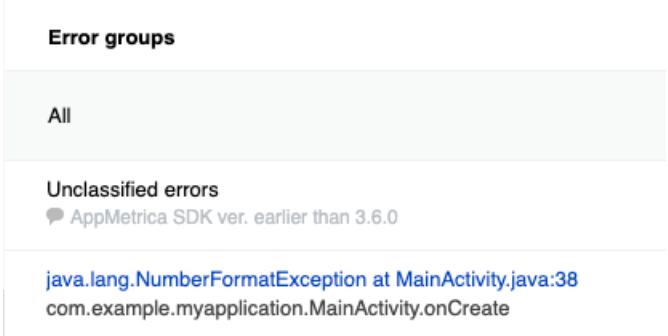


1. groupIdentifier

Example with message

If errors are sent using the YandexMetrica.reportError(String message, Throwable error) method, they're grouped by stack trace.

```
try {
    Integer.valueOf("00xffWr0ng");
} catch (Throwable error) {
    YandexMetrica.reportError("Error while parsing some integer number", error);
}
```



1. Stack trace

Sending ProfileId

If you know the user profile ID before initializing the AppMetrica SDK, pass it before initialization (setUserProfileID) or during initialization with the extended configuration (withUserProfileID).

Otherwise, a user profile is created with the ID `appmetrica_device_id`.

Attention: AppMetrica doesn't display predefined attributes in the web interface if ProfileId sending isn't configured.

You can update `ProfileId` at any time by calling setUserProfileID:

```
YandexMetrica.setUserProfileID("id");
```

Sending profile attributes

To send profile attributes, pass to the instance of the UserProfile class required attributes and send this instance using the YandexMetrica.reportUserProfile(UserProfile profile) method. To create profile attributes, use methods of the Attribute class.

```
// Creating the UserProfile instance.
UserProfile userProfile = UserProfile.newBuilder()
        // Updating predefined attributes.
        .apply(Attribute.name().withValue("John"))
        .apply(Attribute.gender().withValue(GenderAttribute.Gender.MALE))
        .apply(Attribute.birthDate().withAge(24))
        .apply(Attribute.notificationsEnabled().withValue(false))
        // Updating custom attributes.
        .apply(Attribute.customString("string_attribute").withValue("string"))
        .apply(Attribute.customNumber("number_attribute").withValue(55))
        .apply(Attribute.customCounter("counter_attribute").withDelta(1))
        .build();
// Setting the ProfileID using the method of the YandexMetrica class.
YandexMetrica.setUserProfileID("id");

// Sending the UserProfile instance.
YandexMetrica.reportUserProfile(userProfile);
```

Sending E-commerce events

In AppMetrica, it is not possible to segment E-commerce events into “test” and “not test”. If you use the main API key for debugging purchases, the test events are included in general statistics. If you need to debug sending E-commerce events, use a reporter to send statistics to an additional API key.

Step 1. Configure sending E-commerce events to the test API key

Java

For different user actions, there are appropriate types of E-commerce events. To create a specific event type, use the appropriate `ECommerceEvent` class method.

The examples below show how to send specific types of events (Java):

Opening a page

```
Map<String, String> payload = new HashMap<>();
payload.put("configuration", "landscape");
payload.put("full_screen", "true");
// Creating a screen object.
  ECommerceScreen screen = new ECommerceScreen()
 .setCategoriesPath(Arrays.asList( // Optional.
  "Акции",
  "Красная цена"
 ))
 .setName("ProductCardActivity") // Optional.
 .setSearchQuery("даниссимо кленовый сироп") // Optional.
 .setPayload(payload); // Optional.
  ECommerceEvent showScreenEvent = ECommerceEvent.showScreenEvent(screen);
// Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportECommerce(showScreenEvent);
```

Viewing a product profile

```
Map<String, String> payload = new HashMap<>();
payload.put("configuration", "landscape");
payload.put("full_screen", "true");
// Creating a screen object.
  ECommerceScreen screen = new ECommerceScreen()
 .setCategoriesPath(Arrays.asList( // Optional.
  "Акции",
  "Красная цена"
 ))
 .setName("ProductCardActivity") // Optional.
 .setSearchQuery("даниссимо кленовый сироп") // Optional.
 .setPayload(payload); // Optional.
// Creating an actualPrice object.
  ECommercePrice actualPrice = new ECommercePrice(new ECommerceAmount(4.53, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_570_000, "wood"),
  new ECommerceAmount(26.89, "iron"),
  new ECommerceAmount(new BigDecimal(5.1), "gold")
 ));
// Creating an originalPrice object.
  ECommercePrice originalPrice = new ECommercePrice(new ECommerceAmount(5.78, "USD"))
```

```
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_590_000, "wood"),
  new ECommerceAmount(26.92, "iron"),
  new ECommerceAmount(new BigDecimal(5.5), "gold")
 ));
// Creating a product object.
  ECommerceProduct product = new ECommerceProduct("779213")
 .setActualPrice(actualPrice) // Optional.
 .setPromocodes(Arrays.asList("BT79IYX", "UT5412EP")) // Optional.
 .setPayload(payload) // Optional.
 .setOriginalPrice(originalPrice) // Optional.
 .setName("Продукт творожный «Даниссимо» 5.9%, 130 г.") // Optional.
 .setCategoriesPath(Arrays.asList("Продукты", "Молочные продукты", "Йогурты")); // Optional.
                     ECommerceEvent showProductCardEvent = ECommerceEvent.showProductCardEvent(product,
 screen);
// Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API
 key").reportECommerce(showProductCardEvent);
```

Viewing a product page

```
Map<String, String> payload = new HashMap<>();
payload.put("configuration", "landscape");
payload.put("full_screen", "true");
// Creating a screen object.
  ECommerceScreen screen = new ECommerceScreen()
 .setCategoriesPath(Arrays.asList( // Optional.
  "Акции",
  "Красная цена"
 ))
 .setName("ProductCardActivity") // Optional.
 .setSearchQuery("даниссимо кленовый сироп") // Optional.
 .setPayload(payload); // Optional.

// Creating an actualPrice object.
  ECommercePrice actualPrice = new ECommercePrice(new ECommerceAmount(4.53, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_570_000, "wood"),
  new ECommerceAmount(26.89, "iron"),
  new ECommerceAmount(new BigDecimal(5.1), "gold")
 ));
// Creating an originalPrice object.
  ECommercePrice originalPrice = new ECommercePrice(new ECommerceAmount(5.78, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_590_000, "wood"),
  new ECommerceAmount(26.92, "iron"),
  new ECommerceAmount(new BigDecimal(5.5), "gold")
 ));
// Creating a product object.
  ECommerceProduct product = new ECommerceProduct("779213")
 .setActualPrice(actualPrice) // Optional.
 .setPromocodes(Arrays.asList("BT79IYX", "UT5412EP")) // Optional.
 .setPayload(payload) // Optional.
 .setOriginalPrice(originalPrice) // Optional.
 .setName("Продукт творожный «Даниссимо» 5.9%, 130 г.") // Optional.
 .setCategoriesPath(Arrays.asList("Продукты", "Молочные продукты", "Йогурты")); // Optional.
// Creating a referrer object.
  ECommerceReferrer referrer = new ECommerceReferrer()
 .setType("button") // Optional.
 .setIdentifier("76890") // Optional.
 .setScreen(screen); // Optional.
                     ECommerceEvent showProductDetailsEvent =
 ECommerceEvent.showProductDetailsEvent(product, referrer); // Referrer is optional — can be null.
// Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API
 key").reportECommerce(showProductDetailsEvent);
```

Adding or removing an item to/from the cart

```
Map<String, String> payload = new HashMap<>();
payload.put("configuration", "landscape");
payload.put("full_screen", "true");
// Creating a screen object.
  ECommerceScreen screen = new ECommerceScreen()
 .setCategoriesPath(Arrays.asList( // Optional.
  "Акции",
  "Красная цена"
 ))
 .setName("ProductCardActivity") // Optional.
 .setSearchQuery("даниссимо кленовый сироп") // Optional.
 .setPayload(payload); // Optional.
// Creating an actualPrice object.
  ECommercePrice actualPrice = new ECommercePrice(new ECommerceAmount(4.53, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_570_000, "wood"),
  new ECommerceAmount(26.89, "iron"),
  new ECommerceAmount(new BigDecimal(5.1), "gold")
 ));
// Creating an originalPrice object.
  ECommercePrice originalPrice = new ECommercePrice(new ECommerceAmount(5.78, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_590_000, "wood"),
  new ECommerceAmount(26.92, "iron"),
```

```
  new ECommerceAmount(new BigDecimal(5.5), "gold")
 ));
// Creating a product object.
  ECommerceProduct product = new ECommerceProduct("779213")
 .setActualPrice(actualPrice) // Optional.
 .setPromocodes(Arrays.asList("BT79IYX", "UT5412EP")) // Optional.
 .setPayload(payload) // Optional.
 .setOriginalPrice(originalPrice) // Optional.
 .setName("Продукт творожный «Даниссимо» 5.9%, 130 г.") // Optional.
 .setCategoriesPath(Arrays.asList("Продукты", "Молочные продукты", "Йогурты")); // Optional.
// Creating a referrer object.
  ECommerceReferrer referrer = new ECommerceReferrer()
 .setType("button") // Optional.
 .setIdentifier("76890") // Optional.
 .setScreen(screen); // Optional.
// Creating a cartItem object.
  ECommerceCartItem addedItems1 = new ECommerceCartItem(product, actualPrice, 1.0)
 .setReferrer(referrer); // Optional.
                    ECommerceEvent addCartItemEvent = ECommerceEvent.addCartItemEvent(addedItems1);
                    // Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportECommerce(addCartItemEvent);
                    ECommerceEvent removeCartItemEvent = ECommerceEvent.removeCartItemEvent(addedItems1);
// Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportECommerce(removeCartItemEvent);
```

Starting and completing a purchase

```
Map<String, String> payload = new HashMap<>();
payload.put("configuration", "landscape");
payload.put("full_screen", "true");
// Creating a screen object.
  ECommerceScreen screen = new ECommerceScreen()
 .setCategoriesPath(Arrays.asList( // Optional.
  "Акции",
  "Красная цена"
 ))
 .setName("ProductCardActivity") // Optional.
 .setSearchQuery("даниссимо кленовый сироп") // Optional.
 .setPayload(payload); // Optional.
// Creating an actualPrice object.
  ECommercePrice actualPrice = new ECommercePrice(new ECommerceAmount(4.53, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_570_000, "wood"),
  new ECommerceAmount(26.89, "iron"),
  new ECommerceAmount(new BigDecimal(5.1), "gold")
 ));
// Creating an originalPrice object.
  ECommercePrice originalPrice = new ECommercePrice(new ECommerceAmount(5.78, "USD"))
 .setInternalComponents(Arrays.asList( // Optional.
 new ECommerceAmount(30_590_000, "wood"),
  new ECommerceAmount(26.92, "iron"),
  new ECommerceAmount(new BigDecimal(5.5), "gold")
 ));
// Creating a product object.
  ECommerceProduct product = new ECommerceProduct("779213")
 .setActualPrice(actualPrice) // Optional.
 .setPromocodes(Arrays.asList("BT79IYX", "UT5412EP")) // Optional.
 .setPayload(payload) // Optional.
 .setOriginalPrice(originalPrice) // Optional.
 .setName("Продукт творожный «Даниссимо» 5.9%, 130 г.") // Optional.
 .setCategoriesPath(Arrays.asList("Продукты", "Молочные продукты", "Йогурты")); // Optional.
// Creating a referrer object.
  ECommerceReferrer referrer = new ECommerceReferrer()
 .setType("button") // Optional.
 .setIdentifier("76890") // Optional.
 .setScreen(screen); // Optional.
// Creating a cartItem object.
  ECommerceCartItem addedItems1 = new ECommerceCartItem(product, actualPrice, 1.0)
 .setReferrer(referrer); // Optional.
// Creating an order object.
ECommerceOrder order = new ECommerceOrder("88528768", Arrays.asList(addedItems1))
        .setPayload(payload); // Optional.
                    ECommerceEvent beginCheckoutEvent = ECommerceEvent.beginCheckoutEvent(order);
// Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportECommerce(beginCheckoutEvent);
                    ECommerceEvent purchaseEvent = ECommerceEvent.purchaseEvent(order);
// Sending an e-commerce event.
  YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportECommerce(purchaseEvent);
```

Kotlin

For different user actions, there are appropriate types of E-commerce events. To create a specific event type, use the appropriate `ECommerceEvent` class method.

The examples below show how to send specific types of events (Kotlin):

Opening a page

```
val payload = mapOf("configuration" to "landscape", "full_screen" to "true")
// Creating a screen object.
val screen = ECommerceScreen().apply {
 categoriesPath = listOf("Акции", "Красная цена") // Optional.
```

```
 name = "ProductCardActivity" // Optional.
 searchQuery = "даниссимо кленовый сироп" // Optional.
 this.payload = payload // Optional.
 }
val showScreenEvent = ECommerceEvent.showScreenEvent(screen)
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(showScreenEvent)
```

Viewing a product profile

```
val payload = mapOf("configuration" to "landscape", "full_screen" to "true")
// Creating a screen object.
val screen = ECommerceScreen().apply {
 categoriesPath = listOf("Акции", "Красная цена") // Optional.
 name = "ProductCardActivity" // Optional.
 searchQuery = "даниссимо кленовый сироп" // Optional.
 this.payload = payload // Optional.
 }
// Creating an actualPrice object.
val actualPrice = ECommercePrice(ECommerceAmount(4.53, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30570000, "wood"),
   ECommerceAmount(26.89, "iron"),
   ECommerceAmount(BigDecimal(5.1), "gold")
  )
 }
// Creating an originalPrice object.
val originalPrice = ECommercePrice(ECommerceAmount(5.78, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30590000, "wood"),
   ECommerceAmount(26.92, "iron"),
   ECommerceAmount(BigDecimal(5.5), "gold")
  )
 }
// Creating a product object.
val product = ECommerceProduct("779213")
 .apply {
  this.actualPrice = actualPrice // Optional.
  this.originalPrice = originalPrice // Optional.
  promocodes = listOf("BT79IYX", "UT5412EP") // Optional.
  name = "Продукт творожный «Даниссимо» 5.9%, 130 г." // Optional.
  categoriesPath = listOf("Продукты", "Молочные продукты", "Йогурты") // Optional.
  this.payload = payload // Optional.
 }

val showProductCardEvent = ECommerceEvent.showProductCardEvent(product, screen)
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(showProductCardEvent)
```

Viewing a product page

```
val payload = mapOf("configuration" to "landscape", "full_screen" to "true")
// Creating a screen object.
val screen = ECommerceScreen().apply {
 categoriesPath = listOf("Акции", "Красная цена") // Optional.
 name = "ProductCardActivity" // Optional.
 searchQuery = "даниссимо кленовый сироп" // Optional.
 this.payload = payload // Optional.
 }
// Creating an actualPrice object.
val actualPrice = ECommercePrice(ECommerceAmount(4.53, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30570000, "wood"),
   ECommerceAmount(26.89, "iron"),
   ECommerceAmount(BigDecimal(5.1), "gold")
  )
 }
// Creating an originalPrice object.
val originalPrice = ECommercePrice(ECommerceAmount(5.78, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30590000, "wood"),
   ECommerceAmount(26.92, "iron"),
   ECommerceAmount(BigDecimal(5.5), "gold")
  )
 }
// Creating a product object.
val product = ECommerceProduct("779213")
 .apply {
  this.actualPrice = actualPrice // Optional.
  this.originalPrice = originalPrice // Optional.
  promocodes = listOf("BT79IYX", "UT5412EP") // Optional.
  name = "Продукт творожный «Даниссимо» 5.9%, 130 г." // Optional.
  categoriesPath = listOf("Продукты", "Молочные продукты", "Йогурты") // Optional.
  this.payload = payload // Optional.
 }

// Creating a referrer object.
val referrer = ECommerceReferrer()
 .apply {
```

```
  type = "button" // Optional.
  identifier = "76890" // Optional.
  this.screen = screen // Optional.
 }
val showProductDetailsEvent = ECommerceEvent.showProductDetailsEvent(product, referrer) // Referrer is
 optional - can be null.
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(showProductDetailsEvent)
```

Adding or removing an item to/from the cart

```
val payload = mapOf("configuration" to "landscape", "full_screen" to "true")
// Creating a screen object.
val screen = ECommerceScreen().apply {
 categoriesPath = listOf("Акции", "Красная цена") // Optional.
 name = "ProductCardActivity" // Optional.
 searchQuery = "даниссимо кленовый сироп" // Optional.
 this.payload = payload // Optional.
 }
// Creating an actualPrice object.
val actualPrice = ECommercePrice(ECommerceAmount(4.53, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30570000, "wood"),
   ECommerceAmount(26.89, "iron"),
   ECommerceAmount(BigDecimal(5.1), "gold")
  )
 }
// Creating an originalPrice object.
val originalPrice = ECommercePrice(ECommerceAmount(5.78, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30590000, "wood"),
   ECommerceAmount(26.92, "iron"),
   ECommerceAmount(BigDecimal(5.5), "gold")
  )
 }
// Creating a product object.
val product = ECommerceProduct("779213")
 .apply {
  this.actualPrice = actualPrice // Optional.
  this.originalPrice = originalPrice // Optional.
  promocodes = listOf("BT79IYX", "UT5412EP") // Optional.
  name = "Продукт творожный «Даниссимо» 5.9%, 130 г." // Optional.
  categoriesPath = listOf("Продукты", "Молочные продукты", "Йогурты") // Optional.
  this.payload = payload // Optional.
 }

// Creating a referrer object.
val referrer = ECommerceReferrer()
 .apply {
  type = "button" // Optional.
  identifier = "76890" // Optional.
  this.screen = screen // Optional.
 }
// Creating a cartItem object.
val addedItems1 = ECommerceCartItem(product, actualPrice, 1.0)
 .apply {
  this.referrer = referrer // Optional.
 }
val addCartItemEvent = ECommerceEvent.addCartItemEvent(addedItems1)
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(addCartItemEvent)

val removeCartItemEvent = ECommerceEvent.removeCartItemEvent(addedItems1)
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(removeCartItemEvent)
```

Starting and completing a purchase

```
val payload = mapOf("configuration" to "landscape", "full_screen" to "true")
// Creating a screen object.
val screen = ECommerceScreen().apply {
 categoriesPath = listOf("Акции", "Красная цена") // Optional.
 name = "ProductCardActivity" // Optional.
 searchQuery = "даниссимо кленовый сироп" // Optional.
 this.payload = payload // Optional.
 }
// Creating an actualPrice object.
val actualPrice = ECommercePrice(ECommerceAmount(4.53, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30570000, "wood"),
   ECommerceAmount(26.89, "iron"),
   ECommerceAmount(BigDecimal(5.1), "gold")
  )
 }
// Creating an originalPrice object.
val originalPrice = ECommercePrice(ECommerceAmount(5.78, "USD"))
 .apply {
  internalComponents = listOf( // Optional.
   ECommerceAmount(30590000, "wood"),
   ECommerceAmount(26.92, "iron"),
```

```
    ECommerceAmount(BigDecimal(5.5), "gold")
   )
  }
// Creating a product object.
val product = ECommerceProduct("779213")
 .apply {
  this.actualPrice = actualPrice // Optional.
  this.originalPrice = originalPrice // Optional.
  promocodes = listOf("BT79IYX", "UT5412EP") // Optional.
  name = "Продукт творожный «Даниссимо» 5.9%, 130 г." // Optional.
  categoriesPath = listOf("Продукты", "Молочные продукты", "Йогурты") // Optional.
  this.payload = payload // Optional.
 }

// Creating a referrer object.
val referrer = ECommerceReferrer()
 .apply {
  type = "button" // Optional.
  identifier = "76890" // Optional.
  this.screen = screen // Optional.
 }
// Creating a cartItem object.
val addedItems1 = ECommerceCartItem(product, actualPrice, 1.0)
 .apply {
  this.referrer = referrer // Optional.
 }
// Creating an order object.
val order = ECommerceOrder("88528768", listOf(addedItems1))
        .apply {
            this.payload = payload // Optional.
        }
val beginCheckoutEvent = ECommerceEvent.beginCheckoutEvent(order)
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(beginCheckoutEvent)

val purchaseEvent = ECommerceEvent.purchaseEvent(order)
// Sending an e-commerce event.
YandexMetrica.getReporter(applicationContext, "Testing API key").reportECommerce(purchaseEvent)
```

Step 2. Check the test application's report

Make in-app test purchases. After a while, check the “Purchase analysis” report in the AppMetrica interface. Make sure that the report shows E-commerce events.

For more information about the report, see Purchase analysis.

Step 3. Configure sending events to the main API key

After successful testing, configure sending E-commerce events to the main API key.

To send the ECommerceEvent instance to the main API key, use the YandexMetrica.reportECommerce(@NonNull ECommerceEvent event) method.

Java

```
...
// Sending an e-commerce event.
YandexMetrica.reportECommerce(ecommerceEvent);
```

Kotlin

```
...
// Sending an e-commerce event.
YandexMetrica.reportECommerce(ecommerceEvent)
```

Sending Revenue

AppMetrica supports the validation of purchases made using the Google Play Billing library. Validation lets you filter purchases made from hacked apps. If validation is enabled and a purchase fails it, this purchase isn't shown in reports.

Note: Local validation with a public key is used to validate purchases on Android. To enable validation, create a public key and specify it in the settings. For more information, see Sending In-App purchases on Android.

Step 1. Configure sending Revenue to the test API key

AppMetrica doesn't let you segment Revenue into “test” and “not test”. If you use the main API key for debugging purchases, the test purchases are included in general statistics. If you need to debug sending Revenue, use a reporter to send statistics to an additional API key.

This section outlines the steps for sending Revenue to the additional API key:

With validation

To validate purchases on Android, configure sending the Revenue.Receipt instance along with the Revenue:

1. Create the `Revenue.Receipt` instance with information about the purchase and signature. You must use it when creating the `Revenue` instance in Revenue.Builder.withReceipt (Revenue.Receipt receipt).
2. Create the `Revenue` instance using the Revenue.Builder constructor.
3. Send the `Revenue` instance to the test API key using the IReporter. For more information about reporters, see Usage examples.

```
void handlePurchase(Purchase purchase) {
    ...
    // Creating the Revenue.Receipt instance.
    // It is used for checking purchases in Google Play.
    Revenue.Receipt revenueReceipt = Revenue.Receipt.newBuilder()
            .withData(purchase.getOriginalJson())
            .withSignature(purchase.getSignature())
            .build();
    // Creating the Revenue instance.
    Revenue revenue = Revenue.newBuilderWithMicros(99000000, Currency.getInstance("RUB"))
            .withProductID("com.yandex.service.299")
            .withQuantity(2)
            .withReceipt(revenueReceipt)
            .withPayload("{\"source\":\"Google Play\"}")
            .build();
    // Sending the Revenue instance using reporter.
    YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportRevenue(revenue);
}
```

Without validation

To send information about a purchase without validation:

1. Create the Revenue instance using the Revenue.Builder constructor.
2. *(Optional)* To group purchases by OrderID, specify it in the Revenue.Builder.withPayload(String payload) method.

   Note: If the OrderID is not specified, AppMetrica generates the ID automatically.
3. Send the `Revenue` instance to the test API key using the IReporter. For more information about reporters, see Usage examples.

```
// Creating the Revenue instance.
Revenue revenue = Revenue.newBuilderWithMicros(99000000, Currency.getInstance("RUB"))
        .withProductID("com.yandex.service.299")
        .withQuantity(2)
        // Passing the OrderID parameter in the .withPayload(String payload) method to group purchases.
        .withPayload("{\"OrderID\":\"Identifier\", \"source\":\"Google Play\"}")
        .build();
// Sending the Revenue instance using reporter.
YandexMetrica.getReporter(getApplicationContext(), "Testing API key").reportRevenue(revenue);
```

Step 2. Check the test application's report

In the AppMetrica interface, check the “Revenue” report. Make sure that the number of purchases has increased.

For more information about the report, see Revenue.

Step 3. Configure sending Revenue to the main API key

After successful testing, configure sending Revenue to the main API key.

To send the `Revenue` instance to the main API key, use the YandexMetrica.reportRevenue(Revenue revenue) method.

```
...
// Sending the Revenue instance.
YandexMetrica.reportRevenue(revenue);
```

Setting the length of the session timeout

By default, the session timeout is 10 seconds. The minimum acceptable value for the `sessionTimeout` parameter is 10 seconds.

To change the length of the timeout, pass the value in seconds to the withSessionTimeout(int sessionTimeout) method when creating the extended library configuration.

```
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Setting the length of the session timeout.
        .withSessionTimeout(15)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

Setting the app version

By default, the app version is set in the file build.gradle.

To specify the app version from the code, pass the app version to the withAppVersion(String appVersion) method when creating the extended library configuration.

```
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Setting the app version.
        .withAppVersion("1.13.2")
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

where 1.13.2 is the app version.

Determining the library's API level

To determine the API level of the library from the application code, use the YandexMetrica.getLibraryApiLevel() method.

```
int libraryApiLevel = YandexMetrica.getLibraryApiLevel();
```

Determining the library version

To determine the library version from the application code, use the YandexMetrica.getLibraryVersion() method:

```
String libraryVersion = YandexMetrica.getLibraryVersion();
```

Tracking installation sources

Tracking installation source in AppMetrica SDK is enabled by default.

To disable tracking, make the following changes to the `AndroidManifest.xml` file:

```
<manifest
    ...
    xmlns:tools="http://schemas.android.com/tools">
    ...
    <application ...>
        ...
        <receiver android:name="com.yandex.metrica.MetricaEventHandler" tools:node="remove"/>
        ...
    </application>
</manifest>
```

Tracking app openings using deeplinks

To track app openings that use deeplinks, you make changes in the Activity that handles deeplinks. You need to track app openings to correctly track remarketing campaigns.

Note: To work with deeplinks, configure them in your application.

Use the reportAppOpen() method to track app openings:

```
public class DeeplinkActivity extends Activity {

    @Override
    protected void onCreate(@Nullable final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
            YandexMetrica.reportAppOpen(this);
        }
    }
    @Override
    protected void onNewIntent(final Intent intent) {
        super.onNewIntent(intent);
        YandexMetrica.reportAppOpen(intent);
    }
}
```

Note: You can add the opening of the app via the deferred deeplink on Android.

See also

How to create a remarketing campaign in AppMetrica

Requesting a deferred deeplink

To request a deferred deeplink, pass a class instance that implements the DeferredDeeplinkListener interface to the YandexMetrica.requestDeferredDeeplink(DeferredDeeplinkListener listener) method. The method returns the deferred deeplink only at the initial application start after obtaining the Google Play Install Referrer.

Note: If Google Play Install Referrer is obtained immediately after installation, the system will not be able to properly recognize the first application launch.

Java

```
YandexMetrica.requestDeferredDeeplink(new DeferredDeeplinkListener() {
    @Override
    public void onDeeplinkLoaded(String deeplink) {
        Log.i("Deeplink", "deeplink = " + deeplink);
    }

    @Override
    public void onError(Error error, String referrer) {
        Log.i("Deeplink", "Error: " + error.getDescription() + ", unparsed referrer: " + referrer);
    }
});
```

Kotlin

```
YandexMetrica.requestDeferredDeeplink(object : DeferredDeeplinkListener {
    override fun onDeeplinkLoaded(deeplink: String) {
        Log.i("Deeplink", "deeplink = $deeplink")
    }

    override fun onError(error: DeferredDeeplinkListener.Error, referrer: String?) {
        Log.i("Deeplink", "Error: ${error.description}, unparsed referrer: $referrer")
    }
})
```

For more information about deferred deeplinks, see Support for deferred deeplinks

Requesting deferred deeplink parameters

To request parameters for a deferred deeplink, pass to the YandexMetrica.requestDeferredDeeplinkParameters(DeferredDeeplinkParametersListener listener) method an

instance that implements the DeferredDeeplinkParametersListener interface. The method returns the deferred deeplink parameters only at the first application start after Google Play Install Referrer obtaining.

Note: If Google Play Install Referrer is obtained immediately after installation, the system will not be able to properly recognize the first application launch.

```
YandexMetrica.requestDeferredDeeplinkParameters(new DeferredDeeplinkParametersListener() {
    @Override
    public void onParametersLoaded(Map<String, String> parameters) {
        // Processing deeplink parameters.
        for (String key : parameters.keySet()) {
            Log.i("Deeplink params", "key: " + key + " value: " + parameters.get(key));
        }
    }

    @Override
    public void onError(Error error, String referrer) {
        // Handling the error.
        Log.e("Error!", error.getDescription());
    }
});
```

For more information about deferred deeplinks, see Support for deferred deeplinks

Tracking new users

By default, when the app is first launched, all users are identified as new. If the AppMetrica SDK connects to an app that already has active users, you can configure it for new and old users to track statistics correctly. To set this up, initialize the AppMetrica SDK with the extended configuration `YandexMetricaConfig`:

```
boolean isFirstLaunch;
// Implement logic to detect whether the app is opening for the first time.
// For example, you can check for files (settings, databases, and so on),
// which the app creates on its first launch.
if (conditions) {
    isFirstLaunch = true;
}
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        .handleFirstActivationAsUpdate(!isFirstLaunch)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

Disable and enable sending statistics

If you need confirmation from the user before sending statistical data, you should initialize the library with the disabled sending option. Pass the `false` value to the withStatisticsSending() method when creating the extended library configuration.

```
// Creating an extended library configuration.
YandexMetricaConfig config = YandexMetricaConfig.newConfigBuilder(API_key)
        // Disabling sending statistics.
        .withStatisticsSending(false)
        .build();
// Initializing the AppMetrica SDK.
YandexMetrica.activate(getApplicationContext(), config);
```

After the user has confirmed to send statistics (for example, in the application settings or in the agreement on the first start of the app), you should call the YandexMetrica.setStatisticsSending(Context context, boolean enabled) method to enable it:

```
// Checking the status of the boolean variable. It shows the user confirmation.
if (flag) {
    // Enabling sending statistics.
    YandexMetrica.setStatisticsSending(getApplicationContext(), true);
}
```

Alert example

You can use any text to inform users of statistics collection. For example:

This app uses the AppMetrica analytical service (Yandex Metrica for apps) provided by YANDEX LLC, ulitsa Lva Tolstogo 16, Moscow, Russia 119021 (hereinafter referred to as Yandex) based on the Terms of Use.

AppMetrica analyzes app usage data, including the device it is running on, the installation source, calculates conversion, collects statistics of your activity for product analytics, ad campaign analysis, and optimization, as well as for troubleshooting. Information collected in this way cannot identify you.

Depersonalized information about your use of this app collected by AppMetrica tools will be transferred to Yandex and stored on Yandex's server in the EU and the Russian Federation. Yandex will process this information to assess how you use the app, compile reports for us on our app operation, and provide other services.

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Analyzing crashes

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

AppMetrica doesn't conflict with other libraries that collect and process app crashes. If you use such libraries, initialize AppMetrica after installing these libraries.

Note: The library does not symbolicate or deobfuscate app crashes. These operations are performed on the server or on the client side.

To upload mapping files to AppMetrica, use the crash plugin. It automatically uploads the mapping and SO files when building an app. For more information, see uploading mapping files and debugging symbols on Android.

For automatically collecting information about app crashes, AppMetrica uses the standard handler Thread.UncaughtExceptionHandler. If you are using an app crash handler directly inside the app, use the following example of implementing correct data handling:

```
Thread.UncaughtExceptionHandler mAndroidCrashHandler = Thread.getDefaultUncaughtExceptionHandler();

private final Thread.UncaughtExceptionHandler mUncaughtExceptionHandler = new Thread.UncaughtExceptionHandler() {

    @Override
    public void uncaughtException(Thread thread, Throwable exception) {
        try {
            // Put your logic here.
        } finally {
            // Give to the system.
            if (null != mAndroidCrashHandler) {
                mAndroidCrashHandler.uncaughtException(thread, exception);
            }
        }
    }

};

Thread.setDefaultUncaughtExceptionHandler(mUncaughtExceptionHandler);
```

This way you handle the exception yourself, then pass it on, and AppMetrica can send information about it.

If you want to transmit additional information about an app crash, use this example up to initializing the library in the app.

See also

Crash plugin on page 30

Uploading mapping files and debugging symbols on Android

Related information

com.yandex.metrica package

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Crash plugin

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

AppMetrica lets you collect information about native and Java crashes. You can analyze them in the Crashes report. See also Crashes/errors.

To reduce the size of an app, optimize the code during a release build.

If the code was compressed and obfuscated during the build of an Android application, information about crashes is transmitted in obfuscated form. To extract data for analysis from such crash logs, AppMetrica performs server-side deobfuscation.

To do this, upload the mapping files or debug symbols of SO files to AppMetrica, either automatically when building the app or manually via the web interface.

To send files, enable the AppMetrica Build Plugin.

Note: Upload mapping files if you use ProGuard or R8. If you don't compress or obfuscate the code, don't enable the plugin.

Connecting the plugin

Attention: To use the plugin, you need to install the AppMetrica SDK version not lower than 3.8.0.

To enable the plugin:

1. Add the following dependency to the `build.gradle` root file:

```
buildscript {
    ...
    dependencies {
        ...
        classpath 'com.yandex.android:appmetrica-build-plugin:0.7.0'
    }
 }
```

2. Add to the `app/build.gradle` file:

```
...
apply plugin: 'appmetrica-plugin'

appmetrica {
    postApiKey = { applicationVariant -> "Post Api key for variant" }
    // or postApiKey = "Post Api key"
    enable = { applicationVariant -> true }    // Optional.
    mappingBuildTypes = ['release']            // Optional.
    offline = false                            // Optional.
    mappingFile = { applicationVariant -> null }   // Optional.
    enableAnalytics = true                     // Optional.
    ndk {    // Optional.
        enable = { applicationVariant -> false }
        soFiles = { applicationVariant -> listOfSoFiles }             // Optional.
        additionalSoFiles = { applicationVariant -> listOfSoFiles }   // Optional.
        addNdkCrashesDependency = { applicationVariant -> true }      // Optional.
    }
}
```

```
...
```

postApiKey

Post API key or lambda function that returns a Post API key for `ApplicationVariant`. Read more about `ApplicationVariant` in the Android and Javadoc documentation.

You can get the Post API key in the AppMetrica **Settings**. It's used to identify your app.

If `offline = true`, the parameter is optional.

enable

Lambda function that accepts `ApplicationVariant` and returns whether to use the plugin for this build option.

If the plugin isn't used, the `apk` doesn't change and the mapping file doesn't load.

By default, it returns `true` only for `buildType = 'release'`.

Note: Use one of the parameters to specify assembly types: `enable` or `mappingBuildTypes`.

mappingBuildTypes

The list of build types, `buildType`, for which the mapping file will be sent.

Default value: `['release']`.

To disable the uploading of mapping files for a specific type of build, delete that `buildType` from the list.

Note: Use one of the parameters to specify assembly types: `enable` or `mappingBuildTypes`.

offline

Enables the `offline` mode. `Boolean` or a lambda function that accepts `ApplicationVariant`.

Acceptable values:

- `true` — Doesn't upload a file to AppMetrica. If enabled, it outputs the archive path to the log after the build is complete. You can upload it manually via the web interface.

- `false` — Automatically uploads the mapping file.

The default value is `false`.

mappingFile

A lambda function that accepts `ApplicationVariant` and returns the mapping file to upload. If it returns `null`, the default value is used.

The default value is `ApplicationVariant.mappingFileProvider`.

enableAnalytics

Enables sending plugin usage statistics to AppMetrica. `Boolean`.

Acceptable values:

- `true` — Sending statistics is enabled.
- `false` — Sending statistics is disabled.

The default value is `true`.

ndk

This parameter is required for loading characters from the SO file to track native crashes.

enable

Lambda function that accepts `ApplicationVariant` and returns whether to load the characters from the SO files for this build option.

The default value is `false`.

soFiles

Lambda function that accepts `ApplicationVariant` and returns a list of SO files.

By default, the plugin searches for SO files in the Android Studio project folders.

You need to redefine the path if your SO files are in an unusual location or the plugin can't find them.

additionalSoFiles

Lambda function that accepts `ApplicationVariant` and returns a list of additional SO files to load the characters from.

addNdkCrashesDependency

Lambda function that accepts `ApplicationVariant` and returns whether to add a dependency on `mobmetricalib-ndk-crashes`.

The default value is `true`.

3. If the `ndk` parameter is enabled, debug symbols are sent using the Gradle task `upload ${variant.name.capitalize()}Symbols.` To call a task automatically, you can add it to a file `app/build.gradle` the following code:

```
...
afterEvaluate {
    android.applicationVariants.forEach { variant ->
        def uploadSymbolsTask = project.tasks.named("upload${variant.name.capitalize()}Symbols")
        if (uploadSymbolsTask != null) {
            variant.assembleProvider.get()?.finalizedBy(uploadSymbolsTask) // Если используете apk
            project.tasks.named("bundle${variant.name.capitalize()}").get()?.finalizedBy(uploadSymbolsTask) //
 Если используете aab
        }
    }
}
...
```

Manual loading

To manually load, enable and use the plugin in `offline` mode. This is necessary to link the mapping file to the app build.

1. In the `app/build.gradle` file, enable `offline` mode and run the build.

```
appmetrica {
    ...
    offline = true
```

```
}
```

offline

Enables offline mode. Acceptable values:

- `true` — Doesn't upload the mapping file to AppMetrica. If enabled, it outputs the archive path to the log after the build is complete.
- `false` — Automatically uploads the mapping file.

The default value is `false`.

2. In the AppMetrica interface, go to the app settings from the menu on the left.
3. Open **Crashes** → **Android**.
4. Click **Choose file** and upload the ZIP archive.

Build errors

Possible errors during a build:

- `IllegalStateException` — Enable code obfuscation using ProGuard or R8 Compiler.
- `HttpResponseException` — Check your internet connection.

If you encounter further errors, please contact technical support.

See also

Crashes and errors report

Related information

Android documentation

Changelog

Version 0.7.0

Released April 19, 2023.

- Added more details to the error reporting that mapping files are missing.
- Added support for AGP up to version 7.4.+ and Gradle up to 7.5.

Version 0.6.2

Released July 28, 2022

- Improved performance of the plugin.

Version 0.6.1

Released May 20, 2022

- Improved performance of the plugin.

Version 0.6.0

Released May 17, 2022

- Added the mappingFile parameter.
- Fixed the Cannot query the value of this property because it has no value available error that occurred when trying to get a mapping file.

Version 0.5.0

Released April 21, 2022

- Fixed the IndexOutOfBoundsException: toIndex (16) is greater than size (8) raised when processing an SO file.
- Fixed the Cannot query the value of this property because it has no value available crash that occurred when using the flag.
- Added the ndk.addNdkCrashesDependency flag.

Version 0.3.0

Released June 17, 2021

- Added support for gradle 7.
- Added sending analytics.

Version 0.2.4

Released May 26, 2021

- Added support for AGP 4.2.0.
- Improved performance and stability of the plugin.

Version 0.2.2

Released February 19, 2021

- Fixed the `Connection timed out` error when loading mapping files or symbols and using a proxy.

Version 0.2.1

Released 3 November 2020

- Added support for Android App Bundle.

  Attention: Don't run the `apk` and `aab` build with a single gradle command. This may lead to incorrect processing of crashes.
- Added logging.
- Improved SO file search and parsing.

Version 0.1.3

Released 12 June 2020

- Added automatic loading of characters from the SO files to track native crashes.
- Added the `split_version_codes` field to the `info.txt` file to store every loaded `version_code` of the app.

Version 0.1.2

Released 13 March 2020

- Fixed a crash during configuration process if the Post API key is not specified. There is a check during the boot run.
- Fixed the way to get mapping files.
- Added information about an error during uploading mapping files.
- Added the ability to specify custom plugin parameter values for different `ApplicationVariant`.

Version 0.0.1

Released 8 October 2019

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Deferred deeplinks support

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

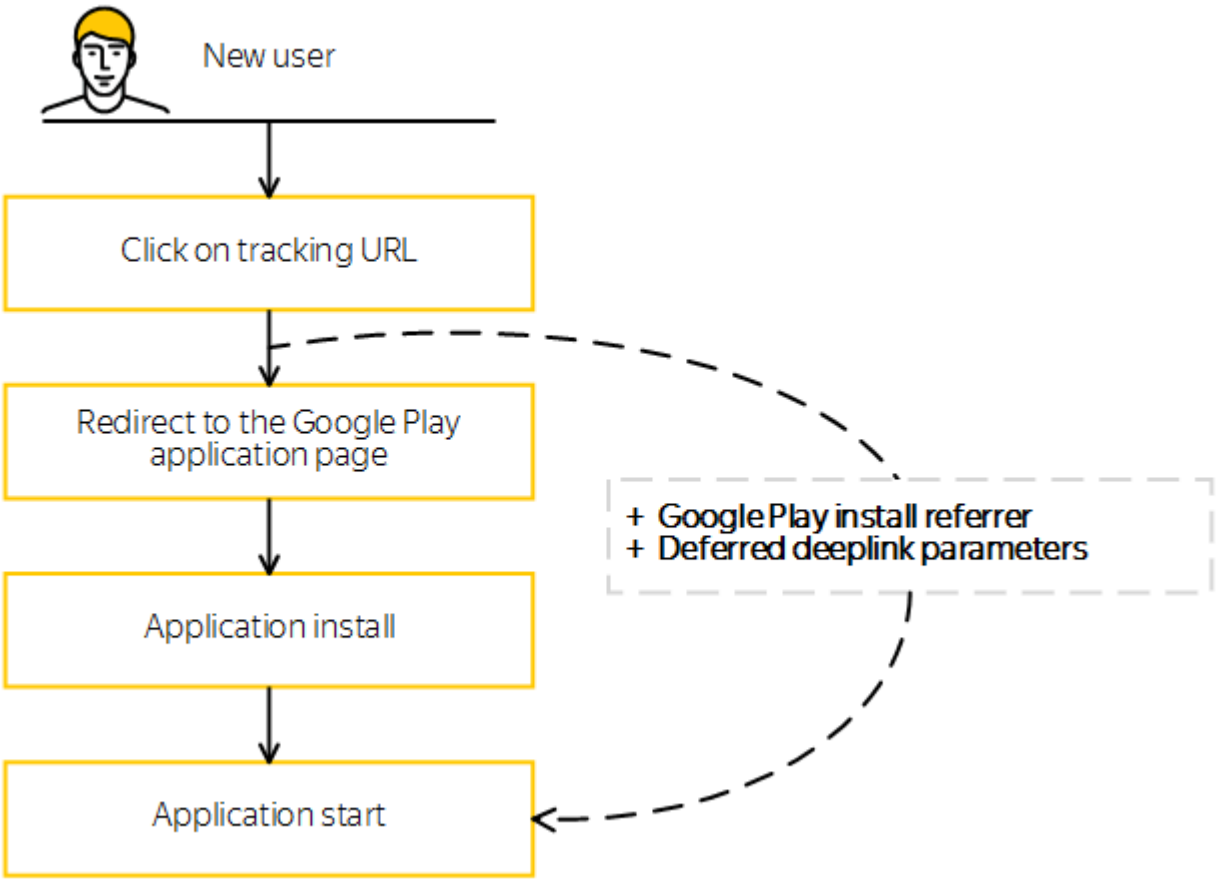
A deferred deeplink is used to transfer parameters to the application at the first start. These parameters can be used to perform actions in the application (to open specific application screen, to display certain content, etc.) depending on the source that led the user.

Unlike regular deeplinks, deferred deeplinks only work at the first application launch.

Note: Deferred deeplinks work only on Android devices.

How deferred deeplinks work

The following scheme shows the deferred deeplinks workflow:



- A user taps a tracking link that contains parameters of a deferred deeplink.
- The tracking URL directs the user to the application store.
- The user downloads the application.

- After the first application start, the deferred deeplink parameters are sent to the application. You should perform a request to get the parameters.

Error descriptions

The DeferredDeeplinkListener and DeferredDeeplinkParametersListener interfaces contain descriptions of errors that may occur when making the request:

`NOT_A_FIRST_LAUNCH`

The parameters or deferred deeplink can't be obtained since they can only be requested at the initial application start.

The first launch of the application is the session of the process (runtime environment, virtual machine) during which the user first requests the parameters. If the deeplink is present at the time of the request, the listener is invoked synchronously in the same thread. At the next start of the process, the library no longer sees the deferred deeplink and the `NOT_A_FIRST_LAUNCH` error is returned.

`PARSE_ERROR`

Depending on the interface, the error means the following:

- If the DeferredDeeplinkListener interface is used: couldn't find the deferred deeplink. This may happen if the referrer didn't contain the `appmetrica_deep_link` parameter.
- If the DeferredDeeplinkParametersListener interface is used: the deferred deeplink doesn't contain valid parameters.

  The `PARSE_ERROR` error is returned if one of the following conditions is not met:

  - `INSTALL_REFERRER` should contain the `appmetrica_deep_link` parameter.
  - The `appmetrica_deep_link` parameter value must contain the valid URI.
  - There must be at least one query parameter in the deeplink URI.

  A valid deeplink example: `sampleapp://samplepath?sampleparam1=samplevalue1`.

  Attention: The maximum number of characters that can be transferred in the deferred deeplink parameters is 7475.

See also

Requesting deferred deeplink parameters on page 27

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Error descriptions

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

The following sections describe common errors that you might encounter while working with the AppMetrica SDK on Android.

Error when adding the library to a project

```
UNEXPECTED TOP-LEVEL EXCEPTION:
com.android.dex.DexIndexOverflowException: method ID not in [0, 0xffff]: 65536
at com.android.dx.merge.DexMerger$6.updateIndex(DexMerger.java:502)
at com.android.dx.merge.DexMerger.mergeMethodIds(DexMerger.java:491)
at com.android.dx.merge.DexMerger$IdMerger.mergeSorted(DexMerger.java:277)
at com.android.dx.command.dexer.Main.runMonoDex(Main.java:303)
at com.android.dx.command.dexer.Main.mergeLibraryDexBuffers(Main.java:454)
at com.android.dx.merge.DexMerger.mergeDexes(DexMerger.java:168)
at com.android.dx.merge.DexMerger.merge(DexMerger.java:189)
at com.android.dx.command.dexer.Main.run(Main.java:246)
at com.android.dx.command.dexer.Main.main(Main.java:215)
```

```
at com.android.dx.command.Main.main(Main.java:106)
```

This error indicates that the method limit was exceeded at the DexIndexOverflowException stage of processing. We recommend reviewing the libraries used — perhaps they are very heavy. If they can't be replaced with lightweight alternatives, you can use multiple DEX files. This might increase the app loading time.

Error initializing AppMetrica with third-party libraries

The code in the `Application.onCreate()` method runs for all processes. If you encounter an initialization error after integrating a third-party library (such as Firebase Cloud Messaging), make sure that the third-party library is initialized only in the main process.

Error examples:

- ```
  Unable to create application your.package.name.YourApp: java.lang.IllegalStateException: Default FirebaseApp is
   not initialized in this process your.package.name:Metrica. Make sure to call FirebaseApp.initializeApp(Context)
   first.
  ```
- ```
  android.database.sqlite.SQLiteException: table httpauth already exists (code 1)
  ```
- ```
  Fatal Exception: java.lang.RuntimeException: Using WebView from more than one process at once with the same data
   directory is not supported.
  ```
- ```
  ANR at com.google.android.gms.ads.*
  ```

To fix the error, implement the main process check before initializing third-party libraries:

```
class MyApplication extends Application {
    @Override
    public void onCreate() {
        if (isMainProcess()) {
            // Initializing third-party libraries after verification.
        }
    }
}
```

Note: Implement verification of the main process on your own. See an example in the Stack Overflow answer.

Incorrect duration of user session during manual tracking

If the manual tracking of the user session is incorrectly implemented, it may lead to an inaccurate determination of its duration.

If the user session duration data doesn't look correct, make sure that when the user session ends, the YandexMetrica.pauseSession() method is always called. If this method is not called, the library considers that the session is active and commits regular data exchange with the server part of AppMetrica.

It is recommended to check the duration of the session timeout. It is specified with the `withSessionTimeout()` method. The timeout specifies the time interval during which the session will be considered active even after the application is closed.

High power consumption of the AppMetrica library

If there is the increased power consumption of the library, make sure that when the user session ends, the `YandexMetrica.pauseSession()` method is always called. If this method is not called, the library considers that the session is active and commits regular data exchange with the server part of AppMetrica.

It is recommended to check the duration of the session timeout. It is specified with the `withSessionTimeout()` method. The timeout specifies the time interval during which the session will be considered active even after the application is closed.

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Migration guide to branch 3.x.x

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

This guide contains examples that demonstrate the differences between the SDK versions 2.80 and 3.0.0. This section only covers methods that do not have backward compatibility.

Added

Reporter activation

Called before any actions with the reporter. If you already get the reporter, activation cannot change the parameters. Example:

```
YandexMetrica.activateReporter(this, ReporterConfig.newConfigBuilder(API_KEY)
             .withSessionTimeout(100)
             .build());
```

Profiles

Added the following classes to work with user profiles:

- `UserProfile`.
- `UserProfile.Builder`.
- `Attribute`.
- `NameAttribute`.
- `NotificationsEnabledAttribute`.
- `StringAttribute`.
- `NumberAttribute`.
- `GenderAttribute`.
- `BirthDateAttribute`.
- `BooleanAttribute`.
- `CounterAttribute`.

Added the following methods to the `YandexMetrica` class for user profiles:

- `setUserProfileID(String profileID)`.
- `reportUserProfile(UserProfile profile)`.

Added the following methods to the `IReporter` interface for user profiles:

- `setUserProfileID(String profileID)`.
- `reportUserProfile(UserProfile profile)`.

Revenue

Added the following classes for revenue tracking:

- `Revenue`.
- `Revenue.Builder`.
- `Revenue.Receipt`.
- `Revenue.Receipt.Builder`.

Added the following method to the `YandexMetrica` class:

- `reportRevenue(Revenue revenue)`.

Added the following method to `IReporter` interface:

- `reportRevenue(Revenue revenue)`.

Renamed

The `YandexMetrica` class.

Version 2.80	Version 3.0.0
`onResumeActivity(Activity activity)`	`// Resumes a session.` `resumeSession(Activity activity)`
`onPauseActivity(Activity activity)`	`// Pauses the session.` `pauseSession(Activity activity)`
`setTrackLocationEnabled(boolean enabled)`	`// Enables device location sending.` `setLocationTracking(boolean enabled)`

The `YandexMetricaConfig` class.

Version 2.80	Version 3.0.0
`getApiKey()`	`// API key of the application.` `public final String apiKey`
`getAppVersion()`	`// App Version.` `public final String appVersion`
`getSessionTimeout()`	`// Session timeout.` `public final Integer sessionTimeout`
`isReportCrashEnabled()`	`// Application crashes reporting.` `public final Boolean crashReporting`
`isReportNativeCrashEnabled()`	`// Application native crashes reporting.` `public final Boolean nativeCrashReporting`
`getLocation()`	`// Location data set by user.` `public final Location location`
`isTrackLocationEnabled()`	`// Sending device location.` `public final Boolean locationTracking`
`isCollectInstalledApps()`	`// Sending data about installed apps on the device.` `public final Boolean installedAppCollecting`
`isLogEnabled`	`// Library logs output.` `public final Boolean logs`
`getPreloadInfo()`	`// Сведения для отслеживания предустановленных приложений.` `public final PreloadInfo preloadInfo`

`handleFirstActivationAsUpdate()`	`// The first app launch with the AppMetrica SDK should be treated`
	`// as the first launch of the updated app version, and not as an install.`
	`public final Boolean firstActivationAsUpdate;`

The `YandexMetricaConfig.Builder` class.

Version 2.80	Version 3.0.0
`setAppVersion()`	`// Sets the app version.` `withAppVersion()`
`setSessionTimeout()`	`// Sets the session timeout.` `withSessionTimeout()`
`setReportCrashEnabled()`	`// Enables application crashes reporting.` `withCrashReporting()`
`setReportNativeCrashesEnabled()`	`// Enables application native crashes reporting.` `withNativeCrashReporting()`
`setLocation()`	`// Sets the device location.` `withLocation()`
`setTrackLocationEnabled()`	`// Enables device location sending.` `withLocationTracking()`
`setCollectInstalledApps()`	`// Enables sending information about pre-installed apps.` `withInstalledAppCollecting()`
`setLogEnabled`	`// Enables library logging.` `withLogs()`
`setPreloadInfo()`	`// Sets the data for tracking pre-installed apps.` `withPreloadInfo()`

Deleted

Version 2.80	Version 3.0.0

Activation with the API key
`YandexMetrica.activate(getApplicationContext(), API_KEY);`
	`YandexMetricaConfig.Builder configBuilder = YandexMetricaConfig.newConfigBuilder(API_KEY);`
	`YandexMetrica.activate(getApplicationContext(), configBuilder.build());`
	You can activate the library using the extended configuration `YandexMetricaConfig`.

Setting the session length. `YandexMetrica.setSessionTimeout(30);`	You can set the session timeout using the extended configuration `YandexMetricaConfig`.

Monitoring app crashes.
`YandexMetrica.setReportCrashesEnabled(true);`
`YandexMetrica.setReportNativeCrashesEnabled(true);`
	You can enable the crash reporting using the extended configuration `YandexMetricaConfig`

Setting the app version `YandexMetrica.setCustomAppVersion("1.0.5");`	You can set the application version using the extended configuration `YandexMetricaConfig`.
Enabling logs `YandexMetrica.setLogEnabled();`	You can enable the library logging using the extended configuration `YandexMetricaConfig`.

Version 2.80	Version 3.0.0

Enabling sending data about pre-installed apps.
`YandexMetrica.setCollectInstalledApps(boolean collect);`
`YandexMetrica.isCollectInstalledApps();`
	You can enable sending data about pre-installed apps using the extended configuration `YandexMetricaConfig`.

Setting environment values. `YandexMetrica.setEnvironmentValue(String key, String value);`	Deleted.

Contact support

If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Changelog

Attention:

This is an archived version of documentation. You can find the current documentation for all platforms here.

Version 5.3.0

Released March 3, 2023

- Fixed the `Class com.yandex.metrica.impl.ob.Zh failed lock verification and will run slower` warning (and similar ones) that could've appeared at app installation and launch.
- Fixed a possible ANR `at com.yandex.metrica.impl.ob.p3$b.onServiceConnected(SourceFile:2)`.
- Fixed errors and improved stability.

Version 5.2.0

Released September 19, 2022

- Added the AdRevenue API for transmitting advertising monetization revenue at the impression level (Impression-Level Revenue Data):
  - The AdRevenue, AdRevenue.Builder classes.
  - The reportAdRevenue(AdRevenue adRevenue) methods in the `YandexMetrica`, reportAdRevenue(AdRevenue adRevenue) class in the `IReporter` interface.
  - The AdType enumeration.
- targetSdkVersion = 33.

Version 5.0.1

Released July 29, 2022

- Fixed an issue that could cause repeated false installations.

Version 5.0.0

Released May 20, 2022

Note:

Starting from Android AppMetrica SDK 5.0.0 and higher, `appmetrica_device_id` changes when the app is reinstalled on a smartphone. This is due to new Google policies. For more information, see the documentation.

- Moved the feature of getting advertising IDs to a separate library named `com.yandex.android:mobmetricalib-identifiers`. For more information, see Getting advertising IDs.
- The flag to send information about the device location is disabled by default. To enable sending location data along with events, use one of the following methods: withLocationTracking(boolean enabled) or setLocationTracking(boolean enabled).
- Added `@NonNull` annotations for the arguments of the DeferredDeeplinkParametersListener interface methods.
- The reportNativeCrash method is deprecated and will be removed in future versions.

Version 4.2.0

Released February 17, 2022

- Added a new API to track crashes and errors from random plugins.

  Interfaces:

  - YandexMetricaPlugins
  - IPluginReporter

  Classes:

  - PluginErrorDetails
  - PluginErrorDetails.Builder
  - StackTraceItem
  - StackTraceItem.Builder
- Supported downgrade to version 4.2.0 and higher with saving important data, such as the device ID. It is not recommended to use downgrade, as it can cause data loss and corruption. Repeated attribution for such users will not be supported.
- Added support for Google Play Billing Library version 4.0.0. Support is still provided for versions 3.x.
- In the DeferredDeeplinkParametersListener method, the `referrer` argument became `@NonNull`. If it is missing, you will get an empty string.
- Removed ROOT status detection on devices running Android 12 and higher. Now all devices running Android 12 will be considered devices without ROOT access.
- Fixed errors and improved stability:
  - Fixed a possible crash when trying to connect to MetricaService: `Process: com.yandex :Metrica java.lang.RuntimeException: Unable to bind to service com.yandex.metrica.MetricaService Caused by: java.lang.IllegalArgumentException at android.os.Parcel.nativeAppendFrom(Native Method)`.
  - Fixed a possible ANR in the process: `Metrica: at com.yandex.metrica.impl.ob.jg.a (jg.java) at com.yandex.metrica.MetricaService.onConfigurationChanged(MetricaService.java:2)`.
  - Fixed a possible crash: `java.lang.NullPointerException: Attempt to read from null array at com.yandex.metrica.impl.ob.tz.a`.

Version 4.1.1

Released December 17, 2021

- Added com.google.android.gms.permission.AD_ID permission. For more information, see the documentation.
- In DeferredDeeplinkListener.Error and DeferredDeeplinkParametersListener.Error, added `NO_REFERRER` value.
- Added preliminary support for App Set ID to receive a referrer from Google Play (dependency `com.google.android.gms:play-services-appset:16.0.0`).
- SDK optimization is supported due to the correction of possible VerifyError when initializing classes.
- Mac address collection is disabled.

Version 4.0.0

Released September 20, 2021

- Google Play Install Referrer via BroadcastReceiver is no longer supported:
  - Removed MetricaEventHandler from AndroidManifest.
  - Removed the `YandexMetrica#registerReferrerBroadcastReceivers` method.
  - Removed the `YandexMetricaConfig#installedAppCollecting` flag.
  - Removed the `YandexMetricaConfig.Builder#withInstalledAppCollecting` methods.
- Added the YandexMetricaConfig.Builder#withSessionsAutoTrackingEnabled(boolean enabled) method for automatic session tracking.
- Added the revenueAutoTrackingEnabled method for the automatic collection of data on In-App purchases.
- Supported publication of the ".module" file for the Google Play SDK Console.
- Auto-tracking of deeplinks has been implemented and enabled by default.
- AppMetrica SDK migrated to AndroidX.
- Added basic support for Android 12: the main functionality of the AppMetrica SDK is available on Android 12 devices in apps with targetSdkVersion = 31.
- minSdkVersion has been upgraded to 14.

Version 3.21.1.

Released 3 June 2021

- The YandexMetricaConfig.Builder#withInstalledAppCollecting method is marked as `deprecated`. The flag value will be ignored (always `false`).

Version 3.21.0

Released May 26, 2021

- Added the YandexMetrica#initWebViewReporting method for sending events from the WebView's JS code.
- Upgraded Play Install Referrer Library to version 2.2.
- Crashpad supported for collecting native crashes. The native library ver. 2.0.0 and higher uses crashpad. Previous versions support breakpad. The mechanism used depends on the version of the native library attached.
- Fixed errors and improved stability.

Version 3.20.2

Released April 29, 2021

- Fixed Unity compatibility issue.

Version 3.20.1

Released April 14, 2021

- Added the option to set user profile ID during activation (YandexMetricaConfig.Builder#withUserProfileID) or before activation (YandexMetrica#setUserProfileID). Use this to get rid of empty profiles without an ID.
- Upgraded Play Install Referrer Library to version 2.1.
- Fixed incorrect deeplink processing from callback `Activity#onNewIntent(Intent)`. You need to make the following changes to the integration:

  Instead of calling:

  ```
  @Override
  protected void onNewIntent(final Intent intent) {
      super.onNewIntent(intent);
      YandexMetrica.reportAppOpen(this);
  }
  ```

  Call:

  ```
  @Override
  protected void onNewIntent(final Intent intent) {
      super.onNewIntent(intent);
      YandexMetrica.reportAppOpen(intent);
  }
  ```

- Fixed errors and improved stability.

Version 3.18.0

Released January 15, 2021

- Supported compliance with new Google policies in the context of location data collection.

Version 3.16.2

Released 13 November 2020

- Fixed a possible crash: `java.lang.NullPointerException: Attempt to invoke virtual method 'android.content.ComponentName android.app.Activity.getComponentName()' on a null object reference`.
- Fixed the price field logging in the Revenue class: GitHub Issue #31.

Version 3.16.1

Released 14 October 2020

- Added a new API for sending E-Commerce events:
  - Added the reportECommerce(ECommerceEvent event) method to the YandexMetrica and IReporter classes.
  - Added new classes:
    - ECommerceAmount
    - ECommerceCartItem
    - ECommerceEvent
    - ECommerceOrder
    - ECommercePrice
    - ECommerceProduct
    - ECommerceReferrer
    - ECommerceScreen

  For more information about E-Commerce events, see .
- Improved sending native crash data.
- Improved the performance of the library and the quality of statistical data.

Version 3.15.0

Released 29 July 2020

- Improved the performance of the library and the quality of statistical data.

Version 3.14.3

Released 9 July 2020

- Fixed a possible Mobile Ads SDK compatibility issue.

Version 3.14.2

Released 8 July 2020

- Added a new API for sending crashes and errors:
  - Added the putErrorEnvironment(String key, String value) method to the YandexMetrica class. You can use it to set the error environment.
  - Added the reportError(String groupIdentifier, String message) and reportError(String groupIdentifier, String message, Throwable error) methods to the YandexMetrica and IReporter classes. They can be used to pass IDs that crashes and errors are grouped by.
- Added the YandexMetrica.requestDeferredDeeplink(DeferredDeeplinkListener listener) method and DeferredDeeplinkListener interface for getting a deferred deeplink. For more information, see Usage examples.
- Added the AppMetrica SDK dependency on the Install Referrer Library.
- Stopped supporting the YandexMetrica.registerReferrerBroadcastReceivers(BroadcastReceiver... anotherReferrerReceivers) method. The method is deprecated since Google stopped sending Broadcast as of March 1, 2020.
- Improved the performance of the library and the quality of statistical data.

Version 3.13.1

Released 3 March 2020

- Improved the performance of the library and the quality of statistical data.

Version 3.10.0

Released 4 February 2020

- Fixed the reportReferralUrl method of the YandexMetrica class. It is no longer deprecated.
- Improved crash handling.
- Improved the performance of the library and the quality of statistical data.

Version 3.8.0

Released 8 October 2019

- Added the ability uploading mapping files using the special plugin.
- Improved the performance of the library and the quality of statistical data.

Version 3.7.2

Released 20 September 2019

- Improved the quality of event delivery.
- Added sending suppressed exceptions along with crashes.
- Added the withMaxReportsInDatabaseCount(int value) method to theYandexMetricaConfig.Builder class. It allows setting the maximum number of events that can be stored in a phone's database before being sent to AppMetrica.
- Added the maxReportsInDatabaseCount field to the YandexMetricaConfig class.
- Fixed a possible `java.lang.SecurityException` crash.

Version 3.6.4

Released 13 June 2019

- Fixed an issue that could cause events to be lost.
- Improved the performance of the library and the quality of statistical data.

Version 3.6.2

Released 21 May 2019

- Improved the performance of the library and the quality of statistical data.

Version 3.6.1

Released 15 May 2019

- Improved the performance of the library and the quality of statistical data.

Version 3.6.0

Released 23 April 2019

- Added the following to the Revenue class:
  - Method newBuilderWithMicros(). Use it instead of deprecated newBuilder().
  - Field priceMicros. Use it instead of deprecated price.

- Updated Google Breakpad. The new version supports all processor architectures. For more information, see Usage examples.

  Note: There are possible errors when collecting native crash data on devices with the x86 processor architecture.
- Improved crashes collecting during the application start.
- Added a thread dump for crashes.
- Fixed an issue that could cause memory leaks in threads.

Version 3.5.3

Released 30 January 2019

- Improved the performance of the library and the quality of statistical data.

Version 3.5.1

Released 28 December 2018

- Improved the performance of the library and the quality of statistical data.

Version 3.5.0

Released 14 December 2018

- Improved the performance of the library and the quality of statistical data.

Version 3.4.0

Released 5 November 2018

- Added support for Android 9.
- Improved library logging.
- Improved the performance of the library and the quality of statistical data.

Version 3.2.2

Released 9 August 2018

- Improved the performance of the library and the quality of statistical data.

Version 3.2.1

Released 30 July 2018

- Improved the performance of the library and the quality of statistical data.

Version 3.2.0

Released 23 July 2018

- Added the setStatisticsSending() method to disable statistics sending.
- Added the requestAppMetricaDeviceID() method to retrieve the AppMetrica device ID (`appmetrica_device_id`).
- Added the sendEventsBuffer() method to force sending stored events from the buffer.

- Improved the performance of the library and the quality of statistical data.

Version 3.1.0

Released 30 May 2018

- Added deeplink attribution (Re-engagement).
- Added a mechanism to speed up sending event postbacks for high-frequency events.
- Added integration with the `Play Install Referrer Library`. The library provides information about when the app started downloading from Google Play. This can improve attribution accuracy.

Version 3.0.0

Released 25 April 2018

- Added methods for creating user profiles.
- Added revenue tracking.
- Unified and revised the API. Read more in the migration guide to Android SDK 3.0.0.
- Improved support for working the library in several processes (with the restriction on activation with the same configuration).
- Changed the way of application crashes tracking in native code. For more information, see Usage examples.
- Improved performance and bug fixes.

Version 2.80

Released 27 December 2017

- Added fixes and minor improvements to the library for Android 8.
- Updated information about the integration process using `.jar`.
- The library's `API LEVEL` is **64**. Change it in the `AndroidManifest.xml` file.

Version 2.78

Released 19 November 2017

- Added fixes and minor improvements to the library.
- The library's `API LEVEL` is **63**. Change it in the `AndroidManifest.xml` file.

Version 2.77

Released 9 October 2017

- Added support for Android 8.
- Added fixes and minor improvements to the library.
- The library's `API LEVEL` is **62**. Change it in the `AndroidManifest.xml` file.

Version 2.76

Released 17 August 2017

- Added the `reportReferralUrl()` method which sets referral URL for app installs.
- Updated information about the integration process by using `.jar`.
- Added fixes and minor improvements to the library.
- The library's `API LEVEL` is **61**. Change it in the `AndroidManifest.xml` file.

Version 2.73

Released 15 June 2017

- Added fixes and minor improvements to the library.
- The library's `API LEVEL` is **58**. Change it in the `AndroidManifest.xml` file.

Version 2.71

Released 1 June 2017

- Added the `reportAppOpen(String deeplink)` method to track app openings with a string deeplink.
- Fixed an error occurred during the deferred deeplink parameters obtaining.
- The library's `API LEVEL` is **56**. Change it in the `AndroidManifest.xml` file.

Version 2.70

Released 16 May 2017

- Added the deferred deeplink support.
- Improved the stability of the library.
- Improved crash detection during the launch of the app.
- Added logging events attributes.
- The library's `API LEVEL` is **55**. Change it in the `AndroidManifest.xml` file.

Version 2.62

Released 30 December 2016

- Improved the stability of the library.
- Added fixes and minor improvements to the library.
- The library's `API LEVEL` is **52**. Change it in the `AndroidManifest.xml` file.

Version 2.60

Released 21 November 2016

- Added tracking of app openings with a deeplink.
- Added the ability to tell AppMetrica that the app was installed on the device before the AppMetrica SDK library was enabled. This allows AppMetrica to distinguish the first app start after integrating the SDK from the very first app start. This type of app launch won't be interpreted as a new user.
- Improved the stability of the library.
- The library's `API LEVEL` is **50**. Change it in the `AndroidManifest.xml` file.

Version 2.51

Released 29 September 2016

- Improved the stability of the library.
- Added support for Android 7.
- Prohibited using a Google account for making backup copies of AppMetrica SDK data.
- The library's `API LEVEL` is **48**. Change it in the `AndroidManifest.xml` file.

Version 2.42

Released 17 June 2016

- Improved performance and bug fixes.
- The library's `API LEVEL` is **45**. Change it in the `AndroidManifest.xml` file.

Version 2.41

Released 27 March 2016

- Optimized energy consumption when sending data.
- The library's `API LEVEL` is **44**. Change it in the `AndroidManifest.xml` file.

Version 2.40

Released 28 March 2016

- Added the ability to send statistics using an API key that differs from the app's API key.
- The library's `API LEVEL` is **43**. Change it in the `AndroidManifest.xml` file.

Version 2.32

Released 29 November 2016

- Improved the stability of the library and the quality of statistical data.
- The library's `API LEVEL` is **41**. Change it in the `AndroidManifest.xml` file.

Version 2.30

Released 16 December 2015

- Added the ability to initialize the library using the extended configuration, which guarantees that all the configuration parameters will be applied when the first event is sent.
- Added the ability to specify information for tracking pre-installed apps.
- Optimization for improved quality of statistics.
- The library's `API LEVEL` is **39**. Change it in the `AndroidManifest.xml` file.

Version 2.23

Released 25 November 2015

- Added support for Yandex Mobile Ads 2.00.
- The library's `API LEVEL` is **38**. Change it in the `AndroidManifest.xml` file.

Version 2.21

Released 20 October 2015

- The library's `API LEVEL` is **36**. Change it in the `AndroidManifest.xml` file.
- Removed the `YandexMetrica.isMetricaProcess()` method.
- Disabled transmitting data about installed apps by default.
- Added the `isCollectInstalledApps()` method for determining whether transmitting information about installed apps on the device is enabled.

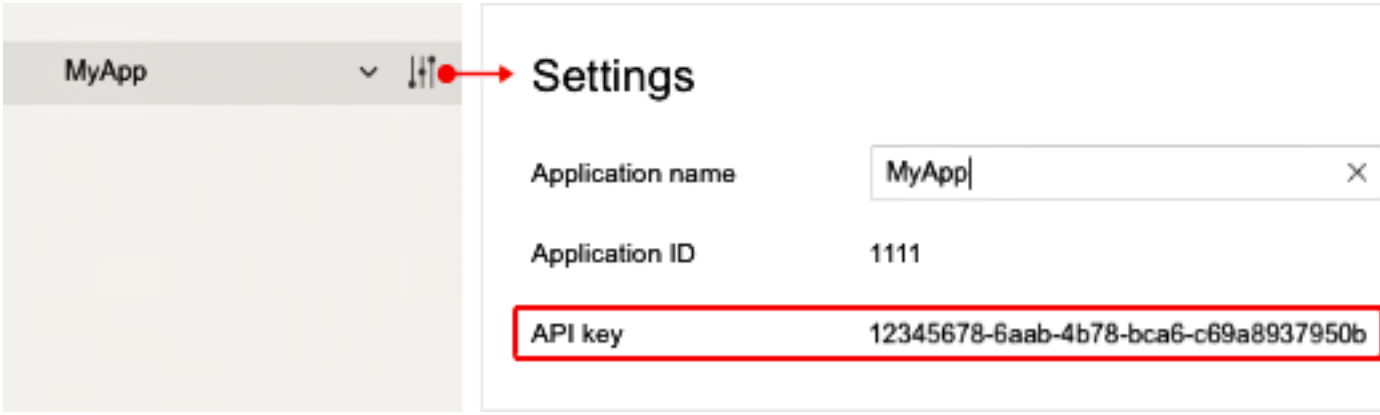
- Added the method `enableActivityAutoTracking(final Application application)` for enabling automatic app lifecycle tracking.
- Added the method `registerReferrerBroadcastReceivers(BroadcastReceiver...anotherReferrerReceivers)`, which allows registering any number of BroadcastReceivers in AppMetrica for tracking `INSTALL_REFERRER` from GooglePlay when using other tracking systems.
- The AAR format of the library is now added from Maven Central by default. To enable the library in JAR format, you must explicitly specify the classifier: `compile "com.yandex.android:mobmetricalib-internal:2.21:jar"`.
- Added the `setLogEnabled()` method, which lets you enable logging the library's activity.
- Fixed an error that led to app crashes: `java.lang.NullPointerException at com.yandex.metrica.impl.ob.f.b`.

Version 2.0

Released 27 August 2015

- The library's `API LEVEL` is **32**. Change it in the `AndroidManifest.xml` file.
- The format of the API key has been changed. The app ID in a new format is available in the AppMetrica web interface when the app editing mode is engaged.

  Where to find the API key



- The method of initializing the library in the app has been renamed from `initialize(android.content.Context, java.lang.String)` to `activate(android.content.Context, java.lang.String)`.
- Changed the length of the session timeout. Now it is 10 seconds, by default.
- The library has been adapted for Android M.
- Significantly improved performance and reduced power consumption.
- Deprecated methods have been removed.

  Click to see the list

  `setReportsEnabled(boolean enabled)` — lets you enable and disable sending reports.

  `setDispatchPeriod(int dispatchPeriodSeconds)` — allows you to set the interval in seconds between sending accumulated events to the server.

  `setMaxReportsCount(int maxReportsCount)` — Allows you to set the maximum number of events that can be stored up before sending all accumulated events to the server.

  `startNewSessionManually()` — allows you to start a new session manually.

  `sendEventsBuffer()` — Allows you to send all accumulated events without waiting for them to automatically be sent to the server.

Version 1.82

Released 19 June 2015

- The library's `API LEVEL` is **31**. Change it in the `AndroidManifest.xml` file.
- Improved quality of calculating statistics for sessions and installs.
- Migrated to `protobuf-nano` — the overall size of the library is smaller and it works faster. If you were using the `protobuf-2.5.0` library with the AppMetrica library, delete it.
- The `AndroidManifest.xml` file does not specify

  `provider`:

  ```
  <provider
      android:name="FULL_PACKAGE_PATH_TO_YOUR_PROVIDER.MetricaContentProvider"
      android:authorities="ROOT_PACKAGE_PATH.MetricaContentProvider"
      android:enabled="true"
      android:exported="true"/>
  ```

  the following `intent-filter` for `com.yandex.metrica.MetricaEventHandler`:

  ```
  <intent-filter>
      <action android:name="com.yandex.metrica.intent.action.SYNC"/>
  </intent-filter>

  <intent-filter>
      <action android:name="android.intent.action.PACKAGE_ADDED"/>
      <action android:name="android.intent.action.PACKAGE_DATA_CLEARED"/>
      <data android:scheme="package"/>
  </intent-filter>
  ```
- To use the AppMetrica library, you do not need to create a custom provider class in the application package with the name `MetricaContentProvider` that inherits from `com.yandex.metrica.MetricaContentProvider`. Delete it from your app.

Version 1.65

Released 24 February 2015

- The library's `API LEVEL` is **21**. Change it in the `AndroidManifest.xml` file.
- Improved reliability of statistics.

Version 1.60

Released 28 October 2014

- Added free support for tracking app installations.
- The library's `API LEVEL` is **16**. Change it in the `AndroidManifest.xml` file.
- The main API class has been renamed from `Counter` to `YandexMetrica`.
- All methods are now static, so any use of methods via the `Counter.sharedInstance()` object should involve the `YandexMetrica class` .
- The API key (metrica:api:key) in the `AndroidManifest.xml` file is ignored. It is set programmatically during library initialization in the app: the "activate" method passes the context and API key of your app.
- The `AndroidManifest.xml` file does not specify the receiver for `com.yandex.metrica.CampaignReceiver`. Its intent-filter is specified in `receiver` for `com.yandex.metrica.MetricaEventHandler`:

  ```
  ...
  <intent-filter>
      <action android:name="com.android.vending.INSTALL_REFERRER"/>
  </intent-filter>
  ...
  ```
- The `setLocation(double, double)` method is prohibited for setting location. Use the `setLocation(android.location.Location)` method.
- Added support for native app crashes.
- Added events with additional parameters (Json, Attributes).
- Improved stability and performance.
- Optimized library start.

- Improved location handling.