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

iOS

Installation and initialization

Installation and initialization

The AppMetrica library consists of two frameworks: core and crash-handling. If you don't use AppMetrica crash handling, add only the core part of the library. It reduces the size of the application.

The AppMetrica library supports the following platforms:

- iOS Extension;
- iOS Watch version 1.x (beta);
- tvOS.

Step 1. Enable the library

The library can work with the following dependency managers:

CocoaPods

The library supports static and dynamic frameworks for CocoaPods.

Note: To connect the AppMetrica SDK starting from 3.17.0, you need CocoaPods version 1.10 or higher.

To connect the crash-handling framework, add the following dependency to your project's `Podfile`:

- Static framework

```
pod 'YandexMobileMetrica', '4.5.2'
```

Example of enabling a static framework on GitHub.

- Dynamic framework

```
pod 'YandexMobileMetrica/Dynamic', '4.5.2'
```

To enable the library without crash handling, use the `YandexMobileMetrica/Static/Core` or `YandexMobileMetrica/Dynamic/Core` dependency.

Note: If the `Podfile` has the line use_frameworks!, we recommend using the dynamic framework.

Carthage

Note: To connect the AppMetrica SDK starting from 3.17.0, you need Carthage version 0.38 or higher.

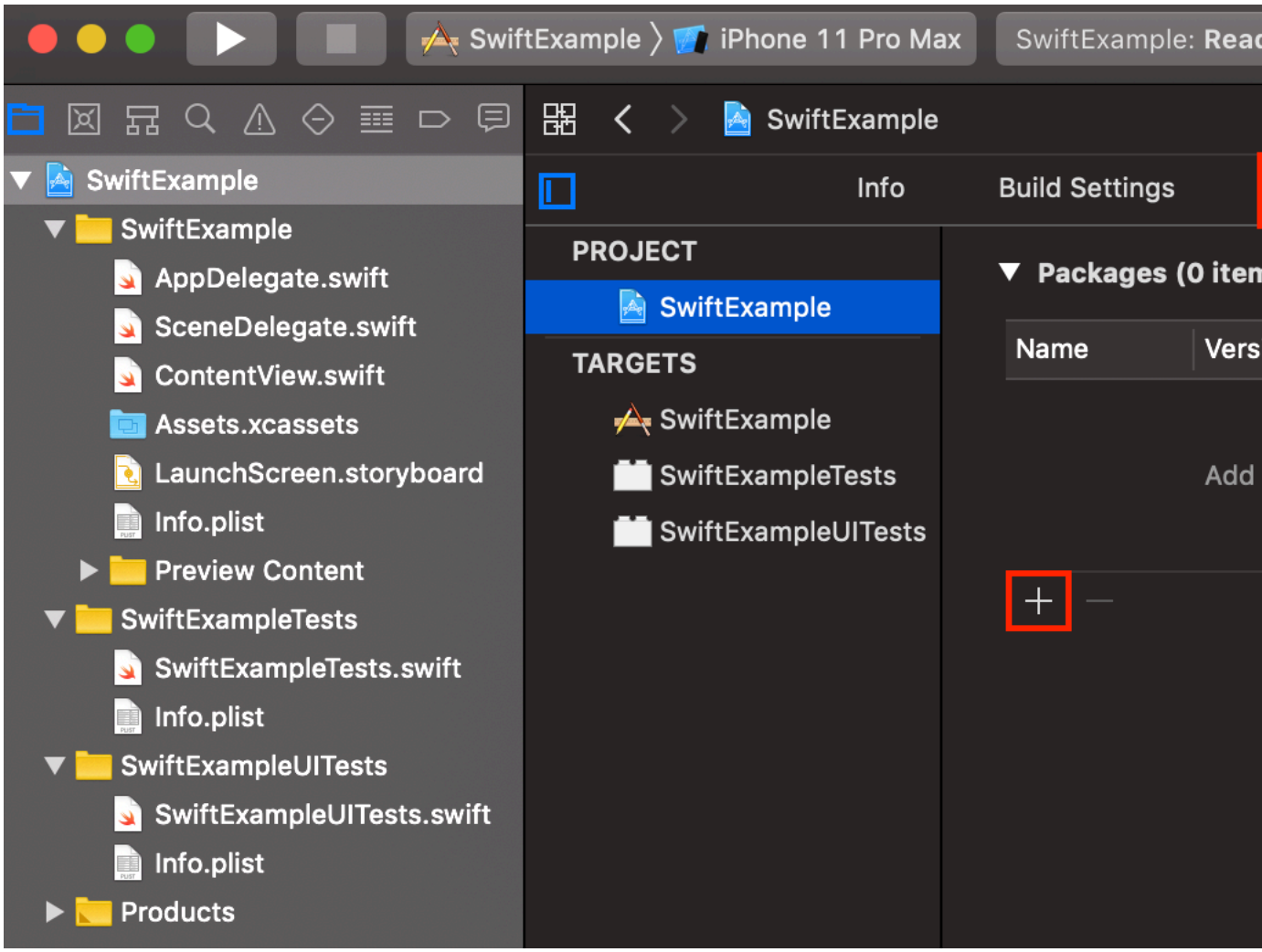
To connect the library, add the following dependency to `Cartfile` and save the file:

```
binary "https://raw.githubusercontent.com/yandexmobile/metrica-sdk-ios/master/YandexMobileMetrica.json" ~> 4.5.2
```

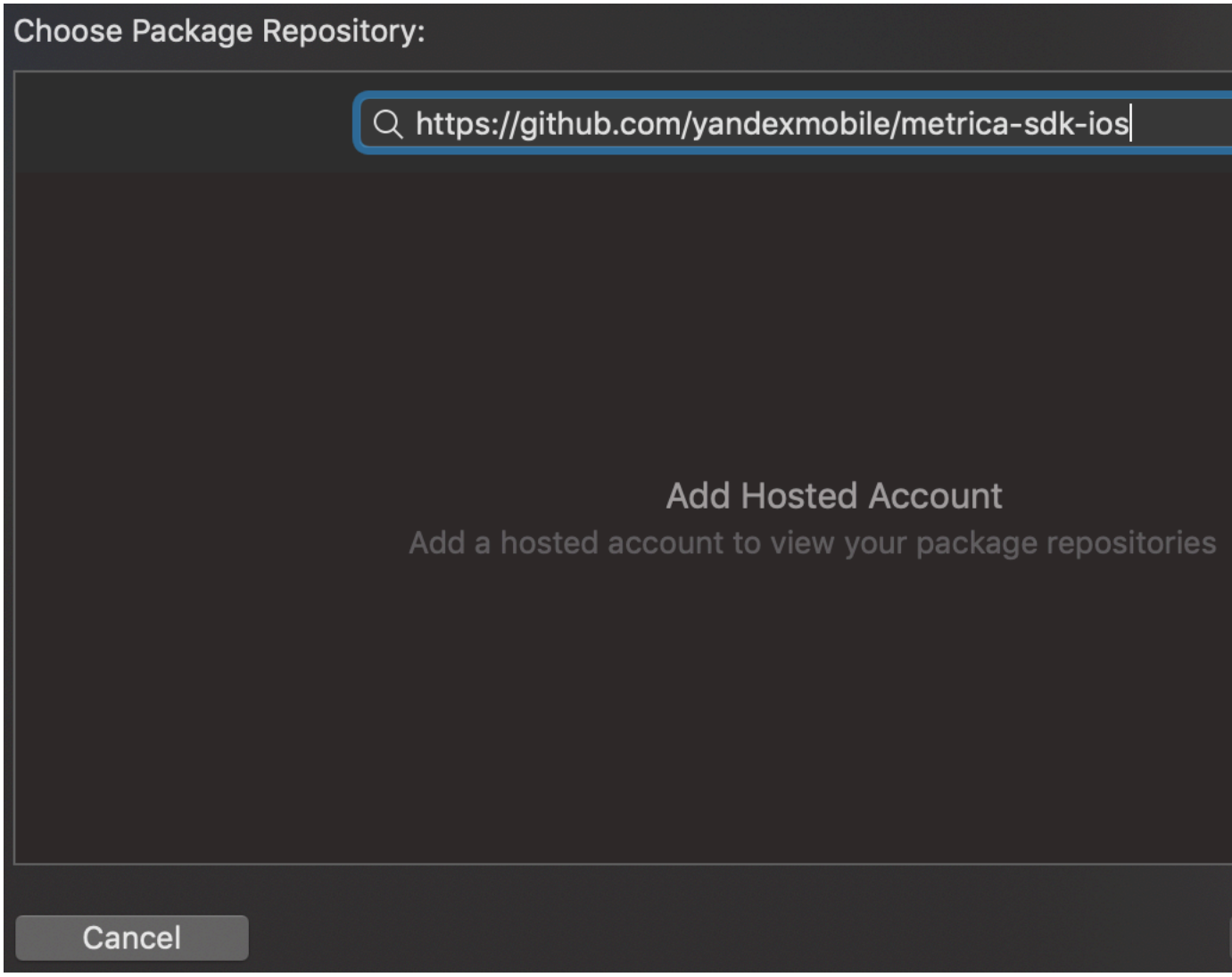
Swift Package Manager

To connect the library, follow these steps:

1. In Xcode, go to the **Swift Packages** tab for your project.



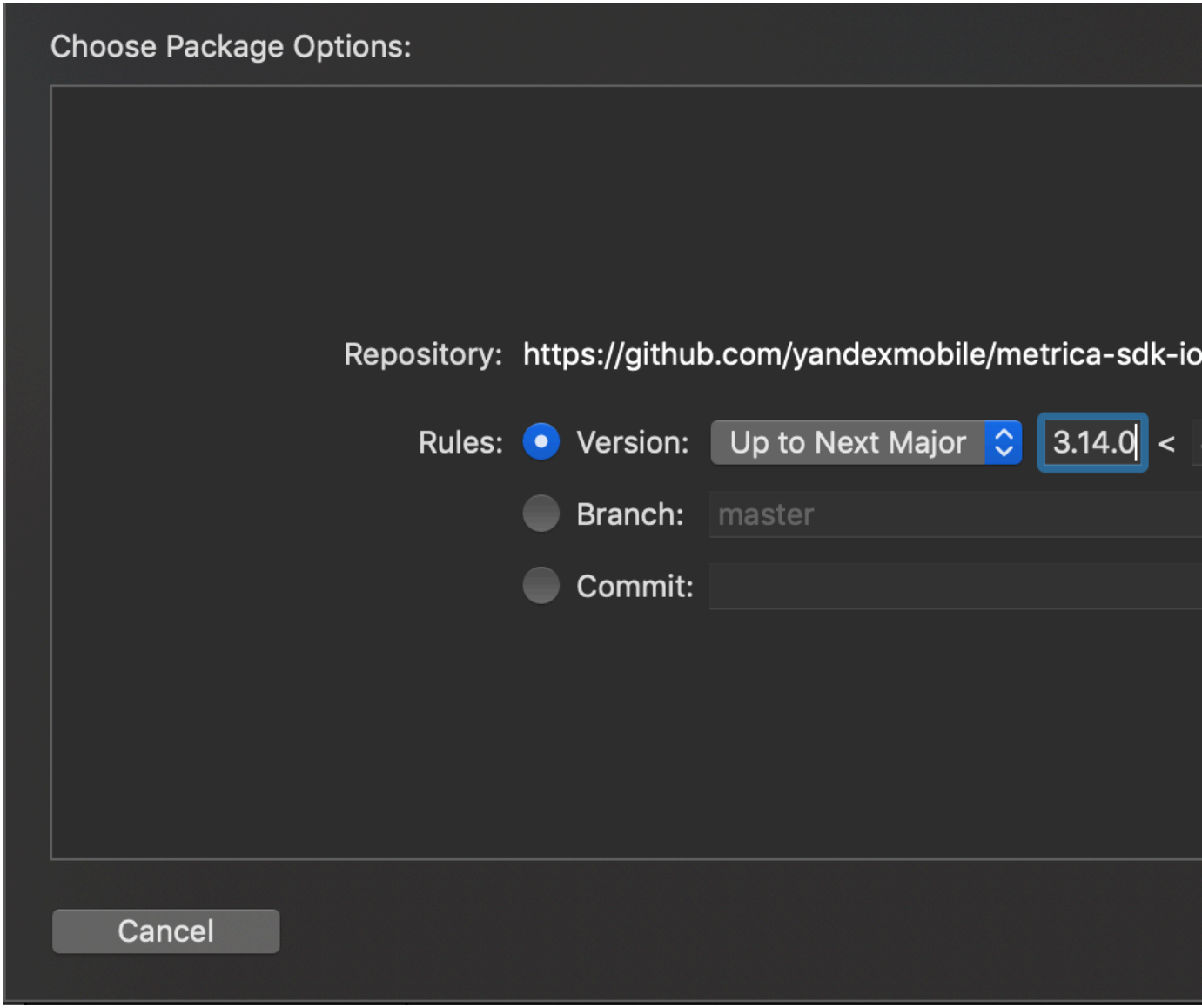
2. Specify the repository URL `https://github.com/yandexmobile/metrica-sdk-ios`, which contains a Swift package.



3. Configure a rule for selecting the package version.

   Restriction:

   Connection using Swift Package Manager is supported starting from version 3.14.0 of the AppMetrica SDK.



4. Select the required libraries.

If you don't use these dependency managers

To enable the library, follow these steps:

1. Download the AppMetrica library.
2. Add `YandexMobileMetrica.framework` to the project.
3. *(Optional)* To enable crash handling, add `YandexMobileMetricaCrashes.framework`.
4. Add the following dependencies: 'SystemConfiguration', 'UIKit', 'Foundation', 'CoreTelephony', 'CoreLocation', 'CoreGraphics', 'AdSupport', 'z', 'sqlite3', 'Security', 'c++', 'WebKit', and 'SafariServices' (with the **Optional** setting).
5. Add `-ObjC` to `Other Linker Flags`.

Step 2. Initialize the library

Objective-C

Add the import:

```
#import <YandexMobileMetrica/YandexMobileMetrica.h>
```

Initialize the library in the `application:didFinishLaunchingWithOptions:` method of your `UIApplicationDelegate`:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Initializing the AppMetrica SDK.
    YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
    [YMMYandexMetrica activateWithConfiguration:configuration];
}
```

Swift

Add the import:

```
import YandexMobileMetrica
```

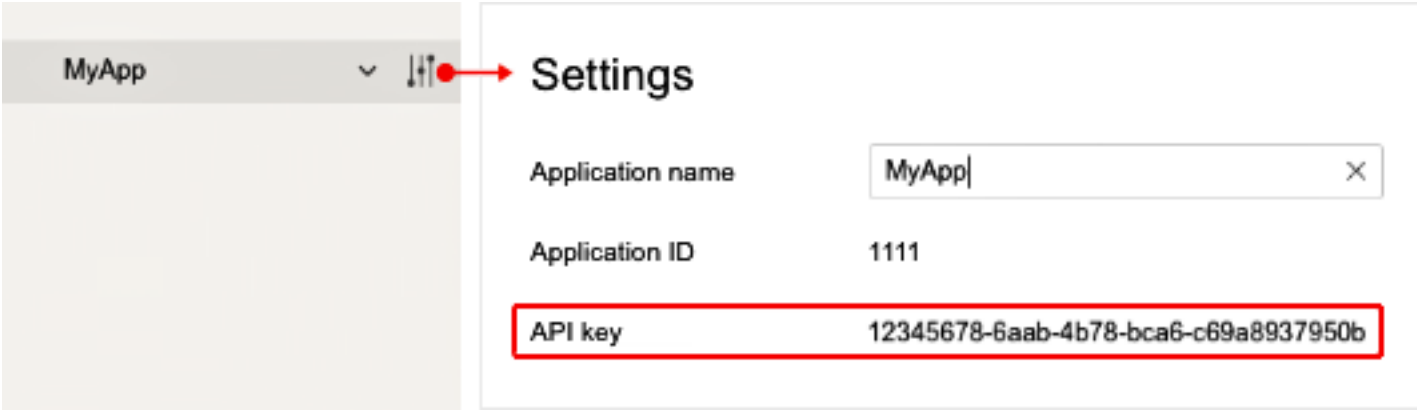
Initialize the library in the `application(_:didFinishLaunchingWithOptions:)` method of your `UIApplicationDelegate`:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
 [UIApplicationLaunchOptionsKey : Any]? = nil) -> Bool
{
    // Initializing the AppMetrica SDK.
    let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API_key")
    YMMYandexMetrica.activate(with: configuration!)
}
```

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. For more information, see Tracking pre-installed apps.

Note: Requirements: deployment target 8.0 and higher.

Step 3. *(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 4. Test the library operation

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.

Problems with Swift Package Manager

Read the article Problems when using Swift Package Manager.

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 an Apple IDFA

   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 Apple IDFA is shown in the **IDFA** 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

My app didn't pass App Store moderation
How to enable user location sending

Related information

Example of library integration

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.

Installation and initialization

The AppMetrica library consists of two frameworks: core and crash-handling. If you don't use AppMetrica crash handling, add only the core part of the library. It reduces the size of the application.

The AppMetrica library supports the following platforms:

- iOS;
- iOS Watch version 1.x (beta);
- tvOS.

Step 1. Enable the library

The library can work with the following dependency managers:

CocoaPods

The library supports static and dynamic frameworks for CocoaPods.

Note: To connect the AppMetrica SDK starting from 3.17.0, you need CocoaPods version 1.10 or higher.

To connect the crash-handling framework, add the following dependency to your project's `Podfile`:

- Static framework

```
pod 'YandexMobileMetrica', '4.5.2'
```

  Example of enabling a static framework on GitHub.
- Dynamic framework

```
pod 'YandexMobileMetrica/Dynamic', '4.5.2'
```

To enable the library without crash handling, use the `YandexMobileMetrica/Static/Core` or `YandexMobileMetrica/Dynamic/Core` dependency.

Note: If the `Podfile` has the line use_frameworks!, we recommend using the dynamic framework.

Carthage

Note: To connect the AppMetrica SDK starting from 3.17.0, you need Carthage version 0.38 or higher.

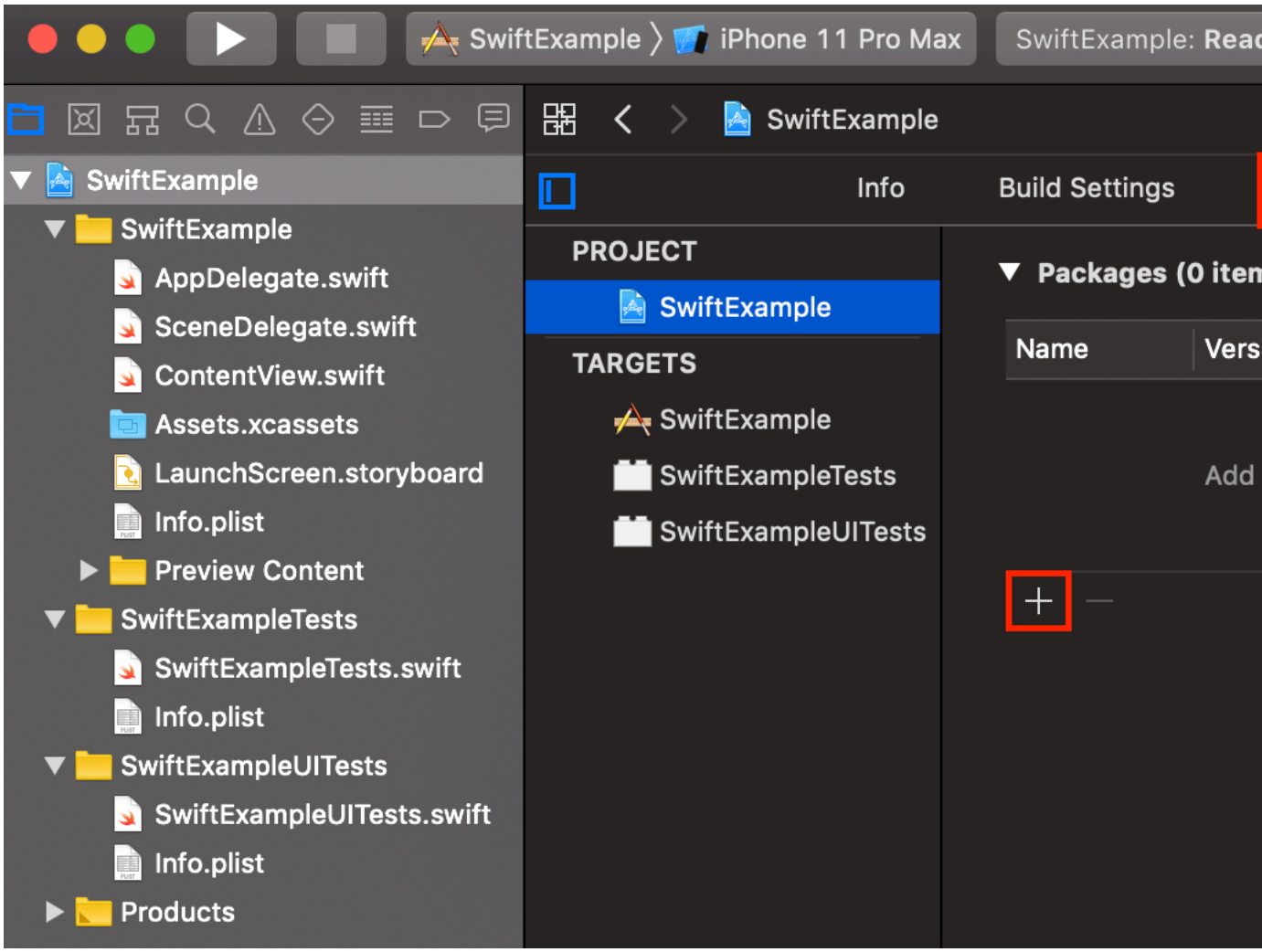
To connect the library, add the following dependency to `Cartfile` and save the file:

```
binary "https://raw.githubusercontent.com/yandexmobile/metrica-sdk-ios/master/YandexMobileMetrica.json" ~> 4.5.2
```

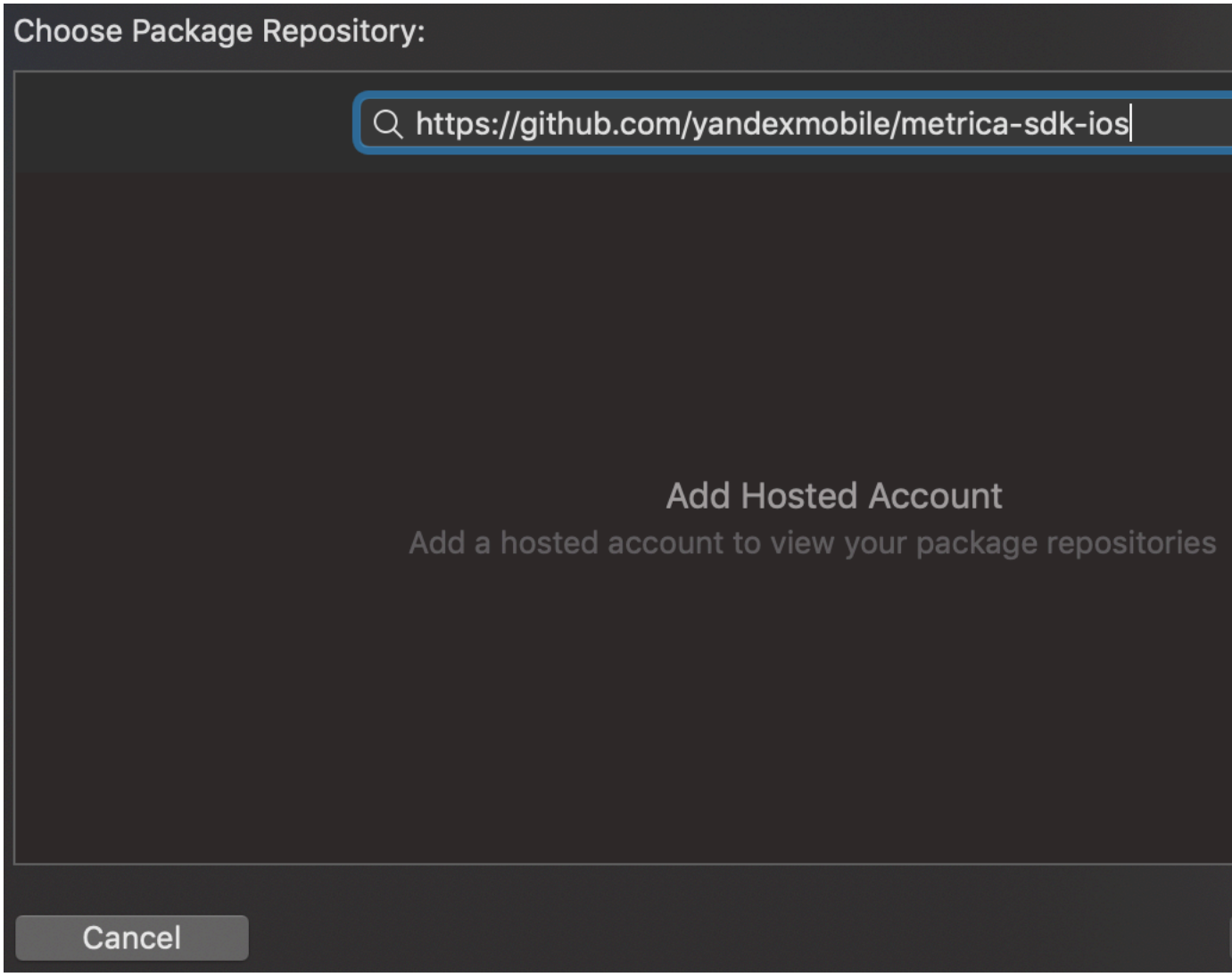
Swift Package Manager

To connect the library, follow these steps:

1. In Xcode, go to the **Swift Packages** tab for your project.



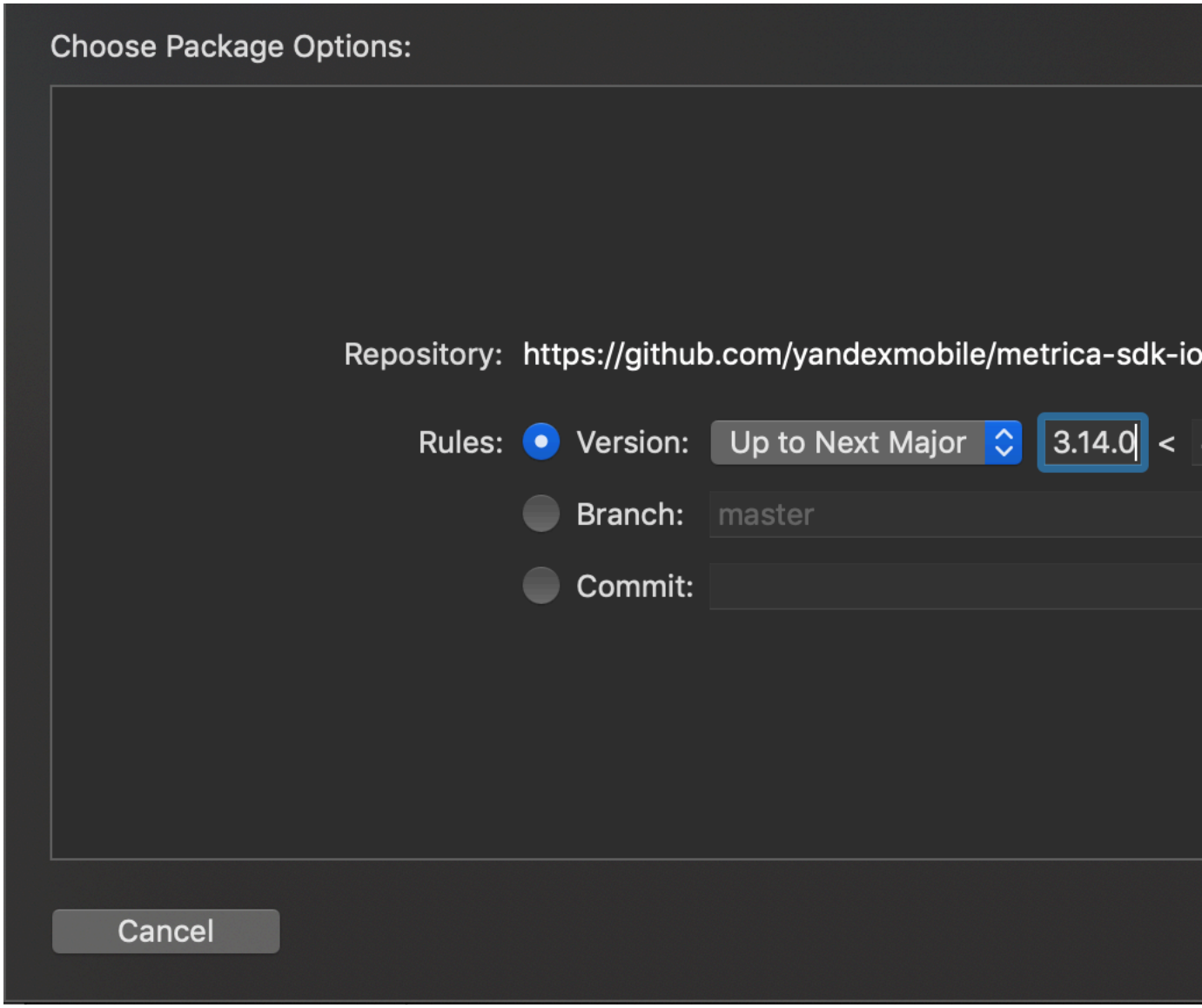
2. Specify the repository URL `https://github.com/yandexmobile/metrica-sdk-ios`, which contains a Swift package.



3. Configure a rule for selecting the package version.

   Restriction:

   Connection using Swift Package Manager is supported starting from version 3.14.0 of the AppMetrica SDK.



4. Select the required libraries.

If you don't use these dependency managers

To enable the library, follow these steps:

1. Download the AppMetrica library.
2. Add `YandexMobileMetrica.framework` to the project.
3. *(Optional)* To enable crash handling, add `YandexMobileMetricaCrashes.framework`.
4. Add the following dependencies: 'SystemConfiguration', 'UIKit', 'Foundation', 'CoreTelephony', 'CoreLocation', 'CoreGraphics', 'AdSupport', 'z', 'sqlite3', 'Security', 'c++', 'WebKit', and 'SafariServices' (with the **Optional** setting).
5. Add `-ObjC` to `Other Linker Flags`.

Step 2. Initialize the library

Objective-C

Add the import:

```
#import <YandexMobileMetrica/YandexMobileMetrica.h>
```

Initialize the library in the `initialize` method of the `NSExtensionPrincipalClass` extension:

```
+ (void)initialize
{
    if ([self class] == [MMSTodayViewController class]) {
        // Initializing the AppMetrica SDK.
        YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
        [YMMYandexMetrica activateWithConfiguration:configuration];
    }
}
```

Swift

Add the import:

```
import YandexMobileMetrica
```

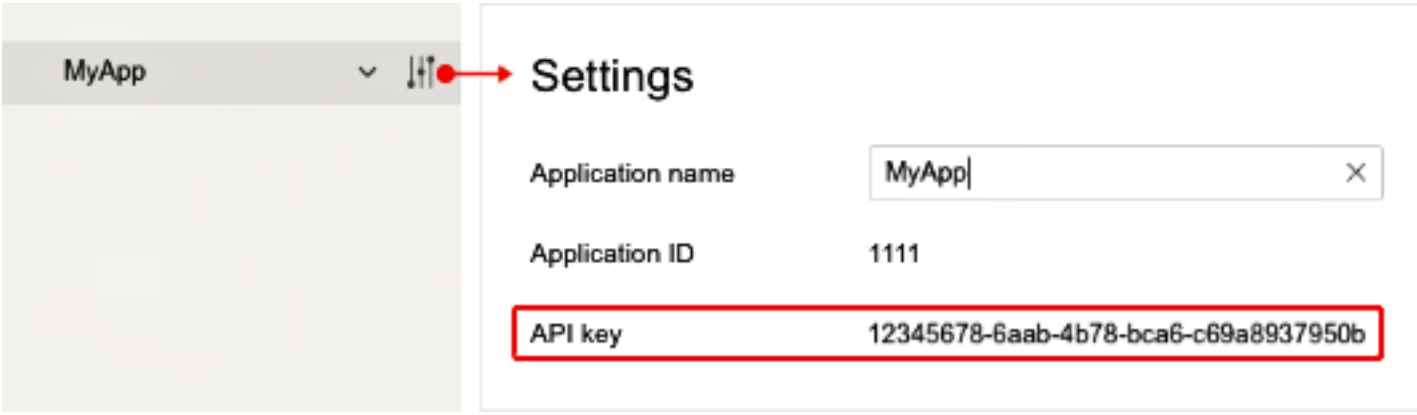
Initialize the library in the `initialize` method of the `NSExtensionPrincipalClass` extension:

```
override class func initialize() {
    if self === MMSTodayViewController.self {
        // Initializing the AppMetrica SDK.
        let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API_key")
        YMMYandexMetrica.activate(with: configuration!)
    }
}
```

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. For more information, see Tracking pre-installed apps.

Note: Requirements: deployment target 8.0 and higher.

Step 3. *(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 4. Test the library operation

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.

Problems with Swift Package Manager

Read the article Problems when using Swift Package Manager.

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 an Apple IDFA

   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 Apple IDFA is shown in the **IDFA** 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

My app didn't pass App Store moderation
How to enable user location sending

Related information

Example of library integration

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.

Installation and initialization

The AppMetrica library consists of two frameworks: core and crash-handling. If you don't use AppMetrica crash handling, add only the core part of the library. It reduces the size of the application.

The AppMetrica library supports the following platforms:

- iOS;
- iOS Extension;
- tvOS.

Step 1. Enable the library

The library can work with the following dependency managers:

CocoaPods

The library supports static and dynamic frameworks for CocoaPods.

Note: To connect the AppMetrica SDK starting from 3.17.0, you need CocoaPods version 1.10 or higher.

To connect the crash-handling framework, add the following dependency to your project's `Podfile`:

- Static framework

```
pod 'YandexMobileMetrica', '4.5.2'
```

  Example of enabling a static framework on GitHub.
- Dynamic framework

```
pod 'YandexMobileMetrica/Dynamic', '4.5.2'
```

To enable the library without crash handling, use the `YandexMobileMetrica/Static/Core` or `YandexMobileMetrica/Dynamic/Core` dependency.

Note: If the `Podfile` has the line use_frameworks!, we recommend using the dynamic framework.

Carthage

Note: To connect the AppMetrica SDK starting from 3.17.0, you need Carthage version 0.38 or higher.

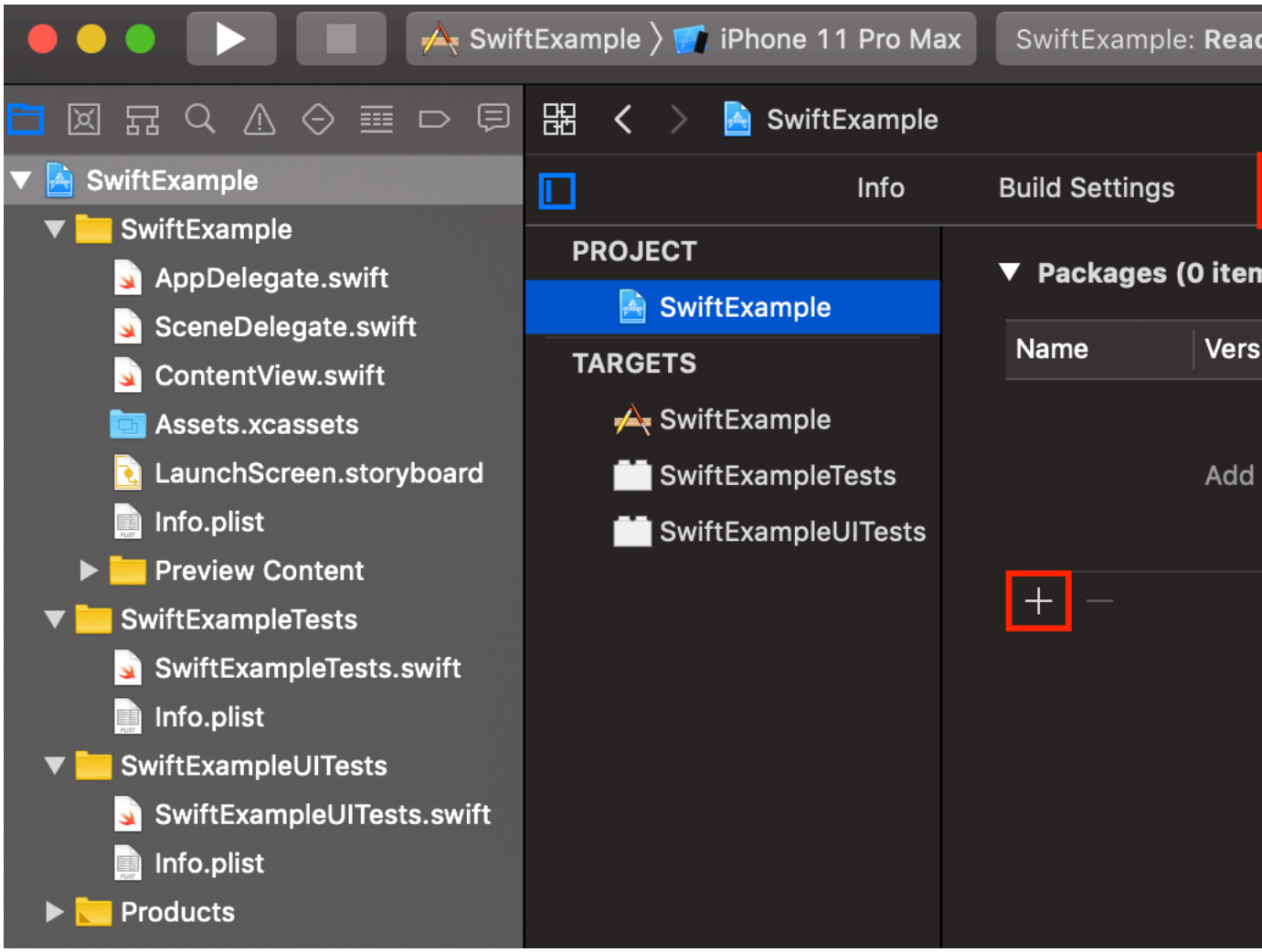
To connect the library, add the following dependency to `Cartfile` and save the file:

```
binary "https://raw.githubusercontent.com/yandexmobile/metrica-sdk-ios/master/YandexMobileMetrica.json" ~> 4.5.2
```

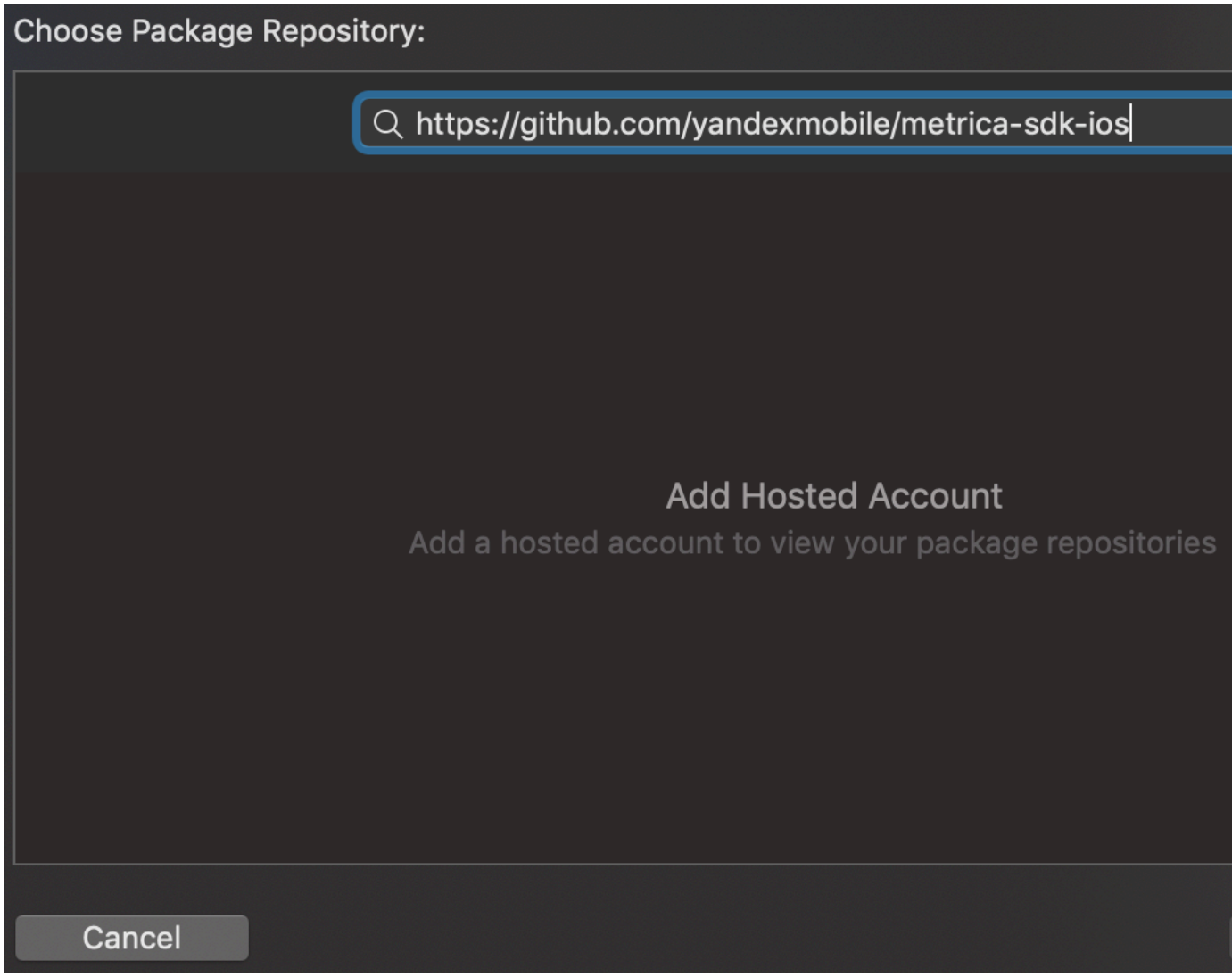
Swift Package Manager

To connect the library, follow these steps:

1. In Xcode, go to the **Swift Packages** tab for your project.



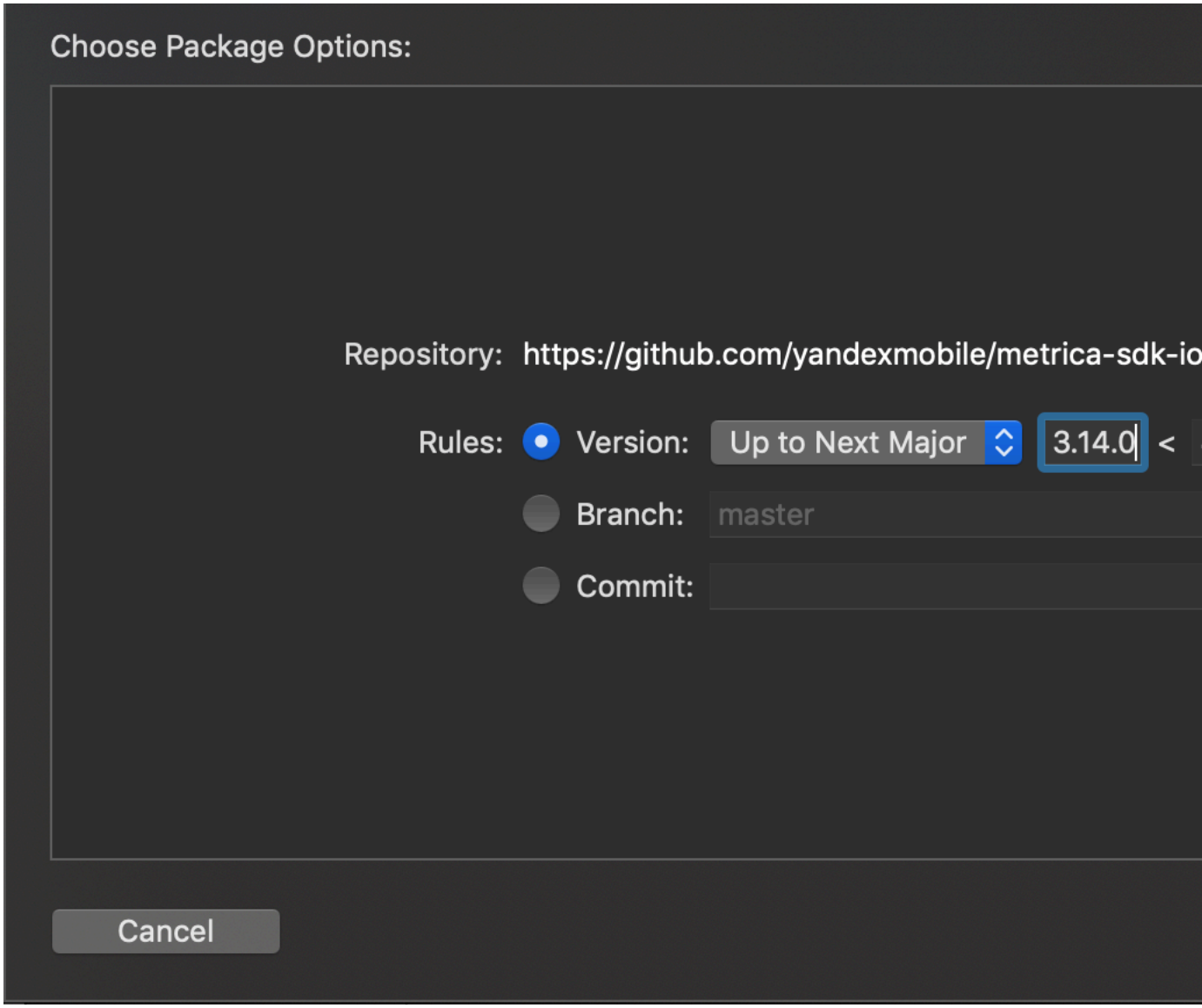
2. Specify the repository URL `https://github.com/yandexmobile/metrica-sdk-ios`, which contains a Swift package.



3. Configure a rule for selecting the package version.

   Restriction:

   Connection using Swift Package Manager is supported starting from version 3.14.0 of the AppMetrica SDK.



4. Select the required libraries.

If you don't use these dependency managers

To enable the library, follow these steps:

1. Download the AppMetrica library.
2. Add `YandexMobileMetrica.framework` to the project.
3. *(Optional)* To enable crash handling, add `YandexMobileMetricaCrashes.framework`.
4. Add the following dependencies: 'SystemConfiguration', 'UIKit', 'Foundation', 'CoreTelephony', 'CoreLocation', 'CoreGraphics', 'AdSupport', 'z', 'sqlite3', 'Security', 'c++', 'WebKit', and 'SafariServices' (with the **Optional** setting).
5. Add `-ObjC` to `Other Linker Flags`.

Step 2. Initialize the library

Objective-C

Add the import:

```
#import <YandexMobileMetrica/YandexMobileMetrica.h>
```

Initialize the library in the `initialize` method in `Main Entry Point` of the extension (there may be multiple activation points):

```
+ (void)initialize
{
    if ([self class] == [MMSInterfaceController class]) {
        // Initializing the AppMetrica SDK.
        YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
        [YMMYandexMetrica activateWithConfiguration:configuration];
    }
}
```

or

```
+ (void)initialize
{
    if ([self class] == [MMSNotificationController class]) {
        // Initializing the AppMetrica SDK.
        YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
        [YMMYandexMetrica activateWithConfiguration:configuration];
    }
}
```

Swift

Add the import:

```
import YandexMobileMetrica
```

Initialize the library in the `initialize` method in `Main Entry Point` of the extension (there may be multiple activation points):

```
override class func initialize() {
    if self === MMSInterfaceController.self {
        // Initializing the AppMetrica SDK.
        let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API_key")
        YMMYandexMetrica.activate(with: configuration!)
    }
}
```

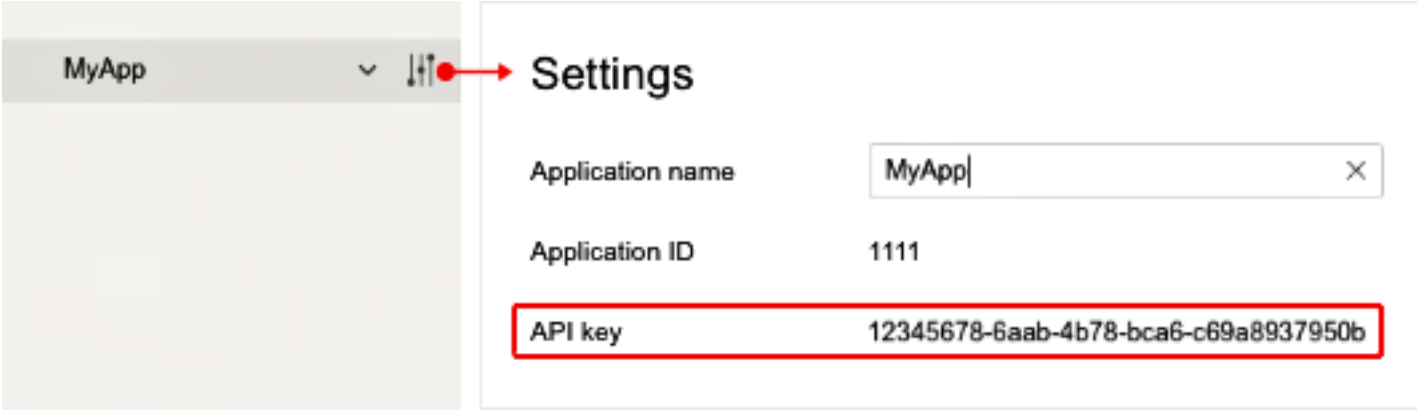
or

```
override class func initialize() {
    if self === MMSNotificationController.self {
        // Initializing the AppMetrica SDK.
        let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API_key")
        YMMYandexMetrica.activate(with: configuration!)
    }
}
```

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. For more information, see Tracking pre-installed apps.

Note: Requirements: deployment target 8.0 and higher.

Step 3. *(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 4. Test the library operation

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.

Problems with Swift Package Manager

Read the article Problems when using Swift Package Manager.

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 an Apple IDFA

   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 Apple IDFA is shown in the **IDFA** 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

My app didn't pass App Store moderation
How to enable user location sending

Related information

Example of library integration

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.

Installation and initialization

The AppMetrica library consists of two frameworks: core and crash-handling. If you don't use AppMetrica crash handling, add only the core part of the library. It reduces the size of the application.

The AppMetrica library supports the following platforms:

- iOS;
- iOS Extension;
- iOS Watch version 1.x (beta).

Step 1. Enable the library

The library can work with the following dependency managers:

CocoaPods

The library supports static and dynamic frameworks for CocoaPods.

Note: To connect the AppMetrica SDK starting from 3.17.0, you need CocoaPods version 1.10 or higher.

To connect the crash-handling framework, add the following dependency to your project's `Podfile`:

- Static framework

```
pod 'YandexMobileMetrica', '4.5.2'
```

Example of enabling a static framework on GitHub.

- Dynamic framework

```
pod 'YandexMobileMetrica/Dynamic', '4.5.2'
```

To enable the library without crash handling, use the `YandexMobileMetrica/Static/Core` or `YandexMobileMetrica/Dynamic/Core` dependency.

Note: If the `Podfile` has the line use_frameworks!, we recommend using the dynamic framework.

Carthage

Note: To connect the AppMetrica SDK starting from 3.17.0, you need Carthage version 0.38 or higher.

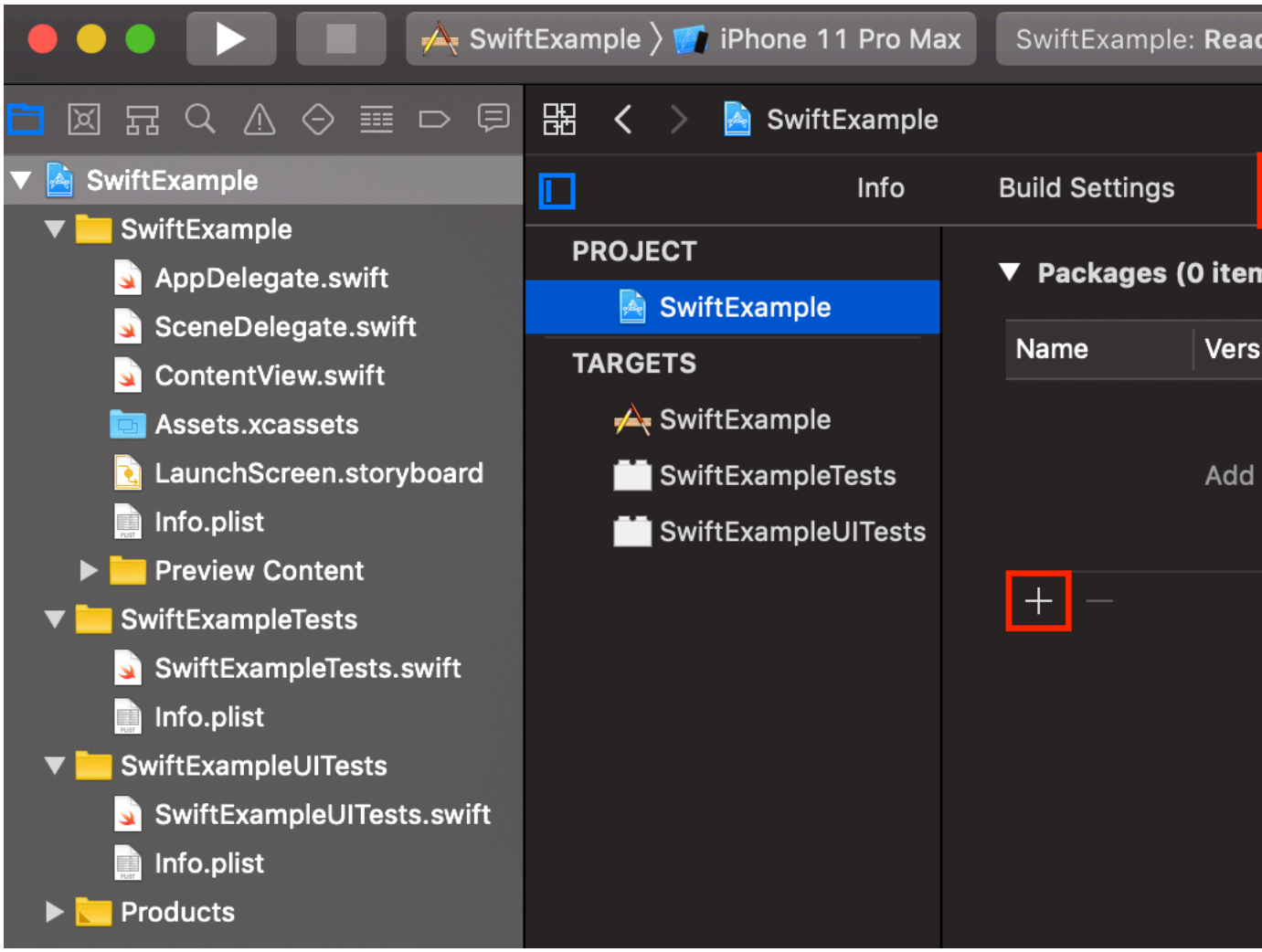
To connect the library, add the following dependency to `Cartfile` and save the file:

```
binary "https://raw.githubusercontent.com/yandexmobile/metrica-sdk-ios/master/YandexMobileMetrica.json" ~> 4.5.2
```

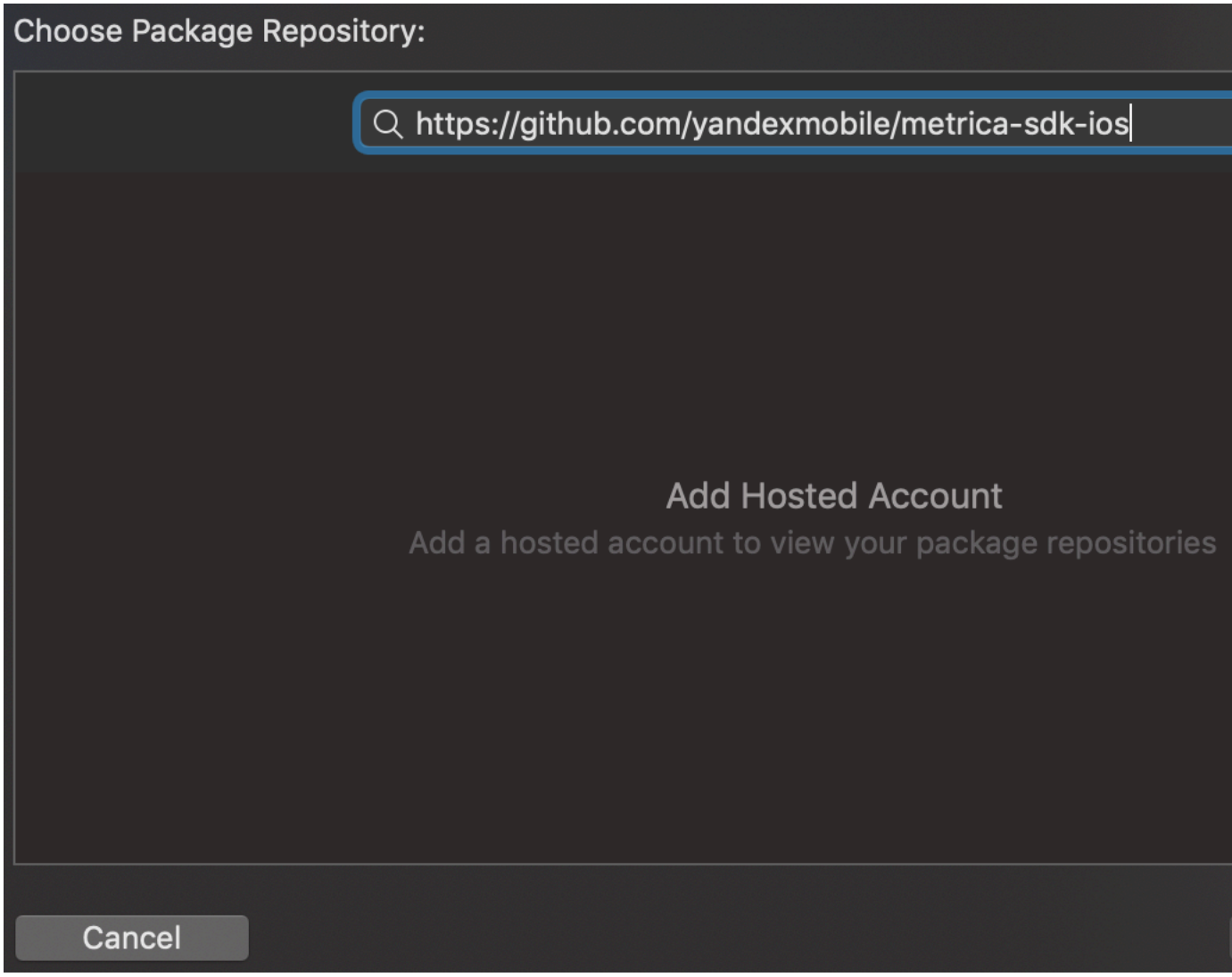
Swift Package Manager

To connect the library, follow these steps:

1. In Xcode, go to the **Swift Packages** tab for your project.



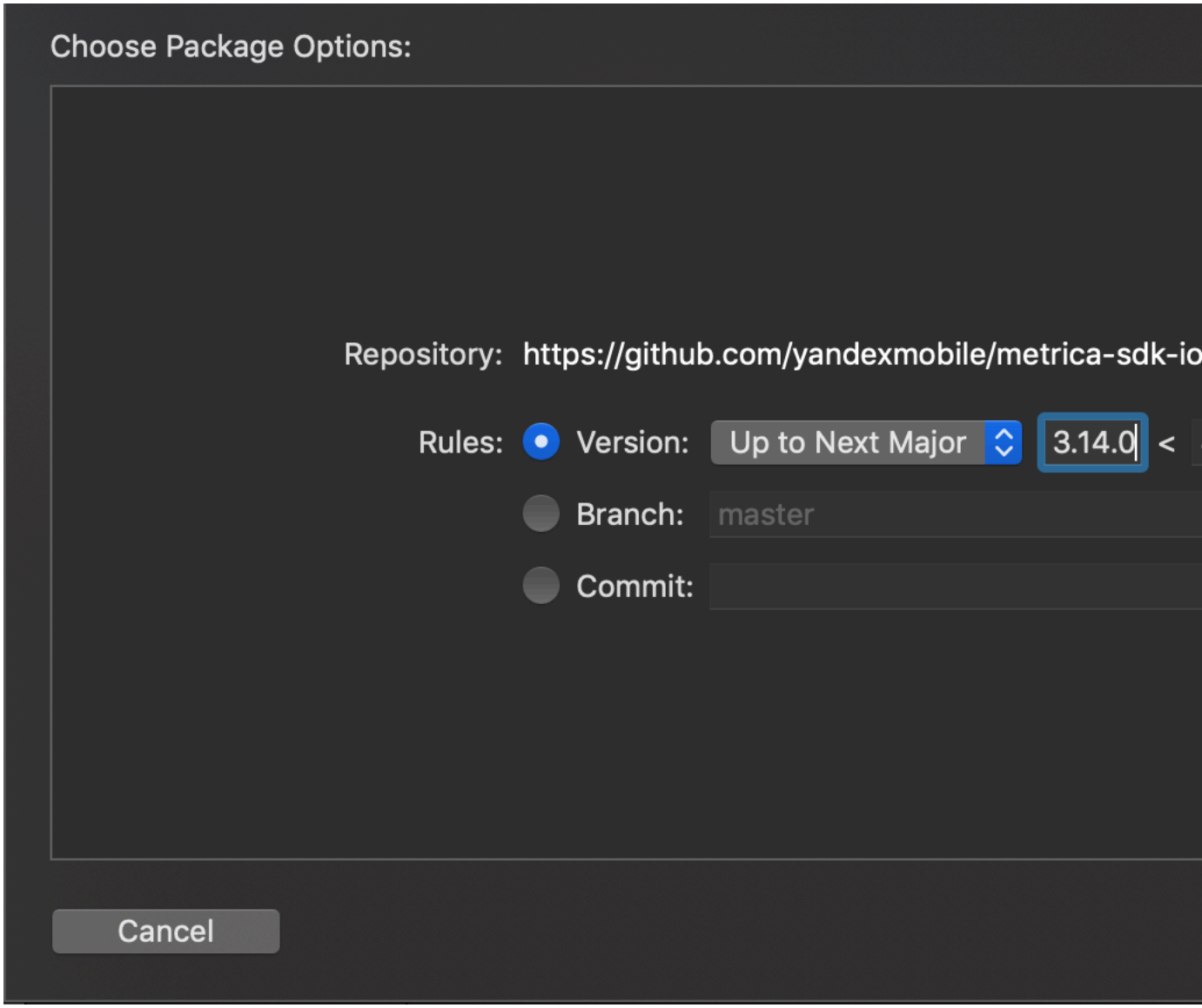
2. Specify the repository URL `https://github.com/yandexmobile/metrica-sdk-ios`, which contains a Swift package.



3. Configure a rule for selecting the package version.

   Restriction:

   Connection using Swift Package Manager is supported starting from version 3.14.0 of the AppMetrica SDK.



4. Select the required libraries.

If you don't use these dependency managers

To enable the library, follow these steps:

1. Download the AppMetrica library.
2. Add `YandexMobileMetrica.framework` to the project.
3. *(Optional)* To enable crash handling, add `YandexMobileMetricaCrashes.framework`.
4. Add the following dependencies: 'SystemConfiguration', 'UIKit', 'Foundation', 'CoreTelephony', 'CoreLocation', 'CoreGraphics', 'AdSupport', 'z', 'sqlite3', 'Security', 'c++', 'WebKit', and 'SafariServices' (with the **Optional** setting).
5. Add `-ObjC` to `Other Linker Flags`.

Step 2. Initialize the library

Objective-C

Add the import:

```
#import <YandexMobileMetrica/YandexMobileMetrica.h>
```

Initialize the library in the `application:didFinishLaunchingWithOptions:` method of your `UIApplicationDelegate`:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Initializing the AppMetrica SDK.
    YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
    [YMMYandexMetrica activateWithConfiguration:configuration];
}
```

Swift

Add the import:

```
import YandexMobileMetrica
```

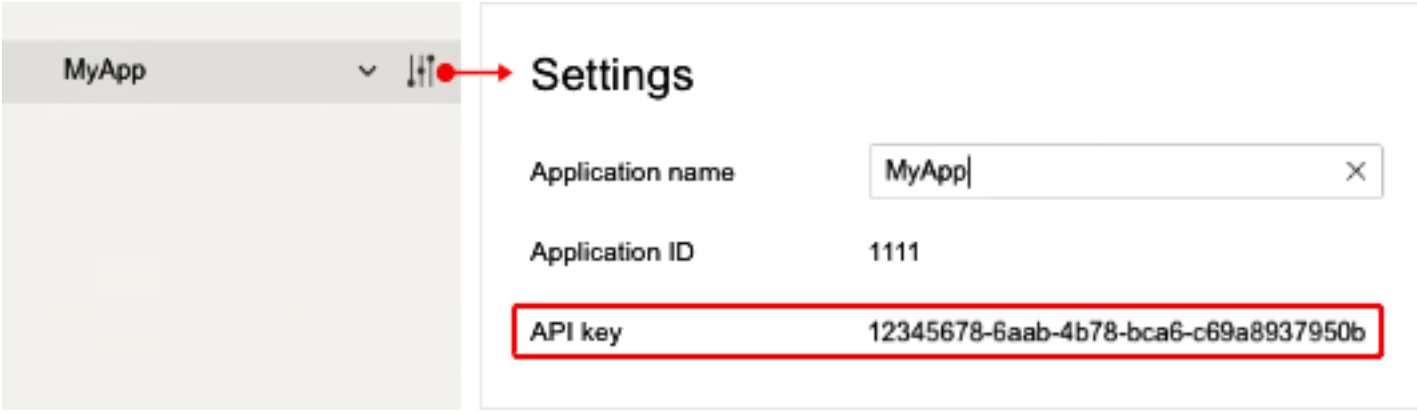
Initialize the library in the `application(_:didFinishLaunchingWithOptions:)` method of your `UIApplicationDelegate`:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
 [UIApplicationLaunchOptionsKey : Any]? = nil) -> Bool
{
    // Initializing the AppMetrica SDK.
    let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API_key")
    YMMYandexMetrica.activate(with: configuration!)
}
```

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. For more information, see Tracking pre-installed apps.

Note: Requirements: deployment target 8.0 and higher.

Step 3. *(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 4. Test the library operation

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.

Problems with Swift Package Manager

Read the article Problems when using Swift Package Manager.

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 an Apple IDFA

   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 Apple IDFA is shown in the **IDFA** 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

My app didn't pass App Store moderation

How to enable user location sending

Related information

Example of library integration

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

```
<script type="text/javascript">
    var prefix;
    var active = "_active_yes";
    if (window.location.origin.includes("appmetrica.yandex")){
        prefix = "doc-c-";
    } else {
        prefix = "";
    };
    var pane = prefix + "tabs-panes__pane";
    var paneActive = pane + active;
    var tab = prefix + "tabs-menu__tab";
    var tabActive = tab + active;

    $(document).ready(function(){
        $( "li." + tab + " > span:contains('Swift'), li." + tab + " > span:contains('Objective-
C')" ).click(function() {
            var scrollBefore = window.scrollY;
            var positionBefore = $( this ).offset().top;
```

```
            if ($( this ).html() == "Swift") {
                $( "li." + tab + " > span:contains('Swift')" ).each(function(){
                    $(this).parent().addClass(tabActive);
                    var id = $(this).parent().attr("id");
                    $("div[aria-labelledby='" + id + "']").addClass(paneActive);
                });
                $( "li." + tab + " > span:contains('Objective-C')" ).each(function(){
                    $(this).parent().removeClass(tabActive);
                    var id = $(this).parent().attr("id");
                    $("div[aria-labelledby='" + id + "']").removeClass(paneActive);
                });
            } else if ($( this ).html() == "Objective-C") {
                $( "li." + tab + " > span:contains('Objective-C')" ).each(function(){
                    $(this).parent().addClass(tabActive);
                    var id = $(this).parent().attr("id");
                    $("div[aria-labelledby='" + id + "']").addClass(paneActive);
                });
                $( "li." + tab + " > span:contains('Swift')" ).each(function(){
                    $(this).parent().removeClass(tabActive);
                    var id = $(this).parent().attr("id");
                    $("div[aria-labelledby='" + id + "']").removeClass(paneActive);
                });
            };
            positionAfter = $( this ).offset().top;
            var scrollAfter = scrollBefore + positionAfter - positionBefore;
            $(window).scrollTop(scrollAfter);
        });
    });
</script>
```

Library initialization with the extended configuration

Objective-C

To initialize the library with the extended startup configuration:

1. Initialize the YMMYandexMetricaConfiguration instance.
2. Specify the configuration settings using methods of the `YMMYandexMetricaConfiguration` class. For example, enable logging or set a session timeout.
3. Pass the `YMMYandexMetricaConfiguration` instance into the method +activateWithConfiguration: of the YMMYandexMetrica class.

The parameters of the extended configuration are applied from the time of library initialization. To configure the library while the application is running, use the methods of the YMMYandexMetrica class.

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Creating an extended library configuration.
    YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
    // Setting up the configuration. For example, to enable logging.
    configuration.logs = YES;
    ...
    // Initializing the AppMetrica SDK.
    [YMMYandexMetrica activateWithConfiguration:configuration];
}
```

Swift

To initialize the library with the extended startup configuration:

1. Initialize the YMMYandexMetricaConfiguration instance.
2. Specify the configuration settings using methods of the `YMMYandexMetricaConfiguration` class. For example, enable logging or set a session timeout.
3. Pass the `YMMYandexMetricaConfiguration` object to the activate(with:) method of the YMMYandexMetrica class.

The parameters of the extended configuration are applied from the time of library initialization. To configure the library while the application is running, use the methods of the YMMYandexMetrica class.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
 [UIApplication.LaunchOptionsKey : AnyObject]? = nil) -> Bool {
    // Creating an extended library configuration.
    let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
    // Setting up the configuration. For example, to enable logging.
    configuration?.logs = true
    ...
    // Initializing the AppMetrica SDK.
    YMMYandexMetrica.activate(with: configuration!)
    return true
}
```

Initializing the library for children's apps

Objective-C

If you have an app for kids, use the appForKids property of the YMMYandexMetricaConfiguration configuration. This property defines the application type as “children's” to match the rules for checking children's apps.

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Creating an extended library configuration.
    YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
    // Setting up the configuration.
    configuration.appForKids = YES;
    ...
    // Initializing the AppMetrica SDK.
    [YMMYandexMetrica activateWithConfiguration:configuration];
}
```

Swift

If you have an app for kids, use the appForKids property of the YMMYandexMetricaConfiguration configuration. This property defines the application type as “children's” to match the rules for checking children's apps.

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
 [UIApplication.LaunchOptionsKey : AnyObject]? = nil) -> Bool {
    // Creating an extended library configuration.
    let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
    // Setting up the configuration.
    configuration?.appForKids = true
    ...
    // Initializing the AppMetrica SDK.
    YMMYandexMetrica.activate(with: configuration!)
    return true
}
```

Note: If this option is enabled, the AppMetrica SDK doesn't send advertising IDs or location information.

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

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.

Objective-C

To initialize a reporter with an extended configuration:

1. Initialize the YMMReporterConfiguration instance.
2. Specify the configuration settings using methods of the `YMMReporterConfiguration` class. For example, enable logging or set a session timeout.
3. Pass the `YMMReporterConfiguration` instance into the method +activateReporterWithConfiguration: of the YMMYandexMetrica class.

This configuration is used for a reporter with the specified API key. You can set up your own configuration for each additional API key.

```
// Creating an extended library configuration.
// To create it, pass an API_key that is different from the app's API_key.
YMMReporterConfiguration *reporterConfiguration = [[YMMReporterConfiguration alloc] initWithApiKey:@"API_key"];
// Setting up the configuration. For example, to enable logging.
reporterConfiguration.logs = YES;
...
// Initializing a reporter.
[YMMYandexMetrica activateReporterWithConfiguration:[reporterConfiguration copy]];
```

Swift

To initialize a reporter with an extended configuration:

1. Initialize the YMMReporterConfiguration instance.
2. Specify the configuration settings using methods of the `YMMReporterConfiguration` class. For example, enable logging or set a session timeout.
3. Pass the `YMMReporterConfiguration` object to the activateReporter(with:) method of the YMMYandexMetrica class.

This configuration is used for a reporter with the specified API key. You can set up your own configuration for each additional API key.

```
// Creating an extended library configuration.
// To create it, pass an API_key that is different from the app's API_key.
let reporterConfiguration = YMMReporterConfiguration.init(apiKey: "API key")
// Setting up the configuration. For example, to enable logging.
reporterConfiguration?.logs = true
...
// Initializing a reporter.
YMMYandexMetrica.activateReporter(with: reporterConfiguration!)
```

Step 2. Configure sending data using a reporter

Objective-C

To send statistics data to another API key:

1. Use the -reporterForApiKey: method of the YMMYandexMetrica class to get the object that implements the YMMYandexMetricaReporting protocol.

   If the reporter was not initialized with the extended configuration, calling this method will initialize the reporter for the specified API key.
2. Use methods of the `YMMYandexMetricaReporting` protocol to send errors, events, and revenue.
3. To ensure that sessions are tracked correctly, set up sending session start and pause events for each reporter.

```
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"API_key"];
[reporter resumeSession];
...
[reporter reportEvent:@"Updates installed" onFailure:^(NSError *error) {
    NSLog(@"REPORT ERROR: %@", [error localizedDescription]);
}];
...
[reporter pauseSession];
```

Swift

To send statistics data to another API key:

1. Use the reporterForApiKey(_:) method of the YMMYandexMetrica class to get the object that implements the YMMYandexMetricaReporting protocol.

   If the reporter was not initialized with the extended configuration, calling this method will initialize the reporter for the specified API key.
2. Use methods of the `YMMYandexMetricaReporting` protocol to send errors, events, and revenue.
3. To ensure that sessions are tracked correctly, set up sending session start and pause events for each reporter.

```
let reporter = YMMYandexMetrica.reporterForApiKey("API_key")
reporter.resumeSession()
...
reporter.reportEvent("Updates installed", onFailure: { (error) in
    print("REPORT ERROR: %@", error?.localizedDescription)
})
...
reporter.pauseSession()
```

Tracking app crashes

Objective-C

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, set the NO value for the crashReporting property of the extended library configuration YMMYandexMetricaConfiguration.

```
// Creating an extended library configuration.
```

```
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
// Disabling sending the information on crashes of the application.
configuration.crashReporting = NO;
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

Swift

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, set the NO value for the crashReporting property of the extended library configuration YMMYandexMetricaConfiguration.

```
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API_key")
// Disabling sending the information on crashes of the application.
configuration?.crashReporting = false
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

Determining the location

AppMetrica can determine the location of a device. Location accuracy depends on the configuration that the library uses for initialization:

With the locationTracking option enabled

Note:

For iOS, the option is enabled by default.

The location is determined with accuracy to the city. You can retrieve this information in reports and via the Logs API.

The app requests GPS access. Battery consumption may increase.

With the locationTracking option disabled

Note:

Starting with the Android AppMetrica SDK 5.0.0, the `locationTracking` option is disabled by default.

If the version is lower than 5.0.0, the `locationTracking` option is enabled by default.

The location is determined by the IP address with accuracy to the country. You can retrieve this information in reports, but not via the Logs API.

The app requests GPS access. Battery consumption does not increase.

Note: If you have enabled IP address masking, AppMetrica determines location with the accuracy to the country by the unmasked part of the IP address.

Objective-C

By default, the AppMetrica SDK is initialized with locationTracking enabled, but it doesn't request permission to get location data. You should implement this using the methods of the CLLocationManager class.

To initialize a library with the disabled locationTracking option, set the NO value for the locationTracking property of the YMMYandexMetricaConfiguration configuration.

```
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc] initWithApiKey:API_key];
// Disabling sending information about the device location.
configuration.locationTracking = NO;
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

To disable locationTracking after initializing the library, use the method +setLocationTracking: of the YMMYandexMetrica class:

```
[YMMYandexMetrica setTrackLocationEnabled:NO];
```

Swift

By default, the AppMetrica SDK is initialized with locationTracking enabled, but it doesn't request permission to get location data. You should implement this using the methods of the CLLocationManager class.

To initialize a library with locationTracking disabled, set the locationTracking property of the YMMYandexMetricaConfiguration configuration to `false`.

```
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
// Disabling sending  information about the location of the device.
configuration?.locationTracking = false
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

To disable locationTracking after initializing the library, use the method setLocationTracking(_:) of the YMMYandexMetrica class:

```
YMMYandexMetrica.setLocationTracking(false)
```

Setting device location manually

Objective-C

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

By default, the device location is detected by the library.

To send custom device location information, pass the CLLocation instance into the method +setLocation: of the YMMYandexMetrica class.

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation
{
    [YMMYandexMetrica setLocation:newLocation];
}
```

To send your own device location information using the startup configuration, pass the CLLocation instance to the location property when creating the extended library configuration YMMYandexMetricaConfiguration.

Swift

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

By default, the device location is detected by the library.

To send custom device location information, pass the CLLocation instance into the method setLocation(_:) of the YMMYandexMetrica class.

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
    YMMYandexMetrica.setLocation(locations.last)
}
```

To send your own device location information using the startup configuration, pass the CLLocation instance to the location property when creating the extended library configuration YMMYandexMetricaConfiguration.

Sending a custom event

Objective-C

To send a custom event without nested parameters, pass in the method +reportEvent:onFailure: of the YMMYandexMetrica class the following parameter:

- `message` — Short name or description of the event;
- `onFailure` — The block the error is passed to. If you do not want to track the error, pass `nil` for this block.

```
[YMMYandexMetrica reportEvent:@"Updates installed" onFailure:^(NSError *error) {
    NSLog(@"DID FAIL REPORT EVENT: %@", message);
    NSLog(@"REPORT ERROR: %@", [error localizedDescription]);
}];
```

Swift

To send a custom event without nested parameters, pass into the method reportEvent(_:onFailure:) of the YMMYandexMetrica class the following parameter:

- `message` — Short name or description of the event;
- `onFailure` — The block the error is passed to. If you do not want to track the error, pass `nil` for this block.

```
YMMYandexMetrica.reportEvent("Updates installed", onFailure: { (error) in
    print("DID FAIL REPORT EVENT: %@", message)
```

```
    print("REPORT ERROR: %@", error?.localizedDescription)
}
```

Sending a custom event with nested parameters

Objective-C

To send a custom event with nested parameters, pass into the method +reportEvent:params:onFailure: of the YMMYandexMetrica class the following parameter:

- `message` — Short name or description of the event;
- `params` — Nested parameters as “key-value” pairs;

  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.
- `onFailure` — The block the error is passed to. If you do not want to track the error, pass `nil` for this block.

```
NSDictionary *params = @{@"key1": @"value1", @"key2": @"value2"};
[YMMYandexMetrica reportEvent:@"EVENT"
                   parameters:params
                    onFailure:^(NSError *error) {
                        NSLog(@"error: %@", [error localizedDescription]);
                    }];
```

Swift

To send a custom event with nested parameters, pass the following parameters to the reportEvent(_:parameters:onFailure:) method of the YMMYandexMetrica class:

- `message` — Short name or description of the event;
- `params` — Nested parameters as “key-value” pairs;

  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.
- `onFailure` — The block the error is passed to. If you do not want to track the error, pass `nil` for this block.

```
let params : [AnyHashable : Any] = ["key1": "value1", "key2": "value2"]
YMMYandexMetrica.reportEvent("EVENT", parameters: params, onFailure: { (error) in
    print("DID FAIL REPORT EVENT: %@", message)
    print("REPORT ERROR: %@", error?.localizedDescription)
})
```

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(Objective-C, Swift):

Objective-C

```
WKWebViewConfiguration *configuration = [[WKWebViewConfiguration alloc] init];
WKUserContentController *userController = [[WKUserContentController alloc] init];
[YMMYandexMetrica initWebViewReporting:userController onFailure:nil];
[userController addScriptMessageHandler:myHandler name:myScriptName];
configuration.userContentController = userController;
WKWebView *webView = [[WKWebView alloc] initWithFrame:CGRectZero configuration:configuration];
```

Swift

```
let configuration = WKWebViewConfiguration()
let userController = WKUserContentController()
YMMYandexMetrica.initWebViewReporting(userController, onFailure: nil)
userController.add(myHandler, name: myScriptName)
configuration.userContentController = userController;
let webView = WKWebView(frame: .zero, configuration: configuration)
```

Call this method before loading any content. We recommend calling this method before creating a webview and before adding your scripts to WKUserContentController.

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 YMMYandexMetrica class and YMMYandexMetricaReporting protocol methods:

- +reportError:onFailure:
- +reportError:options:onFailure:
- +reportNSError:onFailure:
- +reportNSError:options:onFailure:

Note: These methods are supported since AppMetrica SDK version 3.11.1.

To send error messages, you can use the standard NSError class, the simplified YMMError class, or the YMMErrorRepresentable protocol.

Example with NSError

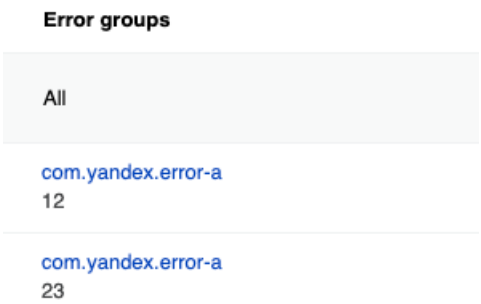
If errors are sent using the NSError class, they're grouped by the domain domain and the code error code.

Objective-C

```
NSError *firstError = [NSError errorWithDomain:@"com.yandex.error-a"
                                          code:12
                                      userInfo:@{
                                          YMMBacktraceErrorKey: NSThread.callStackReturnAddresses,
                                          NSLocalizedDescriptionKey: @"Error A"
                                      }];
[YMMYandexMetrica reportNSError:firstError onFailure:nil];
```

Swift

```
let firstError = NSError(domain: "com.yandex.error-a", code: 12, userInfo: [
    YMMBacktraceErrorKey: Thread.callStackReturnAddresses,
    NSLocalizedDescriptionKey: "Error A"
])
YMMYandexMetrica.report(nserror: firstError, onFailure: nil)
```



1. domain
2. code

Example with YMMError

If errors are sent using the YMMError class or the YMMErrorRepresentable protocol, they're grouped by the identifier ID.

Objective-C

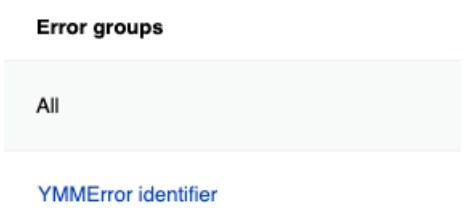
```
YMMError *underlyingError = [YMMError errorWithIdentifier:@"Underlying YMMError"];
YMMError *error = [YMMError errorWithIdentifier:@"YMMError identifier"
                                        message:@"Another custom message"
                                     parameters:@{ @"foo": @"bar" }
                                      backtrace:NSThread.callStackReturnAddresses
                                underlyingError:underlyingError];
```

```
[YMMYandexMetrica reportError:error onFailure:nil];
```

Swift

```
let underlyingError = YMMError.init(identifier: "Underlying YMMError")
let error = YMMError(
    identifier: "YMMError identifier",
    message: "Another custom message",
    parameters: [
        "foo": "bar"
    ],
    backtrace: Thread.callStackReturnAddresses,
    underlyingError: underlyingError)
YMMYandexMetrica.report(error: error, onFailure: nil)
```

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



1. identifier

Sending profile attributes

Objective-C

To send profile attributes, pass in the method +reportUserProfile:onFailure: of the YMMYandexMetrica class the following parameters:

- `userProfile` — The YMMUserProfile instance that contains an array of attribute updates. To create profile attributes, use methods of the YMMProfileAttribute class.
- `onFailure` — The block the error is passed to. If you do not want to track the error, pass `nil` for this block.

```
YMMMutableUserProfile *profile = [[YMMMutableUserProfile alloc] init];
// Updating a single user profile attribute.
id<YMMCustomCounterAttribute> timeLeftAttribute = [YMMProfileAttribute customCounter:@"time_left"];
[profile apply:[timeLeftAttribute withDelta:-4.42]];
// Updating multiple attributes.
[profile applyFromArray:@[
    // Updating predefined attributes.
    [[YMMProfileAttribute name] withValue:@"John"],
    [[YMMProfileAttribute gender] withValue:YMMGenderTypeMale],
    [[YMMProfileAttribute birthDate] withAge:24],
    [[YMMProfileAttribute notificationsEnabled] withValue:NO],
    // Updating custom attributes.
    [[YMMProfileAttribute customString:@"born_in"] withValueIfUndefined:@"Moscow"],
    [[YMMProfileAttribute customString:@"address"] withValueReset],
    [[YMMProfileAttribute customNumber:@"age"] withValue:24],
    [[YMMProfileAttribute customCounter:@"logins_count"] withDelta:1],
    [[YMMProfileAttribute customBool:@"has_premium"] withValue:YES]
]];
// ProfieID is set using the method of the YMMYandexMetrica class.
[YMMYandexMetrica >setUserProfileID:@"id"];

// Sending profile attributes.
[YMMYandexMetrica reportUserProfile:[profile copy] onFailure:^(NSError *error) {
    NSLog(@"Error: %@", error);
}];
```

Swift

To send profile attributes, pass in the method reportUserProfile(_:onFailure:) of the YMMYandexMetrica class the following parameters:

- `userProfile` — The YMMUserProfile instance that contains an array of attribute updates. To create profile attributes, use methods of the YMMProfileAttribute class.
- `onFailure` — The block the error is passed to. If you do not want to track the error, pass `nil` for this block.

```
let profile = YMMMutableUserProfile()
// Updating a single user profile attribute.
let timeLeftAttribute: YMMCustomCounterAttribute = YMMProfileAttribute.customCounter("time_left")
profile.apply(timeLeftAttribute.withDelta(-4.42))
// Updating multiple attributes.
```

```
profile.apply(from: [
    // Updating predefined attributes.
    YMMProfileAttribute.name().withValue("John"),
    YMMProfileAttribute.gender().withValue(YMMGenderType.male),
    YMMProfileAttribute.birthDate().withAge(24),
    YMMProfileAttribute.notificationsEnabled().withValue(false),
    // Updating custom attributes.
    YMMProfileAttribute.customString("born_in").withValueIfUndefined("Moscow"),
    YMMProfileAttribute.customString("address").withValueReset(),
    YMMProfileAttribute.customNumber("age").withValue(24),
    YMMProfileAttribute.customCounter("logins_count").withDelta(1),
    YMMProfileAttribute.customBool("has_premium").withValue(true)
])
// ProfieID is set using the method of the YMMYandexMetrica class.
YMMYandexMetrica.setUserProfileID("id")

// Sending profile attributes.
YMMYandexMetrica.report(profile, onFailure: { (error) in
    print("REPORT ERROR: %@", error.localizedDescription)
})
```

Sending ProfileId

Objective-C

To send ProfileId, use the method +setUserProfileID: of the YMMYandexMetrica class.

If you don't configure ProfileId sending in the SDK, the web interface displays the `appmetrica_device_id` value.

```
[YMMYandexMetrica setUserProfileID:@"id"];
```

Swift

To send a ProfileId, use the method setUserProfileID(_:) of the YMMYandexMetrica class.

If you don't configure ProfileId sending in the SDK, the web interface displays the `appmetrica_device_id` value.

```
YMMYandexMetrica.setUserProfileID("id")
```

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

Objective-C

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

The examples below show how to send specific types of events (Objective-C):

Opening a page

```
// Creating a screen object.
YMMECommerceScreen *screen = [[YMMECommerceScreen alloc] initWithName:@"ProductCardScreen"
                                                    categoryComponents:@[ @"Акции", @"Красная цена" ]
                                                           searchQuery:@"даниссимо кленовый сироп"
                                                               payload:@{ @"full_screen": @"true" }];
// Sending an e-commerce event.
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
[reporter reportECommerce:[YMMECommerce showScreenEventWithScreen:screen] onFailure:nil];
```

Viewing a product profile

```
// Creating a screen object.
YMMECommerceScreen *screen = [[YMMECommerceScreen alloc] initWithName:@"ProductCardScreen"
                                                    categoryComponents:@[ @"Акции", @"Красная цена" ]
                                                           searchQuery:@"даниссимо кленовый сироп"
                                                               payload:@{ @"full_screen": @"true" }];
// Creating an actualPrice object.
YMMECommerceAmount *actualFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"4.53"]];
YMMECommerceAmount *woodActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30570000"]];
YMMECommerceAmount *ironActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.89"]];
YMMECommerceAmount *goldActualPrice =
```

```
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.1"]];
YMMECommercePrice *actualPrice = [[YMMECommercePrice alloc] initWithFiat:actualFiat
                                                      internalComponents:@[ woodActualPrice, ironActualPrice,
 goldActualPrice ]];
// Creating an originalPrice object.
YMMECommerceAmount *originalFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"5.78"]];
YMMECommerceAmount *woodOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30590000"]];
YMMECommerceAmount *ironOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.92"]];
YMMECommerceAmount *goldOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.5"]];
YMMECommercePrice *originalPrice = [[YMMECommercePrice alloc] initWithFiat:originalFiat
                                                        internalComponents:@[ woodOriginalPrice,
 ironOriginalPrice, goldOriginalPrice ]];
// Creating a product object.
YMMECommerceProduct *product = [[YMMECommerceProduct alloc] initWithSKU:@"779213"
                                                                   name:@"Продукт творожный «Даниссимо» 5.9%,
 130 г."
                                                     categoryComponents:@[ @"Продукты", @"Молочные продукты",
 @"Йогурты" ]
                                                                payload:@{ @"full_screen" : @"true" }
                                                           actualPrice:actualPrice
                                                         originalPrice:originalPrice
                                                            promoCodes:@[ @"BT79IYX", @"UT5412EP" ]];
// Sending an e-commerce event.
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
[reporter reportECommerce:[YMMECommerce showProductCardEventWithProduct:product screen:screen]
 onFailure:nil];
```

Viewing a product page

```
// Creating a screen object.
YMMECommerceScreen *screen = [[YMMECommerceScreen alloc] initWithName:@"ProductCardScreen"
                                                    categoryComponents:@[ @"Акции", @"Красная цена" ]
                                                           searchQuery:@"даниссимо кленовый сироп"
                                                               payload:@{ @"full_screen": @"true" }];

// Creating an actualPrice object.
YMMECommerceAmount *actualFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"4.53"]];
YMMECommerceAmount *woodActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30570000"]];
YMMECommerceAmount *ironActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.89"]];
YMMECommerceAmount *goldActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.1"]];
YMMECommercePrice *actualPrice = [[YMMECommercePrice alloc] initWithFiat:actualFiat
                                                      internalComponents:@[ woodActualPrice, ironActualPrice,
 goldActualPrice ]];
// Creating an originalPrice object.
YMMECommerceAmount *originalFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"5.78"]];
YMMECommerceAmount *woodOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30590000"]];
YMMECommerceAmount *ironOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.92"]];
YMMECommerceAmount *goldOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.5"]];
YMMECommercePrice *originalPrice = [[YMMECommercePrice alloc] initWithFiat:originalFiat
                                                        internalComponents:@[ woodOriginalPrice,
 ironOriginalPrice, goldOriginalPrice ]];
// Creating a product object.
YMMECommerceProduct *product = [[YMMECommerceProduct alloc] initWithSKU:@"779213"
                                                                   name:@"Продукт творожный «Даниссимо» 5.9%,
 130 г."
                                                     categoryComponents:@[ @"Продукты", @"Молочные продукты",
 @"Йогурты" ]
                                                                payload:@{ @"full_screen" : @"true" }
                                                           actualPrice:actualPrice
                                                         originalPrice:originalPrice
                                                            promoCodes:@[ @"BT79IYX", @"UT5412EP" ]];
// Creating a referrer object.
YMMECommerceReferrer *referrer = [[YMMECommerceReferrer alloc] initWithType:@"button"
                                                                 identifier:@"76890"
                                                                     screen:screen];
// Sending an e-commerce event.
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
```

```
[reporter reportECommerce:[YMMECommerce showProductDetailsEventWithProduct:product referrer:referrer]
 onFailure:nil];
```

Adding or removing an item to/from the cart

```
// Creating a screen object.
YMMECommerceScreen *screen = [[YMMECommerceScreen alloc] initWithName:@"ProductCardScreen"
                                                           categoryComponents:@[ @"Акции", @"Красная цена" ]
                                                                  searchQuery:@"даниссимо кленовый сироп"
                                                                      payload:@{ @"full_screen": @"true" }];
// Creating an actualPrice object.
YMMECommerceAmount *actualFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"4.53"]];
YMMECommerceAmount *woodActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30570000"]];
YMMECommerceAmount *ironActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.89"]];
YMMECommerceAmount *goldActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.1"]];
YMMECommercePrice *actualPrice = [[YMMECommercePrice alloc] initWithFiat:actualFiat
                                                       internalComponents:@[ woodActualPrice, ironActualPrice,
 goldActualPrice ]];
// Creating an originalPrice object.
YMMECommerceAmount *originalFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"5.78"]];
YMMECommerceAmount *woodOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30590000"]];
YMMECommerceAmount *ironOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.92"]];
YMMECommerceAmount *goldOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.5"]];
YMMECommercePrice *originalPrice = [[YMMECommercePrice alloc] initWithFiat:originalFiat
                                                         internalComponents:@[ woodOriginalPrice,
 ironOriginalPrice, goldOriginalPrice ]];
// Creating a product object.
YMMECommerceProduct *product = [[YMMECommerceProduct alloc] initWithSKU:@"779213"
                                                                   name:@"Продукт творожный «Даниссимо» 5.9%,
 130 г."
                                                     categoryComponents:@[ @"Продукты", @"Молочные продукты",
 @"Йогурты" ]
                                                                payload:@{ @"full_screen" : @"true" }
                                                            actualPrice:actualPrice
                                                         originalPrice:originalPrice
                                                             promoCodes:@[ @"BT79IYX", @"UT5412EP" ]];
// Creating a referrer object.
YMMECommerceReferrer *referrer = [[YMMECommerceReferrer alloc] initWithType:@"button"
                                                                 identifier:@"76890"
                                                                     screen:screen];
// Creating a cartItem object.
NSDecimalNumber *quantity = [NSDecimalNumber decimalNumberWithString:@"1"];
YMMECommerceCartItem *addedItems = [[YMMECommerceCartItem alloc] initWithProduct:product
                                                                        referrer:referrer
                                                                        quantity:quantity
                                                                         revenue:actualPrice];
// Sending an e-commerce event.
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
[reporter reportECommerce:[YMMECommerce addCartItemEvent:addedItems] onFailure:nil];
// Or:
[YMMYandexMetrica reportECommerce:[YMMECommerce removeCartItemEventWithItem:addedItems] onFailure:nil];
```

Starting and completing a purchase

```
// Creating a screen object.
YMMECommerceScreen *screen = [[YMMECommerceScreen alloc] initWithName:@"ProductCardScreen"
                                                           categoryComponents:@[ @"Акции", @"Красная цена" ]
                                                                  searchQuery:@"даниссимо кленовый сироп"
                                                                      payload:@{ @"full_screen": @"true" }];
// Creating an actualPrice object.
YMMECommerceAmount *actualFiat =
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"4.53"]];
YMMECommerceAmount *woodActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30570000"]];
YMMECommerceAmount *ironActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.89"]];
YMMECommerceAmount *goldActualPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.1"]];
YMMECommercePrice *actualPrice = [[YMMECommercePrice alloc] initWithFiat:actualFiat
                                                       internalComponents:@[ woodActualPrice, ironActualPrice,
 goldActualPrice ]];
// Creating an originalPrice object.
YMMECommerceAmount *originalFiat =
```

```
        [[YMMECommerceAmount alloc] initWithUnit:@"USD" value:[NSDecimalNumber
 decimalNumberWithString:@"5.78"]];
YMMECommerceAmount *woodOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"wood" value:[NSDecimalNumber
 decimalNumberWithString:@"30590000"]];
YMMECommerceAmount *ironOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"iron" value:[NSDecimalNumber
 decimalNumberWithString:@"26.92"]];
YMMECommerceAmount *goldOriginalPrice =
        [[YMMECommerceAmount alloc] initWithUnit:@"gold" value:[NSDecimalNumber
 decimalNumberWithString:@"5.5"]];
YMMECommercePrice *originalPrice = [[YMMECommercePrice alloc] initWithFiat:originalFiat
                                                        internalComponents:@[ woodOriginalPrice,
 ironOriginalPrice, goldOriginalPrice ]];
// Creating a product object.
YMMECommerceProduct *product = [[YMMECommerceProduct alloc] initWithSKU:@"779213"
                                                                    name:@"Продукт творожный «Даниссимо» 5.9%,
 130 г."
                                                      categoryComponents:@[ @"Продукты", @"Молочные продукты",
 @"Йогурты" ]
                                                                 payload:@{ @"full_screen" : @"true" }
                                                           actualPrice:actualPrice
                                                         originalPrice:originalPrice
                                                            promoCodes:@[ @"BT79IYX", @"UT5412EP" ]];
// Creating a referrer object.
YMMECommerceReferrer *referrer = [[YMMECommerceReferrer alloc] initWithType:@"button"
                                                                 identifier:@"76890"
                                                                     screen:screen];
// Creating a cartItem object.
NSDecimalNumber *quantity = [NSDecimalNumber decimalNumberWithString:@"1"];
YMMECommerceCartItem *addedItems = [[YMMECommerceCartItem alloc] initWithProduct:product
                                                                        referrer:referrer
                                                                        quantity:quantity
                                                                         revenue:actualPrice];
// Creating an order object.
YMMECommerceOrder *order = [[YMMECommerceOrder alloc] initWithIdentifier:@"88528768"
                                                               cartItems:@[ addedItems ]
                                                                 payload:@{ @"black_friday" : @"true"}];
// Sending an e-commerce event.
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
[reporter reportECommerce:[YMMECommerce beginCheckoutEventWithOrder:order] onFailure:nil];
[YMMYandexMetrica reportECommerce:[YMMECommerce purchaseEventWithOrder:order] onFailure:nil];
```

Swift

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

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

Opening a page

```
// Creating a screen object.
let screen = YMMECommerceScreen(
        name: "ProductCardScreen",
        categoryComponents: ["Акции", "Красная цена"],
        searchQuery: "даниссимо кленовый сироп",
        payload: ["full_screen": "true"]
)
// Sending an e-commerce event.
YMMYandexMetrica.report(eCommerce: .showScreenEvent(screen: screen), onFailure: nil)
```

Viewing a product profile

```
// Creating a screen object.
let screen = YMMECommerceScreen(
        name: "ProductCardScreen",
        categoryComponents: ["Акции", "Красная цена"],
        searchQuery: "даниссимо кленовый сироп",
        payload: ["full_screen": "true"]
)
// Creating an actualPrice object.
let actualPrice = YMMECommercePrice(
        fiat: .init(unit: "USD", value: .init(string: "4.53")),
        internalComponents: [
            .init(unit: "wood", value: .init(string: "30570000")),
            .init(unit: "iron", value: .init(string: "26.89")),
            .init(unit: "gold", value: .init(string: "5.1")),
        ]
)
// Creating a product object.
let product = YMMECommerceProduct(
        sku: "779213",
        name: "Продукт творожный «Даниссимо» 5.9%, 130 г.",
        categoryComponents: ["Продукты", "Молочные продукты", "Йогурты"],
        payload: ["full_screen": "true"],
        actualPrice: actualPrice,
        originalPrice: .init(
                fiat: .init(unit: "USD", value: .init(string: "5.78")),
                internalComponents: [
                    .init(unit: "wood", value: .init(string: "30590000")),
                    .init(unit: "iron", value: .init(string: "26.92")),
```

```
                    .init(unit: "gold", value: .init(string: "5.5")),
                ]
        ),
        promoCodes: ["BT79IYX", "UT5412EP"]
)
// Sending an e-commerce event.
YMMYandexMetrica.report(eCommerce: .showProductCardEvent(product: product, screen: screen), onFailure: nil)
```

Viewing a product page

```
// Creating a screen object.
let screen = YMMECommerceScreen(
        name: "ProductCardScreen",
        categoryComponents: ["Акции", "Красная цена"],
        searchQuery: "даниссимо кленовый сироп",
        payload: ["full_screen": "true"]
)
// Creating an actualPrice object.
let actualPrice = YMMECommercePrice(
        fiat: .init(unit: "USD", value: .init(string: "4.53")),
        internalComponents: [
            .init(unit: "wood", value: .init(string: "30570000")),
            .init(unit: "iron", value: .init(string: "26.89")),
            .init(unit: "gold", value: .init(string: "5.1")),
        ]
)
// Creating a product object.
let product = YMMECommerceProduct(
        sku: "779213",
        name: "Продукт творожный «Даниссимо» 5.9%, 130 г.",
        categoryComponents: ["Продукты", "Молочные продукты", "Йогурты"],
        payload: ["full_screen": "true"],
        actualPrice: actualPrice,
        originalPrice: .init(
                fiat: .init(unit: "USD", value: .init(string: "5.78")),
                internalComponents: [
                    .init(unit: "wood", value: .init(string: "30590000")),
                    .init(unit: "iron", value: .init(string: "26.92")),
                    .init(unit: "gold", value: .init(string: "5.5")),
                ]
        ),
        promoCodes: ["BT79IYX", "UT5412EP"]
)
// Creating a referrer object.
let referrer = YMMECommerceReferrer(type: "button", identifier: "76890", screen: screen)
// Sending an e-commerce event.
YMMYandexMetrica.report(eCommerce: .showProductDetailsEvent(product: product, referrer: referrer), onFailure:
 nil)
```

Adding or removing an item to/from the cart

```
// Creating a screen object.
let screen = YMMECommerceScreen(
        name: "ProductCardScreen",
        categoryComponents: ["Акции", "Красная цена"],
        searchQuery: "даниссимо кленовый сироп",
        payload: ["full_screen": "true"]
)
// Creating an actualPrice object.
let actualPrice = YMMECommercePrice(
        fiat: .init(unit: "USD", value: .init(string: "4.53")),
        internalComponents: [
            .init(unit: "wood", value: .init(string: "30570000")),
            .init(unit: "iron", value: .init(string: "26.89")),
            .init(unit: "gold", value: .init(string: "5.1")),
        ]
)
// Creating a product object.
let product = YMMECommerceProduct(
        sku: "779213",
        name: "Продукт творожный «Даниссимо» 5.9%, 130 г.",
        categoryComponents: ["Продукты", "Молочные продукты", "Йогурты"],
        payload: ["full_screen": "true"],
        actualPrice: actualPrice,
        originalPrice: .init(
                fiat: .init(unit: "USD", value: .init(string: "5.78")),
                internalComponents: [
                    .init(unit: "wood", value: .init(string: "30590000")),
                    .init(unit: "iron", value: .init(string: "26.92")),
                    .init(unit: "gold", value: .init(string: "5.5")),
                ]
        ),
        promoCodes: ["BT79IYX", "UT5412EP"]
)
// Creating a referrer object.
let referrer = YMMECommerceReferrer(type: "button", identifier: "76890", screen: screen)
// Creating a cartItem object.
let addedItems = YMMECommerceCartItem(
        product: product,
        referrer: referrer,
        quantity: .init(string: "1"),
        revenue: actualPrice
)
```

```
// Sending an e-commerce event.
YMMYandexMetrica.report(eCommerce: .addCartItemEvent(cartItem: addedItems), onFailure: nil)
// Or:
YMMYandexMetrica.report(eCommerce: .removeCartItemEvent(cartItem: addedItems), onFailure: nil)
```

Starting and completing a purchase

```
// Creating a screen object.
let screen = YMMECommerceScreen(
        name: "ProductCardScreen",
        categoryComponents: ["Акции", "Красная цена"],
        searchQuery: "даниссимо кленовый сироп",
        payload: ["full_screen": "true"]
)
// Creating an actualPrice object.
let actualPrice = YMMECommercePrice(
        fiat: .init(unit: "USD", value: .init(string: "4.53")),
        internalComponents: [
            .init(unit: "wood", value: .init(string: "30570000")),
            .init(unit: "iron", value: .init(string: "26.89")),
            .init(unit: "gold", value: .init(string: "5.1")),
        ]
)
// Creating a product object.
let product = YMMECommerceProduct(
        sku: "779213",
        name: "Продукт творожный «Даниссимо» 5.9%, 130 г.",
        categoryComponents: ["Продукты", "Молочные продукты", "Йогурты"],
        payload: ["full_screen": "true"],
        actualPrice: actualPrice,
        originalPrice: .init(
                fiat: .init(unit: "USD", value: .init(string: "5.78")),
                internalComponents: [
                    .init(unit: "wood", value: .init(string: "30590000")),
                    .init(unit: "iron", value: .init(string: "26.92")),
                    .init(unit: "gold", value: .init(string: "5.5")),
                ]
        ),
        promoCodes: ["BT79IYX", "UT5412EP"]
)
// Creating a referrer object.
let referrer = YMMECommerceReferrer(type: "button", identifier: "76890", screen: screen)
// Creating a cartItem object.
let addedItems = YMMECommerceCartItem(
        product: product,
        referrer: referrer,
        quantity: .init(string: "1"),
        revenue: actualPrice
)
// Creating an order object.
let order = YMMECommerceOrder(
        identifier: "88528768",
        cartItems: [addedItems],
        payload: ["black_friday": "true"]
)
// Sending an e-commerce event.
YMMYandexMetrica.report(eCommerce: .beginCheckoutEvent(order:order), onFailure: nil)
YMMYandexMetrica.report(eCommerce: .purchaseEvent(order: order), onFailure: nil)
```

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 YMMECommerce instance to the main API key, use the +reportECommerce:onFailure: method of the YMMYandexMetrica class.

Objective-C

```
...
// Sending an e-commerce event.
[YMMYandexMetrica reportECommerce:[YMMECommerce showScreenEventWithScreen:screen] onFailure:nil];
```

Swift

```
...
// Sending an e-commerce event.
YMMYandexMetrica.report(eCommerce: .showScreenEvent(screen: screen), onFailure: nil)
```

Sending Revenue

AppMetrica supports the revenue validation for purchases made using the StoreKit 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: To validate purchases, enable the validation in the settings. For more information, see Sending In-App purchases on iOS.

Step 1. Test sending Revenue

Objective-C

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 iOS, configure sending the transactionID and receiptData fields in the implementation of the completion of the transaction:

1. Initialize the YMMMutableRevenueInfo instance.
2. To validate a purchase, specify transactionID and receiptData. You should receive them before invoking `[[SKPaymentQueue defaultQueue] finishTransaction:transaction]`.
3. Send the `YMMMutableRevenueInfo` instance to the test API key using the YMMYandexMetricaReporting reporter. For more information on reporters, see Sending statistics to an additional API key.

```
- (void)completeTransaction:(SKPaymentTransaction *)transaction
{
    ...
    NSDecimalNumber *price = [NSDecimalNumber decimalNumberWithString:@"2100.5"];
    // Initializing the Revenue instance.
    YMMMutableRevenueInfo *revenueInfo = [[YMMMutableRevenueInfo alloc] initWithPriceDecimal:price
 currency:@"BYN"];
    revenueInfo.productID = @"TV soundbar";
    revenueInfo.quantity = 2;
    revenueInfo.payload = @{ @"source": @"AppStore" };
    // Set purchase information for validation.
    revenueInfo.transactionID = transaction.transactionIdentifier;
    revenueInfo.receiptData = [NSData dataWithContentsOfURL:[[NSBundle mainBundle] appStoreReceiptURL]];
    // Sending the Revenue instance using reporter.
    id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
    [reporter reportRevenue:[revenueInfo copy] onFailure:^(NSError *error) {
        NSLog(@"Revenue error: %@", error);
    }];
    // Remove the transaction from the payment queue.
    [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}
```

Without validation

To send information about a purchase:

1. Initialize the YMMMutableRevenueInfo instance.
2. (Optional) To group purchases by OrderID, specify it in the payload property.

   Note: If the OrderID is not specified, AppMetrica generates the ID automatically.
3. Send the `YMMMutableRevenueInfo` instance to the test API key using the YMMYandexMetricaReporting reporter. For more information on reporters, see Sending statistics to an additional API key.

```
NSDecimalNumber *price = [NSDecimalNumber decimalNumberWithString:@"2100.5"];
// Initializing the Revenue instance.
YMMMutableRevenueInfo *revenueInfo = [[YMMMutableRevenueInfo alloc] initWithPriceDecimal:price
 currency:@"BYN"];
revenueInfo.productID = @"TV soundbar";
revenueInfo.quantity = 2;
// Setting the OrderID parameter in the payload property to group purchases
revenueInfo.payload = @{ @"OrderID": @"Identifier", @"source": @"AppStore" };
// Sending the Revenue instance using reporter.
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:@"Testing API key"];
[reporter reportRevenue:[revenueInfo copy] onFailure:^(NSError *error) {
    NSLog(@"Revenue error: %@", error);
}];
```

Swift

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 iOS, configure sending the transactionID and receiptData fields in the implementation of the completion of the transaction:

1. Initialize the YMMMutableRevenueInfo instance.
2. To validate a purchase, specify transactionID and receiptData. You should receive them before invoking `[[SKPaymentQueue defaultQueue] finishTransaction:transaction]`.
3. Send the `YMMMutableRevenueInfo` instance to the test API key using the YMMYandexMetricaReporting reporter. For more information on reporters, see Sending statistics to an additional API key.

```
func completeTransaction(_ transaction: SKPaymentTransaction) {
    ...
    let price = NSDecimalNumber(string: "2100.5")
    // Initializing the Revenue instance.
    let revenueInfo = YMMMutableRevenueInfo.init(priceDecimal: price, currency: "BYN")
    revenueInfo.productID = "TV soundbar"
    revenueInfo.quantity = 2
    revenueInfo.payload = ["source": "AppStore"]
    // Set purchase information for validation.
    if let url = Bundle.main.appStoreReceiptURL, let data = try? Data(contentsOf: url), let transactionID =
 transaction.transactionIdentifier {
        revenueInfo.transactionID = transactionID
        revenueInfo.receiptData = data
    }
    // Sending the Revenue instance using reporter.
    let reporter = YMMYandexMetrica.reporterForApiKey("API_key")
    reporter.reportRevenue(revenueInfo, onFailure: { (error) in
        print("REPORT ERROR: \(error.localizedDescription)")
    })
    // Remove the transaction from the payment queue.
    SKPaymentQueue.default().finishTransaction(transaction)
}
```

Without validation

To send information about a purchase:

1. Initialize the YMMMutableRevenueInfo instance.
2. (Optional) To group purchases by OrderID, specify it in the payload property.

   Note: If the OrderID is not specified, AppMetrica generates the ID automatically.
3. Send the `YMMMutableRevenueInfo` instance to the test API key using the YMMYandexMetricaReporting reporter. For more information on reporters, see Sending statistics to an additional API key.

```
let price = NSDecimalNumber(string: "2100.5")
// Initializing the Revenue instance.
let revenueInfo = YMMMutableRevenueInfo.init(priceDecimal: price, currency: "BYN")
revenueInfo.productID = "TV soundbar"
revenueInfo.quantity = 2
// To group purchases, set the OrderID parameter in the payload property.
revenueInfo.payload = ["OrderID": "Identifier", "source": "AppStore"]
// Sending the Revenue instance using reporter.
let reporter = YMMYandexMetrica.reporterForApiKey("API_key")
reporter.reportRevenue(revenueInfo, onFailure: { (error) in
    print("REPORT ERROR: \(error.localizedDescription)")
})
```

Step 2. Configure sending Revenue to the main API key

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

Objective-C

To send the `YMMMutableRevenueInfo` instance to the main API key, use the +reportRevenue:onFailure: method of the YMMYandexMetrica class.

```
...
// Sending the Revenue instance.
YMMYandexMetrica.reportRevenue(revenueInfo, onFailure: { error in
    print("Revenue error: \(error)")
})
```

Swift

To send the `YMMMutableRevenueInfo` instance to the main API key, use the reportRevenue(_:onFailure:) method of the YMMYandexMetrica class.

```
...
// Sending the Revenue instance.
YMMYandexMetrica.reportRevenue(revenueInfo, onFailure: { (error) in
    print("Revenue error: \(error)")
```

```
})
```

Setting the length of the session timeout

Objective-C

To change the length of the timeout, pass the value in seconds to the sessionTimeout property of the library configuration YMMYandexMetricaConfiguration.

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

```
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
// Setting the session timeout.
configuration.sessionTimeout = 15;
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

Swift

To change the length of the timeout, pass the value in seconds to the sessionTimeout property of the library configuration YMMYandexMetricaConfiguration.

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

```
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
// Setting the session timeout.
configuration?.sessionTimeout = 15
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

Setting the app version

Objective-C

By default, the app version is set in the app configuration file `Info.plist (CFBundleShortVersionString)`.

To specify the app version from the code, pass the app version to the appVersion property of the library configuration YMMYandexMetricaConfiguration.

```
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
// Setting the app version.
configuration.appVersion = @"1.13.2";
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

Swift

By default, the app version is set in the app configuration file `Info.plist (CFBundleShortVersionString)`.

To specify the app version from the code, pass the app version to the appVersion property of the library configuration YMMYandexMetricaConfiguration.

```
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
// Setting the app version.
configuration?.appVersion = "1.13.2"
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

Tracking app openings using deeplinks

Objective-C

You need to track app openings to correctly track remarketing campaigns.

Note: To work with Universal Links, configure them for your application.

To track app openings via deeplinks and Universal Links, use the +handleOpenURL: method of the YMMYandexMetrica class.

To track app openings using a deeplink, make the following changes in `UIApplicationDelegate`:

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
{
    return [YMMYandexMetrica handleOpenURL:url];
}
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:
(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options
{
    return [YMMYandexMetrica handleOpenURL:url];
}

// Delegate for tracking Universal links.
- (BOOL)application:(UIApplication *)application
    continueUserActivity:(NSUserActivity *)userActivity
    restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler
{
    if ([userActivity.activityType isEqualToString:NSUserActivityTypeBrowsingWeb]) {
        [YMMYandexMetrica handleOpenURL:userActivity.webpageURL];
    }
    return YES;
}
```

Swift

You need to track app openings to correctly track remarketing campaigns.

Note: To work with Universal Links, configure them for your application.

To track app openings via deeplinks or Universal Links, use the handleOpen(_:) method of the YMMYandexMetrica class.

To track app openings using a deeplink, make the following changes in `UIApplicationDelegate`:

```
func application(_ application: UIApplication, handleOpenURL url: URL) -> Bool {
    return YMMYandexMetrica.handleOpen(url)
}

func application(_ application: UIApplication, openURL url: URL, sourceApplication: String?, annotation:
 AnyObject) -> Bool {
    return YMMYandexMetrica.handleOpen(url)
}

// Delegate for tracking Universal links.
func application(_ application: UIApplication, continueUserActivity userActivity: NSUserActivity,
 restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if userActivity.activityType == NSUserActivityTypeBrowsingWeb {
        if let url = userActivity.webpageURL {
            YMMYandexMetrica.handleOpen(url)
        }
    }
    return true
}
```

See also

How to create a remarketing campaign in AppMetrica

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 configure this, initialize the AppMetrica SDK using the extended configuration `YMMYandexMetricaConfiguration`.

Objective-C

```
BOOL isFirstLaunch = NO;
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
// Implement the logic for detecting whether the app is starting 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 = YES;
}
configuration.handleFirstActivationAsUpdate = !isFirstLaunch;
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

Swift

```
var isFirstLaunch = false
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
```

```
// Implement the logic for detecting whether the app is starting 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
}
configuration?.handleFirstActivationAsUpdate = !isFirstLaunch
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

Disable and enable sending statistics

Objective-C

If you need confirmation from the user before sending statistical data, you should initialize the library with the disabled sending option. To do this, set the NO value for the statisticsSending property of the extended library configuration YMMYandexMetricaConfiguration.

```
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
// Disabling sending statistics.
configuration.statisticsSending = NO;
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

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 +setStatisticsSending: method of the YMMYandexMetrica class to enable it.

```
// Checking the status of the boolean variable. It shows the user confirmation.
if (flag) {
    // Enabling sending statistics.
    [YMMYandexMetrica setStatisticsSending:YES];
}
```

Swift

If you need confirmation from the user before sending statistical data, you should initialize the library with the disabled sending option. To do this, set the NO value for the statisticsSending property of the extended library configuration YMMYandexMetricaConfiguration.

```
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
// Disabling sending statistics.
configuration?.statisticsSending = false
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

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 setStatisticsSending(_:) method of the YMMYandexMetrica class to enable it.

```
// Checking the status of the boolean variable. It shows the user confirmation.
if flag {
    // Enabling sending statistics.
    YMMYandexMetrica.setStatisticsSending(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.

Reference

Objective-C

Classes

YMMAdRevenueInfo class

The class contains immutable information about advertising revenue (Ad Revenue).

Use the YMMMutableAdRevenueInfo class to change information about advertising revenue.

The `YMMAdRevenueInfo` instance should be passed to the AppMetrica server using the reportAdRevenue method of the YMMYandexMetrica class.

Instance methods

-initWithAdRevenue:adRevenue:currency:	Initializes the instance of the `YMMAdRevenueInfo` class for sending information about advertising revenue.

Properties

adRevenue	The amount of money received from advertising revenue. Can't be negative.
currency	The currency in which adRevenue is represented. Must be in ISO-4217 format.
adType	Ad type. See the possible values in YMMAdType.
adNetwork	Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitID	Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitName	Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementID	Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementName	Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
precision	Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
payload	Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

Method descriptions

-initWithAdRevenue:adRevenue:currency:

```
- (instancetype)initWithAdRevenue:(NSDecimalNumber *)adRevenue currency:(NSString *)currency
```

Initializes the instance of the `YMMAdRevenueInfo` class for sending information about advertising revenue.

Parameters:

adRevenue	The amount of money received from advertising revenue. Can't be negative.
currency	The currency in which adRevenue is represented. Must be in ISO-4217 format.

Returns:

The `YMMAdRevenueInfo` class instance.

Property descriptions

adRevenue

`(nonatomic, strong, readonly) NSDecimalNumber *adRevenue`

The amount of money received from advertising revenue. Can't be negative.

currency

`(nonatomic, copy, readonly) NSString *currency`

The currency in which adRevenue is represented. Must be in ISO-4217 format.

adType

`(nonatomic, assign, readonly) YMMAdType adType`

Ad type. See the possible values in YMMAdType.

adNetwork

`(nonatomic, copy, nullable, readonly) NSString *adNetwork`

Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitID

`(nonatomic, copy, nullable, readonly) NSString *adUnitID`

Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitName

`(nonatomic, copy, nullable, readonly) NSString *adUnitName`

Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementID

`(nonatomic, copy, nullable, readonly) NSString *adPlacementID`

Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementName

```
(nonatomic, copy, nullable, readonly) NSString *adPlacementName
```

Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

precision

```
(nonatomic, copy, nullable, readonly) NSString *precision
```

Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

payload

```
(nonatomic, copy, nullable, readonly) NSDictionary<NSString *, NSString *> *payload
```

Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

YMMECommerce class

Methods of this class create a `YMMECommerce` instance.

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

Note: You can send the `YMMECommerce` instance using the +reportECommerce:onFailure: method of the YMMYandexMetrica class and the YMMYandexMetricaReporting protocol.

Instance methods

+showScreenEventWithScreen:	Creates an E-commerce event called `ShowScreenEvent`. Use it to report the opening of a page, such as a list of products, search or home page.
+showProductCardEventWithProduct:screen:	Creates an E-commerce event called `ShowProductCardEvent`. Use it to report viewing a product profile among others on the list.
+showProductDetailsEventWithProduct:referrer:	Creates an E-commerce event called `ShowProductDetailsEvent`. Use it to report viewing a product page.
+addCartItemEventWithItem:	Creates an E-commerce event called `AddCartItemEvent`. Use it to report adding an item to the cart.
+removeCartItemEventWithItem:	Creates an E-commerce event called `RemoveCartItemEvent`. Use it to report removing an item from the cart.
+beginCheckoutEventWithOrder:	Creates an E-commerce event called `BeginCheckoutEvent`. Use it to report starting a purchase.
+purchaseEventWithOrder:	Creates an E-commerce event called `PurchaseEvent`. Use it to report a completed purchase.

Method descriptions

+showScreenEventWithScreen:

```
+ (instancetype)showScreenEventWithScreen:(YMMECommerceScreen *)screen
```

Creates an E-commerce event called `ShowScreenEvent`. Use it to report the opening of a page, such as a list of products, search or home page.

Parameters:

screen	The screen that was opened. The YMMECommerceScreen class instance.

Returns:

The `YMMECommerce` class instance.

+showProductCardEventWithProduct:screen:

```
+ (instancetype)showProductCardEventWithProduct:(YMMECommerceProduct *)product
                                         screen:(YMMECommerceScreen *)screen
```

Creates an E-commerce event called `ShowProductCardEvent`. Use it to report viewing a product profile among others on the list.

Tip: Before sending the event, make sure that the product profile was shown on the screen for more than N seconds.

Parameters:

product	The product that was shown. The YMMECommerceProduct class instance.
screen	The screen where the product was displayed. The YMMECommerceScreen class instance.

Returns:

The `YMMECommerce` class instance.

+showProductDetailsEventWithProduct:referrer:

```
+ (instancetype)showProductDetailsEventWithProduct:(YMMECommerceProduct *)product
                                          referrer:(nullable YMMECommerceReferrer *)referrer
```

Creates an E-commerce event called `ShowProductDetailsEvent`. Use it to report viewing a product page.

Parameters:

product	The product that was shown. The YMMECommerceProduct class instance.
referrer	Information about the source of traffic to the product page. The YMMECommerceReferrer class instance.

Returns:

The `YMMECommerce` class instance.

+addCartItemEventWithItem:

```
+ (instancetype)addCartItemEventWithItem:(YMMECommerceCartItem *)item
```

Creates an E-commerce event called `AddCartItemEvent`. Use it to report adding an item to the cart.

Parameters:

item	The item that was added to the cart. The YMMECommerceCartItem class instance.

Returns:

The `YMMECommerce` class instance.

+removeCartItemEventWithItem:

```
+ (instancetype)removeCartItemEventWithItem:(YMMECommerceCartItem *)item
```

Creates an E-commerce event called `RemoveCartItemEvent`. Use it to report removing an item from the cart.

Parameters:

item	The item that was removed from the cart. The YMMECommerceCartItem class instance.

Returns:

The `YMMECommerce` class instance.

+beginCheckoutEventWithOrder:

```
+ (instancetype)beginCheckoutEventWithOrder:(YMMECommerceOrder *)order
```

Creates an E-commerce event called `BeginCheckoutEvent`. Use it to report starting a purchase.

Parameters:

order	Information about the purchase. The YMMECommerceOrder class instance.

Returns:

The `YMMECommerce` class instance.

+purchaseEventWithOrder:

```
+ (instancetype)purchaseEventWithOrder:(YMMECommerceOrder *)order
```

Creates an E-commerce event called `PurchaseEvent`. Use it to report a completed purchase.

Parameters:

order	Information about the purchase. The YMMECommerceOrder class instance.

Returns:

The `YMMECommerce` class instance.

YMMECommerceAmount class

This class contains cost information, such as quantity and units.

Instance methods

-initWithUnit:value:	Initializes the instance of the `YMMECommerceAmount` class for sending information about purchases.

Properties

unit	The unit. For example: USD or RUB. Acceptable value: Up to 20 characters. **Note:** Use ISO 4217 format.
value	Quantity.

Method descriptions

-initWithUnit:value:

```
- (instancetype)initWithUnit:(NSString *)unit value:(NSDecimalNumber *)value
```

Initializes the instance of the `YMMECommerceAmount` class for sending information about purchases.

Parameters:

unit	The unit. For example: USD or RUB. Acceptable value: Up to 20 characters. **Note:** Use ISO 4217 format.
value	Quantity.

Returns:

The `YMMECommerceAmount` class instance.

Property descriptions

unit

`(nonatomic, copy, readonly) NSString *unit`

The unit. For example: USD or RUB. Acceptable value: Up to 20 characters.

Note: Use ISO 4217 format.

value

`(nonatomic, strong, readonly) NSDecimalNumber *value`

Quantity.

YMMECommerceCartItem class

This class contains information about items added to the cart.

Instance methods

-initWithProduct:quantity:revenue:referrer:	Initializes the instance of the `YMMECommerceCartItem` class with information about items added to the cart.

Properties

product	Product. The `YMMECommerceProduct` class instance.
quantity	Quantity.
revenue	The total price of the product in the cart. This price factors in the quantity, discounts to be applied, and so on. The `YMMECommercePrice` class instance.

referrer	The source of traffic to the cart. The `YMMECommerceReferrer` class instance.

Method descriptions

-initWithProduct:referrer:quantity:revenue:

```
- (instancetype)initWithProduct:(YMMECommerceProduct *)product
                       quantity:(NSDecimalNumber *)quantity
                        revenue:(YMMECommercePrice *)revenue
                       referrer:(nullable YMMECommerceReferrer *)referrer;
```

Initializes the instance of the `YMMECommerceCartItem` class with information about items added to the cart.

Parameters:

product	Product. The `YMMECommerceProduct` class instance.
quantity	Quantity.
revenue	The total price of the product in the cart. This price factors in the quantity, discounts to be applied, and so on. The `YMMECommercePrice` class instance.
referrer	The source of traffic to the cart. The `YMMECommerceReferrer` class instance.

Returns:

The `YMMECommerceCartItem` class instance.

Property descriptions

product

`(nonatomic, strong, readonly) YMMECommerceProduct *product`

Product. The `YMMECommerceProduct` class instance.

quantity

`(nonatomic, strong, readonly) NSDecimalNumber *quantity`

Quantity.

revenue

`(nonatomic, strong, readonly) YMMECommercePrice *revenue`

The total price of the product in the cart. This price factors in the quantity, discounts to be applied, and so on. The `YMMECommercePrice` class instance.

referrer

`(nonatomic, strong, readonly, nullable) YMMECommerceReferrer *referrer`

The source of traffic to the cart. The `YMMECommerceReferrer` class instance.

YMMECommerceOrder class

This class contains order information.

Instance methods

-initWithIdentifier:cartItems:	Initializes the instance of the `YMMECommerceOrder` class with order information.
-initWithIdentifier:cartItems:payload:	Initializes the instance of the `YMMECommerceOrder` class with order information.

Properties

identifier	Order ID. Allowed size: Up to 100 characters.
cartItems	A list of items in the cart.
payload	Additional information about the order. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Method descriptions

-initWithIdentifier:cartItems:

```
- (instancetype)initWithIdentifier:(NSString *)identifier
                         cartItems:(NSArray<YMMECommerceCartItem *> *)cartItems
```

Initializes the instance of the `YMMECommerceOrder` class with order information.

Parameters:

identifier	Order ID. Allowed size: Up to 100 characters.
cartItems	A list of items in the cart.

Returns:

The `YMMECommerceOrder` class instance.

-initWithIdentifier:cartItems:payload:

```
- (instancetype)initWithIdentifier:(NSString *)identifier
                         cartItems:(NSArray<YMMECommerceCartItem *> *)cartItems
                           payload:(nullable NSDictionary<NSString *, NSString *> *)payload
```

Initializes the instance of the `YMMECommerceOrder` class with order information.

Parameters:

identifier	Order ID. Allowed size: Up to 100 characters.
cartItems	A list of items in the cart.
payload	Additional information about the order. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Returns:

The `YMMECommerceOrder` class instance.

Property descriptions

identifier

```
(nonatomic, copy, readonly) NSString *identifier
```

Order ID. Allowed size: Up to 100 characters.

cartItems

```
(nonatomic, copy, readonly) NSArray<YMMECommerceCartItem *> *cartItems
```

A list of items in the cart.

payload

```
(nonatomic, copy, readonly, nullable) nullable NSDictionary<NSString *, NSString *>
*payload
```

Additional information about the order. Acceptable sizes:

- The total `payload` size: Up to 20 KB.
- The `key` size: Up to 100 characters.
- The `value` size: Up to 1000 characters.

YMMECommercePrice class

This class contains the product price.

Instance methods

-initWithFiat:	Initializes the instance of the `YMMECommercePrice` class.
-initWithFiat:internalComponents:	Initializes the instance of the `YMMECommercePrice` class.

Properties

fiat	The cost in fiat money. The `YMMECommerceAmount` class instance.
internalComponents	The cost of internal components (the amounts in the internal currency). Allowed size: Up to 30 elements.

Method descriptions

-initWithFiat:

```
- (instancetype)initWithFiat:(YMMECommerceAmount *)fiat
```

Initializes the instance of the `YMMECommercePrice` class.

Parameters:

fiat	The cost in fiat money. The `YMMECommerceAmount` class instance.

Returns:

The `YMMECommercePrice` class instance.

-initWithFiat:internalComponents:

```
- (instancetype)initWithFiat:(YMMECommerceAmount *)fiat
         internalComponents:(nullable NSArray<YMMECommerceAmount *> *)internalComponents;
```

Initializes the instance of the `YMMECommercePrice` class.

Parameters:

fiat	The cost in fiat money. The `YMMECommerceAmount` class instance.
internalComponents	The cost of internal components (the amounts in the internal currency). Allowed size: Up to 30 elements.

Returns:

The `YMMECommercePrice` class instance.

Property descriptions

fiat

`(nonatomic, strong, readonly) YMMECommerceAmount *fiat;`

The cost in fiat money. The `YMMECommerceAmount` class instance.

internalComponents

`(nonatomic, copy, readonly, nullable) NSArray<YMMECommerceAmount *> *internalComponents`

The cost of internal components (the amounts in the internal currency). Allowed size: Up to 30 elements.

YMMECommerceProduct class

This class contains product information.

Instance methods

-initWithSKU:	Initializes the instance of the `YMMECommerceProduct` class with the specified item number.
-initWithSKU:name:categoryComponents:payload:actualPrice:originalPrice:promoCodes:	Initializes the instance of the `YMMECommerceProduct` class with all parameters.

Properties

sku	Item number. Allowed size: Up to 100 characters.
name	Name of the product. Allowed size: Up to 1000 characters.
categoryComponents	The path to the product by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.

payload	Additional information about the product. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.
actualPrice	The actual product price, which is the price after applying all discounts and promo codes.
originalPrice	The initial product price.
promoCodes	A list of promo codes that are applied to the product. Acceptable sizes: • Up to 20 elements. • The promo code length is up to 100 characters.

Method descriptions

-initWithSKU:

```
- (instancetype)initWithSKU:(NSString *)sku
```

Initializes the instance of the `YMMECommerceProduct` class with the specified item number.

Parameters:

sku	Item number. Allowed size: Up to 100 characters.

Returns:

The `YMMECommerceProduct` class instance.

#initWithSKU:name:categoryComponents:payload:actualPrice:originalPrice:promoCodes:

```
- (instancetype)initWithSKU:(NSString *)sku
                       name:(nullable NSString *)name
         categoryComponents:(nullable NSArray<NSString *> *)categoryComponents
                    payload:(nullable NSDictionary<NSString *, NSString *> *)payload
              actualPrice:(nullable YMMECommercePrice *)actualPrice
             originalPrice:(nullable YMMECommercePrice *)originalPrice
                promoCodes:(nullable NSArray<NSString *> *)promoCodes;
```

Initializes the instance of the `YMMECommerceProduct` class with all parameters.

Parameters:

sku	Item number. Allowed size: Up to 100 characters.
name	Name of the product. Allowed size: Up to 1000 characters.
categoryComponents	The path to the product by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.

payload	Additional information about the product. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.
actualPrice	The actual product price, which is the price after applying all discounts and promo codes.
originalPrice	The initial product price.
promoCodes	A list of promo codes that are applied to the product. Acceptable sizes: • Up to 20 elements. • The promo code length is up to 100 characters.

Returns:

The `YMMECommerceProduct` class instance.

Property descriptions

sku

```
(nonatomic, copy, readonly) NSString *sku
```

Item number. Allowed size: Up to 100 characters.

name

```
(nonatomic, copy, readonly, nullable) NSString *name
```

Name of the product. Allowed size: Up to 1000 characters.

categoryComponents

```
(nonatomic, copy, readonly, nullable) NSArray<NSString *> *categoryComponents
```

The path to the product by category. Acceptable sizes:

- Up to 10 elements.
- The size of a single element is up to 100 characters.

payload

```
(nonatomic, copy, readonly, nullable) NSDictionary<NSString *, NSString *> *payload
```

Additional information about the product. Acceptable sizes:

- The total `payload` size: Up to 20 KB.
- The `key` size: Up to 100 characters.
- The `value` size: Up to 1000 characters.

actualPrice

```
(nonatomic, strong, readonly, nullable) YMMECommercePrice *actualPrice
```

The actual product price, which is the price after applying all discounts and promo codes.

originalPrice

`(nonatomic, strong, readonly, nullable) YMMECommercePrice *originalPrice`

The initial product price.

promoCodes

`(nonatomic, copy, readonly, nullable) NSArray<NSString *> *promoCodes`

A list of promo codes that are applied to the product. Acceptable sizes:

- Up to 20 elements.
- The promo code length is up to 100 characters.

YMMECommerceReferrer class

This class contains information about the source of traffic. For example, a link to the page or screen that shows a product profile.

Instance methods

-initWithType:identifier:screen:	Initializes the instance of the `YMMECommerceReferrer` class with information about the source of traffic.

Properties

type	Traffic source type: The type of object that traffic comes from. For example: “button”, “banner”, or “href”. Allowed size: Up to 100 characters.
identifier	Traffic source ID. Allowed size: Up to 2048 characters.
screen	Traffic source screen: The screen that traffic comes from.

Method descriptions

-initWithType:identifier:screen:

```
- (instancetype)initWithType:(nullable NSString *)type
                  identifier:(nullable NSString *)identifier
                      screen:(nullable YMMECommerceScreen *)screen
```

Initializes the instance of the `YMMECommerceReferrer` class with information about the source of traffic.

Parameters:

type	Traffic source type: The type of object that traffic comes from. For example: “button”, “banner”, or “href”. Allowed size: Up to 100 characters.
identifier	Traffic source ID. Allowed size: Up to 2048 characters.
screen	Traffic source screen: The screen that traffic comes from.

Returns:

The `YMMECommerceReferrer` class instance.

Property descriptions

type

`(nonatomic, copy, readonly, nullable) NSString *type`

Traffic source type: The type of object that traffic comes from. For example: “button”, “banner”, or “href”. Allowed size: Up to 100 characters.

identifier

```
(nonatomic, copy, readonly, nullable) NSString *identifier
```

Traffic source ID. Allowed size: Up to 2048 characters.

screen

```
(nonatomic, strong, readonly, nullable) YMMECommerceScreen *screen
```

Traffic source screen: The screen that traffic comes from.

YMMECommerceScreen class

This class contains screen information.

Instance methods

-initWithName:	Initializes the instance of the `YMMECommerceScreen` class with the specified screen name.
-initWithCategoryComponents:	Initializes the instance of the `YMMECommerceScreen` class with the specified path to the screen.
-initWithSearchQuery:	Initializes the instance of the `YMMECommerceScreen` class with the specified search query.
-initWithPayload:	Initializes the instance of the `YMMECommerceScreen` class with the specified additional information.
-initWithName:categoryComponents:searchQuery:payload:	Initializes the instance of the `YMMECommerceScreen` class with all parameters.

Properties

name	Screen name. Acceptable sizes: Up to 100 characters.
categoryComponents	The path to the screen by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.
searchQuery	Search query. Allowed size: Up to 1000 characters.
payload	Additional information about the screen. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Method descriptions

-initWithName:

```
- (instancetype)initWithName:(NSString *)name
```

Initializes the instance of the `YMMECommerceScreen` class with the specified screen name.

Parameters:

name	Screen name. Acceptable sizes: Up to 100 characters.

Returns:

The `YMMECommerceScreen` class instance.

-initWithCategoryComponents:

```
(instancetype)initWithCategoryComponents:(NSArray<NSString *> *)categoryComponents
```

Initializes the instance of the `YMMECommerceScreen` class with the specified path to the screen.

Parameters:

categoryComponents	The path to the screen by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.

Returns:

The `YMMECommerceScreen` class instance.

-initWithSearchQuery:

```
- (instancetype)initWithSearchQuery:(NSString *)searchQuery
```

Initializes the instance of the `YMMECommerceScreen` class with the specified search query.

Parameters:

searchQuery	Search query. Allowed size: Up to 1000 characters.

Returns:

The `YMMECommerceScreen` class instance.

-initWithPayload:

```
- (instancetype)initWithPayload:(NSDictionary<NSString *, NSString *> *)payload;
```

Initializes the instance of the `YMMECommerceScreen` class with the specified additional information.

Parameters:

payload	Additional information about the screen. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Returns:

The `YMMECommerceScreen` class instance.

#initWithName:categoryComponents:searchQuery:payload:

```
- (instancetype)initWithName:(nullable NSString *)name
         categoryComponents:(nullable NSArray<NSString *> *)categoryComponents
                searchQuery:(nullable NSString *)searchQuery
                    payload:(nullable NSDictionary<NSString *, NSString *> *)payload
```

Initializes the instance of the `YMMECommerceScreen` class with all parameters.

Parameters:

name	Screen name. Acceptable sizes: Up to 100 characters.
categoryComponents	The path to the screen by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.
searchQuery	Search query. Allowed size: Up to 1000 characters.
payload	Additional information about the screen. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Returns:

The `YMMECommerceScreen` class instance.

Property descriptions

name

`(nonatomic, copy, readonly, nullable) NSString *name`

Screen name. Acceptable sizes: Up to 100 characters.

categoryComponents

`(nonatomic, copy, readonly, nullable) NSArray<NSString *> *categoryComponents`

The path to the screen by category. Acceptable sizes:

- Up to 10 elements.
- The size of a single element is up to 100 characters.

searchQuery

`(nonatomic, copy, readonly, nullable) NSString *searchQuery`

Search query. Allowed size: Up to 1000 characters.

payload

`(nonatomic, copy, readonly, nullable) NSDictionary<NSString *, NSString *> *payload`

Additional information about the screen. Acceptable sizes:

- The total `payload` size: Up to 20 KB.
- The `key` size: Up to 100 characters.
- The `value` size: Up to 1000 characters.

YMMError class

A simple implementation of the YMMErrorRepresentable protocol.

Instance methods

+errorWithIdentifier:	Creates the `YMMError` instance with the specified ID.
+errorWithIdentifier:message:parameters:	Creates the `YMMError` instance with the specified ID and other parameters.
+errorWithIdentifier:message:parameters:backtrace:underlyingError:	Creates the `YMMError` instance with the specified ID and other parameters.

Method descriptions

+errorWithIdentifier:

```
+ (nonnull instancetype)errorWithIdentifier:(nonnull NSString *)identifier;
```

Creates the `YMMError` instance with the specified ID.

Parameters:

identifier	Error ID. AppMetrica uses it to group errors.

Returns:

The `YMMError` class instance.

+errorWithIdentifier:message:parameters:

```
+ (nonnull instancetype) errorWithIdentifier:(nonnull NSString *)identifier
                                     message:(nullable NSString *)message
                                  parameters:(nullable NSDictionary<NSString *, id> *)parameters;
```

Creates the `YMMError` instance with the specified ID and other parameters.

Parameters:

identifier	Error ID. AppMetrica uses it to group errors.
message	Arbitrary error description
parameters	Additional error parameters

Returns:

The `YMMError` class instance.

+errorWithIdentifier:message:parameters:backtrace:underlyingError

```
+ (nonnull instancetype) errorWithIdentifier:(nonnull NSString *)identifier
                                     message:(nullable NSString *)message
                                  parameters:(nullable NSDictionary<SString *, id> *)parameters
                                   backtrace:(nullable NSArray<NSNumber *> *)backtrace
                             underlyingError:(nullable id<YMMErrorRepresentable>)underlyingError;
```

Creates the `YMMError` instance with the specified ID and other parameters.

Parameters:

identifier	Error ID. AppMetrica uses it to group errors.
message	Arbitrary error description
parameters	Additional error parameters

backtrace	Custom error backtrace
underlyingError	Error instance that matches the YMMErrorRepresentable protocol.

Returns:

The `YMMError` class instance.

YMMMutableAdRevenueInfo class

The mutable version of the YMMAdRevenueInfo class with information about advertising revenue (Ad Revenue).

The `YMMMutableAdRevenueInfo` instance should be passed to the AppMetrica server using the reportAdRevenue method of the YMMYandexMetrica class.

Properties

adType	Ad type. See possible values in YMMAAdType.
adNetwork	Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitID	Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitName	Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementID	Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementName	Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
precision	Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
payload	Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

Property descriptions

adType

```
(nonatomic, assign) YMMAdType adType
```

Ad type. See possible values in YMMAAdType.

adNetwork

```
(nonatomic, copy, nullable) NSString *adNetwork
```

Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitID

```
(nonatomic, copy, nullable) NSString *adUnitID
```

Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitName

```
(nonatomic, copy, nullable) NSString *adUnitName
```

Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementID

```
(nonatomic, copy, nullable) NSString *adPlacementID
```

Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementName

```
(nonatomic, copy, nullable) NSString *adPlacementName
```

Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

precision

```
(nonatomic, copy, nullable) NSString *precision
```

Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

payload

```
(nonatomic, copy, nullable) NSDictionary<NSString *, NSString *> *payload
```

Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

YMMMutableReporterConfiguration class

The mutable version of the YMMReporterConfiguration class with the advanced configuration of the reporter.

Properties

logs	Enables/disables logging the activity of the library.
maxReportsInDatabaseCount	The maximum number of error reports stored in the internal DB.
sessionTimeout	Sets the session timeout in seconds.
statisticsSending	Enables/disables sending statistics to the AppMetrica server.
userProfileID	Sets the ID of the user profile (`ProfileID`) when activated.

Property descriptions

logs

```
(nonatomic, assign) BOOL logs
```

Enables/disables logging the activity of the library.

Logging is disabled by default.

maxReportsInDatabaseCount

```
@property (assign, readwrite, nonatomic) NSUInteger maxReportsInDatabaseCount;
```

The maximum number of error reports stored in the internal DB.

The allowed range of values is [100; 10000]. Values outside this range are automatically replaced with values from the nearest range limits.

Default value: 1000.

Note: Separate databases are used for various `apiKeys` and independent limits on the number of events can be set for them. This parameter only affects the limitation for the corresponding `apiKey`. To change the maximum allowed number of events for other `apiKeys`, use YMMYandexMetricaConfiguration.maxReportsInDatabaseCount.

sessionTimeout

```
(nonatomic, assign) NSUInteger sessionTimeout
```

Sets the session timeout in seconds.

The default value is `10` (minimum allowed value).

statisticsSending

```
(nonatomic, assign) BOOL statisticsSending
```

Enables/disables sending statistics to the AppMetrica server.

Note: Disable sending statistics to the reporter does not affect the sending of data from the main API key. But disabling data sending for the main API key stops sending statistics from all reporters.

By default, sending is enabled.

userProfileID

```
(nonatomic, copy, nullable) NSString *userProfileID
```

Sets the ID of the user profile (`ProfileID`) when activated.

Attention: The maximum length of the `ProfileID` string is 200 characters.

YMMMutableRevenueInfo class

The mutable version of the YMMRevenueInfo class with information about purchases.

The instance of the `YMMRevenueInfo` class should be sent to the AppMetrica server using the `reportRevenue` method of the YMMYandexMetrica class.

Properties

payload	Additional information to be passed about the purchase. For instance, it can be used for categorizing your products.
productID	ID of the product purchased. The value can contain up to 200 characters.
quantity	Quantity of products purchased.
receiptData	Details about the in-app purchase order from App Store. This property is used for validating purchases in the app. For more information, see Apple documentation.
transactionID	Transaction ID transactionIdentifier from the SKPaymentTransaction class. This property is used for validating purchases in the app.

Property descriptions

payload

`(nonatomic, copy) NSDictionary *payload`

Additional information to be passed about the purchase. For instance, it can be used for categorizing your products.

It should contain the `NSDictionary` object that can be converted to valid JSON. The maximum size of the value is 30 KB.

productID

`(nonatomic, copy) NSString *productID`

ID of the product purchased. The value can contain up to 200 characters.

quantity

`(nonatomic, assign) NSUInteger quantity`

Quantity of products purchased.

It is used in the following formula:

```
Revenue = quantity * price.
```

Note: The value cannot be negative. If the value is equal to 0, the purchase is ignored.

receiptData

`(nonatomic, copy) NSData *receiptData`

Details about the in-app purchase order from App Store. This property is used for validating purchases in the app. For more information, see Apple documentation.

See the example of getting `receiptData` in the section Sending Revenue.

Attention: The value must be received before invoking `SKPaymentQueue.default().finishTransaction(transaction)` and transmitted together with transactionID.

transactionID

`(nonatomic, copy) NSString *transactionID`

Transaction ID transactionIdentifier from the SKPaymentTransaction class. This property is used for validating purchases in the app.

See the example of getting `transactionIdentifier` in the section Sending Revenue.

Attention: This value should be passed along with receiptData.

YMMMutableUserProfile class

The mutable version of the YMMUserProfile class with information about a user profile.

A user profile is a set of user attributes. User profile details are displayed in the AppMetrica report on user profiles.

The `YMMMutableUserProfile` instance should be passed to the AppMetrica server by using the `reportUserProfile` method of the YMMYandexMetrica class. Use methods of the YMMProfileAttribute class to create user attributes.

Instance methods

-apply:	Updates a single attribute of the user profile.
-applyFromArray:	Updates several attributes of the user profile.

Method descriptions

-apply:

```
- (void)apply:(YMMUserProfileUpdate *)update
```

Updates a single attribute of the user profile.

Parameters:

update	The `YMMUserProfileUpdate` class instance.

-applyFromArray:

```
- (void)applyFromArray:(NSArray<YMMUserProfileUpdate *>*)updatesArray
```

Updates several attributes of the user profile.

Parameters:

updatesArray	Array of `YMMUserProfileUpdate` objects.

YMMProfileAttribute class

Methods of the class create predefined and custom profile attributes.

AppMetrica lets you create up to 100 custom attributes.

Instance methods

-birthDate	Creates a birth date attribute.
-customBool:	Creates a custom attribute with the `bool` type.
-customCounter:	Creates a custom attribute of the counter type.
-customNumber:	Creates a custom attribute with the `double` type.
-customString:	Creates a custom attribute with the `string` type.
-gender	Creates a gender attribute.
-name	Creates a name attribute.
-notificationsEnabled	Creates a NotificationsEnabled attribute.

Method descriptions

+birthDate

```
+ (id<YMMBirthDateAttribute>)birthDate
```

Creates a birth date attribute.

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

Returns:

The instance that implements the `YMMBirthDateAttribute` protocol.

+customBool:

```
+ (id<YMMCustomBoolAttribute>)customBool:(NSString *)name
```

Creates a custom attribute with the bool type.

`bool`

tab.

Parameters:

name	Attribute name. The value can contain up to 200 characters.

Returns:

The instance that implements the `YMMCustomBoolAttribute` protocol.

+customCounter:

```
+ (id>YMMCustomCounterAttribute>)customCounter:(NSString *)name
```

Creates a custom attribute of the counter type.

Parameters:

name	Creates a custom attribute of the counter type.

Returns:

The instance that implements the `YMMCustomCounterAttribute` protocol.

+customNumber:

```
+ (id<YMMCustomNumberAttribute>)customNumber:(NSString *)name
```

Creates a custom attribute with the `double` type.

Parameters:

name	Creates a custom attribute of the counter type.

Returns:

The instance that implements the `YMMCustomNumberAttribute` protocol.

+customString:

```
+ (id<YMMCustomStringAttribute>)customString:(NSString *)name
```

Creates a custom attribute with the `string` type.

Parameters:

name	Attribute name. The value can contain up to 200 characters.

Returns:

The instance that implements the `YMMCustomStringAttribute` protocol.

+gender

```
+ (id<YMMGenderAttribute>)gender
```

Creates a gender attribute.

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

Returns:

The instance that implements the `YMMGenderAttribute` protocol.

+name

```
+ (id<YMMNameAttribute>)name
```

Creates a name attribute.

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

Returns:

The instance that implements the `YMMNameAttribute` protocol.

+notificationsEnabled

```
+ (id<YMMNotificationsEnabledAttribute>)notificationsEnabled
```

Creates a NotificationsEnabled attribute.

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

Returns:

The instance that implements the `YMMNotificationsEnabledAttribute` protocol.

YMMReporterConfiguration class

This class contains the extended immutable configuration of the reporter.

Use the YMMMutableReporterConfiguration class to change the configuration of a reporter.

Instance methods

-initWithApiKey:	Initializes an instance of the class `YMMReporterConfiguration` with the specified API key.

Properties

apiKey	API key that differs from the main application API key.
logs	The flag indicating that the logging of the reporter is enabled.
maxReportsInDatabaseCount	The maximum number of error reports stored in the internal DB.
sessionTimeout	Session timeout in seconds.
statisticsSending	A flag indicating that sending statistics is enabled.
userProfileID	Sets the ID of the user profile (`ProfileID`) when activated.

Method descriptions

-initWithApiKey:

```
- (instancetype)initWithApiKey:(NSString *)apiKey
```

Initializes an instance of the class `YMMReporterConfiguration` with the specified API key.

Parameters:

apiKey	API key that differs from the main application API key.

Returns:

The instance of the `YMMReporterConfiguration` class.

Property descriptions

apiKey

```
(nonatomic, copy, nullable, readonly) NSString *apiKey
```

API key that differs from the main application API key.

logs

```
(nonatomic, assign, readonly) BOOL logs
```

The flag indicating that the logging of the reporter is enabled.

The default value is `NO`.

Possible values:

- `YES` — Reporter logging is enabled.
- `NO` — Reporter logging is disabled.

maxReportsInDatabaseCount

```
@property (readonly, assign, nonatomic) NSUInteger maxReportsInDatabaseCount;
```

The maximum number of error reports stored in the internal DB.

The allowed range of values is [100; 10000]. Values outside this range are automatically replaced with values from the nearest range limits.

Default value: 1000.

Note: Separate databases are used for various `apiKeys` and independent limits on the number of events can be set for them. This parameter only affects the limitation for the corresponding `apiKey`. To change the maximum allowed number of events for other `apiKeys`, use YMMYandexMetricaConfiguration.maxReportsInDatabaseCount.

sessionTimeout

```
(nonatomic, assign, readonly) NSUInteger sessionTimeout
```

Session timeout in seconds.

The default value is `10` (minimum allowed value).

statisticsSending

```
(nonatomic, assign, readonly) BOOL statisticsSending
```

A flag indicating that sending statistics is enabled.

The default value is `YES`.

Possible values:

- `YES` — Sending statistics is enabled.
- `NO` — Sending statistics is disabled.

userProfileID

`(nonatomic, copy, nullable) NSString *userProfileID`

Sets the ID of the user profile (`ProfileID`) when activated.

Attention: The maximum length of the `ProfileID` string is 200 characters.

YMMRevenueInfo class

The class contains immutable information about the revenue from in-app purchases.

Use the YMMMutableRevenueInfo class to change information about revenue.

The instance of the `YMMRevenueInfo` class should be sent to the AppMetrica server using the `reportRevenue` method of the `YMMYandexMetrica` class.

Instance methods

-initWithPrice:currency:	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.  **Attention:** Deprecated method. Use the -initWithPriceDecimal:currency: method instead.
-initWithPriceDecimal:currency:	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.
-initWithPrice:currency:quantity:productID:transactionID:receiptData:payload:	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.  **Attention:** Deprecated method. Use the #initWithPriceDecimal:currency:quantity:productID:transactionID: method instead.
-initWithPriceDecimal:currency:quantity:productID:transactionID:receiptData:payload:	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Properties

currency	Currency code of the purchase in the ISO 4217 format.
payload	Additional information to be passed about the purchase.
price	Price. It can be negative, e.g. for refunds.  **Attention:** The property is deprecated. Use the priceDecimal property instead.
priceDecimal	The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.
productID	ID of the product purchased. The value can contain up to 200 characters.
quantity	Quantity of products purchased.
receiptData	Details about the in-app purchase order from App Store.
transactionID	Details about the in-app purchase order from App Store.

Method descriptions

-initWithPrice:currency:

```
- (instancetype)initWithPrice:(double)price currency:(NSString *)currency
```

Attention: Deprecated method. Use the -initWithPriceDecimal:currency: method instead.

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

price	Price. It can be negative, e.g. for refunds.
currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: `RUB`. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.

Returns:

The instance of the `YMMRevenueInfo` class.

-initWithPriceDecimal:currency:

```
- (instancetype)initWithPriceDecimal:(NSDecimalNumber *)priceDecimal currency:(NSString *)currency
```

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

priceDecimal	The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.
currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: `RUB`. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.

Returns:

The instance of the `YMMRevenueInfo` class.

-initWithPrice:currency:quantity:productID:transactionID:receiptData:payload:

```
- (instancetype)initWithPrice:(double)price
                     currency:(NSString *)currency
                     quantity:(NSUInteger)quantity
                    productID:(NSString *)productID
                transactionID:(NSString *)transactionID
                  receiptData:(NSData *)receiptData
                      payload:(NSDictionary *)payload
```

Attention: Deprecated method. Use the #initWithPriceDecimal:currency:quantity:productID:transactionID:receiptData:payload: method instead.

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

price	Price. It can be negative, e.g. for refunds.
currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: `RUB`. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.
quantity	Quantity of products purchased. It is used in the following formula: `Revenue = quantity * price.` **Note:** The value cannot be negative. If the value is equal to 0, the purchase is ignored.
productID	ID of the product purchased. The value can contain up to 200 characters.
transactionID	Details about the in-app purchase order from App Store.
receiptData	Details about the in-app purchase order from App Store.
payload	Additional information to be passed about the purchase. For instance, it can be used for categorizing your products. It should contain the `NSDictionary` object that can be converted to valid JSON. The maximum size of the value is 30 KB.

Returns:

The instance of the `YMMRevenueInfo` class.

#initWithPriceDecimal:currency:quantity:productID:transactionID:receiptData:payload:

```
- (instancetype)initWithPriceDecimal:(NSDecimalNumber *)priceDecimal
                            currency:(NSString *)currency
                            quantity:(NSUInteger)quantity
                           productID:(NSString *)productID
                       transactionID:(NSString *)transactionID
                         receiptData:(NSData *)receiptData
                             payload:(NSDictionary *)payload
```

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

priceDecimal	The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.

currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: RUB. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.
quantity	Quantity of products purchased. It is used in the following formula: `Revenue = quantity * price.` **Note:** The value cannot be negative. If the value is equal to 0, the purchase is ignored.
productID	ID of the product purchased. The value can contain up to 200 characters.
transactionID	Details about the in-app purchase order from App Store.
receiptData	Details about the in-app purchase order from App Store.
payload	Additional information to be passed about the purchase. For instance, it can be used for categorizing your products. It should contain the `NSDictionary` object that can be converted to valid JSON. The maximum size of the value is 30 KB.

Returns:

The instance of the `YMMRevenueInfo` class.

Property descriptions

currency

```
(nonatomic, copy, readonly) NSString *currency
```

Currency code of the purchase in the ISO 4217 format.

payload

```
(nonatomic, copy, readonly) NSDictionary *payload
```

Additional information to be passed about the purchase.

price

```
(nonatomic, strong, nullable, readonly) double price
```

Attention: The property is deprecated. Use the priceDecimal property instead.

Price. It can be negative, e.g. for refunds.

priceDecimal

`(nonatomic, assign, readonly) NSDecimalNumber *priceDecimal`

The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.

productID

`(nonatomic, copy, readonly) NSString *productID`

ID of the product purchased. The value can contain up to 200 characters.

quantity

`(nonatomic, assign, readonly) NSUInteger quantity`

Quantity of products purchased.

receiptData

`(nonatomic, copy, readonly) NSData *receiptData`

Details about the in-app purchase order from App Store.

transactionID

`(nonatomic, copy, reaodnly) NSString *transactionID`

Details about the in-app purchase order from App Store.

YMMUserProfile class

Immutable class for storing the user profile.

Use the YMMMutableUserProfile class to change the user profile.

A user profile is a set of user attributes. User profile details are displayed in the AppMetrica report on user profiles.

The `YMMUserProfile` instance should be passed to the AppMetrica server by using the `reportUserProfile` method of the YMMYandexMetrica class. Use methods of the YMMProfileAttribute class to create user attributes.

Instance methods

-initWithUpdates:	Initializes the user profile with the specified set of updates.

Properties

updates	Array that contains attribute updates.

Method descriptions

-initWithUpdates:

```
- (instancetype)initWithUpdates:(NSArray<YMMUserProfileUpdate *> *)updates
```

Initializes the user profile with the specified set of updates.

Parameters:

updates	Array that contains attribute updates.

Returns:

The instance of the `YMMUserProfile` class.

Property descriptions

updates

`(nonatomic, copy, readonly) NSArray<YMMUserProfileUpdate *> *updates`

Array that contains attribute updates.

YMMYandexMetrica class

Methods of the class are used for configuring the library.

Instance methods

+activateWithConfiguration:	Initializes the library in an application with the extended startup configuration.
+activateReporterWithConfiguration:	Initializes a reporter with extended configuration.
+handleOpenURL:	Processes the URL that opened the application.
+initWebViewReporting:	Adds a JavaScript interface with the AppMetrica name in a window to the specified webview. This lets you send client events from JavaScript code.
+libraryVersion	Returns the current version of the AppMetrica library.
+pauseSession:	Pauses the user session.
-reporterForApiKey:	Creates a reporter for sending events to an additional API key.
+reportAdRevenue:onFailure:	Sends information about advertising revenue to the AppMetrica server.
+reportECommerce:onFailure:	Sends a message about an E-commerce event.
+reportError:exception:onFailure:	Sends a custom error message. **Attention:** Deprecated method. Use the +reportError:options:onFailure: or +reportError:onFailure: method instead.
+reportError:onFailure:	Sends a `YMMErrorRepresentable` message.
+reportError:options:onFailure:	Sends a `YMMErrorRepresentable` message.
+reportNSError:onFailure:	Sends an `NSError` message.
+reportNSError:options:onFailure:	Sends an `NSError` message.
+reportEvent:params:onFailure:	Sends an event message with additional parameters.
+reportEvent:onFailure	Sends an event message.
+reportReferralUrl:	Sets referral URL for app installs. This method can be used to track some traffic sources.
+reportRevenue:onFailure:	Sends information about the purchase to the AppMetrica server.
+reportUserProfile:onFailure:	Sends information about the user profile update to the AppMetrica server.
+requestAppMetricaDeviceIDWithCompletionQueue:	Requests the unique AppMetrica ID (`deviceID`).

+resumeSession:	Resumes the session, or creates a new one if the session timeout has expired.
+sendEventsBuffer:	Sends stored events from the buffer.
+setErrorEnvironmentValue:forKey:	Sets the key-value pair associated with crashes and errors.
+setLocation:	Sets custom location of the device.
+setLocationTracking:	Enables/disables sending location of the device .
+setStatisticsSending:	Enables/disables sending statistics to the AppMetrica server.
+setUserProfileID:	Sets the ID of the user profile. AppMetrica doesn't display predefined attributes in the web interface if `ProfieId` sending isn't configured.

Method descriptions

+activateWithConfiguration:

```
+ (void)activateWithConfiguration:(YMMYandexMetricaConfiguration *)configuration
```

Initializes the library in an application with the extended startup configuration.

Parameters:

configuration	The instance of the YMMYandexMetricaConfiguration class which contains the extended startup configuration for the library.

+activateReporterWithConfiguration:

```
+ (void)activateReporterWithConfiguration:(YMMReporterConfiguration *)configuration
```

Initializes a reporter with extended configuration.

The configuration of the reporter should be initialized before the first call to the reporter. Otherwise, the configuration of the reporter is ignored.

The reporter should be activated with the configuration using a different API key instead of the app's API key.

Parameters:

configuration	The instance of the YMMReporterConfiguration class which contains the extended configuration for the reporter.

+handleOpenURL:

```
+ (BOOL)handleOpenURL:(NSURL *)url
```

Processes the URL that opened the application.

Used for tracking app openings via deeplink

Parameters:

url	URL that opened the application.

Returns:

- YES, if the deeplink is intended for AppMetrica.
- N0if the deeplink is not intended for AppMetrica.

There are no such deeplink at the moment. The method always returns N0.

+initWebViewReporting:

```
+ (void)initWebViewReporting:(WKUserContentController *)userContentController
                   onFailure:(void (^)(NSError *))onFailure
```

Adds a JavaScript interface with the AppMetrica name in a window to the specified webview. This lets you send client events from JavaScript code.

Notes:

- The method must be called from the main queue.
- The method is not available on tvOS.
- Call this method before loading any content. We recommend calling this method before creating a webview and before adding your scripts to WKUserContentController.

For more information, see Usage examples.

Parameters:

userContentController	The WKUserContentController object used for this WKWebView.
onFailure	A callback method to be called in the event of an error.

+libraryVersion:

```
+ (NSString *)libraryVersion
```

Returns the current version of the AppMetrica library.

Returns:

Library version.

+pauseSession:

```
+ (void)resumeSession
```

Pauses the user session.

Note: The session duration depends on specified timeout. If the time interval between pausing and resuming the session is less than the specified timeout, the current session will be resumed; otherwise, a new one will be created.

For more information about sessions, see Tracking user activity.

-reporterForApiKey:

```
- (nullable id<YMMYandexMetricaReporting>)reporterForApiKey:(NSString *)apiKey
```

Creates a reporter for sending events to an additional API key.

To initialize a reporter with the extended configuration, use the activateReporterWithConfiguration: method. The configuration of the reporter should be initialized before the first call to the reporter. Otherwise, the configuration of the reporter is ignored.

Parameters:

apiKey	API key that differs from the main application API key.

Returns:

The instance that implements the `YMMYandexMetricaReporting` protocol for a specified API key.

+reportAdRevenue:onFailure:

```
+ (void)reportAdRevenue:(YMMAdRevenueInfo *)adRevenue
             onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends information about advertising revenue to the AppMetrica server.

Parameters:

adRevenue	The instance of the YMMAdRevenueInfo class which contains information about advertising revenue.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportECommerce:onFailure:

```
+ (void)reportECommerce:(YMMECommerce *)eCommerce
             onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends a message about an E-commerce event.

Parameters:

eCommerce	The YMMECommerce class instance.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportError:exception:onFailure:

```
+ (void)reportError:(NSString *)message
          exception:(NSException *)exception
          onFailure:(void (^)(NSError *error))onFailure
```

Sends a custom error message.

Attention: Deprecated method. Use the +reportError:options:onFailure: or +reportError:onFailure: method instead.

Parameters:

message	Short name or description of the error.
exception	The instance of the NSException class.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportError:onFailure:

```
+ (void)reportError:id<YMMErrorRepresentable>
          onFailure:(void (^)(NSError *error))onFailure
```

Sends a `YMMErrorRepresentable` message.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportError:options:onFailure:

```
+ (void)reportError:(id<YMMErrorRepresentable>)error
            options:(YMMErrorReportingOptions)options
          onFailure:(void (^)(NSError *error))onFailure
```

Sends a `YMMErrorRepresentable` message.

Use this method to set the send parameters.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportNSError:onFailure:

```
+ (void)reportNSError:(NSError *)error
            onFailure:(void (^)(NSError *error))onFailure
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportNSError:options:onFailure:

```
+ (void)reportNSError:(NSError *)error
              options:(YMMErrorReportingOptions)options
            onFailure:(void (^)(NSError *error))onFailure
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Use this method to set the send parameters.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportEvent:onFailure:

```
+ (void)reportEvent:(NSString *)message onFailure:(void (^)(NSError *error))onFailure
```

Sends an event message.

Parameters:

message	Short name or description of the event
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportEvent:params:onFailure:

```
+ (void)reportEvent:(NSString *)message
         parameters:(NSDictionary *)params
          onFailure:(void (^)(NSError *error))onFailure
```

Sends an event message with additional parameters.

Parameters:

message	Short name or description of the event
params	Parameters as “key-value” pairs.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportReferralUrl:

```
+ (void)reportReferralUrl:(NSURL *)url
```

Sets referral URL for app installs. This method can be used to track some traffic sources.

Parameters:

url	Referral URL of the app install.

+reportRevenue:onFailure:

```
+ (void)reportRevenue:(YMMRevenueInfo *)revenueInfo
            onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends information about the purchase to the AppMetrica server.

Parameters:

revenueInfo	The instance of the YMMRevenueInfo class which contains information about a purchase.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportUserProfile:onFailure:

```
+ (void)reportUserProfile:(YMMUserProfile *)userProfile
                onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends information about the user profile update to the AppMetrica server.

Parameters:

userProfile	The instance of the YMMUserProfileclass which contains information about the user profile.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+requestAppMetricaDeviceIDWithCompletionQueue:block:

```
+ (void)requestAppMetricaDeviceIDWithCompletionQueue:(nullable dispatch_queue_t)queue
                                     completionBlock:(YMMAppMetricaDeviceIDRetrievingBlock)block
```

Requests the unique AppMetrica ID (.

`deviceID`

).

In the Logs API and Post API `deviceID` referred to as `appmetrica_device_id`.

Parameters:

queue	The queue where the callback block will be called.
block	Callback block for receiving appmetrica_device_id. Includes an ID `appMetricaDeviceID` and error `error`, if the ID could not be obtained.

+resumeSession:

```
+ (void)resumeSession
```

Resumes the session, or creates a new one if the session timeout has expired.

Note: The session duration depends on specified timeout. If the time interval between pausing and resuming the session is less than the specified timeout, the current session will be resumed; otherwise, a new one will be created.

For more information about sessions, see Tracking user activity.

+sendEventsBuffer:

```
+ (void)sendEventsBuffer
```

Sends stored events from the buffer.

AppMetrica SDK does not send an event immediately after it occurred. The library stores event data in the buffer. Method

`sendEventsBuffer()`

sends data from the buffer and flushes it. Use the method to force sending stored events after passing important checkpoints of user scenarios.

Attention:

Frequent use of the method can lead to increased outgoing internet traffic and energy consumption.

+setErrorEnvironmentValue:forKey:

```
+ (void)setErrorEnvironmentValue:(nullable NSString *)value forKey:(NSString *)key;
```

Sets the key-value pair associated with crashes and errors.

Note: AppMetrica uses these values as additional information for further unhandled exceptions. If you pass the `nil` value, AppMetrica deletes the previous pair.

Parameters:

value	Value
key	Key

+setLocation:

```
+ (void)setLocation:(CLLocation *)location
```

Sets custom location of the device.

Parameters:

location	Information about the location of the device.

+setLocationTracking:

```
+ (void)setLocationTracking:(BOOL)enabled
```

Enables/disables sending location of the device .

Parameters:

enabled	A flag indicating if sending information about the device location is enabled. The default value is YES. Possible values: • YES — Sending information about the device location is enabled. • NO — Sending information about the location of the device is turned off.

+setStatisticsSending:

```
+ (void)setStatisticsSending:(BOOL)enabled
```

Enables/disables sending statistics to the AppMetrica server.

For more information about using the method, see Disable and enable sending statistics.

Note:

Disabling sending also turns off sending data from all reporters that initialized with the other API key.

Parameters:

enabled	A flag indicating that sending statistics is enabled. The default value is YES. Possible values: • YES — Sending statistics is enabled. • NO — Sending statistics is disabled.

+setUserProfileID:

```
+ (void)setUserProfileID:(nullable NSString *)userProfileID
```

Sets the ID of the user profile. If the

`ProfileId`

isn't sent, predefined attributes aren't displayed in the web interface.

Parameters:

userProfileID	User profile ID.

YMMYandexMetricaConfiguration class

This class contains the extended startup configuration for the library.

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

Instance methods

-initWithApiKey:	Initializes the instance of the `YMMYandexMetricaConfiguration` class with the specified API key.

Properties

apiKey	The API key of the application.
appForKids	Defines the application type as “children's” to match the rules for checking children's apps. If this option is enabled, the AppMetrica SDK doesn't send advertising IDs or location information. **Note:** Use this property if you have an app for kids.
appOpenTrackingEnabled	Enables/disables the automatic collection and sending of data about app launches via a deeplink.
appVersion	App version.
crashReporting	Enables/disables collecting and sending information about app crashes.
handleActivationAsSessionStart	Defines the AppMetrica SDK initialization as the beginning of a user session.
handleFirstActivationAsUpdate	Defines the first launch of the app as an update.
maxReportsInDatabaseCount	The maximum number of error reports stored in the internal DB.
location	Sets custom location of the device.

locationTracking	Enables/disables sending location of the device.
logs	Enables/disables logging the activity of the library.
preloadInfo	Sets the instance of the YMMYandexMetricaPreloadInfo class for tracking pre-installed apps.
revenueAutoTrackingEnabled	Enables/disables the automatic collection of information about In-App purchases.
sessionsAutoTracking	Enables/disables automatic tracking of the application lifecycle.
sessionTimeout	Sets the session timeout in seconds.
statisticsSending	Enables/disables sending statistics to the AppMetrica server.
userProfileID	Sets the ID of the user profile (`ProfileID`) when activated.

Method descriptions

-initWithApiKey:

```
- (instancetype)initWithApiKey:(NSString *)apiKey
```

Initializes the instance of the `YMMYandexMetricaConfiguration` class with the specified API key.

Parameters:

apiKey	The API key of the application.

Returns:

The instance of the `YMMYandexMetricaConfiguration` class.

Property descriptions

apiKey

`(nonatomic, copy, readonly) NSString *apiKey`

The API key of the application.

appForKids

`(nonatomic, assign) BOOL appForKids`

Defines the application type as “children's” to match the rules for checking children's apps. If this option is enabled, the AppMetrica SDK doesn't send advertising IDs or location information.

Note: Use this property if you have an app for kids.

appOpenTrackingEnabled

`(nonatomic, assign) BOOL appOpenTrackingEnabled`

Enables/disables the automatic collection and sending of data about app launches via a deeplink.

Attention: Automatic tracking captures only those deeplinks that resulted in app launches. To track the deeplinks inside the running application, also set up tracking.

The option is enabled by default.

Possible values:

- `YES` — Automatic collection and sending of data about the app launch via a deeplink is enabled.
- `NO` — Automatic collection and sending of data about the app launch via a deeplink is disabled.

appVersion

`(nonatomic, copy) NSString *appVersion`

App version.

crashReporting

`(nonatomic, assign) BOOL crashReporting`

Enables/disables collecting and sending information about app crashes.

Possible values:

- `YES` — Sending information about crashes is enabled.
- `NO` — Sending information about crashes is disabled.

handleActivationAsSessionStart

`(nonatomic, assign) BOOL handleActivationAsSessionStart`

Defines the AppMetrica SDK initialization as the beginning of a user session.

This option is disabled by default.

Possible values:

- `YES` — The user session is created when the library is initialized.
- `NO` — A background session is created when the library initializes, and a user session is created after a UIApplicationDidBecomeActiveNotification system event.

handleFirstActivationAsUpdate

`(nonatomic, assign) BOOL handleFirstActivationAsUpdate`

Defines the first launch of the app as an update.

Note:

If the first launch of the app is defined as an update, the installation is not shown as a new installation in reports, and is not attributed to partners.

Possible values:

- `YES` — The first launch is defined as an update.
- `NO` — The first launch is defined as a new installation.

maxReportsInDatabaseCount

`@property (assign, readwrite, nonatomic) NSUInteger maxReportsInDatabaseCount;`

The maximum number of error reports stored in the internal DB.

The allowed range of values is [100; 10000]. Values outside this range are automatically replaced with values from the nearest range limits.

Default value: 1000.

Note: Separate databases are used for various `apiKeys` and independent limits on the number of events can be set for them. This parameter only affects the limitation for the corresponding `apiKey`. To change the maximum allowed number of events for other `apiKeys`, use YMMReporterConfiguration.maxReportsInDatabaseCount.

location

`(nonatomic, strong, nullable) CLLocation *location`

Sets custom location of the device.

locationTracking

`(nonatomic, assign) BOOL locationTracking`

Enables/disables sending location of the device.

By default, sending is enabled.

logs

`(nonatomic, assign) BOOL logs`

Enables/disables logging the activity of the library.

Logging is disabled by default.

preloadInfo

`(nonatomic, copy) YMMYandexMetricaPreloadInfo *preloadInfo`

Sets the instance of the YMMYandexMetricaPreloadInfo class for tracking pre-installed apps.

For more information, see Tracking pre-installed apps.

revenueAutoTrackingEnabled

`(nonatomic, assign) BOOL revenueAutoTrackingEnabled`

Enables/disables the automatic collection of information about In-App purchases.

The option is enabled by default.

Possible values:

- `YES` — Automatic collection and sending of information about In-App purchases is enabled.
- `NO` — Automatic collection and sending of information about In-App purchases is disabled.

sessionsAutoTracking

`(nonatomic, assign) BOOL sessionsAutoTracking`

Enables/disables automatic tracking of the application lifecycle.

The option is enabled by default.

If the option is disabled, you should manually set up session control using the methods +pauseSession: and +resumeSession:. For more information, see Manual session tracking.

AppMetrica uses UIApplicationDidBecomeActiveNotification and UIApplicationWillResignActiveNotification to track sessions. The maximum session length is 24 hours. To extend the session after 24 hours, invoke the +resumeSession: method manually.

sessionTimeout

`(nonatomic, assign) NSUInteger sessionTimeout`

Sets the session timeout in seconds.

The default value is `10` (minimum allowed value).

For more information about sessions, see Tracking user activity.

statisticsSending

`(nonatomic, assign) BOOL statisticsSending`

Enables/disables sending statistics to the AppMetrica server.

Note: Disabling sending also turns off sending data from all reporters that initialized with the other `apiKey`.

userProfileID

`(nonatomic, copy, nullable) NSString *userProfileID`

Sets the ID of the user profile (`ProfileID`) when activated.

Attention: The maximum length of the `ProfileID` string is 200 characters.

YMMYandexMetricaPreloadInfo class

This class contains information for tracking pre-installed apps.

Instance methods

-initWithTrackingIdentifier:	Initializes the instance of the `YMMYandexMetricaPreloadInfo` class with the specified `trackingID`.
-setAdditionalInfo:key:	Returns “key-value” additional parameters that are used for tracking pre-installed apps.

Method descriptions

-initWithTrackingIdentifier:

```
- (instancetype)initWithTrackingIdentifier:(NSString *)trackingID
```

Initializes the instance of the `YMMYandexMetricaPreloadInfo` class with the specified `trackingID`.

Parameters:

trackingID	Tracking ID for tracking pre-installed apps.

Returns:

The instance of the `YMMYandexMetricaPreloadInfo` class.

-setAdditionalInfo:key:

```
- (void)setAdditionalInfo:(NSString *)info forKey:(NSString *)key
```

Returns “key-value” additional parameters that are used for tracking pre-installed apps.

This method may be invoked repeatedly for setting multiple pairs of additional information

Parameters:

info	Value of the additional parameter. Can't be `nil`. Pairs with the `nil` value are ignored.
key	Key for the additional parameter. Can't be `nil`. Pairs with the `nil` key are ignored.

Protocols

YMMBirthDateAttribute protocol

The protocol defines methods for updating the age or date of birth of a user profile.

Instance methods

-withAge:	Updates the attribute value.
-withYear:	Updates the attribute value.
-withYear:month:	Updates the attribute value.

-withYear:month:day:	Updates the attribute value.
-withDateComponents:	Updates the attribute value.
-withValueReset	Resets the attribute value.

Method descriptions

-withAge:

```
- (YMMUserProfileUpdate *)withAge:(NSUInteger)value
```

Updates the attribute value.

Parameters:

value	Age.

Returns:

The `YMMUserProfileUpdate` class instance.

-withYear:

```
- (YMMUserProfileUpdate *)withYear:(NSUInteger)year
```

Updates the attribute value.

Parameters:

year	Year of birth.

Returns:

The `YMMUserProfileUpdate` class instance.

-withYear:month:

```
- (YMMUserProfileUpdate *)withYear:(NSUInteger)year month:(NSUInteger)month
```

Updates the attribute value.

Parameters:

year	Year of birth.
month	Month of birth.

Returns:

The `YMMUserProfileUpdate` class instance.

-withYear:month:day:

```
- (YMMUserProfileUpdate *)withYear:(NSUInteger)year
                             month:(NSUInteger)month
                               day:(NSUInteger)day
```

Updates the attribute value.

Parameters:

year	Year of birth.
month	Month of birth.

day	Day of birth.

Returns:

The `YMMUserProfileUpdate` class instance.

-withDateComponents:

```
- (YMMUserProfileUpdate *)withDateComponents:(NSDateComponents *)dateComponents
```

Updates the attribute value.

Parameters:

dateComponents	The instance of the `NSDateComponents` class.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomBoolAttribute protocol

The protocol defines methods for updating the value of a boolean attribute.

Instance methods

-withValue:	Updates the attribute value.
-withValueIfUndefined:	Updates the attribute value if it wasn't set earlier.
-withValueReset	Resets the attribute value.

Method descriptions

-withValue:

```
- (YMMUserProfileUpdate *)withValue:(BOOL)value
```

Updates the attribute value.

Parameters:

value	Attribute value: `YES` or `NO`

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueIfUndefined:

```
- (YMMUserProfileUpdate *)withValueIfUndefined:(BOOL)value
```

Updates the attribute value if it wasn't set earlier.

Parameters:

value	Attribute value: `YES` or `NO`

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomCounterAttribute protocol

The protocol defines methods for updating the value of the tag type attribute.

Instance methods

-withDelta:	Updates the attribute value with the specified delta value.

Method descriptions

-withDelta:

```
- (YMMUserProfileUpdate *)withDelta:(double)value
```

Updates the attribute value with the specified delta value.

Parameters:

value	The delta value which you want to add or remove to the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomNumberAttribute protocol

The protocol defines methods for updating the value of a numeric attribute.

Instance methods

-withValue:	Updates the attribute value.
-withValueIfUndefined:	Updates the attribute value if it wasn't set earlier.
-withValueReset	Resets the attribute value.

Method descriptions

-withValue:

```
- (YMMUserProfileUpdate *)withValue:(double)value
```

Updates the attribute value.

Parameters:

value	The value of the numeric attribute. The data type is `double`.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueIfUndefined:

```
- (YMMUserProfileUpdate *)withValueIfUndefined:(double)value
```

Updates the attribute value if it wasn't set earlier.

Parameters:

value	The value of the numeric attribute. The data type is `double`.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomStringAttribute protocol

The protocol defines methods for updating the value of a string attribute.

Instance methods

-withValue:	Updates the attribute value.
-withValueIfUndefined:	Updates the attribute value if it wasn't set earlier.
-withValueReset	Resets the attribute value.

Method descriptions

-withValue:

```
- (YMMUserProfileUpdate *)withValue:(nullable NSString *)value
```

Updates the attribute value.

Parameters:

value	Updates the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueIfUndefined:

```
- (YMMUserProfileUpdate *)withValueIfUndefined:(nullable NSString *)value
```

Updates the attribute value if it wasn't set earlier.

Parameters:

value	Attribute value as a string. The maximum value length is 200 characters.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMErrorRepresentable protocol

Protocol for errors that can be sent in AppMetrica.

Each class instance that implements the protocol must have an ID. AppMetrica uses IDs to group errors.

All error information that is sent is available in AppMetrica reports.

You can implement this protocol to send your own errors. You can also use the default YMMError implementation.

Properties

identifier	Error ID. AppMetrica uses it to group errors.
message	Arbitrary error description.
parameters	Additional error parameters.
backtrace	Custom error backtrace. You can get it using the `NSThread.callStackReturnAddresses` method.
underlyingError	Error instance that matches the `YMMErrorRepresentable` protocol.

Property descriptions

identifier

```
@required
@property (readonly, copy, nonatomic) NSString *_Nonnull identifier;
```

Error ID. AppMetrica uses it to group errors.

The maximum value length is 300 characters. If the value exceeds the limit, AppMetrica truncates it.

Note: AppMetrica doesn't use nested error (underlyingError) IDs for grouping.

message

```
@optional
@property (readonly, copy, nonatomic, nullable) NSString *message;
```

Arbitrary error description.

The maximum value length is 1000 characters. If the value exceeds the limit, AppMetrica truncates it.

parameters

```
@optional
@property (readonly, copy, nonatomic, nullable) NSDictionary<NSString *, id> *parameters;
```

Additional error parameters.

Parameters are represented as key-value pairs, where the key and value are strings. If the key or value differs from the string type, AppMetrica calls the `description` method automatically.

The maximum number of parameters is 50. The maximum allowed parameter size is 100 characters for the key and 2000 characters for the value. If the value exceeds the limit, AppMetrica truncates it.

backtrace

```
@optional
@property (readonly, copy, nonatomic, nullable) NSArray<NSNumber *> *backtrace;
```

Custom error backtrace. You can get it using the `NSThread.callStackReturnAddresses` method.

The maximum number of stack frames is 200. If the value exceeds the limit, AppMetrica truncates it.

underlyingError

```
@optional
@property (readonly, strong, nonatomic, nullable) id<YMMErrorRepresentable> underlyingError;
```

Error instance that matches the `YMMErrorRepresentable` protocol.

The maximum number of nested errors is 10. If the value exceeds the limit, AppMetrica truncates it.

YMMGenderAttribute protocol

The protocol defines methods for updating the gender of a user profile.

Instance methods

-withValue:	Updates the gender attribute with the specified value from the `YMMGenderType` enum.
-withValueReset	Resets the attribute value.

Method descriptions

-withValue:

```
- (YMMUserProfileUpdate *)withValue:(YMMGenderType)value
```

Updates the gender attribute with the specified value from the `YMMGenderType` enum.

Parameters:

value	The value from the `YMMGenderType` enum.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMNameAttribute protocol

The protocol defines methods for updating the name of a user profile.

Instance methods

-withValue:	Updates the attribute value.
-withValueReset	Resets the attribute value.

Method descriptions

-withValue:

```
- (YMMUserProfileUpdate *)withValue:(nullable NSString *)value
```

Updates the attribute value.

Parameters:

value	The name of the user profile. The maximum length of the user profile name is 100 characters.

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMNotificationsEnabledAttribute protocol

The protocol defines methods for updating the notification status of a user profile.

It indicates whether the user has enabled notifications for the application.

Instance methods

-withValue:	Updates the attribute value.
-withValueReset	Resets the attribute value.

Method descriptions

-withValue:

```
- (YMMUserProfileUpdate *)withValue:(BOOL)value
```

Updates the attribute value.

Parameters:

value	Attribute value: `YES` or `NO`

Returns:

The `YMMUserProfileUpdate` class instance.

-withValueReset

```
- (YMMUserProfileUpdate *)withValueReset
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMYandexMetricaReporting protocol

Instance methods

-pauseSession	Pauses the user session.
-reportAdRevenue:onFailure:	Sends information about advertising revenue to the AppMetrica server.
+reportECommerce:onFailure:	Sends a message about an E-commerce event.
-reportError:exception:onFailure:	Sends a custom error message. **Attention:** Deprecated method.
+reportError:onFailure:	Sends a `YMMErrorRepresentable` message.
+reportError:options:onFailure:	Sends a `YMMErrorRepresentable` message.
+reportNSError:onFailure:	Sends an `NSError` message.
+reportNSError:options:onFailure:	Sends an `NSError` message.
-reportEvent:onFailure:	Sends a custom event message.
-reportEvent:params:onFailure:	Sends a custom event message with additional parameters.
-reportRevenue:onFailure:	Sends information about the purchase to the AppMetrica server.
-reportUserProfile:onFailure:	Sends information about the user profile update to the AppMetrica server.
-resumeSession	Resumes the session, or creates a new one if the session timeout has expired.
-setStatisticsSending:	Enables/disables sending statistics to the AppMetrica server.
-setUserProfileID:	Sets the ID of the user profile. AppMetrica doesn't display predefined attributes in the web interface if `ProfieId` sending isn't configured.

Method descriptions

-pauseSession

```
- (void)pauseSession
```

Pauses the user session.

-reportAdRevenue:onFailure:

```
- (void)reportAdRevenue:(YMMAdRevenueInfo *)adRevenue
             onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends information about advertising revenue to the AppMetrica server.

Parameters:

adRevenue	The instance of the YMMAdRevenueInfo class which contains information about advertising revenue.

onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportECommerce:onFailure:

```
+ (void)reportECommerce:(YMMECommerce *)eCommerce
             onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends a message about an E-commerce event.

Parameters:

eCommerce	The YMMECommerce class instance.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

-reportError:exception:onFailure:

```
- (void)reportError:(NSString *)name
         exception:(nullable NSException *)exception
         onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends a custom error message.

Attention: Deprecated method.

Parameters:

name	Short name or description of the error.
exception	The instance of the NSException class.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

-reportEvent:onFailure:

```
- (void)reportEvent:(NSString *)name
         onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends a custom event message.

Parameters:

name	Short name or description of the event
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportError:onFailure:

```
- (void)reportError:(nonnull id<YMMErrorRepresentable>)error
             onFailure:(void (^)(NSError *error))onFailure
```

Sends a `YMMErrorRepresentable` message.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.

onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportError:options:onFailure:

```
- (void)reportNSError:(nonnull id<YMMErrorRepresentable>)error
              options:(YMMErrorReportingOptions)options
              onFailure:(void (^)(NSError *error))onFailure
```

Sends a `YMMErrorRepresentable` message.

Use this method to set the send parameters.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportError:onFailure:

```
- (void)reportNSError:(NSError *)error
              onFailure:(void (^)(NSError *error))onFailure
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

+reportNSError:options:onFailure:

```
- (void)reportNSError:(NSError *)error
              options:(YMMErrorReportingOptions)options
              onFailure:(void (^)(NSError *error))onFailure
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Use this method to set the send parameters.

Limits for an `NSError`:

- 200 characters for the `domain`.

- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

-reportEvent:params:onFailure:

```
- (void)reportEvent:(NSString *)name
         parameters:(nullable NSDictionary *)params
          onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends a custom event message with additional parameters.

Parameters:

name	Short name or description of the event
params	Parameters as “key-value” pairs.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

-reportRevenue:onFailure:

```
- (void)reportRevenue:(YMMRevenueInfo *)revenueInfo
            onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends information about the purchase to the AppMetrica server.

Parameters:

revenueInfo	The instance of the YMMRevenueInfo class which contains information about a purchase.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

-reportUserProfile:onFailure:

```
- (void)reportUserProfile:(YMMUserProfile *)userProfile
                onFailure:(nullable void (^)(NSError *error))onFailure
```

Sends information about the user profile update to the AppMetrica server.

Parameters:

userProfile	The instance of the YMMUserProfileclass which contains information about the user profile.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

-resumeSession

```
- (void)resumeSession
```

Resumes the session, or creates a new one if the session timeout has expired.

-setStatisticsSending:

```
- (void)setStatisticsSending:(BOOL)enabled
```

Enables/disables sending statistics to the AppMetrica server.

Note:

Disable sending statistics to the reporter does not affect the sending of data from the main API key. But disabling data sending for the main API key stops sending statistics from all reporters.

Parameters:

enabled	A flag indicating that sending statistics is enabled. The default value is YES. Possible values: • YES — Sending statistics is enabled. • NO — Sending statistics is disabled.

-setUserProfileID:

```
-(void)setUserProfileID:(nullable NSString *)userProfileID
```

Sets the ID of the user profile. If the

`ProfileId`

isn't sent, predefined attributes aren't displayed in the web interface.

Parameters:

userProfileID	User profile ID.

Enumerations
YMMAdType

Contains values of advertising types (AdType).

Enumerations

YMMAdType

Enumeration descriptions

YMMAdType

```
typedef NS_ENUM(NSUInteger, YMMAdType) {
    YMMAdTypeUnknown = 0,
    YMMAdTypeNative = 1,
    YMMAdTypeBanner = 2,
    YMMAdTypeRewarded = 3,
    YMMAdTypeInterstitial = 4,
    YMMAdTypeMrec = 5,
    YMMAdTypeOther = 6,
};
```

YMMGenderType

Contains possible gender values for the `YMMGenderAttribute` class.

Enumerations

YMMGenderType

######## ############

YMMGenderType

`typedef NS_ENUM(NSUInteger, YMMGenderType)`

Constants	Description
YMMGenderTypeMale	Male.
YMMGenderTypeFemale	Female.
YMMGenderTypeOther	Other. For example, there is not enough info to detect the gender. **Note:** You can set the `YMMGenderTypeOther` as the attribute value and pass additional information using custom attributes.

YMMErrorReportingOptions

Contains parameters for sending errors.

Enumerations

YMMErrorReportingOptions

######## ############

YMMErrorReportingOptions

`YMMErrorReportingOptions`

Constants	Description
YMMErrorReportingOptionsNoBacktrace = 1 << 0	Option that doesn't attach the send call backtrace to the error. Can speed up sending reports. **Note:** This option doesn't affect the backtraces that are passed in the error itself.

Constants
YMMBacktraceErrorKey

```
extern const NSErrorUserInfoKey _Nonnull YMMBacktraceErrorKey
```

A key from the `userInfo` dictionary of an `NSError`. It should contain the error backtrace.
You can get it using the `NSThread.callStackReturnAddresses` (Objective-C) or `Thread.callStackReturnAddresses`(Swift) method.

AppMetrica automatically parses the passed value.

Swift

Classes

YMMAdRevenueInfo class

The class contains immutable information about advertising revenue (Ad Revenue).

Use the YMMMutableAdRevenueInfo class to change information about revenue.

The `YMMAdRevenueInfo` instance should be passed to the AppMetrica server using the reportAdRevenue method of the YMMYandexMetrica class.

Instance methods

init(adRevenue:currency:)	Initializes the instance of the `YMMAdRevenueInfo` class for sending information about advertising revenue.

Properties

adRevenue	The amount of money received from advertising revenue. Can't be negative.
currency	The currency in which adRevenue is represented. Must be in ISO-4217 format.
adType	Ad type. See the possible values in YMMAdType.
adNetwork	Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitID	Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitName	Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementID	Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementName	Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
precision	Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
payload	Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

Method descriptions

init(adRevenue:currency:)

```
init(adRevenue: NSDecimalNumber, currency: String)
```

Initializes the instance of the `YMMAdRevenueInfo` class for sending information about advertising revenue.

Parameters:

adRevenue	The amount of money received from advertising revenue. Can't be negative.
currency	The currency in which adRevenue is represented. Must be in ISO-4217 format.

Returns:

The `YMMAdRevenueInfo` class instance.

Property descriptions

adRevenue

```
var adRevenue: NSDecimalNumber { get }
```

The amount of money received from advertising revenue. Can't be negative.

currency

```
var currency: String { get }
```

The currency in which adRevenue is represented. Must be in ISO-4217 format.

adType

```
var adType: YMMAdType { get }
```

Ad type. See possible values in YMMAAdType.

adNetwork

```
var adNetwork: String { get }
```

Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitID

```
var adUnitID: String { get }
```

Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitName

```
var adUnitName: String { get }
```

Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementID

```
var adPlacementID: String { get }
```

Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementName

```
var adPlacementName: String { get }
```

Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

precision

```
var precision: String { get }
```

Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

payload

```
var payload: [String: String]? { get }
```

Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

YMMECommerce class

Methods of this class create a `YMMECommerce` instance.

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

Note: You can send the `YMMECommerce` instance using the report(eCommerce:onFailure:) method of the YMMYandexMetrica class and the YMMYandexMetricaReporting protocol.

Instance methods

showScreenEvent(screen:)	Creates an E-commerce event called `ShowScreenEvent`. Use it to report the opening of a page, such as a list of products, search or home page.
showProductCardEvent(product:screen:)	Creates an E-commerce event called `ShowProductCardEvent`. Use it to report viewing a product profile among others on the list.
showProductDetailsEvent(product:referrer:)	Creates an E-commerce event called `ShowProductDetailsEvent`. Use it to report viewing a product page.
addCartItemEvent(cartItem:)	Creates an E-commerce event called `AddCartItemEvent`. Use it to report adding an item to the cart.
removeCartItemEvent(cartItem:)	Creates an E-commerce event called `RemoveCartItemEvent`. Use it to report removing an item from the cart.
beginCheckoutEvent(order:)	Creates an E-commerce event called `BeginCheckoutEvent`. Use it to report starting a purchase.
purchaseEvent(order:)	Creates an E-commerce event called `PurchaseEvent`. Use it to report a completed purchase.

Method descriptions

showScreenEvent(screen:)

```
static func showScreenEvent(screen: YMMECommerceScreen) -> YMMECommerce
```

Creates an E-commerce event called `ShowScreenEvent`. Use it to report the opening of a page, such as a list of products, search or home page.

Parameters:

screen	The screen that was opened. The YMMECommerceScreen class instance.

Returns:

The `YMMECommerce` class instance.

showProductCardEvent(product:screen:)

```
static func showProductCardEvent(product: YMMECommerceProduct,  screen: YMMECommerceScreen) -> YMMECommerce
```

Creates an E-commerce event called `ShowProductCardEvent`. Use it to report viewing a product profile among others on the list.

Tip: Before sending the event, make sure that the product profile was shown on the screen for more than N seconds.

Parameters:

product	The product that was shown. The YMMECommerceProduct class instance.
screen	The screen where the product was displayed. The YMMECommerceScreen class instance.

Returns:

The `YMMECommerce` class instance.

showProductDetailsEvent(product:referrer:)

```
static func showProductDetailsEvent(product: YMMECommerceProduct, referrer: YMMECommerceReferrer?) -> YMMECommerce
```

Creates an E-commerce event called `ShowProductDetailsEvent`. Use it to report viewing a product page.

Parameters:

product	The product that was shown. The YMMECommerceProduct class instance.
referrer	Information about the source of traffic to the product page. The YMMECommerceReferrer class instance.

Returns:

The `YMMECommerce` class instance.

addCartItemEvent(cartItem:)

```
static func addCartItemEvent(cartItem: YMMECommerceCartItem) -> YMMECommerce
```

Creates an E-commerce event called `AddCartItemEvent`. Use it to report adding an item to the cart.

Parameters:

item	The item that was added to the cart. The YMMECommerceCartItem class instance.

Returns:

The `YMMECommerce` class instance.

removeCartItemEvent(cartItem:)

```
static func removeCartItemEvent(cartItem: YMMECommerceCartItem) -> YMMECommerce
```

Creates an E-commerce event called `RemoveCartItemEvent`. Use it to report removing an item from the cart.

Parameters:

item	The item that was removed from the cart. The YMMECommerceCartItem class instance.

Returns:

The `YMMECommerce` class instance.

beginCheckoutEvent(order:)

```
static func beginCheckoutEvent(order: YMMECommerceOrder) -> YMMECommerce
```

Creates an E-commerce event called `BeginCheckoutEvent`. Use it to report starting a purchase.

Parameters:

order	Information about the purchase. The YMMECommerceOrder class instance.

Returns:

The `YMMECommerce` class instance.

purchaseEvent(order:)

```
static func purchaseEvent(order: YMMECommerceOrder) -> YMMECommerce
```

Creates an E-commerce event called `PurchaseEvent`. Use it to report a completed purchase.

Parameters:

order	Information about the purchase. The YMMECommerceOrder class instance.

Returns:

The `YMMECommerce` class instance.

YMMECommerceAmount class

This class contains cost information, such as quantity and units.

Instance methods

init(unit:value:)	Initializes the instance of the `YMMECommerceAmount` class for sending information about purchases.

Properties

unit	The unit. For example: USD or RUB. Acceptable value: Up to 20 characters.
value	Quantity.

Method descriptions

init(unit:value:)

```
init(unit: String, value: NSDecimalNumber)
```

Initializes the instance of the `YMMECommerceAmount` class for sending information about purchases.

Parameters:

unit	The unit. For example: USD or RUB. Acceptable value: Up to 20 characters.
value	Quantity.

Returns:

The `YMMECommerceAmount` class instance.

Property descriptions

unit

```
var unit: String { get }
```

The unit. For example: USD or RUB. Acceptable value: Up to 20 characters.

Use ISO 4217 format.

value

```
var value: NSDecimalNumber { get }
```

Quantity.

YMMECommerceCartItem class

This class contains information about items added to the cart.

Instance methods

init(product:quantity:revenue:referrer:)	Initializes the instance of the `YMMECommerceCartItem` class with information about items added to the cart.

Properties

product	Product. The `YMMECommerceProduct` class instance.
quantity	Quantity.
revenue	The total price of the product in the cart. This price factors in the quantity, discounts to be applied, and so on. The `YMMECommercePrice` class instance.
referrer	The source of traffic to the cart. The `YMMECommerceReferrer` class instance.

Method descriptions

init(product:quantity:revenue:referrer:)

```
init(product: YMMECommerceProduct, quantity: NSDecimalNumber, revenue: YMMECommercePrice,
 referrer: YMMECommerceReferrer?)
```

Initializes the instance of the `YMMECommerceCartItem` class with information about items added to the cart.

Parameters:

product	Product. The `YMMECommerceProduct` class instance.
quantity	Quantity.

revenue	The total price of the product in the cart. This price factors in the quantity, discounts to be applied, and so on. The `YMMECommercePrice` class instance.
referrer	The source of traffic to the cart. The `YMMECommerceReferrer` class instance.

Returns:

The `YMMECommerceCartItem` class instance.

Property descriptions

product

```
var product: YMMECommerceProduct { get }
```

Product. The `YMMECommerceProduct` class instance.

quantity

```
var quantity: NSDecimalNumber { get }
```

Quantity.

revenue

```
var revenue: YMMECommercePrice { get }
```

The total price of the product in the cart. This price factors in the quantity, discounts to be applied, and so on. The `YMMECommercePrice` class instance.

referrer

```
var referrer: YMMECommerceReferrer? { get }
```

The source of traffic to the cart. The `YMMECommerceReferrer` class instance.

YMMECommerceOrder class

This class contains order information.

Instance methods

init(identifier:cartItems:)	Initializes the instance of the `YMMECommerceOrder` class with order information.
init(identifier:cartItems:payload:)	Initializes the instance of the `YMMECommerceOrder` class with order information.

Properties

identifier	Order ID. Allowed size: Up to 100 characters.
cartItems	A list of items in the cart.
payload	Additional information about the order. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Method descriptions

init(identifier:cartItems:)

```
init(identifier: String, cartItems: [YMMECommerceCartItem])
```

Initializes the instance of the `YMMECommerceOrder` class with order information.

Parameters:

identifier	Order ID. Allowed size: Up to 100 characters.
cartItems	A list of items in the cart.

Returns:

The `YMMECommerceOrder` class instance.

init(identifier:cartItems:payload:)

```
init(identifier: String, cartItems: [YMMECommerceCartItem], payload: [String: String]?)
```

Initializes the instance of the `YMMECommerceOrder` class with order information.

Parameters:

identifier	Order ID. Allowed size: Up to 100 characters.
cartItems	A list of items in the cart.
payload	Additional information about the order. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Returns:

The `YMMECommerceOrder` class instance.

Property descriptions

identifier

```
var identifier: String { get }
```

Order ID. Allowed size: Up to 100 characters.

cartItems

```
var cartItems: [YMMECommerceCartItem] { get }
```

A list of items in the cart.

payload

```
var payload: [String: String]? { get }
```

Additional information about the order. Acceptable sizes:

- The total `payload` size: Up to 20 KB.
- The `key` size: Up to 100 characters.

- The `value` size: Up to 1000 characters.

YMMECommercePrice class

This class contains the product price.

Instance methods

init(fiat:)	Initializes the instance of the `YMMECommercePrice` class.
init(fiat:internalComponents:)	Initializes the instance of the `YMMECommercePrice` class.

Properties

fiat	The cost in fiat money. The `YMMECommerceAmount` class instance.
internalComponents	The cost of internal components (the amounts in the internal currency). Allowed size: Up to 30 elements.

Method descriptions

init(fiat:)

```
init(fiat: YMMECommerceAmount)
```

Initializes the instance of the `YMMECommercePrice` class.

Parameters:

fiat	The cost in fiat money. The `YMMECommerceAmount` class instance.

Returns:

The `YMMECommercePrice` class instance.

init(fiat:internalComponents:)

```
init(fiat: YMMECommerceAmount, internalComponents: [YMMECommerceAmount]?)
```

Initializes the instance of the `YMMECommercePrice` class.

Parameters:

fiat	The cost in fiat money. The `YMMECommerceAmount` class instance.
internalComponents	The cost of internal components (the amounts in the internal currency). Allowed size: Up to 30 elements.

Returns:

The `YMMECommercePrice` class instance.

Property descriptions

fiat

```
var fiat: YMMECommerceAmount { get }
```

The cost in fiat money. The `YMMECommerceAmount` class instance.

internalComponents

```
var internalComponents: [YMMECommerceAmount]? { get }
```

The cost of internal components (the amounts in the internal currency). Allowed size: Up to 30 elements.

YMMECommerceProduct class

This class contains product information.

Instance methods

init(sku:)	Initializes the instance of the `YMMECommerceProduct` class with the specified item number.
init(sku:name:categoryComponents:payload:actualPrice:originalPrice:promoCodes:)	Initializes the instance of the `YMMECommerceProduct` class with all parameters.

Properties

sku	Item number. Allowed size: Up to 100 characters.
name	Name of the product. Allowed size: Up to 1000 characters.
categoryComponents	The path to the product by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.
payload	Additional information about the product. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.
actualPrice	The actual product price, which is the price after applying all discounts and promo codes.
originalPrice	The initial product price.
promoCodes	A list of promo codes that are applied to the product. Acceptable sizes: • Up to 20 elements. • The promo code length is up to 100 characters.

Method descriptions

init(sku:)

```
init(sku: String)
```

Initializes the instance of the `YMMECommerceProduct` class with the specified item number.

Parameters:

sku	Item number. Allowed size: Up to 100 characters.

Returns:

The `YMMECommerceProduct` class instance.

init(sku:name:categoryComponents:payload:actualPrice:originalPrice:promoCodes:)

```
init(sku: String, name: String?, categoryComponents: [String]?, payload: [String, String]?,
 actualPrice: YMMECommercePrice?, originalPrice: YMMECommercePrice?, promoCodes: [String]?)
```

Initializes the instance of the `YMMECommerceProduct` class with all parameters.

Parameters:

sku	Item number. Allowed size: Up to 100 characters.
name	Name of the product. Allowed size: Up to 1000 characters.
categoryComponents	The path to the product by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.
payload	Additional information about the product. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.
actualPrice	The actual product price, which is the price after applying all discounts and promo codes.
originalPrice	The initial product price.
promoCodes	A list of promo codes that are applied to the product. Acceptable sizes: • Up to 20 elements. • The promo code length is up to 100 characters.

Returns:

The `YMMECommerceProduct` class instance.

Property descriptions

sku

```
var sku: String { get }
```

Item number. Allowed size: Up to 100 characters.

name

```
var name: String? { get }
```

Name of the product. Allowed size: Up to 1000 characters.

categoryComponents

```
var categoryComponents: [String]? { get }
```

The path to the product by category. Acceptable sizes:

- Up to 10 elements.
- The size of a single element is up to 100 characters.

payload

```
var payload: [String : String]? { get }
```

Additional information about the product. Acceptable sizes:

- The total `payload` size: Up to 20 KB.
- The `key` size: Up to 100 characters.
- The `value` size: Up to 1000 characters.

actualPrice

```
var actualPrice: YMMECommercePrice? { get }
```

The actual product price, which is the price after applying all discounts and promo codes.

originalPrice

```
var originalPrice: YMMECommercePrice? { get }
```

The initial product price.

promoCodes

```
var promoCodes: [String]? { get }
```

A list of promo codes that are applied to the product. Acceptable sizes:

- Up to 20 elements.
- The promo code length is up to 100 characters.

YMMECommerceReferrer class

This class contains information about the source of traffic. For example, a link to the page or screen that shows a product profile.

Instance methods

init(type:identifier:screen:)	Initializes the instance of the `YMMECommerceReferrer` class with information about the source of traffic.

Properties

type	Traffic source type: The type of object that traffic comes from. For example: “button”, “banner”, or “href”. Allowed size: Up to 100 characters.
identifier	Traffic source ID. Allowed size: Up to 2048 characters.
screen	Traffic source screen: The screen that traffic comes from.

Method descriptions

init(type:identifier:screen:)

```
init(type: String?, identifier: String?, screen: YMMECommerceScreen?)
```

Initializes the instance of the `YMMECommerceReferrer` class with information about the source of traffic.

Parameters:

type	Traffic source type: The type of object that traffic comes from. For example: “button”, “banner”, or “href”. Allowed size: Up to 100 characters.
identifier	Traffic source ID. Allowed size: Up to 2048 characters.
screen	Traffic source screen: The screen that traffic comes from.

Returns:

The `YMMECommerceReferrer` class instance.

Property descriptions

type

```
var type: String? { get }
```

Traffic source type: The type of object that traffic comes from. For example: “button”, “banner”, or “href”. Allowed size: Up to 100 characters.

identifier

```
var identifier: String? { get }
```

Traffic source ID. Allowed size: Up to 2048 characters.

screen

```
var screen: YMMECommerceScreen? { get }
```

Traffic source screen: The screen that traffic comes from.

YMMECommerceScreen class

This class contains screen information.

Instance methods

init(name:)	Initializes the instance of the `YMMECommerceScreen` class with the specified screen name.
init(categoryComponents:)	Initializes the instance of the `YMMECommerceScreen` class with the specified path to the screen.
init(searchQuery:)	Initializes the instance of the `YMMECommerceScreen` class with the specified search query.
init(payload:)	Initializes the instance of the `YMMECommerceScreen` class with the specified additional information.
init(name:categoryComponents:searchQuery:payload:)	Initializes the instance of the `YMMECommerceScreen` class with all parameters.

Properties

name	Screen name. Allowed size: Up to 100 characters.
categoryComponents	The path to the screen by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.
searchQuery	Search query. Allowed size: Up to 1000 characters.
payload	Additional information about the screen. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Method descriptions

init(name:)

```
init(name: String)
```

Initializes the instance of the `YMMECommerceScreen` class with the specified screen name.

Parameters:

name	Screen name. Allowed size: Up to 100 characters.

Returns:

The `YMMECommerceScreen` class instance.

init(categoryComponents:)

```
init(categoryComponents: [String])
```

Initializes the instance of the `YMMECommerceScreen` class with the specified path to the screen.

Parameters:

categoryComponents	The path to the screen by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.

Returns:

The `YMMECommerceScreen` class instance.

init(searchQuery:)

```
init(searchQuery: String)
```

Initializes the instance of the `YMMECommerceScreen` class with the specified search query.

Parameters:

searchQuery	Search query. Allowed size: Up to 1000 characters.

Returns:

The `YMMECommerceScreen` class instance.

init(payload:)

```
init(payload: [String: String])
```

Initializes the instance of the `YMMECommerceScreen` class with the specified additional information.

Parameters:

payload	Additional information about the screen. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Returns:

The `YMMECommerceScreen` class instance.

init(name:categoryComponents:searchQuery:payload:)

```
init(name: String?, categoryComponents: [String]?, searchQuery: String?, payload: [String: String]?)
```

Initializes the instance of the `YMMECommerceScreen` class with all parameters.

Parameters:

name	Screen name. Allowed size: Up to 100 characters.
categoryComponents	The path to the screen by category. Acceptable sizes: • Up to 10 elements. • The size of a single element is up to 100 characters.
searchQuery	Search query. Allowed size: Up to 1000 characters.
payload	Additional information about the screen. Acceptable sizes: • The total `payload` size: Up to 20 KB. • The `key` size: Up to 100 characters. • The `value` size: Up to 1000 characters.

Returns:

The `YMMECommerceScreen` class instance.

Property descriptions

name

```
var name: String? { get }
```

Screen name. Allowed size: Up to 100 characters.

categoryComponents

```
var categoryComponents: [String]? { get }
```

The path to the screen by category. Acceptable sizes:

- Up to 10 elements.
- The size of a single element is up to 100 characters.

searchQuery

```
var searchQuery: String? { get }
```

Search query. Allowed size: Up to 1000 characters.

payload

```
var payload: [String: String]? { get }
```

Additional information about the screen. Acceptable sizes:

- The total `payload` size: Up to 20 KB.
- The `key` size: Up to 100 characters.
- The `value` size: Up to 1000 characters.

YMMError class

A simple implementation of the YMMErrorRepresentable protocol.

Instance methods

init(identifier:)	Creates the `YMMError` instance with the specified ID.
init(identifier:message:parameters:)	Creates the `YMMError` instance with the specified ID and other parameters.
init(identifier:message:parameters:backtrace:underlyingError:)	Creates the `YMMError` instance of the user with the specified ID and other parameters.

Method descriptions

init(identifier:)

```
convenience init(identifier: String)
```

Creates the `YMMError` instance with the specified ID.

Parameters:

identifier	Error ID. AppMetrica uses it to group errors.

Returns:

The `YMMError` class instance.

init(identifier:message:parameters:)

```
convenience init(identifier: String, message: String?, parameters: [String : Any]?)
```

Creates the `YMMError` instance with the specified ID and other parameters.

Parameters:

identifier	Error ID. AppMetrica uses it to group errors.
message	Arbitrary error description
parameters	Additional error parameters

Returns:

The `YMMError` class instance.

init(identifier:message:parameters:backtrace:underlyingError:)

```
convenience init(identifier: String, message: String?, parameters: [String : Any]?, backtrace: [NSNumber]?,
 underlyingError: YMMErrorRepresentable?)
```

Creates the `YMMError` instance of the user with the specified ID and other parameters.

Parameters:

identifier	Error ID. AppMetrica uses it to group errors.
message	Arbitrary error description
parameters	Additional error parameters
backtrace	Custom error backtrace
underlyingError	Error instance that matches the YMMErrorRepresentable protocol.

Returns:

The `YMMError` class instance.

YMMMutableAdRevenueInfo class

The mutable version of the YMMAdRevenueInfo class with information about advertising revenue.

The `YMMMutableAdRevenueInfo` instance should be passed to the AppMetrica server using the reportAdRevenue method of the YMMYandexMetrica class.

Properties

adType	Ad type. See possible values in YMMAAdType.
adNetwork	Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitID	Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adUnitName	Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementID	Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
adPlacementName	Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.
precision	Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

payload	Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

Property descriptions

adType

```
var adType: YMMAdType
```

Ad type. See possible values in YMMAAdType.

adNetwork

```
var adNetwork: String
```

Yandex Advertising Network. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitID

```
var adUnitID: String
```

Ad block ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adUnitName

```
var adUnitName: String
```

Ad block name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementID

```
var adPlacementID: String
```

Ad display location ID. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

adPlacementName

```
var adPlacementName: String
```

Ad display location name. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

precision

```
var precision: String
```

Accuracy. For example: “publisher_defined”, “estimated”. The maximum length is 100 characters. If the value exceeds this limit, it will be truncated by AppMetrica.

payload

```
var payload: [String: String]?
```

Accuracy. For example: “publisher_defined”, “estimated”. Arbitrary payload: Additional information presented as key-value pairs. The maximum size is 30 KB. If the value exceeds this limit, it will be truncated by AppMetrica.

YMMMutableReporterConfiguration class

The mutable version of the YMMReporterConfiguration class with the advanced configuration of the reporter.

Properties

logs	Enables/disables logging the activity of the library.
maxReportsInDatabaseCount	The maximum number of error reports stored in the internal DB.
sessionTimeout	Sets the session timeout in seconds.
statisticsSending	Enables/disables sending statistics to the AppMetrica server.
userProfileID	Sets the ID of the user profile (`ProfileID`) when activated.

Property descriptions

logs

```
var logs: Bool
```

Enables/disables logging the activity of the library.

The default value is `false`.

maxReportsInDatabaseCount

```
var maxReportsInDatabaseCount: UInt { get set }
```

The maximum number of error reports stored in the internal DB.

The allowed range of values is [100; 10000]. Values outside this range are automatically replaced with values from the nearest range limits.

Default value: 1000.

Note: Separate databases are used for various `apiKeys` and independent limits on the number of events can be set for them. This parameter only affects the limitation for the corresponding `apiKey`. To change the maximum allowed number of events for other `apiKeys`, use YMMYandexMetricaConfiguration.maxReportsInDatabaseCount.

sessionTimeout

```
var sessionTimeout: UInt
```

Sets the session timeout in seconds.

The default value is `10` (minimum allowed value).

statisticsSending

```
var statisticsSending: Bool
```

Enables/disables sending statistics to the AppMetrica server.

Note:

Disable sending statistics to the reporter does not affect the sending of data from the main API key. But disabling data sending for the main API key stops sending statistics from all reporters.

The default value is `true`.

userProfileID

```
var userProfileID: String?
```

Sets the ID of the user profile (`ProfileID`) when activated.

Attention: The maximum length of the `ProfileID` string is 200 characters.

YMMMutableRevenueInfo class

The mutable version of the YMMRevenueInfo class with information about purchases.

The instance of the `YMMRevenueInfo` class should be sent to the AppMetrica server using the `reportRevenue` method of the YMMYandexMetrica class.

Properties

payload	Additional information to be passed about the purchase. For instance, it can be used for categorizing your products.
productID	ID of the product purchased. The value can contain up to 200 characters.
quantity	Quantity of products purchased.
receiptData	Details about the in-app purchase order from App Store. This property is used for validating purchases in the app. For more information, see Apple documentation.
transactionID	Transaction ID transactionIdentifier from the SKPaymentTransaction class. This property is used for validating purchases in the app.

Property descriptions

payload

```
var payload: [AnyHashable : Any]
```

Additional information to be passed about the purchase. For instance, it can be used for categorizing your products.

It should contain the `NSDictionary` object that can be converted to valid JSON. The maximum size of the value is 30 KB.

productID

```
var productID: String
```

ID of the product purchased. The value can contain up to 200 characters.

quantity

```
var quantity: UInt
```

Quantity of products purchased.

It is used in the following formula:

```
Revenue = quantity * price.
```

Note: The value cannot be negative. If the value is equal to 0, the purchase is ignored.

receiptData

```
var receiptData: Data
```

Details about the in-app purchase order from App Store. This property is used for validating purchases in the app. For more information, see Apple documentation.

See the example of getting `receiptData` in the section Sending Revenue.

Attention: The value must be received before invoking `SKPaymentQueue.default().finishTransaction(transaction)` and transmitted together with transactionID.

transactionID

`var transactionID: String`

Transaction ID transactionIdentifier from the SKPaymentTransaction class. This property is used for validating purchases in the app.

See the example of getting `transactionIdentifier` in the section Sending Revenue.

Attention: The value must be received before invoking `SKPaymentQueue.default().finishTransaction(transaction)` and transmitted together with transactionID.

YMMMutableUserProfile class

The mutable version of the YMMUserProfile class with information about a user profile.

A user profile is a set of user attributes. User profile details are displayed in the AppMetrica report on user profiles.

The `YMMMutableUserProfile` instance should be passed to the AppMetrica server by using the `reportUserProfile` method of the YMMYandexMetrica class. Use methods of the YMMProfileAttribute class to create user attributes.

Instance methods

apply(_:)	Updates a single attribute of the user profile.
apply(from:)	Updates several attributes of the user profile.

Method descriptions

apply(_:)

```
func apply(_ update: YMMUserProfileUpdate)
```

Updates a single attribute of the user profile.

Parameters:

update	The `YMMUserProfileUpdate` class instance.

apply(from:)

```
func apply(from updatesArray: [YMMUserProfileUpdate])
```

Updates several attributes of the user profile.

Parameters:

updatesArray	Array of `YMMUserProfileUpdate` objects.

YMMProfileAttribute class

Methods of the class create predefined and custom profile attributes.

AppMetrica lets you create up to 100 custom attributes.

Instance methods

birthDate()	Creates a birth date attribute.
customBool(_:)	Creates a custom attribute with the `bool` type.
customCounter(_:)	Creates a custom attribute of the counter type.
customNumber(_:)	Creates a custom attribute with the `double` type.

customString(_:)	Creates a custom attribute with the `string` type.
gender()	Creates a gender attribute.
name()	Creates a name attribute.
notificationsEnabled()	Creates a NotificationsEnabled attribute.

Method descriptions

birthDate()

```
class func birthDate() -> YMMBirthDateAttribute
```

Creates a birth date attribute.

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

Returns:

The instance that implements the `YMMBirthDateAttribute` protocol.

customBool(_:)

```
class func customBool(_ name: String) -> YMMCustomBoolAttribute
```

Creates a custom attribute with the `bool` type.

Parameters:

name	Attribute name. The value can contain up to 200 characters.

Returns:

The instance that implements the `YMMCustomBoolAttribute` protocol.

customCounter(_:)

```
class func customCounter(_ name: String) -> YMMCustomCounterAttribute
```

Creates a custom attribute of the counter type.

Parameters:

name	Attribute name. The value can contain up to 200 characters.

Returns:

The instance that implements the `YMMCustomCounterAttribute` protocol.

customNumber(_:)

```
class func customNumber(_ name: String) -> YMMCustomNumberAttribute
```

Creates a custom attribute with the `double` type.

Parameters:

name	Attribute name. The value can contain up to 200 characters.

Returns:

The instance that implements the `YMMCustomNumberAttribute` protocol.

customString(_:)

```
class func customString(_ name: String) -> YMMCustomStringAttribute
```

Creates a custom attribute with the `string` type.

Parameters:

name	Attribute name. The value can contain up to 200 characters.

Returns:

The instance that implements the `YMMCustomStringAttribute` protocol.

gender()

```
class func gender() -> YMMGenderAttribute
```

Creates a gender attribute.

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

Returns:

The instance that implements the `YMMGenderAttribute` protocol.

name()

```
class func name() -> YMMNameAttribute
```

Creates a name attribute.

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

Returns:

The instance that implements the `YMMNameAttribute` protocol.

notificationsEnabled()

```
class func notificationsEnabled() -> YMMNotificationsEnabledAttribute
```

Creates a NotificationsEnabled attribute.

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

Returns:

The instance that implements the `YMMNotificationsEnabledAttribute` protocol.

YMMReporterConfiguration class

This class contains the extended immutable configuration of the reporter.

Use the YMMMutableReporterConfiguration class to change the configuration of a reporter.

Instance methods

init?(apiKey:)	Initializes an instance of the class `YMMReporterConfiguration` with the specified API key.

Properties

apiKey	API key that differs from the main application API key.

logs	The flag indicating that the logging of the reporter is enabled.
maxReportsInDatabaseCount	The maximum number of error reports stored in the internal DB.
sessionTimeout	Session timeout in seconds.
statisticsSending	A flag indicating that sending statistics is enabled.
userProfileID	Sets the ID of the user profile (`ProfileID`) when activated.

Method descriptions

init?(apiKey:)

```
init?(apiKey: String)
```

Initializes an instance of the class `YMMReporterConfiguration` with the specified API key.

Parameters:

apiKey	API key that differs from the main application API key.

Returns:

The instance of the `YMMReporterConfiguration` class.

Property descriptions

apiKey

```
var apiKey: String? { get }
```

API key that differs from the main application API key.

logs

```
var logs: Bool { get }
```

The flag indicating that the logging of the reporter is enabled.

The default value is `false`.

Possible values:

- `true` — Reporter logging is enabled.
- `false` — Reporter logging is disabled.

maxReportsInDatabaseCount

```
var maxReportsInDatabaseCount: UInt { get }
```

The maximum number of error reports stored in the internal DB.

The allowed range of values is [100; 10000]. Values outside this range are automatically replaced with values from the nearest range limits.

Default value: 1000.

Note: Separate databases are used for various `apiKeys` and independent limits on the number of events can be set for them. This parameter only affects the limitation for the corresponding `apiKey`. To change the maximum allowed number of events for other `apiKeys`, use YMMYandexMetricaConfiguration.maxReportsInDatabaseCount.

sessionTimeout

```
var sessionTimeout: UInt { get }
```

Session timeout in seconds.

The default value is `10` (minimum allowed value).

statisticsSending

```
var statisticsSending: Bool { get }
```

A flag indicating that sending statistics is enabled.

The default value is `true`.

Possible values:

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

userProfileID

```
var userProfileID: String?
```

Sets the ID of the user profile (`ProfileID`) when activated.

Attention: The maximum length of the `ProfileID` string is 200 characters.

YMMRevenueInfo class

The class contains immutable information about the revenue from in-app purchases.

Use the YMMMutableRevenueInfo class to change information about revenue.

The instance of the `YMMRevenueInfo` class should be sent to the AppMetrica server using the `reportRevenue` method of the `YMMYandexMetrica` class.

Instance methods

init(price:currency:)	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.  **Attention:** Deprecated method. Use the init!(priceDecimal:currency:) method instead.
init(priceDecimal:currency:)	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.
init(price:currency:quantity:productID:transactionID:receiptData:payload:)	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.  **Attention:** Deprecated method. Use the init!(priceDecimal:currency:quantity:productID:transactionID:receiptD method instead.
init(priceDecimal:currency:quantity:productID:transactionID:receiptData:payload:)	Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Properties

currency	Currency code of the purchase in the ISO 4217 format.
payload	Additional information to be passed about the purchase.

price	Price. It can be negative, e.g. for refunds.  **Attention:** The property is deprecated. Use the priceDecimal property instead.
priceDecimal	The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.
productID	ID of the product purchased. The value can contain up to 200 characters.
quantity	Quantity of products purchased.
receiptData	Details about the in-app purchase order from App Store.
transactionID	Details about the in-app purchase order from App Store.

Method descriptions

init(price:currency:)

```
init(price: Double, currency: String)
```

 **Attention:** Deprecated method. Use the init!(priceDecimal:currency:) method instead.

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

price	Price. It can be negative, e.g. for refunds.
currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: `RUB`. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.

Returns:

The instance of the `YMMRevenueInfo` class.

init(priceDecimal:currency:)

```
init(priceDecimal: NSDecimalNumber, currency: String)
```

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

priceDecimal	The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.

currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: RUB. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.

Returns:

The instance of the `YMMRevenueInfo` class.

init(price:currency:quantity:productID:transactionID:receiptData:payload:)

```
init(price: Double, currency: String, quantity: UInt, productID: String?, transactionID: String?, receiptData:
 Data?, payload: [AnyHashable : Any]?)
```

Attention: Deprecated method. Use the init!(priceDecimal:currency:quantity:productID:transactionID:receiptData:payload:) method instead.

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

price	Price. It can be negative, e.g. for refunds.
currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: RUB. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.
quantity	Quantity of products purchased. It is used in the following formula: `Revenue = quantity * price.` **Note:** The value cannot be negative. If the value is equal to 0, the purchase is ignored.
productID	ID of the product purchased. The value can contain up to 200 characters.
transactionID	Details about the in-app purchase order from App Store.
receiptData	Details about the in-app purchase order from App Store.
payload	Additional information to be passed about the purchase. For instance, it can be used for categorizing your products. It should contain the `NSDictionary` object that can be converted to valid JSON. The maximum size of the value is 30 KB.

Returns:

The instance of the `YMMRevenueInfo` class.

init(priceDecimal:currency:quantity:productID:transactionID:receiptData:payload:)

```
init(priceDecimal: NSDecimalNumber, currency: String, quantity: UInt, productID: String?, transactionID: String?,
 receiptData: Data?, payload: [AnyHashable : Any]?)
```

Initializes the instance of the `YMMRevenueInfo` class for sending information about purchases.

Parameters:

priceDecimal	The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.
currency	Currency code of the purchase in the ISO 4217 format. The value should contain 3 Latin letters in uppercase. Example: `RUB`. **Note:** If the value is not in the ISO 4217 format, the purchase is ignored.
quantity	Quantity of products purchased. It is used in the following formula: `Revenue = quantity * price.` **Note:** The value cannot be negative. If the value is equal to 0, the purchase is ignored.
productID	ID of the product purchased. The value can contain up to 200 characters.
transactionID	Details about the in-app purchase order from App Store.
receiptData	Details about the in-app purchase order from App Store.
payload	Additional information to be passed about the purchase. For instance, it can be used for categorizing your products. It should contain the `NSDictionary` object that can be converted to valid JSON. The maximum size of the value is 30 KB.

Returns:

The instance of the `YMMRevenueInfo` class.

Property descriptions

currency

```
var currency: String { get }
```

Currency code of the purchase in the ISO 4217 format.

payload

```
var payload: [AnyHashable : Any]? { get }
```

Additional information to be passed about the purchase.

price

```
var price: Double { get }
```

Attention: The property is deprecated. Use the priceDecimal property instead.

Price. It can be negative, e.g. for refunds.

priceDecimal

```
var priceDecimal: NSDecimalNumber? { get }
```

The price that is set using the NSDecimalNumber object. It can be negative, e.g. for refunds.

productID

```
var productID: String? { get }
```

ID of the product purchased. The value can contain up to 200 characters.

quantity

```
var quantity: UInt { get }
```

Quantity of products purchased.

receiptData

```
var receiptData: Data? { get }
```

Details about the in-app purchase order from App Store.

transactionID

```
var transactionID: String? { get }
```

Details about the in-app purchase order from App Store.

YMMUserProfile class

Immutable class for storing the user profile.

Use the YMMMutableUserProfile class to change the user profile.

A user profile is a set of user attributes. User profile details are displayed in the AppMetrica report on user profiles.

The `YMMUserProfile` instance should be passed to the AppMetrica server by using the `reportUserProfile` method of the YMMYandexMetrica class. Use methods of the YMMProfileAttribute class to create user attributes.

Instance methods

init(updates:)	Initializes the user profile with the specified set of updates.

Properties

updates	Array that contains attribute updates.

Method descriptions

init(updates:)

```
init(updates: [YMMUserProfileUpdate])
```

Initializes the user profile with the specified set of updates.

Parameters:

updates	Array that contains attribute updates.

Returns:

The instance of the `YMMUserProfile` class.

Property descriptions

updates

```
var updates: [YMMUserProfileUpdate] { get }
```

Array that contains attribute updates.

YMMYandexMetrica class

Methods of the class are used for configuring the library.

Instance methods

activate(with:)	Initializes the library in an application with the extended startup configuration.
activateReporter(with:)	Initializes a reporter with extended configuration.
handleOpen(_:)	Processes the URL that opened the application.
initWebViewReporting(_:onFailure:)	Adds a JavaScript interface with the AppMetrica name in a window to the specified webview. This lets you send client events from JavaScript code.
libraryVersion()	Returns the current version of the AppMetrica library.
pauseSession()	Pauses the user session.
reporterForApiKey(_:)	Creates a reporter for sending events to an additional API key.
report(eCommerce:onFailure:)	Sends a message about an E-commerce event.
reportError(_:exception:onFailure:)	Sends a custom error message.
report(error:onFailure:)	Sends a `YMMErrorRepresentable` message.
report(error:options:onFailure:)	Sends a `YMMErrorRepresentable` message.
report(nserror:onFailure:)	Sends an `NSError` message.
report(nserror:options:onFailure:)	Sends an `NSError` message.
reportEvent(_:onFailure:)	Sends an event message.
reportEvent(_:parameters:onFailure:)	Sends an event message with additional parameters.
reportReferralUrl(_:)	Sets referral URL for app installs. This method can be used to track some traffic sources.

reportRevenue(_:onFailure:)	Sends information about the purchase to the AppMetrica server.
report(adRevenue:onFailure:)	Sends information about advertising revenue to the AppMetrica server.
report(_:onFailure:)	Sends information about the user profile update to the AppMetrica server.
requestAppMetricaDeviceID(withCompletionQueue:completionBlock:)	Requests the unique AppMetrica ID (`deviceID`).
requestAppMetricaDeviceID(withCompletionQueue:)	Requests the unique AppMetrica ID (`deviceID`).
resumeSession()	Resumes the session, or creates a new one if the session timeout has expired.
sendEventsBuffer()	Sends stored events from the buffer.
setErrorEnvironmentValue(_:forKey:)	Sets the key-value pair associated with crashes and errors.
setLocation(_:)	Sets custom location of the device.
setLocationTracking(_:)	Enables/disables sending location of the device .
setStatisticsSending(_:)	Enables/disables sending statistics to the AppMetrica server.
setUserProfileID(_:)	Sets the ID of the user profile. AppMetrica doesn't display predefined attributes in the web interface if `ProfieId` sending isn't configured.

Method descriptions

activate(with:)

```
class func activate(with configuration: YMMYandexMetricaConfiguration)
```

Initializes the library in an application with the extended startup configuration.

Parameters:

configuration	The instance of the YMMYandexMetricaConfiguration class which contains the extended startup configuration for the library.

activateReporter(with:)

```
class func activateReporter(with configuration: YMMReporterConfiguration)
```

Initializes a reporter with extended configuration.

The configuration of the reporter should be initialized before the first call to the reporter. Otherwise, the configuration of the reporter is ignored.

The reporter should be activated with the configuration using a different API key instead of the app's API key.

Parameters:

configuration	The instance of the YMMReporterConfiguration class which contains the extended configuration for the reporter.

handleOpen(_:)

```
class func handleOpen(_ url: URL) -> Bool
```

Processes the URL that opened the application.

Used for tracking app openings via deeplink.

Parameters:

url	The URL that opened the application.

Returns:

- `true`, if the deeplink is intended for AppMetrica.
- `false`, if the deeplink is not intended for AppMetrica.

There are no such deeplink at the moment. The method always returns NO.

initWebViewReporting(_:onFailure:)

```
class func initWebViewReporting(_ userContentController: WKUserContentController, onFailure: ((Error) -> Void)? =
 nil)
```

Adds a JavaScript interface with the AppMetrica name in a window to the specified webview. This lets you send client events from JavaScript code.

Notes:

- The method must be called from the main queue.
- The method is not available on tvOS.
- Call this method before loading any content. We recommend calling this method before creating a webview and before adding your scripts to WKUserContentController.

For more information, see Usage examples.

Parameters:

userContentController	The WKUserContentController object used for this WKWebView.
onFailure	A callback method to be called in the event of an error.

libraryVersion()

```
open class func libraryVersion() -> String
```

Returns the current version of the AppMetrica library.

Returns:

Library version.

pauseSession()

```
class func pauseSession()
```

Pauses the user session.

Note: The session duration depends on specified timeout. If the time interval between pausing and resuming the session is less than the specified timeout, the current session will be resumed; otherwise, a new one will be created.

For more information about sessions, see Tracking user activity.

reporterForApiKey(_:)

```
class func reporterForApiKey(_ apiKey: String) -> YMMYandexMetricaReporting?
```

Creates a reporter for sending events to an additional API key.

To initialize a reporter with the extended configuration, use the activateReporter(with:) method. The configuration of the reporter should be initialized before the first call to the reporter. Otherwise, the configuration of the reporter is ignored.

Parameters:

apiKey	API key that differs from the main application API key.

Returns:

The instance that implements the `YMMYandexMetricaReporting` protocol for a specified API key.

report(eCommerce:onFailure:)

```
class func report(eCommerce: YMMECommerce, onFailure: ((Error) -> Void)? = nil)
```

Sends a message about an E-commerce event.

Parameters:

eCommerce	The YMMECommerce class instance.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportError(_:exception:onFailure:)

```
class func reportError(_ message: String, exception: NSException?, onFailure: ((Error) -> Void)? = nil)
```

Sends a custom error message.

Parameters:

message	Short name or description of the error.
exception	The instance of the NSException class.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(error:onFailure:)

```
class func report(error: YMMErrorRepresentable, onFailure: ((Error) -> Void)? = nil)
```

Sends a `YMMErrorRepresentable` message.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(error:options:onFailure:)

```
class func report(error: YMMErrorRepresentable, options: YMMErrorReportingOptions = [], onFailure: ((Error) ->
 Void)? = nil)
```

Sends a `YMMErrorRepresentable` message.

Use this method to set the send parameters.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(nserror:onFailure:)

```
class func report(nserror error: Error, onFailure: ((Error) -> Void)? = nil)
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(nserror:options:onFailure:)

```
class func report(nserror error: Error, options: YMMErrorReportingOptions = [], onFailure: ((Error) -> Void)? = nil)
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Use this method to set the send parameters.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
options	Parameters for sending an error.

onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportEvent(_:onFailure:)

```
class func reportEvent(_ message: String, onFailure: ((Error) -> Void)? = nil)
```

Sends an event message.

Parameters:

message	Short name or description of the event
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportEvent(_:parameters:onFailure:)

```
class func reportEvent(_ message: String, parameters params: [AnyHashable : Any]?, onFailure: ((Error) -> Void)? =
 nil)
```

Sends an event message with additional parameters.

Parameters:

message	Short name or description of the event
params	Parameters as “key-value” pairs.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportReferralUrl(_:)

```
class func reportReferralUrl(_ url: NSURL)
```

Sets referral URL for app installs. This method can be used to track some traffic sources.

Parameters:

url	Referral URL of the app install.

reportRevenue(_:onFailure:)

```
class func reportRevenue(_ revenueInfo: YMMRevenueInfo, onFailure: ((NSError) -> Void)?)
```

Sends information about the purchase to the AppMetrica server.

Parameters:

revenueInfo	The instance of the YMMRevenueInfo class which contains information about a purchase.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(adRevenue:onFailure:)

```
class func report(_ adRevenue: YMMAdRevenueInfo, onFailure: ((NSError) -> Void)?)
```

Sends information about advertising revenue to the AppMetrica server.

Parameters:

adRevenue	The instance of the YMMAdRevenueInfo class which contains information about advertising revenue.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportUserProfile(_:onFailure:)

```
class func reportUserProfile(_ userProfile: YMMUserProfile, onFailure: ((NSError) -> Void)?)
```

Sends information about the user profile update to the AppMetrica server.

Parameters:

userProfile	The instance of the YMMUserProfileclass which contains information about the user profile.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

requestAppMetricaDeviceIDWithCompletionQueue(_:block:)

```
class func requestAppMetricaDeviceIDWithCompletionQueue(_ queue: dispatch_queue_t?, completionBlock block:
 YMMAppMetricaDeviceIDRetrievingBlock)
```

Requests the unique AppMetrica ID (`deviceID`).

Note: In the Logs API and Post API `deviceID` referred to as `appmetrica_device_id`.

Parameters:

queue	The queue where the callback block will be called.
block	Callback block for receiving appmetrica_device_id. Includes an ID `appMetricaDeviceID` and error `error`, if the ID could not be obtained.

requestAppMetricaDeviceID(withCompletionQueue:)

```
class func requestAppMetricaDeviceID(withCompletionQueue queue: DispatchQueue?) async throws -> String
```

Requests the unique AppMetrica ID (`deviceID`).

The unique appmetrica identifier `deviceID` is available in AppMetricaDeviceIDListener.onLoaded(String deviceID) , if there are no errors. If an error occurs, the error value is stored in AppMetricaDeviceIDListener.onError(Error error, String referrer).

Note: In the Logs API and Post API `deviceID` referred to as `appmetrica_device_id`.

Parameters:

queue	Queue for sending `deviceID`.

Returns:

`deviceID`.

resumeSession()

```
class func resumeSession()
```

Resumes the session, or creates a new one if the session timeout has expired.

Note:

The session duration depends on specified timeout. If the time interval between pausing and resuming the session is less than the specified timeout, the current session will be resumed; otherwise, a new one will be created.

For more information about sessions, see Tracking user activity.

sendEventsBuffer()

```
class func sendEventsBuffer()
```

Sends stored events from the buffer.

AppMetrica SDK does not send an event immediately after it occurred. The library stores event data in the buffer. The `sendEventsBuffer()` method sends data from the buffer and flushes it. Use the method to force sending stored events after passing important checkpoints of user scenarios.

Attention:

Frequent use of the method can lead to increased outgoing internet traffic and energy consumption.

setErrorEnvironmentValue(_:forKey:)

```
class func setErrorEnvironmentValue(_ value: String?, forKey key: String)
```

Sets the key-value pair associated with crashes and errors.

Note: AppMetrica uses these values as additional information for further unhandled exceptions. If you pass the `nil` value, AppMetrica deletes the previous pair.

Parameters:

value	Value
key	Key

setLocation(_:)

```
class func setLocation(_ location: CLLocation?)
```

Sets custom location of the device.

Parameters:

location	Information about the location of the device.

setLocationTracking(_:)

```
class func setLocationTracking(_ enabled: Bool)
```

Enables/disables sending location of the device .

Parameters:

enabled	A flag indicating if sending information about the device location is enabled. The default value is `true`. Possible values: • `true` — Sending information about the device location is enabled. • `false` — Sending information about the location of the device is disabled.

setStatisticsSending(_:)

```
class func setStatisticsSending(_ enabled: Bool)
```

Enables/disables sending statistics to the AppMetrica server.

For more information about using the method, see Disable and enable sending statistics.

Note: Disabling sending also turns off sending data from all reporters that initialized with the other API key.

Parameters:

enabled	A flag indicating that sending statistics is enabled. The default value is `true`. Possible values: • `true` — Sending statistics is enabled. • `false` — Sending statistics is disabled.

setUserProfileID(_:)

```
class func setUserProfileID(_ userProfileID: String?)
```

Sets the ID of the user profile. AppMetrica doesn't display predefined attributes in the web interface if `ProfieId` sending isn't configured.

Parameters:

userProfileID	User profile ID.

YMMYandexMetricaConfiguration class

This class contains the extended startup configuration for the library.

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

Instance methods

init?(apiKey:)	Initializes the instance of the `YMMYandexMetricaConfiguration` class with the specified API key.

Properties

apiKey	The API key of the application.

appForKids	Defines the application type as “children's” to match the rules for checking children's apps. If this option is enabled, the AppMetrica SDK doesn't send advertising IDs or location information. **Note:** Use this property if you have an app for kids.
appOpenTrackingEnabled	A flag that enables/disables the automatic collection and sending of information about app launches via a deeplink.
appVersion	App version.
crashReporting	Enables/disables collecting and sending information about app crashes.
handleActivationAsSessionStart	Defines the AppMetrica SDK initialization as the beginning of a user session.
handleFirstActivationAsUpdate	Defines the first launch of the app as an update.
maxReportsInDatabaseCount	The maximum number of error reports stored in the internal DB.
location	Sets custom location of the device.
locationTracking	Enables/disables sending location of the device.
logs	Enables/disables logging the activity of the library.
preloadInfo	Sets the instance of the YMMYandexMetricaPreloadInfo class for tracking pre-installed apps.
revenueAutoTrackingEnabled	Enables/disables the automatic collection of information about In-App purchases.
sessionsAutoTracking	Enables/disables automatic tracking of the application lifecycle.
sessionTimeout	Sets the session timeout in seconds.
statisticsSending	Enables/disables sending statistics to the AppMetrica server.
userProfileID	Sets the ID of the user profile (`ProfileID`) when activated.

Method descriptions

init?(apiKey:)

```
public init?(apiKey: String)
```

Initializes the instance of the `YMMYandexMetricaConfiguration` class with the specified API key.

Parameters:

apiKey	The API key of the application.

Returns:

The instance of the `YMMYandexMetricaConfiguration` class.

Property descriptions

apiKey

```
var apiKey: String { get }
```

The API key of the application.

appForKids

```
var appForKids: Bool
```

Defines the application type as “children's” to match the rules for checking children's apps. If this option is enabled, the AppMetrica SDK doesn't send advertising IDs or location information.

Note: Use this property if you have an app for kids.

appOpenTrackingEnabled

```
var appOpenTrackingEnabled: Bool
```

Enables/disables the automatic collection and sending of data about app launches via a deeplink.

Attention: Automatic tracking captures only those deeplinks that resulted in app launches. To track the deeplinks inside the running application, also set up tracking.

The option is enabled by default.

Possible values:

- `YES` — Automatic collection and sending of data about the app launch via a deeplink is enabled.
- `NO` — Automatic collection and sending of data about the app launch via a deeplink is disabled.

appVersion

```
var appVersion: String?
```

App version.

crashReporting

```
var crashReporting: Bool
```

Enables/disables collecting and sending information about app crashes.

Possible values:

- `YES` — Sending information about crashes is enabled.
- `NO` — Sending information about crashes is disabled.

handleActivationAsSessionStart

```
var handleActivationAsSessionStart: Bool
```

Defines the AppMetrica SDK initialization as the beginning of a user session.

This option is disabled by default.

Possible values:

- `YES` — The user session is created when the library is initialized.
- `NO` — A background session is created when the library initializes, and a user session is created after a `UIApplicationDidBecomeActiveNotification` system event.

handleFirstActivationAsUpdate

```
var handleFirstActivationAsUpdate: Bool
```

Defines the AppMetrica SDK initialization as the beginning of a session.

This option is disabled by default.

Possible values:

- `YES` — The first launch is defined as an update.
- `NO` — The first launch is defined as a new installation.

maxReportsInDatabaseCount

```
var maxReportsInDatabaseCount: UInt
```

The maximum number of error reports stored in the internal DB.

The allowed range of values is [100; 10000]. Values outside this range are automatically replaced with values from the nearest range limits.

Default value: 1000.

Note: Separate databases are used for various `apiKeys` and independent limits on the number of events can be set for them. This parameter only affects the limitation for the corresponding `apiKey`. To change the maximum allowed number of events for other `apiKeys`, use YMMReporterConfiguration.maxReportsInDatabaseCount.

location

```
var location: CLLocation?
```

Sets custom location of the device.

locationTracking

```
var locationTracking: Bool
```

Enables/disables sending location of the device.

By default, sending is enabled.

logs

```
var logs: Bool
```

Enables/disables logging the activity of the library.

Logging is disabled by default.

preloadInfo

```
var preloadInfo: YMMYandexMetricaPreloadInfo?
```

Sets the instance of the YMMYandexMetricaPreloadInfo class for tracking pre-installed apps.

For more information, see Tracking pre-installed apps.

revenueAutoTrackingEnabled

```
var revenueAutoTrackingEnabled: Bool
```

Enables/disables the automatic collection of information about In-App purchases.

The option is enabled by default.

Possible values:

- `YES` — Automatic collection and sending of information about In-App purchases is enabled.
- `NO` — Automatic collection and sending of information about In-App purchases is disabled.

sessionsAutoTracking

```
var sessionsAutoTracking: Bool
```

Enables/disables automatic tracking of the application lifecycle.

The option is enabled by default.

If the option is disabled, you should manually set up session control using the methods +pauseSession: and +resumeSession:. For more information, see Manual session tracking.

AppMetrica uses UIApplicationDidBecomeActiveNotification and UIApplicationWillResignActiveNotification to track sessions. The maximum session length is 24 hours. To extend the session after 24 hours, invoke the +resumeSession: method manually.

sessionTimeout

```
var sessionTimeout: UInt
```

Sets the session timeout in seconds.

The default value is `10` (minimum allowed value).

For more information about sessions, see Tracking user activity.

statisticsSending

```
var statisticsSending: Bool
```

Enables/disables sending statistics to the AppMetrica server.

Note: Disabling sending also turns off sending data from all reporters that initialized with the other `apiKey`.

userProfileID

```
var userProfileID: String?
```

Sets the ID of the user profile (`ProfileID`) when activated.

Attention: The maximum length of the `ProfileID` string is 200 characters.

YMMYandexMetricaPreloadInfo class

This class contains information for tracking pre-installed apps.

Instance methods

init?(trackingIdentifier:)	Initializes the instance of the `YMMYandexMetricaPreloadInfo` class with the specified `trackingID`.
setAdditional(_:forKey:)	Returns “key-value” additional parameters that are used for tracking pre-installed apps.

Method descriptions

init?(trackingIdentifier:)

```
init?(trackingIdentifier trackingID: String)
```

Initializes the instance of the `YMMYandexMetricaPreloadInfo` class with the specified `trackingID`.

Parameters:

trackingID	Tracking ID for tracking pre-installed apps.

Returns:

The instance of the `YMMYandexMetricaPreloadInfo` class.

setAdditional(_:forKey:)

```
func setAdditional(_ info: String, forKey key: String)
```

Returns “key-value” additional parameters that are used for tracking pre-installed apps.

This method may be invoked repeatedly for setting multiple pairs of additional information

Parameters:

info	Value of the additional parameter. Can't be `nil`. Pairs with the `nil` value are ignored.
key	Key for the additional parameter. Can't be `nil`. Pairs with the `nil` key are ignored.

Protocols
YMMBirthDateAttribute protocol

The protocol defines methods for updating the age or date of birth of a user profile.

Instance methods

-withAge(_:)	Updates the attribute value.
-withDate(year:)	Updates the attribute value.
-withDate(year:month:)	Updates the attribute value.
-withDate(year:month:day:)	Updates the attribute value.
-withDate(dateComponents:)	Updates the attribute value.
-withValueReset()	Resets the attribute value.

Method descriptions

withAge(_:)

```
func withAge(_ value: UInt) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

value	Age.

Returns:

The `YMMUserProfileUpdate` class instance.

withDate(year:)

```
func withDate(year year: UInt) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

year	Year of birth.

Returns:

The `YMMUserProfileUpdate` class instance.

withDate(year:month:)

```
func withDate(year year: UInt, month: UInt) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

year	Year of birth.
month	Month of birth.

Returns:

The `YMMUserProfileUpdate` class instance.

withDate(year:month:day:)

```
func withDate(year year: UInt, month: UInt, day: UInt) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

year	Year of birth.
month	Month of birth.
day	Day of birth.

Returns:

The `YMMUserProfileUpdate` class instance.

withDate(dateComponents:)

```
func withDate(dateComponents dateComponents: DateComponents) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

dateComponents	The instance of the DateComponents class.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomBoolAttribute protocol

The protocol defines methods for updating the value of a boolean attribute.

Instance methods

withValue(_:)	Updates the attribute value.
withValueIfUndefined(_:)	Updates the attribute value if it wasn't set earlier.
withValueReset()	Resets the attribute value.

Method descriptions

withValue(_:)

```
func withValue(_ value: Bool) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

value	Attribute value: `true` or `false`.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueIfUndefined(_:)

```
func withValueIfUndefined(_ value: Bool) -> YMMUserProfileUpdate
```

Updates the attribute value if it wasn't set earlier.

Parameters:

value	Attribute value: `true` or `false`.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomCounterAttribute protocol

The protocol defines methods for updating the value of the tag type attribute.

Instance methods

withDelta(_:)	Updates the attribute value with the specified delta value.

Method descriptions

withDelta(_:)

```
func withDelta(_ value: Double) -> YMMUserProfileUpdate
```

Updates the attribute value with the specified delta value.

Parameters:

value	The delta value which you want to add or remove to the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomNumberAttribute protocol

The protocol defines methods for updating the value of a numeric attribute.

Instance methods

withValue(_:)	Updates the attribute value.
withValueIfUndefined(_:)	Updates the attribute value if it wasn't set earlier.
withValueReset()	Resets the attribute value.

Method descriptions

withValue(_:)

```
func withValue(_ value: Double) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

value	The value of the numeric attribute. The data type is `Double`.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueIfUndefined(_:)

```
func withValueIfUndefined(_ value: Double) -> YMMUserProfileUpdate
```

Updates the attribute value if it wasn't set earlier.

Parameters:

value	The value of the numeric attribute. The data type is `Double`.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMCustomStringAttribute protocol

The protocol defines methods for updating the value of a string attribute.

Instance methods

withValue(_:)	Updates the attribute value.
withValueIfUndefined(_:)	Updates the attribute value if it wasn't set earlier.
withValueReset()	Resets the attribute value.

Method descriptions

withValue(_:)

```
func withValue(_ value: String?) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

value	Attribute value as a string. The maximum value length is 200 characters.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueIfUndefined(_:)

```
func withValueIfUndefined(_ value: String?) -> YMMUserProfileUpdate
```

Updates the attribute value if it wasn't set earlier.

Parameters:

value	Attribute value as a string. The maximum value length is 200 characters.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMErrorRepresentable protocol

Protocol for errors that can be sent in AppMetrica.

Each class instance that implements the protocol must have an ID. AppMetrica uses IDs to group errors.

All error information that is sent is available in AppMetrica reports.

You can implement this protocol to send your own errors. You can also use the default YMMError implementation.

Properties

identifier	Error ID. AppMetrica uses it to group errors.
message	Arbitrary error description.
parameters	Additional error parameters.
backtrace	Custom error backtrace. You can get it using the `NSThread.callStackReturnAddresses` method.
underlyingError	Error instance that matches the `YMMErrorRepresentable` protocol.

Property descriptions

identifier

```
var identifier: String { get }
```

Error ID. AppMetrica uses it to group errors.

The maximum value length is 300 characters. If the value exceeds the limit, AppMetrica truncates it.

Note: AppMetrica doesn't use nested error (underlyingError) IDs for grouping.

message

```
optional var message: String? { get }
```

Arbitrary error description.

The maximum value length is 1000 characters. If the value exceeds the limit, AppMetrica truncates it.

parameters

```
optional var parameters: [String : Any]? { get }
```

Additional error parameters.

Parameters are represented as key-value pairs, where the key and value are strings. If the key or value differs from the string type, AppMetrica calls the `description` method automatically.

The maximum number of parameters is 50. The maximum allowed parameter size is 100 characters for the key and 2000 characters for the value. If the value exceeds the limit, AppMetrica truncates it.

backtrace

```
optional var backtrace: [NSNumber]? { get }
```

Custom error backtrace. You can get it using the `NSThread.callStackReturnAddresses` method.

The maximum number of stack frames is 200. If the value exceeds the limit, AppMetrica truncates it.

underlyingError

```
optional var underlyingError: YMMErrorRepresentable? { get }
```

Error instance that matches the `YMMErrorRepresentable` protocol.

The maximum number of nested errors is 10. If the value exceeds the limit, AppMetrica truncates it.

YMMGenderAttribute protocol

The protocol defines methods for updating the gender of a user profile.

Instance methods

withValue(_:)	Updates the gender attribute with the specified value from the `YMMGenderType` enum.
withValueReset()	Resets the attribute value.

Method descriptions

withValue(_:)

```
func withValue(_ value: YMMGenderType) -> YMMUserProfileUpdate
```

Updates the gender attribute with the specified value from the `YMMGenderType` enum.

Parameters:

value	The value from the `YMMGenderType` enum.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMNameAttribute protocol

The protocol defines methods for updating the name of a user profile.

Instance methods

withValue(_:)	Updates the attribute value.
withValueReset()	Resets the attribute value.

Method descriptions

withValue(_:)

```
func withValue(_ value: String?) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

value	The name of the user profile. The maximum length of the user profile name is 100 characters.

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMNotificationsEnabledAttribute protocol

The protocol defines methods for updating the notification status of a user profile.

It indicates whether the user has enabled notifications for the application.

Instance methods

withValue(_:)	Updates the attribute value.

withValueReset()	Resets the attribute value.

Method descriptions

withValue(_:)

```
func withValue(_ value: Bool) -> YMMUserProfileUpdate
```

Updates the attribute value.

Parameters:

value	Attribute value: `YES` or `NO`

Returns:

The `YMMUserProfileUpdate` class instance.

withValueReset()

```
func withValueReset() -> YMMUserProfileUpdate
```

Resets the attribute value.

Returns:

The `YMMUserProfileUpdate` class instance.

YMMYandexMetricaReporting protocol

Instance methods

pauseSession()	Pauses the user session.
report(eCommerce:onFailure:)	Sends a message about an E-commerce event.
reportError(_:exception:onFailure:)	Sends a custom error message.  **Attention:** Deprecated method.
report(error:onFailure:)	Sends a `YMMErrorRepresentable` message.
report(error:options:onFailure:)	Sends a `YMMErrorRepresentable` message.
report(nserror:onFailure:)	Sends an `NSError` message.
report(nserror:options:onFailure:)	Sends an `NSError` message.
report(adRevenue:onFailure:)	Sends information about advertising revenue to the AppMetrica server.
reportEvent(_:onFailure:)	Sends a custom event message.
reportEvent(_:parameters:onFailure:)	Sends a custom event message with additional parameters.
reportRevenue(_:onFailure:)	Sends information about the purchase to the AppMetrica server.
report(_:onFailure:)	Sends information about the user profile update to the AppMetrica server.
resumeSession()	Resumes the session, or creates a new one if the session timeout has expired.

sendEventsBuffer()	Sends stored events from the buffer.
setStatisticsSending(_:)	Enables/disables sending statistics to the AppMetrica server.
setUserProfileID(_:)	Sets the ID of the user profile. AppMetrica doesn't display predefined attributes in the web interface if `ProfieId` sending isn't configured.

Method descriptions

pauseSession()

```
func pauseSession()
```

Pauses the user session.

report(eCommerce:onFailure:)

```
class func report(eCommerce: YMMECommerce, onFailure: ((Error) -> Void)? = nil)
```

Sends a message about an E-commerce event.

Parameters:

eCommerce	The YMMECommerce class instance.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportError(_:exception:onFailure:)

```
func reportError(_ name: String, exception: NSException?, onFailure: ((Error) -> Void)? = nil)
```

Sends a custom error message.

Attention: Deprecated method.

Parameters:

name	Short name or description of the error.
exception	The instance of the NSException class.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(error:onFailure:)

```
func report(error: YMMErrorRepresentable, onFailure: ((Error) -> Void)? = nil)
```

Sends a `YMMErrorRepresentable` message.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(error:options:onFailure:)

```
func report(error: YMMErrorRepresentable, options: YMMErrorReportingOptions = [], onFailure: ((Error) -> Void)? =
 nil)
```

Sends a `YMMErrorRepresentable` message.

Use this method to set the send parameters.

Note: For more information, see the YMMErrorRepresentable protocol description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(nserror:onFailure:)

```
func report(nserror error: Error, onFailure: ((Error) -> Void)? = nil)
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(nserror:options:onFailure:)

```
func report(nserror error: Error, options: YMMErrorReportingOptions = [], onFailure: ((Error) -> Void)? = nil)
```

Sends an `NSError` message.

AppMetrica uses the domain and code to group errors.

Use this method to set the send parameters.

Limits for an `NSError`:

- 200 characters for the `domain`.
- 50 key-value pairs for `userInfo`, 100 characters for the key, and 2000 characters for the value.
- 10 nested errors that use `NSUnderlyingErrorKey` as a key in `userInfo`.
- 200 stack frames in the `YMMBacktraceErrorKey` backtrace as a key in `userInfo`.

If the value exceeds the limit, AppMetrica truncates it.

Note: You can send your own error backtrace in `NSError`. For more information, see the YMMBacktraceErrorKey constant description.

Parameters:

error	The error to send.
options	Parameters for sending an error.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

report(adRevenue:onFailure:)

```
func report(_ adRevenue: YMMAdRevenueInfo, onFailure: ((NSError) -> Void)?)
```

Sends information about advertising revenue to the AppMetrica server.

Parameters:

adRevenue	The instance of the YMMAdRevenueInfo class which contains information about advertising revenue.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportEvent(_:onFailure:)

```
func reportEvent(_ name: String, onFailure: ((Error) -> Void)? = nil)
```

Sends a custom event message.

Parameters:

name	Short name or description of the event
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportEvent(_:parameters:onFailure:)

```
func reportEvent(_ name: String, parameters params: [AnyHashable : Any]?, onFailure: ((Error) -> Void)? = nil)
```

Sends a custom event message with additional parameters.

Parameters:

name	Short name or description of the event
params	Parameters as “key-value” pairs.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportRevenue(_:onFailure:)

```
func reportRevenue(_ revenueInfo: YMMRevenueInfo, onFailure: ((NSError) -> Void)?)
```

Sends information about the purchase to the AppMetrica server.

Parameters:

revenueInfo	The instance of the YMMRevenueInfo class which contains information about a purchase.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

reportUserProfile(_:onFailure:)

```
func reportUserProfile(_ userProfile: YMMUserProfile, onFailure: ((NSError) -> Void)?)
```

Sends information about the user profile update to the AppMetrica server.

Parameters:

userProfile	The instance of the YMMUserProfileclass which contains information about the user profile.
onFailure	The block that is executed when an error occurs. The error is passed as a block argument.

resumeSession()

```
func resumeSession()
```

Resumes the session, or creates a new one if the session timeout has expired.

sendEventsBuffer()

```
func sendEventsBuffer()
```

Sends stored events from the buffer.

AppMetrica SDK does not send an event immediately after it occurred. The library stores event data in the buffer. Method

`sendEventsBuffer()`

sends data from the buffer and flushes it. Use the method to force sending stored events after passing important checkpoints of user scenarios.

Attention:

Frequent use of the method can lead to increased outgoing internet traffic and energy consumption.

setStatisticsSending(_:)

```
func setStatisticsSending(_ enabled: Bool)
```

Enables/disables sending statistics to the AppMetrica server.

Note:

Disable sending statistics to the reporter does not affect the sending of data from the main API key. But disabling data sending for the main API key stops sending statistics from all reporters.

Parameters:

enabled	A flag indicating that sending statistics is enabled. The default value is `true`. Possible values: • `true` — Sending statistics is enabled. • `false` — Sending statistics is disabled.

setUserProfileID(_:)

```
func setUserProfileID(_ userProfileID: String?)
```

Sets the ID of the user profile. If the

`ProfileId`

isn't sent, predefined attributes aren't displayed in the web interface.

Parameters:

userProfileID	User profile ID.

Enumerations
YMMAdType

Contains values of advertising types (AdType).

Enumerations

YMMAdType

Enumeration descriptions

YMMAdType

```
enum YMMAdType: Int {
    case unknown = 0
    case native = 1
    case banner = 2
    case rewarded = 3
    case interstitial = 4
    case mrec = 5
    case other = 6
}
```

YMMGenderType

Contains possible gender values for the `YMMGenderAttribute` class.

Enumerations

YMMGenderType

######## ############

YMMGenderType

`enum YMMGenderType : UInt`

Constants	Description
YMMGenderType.male	Male.

Constants	Description
YMMGenderType.female	Female.
YMMGenderType.other	Other. For example, there is not enough info to detect the gender. **Note:** You can set the `YMMGenderTypeOther` as the attribute value and pass additional information using custom attributes.

YMMErrorReportingOptions

Contains parameters for sending errors.

Enumerations

YMMErrorReportingOptions

######## ############

YMMErrorReportingOptions

`YMMErrorReportingOptions`

Constants	Description
`static var noBacktrace: YMMErrorReportingOptions { get }`	Option that doesn't attach the send call backtrace to the error. Can speed up sending reports. **Note:** This option doesn't affect the backtraces that are passed in the error itself.

Constants
YMMBacktraceErrorKey

```
let YMMBacktraceErrorKey: String
```

A key from the `userInfo` dictionary of an `NSError`. It should contain the error backtrace.
You can get it using the `NSThread.callStackReturnAddresses` (Objective-C) or `Thread.callStackReturnAddresses`(Swift) method.

AppMetrica automatically parses the passed value.

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

Objective-C

To change the length of the timeout, pass the value in seconds to the sessionTimeout property of the library configuration YMMYandexMetricaConfiguration.

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

```
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc]
 initWithApiKey:@"API_key"];
// Setting the session timeout.
configuration.sessionTimeout = 15;
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

Swift

To change the length of the timeout, pass the value in seconds to the sessionTimeout property of the library configuration YMMYandexMetricaConfiguration.

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

```
// Creating an extended library configuration.
let configuration = YMMYandexMetricaConfiguration.init(apiKey: "API key")
// Setting the session timeout.
configuration?.sessionTimeout = 15
// Initializing the AppMetrica SDK.
YMMYandexMetrica.activate(with: configuration!)
```

Tracking sessions manually

AppMetrica automatically tracks the lifecycle of applications by default. To track sessions manually:

1. Initialize the library with automatic session tracking disabled sessionsAutoTracking.

```
// Creating an extended library configuration.
YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc] initWithApiKey:API_key];
// Disabling automatic tracking user activity.
configuration.sessionsAutoTracking = NO;
...
// Initializing the AppMetrica SDK.
[YMMYandexMetrica activateWithConfiguration:configuration];
```

2. Set up session control using the methods +resumeSession: and +pauseSession:.

```
[YMMYandexMetrica resumeSession];
...
[YMMYandexMetrica pauseSession];
```

When using manual tracking, make sure that the current active session always ends with the `+pauseSession:` method invoke. If you don't invoke the `+pauseSession:` method the session will be ended the next time the app is launched.

For correct tracking, the reporters should be manually configured to send events about the start and the pause of the session for each reporter:

```
id<YMMYandexMetricaReporting> reporter = [YMMYandexMetrica reporterForApiKey:API_key];
[reporter resumeSession];
...
[reporter reportEvent:@"Updates installed" onFailure:^(NSError *error) {
    NSLog(@"REPORT ERROR: %@", [error localizedDescription]);
}];
...
[reporter pauseSession];
```

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.

Analysis of app crashes

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

The library does not symbolicate or deobfuscate app crashes. These operations are performed on the server or on the client side. For more information, see Uploading dSYM files on iOS.

Related information

iOS reference on page 78

Contact support

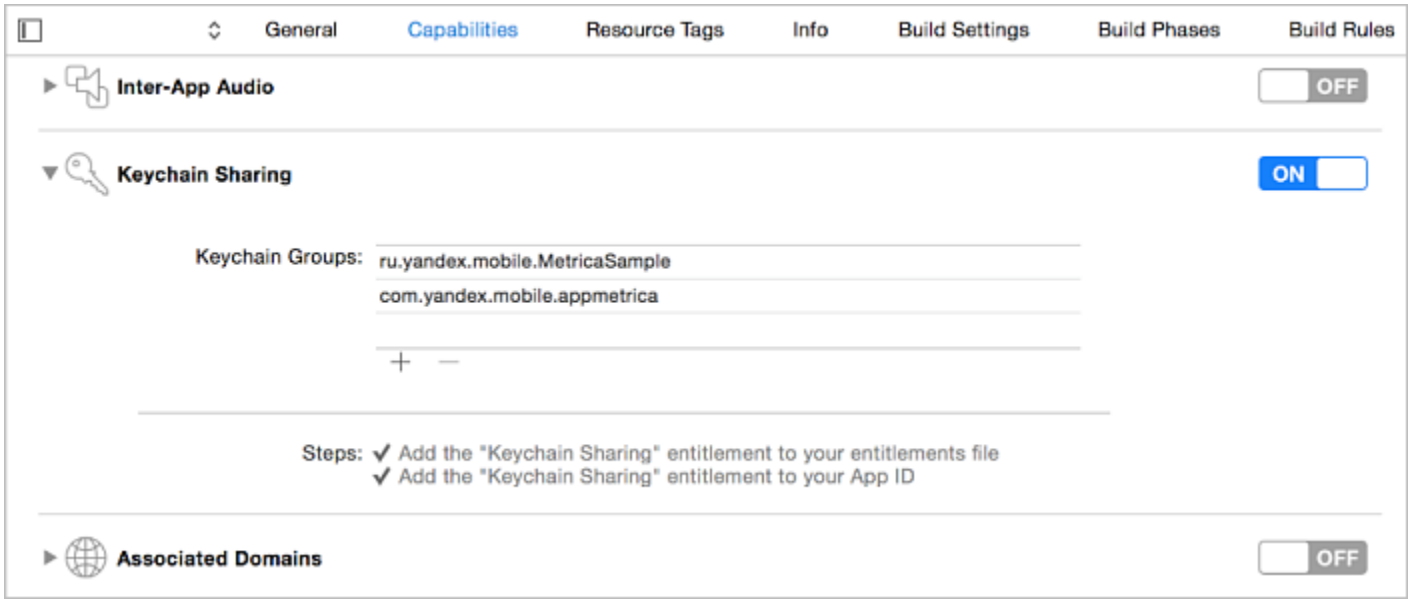
If you didn't find the answer you were looking for, you can use the feedback form to submit your question. Please describe the problem in as much detail as possible. Attach a screenshot if possible.

Device identification using a vendor keychain

By storing `device IDs` in Keychain, you can uniquely identify a device for custom reports in AppMetrica. However, if a device has multiple apps from the same developer installed, the device might be identified differently in the reports for each of the apps. To avoid this issue, use a vendor keychain.

Setting up a vendor keychain

1. In the Xcode interface, go to the **Capabilities** pane in the target settings and turn on **Keychain Sharing**.
2. In **Keychain Groups**, add the group `com.yandex.mobile.appmetrica`.



Entitlements of the target after setup:

Key	Type	Value
Entitlements File	Dictionary	(1 item)
Keychain Access Groups	Array	(2 items)
Item 0	String	$(AppIdentifierPrefix)ru.yandex.mobile.MetricaSample
Item 1	String	$(AppIdentifierPrefix)com.yandex.mobile.appmetrica

After you have set this up, the AppMetrica SDK automatically uses the vendor keychain data.

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.

Universal Links support

iOS requires a secure connection between an application and its associated domains. In order to establish such a connection, AppMetrica puts a file called `apple-app-site-association` on its domain. File contains *Bundle ID* and *App Prefix* applications.

Note: Starting from iOS 12.3.1 and higher, the redirect service won't be able to redirect a user to a deeplink. To make sure deeplinks work properly, use an app#to#app tracking link (see Using a direct Universal Link).

Setting up Universal Links

To activate Universal Links for your app, follow these steps in the AppMetrica web interface:

1. Go to the **Applications** section.
2. Select the app and go to **Settings**.
3. Open the **Main** tab and enter your app's Bundle ID and App Prefix.

   How to get the Bundle ID

   You can find the Bundle ID in the Apple developer console, in **Organization Profile** → **Account Summary**. In Xcode, you can find it in **Target** → **General**.

   How to get the App Prefix

   You can find the App Prefix in the Apple Developer Console. In most cases, the App Prefix is the same as the Team ID. This ID is also available in the Apple Developer Console (in the **Member Center**, click your name in the upper-right corner of the window and choose **View Account** → **Developer Account Summary**).

4. Set the **Use Universal Link** option to **On** and click **Save settings**. After this, you will see a link in the **Universal Link** field that looks like `applinks:<app_id>.redirect.appmetrica.yandex.com`, where `<app_id>` is your app's ID in AppMetrica (**application ID**).

Preparing your app

If your app is already registered in the Apple Developer Center, go to the developer console and enable **Associated Domains** for the application ID:

1. Go to **Identifiers** → **App IDs**.
2. Select the app, then enable the **Associated Domains** option for the **Development** and **Distribution** columns.

Configuring your app

In the Xcode interface, follow these steps:

1. Select the app **target**.
2. In **Capabilities**, enable the **Associated Domains** option.
3. In **Domains**, enter the Universal Link generated by AppMetrica.

Using a direct Universal Link

A direct Universal Link has the following format:

```
https://<application_id>.redirect.appmetrica.yandex.com/?appmetrica_tracking_id=<tracking_id>
```

Depending on the circumstances, clicking a link like this makes one of the following happen:

- The app opens, if it is already installed.
- The app's page in the App Store opens, if the app isn't installed.

Advantages

Maximum coverage of sources where the Universal Link can be published. Direct links aren't confined to browsers – they also work from mobile app ads (app-to-app advertising).

Disadvantages

Statistics might be incomplete: if the app is installed, the direct link opens it without going through the redirect service. This click-through isn't registered in AppMetrica. Since this doesn't affect statistics on new users, the data loss is

inconsequential. If the app isn't installed, AppMetrica registers a click-through to the App Store using the direct Universal Link.

Testing

Now your app is fully configured and you can use Universal Links. While the application included Universal Links, when referring to the domain of the view `applinks:<app_id>.redirect.appmetrica.yandex.com`, will navigate directly to the app.

Note: To test how the Universal Link works, you must use a real device. If the **Associated Domain** is changed, make sure the app is reinstalled.

If Safari opens instead of the app, reinstall the app. If the problem persists, add a new **Associated Domains** row in Xcode (so it will change), and then reinstall the app.

Make sure that your link has the same domain as your app.

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

This guide contains examples that demonstrate the differences between the SDK versions 2.9.6 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:

```
YMMMutableReporterConfiguration *configuration = [[YMMMutableReporterConfiguration alloc]
 initWithApiKey:reporter_API_key];
configuration.sessionTimeout = 32; // Session timeout of the reporter.
configuration.logs = YES; // Enabling reporter logging.
[YMMYandexMetrica activateReporterWithConfiguration:[configuration copy]];
```

Profiles

Added the following classes to work with user profiles:

- `YMMUserProfile`.
- `YMMProfileAttribute`.
- `YMMUserProfileUpdate`.

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

- `+ (void)setUserProfileID:(nullable NSString *)userProfileID;`.
- `+ (void)reportUserProfile:(YMMUserProfile *)userProfile onFailure:(nullable void (^)(NSError *error))onFailure;`.

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

- `- (void)setUserProfileID:(nullable NSString *)userProfileID;`.
- `- (void)reportUserProfile:(YMMUserProfile *)userProfile onFailure:(nullable void (^)(NSError *error))onFailure;`.

Revenue

Added the `YMMRevenueInfo` class for revenue tracking.

Added the following method to the `YMMYandexMetrica` class:

- `+(void)reportRevenue:(YMMRevenueInfo *)revenueInfo onFailure:(nullable void (^)(NSError *error))onFailure;`.

Added the following method to the `YMMYandexMetricaReporting` protocol:

- `- (void)reportRevenue:(YMMRevenueInfo *)revenueInfo onFailure:(nullable void (^)(NSError *error))onFailure;`.

Renamed

Version 2.9.6	Version 3.0.0

`setTrackLocationEnabled`	`// Enables device location sending.`
	`setLocationTracking`
	
	The `YMMYandexMetrica` class.

`handleFirstActivationAsUpdateEnabled`	`// 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.`
	`handleFirstActivationAsUpdate`
	
	The `YMMYandexMetricaConfiguration` class.

`trackLocationEnabled`	`// Enables device location sending.`
	`locationTracking`
	
	The `YMMYandexMetricaConfiguration` class.

`reportCrashesEnabled`	`// Enables application crashes reporting.`
	`crashReporting`
	
	The `YMMYandexMetricaConfiguration` class.

`customAppVersion`	`// Sets the app version.`
	`appVersion`
	
	The `YMMYandexMetricaConfiguration` class.

`loggingEnabled`	`// Enables logging.`
	`logs`
	
	The `YMMYandexMetricaConfiguration` class.

Deleted

Version 2.9.6	Version 3.0.0

Activation with the API key
`[YMMYandexMetrica activateWithApiKey:API_key];`
	`YMMYandexMetricaConfiguration *configuration = [[YMMYandexMetricaConfiguration alloc] initWithApiKey:API_key];`
	`[YMMYandexMetrica activateWithConfiguration:configuration];`
	
	You can activate the library using the extended configuration `YMMYandexMetricaConfiguration`.

Setting the session length. `[YMMYandexMetrica setSessionTimeout:30];`	You can set the session timeout using the extended configuration `YandexMetricaConfig`.
Monitoring app crashes. `[YMMYandexMetrica setReportCrashesEnabled:YES];`	You can enable the crash reporting using the extended configuration `YMMYandexMetricaConfiguration`.

Version 2.9.6	Version 3.0.0
Setting the app version `[YMMYandexMetrica setCustomAppVersion:@"1.0.5"];`	You can set the application version using the extended configuration `YMMYandexMetricaConfiguration`.
Enabling logs `[YMMYandexMetrica setLoggingEnabled:YES];`	You can enable the library logging using the extended configuration `YMMYandexMetricaConfiguration`.
Setting environment values. `[YMMYandexMetrica setEnvironmentValue:@"bar" forKey:@"foo"];`	Deleted.
Enables deeplink tracking by using provided application's URL scheme. `[YMMYandexMetrica enableTrackingWithURLScheme:urlScheme];`	Deleted. This attribution method is no longer supported.
Activation with the numberic key. `[YMMYandexMetrica startWithAPIKey:13];`	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

Version 4.5.2

Released May 22, 2023

- Fixed `posix_spawn` Xcode crash with sanitizer enabled.

Version 4.5.0

Released March 21, 2023

- Fixed the `locationServicesEnabled invoked on the main thread` error on iOS 16.

Version 4.4.0

Released September 19, 2022

- Added the AdRevenue API for transmitting advertising monetization revenue at the impression level (Impression-Level Revenue Data):
  - The YMMAdRevenueInfo, YMMMutableAdRevenueInfo classes.
  - The reportAdRevenue methods in the `YMMYandexMetrica`, reportAdRevenue class in the `YMMYandexMetricaReporting` protocol.
  - The YMMAdType enumeration.

Version 4.2.0

Released February 18, 2022

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

  Protocols:

  - YMMYandexMetricaPlugins.
  - YMMYandexMetricaPluginReporting.

  Classes:

  - YMMPluginErrorDetails.
  - YMMStackTraceElement.
- Added API for the correct operation of the SDK in the context of session auto-tracking if activated from plugins: `YMMYandexMetricaPlugins.handlePluginInitFinished`.
- Added the ability to send errors from reporters without activating the master key. In this case, meta information from KSCrash (system data) will be missing.

Version 4.0.0

Released September 20, 2021

- Supported the option to set the user profile ID when activating (`[YMMYandexMetricaConfiguration userProfileID]`) or before activating (`[YMMYandexMetrica setUserProfileID]`) the master key, as well as when activating the reporter (`[YMMReporterConfiguration userProfileID]`).
- Added the appOpenTrackingEnabled property for automatic tracking of app openings via a deeplink (Objective-C | Swift).
- Added the revenueAutoTrackingEnabled property for the automatic collection of data on In-App purchases (Objective-C | Swift).
- Added managing the Conversion Value.

Version 3.17.0

Released June 8, 2021

- Fixed the error `Main Thread Checker: UI API called on a background thread: -[WKUserScript initWithSource:injectionTime:forMainFrameOnly:]` that may occur.
- Added a library version for iOS simulators that are run on Apple Silicon M1 Macs (`ios-arm64-simulator`).
- The library is now delivered only as an XCFramework, which caused the following changes:
  - The minimum supported version of CocoaPods is 1.10 and Carthage is 0.38. These versions now support XCFrameworks.
  - The AppMetrica SDK is now connected on tvOS the same way as on iOS: the `YandexMobileMetrica/Dynamic-TV` and `YandexMobileMetrica/Static-TV` subspecs are no longer available.

Version 3.16.0

Released May 27, 2021

- Added the `initWebViewReporting` method (Objective-C, Swift) to send events from the WebView JS code.
- Fixed errors and improved stability.

Version 3.15.1

Released April 20, 2021

- Improved Apple Search Ads attribution via AdServices Framework. Upgrade to this version to keep Apple Search Ads tracking on iOS 14.5+.

Version 3.15.0

Released March 30, 2021

- Added support for attributing installations on devices running iOS version 14.5+ via SKAdNetwork. Upcoming versions of the SDK will make it possible to transmit conversion values.
- Data can now be received for attributing installations from Apple Search Ads via AdServices Framework (for devices running iOS 14.3+). Attribution is implemented in the AppMetrica backend and doesn't require further updates.
- Operator and Connection type collection is disabled.

Version 3.14.0

Released 29 December 2020.

- Added AppMetrica distribution using Swift Package Manager. For more information, see Installation and initialization.
- The minimum supported iOS version is 9.0 or later.
- Fixed an issue in linking crashes to sessions.

Version 3.12.0

Released 14 October 2020

- Added a new API for sending E-Commerce events:
  - Added the +reportECommerce:onFailure: method to the YMMYandexMetrica class and the YMMYandexMetricaReporting protocol.
  - Added new classes:
    - YMMECommerce
    - YMMECommerceAmount
    - YMMECommerceCartItem
    - YMMECommerceOrder
    - YMMECommercePrice
    - YMMECommerceProduct
    - YMMECommerceReferrer
    - YMMECommerceScreen

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

Version 3.11.1

Released 8 July 2020

- Added a new API for sending crashes and errors:
  - The YMMError class.
  - The YMMErrorRepresentable protocol.
  - YMMErrorReportingOptions enumeration.
  - The YMMBacktraceErrorKey constant.
  - Added the following methods to the YMMYandexMetrica class and YMMYandexMetricaReporting protocol:
    - +reportError:onFailure:
    - +reportError:options:onFailure:
    - +reportNSError:onFailure:
    - +reportNSError:options:onFailure:
    - +setErrorEnvironmentValue:forKey:
  - Added the maxReportsInDatabaseCount property to the YMMYandexMetricaConfiguration, YMMReporterConfiguration, and YMMMutableReporterConfiguration classes.
- Stopped supporting the +reportError:exception:onFailure method. Use the new +reportError:onFailure:, +reportError:options:onFailure:, +reportNSError:onFailure:, or +reportNSError:options:onFailure: methods instead.

- Added support for children's apps. Use the appForKids property of the YMMYandexMetricaConfiguration configuration. For more information, see Usage examples.
- Fixed support for tvOS.
- Improved the performance of the library and the quality of statistical data.

Version 3.9.4

Released 3 February 2020

- Fixed possible crashes in AppMetrica SDK 3.9.1 and 3.9.2.

Version 3.9.2

Released 27 December 2019

- Fixed incorrect `appmetrica_device_id` generation process.
- Fixed possible deadlock during the activation.
- Fixed the `reportReferralUrl` method (Objective-C | Swift). It is no longer deprecated.
- Fixed getting information about `code` and `sudcode` for Mach exceptions.
- Fixed framework for tvOS.
- Improved the performance of the library and the quality of statistical data.

Version 3.8.2

Released 4 October 2019

- Fixed a priceDecimal serialization error that caused negative Revenue values.

Version 3.8.1

Released 30 September 2019

- Fixed an issue in the dynamic framework.
- Fixed collection of information on location. When you set your own location, automatic detection is disabled.

Version 3.8.0

Released 25 September 2019

- Added the `helper` command-line tool for Uploading dSYM files on iOS.

Version 3.7.1

Released 11 July 2019

- Added to the YMMRevenueInfo class:
  - Method -initWithPriceDecimal:currency:. Use it instead of deprecated #initWithPrice:currency:.
  - Method -initWithPriceDecimal:currency:quantity:productID:transactionID:receiptData:payload:. Use it instead of deprecated #initWithPrice:currency:quantity:productID:transactionID:receiptData:payload:.
  - Property priceDecimal. Use it instead of deprecated price.
- Added methods for manual control of sessions to YMMYandexMetrica:
  - Method +pauseSession:.
  - Method +resumeSession:.
- Added properties for control sessions to YMMYandexMetricaConfiguration:
  - Property handleActivationAsSessionStart.
  - Property sessionsAutoTracking.
- Stopped supporting the method +reportReferralUrl:. Deprecated method.
- Fixed the issue with additional information in crash logs: active_time_since_launch, active_time_since_last_crash, etc.

Version 3.6.0

Released 18 February 2019

- Fixed possible loss of crash reports on devices with a 32-bit processor.
- Fixed possible crashes which affected the AppMetrica SDK versions 3.1.0 to 3.5.0.
- Improved the performance of the library and the quality of statistical data.

Version 3.5.0

Released 25 December 2018

- Added support for tvOS 9 and higher.
- Improved the performance of the library and the quality of statistical data.

Version 3.4.1

Released 15 November 2018

- Fixed the issue when enabling the static framework in a project using Swift.

Version 3.4.0

Released 2 November 2018

- Separated the library into two frameworks: core and crash-handling. For more information, see Installation and initialization.
- Fixed the +sendEventsBuffer: method to work correctly in the background.
- Improved the performance of the library and the quality of statistical data.

Version 3.3.0

Released 6 September 2018

- Improved processing of information transmitted by methods +reportUserProfile:onFailure: and +reportRevenue:onFailure:.
- Improved the performance of the library and the quality of statistical data.

Version 3.2.0

Released 20 July 2018

- Added the +setStatisticsSending: method to disable statistics sending.
- Added the +requestAppMetricaDeviceIDWithCompletionQueue:block: 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 3.1.2

Released 2 July 2018

- Changed the SDK to meet the requirements of the Apple App Store Review Team. Update the AppMetrica SDK to avoid any issues during the App Store moderation process.

  Attention: Previous versions of iOS SDK (2.8.0–3.1.1) are not available for download. If you are using library version 2.9.x, update it to version 2.9.8.

Version 3.1.1

Released 13 June 2018

- Fixed an issue in the AppMetrica SDK 3.1.0 related to internal data loss.

Version 3.1.0

Released 8 June 2018

- Added deeplink attribution (Re-engagement).
- Fixed possible deadlock affecting AppMetrica SDK versions 3.0.0 and 3.0.1
- Improved the performance of the library and the quality of statistical data.

Version 3.0.1

Released 21 May 2018

- Improved the stability of the library.

Version 3.0.0

Released 18 April 2018

- Added methods for creating user profiles.
- Added revenue tracking.
- Unified and revised the API. Read more in the migration guide to iOS SDK 3.0.0.
- Changed the app version and build number order in crash reports to match the Apple format.
- Extended logging of events flow.
- Stopped supporting iOS 6 and iOS 7.
- Improved the performance of the library and the quality of statistical data.

Version 2.9.8

Released 2 July 2018

- Changed the SDK to meet the requirements of the Apple App Store Review Team. Download the AppMetrica SDK 2.9.8 and update it in the app to avoid issues during the App Store moderation process. tab.

Attention: Previous versions of iOS SDK (2.8.0–3.1.1) are not available for download.

Version 2.9.6

Released 12 January 2018

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

Version 2.9.4

Released 4 November 2017

- Fixed possible errors on the simulator.
- Added Apple Search Ads tracking support.
- Improved the performance of the library and the quality of statistical data.

Version 2.9.1

Released 14 August 2017

- Added the +reportReferralUrl: method which sets referral URL for app installs.

Version 2.9.0

Released 18 July 2017

- Added the ability to send statistics using an API key that differs from the app's API key.
- Improved the performance of the library and the quality of statistical data.

Version 2.8.3

Released 19 June 2017

- Fixed bitcode problems with Xcode 8.2.1.

Version 2.8.1

Released 12 June 2017

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

Version 2.8.0

Released 4 April 2017

- Fixed incorrect version/build number of application in crash reports.
- Fixed custom location setting.
- Improved the performance of the library and the quality of statistical data.

Version 2.7.0

Released 16 December 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. For more information, see Usage examples.
- Improved the stability of the library.

Version 2.6.5

Released 17 November 2016

- Corrected meta information for the dynamic framework.

Version 2.6.2

Released 10 October 2016

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

Version 2.6.1

Released 30 September 2016

- Improved support for iOS 6.

Version 2.6.0

Released 20 September 2016

- Improved support for Swift.
- Improved support for iOS 10.
- Added a dynamic framework to the library.

Version 2.5.1

Released 11 July 2016

- Optimization for improved quality of statistics.
- Fixed an error that was causing app crashes.

Version 2.5.0

Released 24 May 2016

- Optimization for improved quality of statistics.

Version 2.4.1

Released 25 April 2016

- Optimization for improved quality of statistics.

Version 2.4.0

Released 23 March 2016

- Added the Referrer-based Tracking technology.
- Optimization for improved quality of statistics.

Version 2.3.1

Released 25 February 2016

- Eliminated external dependencies.
- Added a distribution model using the static framework.
- Optimization for improved quality of statistics.

Version 2.3.0

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.

Version 2.1.1

Released 1 October 2015

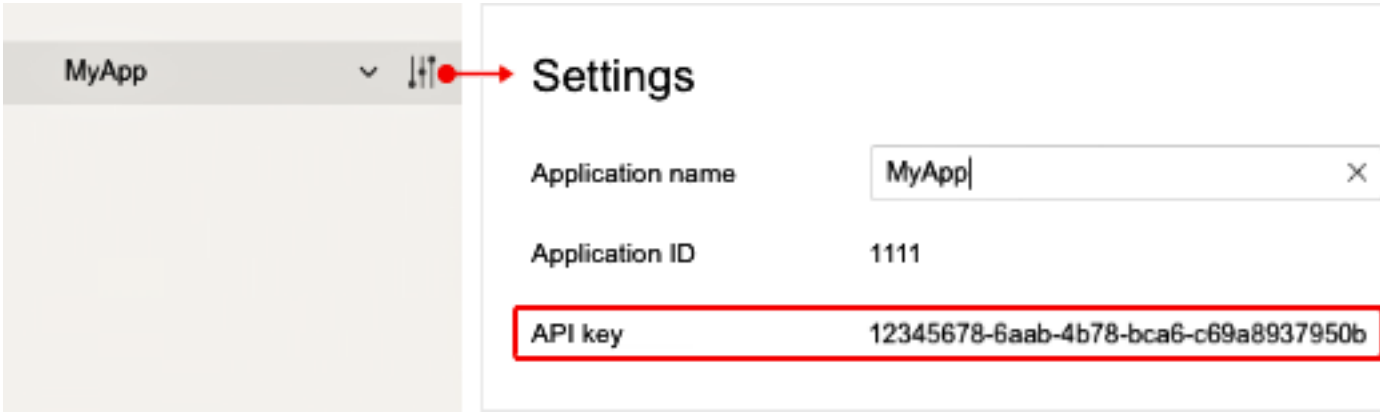
- Added support for extensions.
- Added support for Bitcode technology.
- Improved error messages.

Version 2.0

Released 27 August 2015

- The format of "Api key" has 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 Api key



- The method of initializing the library in the app has been renamed from `[YMMYandexMetrica startWithAPIKey:(NSString *)apiKey];` to `[YMMYandexMetrica activateWithApiKey:(NSString *)apiKey];`.
- Changed the length of the session timeout. Now it is 10 seconds, by default.
- The library has been adapted for iOS 9.
- Improved quality of calculating statistics for app installations and device identification for tracking.
- Significantly improved performance and reduced power consumption.
- Improved error messages.
- Deprecated methods have been removed.

  Click to see the list

  `+(void)setLogLevel:(NSUInteger)level;` — allows you to set the logging level.
  `+(void)setReportsEnabled:(BOOL)enabled` — Lets you enable and disable sending reports.
  `+(void)setDispatchPeriod(NSUInteger)dispatchPeriodSeconds;` — Allows you to set the interval in seconds between sending accumulated events to the server.
  `+(void)setMaxReportsCount(NSUInteger)maxReportsCount;` — Allows you to set the maximum number of events that can be stored up before sending all accumulated events to the server.
  `+(void)startNewSessionManually;` — allows you to start a new session manually.
  `+(void)sendEventsBuffer;` — Allows you to send all accumulated events without waiting for them to automatically be sent to the server.
  `+(BOOL)isReportsEnabled;` — Determines whether reports are sent.
  `+(BOOL)isReportCrashesEnabled;` — Determines whether app crashes are monitored.
  `+(BOOL)isTrackLocationEnabled;` — Determines whether location data is transmitted.
  `+(NSUInteger)sessionTimeoutSeconds;` — Determines what session length (in seconds) is set.
  `+(NSUInteger)maxReportsCount;` — Determines the maximum number of events in storage that is set. When this number is reached, all the accumulated events are sent to the server.
  `+(NSUInteger)dispatchPeriod;` — Determines the setting for the interval in seconds between sending accumulated events to the server.

Version 1.8.2

Released 18 June 2015

- Improved quality of calculating statistics for sessions and app installations.

Version 1.6.1

Released 12 November 2014

- Added free support for tracking app installations.
- Improved stability and performance.
- Switched to reading idfa from AdSupport within the library.

- Added events with additional parameters.
- Renamed the main library class from `YMMCounter` to `YMMYandexMetrica`.
- Changed the `ApiKey` type from `NSUInteger` to `NSString`.
- Migrated the library to XCode6 and iOS 8.
- Increased the minimal supported version to iOS 6.
- Improved location handling. Transmitting location can be disabled.
- Switched to asynchronous error processing using blocks.
- Optimized library start.