

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.

© 2008—2024 Yandex LLC. All rights reserved.

Copyright Disclaimer

Yandex (and its applicable licensor) has exclusive rights for all results of intellectual activity and equated to them means of individualization, used for development, support, and usage of the service AppMetrica. It may include, but not limited to, computer programs (software), databases, images, texts, other works and inventions, utility models, trademarks, service marks, and commercial denominations. The copyright is protected under provision of Part 4 of the Russian Civil Code and international laws.

You may use AppMetrica or its components only within credentials granted by the Terms of Use of AppMetrica or within an appropriate Agreement.

Any infringements of exclusive rights of the copyright owner are punishable under civil, administrative or criminal Russian laws.

Contact information

Yandex LLC

<https://www.yandex.com>

Tel.: +7 495 739 7000

Email: pr@yandex-team.ru

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

Contents

Push SDK integration.....	4
.....	4
Android.....	4
Installation and initialization.....	4
Configuring your app.....	8
Using with other push services.....	10
Reference.....	12
Migrating from GCM to Firebase.....	20
Changelog.....	21
iOS.....	24
Installation and initialization.....	24
Objective-C reference.....	31
Swift reference.....	37
Configuring interaction tracking.....	43
Uploading attached files.....	44
Configuring your app.....	46
Changelog.....	47
Windows.....	48
Installation and initialization.....	48
Configuring your app.....	49
Changelog.....	50
Plugins.....	50
Unity.....	50
Flutter.....	55
Cordova (not supported).....	56

AppMetrica Push SDK

The AppMetrica Push SDK is a set of libraries for working with push notifications. After enabling the AppMetrica Push SDK, you can [create and configure push notification campaigns](#), then monitor statistics in the AppMetrica web interface.

The SDK is available for the following platforms:

- Android (starting from version 5).
- iOS and iPadOS (starting from version 9).
- Windows (Development is on hold. We do not guarantee the SDK will work correctly.).

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.

Android

Connecting the AppMetrica Push SDK

The Push SDK for Android is a library in the AAR format. The library is available in the [Maven repository](#).

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

Step 1. Prepare your app

Before integrating the AppMetrica Push SDK library, prepare your app:

1. [Integrate the AppMetrica SDK library](#) at least version 3.0.0.
2. [Configure your app](#) for sending push notifications.

Step 2. Enable the library

Since version [1.4.0](#) to the Push SDK added the [OKHttp](#) library. The SDK uses it for caching images that are displayed in push notifications. Caching rules are taken from the `cache-control` HTTP header. If you do not want the images to be cached, connect the library without caching.

1. In the `build.gradle` file add the following dependencies in the `dependencies` block:

With caching

```
dependencies {
    ...
    implementation "com.yandex.android:mobmetricapushlib:2.3.2"
    implementation "androidx.legacy:legacy-support-v4:1.0.0"
    ...
}
```

Without caching

```
dependencies {
    ...
    implementation ("com.yandex.android:mobmetricapushlib:2.3.2") {
        exclude group: 'com.squareup.okhttp3', module: 'okhttp'
    }
    implementation "androidx.legacy:legacy-support-v4:1.0.0"
    ...
}}
```

2. Connect the transport.

Firebase

- a. In the `build.gradle` file, add Firebase dependencies in the `dependencies` block:

```
dependencies {
    ...
    // minimum support version 20.3.0
    implementation "com.google.firebase:firebase-messaging:22.0.0"
    implementation "com.google.android.gms:play-services-base:17.5.0"
    ...
}
```

- b. Initialize Firebase using one of the following methods:

Using Google Services Plugin

1. [Download](#) the configuration file `google-services.json` and put it in the project module's directory (such as `app`).
2. In order to work with the file correctly, enable the Google Services plugin in the project by adding the following lines to the `build.gradle` file:

project

```
buildscript{
    ...
    dependencies {
        classpath 'com.google.gms:google-services:4.3.4'
        ...
    }
    ...
}
```

application (module)

```
// In the end of the file.
apply plugin: 'com.google.gms.google-services'
```

Without using the plugin

Make changes in the application element in the `AndroidManifest.xml` file:

```
<meta-data android:name="ymp_firebase_default_app_id" android:value="APP_ID"/>
<meta-data android:name="ymp_gcm_default_sender_id" android:value="number:SENDER_ID"/>
<meta-data android:name="ymp_firebase_default_api_key" android:value="API_KEY"/>
<meta-data android:name="ymp_firebase_default_project_id" android:value="PROJECT_ID"/>
```

`APP_ID` — ID of the app in Firebase. You can find it in the [Firebase console](#): go to the **Project settings**. In the **Your application** section copy the value of the **application ID** field.

`SENDER_ID` — The unique ID of the sender in Firebase. You can find it in the [Firebase console](#): go to **Project settings** → **Cloud Messaging** and copy the value of the **Sender ID** field.

`API_KEY` — App key in Firebase. You can find it in the `current_key` field of the `google-services.json` file. You can download the file in the [Firebase console](#).

`PROJECT_ID` — App ID in Firebase. You can find it in the `project_id` field of the `google-services.json` file. You can download the file in the [Firebase console](#).

Using with other Firebase projects

Make changes in the application element in the `AndroidManifest.xml` file:

```
<meta-data android:name="ymp_firebase_app_id" android:value="APP_ID"/>
<meta-data android:name="ymp_gcm_sender_id" android:value="number:SENDER_ID"/>
<meta-data android:name="ymp_firebase_api_key" android:value="API_KEY"/>
```

```
<meta-data android:name="ymp_firebase_project_id" android:value="PROJECT_ID" />
```

APP_ID — ID of the app in Firebase. You can find it in the [Firebase console](#): go to the **Project settings**. In the **Your application** section copy the value of the **application ID** field.

SENDER_ID — The unique ID of the sender in Firebase. You can find it in the [Firebase console](#): go to **Project settings** → **Cloud Messaging** and copy the value of the **Sender ID** field.

API_KEY — App key in Firebase. You can find it in the `current_key` field of the `google-services.json` file. You can download the file in the [Firebase console](#).

PROJECT_ID — App ID in Firebase. You can find it in the `project_id` field of the `google-services.json` file. You can download the file in the [Firebase console](#).



Attention: You should initialize default Firebase project.

HMS

- a. Add [HMS Push Kit](#) to the project.
- b. In the `build.gradle` file add the following dependencies in the `dependencies` block:

```
dependencies {
    ...
    implementation "com.yandex.android:appmetricapush-provider-hms:2.3.2"
    ...
}
```



Attention: If you're only going to use HMS, exclude the Firebase dependency from the main library:

```
dependencies {
    ...
    implementation ("com.yandex.android:mobmetricapushlib:2.3.2") {
        exclude group: 'com.yandex.android', module: 'appmetricapush-provider-firebase'
    }
    ...
}
```

- c. Make changes in the application element in the `AndroidManifest.xml` file:

```
<meta-data android:name="ymp_hms_default_app_id" android:value="number:APP_ID" />
```

APP_ID — App ID in HMS. You can find it in the `app_id` field of the `agconnect-services.json` file. You can download the file in the [Huawei console](#).

The AppMetrica Push SDK can simultaneously work with Firebase and HMS.

Step 3. Initialize the library

Initialize the library in the app — extend the `Application` class and override the `onCreate()` method as follows:

Firebase only

Java

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        YandexMetricaPush.init(getApplicationContext());
    }
}
```

Kotlin

```
class MyApp : Application() {
    override fun onCreate() {
        super.onCreate()
        YandexMetricaPush.init(applicationContext)
    }
}
```

HMS only

Java

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        YandexMetricaPush.init(getApplicationContext(), new HmsPushServiceControllerProvider(this));
    }
}
```

```
    }
}
```

Kotlin

```
class MyApp : Application() {
    override fun onCreate() {
        super.onCreate()
        YandexMetricaPush.init(applicationContext, HmsPushServiceControllerProvider(this))
    }
}
```

Firebase and HMS**Java**

```
public class MyApp extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        YandexMetricaPush.init(
            getApplicationContext(),
            new FirebasePushServiceControllerProvider(this),
            new HmsPushServiceControllerProvider(this)
        );
    }
}
```

Kotlin

```
class MyApp : Application() {
    override fun onCreate() {
        super.onCreate()
        YandexMetricaPush.init(
            applicationContext,
            FirebasePushServiceControllerProvider(this),
            HmsPushServiceControllerProvider(this)
        )
    }
}
```



Attention: The AppMetrica Push SDK library must be initialized in the main process.

Step 4. (Optional) Configure Silent Push Notifications

Configure processing silent push notifications.

1. Create a special BroadcastReceiver:

```
import com.yandex.metrica.push.YandexMetricaPush;
public class SilentPushReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        // Extract push message payload from your push message.
        String payload = intent.getStringExtra(YandexMetricaPush.EXTRA_PAYLOAD);
        ...
    }
}
```

2. Make changes in the application element in the AndroidManifest.xml file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application>
    ...
    <receiver android:name=".SilentPushReceiver">
      <intent-filter>
        <!-- Receive silent push notifications. -->
        <action android:name="{applicationId}.action.ymp.SILENT_PUSH_RECEIVE" />
      </intent-filter>
    </receiver>
    ...
  </application>
</manifest>
```

applicationId — Unique app ID in Gradle (package name). For example, com.example.name.

Step 5. (Optional) Enable push tokens update

Attention:

If you have your own service to handle push notifications (a class inherited from `FirebaseMessagingService` or `HmsMessageService`), check that you are not handling push notifications from AppMetrica.

To check that the push notification is not from AppMetrica, use the [MetricaMessagingService.isNotificationRelatedToSDK](#) or [MetricaHmsMessagingService.isNotificationRelatedToSDK](#) method.

The FCM service can withdraw the push token of the device, for example, if the user did not launch the application for a long time. AppMetrica stores push tokens on the server and can not send a push notification to a device with an obsolete token.

To automatically collect current push token go to the application settings in the AppMetrica interface and enable the **Update tokens with a Silent Push notification** option in the **Push Notifications** tab.

Sending additional information

You can send additional information with the push notification if necessary. This data is specified in the AppMetrica web interface when [configuring the push campaign](#).

```
public class TargetActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        handlePayload(getIntent());
    }

    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        handlePayload(intent);
    }

    private void handlePayload(Intent intent) {
        // Handle your payload.
        String payload = intent.getStringExtra(YandexMetricaPush.EXTRA_PAYLOAD);
        ...
    }
}
```

Detecting the application launch via push notification

To distinguish app launches initiated by opening an AppMetrica push notification from the total number of app starts, you should check the [Intent action](#) of the app. If you specified a deeplink as the action, this will be the Intent action. If the action is set as opening the app, the Intent action passes the value `YandexMetricaPush#OPEN_DEFAULT_ACTIVITY_ACTION`.

Setting the default icon

To set the default push notification icon, make changes in the application element in the `AndroidManifest.xml` file:

```
<meta-data android:name="com.yandex.metrica.push.default_notification_icon"
    android:resource="@drawable/large_icon" />
```

`ICON_RESOURCE` — Icon resource. For example, `@drawable/large_icon`.

See also

[Configuring an Android application to send push notifications](#) on page 8

[How can I make sure that I have the latest versions of the Android libraries installed?](#)

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.

Configuring an Android application to send push notifications

For using Firebase

Step 1. Create a project in Firebase

1. Go to the [Firebase Console](#) and choose an action:

Create a new project (if this is your first project)

- a. Enter a name for the project.
- b. Select the country your organization is officially registered in and click **Create project**.



Import a Google project (if you used Google APIs to create a project)

- a. In the drop-down list, select the name of the project you are planning to run push campaigns for.
- b. Select the country your organization is officially registered in and click **Add Firebase**.

2. Click **Add Firebase to your Android app** and follow the instructions.

Step 2. Configure AppMetrica to work with FCM

Get a server key for using Firebase Cloud Messaging:

1. In the Firebase Console, select the project you are planning to run push campaigns for.
2. In the menu on the left, click  next to the project name and go to the **Project Settings** section.
3. Go to the **Cloud Messaging** tab.
4. In the **Cloud Messaging API** block, select  → **Manage API in Google Cloud Console** → **Enable**.
5. Go back to the **Cloud Messaging** tab.
6. In the **Cloud Messaging API** block, copy the value of the **Server key** field.

Use this key in the AppMetrica interface:

1. In the [Applications](#) section, select the app that you want to run push campaigns for.
2. In the menu on the left, select **Settings**.
3. Go to the **Push notifications** tab.
4. In the **Android** section, enter the value you copied from the Firebase Console in **Server key** and click **Submit**.

To use Huawei Mobile Services (HMS)

Step 1. Create and configure a project in the Huawei console

Follow all the steps in the [Huawei](#) documentation.



Attention: Make sure that [Configuring the Signing Certificate Fingerprint](#) specifies SHA-256 certificate fingerprint for all app signatures, including the debug version. Otherwise, the device can't receive push notifications.

Step 2. Configure AppMetrica to work with HMS

Get the app ID and app secret from the [Huawei console](#):

1. In the [list of projects](#) in the Huawei console, select your project.
2. On the project page, copy the App ID and App secret field values.

Use these keys in the AppMetrica interface:

1. In the [Applications](#) section, select the app that you want to run push campaigns for.
2. In the menu on the left, select **Settings**.
3. Go to the **Push notifications** tab.
4. In the **Huawei** block, fill in the **App ID** and **App secret** fields with values from the Huawei console and click **Submit**.

See also

[Connecting the AppMetrica Push SDK](#) on page 4

Launching a push campaign

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.

Using with other push services

You can use AppMetrica Push SDK and other push services at the same time. To do this, you need to create a FCM or HMS service that will redirect messages between integrated SDKs.

Using with other Firebase push services

Step 1. Make changes in AndroidManifest.xml

Make changes in the application element in the `AndroidManifest.xml` file:

```
<service android:name=".FirebaseMessagingMasterService"
    android:enabled="true"
    android:exported="false">
    <intent-filter android:priority="100">
        <action android:name="com.google.firebase.MESSAGING_EVENT"/>
    </intent-filter>
</service>
<service android:name="com.yandex.metrica.push.firebase.MetricaMessagingService" tools:node="remove"/>
```

Step 2. Add push notifications handling

Declare the derived `FirebaseMessagingMasterService` class from the base `FirebaseMessagingService` class for handling push notifications:

Java

```
public class FirebaseMessagingMasterService extends FirebaseMessagingService {
    @Override
    public void onMessageReceived(RemoteMessage message) {
        super.onMessageReceived(message);
        if (MetricaMessagingService.isNotificationRelatedToSDK(message) {
            new MetricaMessagingService().processPush(this, message);
            return;
        }
        // Implement the logic for sending messages to other SDKs or handle own pushes.
    }
}
```

Kotlin

```
class FirebaseMessagingMasterService : FirebaseMessagingService() {
    override fun onMessageReceived(message: RemoteMessage) {
        super.onMessageReceived(message)
        if (MetricaMessagingService.isNotificationRelatedToSDK(message) {
            MetricaMessagingService().processPush(this, message)
            return
        }
        // Implement the logic for sending messages to other SDKs or handle own pushes.
    }
}
```

Step 3. Add push token processing

Add push token processing to the `FirebaseMessagingMasterService` class code:

Java

```
public class FirebaseMessagingMasterService extends FirebaseMessagingService {
    ...
    @Override
    public void onNewToken(@NonNull String token) {
        super.onNewToken(token);
        new MetricaMessagingService().processToken(this, token);
        // Implement the logic for sending tokens to other SDKs.
    }
}
```

Kotlin

```
class FirebaseMessagingMasterService : FirebaseMessagingService() {
    ...

    override fun onNewToken(token: String) {
        super.onNewToken(token);
        MetricaMessagingService().processToken(this, token)

        // Implement the logic for sending tokens to other SDKs.
    }
}
```

Using with other HMS push services**Step 1. Make changes in AndroidManifest.xml**

Make changes in the application element in the `AndroidManifest.xml` file:

```
<service
    android:name=".HmsMessagingMasterService"
    android:exported="true"
    android:permission="${applicationId}.permission.PROCESS_PUSH_MSG">
    <intent-filter android:priority="100">
        <action android:name="com.huawei.push.action.MESSAGING_EVENT" />
    </intent-filter>
</service>
<service android:name="com.yandex.appmetrica.push.hms.MetricaHmsMessagingService" tools:node="remove"/>
```

Step 2. Add push notifications handling

Declare the derived `HmsMessagingMasterService` class from the base `HmsMessageService` class for handling push notifications:

Java

```
public class HmsMessagingMasterService extends HmsMessageService {
    @Override
    public void onMessageReceived(RemoteMessage message) {
        super.onMessageReceived(message);
        if (MetricaHmsMessagingService.isNotificationRelatedToSDK(message) {
            new MetricaHmsMessagingService().processPush(this, message);
            return;
        }

        // Implement the logic for sending messages to other SDKs or handle own pushes.
    }
}
```

Kotlin

```
class HmsMessagingMasterService : HmsMessageService() {
    override fun onMessageReceived(message: RemoteMessage) {
        super.onMessageReceived(message)
        if (MetricaHmsMessagingService.isNotificationRelatedToSDK(message) {
            MetricaHmsMessagingService().processPush(this, message)
            return
        }

        // Implement the logic for sending messages to other SDKs or handle own pushes.
    }
}
```

Step 3. Add push token processing

Add push token processing to the `HmsMessagingMasterService` class code:

Java

```
public class HmsMessagingMasterService extends HmsMessageService {
    ...

    @Override
    public void onNewToken(@Nullable String token) {
        super.onNewToken(token);
        new MetricaHmsMessagingService().processToken(this, token);

        // Implement the logic for sending tokens to other SDKs.
    }
}
```

Kotlin

```
class HmsMessagingMasterService : HmsMessageService() {
```

```
...
    override fun onNewToken(token: String?) {
        super.onNewToken(token);
        MetricaHmsMessagingService().processToken(this, token)

        // Implement the logic for sending tokens to other SDKs.
    }
}
```

See also

[Connecting the AppMetrica Push SDK on page 4](#)

[Launching a push campaign](#)

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**com.yandex.metrika.push****Interfaces**

TokenUpdateListener interface

com.yandex.metrika.push

```
public interface TokenUpdateListener
```

The interface defines callback methods when updating the token.

To subscribe to token updates, create an instance that implements the `TokenUpdateListener` interface and pass it to the `YandexMetrikaPush.setTokenUpdateListener(TokenUpdateListener listener)` method.

Methods

```
void onTokenUpdated(Map<String, String> tokens)
```

Called when the first token is received or updated.

Method descriptions

onTokenUpdated

```
void onTokenUpdated(Map<String, String> tokens)
```

Called when the first token is received or updated.

Parameters:

tokens

List of tokens for push providers that the AppMetrica Push SDK was initialized with. The keys can take the values `firebase` and `hms`.

Classes

YandexMetrikaPush class

com.yandex.metrika.push

```
public final class YandexMetrikaPush
```

Methods of the class are used for configuring the Push SDK library.

Methods

void [init](#)(@NonNull final [Context](#) ctx)

Initializes the library in the app. Method should be invoked after AppMetrica SDK initialization.

void [init](#)(@NonNull Context ctx, PushServiceControllerProvider... providers)

Initializes the library in the app with a list of push transports. Method should be invoked after AppMetrica SDK initialization.

String [getToken](#)()



Attention: Deprecated method. Use the [getTokens](#)() method instead.

Returns the push token.

Map<String, String> [getTokens](#)()

Returns a list of tokens for push providers that AppMetrica Push SDK was initialized with. The keys can take the values `firebase` and `hms`.

[NotificationChannel](#) [getDefaultNotificationChannel](#)()

Returns the push notification channel `NotificationChannel`, which is used by default. You can set settings for it using the methods [NotificationChannel](#) until the first push notification is received.

void [setTokenUpdateListener](#)(@NonNull [TokenUpdateListener](#) listener)

Subscribes to token updates.

Fields

String	OPEN_DEFAULT_ACTIVITY_ACTION	Intent action for execute the Activity by default. It can be used to detect the app start via AppMetrica push notification.
String	EXTRA_PAYLOAD	An arbitrary data string that is passed in the push notification: <ul style="list-style-type: none"> In the Additional data field when sending from the AppMetrica interface. In the <code>data</code> field when sending using the Push API.

Method descriptions

init

void [init](#)(@NonNull final [Context](#) ctx)

Initializes the library in the app. Method should be invoked after AppMetrica SDK initialization.

setTokenUpdateListener

```
void setTokenUpdateListener(@NonNull TokenUpdateListener listener)
```

Subscribes to token updates.

Parameters:

listener

The instance of the [TokenUpdateListener](#) class. It's called when the first token is received or updated.

Field descriptions

OPEN_DEFAULT_ACTIVITY_ACTION

```
public final String OPEN_DEFAULT_ACTIVITY_ACTION =
"com.yandex.metrika.push.action.OPEN"
```

[Intent action](#) for execute the [Activity](#) by default. It can be used to detect the app start via AppMetrica push notification.

```
public class LaunchActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent intent = getIntent();
        String action = intent.getAction();
        if (YandexMetricaPush.OPEN_DEFAULT_ACTIVITY_ACTION.equals(action)) {
            // Handle the app start via AppMetrica push notification.
            ...
        }
    }
}
```

EXTRA_PAYLOAD

```
public final String EXTRA_PAYLOAD = ".extra.payload"
```

An arbitrary data string that is passed in the push notification:

- In the **Additional data** field when sending from the AppMetrica interface.
- In the data field when sending using the [Push API](#).

com.yandex.appmetrica.push.firebase**Classes**

FirebasePushServiceControllerProvider class

```
com.yandex.appmetrica.push.firebase
```

```
public class FirebasePushServiceControllerProvider
```

A class that helps initialize AppMetrica Push SDK with Firebase.

Constructors

```
FirebasePushServiceControllerProvider(@NonNull Context ctx)
```

Constructor descriptions

FirebasePushServiceControllerProvider

```
public FirebasePushServiceControllerProvider(@NonNull Context ctx)
```

Parameters:

ctx

The instance of the Context class.

com.yandex.appmetrica.push.hms**Classes**

HmsPushServiceControllerProvider class

```
com.yandex.appmetrica.push.hms
```

```
public class HmsPushServiceControllerProvider
```

A class that helps initialize the AppMetrica Push SDK with HMS.

Constructors

```
HmsPushServiceControllerProvider(@NonNull Context ctx)
```

Constructor descriptions

HmsPushServiceControllerProvider

```
public HmsPushServiceControllerProvider(@NonNull Context ctx)
```

Parameters:

ctx	The instance of the Context class.
-----	------------------------------------

MetricaHmsMessagingService class

```
com.yandex.appmetrica.push.hms
```

```
public class MetricaHmsMessagingService
```

Methods of the class are used to configure simultaneous use of the AppMetrica Push SDK and another push service. For more information, see [Using with other push services](#).

Methods

```
void onMessageReceived(@NonNull final RemoteMessage message)
```

Called when a push notification comes from Firebase. For more information, see the [HmsMessageService.onMessageReceived](#) method description.

```
void onNewToken(@NonNull String token)
```

Called when the system generates a new push token. For more information, see the [HmsMessageService.onNewToken\(String token\)](#) method description.

```
void processPush(@NonNull final Context context, @NonNull final RemoteMessage message)
```

Sends information about push notification to AppMetrica Push SDK. AppMetrica automatically recognizes own messages and processes only them.

Sends information about push notification to AppMetrica Push SDK. AppMetrica automatically recognizes own messages and processes only them.

This method must be called:

- [When you simultaneously use](#) the AppMetrica Push SDK and HMS.
- If you need to change push notification data. If you don't need to change data, use the method `processPush(@NonNull final Context context, @NonNull final RemoteMessage message)`.

Parameters:

context	The instance of the Context class.
message	The instance of the <code>Bundle</code> class. To get it, convert data from <code>RemoteMessage.getData()</code> to <code>Bundle</code> .

processToken

```
public void processToken(@NonNull final Context context, @NonNull final String token)
```

Sends information about a push token to AppMetrica Push SDK.

This method must be called [when you simultaneously use](#) AppMetrica Push SDK and Firebase Cloud Messaging.

Parameters:

context	The instance of the Context class.
token	Token used for sending push notifications.

isNotificationRelatedToSDK

```
static boolean isNotificationRelatedToSDK(@NonNull final RemoteMessage notification)
```

Checks whether the push notification received belongs to the AppMetrica PushSDK.

Parameters:

notification	The instance of RemoteMessage .
--------------	---

com.yandex.metrica.push.firebase

Classes

MetricaMessagingService class

```
com.yandex.metrica.push.firebase
```

```
public class MetricaMessagingService extends FirebaseMessagingService
```

Methods of the class are used to configure simultaneous use of the AppMetrica Push SDK and another push service. For more information, see [Using with other push services](#).

Methods

```
void onMessageReceived(@NonNull final RemoteMessage message)
```

Called when a push notification is received from Firebase. For more information, see the method description [FirebaseMessagingService.onMessageReceived](#).

processPush

```
public void processPush(@NonNull final Context context, @NonNull final RemoteMessage message)
```

Sends information about push notification to AppMetrica Push SDK. AppMetrica automatically recognizes own messages and processes only them.

This method must be called [when you simultaneously use](#) AppMetrica Push SDK and Firebase Cloud Messaging.

Parameters:

context	The instance of the Context class.
message	The instance of RemoteMessage .

processPush

```
public void processPush(@NonNull final Context context, @NonNull final Bundle data)
```

Sends information about push notification to AppMetrica Push SDK. AppMetrica automatically recognizes own messages and processes only them.

This method must be called:

- [When you use simultaneously](#) AppMetrica Push SDK and Firebase Cloud Messaging;
- If you need to change push notification data. If you don't need to change data, use the method [processPush\(@NonNull final Context context, @NonNull final RemoteMessage message\)](#).

Parameters:

context	The instance of the Context class.
message	The instance of the Bundle class. To get it, convert data from RemoteMessage.getData() to Bundle .

processToken

```
public void processToken(@NonNull final Context context, @NonNull final String token)
```

Sends information about a push token to AppMetrica Push SDK.

This method must be called [when you simultaneously use](#) AppMetrica Push SDK and Firebase Cloud Messaging.

Parameters:

context	The instance of the Context class.
token	Token used for sending push notifications.

isNotificationRelatedToSDK

```
static boolean isNotificationRelatedToSDK(@NonNull final RemoteMessage notification)
```

Checks whether the push notification received belongs to the AppMetrica PushSDK.

Parameters:

notification	The instance of RemoteMessage .
--------------	---

Migrating from GCM to Firebase

Since the AppMetrica Push SDK version [1.0.0](#), it uses the Firebase Cloud Messaging (FCM) service to send push notifications.

This section explains the steps for migrating the application from Google Cloud Messaging to Firebase Cloud Messaging.

Step 1. Import a project

Import a Google project (if you used Google APIs to create a project):

1. Go to the [Firebase console](#).
2. Click the **Add project** button.
3. In the drop-down list, select the name of the project you are planning to run push campaigns for.
4. Select the country your organization is officially registered in and click **Add Firebase**.
5. Click **Add Firebase to your Android app** and follow the instructions.

Step 2. Configuring your app

Edit the `build.gradle` file of the application (module):

1. Delete the following dependency:

```
dependencies {  
    ...  
    compile "com.google.android.gms:play-services-gcm:${versionGcm}"  
    ...  
}
```

2. Add the following dependencies:

```
dependencies {  
    ...  
    compile "com.google.firebase:firebase-messaging:19.0.1"  
    compile "com.google.android.gms:play-services-base:16.1.0"  
    compile "androidx.legacy:legacy-support-v4:1.0.0"  
    ...  
}
```

3. Update the AppMetrica library version:

```
compile "com.yandex.android:mobmetricapushlib:APPMETRICIA_VERSION"
```

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 2.3.2

Released October 4, 2023

- Fixed the No field Companion of type Lio/appmetrica/analytics/ModuleEvent\$Companion error.

Version 2.3.1

Released July 31, 2023

- Improved performance and stability of the library.

Version 2.2.0

Released June 2, 2022

- `minSdkVersion` (in `appmetricapush-provider-hms`) increased to 17.
- Updated the `hms` version to 6.5.0.300.
- Accelerated PushSDK activation.

Version 2.1.1

Released 14 March 2022

- Fixed a bug `BadParcelableException: Parcelable protocol requires a Parcelable.Creator object called CREATOR`.

Version 2.1.0

Released February 24, 2022

- Improved token acquisition.
- The `YandexMetricaPush.getTokens` method returns new tokens when called from `TokenUpdateListener.onTokenUpdated`.
- Improved performance and stability of the library.

Version 2.0.0

Released 28 October 2021

- Fixed a bug that occasionally occurred when processing a click on a push in Android 12.
- Switching from support dependencies to AndroidX.

Version 1.14.0

Released 23 September 2021

- AppMetrica version updated to 4.0.0.
- Added support for Android 12.
- Improved performance and stability of the library.

Version 1.13.0

Released 22 July 2021

- Added support for `com.google.firebase:firebase-messaging:22.0.0`. Minimum supported version: `20.3.0`.

Version 1.12.0

Released July 7, 2021

- Fixed a possible cause for rejecting an app's publication in Google Play due to [Implicit PendingIntent](#).

Version 1.11.1

Released May 14, 2021

- Fixed a build issue when multiple push providers connect at the same time.

Version 1.11.0

Released April 28, 2021

- Added the [isNotificationRelatedToSDK](#) method. You can use this method to determine whether the push notification received belongs to the AppMetrica PushSDK.
- Added the [processToken](#) method. This method passes the push token to AppMetrica when [AppMetrica PushSDK is used together with other SDKs](#).
- Improved performance and stability of the library.

Version 1.10.0

Released 15 December 2020

- Added support of HMS push notifications. For more information, see [Installation and initialization](#) and [Configuring your app](#).
- Stopped supporting the `YandexMetricaPush.getToken()` method. Use the `YandexMetricaPush.getTokens()` method instead.
- Added the [TokenUpdateListener](#) interface.

- Added methods:
 - [YandexMetricaPush.init\(Context ctx, PushServiceControllerProvider... providers\)](#).
 - [YandexMetricaPush.getTokens\(\)](#).
 - [YandexMetricaPush.setTokenUpdateListener\(TokenUpdateListener listener\)](#).
- The manifest supports the new required Firebase ID: `ymp_firebase_default_project_id`.
- Deleted request for permission to track location [FINE_LOCATION](#).
- Improved performance and stability of the library.

Version 1.5.1

Released 3 September 2019

- Fixed an issue that led to the [JobIntentService](#) crash.

Version 1.5.0

Release 8 August 2019

- Added the feature for [setting default push notification icon](#) .
- Stop supporting the class [MetricaInstanceIdService](#). Use the [MetricaMessagingService.onNewToken\(String token\)](#) method to update the token.

Version 1.4.1

Released 18 March 2019

- Fixed the obfuscation problem.

Version 1.4.0

Released 14 March 2019

- Added images caching using the library [OKHttp](#).
- Added dependency on `com.squareup.okhttp3:okhttp` version 3.12.1.
- Added the following rule: AppMetrica Push SDK you can initialize only in the main process. Initialization in other processes is ignored.
- Fixed initialization of the Push SDK with `com.google.firebase:firebase-auth`.
- Added real-time notification status tracking (for Android 9).

Version 1.3.0

Released 1 October 2018

- Added the method `YandexMetricaPush.getDefaultNotificationChannel()` to receive the push notification channel that is used by default. You can set settings for it using the methods [NotificationChannel](#) until the first push notification is received.
- Changed the default channel priority from [IMPORTANCE_DEFAULT](#) (with sound) to [IMPORTANCE_LOW](#) (without sound).

Version 1.2.0

Released 20 August 2018

- Added support of [push notification channels](#) (for Android 8 and higher).
- Added tracking of disabled notifications.

Version 1.1.0

Released 15 May 2018

- Added support for [AppMetrica SDK 3.0.0](#).

Version 1.0.0

Released 15 February 2018

- Migrated from GCM to Firebase.
- Fixed an issue with PackageManager.
- Updated minimal versions of the following components:
 - Support Library is **26.0.0**.
 - Android API is **14**.

Version 0.6.1

Released 31 October 2017

- Added support for Android 8.

Version 0.5.0

Released 21 January 2017

- When opening the [default launcher activity](#) from a push notification, the [Intent action](#) is added from `YandexMetricaPush#OPEN_DEFAULT_ACTIVITY_ACTION`.

Version 0.4.0

Released 21 November 2016

- Added support for push notification styles: [BigTextStyle](#) and [BigPictureStyle](#).
- Added support for configuring sound alerts when push notifications are received.

Version 0.3.0

Released 10 October 2016

- Integrated with the AppMetrica SDK library.

iOS

Connecting the AppMetrica Push SDK

Before using the AppMetrica Push SDK 1.3.0, you need to [enable and initialize the AppMetrica SDK](#) version 3.4.1 or higher.

Step 1. Enable the library

The library can work with the following dependency managers:

CocoaPods

The library supports static and dynamic frameworks for CocoaPods. To enable the library, add a dependency to the project's Podfile:

- Static framework

```
pod 'YandexMobileMetricaPush', '1.3.0'
```

- Dynamic framework

```
pod 'YandexMobileMetricaPush/Dynamic', '1.3.0'
```

Carthage

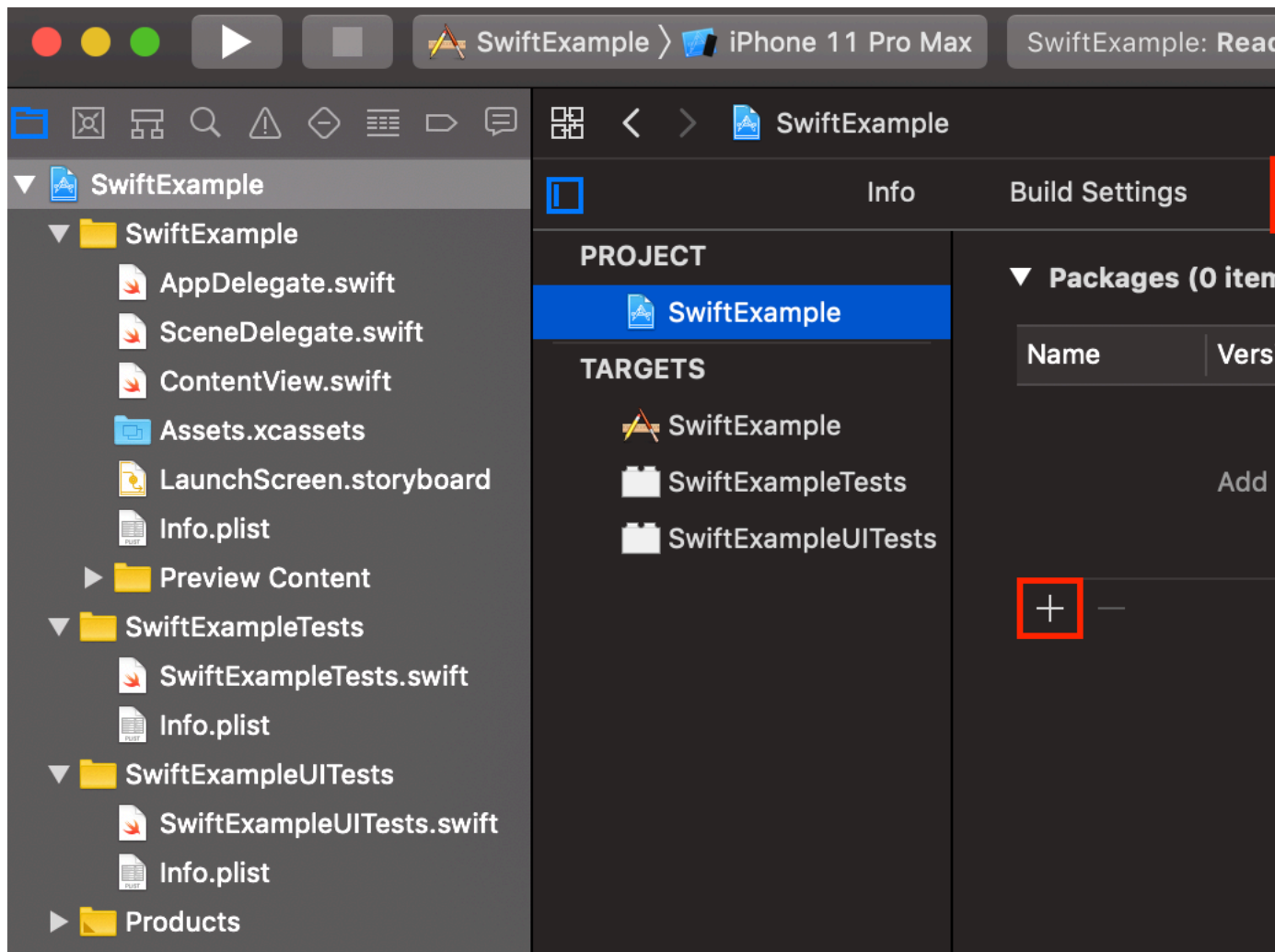
To enable the library, add the following dependency to the `Cartfile` and save the file:

```
binary "https://raw.githubusercontent.com/yandexmobile/metrica-push-sdk-ios/master/YandexMobileMetricaPush.json"
  ~> 1.3.0
```

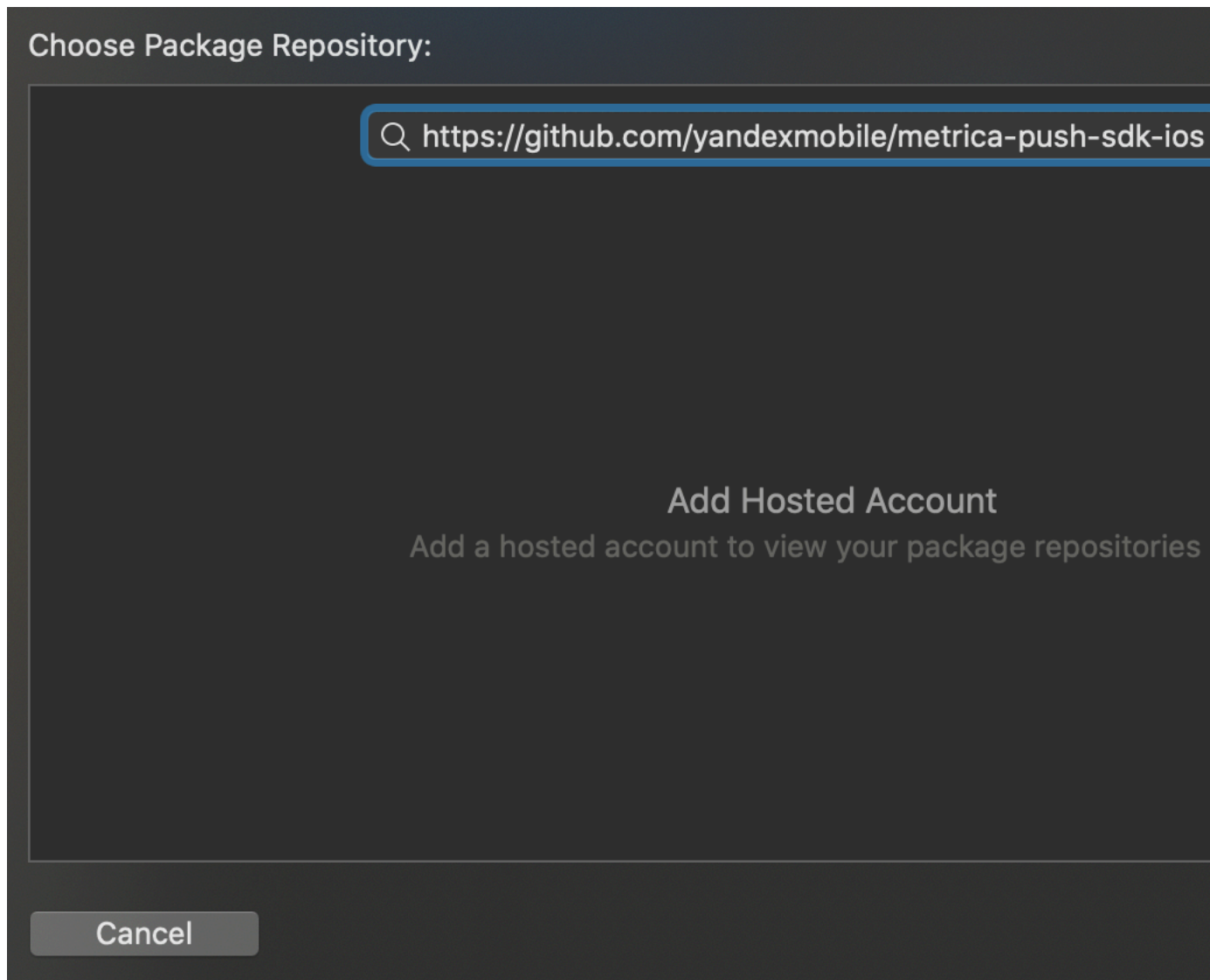
Swift Package Manager

To connect the library, follow these steps:

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



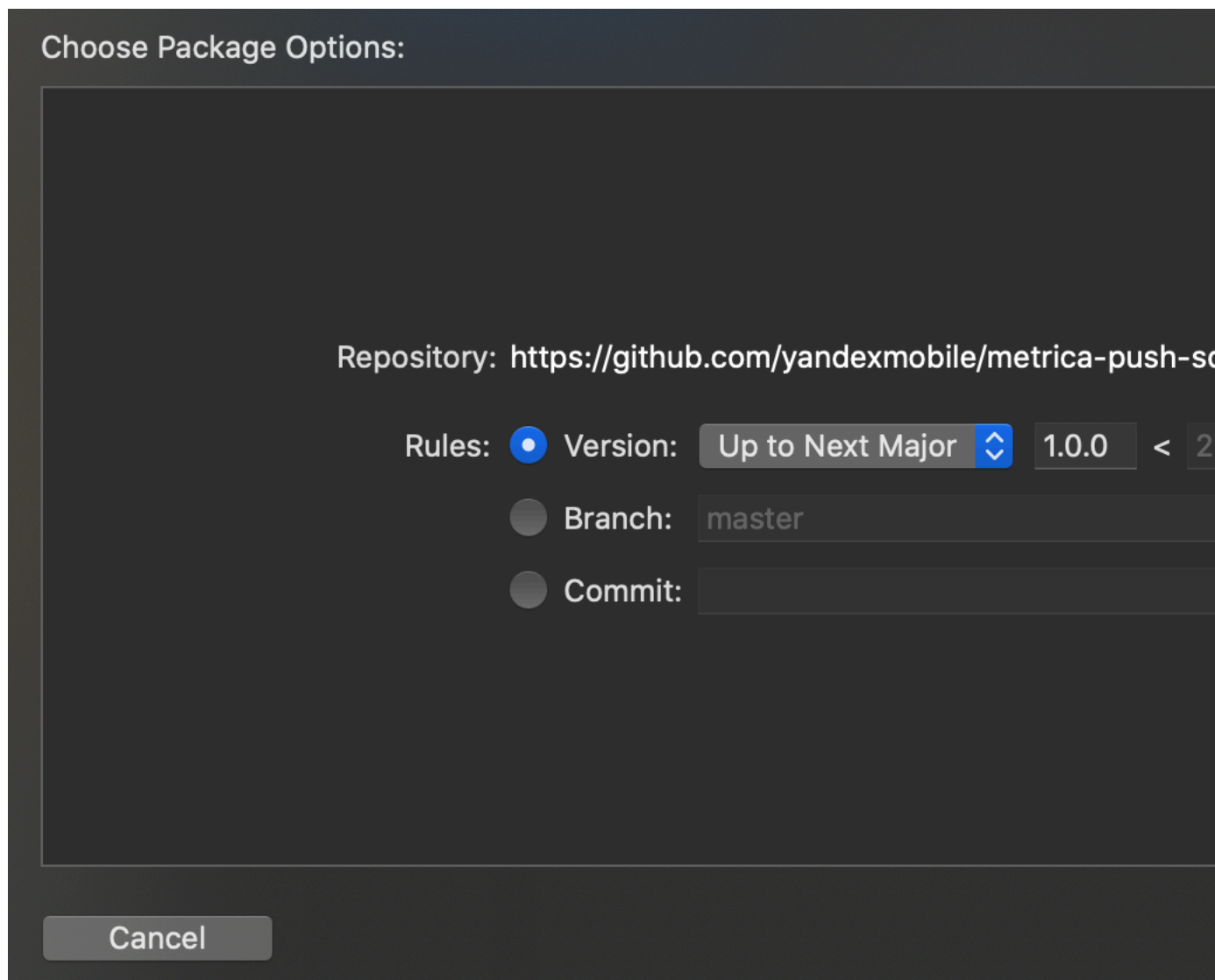
2. Specify the repository URL `https://github.com/yandexmobile/metrica-push-sdk-ios`, which contains a Swift package.



3. Configure a rule for selecting the package version.

Restriction:

Connection using Swift Package Manager is supported starting from version 1.0.0 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 Push library.](#)
2. Add `YandexMobileMetricaPush.framework` to the project.

Note: The AppMetrica SDK and AppMetrica Push SDK libraries must both be enabled in one of these ways.

Step 2. Register your app in the Apple Push Notification Service (APNs)

Registration prepares the app to work with push notifications. To send notifications to devices with iOS version 7 and higher, make the following changes to the application code:

Swift

```
// Register for push notifications
if #available(iOS 10.0, *) {
    // iOS 10.0 and above.
    let center = UNUserNotificationCenter.current()
    center.requestAuthorization(options:[.badge, .alert, .sound]) { (granted, error) in
        // Enable or disable features based on authorization.
    }
} else {
```

```
// iOS 8 and iOS 9.
let settings = UIUserNotificationSettings(types: [.badge, .alert, .sound], categories: nil)
application.registerUserNotificationSettings(settings)
}
application.registerForRemoteNotifications()
```

This data is usually passed in the following method:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
    launchOptions: [UIApplicationLaunchOptionsKey :Any]? = nil) -> Bool
```

For more details about the methods used, see the documentation at developer.apple.com:

[UNUserNotificationCenter.current\(\)](#)
[UNUserNotificationCenter.requestAuthorization\(options:completionHandler:\)](#)
[UIUserNotificationSettings\(types:categories:\)](#)
[UIApplication.registerUserNotificationSettings\(_:\)](#)
[UIApplication.registerForRemoteNotifications\(\)](#)

Objective-C

```
if ([application respondsToSelector:@selector(registerForRemoteNotifications)]) {
    if (NSClassFromString(@"UNUserNotificationCenter") != Nil) {
        // iOS 10.0 and above
        UNAuthorizationOptions options =
            UNAuthorizationOptionAlert |
            UNAuthorizationOptionBadge |
            UNAuthorizationOptionSound;
        UNUserNotificationCenter *center = [UNUserNotificationCenter currentNotificationCenter];
        [center requestAuthorizationWithOptions:options completionHandler:^(BOOL granted, NSError *error) {
            // Enable or disable features based on authorization.
        }];
    }
    else {
        // iOS 8 and iOS 9
        UIUserNotificationType userNotificationTypes =
            UIUserNotificationTypeAlert |
            UIUserNotificationTypeBadge |
            UIUserNotificationTypeSound;
        UIUserNotificationSettings *settings =
            [UIUserNotificationSettings settingsForTypes:userNotificationTypes categories:nil];
        [application registerUserNotificationSettings:settings];
    }
    [application registerForRemoteNotifications];
}
```

This data is usually passed in the following method:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
```

For more details about the methods used, see the documentation at developer.apple.com:

[\[UNUserNotificationCenter currentNotificationCenter\]](#)
[\[UNUserNotificationCenter requestAuthorizationWithOptions:completionHandler:\]](#)
[\[UIUserNotificationSettings settingsForTypes:categories:\]](#)
[\[UIApplication registerUserNotificationSettings\]](#)
[\[UIApplication registerForRemoteNotifications\]](#)

Step 3. Register a device token for your app

To send push notifications using AppMetrica, your app's device token is required. To register it:

Swift

Add the following code to AppDelegate:

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken:
    NSData)
{
    // If the AppMetrica SDK library was not initialized before this step,
    // calling the method causes the app to crash.
    YMPYandexMetricaPush.setDeviceTokenFrom(deviceToken)
}
```

Objective-C

Add the following code to your implementation of UIApplicationDelegate:

```
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData
    *)deviceToken
```

```
{
  // If the AppMetrica SDK library was not initialized before this step,
  // calling the method causes the app to crash.
  [YMPYandexMetricaPush setDeviceTokenFromData:deviceToken];
}
```

To register the device token and send the APN environments, add the following code:

Swift

```
func application(_ application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data)
{
  // If the AppMetrica SDK library was not initialized before this step,
  // calling the method causes the app to crash.
  #if DEBUG
    let pushEnvironment = YMPYandexMetricaPushEnvironment.development
  #else
    let pushEnvironment = YMPYandexMetricaPushEnvironment.production
  #endif
  YMPYandexMetricaPush.setDeviceTokenFrom(deviceToken, pushEnvironment: pushEnvironment)
}
```

Objective-C

```
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
  // If the AppMetrica SDK library was not initialized before this step,
  // calling the method causes the app to crash.
  #ifdef DEBUG
    YMPYandexMetricaPushEnvironment pushEnvironment = YMPYandexMetricaPushEnvironmentDevelopment;
  #else
    YMPYandexMetricaPushEnvironment pushEnvironment = YMPYandexMetricaPushEnvironmentProduction;
  #endif
  [YMPYandexMetricaPush setDeviceTokenFromData:deviceToken pushEnvironment:pushEnvironment];
}
```



Attention: AppMetrica allows you to send push notifications to Sandbox APNs. However, push notification processing may not work correctly if versions of the application with different environments were run on the device (*development* and *production*). To avoid this issue, you can use a separate test API key for *development* environment.

Step 4. Configure handling the opening of push notifications.

Configure handling the opening of push notifications:

1. Add the following code to the appropriate AppDelegate | UIApplicationDelegate methods:

Swift

```
func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable : Any])
{
  self.handlePushNotification(userInfo)
}

func application(_ application: UIApplication, didReceiveRemoteNotification userInfo: [AnyHashable : Any],
  fetchCompletionHandler completionHandler: @escaping (UIBackgroundFetchResult) -> Void)
{
  self.handlePushNotification(userInfo)
  completionHandler(.newData)
}

func handlePushNotification(_ userInfo: [AnyHashable : Any])
{
  // Track received remote notification.
  // Method [YMPYandexMetrica activateWithApiKey:] should be called before using this method.
  YMPYandexMetricaPush.handleRemoteNotification(userInfo)
}
```

Objective-C

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
  [YMPYandexMetricaPush handleApplicationDidFinishLaunchingWithOptions:launchOptions];
  return YES;
}

- (void)application:(UIApplication *)application
  didReceiveRemoteNotification:(NSDictionary *)userInfo
{
  [YMPYandexMetricaPush handleRemoteNotification:userInfo];
}

- (void)application:(UIApplication *)application
  didReceiveRemoteNotification:(NSDictionary *)userInfo
  fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
```

```
{
  [YMPYandexMetricaPush handleRemoteNotification:userInfo];
}
```

2. If you also use [UISceneDelegate](#), add the following code to the `scene(_:willConnectTo:options:)` method:

Swift

```
func scene(_ scene: UIScene, willConnectTo
          session: UISceneSession, options
          connectionOptions: UIScene.ConnectionOptions) {
    YMPYandexMetricaPush.handleSceneWillConnectToSession(with: connectionOptions)
}
```

Objective-C

```
- (void)scene:(UIScene *)scene willConnectToSession:(UISceneSession *)session
  options:(UISceneConnectionOptions *)connectionOptions
{
    [YMPYandexMetricaPush handleSceneWillConnectToSessionWithOptions:connectionOptions];
}
```

Handling push notifications for iOS 9 and below

If you have iOS 9 and lower or you don't use the new notification feature on iOS 10, you'll need to track the receipt of push notifications yourself. To track push notification openings and other actions with them, use the appropriate `UIApplicationDelegate` methods.

Handling push notifications for iOS 10 and higher

If you have iOS 10 or higher and want to use the new type of push notifications introduced in iOS 10, use the `YMPUserNotificationCenterDelegate` delegate. It handles the receipt of push notifications automatically when they're opened.

Make the following changes to the code:

Swift

```
import UserNotifications

// In the "func application(_ application: UIApplication, didFinishLaunchingWithOptions
// launchOptions: [UIApplicationLaunchOptionsKey : Any]? = nil) -> Bool" method:
if #available(iOS 10.0, *) {
    let delegate = YMPYandexMetricaPush.userNotificationCenterDelegate()
    UNUserNotificationCenter.current().delegate = delegate
}
```

Objective-C

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

// In the "- (BOOL)application:(UIApplication *)application
// didFinishLaunchingWithOptions:(NSDictionary *)launchOptions" method:
if ([UNUserNotificationCenter class] != Nil) {
    [UNUserNotificationCenter currentNotificationCenter].delegate =
    [YMPYandexMetricaPush userNotificationCenterDelegate];
}
```

To track push notification openings and other actions with them, create your own delegate named `UNUserNotificationCenterDelegate` and add it to `nextDelegate`:

Swift

```
YMPYandexMetricaPush.userNotificationCenterDelegate().nextDelegate = yourDelegate
```

Objective-C

```
[YMPYandexMetricaPush userNotificationCenterDelegate].nextDelegate = yourDelegate;
```

After that, you can use the appropriate methods of your delegate.

Step 5. (Optional) Enable push tokens update

The APNS service can withdraw the push token of the device, for example, if the user did not launch the application for a long time. AppMetrica stores push tokens on the server and can not send a push notification to a device with an obsolete token.

To automatically collect current push token go to the application settings in the AppMetrica interface and enable the **Update tokens with a Silent Push notification** option in the **Push Notifications** tab.

Step 6. (Optional) Configure uploading attached files

Note:

The functionality is not available in the web interface of push campaigns.

You can configure uploading attached files in push notifications:

1. Configure uploading attached files in push notifications by calling the [downloadAttachmentsForNotificationRequest](#) method in the Push SDK. See an example of integration in the article [Uploading attached files](#).
2. Add attachments (the `attachments` parameter) using the [Sending push messages](#) operation in the Push API.

Sending additional information

You can send additional information with the push notification if necessary. This data is specified in the AppMetrica web interface when [configuring the push campaign](#). To get this information, use the following method:

Swift

```
let userData = YMPYandexMetricaPush.userData(forNotification: userInfo)
```

Objective-C

```
NSString *userData = [YMPYandexMetricaPush userDataForNotification:userInfo];
```

where `userInfo` contains information about the push notification.

Defining the recipient of a notification

AppMetrica allows you to detect “own” push notifications, if several Push SDKs were built into the application.

To detect, if the AppMetrica is the recipient of a notification, use the following method:

Swift

```
let isRelatedToAppMetricaSDK = YMPYandexMetricaPush.isNotificationRelated(toSDK: userInfo)
```

Objective-C

```
BOOL isRelatedToAppMetricaSDK = [YMPYandexMetricaPush isNotificationRelatedToSDK:userInfo];
```

See also

[Configuring an iOS application to send push notifications](#) on page 46

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.

Objective-C reference

Classes

YMPYandexMetricaPush class

The main class for push notifications handling.

Instance methods

+downloadAttachmentsForNotificationRequest:	Uploads attached files in push notifications. The method is available for iOS 10.0 and higher.
+handleApplicationDidFinishLaunchingWithOptions:	Handles push notification openings from the method application:didFinishLaunchingWithOptions: . Method should be invoked after AppMetrica SDK initialization.
+handleDidReceiveNotificationRequest:	Handles push notifications receiving from Notification Service Extension.
+handleRemoteNotification:	Handles push notification openings from the method application:didReceiveRemoteNotification:fetchCompletionHandler: . Method should be invoked after AppMetrica SDK initialization.
+handleSceneWillConnectToSession:	Handles opening push notifications from the method scene:willConnectToSession:options: . Method should be invoked after AppMetrica SDK initialization.
+isNotificationRelatedToSDK:	Returns YES if a push notification is related to AppMetrica.
+setDeviceTokenFromData:	Registers the device token for an application with a production environment. Method should be invoked after AppMetrica SDK initialization.
+setDeviceTokenFromData:pushEnvironment:	Registers the device token of the application with the specified environment. Method should be invoked after AppMetrica SDK initialization.
+setExtensionAppGroup:	Registers the App Groups shared group for the app and Notification Service Extension.
+userDataForNotification:	Returns an arbitrary data string that is passed in the push notification: <ul style="list-style-type: none"> • In the Additional data field when sending from the AppMetrica interface. • In the data field when sending using the Push API.
+userNotificationCenterDelegate:	Returns a delegate YMPUserNotificationCenterDelegate , which handles foreground push notifications on iOS 10 and higher.
+userNotificationCenterHandler:	Returns a delegate YMPUserNotificationCenterHandling , which allows you to manually handle foreground push notifications on iOS 10 and higher.

*Method descriptions***+downloadAttachmentsForNotificationRequest:**

```
+ (void)downloadAttachmentsForNotificationRequest:(UNNotificationRequest *)request
                                     callback:(YMPAttachmentsDownloadCallback)callback;
```

Uploads attached files in push notifications. The method is available for iOS 10.0 and higher.

Parameters:

request	The instance of UNNotificationRequest .
---------	---

callback

The callback block for uploading attached files. Format: `typedef void (^YMPAttachmentsDownloadCallback) (NSArray<UNNotificationAttachment *> * _Nullable attachments, NSError * _Nullable error)`. Includes an array of attachments and an error if an error occurs when uploading files.

+handleApplicationDidFinishLaunchingWithOptions:

```
+ (void)handleApplicationDidFinishLaunchingWithOptions:(nullable NSDictionary *)launchOptions
```

Handles push notification openings from the method [application:didFinishLaunchingWithOptions:](#). Method should be invoked after AppMetrica SDK initialization.

Parameters:

launchOptions

Parameters as key-value pairs that contain information about the application start.

+handleDidReceiveNotificationRequest:

```
+ (void)handleDidReceiveNotificationRequest:(UNNotificationRequest *)request
```

Handles push notifications receiving from Notification Service Extension.

You should call the method in the implementation of [didReceiveNotificationRequest:withContentHandler:](#).

Parameters:

request

The instance of [UNNotificationRequest](#).

+handleRemoteNotification:

```
+ (void)handleRemoteNotification:(NSDictionary *)userInfo
```

Handles push notification openings from the method [application:didReceiveRemoteNotification:fetchCompletionHandler:](#). Method should be invoked after AppMetrica SDK initialization.

Parameters:

userInfo

Parameters of push notifications as key-value pairs that are transmitted by the system.

+handleSceneWillConnectToSession:

```
+ (void)handleSceneWillConnectToSessionWithOptions:(UISceneConnectionOptions *)connectionOptions
```

Handles opening push notifications from the method [scene:willConnectToSession:options:](#). Method should be invoked after AppMetrica SDK initialization.

Parameters:

connectionOptions

The [UISceneConnectionOptions](#) class object with connection parameters that are transmitted by the system.

+isNotificationRelatedToSDK:

```
+ (BOOL)isNotificationRelatedToSDK:(NSDictionary *)userInfo;
```

Returns YES if a push notification is related to AppMetrica.

Parameters:

userInfo Parameters of push notifications as key-value pairs that are transmitted by the system.

Returns:

- YES — If the push notification refers to AppMetrica.
- NO — If the push notification is not related to AppMetrica.

+setDeviceTokenFromData:

```
+ (void)setDeviceTokenFromData:(nullable NSData *)data
```

Registers the device token for an application with a production environment. Method should be invoked after AppMetrica SDK initialization.

Parameters:

data Device token of the application.
If you pass the `nil` value, the previous device token is revoked.

+setDeviceTokenFromData:pushEnvironment:

```
+ (void)setDeviceTokenFromData:(nullable NSData *)data  
pushEnvironment:(YMPYandexMetricaPushEnvironment)pushEnvironment
```

Registers the device token of the application with the specified environment. Method should be invoked after AppMetrica SDK initialization.



Attention: AppMetrica allows you to send push notifications to Sandbox APNs. However, push notification processing may not work correctly if versions of the application with different environments were run on the device (*development* and *production*). To avoid this issue, you can use a separate test API key for *development* environment.

Parameters:

data Device token of the application.
If you pass the `nil` value, the previous device token is revoked.

pushEnvironment APNs app environment.

+setExtensionAppGroup:

```
+ (void)setExtensionAppGroup:(NSString *)appGroup
```

Registers the App Groups shared group for the app and Notification Service Extension.

Registration is necessary for tracking the delivery of push notifications. For more information, see [Configuring statistics collection for push notifications](#).

Parameters:

appGroup The name of the shared App Groups group.

+userDataForNotification:

```
+ (nullable NSString *)userDataForNotification:(NSDictionary *)userInfo
```

Returns an arbitrary data string that is passed in the push notification:

- In the **Additional data** field when sending from the AppMetrica interface.
- In the data field when sending using the [Push API](#).

Parameters:

userInfo	Parameters of push notifications as key-value pairs that are transmitted by the system.
----------	---

Returns:

An arbitrary data string.

+userNotificationCenterDelegate

```
+ (id<YMPUserNotificationCenterDelegate>)userNotificationCenterDelegate
```

Returns a delegate [YMPUserNotificationCenterDelegate](#), which handles foreground push notifications on iOS 10 and higher.

To handle foreground push notifications, add this code to the [application: didFinishLaunchingWithOptions:](#) method:

```
[UNUserNotificationCenter currentNotificationCenter].delegate = [YMPYandexMetricaPush userNotificationCenterDelegate];
```

For manual handling of push notifications, use [+userNotificationCenterHandler](#).

Returns:

A delegate that implements the [YMPUserNotificationCenterDelegate](#) protocol.

+userNotificationCenterHandler

```
+ (id<YMPUserNotificationCenterHandling>)userNotificationCenterHandler
```

Returns a delegate [YMPUserNotificationCenterHandling](#), which allows you to manually handle foreground push notifications on iOS 10 and higher.

Use this delegate if you implement the [UNUserNotificationCenterDelegate](#) protocol with custom logic. In this case, you should implement all methods of the [UNUserNotificationCenterDelegate](#) delegate and call the corresponding methods in [YMPUserNotificationCenterHandling](#).

For simplified push notification handling, use [+userNotificationCenterDelegate](#).

Returns:

A delegate that implements the [YMPUserNotificationCenterHandling](#) protocol.

Protocols**YMPUserNotificationCenterDelegate protocol**

A delegate for handling foreground push notifications on iOS 10 and higher.

To handle foreground push notifications, add this code to the [application: didFinishLaunchingWithOptions:](#) method:

```
[UNUserNotificationCenter currentNotificationCenter].delegate = [YMPYandexMetricaPush userNotificationCenterDelegate];
```

Properties[presentationOptions](#)

Parameters for displaying push notifications. They are passed to the handler [userNotificationCenter:willPresentNotification:withCompletionHandler:](#).

[nextDelegate](#)

A delegate to which calls of this protocol will be proxied.

*Property descriptions***presentationOptions**

(nonatomic, assign) [UNNotificationPresentationOptions](#) presentationOptions

Parameters for displaying push notifications. They are passed to the handler [userNotificationCenter:willPresentNotification:withCompletionHandler:](#).

The delegate invokes the handler if the [nextDelegate](#) property is not set or if the object is in [nextDelegate](#) does not respond to the selector.

nextDelegate

(nonatomic, weak, nullable) id<UNUserNotificationCenterDelegate> nextDelegate;

A delegate to which calls of this protocol will be proxied.

YMPUserNotificationCenterHandling Class

A delegate for manual handling foreground push notifications on iOS 10 and higher.

Use this delegate if you implement the [UNUserNotificationCenterDelegate](#) protocol with custom logic. You should implement all methods of [UNUserNotificationCenterDelegate](#) and call proper methods in [YMPUserNotificationCenterHandling](#).

The implementation of this delegate is called by the [YMPYandexMetricaPush.userNotificationCenterDelegate](#) method.

Instance methods[-userNotificationCenterWillPresentNotification:](#)

You should call this method in your implementation of [userNotificationCenter:willPresentNotification:withCompletionHandler:](#).

[-userNotificationCenterDidReceiveNotificationResponse:](#)

You should call this method in your implementation of [userNotificationCenter:didReceiveNotificationResponse:withCompletionHandler:](#).

[-userNotificationCenterOpenSettingsForNotification:](#)

You should call this method in your implementation of [userNotificationCenter:openSettingsForNotification:](#).

*Method descriptions***-userNotificationCenterWillPresentNotification:**

```
- (void)userNotificationCenterWillPresentNotification:(UNNotification *)notification
```

You should call this method in your implementation of [userNotificationCenter:willPresentNotification:withCompletionHandler:](#).

Parameters:

notification

The instance of [UNNotification](#).

+handleRemoteNotification:	Handles push notification openings from the method application(_:didReceiveRemoteNotification:fetchCompletionHandler:) . Method should be invoked after AppMetrica SDK initialization.
+handleSceneWillConnectToSession:	Handles push notification openings from the method scene(_:willConnectTo:options:) . Method should be invoked after AppMetrica SDK initialization.
+isNotificationRelatedToSDK:	Returns YES if a push notification is related to AppMetrica.
+setDeviceTokenFromData:	Registers the device token for an application with a production environment. Method should be invoked after AppMetrica SDK initialization.
+setDeviceTokenFromData:pushEnvironment:	Registers the device token of the application with the specified environment. Method should be invoked after AppMetrica SDK initialization.
+setExtensionAppGroup:	Registers the App Groups shared group for the app and Notification Service Extension.
+userDataForNotification:	Returns an arbitrary data string that is passed in the push notification: <ul style="list-style-type: none"> • In the Additional data field when sending from the AppMetrica interface. • In the data field when sending using the Push API.
+userNotificationCenterDelegate:	Returns a delegate YMPUserNotificationCenterDelegate , which handles foreground push notifications on iOS 10 and higher.
+userNotificationCenterHandler:	Returns a delegate YMPUserNotificationCenterHandling , which allows you to manually handle foreground push notifications on iOS 10 and higher.

Method descriptions

downloadAttachmentsForNotificationRequest:

```
class func downloadAttachments(request: UNNotificationRequest, callback: YMPAttachmentsDownloadCallback)
```

Uploads attached files in push notifications. The method is available for iOS 10.0 and higher.

Parameters:

request	The instance of UNNotificationRequest .
callback	The callback block for uploading notification contents. Format: <code>public typealias YMPAttachmentsDownloadCallback = ([UNNotificationAttachment]?, Error?) -> Void</code> . Includes an array of attachments and an error if an error occurs when uploading files.

handleApplicationDidFinishLaunching(withOptions:)

```
class func handleApplicationDidFinishLaunching(withOptions launchOptions: [AnyHashable : Any]?)
```

Handles push notification openings from the method [application\(_: didFinishLaunchingWithOptions:\)](#). Method should be invoked after AppMetrica SDK initialization.

Parameters:

launchOptions Parameters as key-value pairs that contain information about the application start.

handleDidReceive(_:)

```
class func handleDidReceive(_ request: UNNotificationRequest?)
```

Handles push notifications receiving from Notification Service Extension.

You should call the method in the implementation of [didReceive\(_: withContentHandler:\)](#).

Parameters:

request The instance of [UNNotificationRequest](#).

handleRemoteNotification(_:)

```
class func handleRemoteNotification(_ userInfo: [AnyHashable : Any]?)
```

Handles push notification openings from the method [application\(_: didReceiveRemoteNotification:fetchCompletionHandler:\)](#) Method should be invoked after AppMetrica SDK initialization.

Parameters:

userInfo Parameters of push notifications as key-value pairs that are transmitted by the system.

handleSceneWillConnectToSession(with: connectionOptions)

```
class func handleSceneWillConnectToSession(with: connectionOptions)
```

Handles push notification openings from the method [scene\(_: willConnectTo:options:\)](#). Method should be invoked after AppMetrica SDK initialization.

Parameters:

connectionOptions The [UIScene.ConnectionOptions](#) class object with connection parameters that are transmitted by the system.

isNotificationRelated(toSDK:)

```
class func isNotificationRelated(toSDK userInfo: [AnyHashable : Any]?) -> Bool
```

Returns YES if a push notification is related to AppMetrica.

Parameters:

userInfo Parameters of push notifications as key-value pairs that are transmitted by the system.

Returns:

- YES — If the push notification refers to AppMetrica.
- NO — If the push notification is not related to AppMetrica.

setDeviceTokenFrom(_:)

```
class func setDeviceTokenFrom(_ data: Data?)
```

Registers the device token for an application with a production environment. Method should be invoked after AppMetrica SDK initialization.

Parameters:

data	Device token of the application. If you pass the <code>nil</code> value, the previous device token is revoked.
------	---

setDeviceTokenFrom(_:pushEnvironment:)

```
class func setDeviceTokenFrom(_ data: Data?, pushEnvironment: YMPYandexMetricaPushEnvironment)
```

Registers the device token of the application with the specified environment. Method should be invoked after AppMetrica SDK initialization.



Attention: AppMetrica allows you to send push notifications to Sandbox APNs. However, push notification processing may not work correctly if versions of the application with different environments were run on the device (*development* and *production*). To avoid this issue, you can use a separate test API key for *development* environment.

Parameters:

data	Device token of the application. If you pass the <code>nil</code> value, the previous device token is revoked.
pushEnvironment	APNs app environment.

setExtensionAppGroup(_:)

```
class func setExtensionAppGroup(_ appGroup: String?)
```

Registers the App Groups shared group for the app and Notification Service Extension.

Registration is necessary for tracking the delivery of push notifications. For more information, see [Configuring statistics collection for push notifications](#).

Parameters:

appGroup	The name of the shared App Groups group.
----------	--

userData(forNotification:)

```
class func userData(forNotification userInfo: [AnyHashable : Any]?) -> String?
```

Returns an arbitrary data string that is passed in the push notification:

- In the **Additional data** field when sending from the AppMetrica interface.
- In the `data` field when sending using the [Push API](#).

Parameters:

userInfo	Parameters of push notifications as key-value pairs that are transmitted by the system.
----------	---

Returns:

An arbitrary data string.

userNotificationCenterDelegate()

```
class func userNotificationCenterDelegate() -> YMPUserNotificationCenterDelegate?
```

Returns a delegate [YMPUserNotificationCenterDelegate](#), which handles foreground push notifications on iOS 10 and higher.

To handle foreground push notifications, add this code to the [application\(_:didFinishLaunchingWithOptions:\)](#) method:

```
let delegate = YMPYandexMetricaPush.userNotificationCenterDelegate()
UNUserNotificationCenter.current().delegate = delegate
```

To manually handle push notifications, use [userNotificationCenterHandler\(\)](#).

Returns:

A delegate that implements the [YMPUserNotificationCenterDelegate](#) protocol.

userNotificationCenterHandler()

```
class func userNotificationCenterHandler() -> YMPUserNotificationCenterHandling?
```

Returns a delegate [YMPUserNotificationCenterHandling](#), which allows you to manually handle foreground push notifications on iOS 10 and higher.

Use this delegate if you implement the [UNUserNotificationCenterDelegate](#) protocol with custom logic. In this case, you should implement all methods of the [UNUserNotificationCenterDelegate](#) delegate and call the corresponding methods in [YMPUserNotificationCenterHandling](#).

For simplified push notification handling, use [userNotificationCenterDelegate\(\)](#)

Returns:

A delegate that implements the [YMPUserNotificationCenterHandling](#) protocol.

Protocols**YMPUserNotificationCenterDelegate protocol**

A delegate for handling foreground push notifications on iOS 10 and higher.

To handle foreground push notifications, add this code to the [application\(_:didFinishLaunchingWithOptions:\)](#) method:

```
let delegate = YMPYandexMetricaPush.userNotificationCenterDelegate()
UNUserNotificationCenter.current().delegate = delegate
```

Properties[presentationOptions](#)

Parameters for displaying push notifications. They are passed to the handler [userNotificationCenter\(_:willPresent:withCompletionHandler:\)](#).

[nextDelegate](#)

A delegate to which calls of this protocol will be proxied.

*Property descriptions***presentationOptions**

```
var presentationOptions: UNNotificationPresentationOptions
```

Parameters for displaying push notifications. They are passed to the handler [userNotificationCenter\(_:willPresent:withCompletionHandler:\)](#).

The delegate invokes the handler if the `nextDelegate` property is not set or if the object is in `nextDelegate` does not respond to the selector.

nextDelegate

```
var nextDelegate: id<UNUserNotificationCenterDelegate>
```

A delegate to which calls of this protocol will be proxied.

YMPUserNotificationCenterHandling Class

A delegate for manual handling foreground push notifications on iOS 10 and higher.

Use this delegate if you implement the [UNUserNotificationCenterDelegate](#) protocol with custom logic. You should implement all methods of [UNUserNotificationCenterDelegate](#) and call proper methods in [YMPUserNotificationCenterHandling](#).

The implementation of this delegate is provided by the [YMPYandexMetricaPush.userNotificationCenterDelegate](#) method.

Instance methods

[userNotificationCenterWillPresent\(_:\)](#)

You should call this method in your implementation of [userNotificationCenter\(_:willPresent:withCompletionHandler:\)](#).

[userNotificationCenterDidReceive\(_:\)](#)

You should call this method in your implementation of [userNotificationCenter\(_:didReceive:withCompletionHandler:\)](#).

[userNotificationCenterOpenSettings\(_:\)](#)

You should call this method in your implementation of [userNotificationCenter\(_:openSettingsFor:\)](#).

Method descriptions

userNotificationCenterWillPresent(_:)

```
func userNotificationCenterWillPresent(_ notification: UNNotification?)
```

You should call this method in your implementation of [userNotificationCenter\(_:willPresent:withCompletionHandler:\)](#).

Parameters:

`notification` The instance of [UNNotification](#).

userNotificationCenterDidReceive(_:)

```
func userNotificationCenterDidReceive(_ response: UNNotificationResponse?)
```

You should call this method in your implementation of [userNotificationCenter\(_:didReceive:withCompletionHandler:\)](#).

Parameters:

`notification` The instance of [UNNotificationResponse](#).

userNotificationCenterOpenSettings(_:)

```
func userNotificationCenterOpenSettings(for notification: UNNotification?)
```

You should call this method in your implementation of `userNotificationCenter` (`_: openSettingsFor:`).

Parameters:

notification The instance of `UNNotification`.

Enumerations

YMPYandexMetricaPushEnvironment

Contains possible types of environments.

Enumerations

`YMPYandexMetricaPushEnvironment`

#####

YMPYandexMetricaPushEnvironment

```
enum YMPYandexMetricaPushEnvironment : NSUInteger
```

Constants

`YMPYandexMetricaPushEnvironment.Production`
`YMPYandexMetricaPushEnvironment.Development`

Description

Environment for production certificates.
 Environment for production and development certificates.

Configuring push notification interactions tracking

In AppMetrica you can configure tracking interactions with push notifications (such as delivery and dismiss) for iOS 10 and above.

Collecting delivery statistics

To collect the statistics of delivered push notification, follow these steps:

Step 1. Create Notification Service Extension

1. In Xcode, select **File** → **New** → **Target**.
2. In the **iOS** extensions section, choose **Notification Service Extension** from the list and click **Next**.
3. Enter the name of the extension in the **Product Name** field and click **Finish**.

Step 2. Create a shared App Groups group

1. In the Xcode project settings, go to the **Capabilities** tab.
2. Switch on the **App Groups** option for the created extension and for the application. To switch between an extension and an app, go to the project settings panel and click or the drop-down element `NotificationServiceExtension`.
3. In the **App Groups** section use the **+** button to create a group. You will need the group name during further configuration.
4. Select the group you created for the app and for the extension.

Step 3. Make changes in NotificationService

Objective-C

In the `NotificationService.m` file add the following code to the corresponding method:

```
- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request
withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler
{
    [YMPYandexMetricaPush setExtensionAppGroup:appGroup];
    [YMPYandexMetricaPush handleDidReceiveNotificationRequest:request];
}
```

```
    ...
}
```

Swift

In the `NotificationService.swift` file add the following code to the corresponding method:

```
func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler: @escaping
(UNNotificationContent) -> Void) {
    ...
    if let bestAttemptContent = bestAttemptContent {
        ...
        YMPYandexMetricaPush.setExtensionAppGroup(appGroup)
        YMPYandexMetricaPush.handleDidReceive(request)
        ...
    }
}
```

`appGroup` — the name of the shared App Groups group.

Step 4. Configure handling of push notifications.**Objective-C**

Add the following code to the corresponding implementation of the `UIApplicationDelegate` method:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Activating AppMetrica.
    ...
    [YMPYandexMetricaPush setExtensionAppGroup:appGroup];
    [YMPYandexMetricaPush handleApplicationDidFinishLaunchingWithOptions:launchOptions];
    ...
}
```

Swift

Add the following code to the corresponding implementation of the `AppDelegate` method:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey :Any]? = nil) -> Bool
{
    // Activating AppMetrica.
    ...
    YMPYandexMetricaPush.setExtensionAppGroup(appGroup)
    YMPYandexMetricaPush.handleApplicationDidFinishLaunching(options: launchOptions)
    ...
}
```

`appGroup` — the name of the shared App Groups group.

Collecting push notification dismiss interactions

To collect the statistics of dismiss interactions of push notifications, set the following option for the `UNNotificationCategory` category:

Objective-C

```
options:UNNotificationCategoryOptionCustomDismissAction
```

Swift

```
options: UNNotificationCategory.customDismissAction
```

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.

Uploading attached files**Note:**

You can add attachments in the Push API using the [Sending push messages](#) operation (the attachments parameter).

Configure uploading attached files in push notifications by calling the `downloadAttachmentsForNotificationRequest` method ([objective-c/swift](#)):

Objective-C

```
@implementation NotificationService

- (void)didReceiveNotificationRequest:(UNNotificationRequest *)request withContentHandler:(void (^)(UNNotificationContent * _Nonnull))contentHandler {
    self.contentHandler = contentHandler;
    self.bestAttemptContent = [request.content mutableCopy];

    [YMPYandexMetricaPush setExtensionAppGroup:@"EXTENSION_AND_APP_SHARED_APP_GROUP_NAME"];
    [YMPYandexMetricaPush handleDidReceiveNotificationRequest:request];

    [YMPYandexMetricaPush downloadAttachmentsForNotificationRequest:request
     *attachments, NSError *error) {
        callback:^(NSArray<UNNotificationAttachment *>
        if (error != nil) {
            NSLog(@"Error: %@", error);
        }
        [self
    completeWithBestAttemptAndAttachments:attachments];
    }

- (void)serviceExtensionTimeWillExpire {
    [self completeWithBestAttemptAndAttachments:nil];
}

- (void)completeWithBestAttemptAndAttachments:(NSArray<UNNotificationAttachment *> *)attachments
{
    @synchronized (self) {
        if (self.contentHandler != nil) {
            if (attachments != nil) {
                self.bestAttemptContent.attachments = attachments;
            }
            self.contentHandler(self.bestAttemptContent);
            self.contentHandler = nil;
        }
    }
}

@end
```

Swift

```
class NotificationService: UNNotificationServiceExtension {

    private var contentHandler: ((UNNotificationContent) -> Void)?
    private var bestAttemptContent: UNMutableNotificationContent?
    private let syncQueue = DispatchQueue(label: "NotificationService.syncQueue")

    override func didReceive(_ request: UNNotificationRequest, withContentHandler contentHandler: @escaping (UNNotificationContent) -> Void) {
        self.contentHandler = contentHandler
        self.bestAttemptContent = (request.content.mutableCopy() as? UNMutableNotificationContent)

        // ...

        YMPYandexMetricaPush.downloadAttachments(for: request) { attachments, error in
            if let error = error {
                print("Error: \(error)")
            }
            self.completeWithBestAttempt(attachments: attachments)
        }
    }

    override func serviceExtensionTimeWillExpire() {
        completeWithBestAttempt(attachments: nil)
    }

    func completeWithBestAttempt(attachments: [UNNotificationAttachment]?) {
        syncQueue.sync {
            if let contentHandler = contentHandler, let bestAttemptContent = bestAttemptContent {
                if let attachments = attachments {
                    bestAttemptContent.attachments = attachments
                }
                contentHandler(bestAttemptContent)
                self.bestAttemptContent = nil
                self.contentHandler = nil
            }
        }
    }
}
```

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.

Configuring an iOS application to send push notifications

The Universal Push Notification Client SSL Certificate is required for sending push notifications to iOS apps. To get this certificate, you need to specify your app's identifier (App ID). You can [create one in the Apple Developer Console](#).

If you already have an App ID, request a certificate.

Step 1. Request a certificate

1. Open **Keychain Access** on your computer.
2. Go to **Keychain Access** → **Certificate Assistant** → **Request a Certificate From a Certificate Authority** and enter the required information. Pay attention to the password. You will need it when setting up the certificate in the AppMetrica interface.
3. Under **Request is**, enable the **Saved to disk** option and click **Continue**.
4. Save the request file on your computer and click **Done**.

Step 2. Get a certificate

1. In the Apple Developer Console, go to [Certificates, Identifiers & Profiles](#).
2. In the **Certificates** menu, select **All** and go to the **Production** section.
3. Enable the **Apple Push Notification service SSL (Sandbox & Production)** option and click **Continue**.
4. In the **App ID** drop-down list, select the app ID that needs a certificate, and click **Continue**. The selected **App ID** must match your **bundle ID**.
5. Since the certificate request has already been sent, click **Continue**.
6. Click **Choose File**. In the window that opens, select the certificate request file (for example, `example.certSigningRequest`) and click **Choose**, then **Continue**.
7. Click **Download** to download the certificate. Then click **Done**.
8. Double-click the file to install the certificate.

Step 3. Export a Private Key from the installed certificate

1. Open Keychain Access on your computer and choose **Certificates** in the menu.
2. In the list, select the certificate that you installed.
3. Choose **File** → **Export Items** in the menu. Then enter a name for the file to export and set the format to Personal Information Exchange (P12). Click **Save**.
4. In **Password**, set a password for the key and click **OK**. If you don't set a password, the certificate won't be accepted in the AppMetrica interface.
5. To activate the export, enter the password for your computer. Then click **Allow**. The P12 file is saved on your computer.

Step 4. Use the Private Key in AppMetrica

1. In the AppMetrica interface, go to the [Applications](#) section and select the app that you want to run push campaigns for.
2. In the menu on the left, select **Settings**.
3. Go to the **Push notifications** tab.
4. Under **iOS**, click **Choose file** (next to **Universal Push Notification Client SSL Certificate**).
5. In the window that opens, select the key file in P12 format and upload it.
6. In **Password**, enter the password you set when saving the key (step 3).

See also

[Connecting the AppMetrica Push SDK](#) on page 24
[Launching a push campaign](#)

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 1.3.0

Released July 1, 2022

- Added the `YMPYandexMetricaPush.handleSceneWillConnectToSession` method ([Objective-C](#) | [Swift](#)) to work with [UIScene](#) (with iOS 13 or higher).

Version 1.1.1

Released September 29, 2021

- Added support for [AppMetrica SDK 4.0.0](#).

Version 1.0.0

Released 13 July 2021

- The minimum supported iOS version is 9.0 or later.
- 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.
- Added a library version for iOS simulators that are run on Apple Silicon M1 Macs (`ios-arm64-simulator`).
- Added Push SDK distribution using Swift Package Manager. For more information, see [Installation and initialization](#).

Version 0.9.2

Released 12 July 2021

- Feature added to upload attached files in push notifications using the [downloadAttachmentsForNotificationRequest](#) method for iOS 10 and higher. See an example of integration in the article [Uploading attached files](#).

Version 0.8.0

Released 26 April 2019

- Added the feature for manual push notification tracking with the custom [UNUserNotificationCenterDelegate](#) implementation:
 - Added the `+userNotificationCenterHandler` method.
 - Added the `YMPUserNotificationCenterHandling` protocol.

Version 0.7.2

Released 27 November 2018

- Fixed an issue in the dynamic framework.

Version 0.7.1

Released 19 November 2018

- Added a proxy delegate for [userNotificationCenter:openSettingsForNotification:](#) (YMPUserNotificationCenterDelegate).
- Fixed crash on setting a nil token to the `+setDeviceTokenFromData:` method.
- Added support for [AppMetrica SDK 3.4.0](#).

Version 0.7.0

Released 31 August 2018

- Added the tracking of the [shows/dismisses of the push notifications](#) for iOS 10 and higher.

Version 0.6.0

Released 20 April 2018

- Added support for [AppMetrica SDK 3.0.0](#).

Version 0.5.1

Released 21 March 2018

- Fixed crash when opening URL from push notification.
- Fixed a bug that occurs when a URL of the same push notification is opened more than once.

Version 0.5.0

Released 26 October 2017

- Added the in-app notifications support for iOS 10 and higher. Requires additional integration ([Objective-C](#) / [Swift](#)).
- Added method for identification of push notifications related to AppMetrica Push SDK ([Objective-C](#) / [Swift](#)).
- Added method for sending APNs environment with device token ([Objective-C](#) / [Swift](#)).

Version 0.4.0

Released 24 November 2016

- Links from push notifications are now opened.
- Added a dynamic framework.
- The DynamicDependencies framework has been changed to Dynamic.
- The StaticDependencies framework has been changed to Static.

Version 0.3.0

Released 10 October 2016

- Integrated with the AppMetrica Mobile SDK library.

Windows

Connecting the AppMetrica Push SDK



Attention: We discontinued the development of new versions of the AppMetrica Windows SDK.

You can use the Windows SDK for mobile and desktop applications (developed on the UWP 10 platform).

Before using the AppMetrica Push SDK, you need to [enable and initialize the AppMetrica SDK](#) version 3.2.0 or higher.

Step 1. [Download](#) the AppMetrica Push SDK library.

Step 2. Initialize the library in the application by adding to the `App.xaml.cs` file the `Activate()` method call. Call it in the `OnLaunched` method implementation of the `App.xaml.cs` or in the `MainPage` constructor of your application:

```
...
YandexMetricaPush.Activate(AppSid);
...
```

AppSid — String with the Windows Store security identifier. It consists of the `ms-app://` prefix and the identifier value. For example, `ms-app://s-1-15-2-3792079137-3272192291-.....`

Note: You can get the `AppSid` by using the `WebAuthenticationBroker.GetCurrentApplicationCallbackUri().ToString();` method.

Step 3. Configure handling the opening of push notifications. You can use either [Windows Push Notification Services](#) or [Microsoft Push Notification Service](#) for working with push notifications in the app. Depending on which service is used, use one of the following methods to configure handling push notifications:

Windows Push Notification Services (WNS)

In the `OnLaunched` method for the `Application` class, add the `ProcessApplicationLaunch` method call for the `YandexMetricaPush` class:

```
protected override void OnLaunched(LaunchActivatedEventArgs e) {
    YandexMetricaPush.ProcessApplicationLaunch(e);
    ...
}
```

Microsoft Push Notification Service (MPNS)

In the `OnNavigatedTo` method for the `Page` class (the start page specified in the push notification parameters), add the `ProcessApplicationLaunch` method call for the `YandexMetricaPush` class:

```
protected override void OnNavigatedTo(NavigationEventArgs e) {
    YandexMetricaPush.ProcessApplicationLaunch(e);
    ...
}
```

See also

[Configuring a Windows application to send push notifications](#) on page 49

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.

Configuring a Windows application to send push notifications



Attention: We discontinued the development of new versions of the AppMetrica Windows SDK.

This section explains the steps for setting up the application with AppMetrica Push SDK on Windows.

Step 1. Register your app in the dashboard

1. In the [Windows Dev Center](#), go to **Dashboard** → **Submit your app**.
2. Click **App name**. Then enter the name of your app and click **Reserve name**. The name must be unique.

Step 2. Get the identity for your app

1. Go to the **Dashboard** and choose the app.
2. Go to the **Services** section. Under **Push notifications**, click **Get started**.
3. Under **Windows Push Notification Services (WNS)** and **Microsoft Azure Mobile Apps**, click the **Live Services site** link. In the menu on the left, choose **App identity**.

4. Note the **Application Secrets** and **Package SID** fields. You will need these values for [configuring AppMetrica](#).

If you use [Windows Push Notification Services](#), these values are also necessary for associating your project with the store in the dashboard. Add these values to your project manifest in one of the following ways:

- [Automatically](#) in Visual Studio.
- Manually in the manifest code.

Step 3. Configure AppMetrica

1. In the AppMetrica interface, go to the [Applications](#) section and select the app that you want to run push campaigns for.
2. In the menu on the left, select **Settings**.
3. Go to the **Push notifications** tab.
4. Under **Windows**, enter the values of **Application Secrets** and **Package SID** that you [copied from the Windows Dev Center interface](#) in the **Client secret** and **Security ID** fields. Changes are saved automatically.

See also

[Connecting the AppMetrica Push SDK](#) on page 48

[Launching a push campaign](#)

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 0.3.1



Attention: We discontinued the development of new versions of the AppMetrica Windows SDK.

Released 25 October 2017

- Integrated with the [AppMetrica SDK library \(3.5.0\)](#).

Plugins

Unity

Installation and initialization

AppMetrica Push Unity is a plugin for the [Unity3d](#) game platform that includes support for the AppMetrica Push SDK for the Android and iOS platforms.

Before using the AppMetrica Push Unity plugin, you need to [enable and initialize the AppMetrica Unity plugin](#) version 4.0.0 or later.



Attention: To update the plugin, delete the Assets/AppMetricaPush directory and import the new plugin version (pay attention to the [Android configuration](#)).

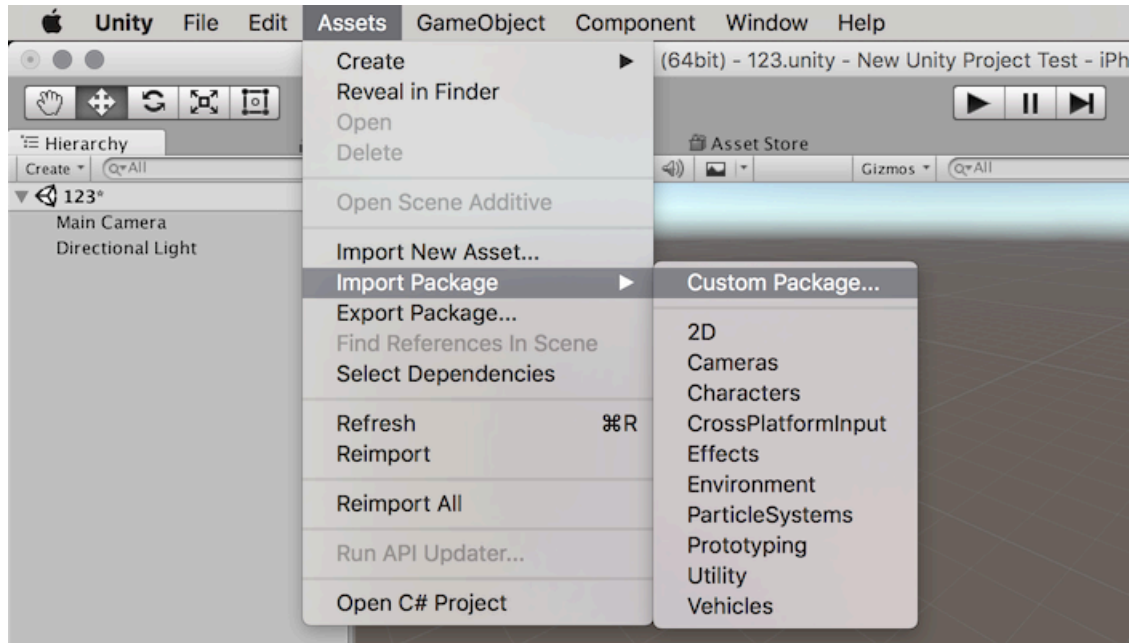
Integrating the plugin

Note:

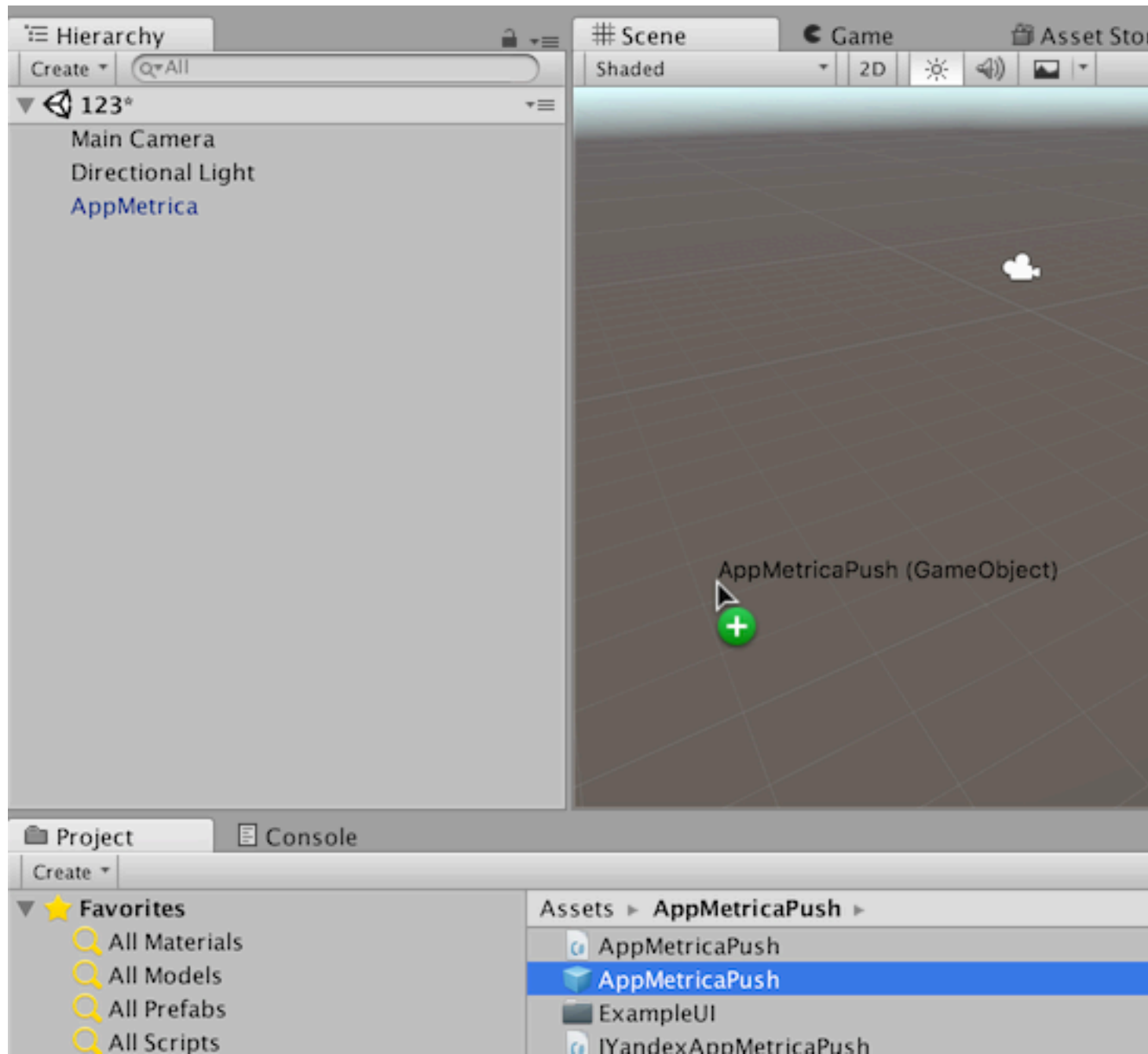
The plugin uses the [External Dependency Manager for Unity](#) to resolve dependencies.

Step 1. [Download](#) the AppMetrica Push Unity plugin.

Step 2. Add the plugin to the project — open the project in Unity Editor and import the AppMetricaPush.unitypackage plugin (**Assets** → **Import Package** → **Custom Package**).



Step 3. Open the Assets/AppMetricaPush/ folder and drag the AppMetrica prefab to the project's main stage.



If the plugin is integrated this way, the AppMetricaPush script on the added prefab automatically initializes the AppMetrica Push SDK.

The added AppMetricaPush prefab is a singleton. It isn't deleted when switching to a new Unity stage, and it deletes other objects that the AppMetricaPush script is installed on.

Configuring the plugin

iOS configuration

To receive push notifications, ask the user for permission. We recommend using the [Mobile Notifications](#) Unity package and request permission according to the [instructions](#).

Note: The AppMetrica Push Unity Plugin uses “swizzling”: it intercepts the execution of certain methods of the `UnityAppController` class by using the ObjectiveC runtime. The code is in the `AppMetricaPush/Plugins/iOS/YMPBridge.m` file.

Android configuration

Configuring AndroidManifest.xml

Make changes in the application element in the `AndroidManifest.xml` file:

```
<meta-data android:name="ymp_firebase_default_app_id" android:value="APP_ID" />
<meta-data android:name="ymp_gcm_default_sender_id" android:value="number:SENDER_ID" />
<meta-data android:name="ymp_firebase_default_api_key" android:value="API_KEY" />
<meta-data android:name="ymp_firebase_default_project_id" android:value="PROJECT_ID" />
```

APP_ID — ID of the app in Firebase. You can find it in the [Firebase console](#): go to the **Project settings**. In the **Your application** section copy the value of the **application ID** field.

SENDER_ID — The unique ID of the sender in Firebase. You can find it in the [Firebase console](#): go to **Project settings** → **Cloud Messaging** and copy the value of the **Sender ID** field.

API_KEY — App key in Firebase. You can find it in the `current_key` field of the `google-services.json` file. You can download the file in the [Firebase console](#).

PROJECT_ID — App ID in Firebase. You can find it in the `project_id` field of the `google-services.json` file. You can download the file in the [Firebase console](#).

Note: If you use [the AppMetrica Push SDK integration sample](#), change the package attribute value in the manifest element to the `package_id` of your application.

There is an example of the file in the `Assets/AppMetricaPush/Plugins/Android/` directory (`AndroidManifest.xml`).

Configuring the location tracking

By default, the AppMetrica Push SDK enables device location tracking.

See also

[Configuring an Android application to send push notifications](#) on page 8

[Configuring an iOS application to send push notifications](#) on page 46

[Launching a push campaign](#)

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.

Migrating from GCM to Firebase

Since the AppMetrica Push SDK version [0.2.0](#), it uses the Firebase Cloud Messaging (FCM) service to send push messages on the Android platform.

This section explains the steps for migrating the application from Google Cloud Messaging to Firebase Cloud Messaging.

Step 1. Import a project

Import a Google project (if you used Google APIs to create a project):

1. Go to the [Firebase console](#).
2. Click the **Add project** button.
3. In the drop-down list, select the name of the project you are planning to run push campaigns for.
4. Select the country your organization is officially registered in and click **Add Firebase**.
5. Click **Add Firebase to your Android app** and follow the instructions.

Step 2. Configuring your app

Edit the `AndroidManifest.xml` file:

1. Rename `ymp_gcm_project_number` to `ymp_gcm_default_sender_id` and get the following result:

```
<meta-data android:name="ymp_gcm_default_sender_id" android:value="number:SENDER_ID" />
```

SENDER_ID — The unique ID of the sender in Firebase. You can find it in the [Firebase console](#): go to **Project settings** → **Cloud Messaging** and copy the value of the **Sender ID** field.

2. Add the following to the application element:

```
<meta-data android:name="ymp_firebase_default_app_id" android:value="APP_ID" />
```

APP_ID — ID of the app in Firebase. You can find it in the [Firebase console](#): go to the **Project settings**. In the **Your application** section copy the value of the **application ID** field.

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 1.1.0

Released August 15, 2022

- Updated versions of the AppMetrica Push SDK ([iOS 1.3.0](#), [Android 2.2.0](#)).
- Added support for Unity 2022.1.
- Fixed plugin operation on devices with Android 7 and below due to the `java.lang.NoClassDefFoundError: android.app.NotificationChannel` error.

Version 1.0.0

Released May 27, 2022

- Updated the AppMetrica Push SDK versions ([iOS 1.1.1](#), [Android 2.1.1](#)).
- Discontinued support for the AppMetrica Unity plugin lower than version 4.0.0.
- Added support for the [External Dependency Manager for Unity](#) to resolve dependencies.

Version 0.2.0

Released 11 July 2018

- Updated AppMetrica Push SDK versions ([iOS 0.6.0](#), [Android 1.1.0](#)).
- Stopped supporting AppMetrica Unity plugin lower than version 3.0.0.
- Stopped supporting iOS 7.

Version 0.1.1

Released 19 November 2017

- Updated AppMetrica Push SDK versions ([iOS 0.5.0](#), [Android 0.6.1](#)).
- Added support for Android 8.
- Added the in-app notifications support for iOS 10 and higher.
- Update the GCM and support libraries versions.
- The minimum version of the Android API is 14.

Version 0.1.0

Released 31 January 2017

- Integration with the AppMetrica SDK for the iOS and Android platforms.

Flutter

About the plugin

The [AppMetrica Push SDK for Flutter](#) plugin lets you send push notifications to complex user segments, flexibly set up the time, and run A/B testing of your campaigns. The number of push notifications is unlimited.

Key features

- Flexible targeting. Use all data about your users processed by AppMetrica to create complex segments for personalized communication.
- A/B testing. Try different combinations of push content for the same audience or test responses from different user segments.
- Custom push notification content. You can use various combinations of texts, images, icons, and calls to action.
- Planning. All push notifications can be scheduled for a specific time according to the recipients' time zone.
- Push API. Send individual notifications using custom triggers, including events outside the app.
- Detailed statistics. To carefully evaluate the effectiveness of your push campaign, you can track how each push notification affected user behavior and compare your push campaign with your key indicators.

Installation and initialization

To integrate AppMetrica Push SDK into Flutter, use the [AppMetrica Push SDK for Flutter](#) plugin:

1. Install the AppMetrica Push SDK plugin in your project. From the root of the project, run the command:

```
flutter pub add appmetrica_push_plugin
```

After adding the plugin, you'll see a line with the following dependency in the `pubspec.yaml` file:

```
dependencies:  
  appmetrica_push_plugin: ^0.3.0
```

2. Add `appmetrica_plugin` and `appmetrica_push_plugin` import:

```
import 'package:appmetrica_plugin/appmetrica_plugin.dart';  
import 'package:appmetrica_push_plugin/appmetrica_push_plugin.dart';
```

3. Initialize the AppMetrica SDK library using `AppMetrica.activate` and your API key:

```
AppMetrica.activate(AppMetricaConfig("insert_your_api_key_here"));
```

4. Initialize the AppMetrica Push SDK using `AppMetricaPush.activate` and your API key:

```
AppMetricaPush.activate();
```

5. To complete the integration of sending push notifications, use the documentation for each native platform: [Android](#) and [iOS](#).

iOS configuration

1. Add the SSL certificate to AppMetrica according to the [instructions](#).
2. Add **Capability Push Notifications** to your XCode project.

Changelog

Version 0.3.0

Released June 16, 2023

- The native android part of the plugin was rewritten on java due to the problems with kotlin versions.
- Updated dev_dependencies.

Version 0.2.0

Released August 29, 2022

- Updated versions of the AppMetrica Push SDK ([iOS 1.3.0](#), [Android 2.2.0](#)).

- Updated the appmetrica_plugin version to [1.0.1](#).

Version 0.1.0

Released March 15, 2022

Initial release. Supports most of the features available in the AppMetrica Push SDK. Native SDK versions:

- Android: 2.1.1.
- iOS: 1.1.

Cordova (not supported)

Installation and initialization



Attention:

Support and development of the plugin has been stopped. We do not guarantee the SDK will work correctly.

AppMetrica Push Cordova is a plugin for the [Cordova \(PhoneGap\)](#) platform that includes support for the AppMetrica Push SDK for Android and iOS through the JavaScript interface.

Before using the AppMetrica Push Cordova plugin, you need to [enable and initialize the AppMetrica Cordova plugin](#) version 0.2.0 or later.

Integrating the plugin

Step 1. [Add the supported platforms](#) to your project

Step 2. Add the plugin to the project by using one of the following console commands:

```
cordova plugin add yandex-appmetrica-push-plugin-cordova
```

or

```
cordova plugin add https://github.com/yandexmobile/metrica-push-plugin-cordova.git
```

When you add another supported platform to the project, the plugin automatically downloads the corresponding SDK library.

Step 3. *(For Android projects only)* Add the following to the application element of the `AndroidManifest.xml` file:

```
<meta-data android:name="ymp_gcm_project_number" android:value="number:SENDER_ID" />
```

For example:

1. Add the cordova-custom-config plugin:

```
cordova plugin add cordova-custom-config
```

2. Make the following changes to the `config.xml` file of the android project:

```
<config-file parent="./application" target="AndroidManifest.xml">
  <meta-data android:name="ymp_gcm_project_number" android:value="number:SENDER_ID" />
</config-file>
```

What is SENDER_ID?

`SENDER_ID` — Unique app identifier in GCM (Google Cloud Messaging).

Connection examples

In the example below:

- The JavaScript object with the configuration is created.
- The AppMetrica Cordova plugin is activated.
- The AppMetrica Push Cordova plugin is initialized.

- The push token of the device is output to the log.

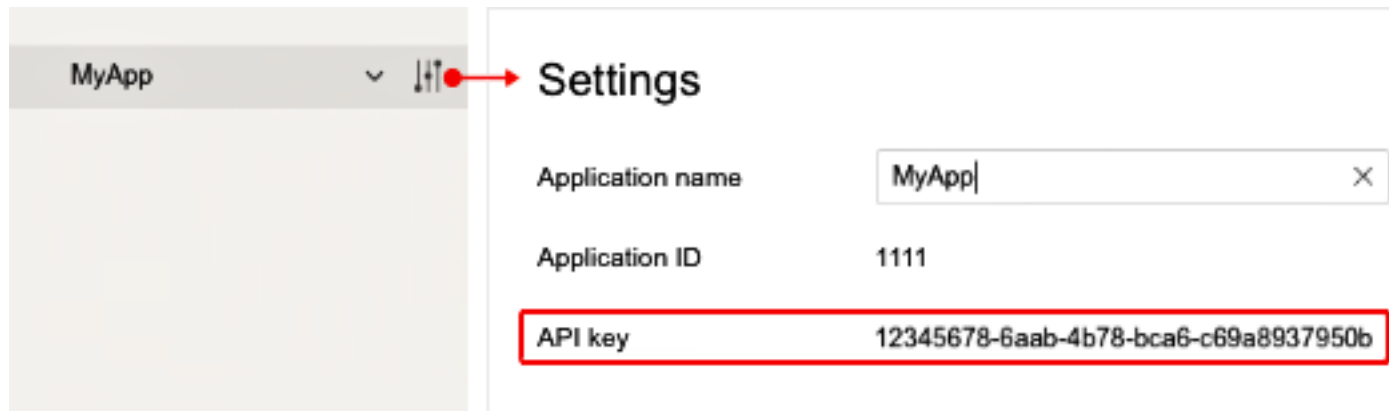
```
document.addEventListener('deviceready', onDeviceReady, false);
function onDeviceReady () {
  var configuration = {
    apiKey: 'Your API key here'
  }
  window.appMetrica.activate(configuration);

  window.appMetricaPush.init();
  window.appMetricaPush.getToken(function (token) {
    console.log("Token: " + token);
  });
}
```

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.



API methods

In the code, use `window.appMetricaPush` for accessing AppMetrica Push.

init()

```
init()
```

Initializes the AppMetrica Push Cordova plugin.

getToken()

```
getToken(function (token) {
  // Token has the String type.
})
```

Returns the push token of the device.

Related information

[Integration examples](#)

Changelog



Attention:

Support and development of the plugin has been stopped. We do not guarantee the SDK will work correctly.

Version 0.1.0

Released 27 December 2017

- Integration with AppMetrica SDKs ([iOS 0.5.0](#), [Android 0.6.1](#)).
- Added the [plugin integration example](#).