

RuStore Developer Documentation

Table of Contents

For developers	12
Developer account	13
Create a developer account	13
How to create a developer account for a company / individual entrepreneur	14
How to create a developer account for a foreign company	17
How to get a VK ID	24
How to add more developers	25
Adding Users and Selecting a Role	26
App publishing and verification	30
Declare app permissions	32
How to declare app permissions	33
Data types and categories	35
Types of permissions	39
Publish your app	53
How to upload a new app version	58
Adding TV version	
Uploading App Bundles	
ASO Recommendations	
Manage availability for your app	60
Customize your app release options	61
Auto & manual app release	61
Delayed app release	64
Staged app release	66
How to manage APK signing keys	69
Paid apps	71
App Review Guidelines	75
Age ratings	81
How to choose an appropriate category for your app	84
RuStore Ratings & Reviews	88
How to deal with negative reviews	89
How to capture HAR logs (Google Chrome)	90
Monetization	92
How to enable monetization on the RuStore	
Test payments	93
Monetization for foreign companies on RuStore	96
Create and fill out a monetization request	
Payments management	96
App Statistics	105
Monetization report	111
How to create a paid in-app product	113

How to create an app subscription	114
Ads and promotion	115
How to add a "Download from the RuStore" button	115
Requests for in-app events	
VK Ads on RuStore	117
How to set up an advertising campaign via VK Ads	117
How to add an app via VK Ads	118
Tools	123
General information	123
Connecting a tool	123
RuStore Remote Config	124
SDK and app configuration	125
Tracer	145
Quickstart	145
Migrating to a new version	148
Tracer Modules	149
Developer Documentation	156
RuStore Billing SDK	156
Kotlin	156
Quick Start	157
General Information	161
Payment functions availability	165
How to get up-to-date information on the product list	166
How to get the user's list of products	170
How to get purchase info	174
How to handle purchases	177
Server purchase validation	178
Purchase confirmation	179
Purchase cancellation	181
Error Handling	181
Consumption and cancellation scenario	183
Event Logging	184
Theme Changing	186
Error handling	186
SDK payment error codes	188
Migration to Payments SDK v2.2.0 and higher	190
1.x.x to 3.x.x Payments Migration	196
Payment Errors FAQ	202
RuStore SDK payments Release Notes	209
Godot	212
General Information	212
Embed in your project	212
Exporting a project with an enabled plugin	213

How to initialize the library	214
Payment functions availability	215
Getting the product list	215
How to get the user's list of products	218
Getting purchase info	220
How to handle purchases	222
Purchase confirmation	224
Purchase cancellation	226
Java	227
Quick Start	228
General Information	234
Payment functions availability	238
How to get the user's list of products	239
How to get the user's list of purchases	243
How to get purchase info	247
How to handle purchases	251
Server purchase validation	253
Purchase confirmation	254
Purchase cancellation	255
Consumption and cancellation scenario	256
Event Logging	257
Theme Changing	259
Error handling	260
Migration to Payments SDK from v.1.x.x to v3.x.x	261
Model update	262
RuStore SDK payments Release Notes	268
Unity	269
General Information	270
Payment functions availability	274
How to get the user's list of products	275
How to get the user's list of purchases	277
How to handle purchases	282
Purchase confirmation	284
Purchase cancellation	285
Confirmation and cancellation scenario	286
Error handling	287
Unity plug-in revision history	289
Flutter	290
General Information	291
Payment functions availability	294
How to get the list of products	295
How to get the user's list of purchases	298
How to handle purchases	301
Purchase confirmation	304

Unreal Engine	305
General Information	305
Calling CheckPurchasesAvailability	309
Getting the List of Purchases	319
Getting Purchase Info	326
How to handle purchases	329
Consumption and cancellation scenario	348
Error handling	349
React Native	352
General Information	352
Embed in your project	352
How to initialize the library	354
Payment functions availability	354
Getting the List of Products	355
Getting the List of Purchases	358
Getting Specific Purchase Info	359
How to handle purchases	360
Purchase confirmation	363
Purchase cancellation	364
Consumption and cancellation scenario	365
Unreal	366
Push notifications SDK	385
Kotlin	385
General	387
Checking ability to receive push notification	391
Methods for working with push token	392
Methods for working with push topics	393
Getting data from RuStore SDK	394
Notification structure	396
Event logging	398
Error handling	400
RuStore SDK for revision history	402
Unreal	404
General information	404
Connecting to project	404
Initialization	405
Initialization	405
Push notification availability check	409
Methods for working with push tokens	410
Retrieving the user's push token	410
Methods for working with push topics	412
Topic-based subscription for push notifications	412
Error handling	414
Unity	415

General	415
Connecting to the project	415
Checking ability to receive push notification	419
Push Token Management Methods	420
Push Topics Management Methods	421
Notification structure	424
Error handling	426
Flutter	427
General	428
Checking ability to receive push notification	431
How to work with push tokens and push notifications	432
Notification structure	434
Sending push notifications	436
Push notification examples	439
Sending push notifications by topic	443
Sending Push Notification to topic	443
Subscribing to push notifications by topic	444
Unsubscribing from push notifications by topic	446
Java	448
General	448
Methods for working with push topics	451
Notification structure	455
Event logging	458
Error handling	460
Defold	462
General	462
Connecting to the project	463
Push token and message methods	464
User-group focused Push Notifications API	
Testing push notification integration	
RuStore General Push Notifications SDK	466
Kotlin	466
App Feedback and Rating SDK	486
Android (Kotlin/Java)	486
App Feedback and Rating SDK	487
Importing SDK to your project	489
Creating RuStoreReviewManager	490
Getting ReviewInfo object	491
Starting app rating	492
Possible errors	493
RuStore Change History Feedback and Rating SDK	494
Unreal	495
General information	495
Conditions for proper SDK performance	495

Connecting to project	495
Initialization	496
Launching rating flow	496
Error handling	497
Unity	498
General	499
Importing SDK to your project	501
Creating RuStoreReviewManager	502
App release preparation	503
Starting app rating	504
Handling errors	505
Flutter	506
General	507
Importing SDK to your Project	509
Feedback request	510
Godot	
React Native	
App Update SDK	511
Kotlin/Java	511
Unreal	524
Unity	530
Flutter	541
Unreal	549
List of App Update Dependencies	561
List of available SDK	561
Compatible SDK versions	562
RuStore Geo	562
Terms of Use of RuStore Geo Functionality on the RuStore	563
General	567
Access to services	568
Search and geocoding services	570
Search for Places of Interest	570
Geocoding service	580
Address and Place Autocomplete	601
Map display services	610
Interactive map	611
Static map	757
Map styles	772
Routing Services	773
Route Planner	774
Distance Matrix	822
Reachable Area	830
Best Route Planner	843

Polyline Route Decoding	850
Maps Mobile SDK	855
Android	856
iOS	867
Additional services	880
Time zone detection	894
Postal code search	897
RuStore Deeplinks	910
Task API	911
Getting started with RuStore APIs	915
List of available RuStore API methods	915
How to sign up and start using API RuStore	919
Generating an authorization token	924
How to retrieve payment data using a purchase token	928
How to retrieve subscription data via a subscription token	943
How to retrieve subscription data by an invoice ID (V2)	951
Getting subscription data (v3)	
How to retrieve subscription status by a subscription token	958
Confirming subscription	
Uploading and Publishing Apps using the RuStore API	960
Creating a draft release	960
Manual publication	969
Changing publication settings	971
Getting app version status	975
Uploading screenshots	983
Uploading an app icon	986
Uploading an APK file	988
List of App Categories	992
Deleting a draft release	995
Submitting a draft app release for review	997
Using the RuStore API for Review Management	999
General Review Management Workflow	999
Getting app feedback	1000
Receiving feedback in .csv	1008
Leaving a reply to review	1011
Getting review response status	1013
Confirm subscription via a subscription token	1016
Editing review response	1018
Deleting review response	1020
Getting app rating	1021
Using RuStore API for Access Control	

For developers

The [RuStore Console](#) is a content management system designed to manage applications.

Users can log in to the system using a single VK ID.

Developers have access to the RuStore Console through the web interface.

Developer account

Create a developer account

You can create an account for a legal entity or an individual on the [RuStore Console](#).

Registration of legal entities/individual entrepreneurs is intended for:

- commercial and industrial companies;
- state-owned enterprises;
- professional developers.

When creating an account of a legal entity / individual entrepreneur, it is required to sign a registration form with a qualified e-signature.

Registration of individuals is intended for personal use and is perfect for:

- students;
- non-professionals;
- semi-professional developers.

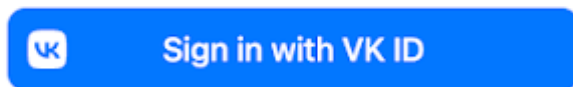
Individuals are required to confirm their accounts through remote identity verification.

How to create a developer account for a company / individual entrepreneur

1. Open the [RuStore Console](#) in your browser.
2. Click "Go to the Console".
3. Then click "Sign in with VK ID".

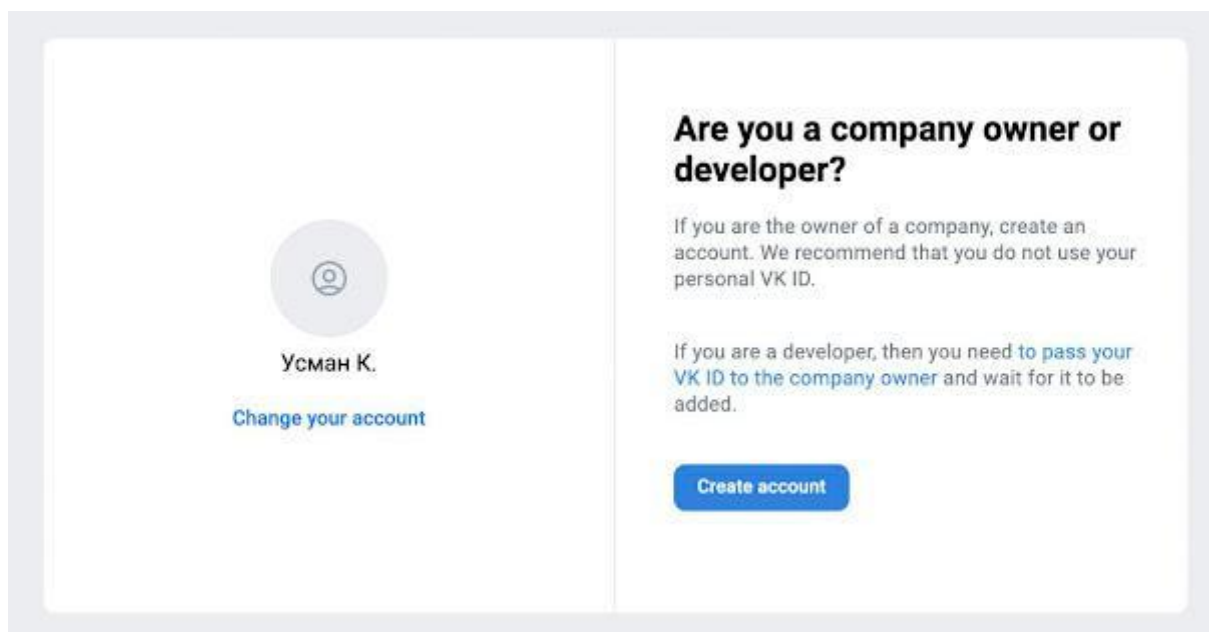


Log in to the Console



By continuing, you accept the distribution agreement, user agreement and privacy Policy

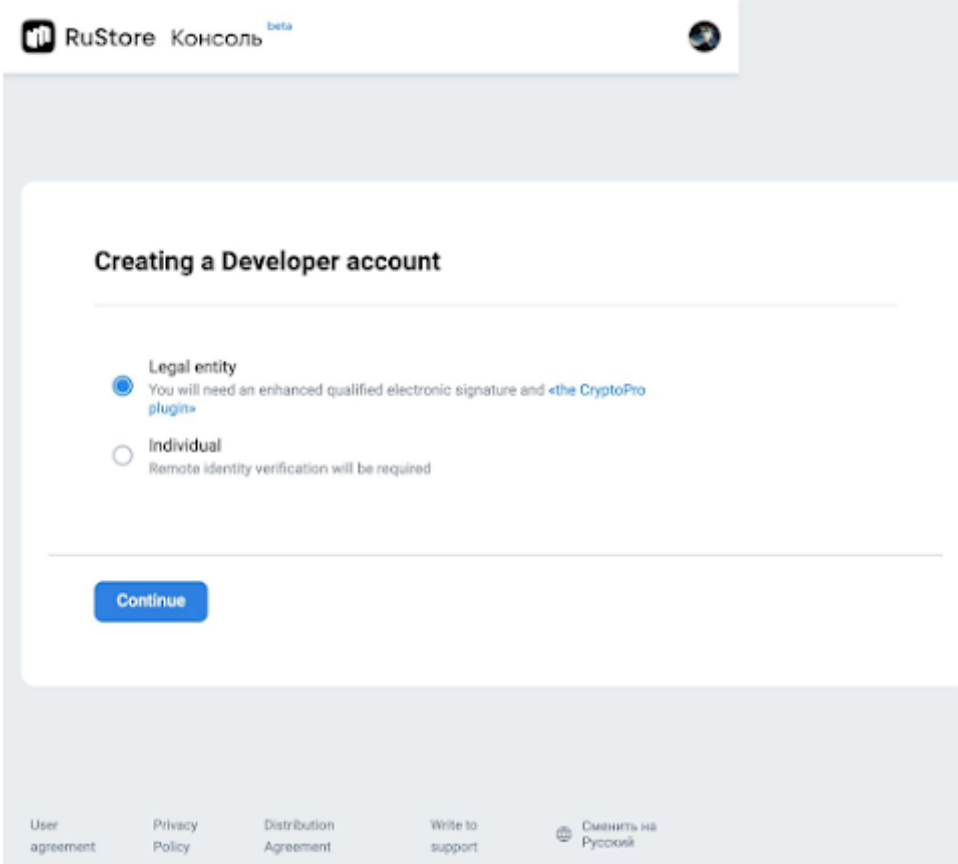
4. Enter the phone number connected to your VK ID or create a new account.
5. Click "Continue".
6. Select "Create account".



You can go to the account creation form from the [page](#) of the [RuStore Console](#). To do that, click "Create account".

7. Select "Legal Entity".

8. Click "Continue".



The screenshot shows the RuStore Console interface. At the top left, there is a logo for RuStore Консоль with a 'beta' tag. The main content area is titled "Creating a Developer account". Below the title, there are two radio button options: "Legal entity" (selected) and "Individual". The "Legal entity" option includes a note: "You will need an enhanced qualified electronic signature and «the CryptoPro plugin»". The "Individual" option includes a note: "Remote identity verification will be required". Below the options is a blue "Continue" button. At the bottom of the page, there are links for "User agreement", "Privacy Policy", "Distribution Agreement", "Write to support", and a language switcher "Сменить на Русский".

9. Fill in all the required fields.

10. Click "Submit".

Fill in the "Contact person" section. Specify your contact details so that we can get back to you quickly whenever so requested.

Developer registration form

Company information

Organization name (Enter the full, legal name of your organization)

Tencent LLC

Country

China

Organization phone number

+1 40 0061 7810

Tax or registration number

Organization website

www.tencent.com

Contact Person

Name

Guangming

Name

Taibai

VK ID ⓘ

785222546

Phone number

+7 (909) 860-13-33

E-mail

taibai.corp@gmail.com

Submit

How to create a developer account for a foreign company

1. Open the [RuStore Console](#) in your browser.
2. Click **Sign in to RuStore Console**.

Platform for your Android applications

Deliver your games and apps to millions of Android users

[Sign in to RuStore Console](#)

[Sign up](#)

3. Select **Legal entity**.

Please note that currently a foreign company can sign up as a legal entity only.

Creating a Developer account

Type of account

Legal entity
You will need an enhanced qualified e-signature and [«the Crypto plug-in»](#)

Individual
Remote identity verification will be required. Available only for residents of Russia

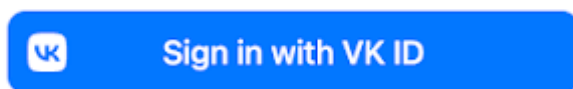
[Continue](#)

The Crypto Pro plug-in is only required for Russian companies.

4. Click **Sign in with VK ID**.



Log in to the Console



By continuing, you accept the distribution agreement, user agreement and privacy Policy

5. Enter the phone number linked to your VK ID or create a new account. You can use international phone numbers to create an account.

6. Click **Continue**.

7. Then click **Create Account**.

Please note that you can only link one company to one account. If a company account has already been created by another employee, please contact them to add your account to the company account.



Дмитрий С.

[Change account](#)

Are you a company owner or developer?

If you are a company owner, create an account. We recommend that you do not use your personal VK ID.

If you are a developer, you need to give your VK ID to the company owner and wait to be added.

[Create Account](#)

8. Click **Continue**.

Legal entity registration

Company information

Country of residence

China

Trade name (visible to users)

Tencent

Legal name

Tencent LLC

Tax or registration number

65780000802100123A

Company phone number

+1 40 0061 7810

Company website

www.tencent.com

Contact Person

Name

Guangming

Last name

Tai bai

Middle name (if included)

The representative's VK ID

785222546

The representative's phone number

+1 40 0061 7810

The representative's e-mail

taibai.corp@gmail.com



Documents


Please attach documents in English or Russian

Link to your app in other stores (Google Play, GetApps or AppGallery)

<https://play.google.com/store/apps/details?id=com.plarium.mech&hl=ru&gl=US>

Documents of the company incorporation



 Establishment
8 MB ✓ Uploaded 


 Document
8 MB ✓ Uploaded 

Upload 1-3 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

[Choose file](#)



Partner representative's authority documents for RuStore registration

 Registration
8 MB ✓ Uploaded 

 Authority
8 MB Uploaded

Upload 1-3 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

Partner's domain name/trademark/logo registration certificate

 Certificate
8 MB ✓ Uploaded 

Upload 1-5 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

[Choose file](#)

[Submit a request for moderation](#)

9. Then fill in all the required fields.
 - The trade name is the name the RuStore and the www.rustore.ru users will see. Please enter a name that is familiar to your users, or the one you use for promotion. This will help users easily discover you on the RuStore catalog.

Note. We may ask you to verify your rights to the specified trade name. You are not required to have your rights legally registered, however, in this case, we will ask you to ensure that the use of this trade name does not violate the rights of third parties.

- The legal name is your company name, which is indicated in its constituent documents. Please be careful when filling out this field, as we will check whether this name corresponds to the one specified in the documents you provided.
- When filling in the Contact person section, specify the up-to-date contact details so that we can get back to you quickly whenever so required.
- The Phone Number and Email sections are intended for internal communications with RuStore and will not be visible to users. Please enter the international phone number in the **Representative number** field.
- Make sure that the **Tax or registration number** field matches the information in your constituent documents.

10. In the Documents section, attach all the required documents in English or in Russian:

Note. Please provide certified translation of your company documents.

The Link to your app in other stores is a mandatory field. If your app has not been published in other stores yet, you can attach a link to an official website where your app is posted.

- company establishment documents that confirm that the company is registered within the corresponding jurisdiction, for example: business license, Charter certified by a government agency, certificate of registration, and others.
- documents confirming partner representative's authority to sign up on the RuStore, e.g. power of attorney, appointment letter, Charter, etc. Please make sure that the representative's details correspond to the contact person specified in the section above;
- partner's domain name/trademark/logo registration certificate, for example: registration certificate, license agreement, copyright holder permission, etc. If there are no registered rights, you must provide a letter confirming the absence of logo, domain name, trademarks infringement and unregistered rights letter of guarantee.

11. Click **Submit a request for app review**. Your request will be reviewed and approved by a moderator.



Your sign-up request has been accepted

Access to your developer account will be granted within a few days. We will send an email to confirm your account creation

[Back to main page](#)

Please make sure that your sign-up request meets the RuStore Guidelines. Otherwise we will notify you via email of your request status.

How to install the CryptoPro plug-in

The CryptoPro EDS Browser plug-in is designed to create and verify digital signatures on web pages. It supports a wide range of algorithms, both built into the OS and additionally installed.

Note. The CryptoPro plug-in is required for Russian companies only.

Follow the steps below to install the CryptoPro EDS Browser plug-in:

1. Download and install a cryptographic provider for generating and verifying digital signatures according to GOST algorithms. We recommend [CryptoPro CSP](#). Install the certificate from the provider's store.
2. Download and install the [CryptoPro EDS Browser plug-in](#) for the interaction of web pages in your browser with the crypto provider in the OS. This application is designed to create and verify digital signatures on web pages. [How to install CryptoPro ES Browser plug-in on Windows?](#)
3. Check the created digital signature on a [special site](#) to make sure that everything is set correctly and all system elements interact steadily.

How to get a VK ID

VK ID is a single account for all VK services. If you already have a VKontakte profile, it will automatically become your VK ID.

If you do not want to link your [RuStore Console](#) account to a personal VKontakte profile or you do not have one, you can create a new profile and use it as a corporate one.

How to get a VK ID:

1. Log in to your [VKontakte](#) profile.
2. Go to the "Settings" section.
3. Find the "Page address" block.
4. Click "Edit" to see the page number if you previously assigned an alphanumeric identifier.

This is your VK ID.

How to add more developers

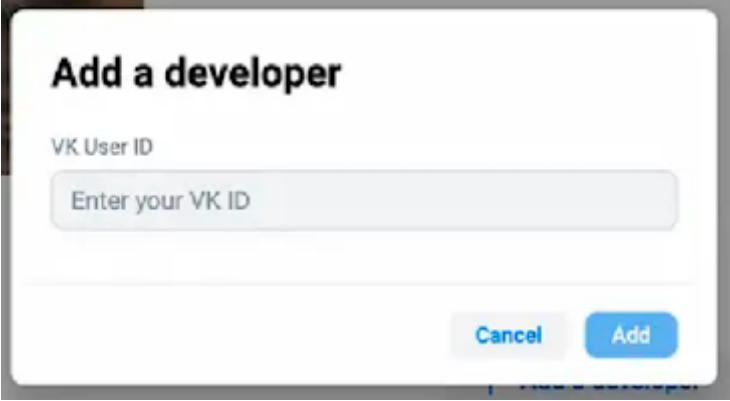
The company employee, whose VK ID was specified upon signing up, is considered the company owner. Each company can have only one owner.

A company owner can only register one Company, hence, it is recommended to create a single corporate VK ID (of an individual) for app management.

The company owner can create applications and assign / add developers for each app by their VK IDs.

How to add an additional developer

1. The developer who needs to be added to the app management should be authorized via his VK ID (of an individual) on the [RuStore Console](#) (no further operations on the [RuStore Console](#) are required at this step).
2. The developer should send his VK ID as "12345678" to the company owner. [How to get a VK ID?](#)
3. The company owner should add the developer's VK ID to the corresponding app.



Add a developer

VK User ID

Enter your VK ID

Cancel Add

Once there, the developer will have access to the corresponding application. He can change the app description and data, or either create, prepare and publish new app versions.

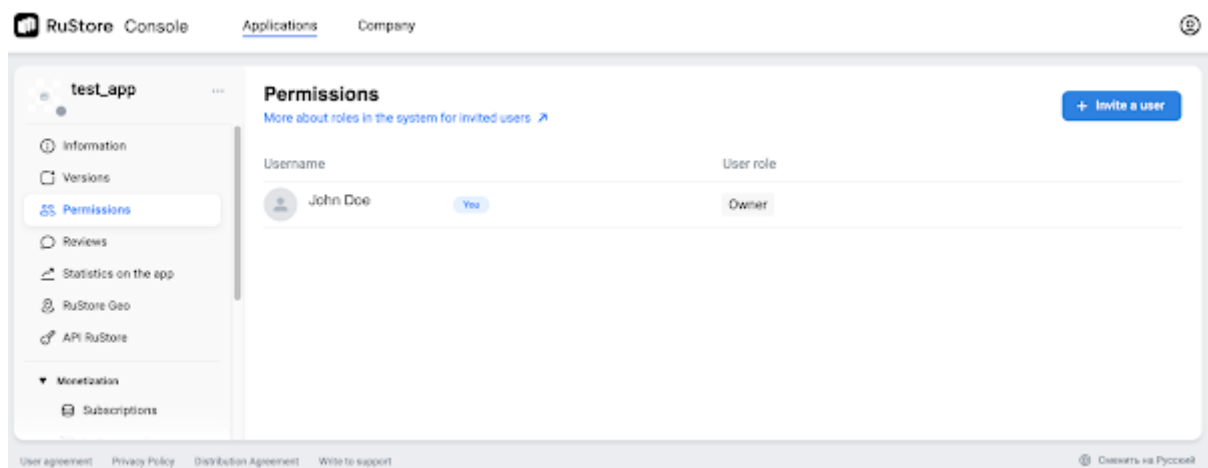
The developer is not allowed to register companies, create new apps, or add other developers to applications.

Adding Users and Selecting a Role

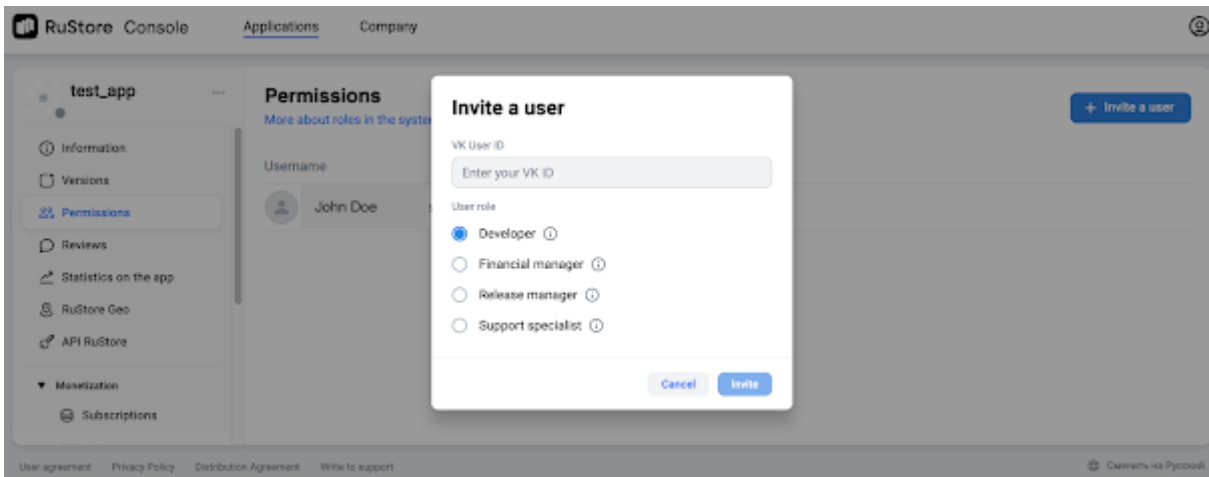
To enhance the management of application operations for employees, you can impose restrictions and assign predefined roles. This approach provides clear configuration options for their interactions with the applications. For example, allow an employee to only publish applications or make refunds.

How to Add Users

1. Select an application.
2. Go to **Permissions** in the left menu.
3. Click **Invite User** at the top.



4. Specify the user's VK ID. [How to get your VK ID?](#)
5. Select a user role. You can only select one option from the list.



6. Click **Invite**.

How to Edit Permissions

1. Select a user.

2. Click  next to it.

3. Select a new role.

VK User ID

Enter your VK ID

User role

- Developer ⓘ
- Financial manager ⓘ
- Release manager ⓘ
- Support specialist ⓘ

Cancel

Save

4. Click **Save**.

How to Delete a User

1. Select a user.
2. Click **X** next to it.
3. Click **Delete**.

Please be aware that once a user is deleted, it cannot be undone, and you will need to re-add the user if he was deleted by mistake.

What are the available role options to choose from?

There are five possible user roles in RuStore:

- Company owner;
- Developer;
- Financial Manager;
- Release manager;
- Support specialist.

	Company owner	Release manager	Developer	Financial Manager	Support specialist
App distribution					
View app download statistics	✓	✓	✓	✓	✓
Upload, edit app versions, submit them for moderation	✓	✓	✓		
Invite users and manage permissions	✓	✓			
Create and delete applications	✓				
Services					
Connect push notifications, view statistics	✓	✓	✓		

Connect RuStore Geo	✓	✓	✓		
Connect RuStore API	✓				
Monetization					
Create subscriptions, inn-app products	✓	✓		✓	
Manage payments, make refunds	✓			✓	
View financial statistics	✓			✓	
Fill in and edit company bank details	✓				
Enable monetization	✓				
Support					
Check reviews and respond to them	✓	✓			✓

App publishing and verification

Once you've created your [RuStore Console](#) developer account, you can publish your application on the RuStore.

How much does it cost to publish an application on the RuStore

RuStore does not charge for hosting of both free or paid apps.

Declare app permissions

RuStore is primarily committed to ensure user safety. To keep user data safe, RuStore checks all requested permissions for each application. Furthermore, we have launched a system capable of recognizing user data types collected and transmitted by the application.

Declaring permissions and providing information about the requested data increases transparency and helps the store and users understand how user data is processed and what it is used for.

How to declare app permissions

To ensure safety of user data, the [RuStore Console](#) features a new User Data Safety section that requires a developer to provide information about categories and data types collected by the app, as well as about dangerous permissions. This section is divided into two subsections, i.e. “Dangerous permissions” and “Data types and categories”.

The “Dangerous permissions” subsection provides a list of dangerous permissions and will only be displayed if your app possesses any of them.

In case there are any dangerous permissions detected in your app, you need to fill in the “Purpose” field, specifying the reason why your app requests this or that permission.

Then, specify all data types your app requests the access to in The “Data types and categories” subsection.

For more information about available data types, see [Categories and data types](#).

The “Data types and categories” section is available for all app versions regardless of a permission type.

Categories ⓘ

To learn how to choose a category for an app, see [the developer's guide](#)

Main

Выберите категорию



Additional information

Выберите категорию



Age category

Выберите категорию



Once you have completed all the required fields and sent it for moderation, the RuStore team will start verifying your application.

Please make sure to check out RuStore [Guidelines](#) before submitting your app for moderation.

If there are any dangerous permissions detected, the RuStore team will carry out a thorough check.

User Data Safety

Data types and categories

This information will make it clear to users how you use their data and keep them safe. Read more in the [Developer's guide](#)

Types of user data your app collects

If your app will be proved not to comply with the permission rules, we will get back to you by email for some clarifications.

You can proceed to app release once your app has successfully completed the Developer Guidelines Check.

Data types and categories

Data types and categories are required to make it clear to the user how your app uses their personal data and ensures its safety.

Information on data types and categories collected by your app will be available to users on the homepage of RuStore.

When uploading or updating an app, developers will be asked to specify all user data collected and transferred by the app, namely:

- all user data types collected and/or transferred by the app:
- all user data sent from a user's device via libraries or SDK regardless of the recipient, i.e. a developer or third party;
- all user data sent from the developer's server to a third party or app on the same device.

Categories and data types

Category	Data type	Description
Location	Approximate location	User device's physical location to an area greater than or equal to 3 square kilometers, such as the city the user is in.
	Precise location	User device's physical location within an area less than 3 square kilometers.
Personal info	Name	User first or last name, or nickname.
	Email address	User email address.
	User IDs	Identifiers that relate to an identifiable person. For example, an account ID, account number, or account name.
	Address	User address, such as a mailing or home address.
	Phone number	User phone number.
	Race and ethnicity	Information about the user's race or ethnicity.

	Political or religious beliefs	Information about the user's political or religious beliefs.
	Sexual orientation	Information about the user's sexual orientation.
	Other info	Any other personal information such as date of birth, gender identity, veteran status, etc.
Financial info	User payment info	Information about the user's financial accounts, such as credit card number.
	Purchase history	Information about purchases or transactions the user has made.
	Credit score	Information about the user's credit. For example, the user's credit history or credit score.
	Other financial info	Any other financial information, such as the user's salary or debts.
Health and fitness	Health info	Information about the user's health, such as medical records or symptoms.
	Fitness info	Information about the user's fitness, such as exercise or other physical activity.
Messages	Emails	User emails, including the email subject line, sender, recipients, and the content of the email.
	SMS or MMS	User text messages, including the sender, recipients, and the content of the message.
	Other in-app messages	Any other types of messages. For example, instant messages or chat content.
Photos and videos	Photos	User photos.
	Videos	User videos.
Audio files	Voice or sound recordings	User voice, such as a voicemail or a sound recording.
	Music files	User music files.

	Other audio files	Any other audio files the user created or provided.
Files and docs	Files and docs	User files or documents, or information about the user's files or documents, such as file names.
Calendar	Calendar events	Information from the user's calendar, such as events, event notes, and attendees.
Contacts	Contacts	Information about the user's contacts, such as contact names, message history, and social graph information like usernames, contact recency, contact frequency, interaction duration, and call history.
App activity	App interactions	Information about how the user interacts with the app. For example, the number of times the user visits a page or sections the user taps on.
	In-app search history	Information about what the user has searched for in the app.
	Installed apps	Information about the apps installed on the user's device.
	Other user-generated content	Any other content the user generated that is not listed here, or in any other section. For example, bios, notes, or open-ended responses.
	Other actions	Any other activity or actions in-app not listed here, such as gameplay, likes, and dialog options.
Web browsing	Web browsing history	Information about the websites the user has visited.
App info and performance	Crash logs	Crash data from the app. For example, the number of times the app has crashed on the device or other information directly related to a crash.
	Diagnostics	Information about the performance of the app on the device. For example, battery life, loading time, latency, framerate, or any technical diagnostics.

	Other app performance data	Any other app performance data not listed here.
Device or other IDs	Device or other IDs	Identifiers that relate to an individual device, browser, or app. For example, an IMEI number, MAC address, Widevine Device ID, Firebase installation ID, or advertising identifier.

Types of permissions

All the permissions that are required by your app are subject to primarily check once added to the [RuStore Console](#).

The developer is obliged to declare all permissions, except for **Normal**, i.e. [Dangerous](#), [Special](#), [Signature](#).

*The apps that use permissions from the “**Not for use by third-party applications**” category will be blocked automatically.*

[Dangerous](#), [Special](#) and [Signature](#) permissions are considered altogether as “Dangerous permissions”.

All the available permissions are listed below.

Permission levels

Level	Name	Description
Not for use by third-party applications	ACCESS_CHECKIN_PROPERTIES	Allows read/write access to the "properties" table in the checkin database, to change values that get uploaded.
	ACCOUNT_MANAGER	Allows applications to call into AccountAuthenticators.
	BIND_APPWIDGET	Allows an application to tell the AppWidget service which application can access AppWidget's data.
	BLUETOOTH_PRIVILEGED	Allows applications to pair bluetooth devices without user interaction, and to allow or disallow phonebook access or message access.
	BROADCAST_PACKAGE_REMOVE D	Allows an application to broadcast a notification that an application package has been removed.
	BROADCAST_SMS	Allows an application to broadcast an SMS receipt notification.
	CALL_PRIVILEGED	Allows an application to call any phone number, including emergency numbers, without going through the Dialer user interface for the user to confirm the call being placed.

CAPTURE_AUDIO_OUTPUT	Allows an application to capture audio output.
CHANGE_COMPONENT_ENABLED_STATE	Allows an application to change whether an application component (other than its own) is enabled or not.
CONTROL_LOCATION_UPDATES	Allows enabling/disabling location update notifications from the radio.
DELETE_PACKAGES	Allows an application to delete packages.
DIAGNOSTIC	Allows applications to RW to diagnostic resources.
DUMP	Allows an application to retrieve state dump information from system services.
FACTORY_TEST	Run as a manufacturer test application, running as the root user.
INSTALL_LOCATION_PROVIDER	Allows an application to install a location provider into the Location Manager.
INSTALL_PACKAGES	Allows an application to install packages.
LOCATION_HARDWARE	Allows an application to use location features in hardware, such as the geofencing api.
MANAGE_WIFI_INTERFACES	Allows applications to get notified when a Wi-Fi interface request cannot be satisfied without tearing down one or more other interfaces, and provide a decision whether to approve the request or reject it.
MANAGE_WIFI_NETWORK_SELECTION	This permission is used to let OEMs grant their trusted app access to a subset of privileged wifi APIs to improve wifi performance.
MASTER_CLEAR	Not for use by third-party applications.
MEDIA_CONTENT_CONTROL	Allows an application to know what content is playing and control its playback.
MODIFY_PHONE_STATE	Allows modification of the telephony state - power on, mmi, etc.
MOUNT_FORMAT_FILESYSTEMS	Allows formatting file systems for removable storage.
MOUNT_UNMOUNT_FILESYSTEMS	Allows mounting and unmounting file systems for removable storage.

OVERRIDE_WIFI_CONFIG	Allows an application to modify any wifi configuration, even if created by another application.
READ_INPUT_STATE	This constant was deprecated in API level 16. The API that used this permission has been removed.
READ_LOGS	Allows an application to read the low-level system log files.
REBOOT	Required to be able to reboot the device.
REQUEST_COMPANION_PROFILE_APP_STREAMING	Allows application to request to be associated with a virtual display capable of streaming Android applications (AssociationRequest.DEVICE_PROFILE_APP_STREAMING) by CompanionDeviceManager.
REQUEST_COMPANION_PROFILE_AUTOMOTIVE_PROJECTION	Allows application to request to be associated with a vehicle head unit capable of automotive projection (AssociationRequest.DEVICE_PROFILE_AUTOMOTIVE_PROJECTION) by CompanionDeviceManager.
REQUEST_COMPANION_PROFILE_COMPUTER	Allows application to request to be associated with a computer to share functionality and/or data with other devices, such as notifications, photos and media (AssociationRequest.DEVICE_PROFILE_COMPUTER) by CompanionDeviceManager.
SEND_RESPOND_VIA_MESSAGE	Allows an application (Phone) to send a request to other applications to handle the respond-via-message action during incoming calls.
SET_ALWAYS_FINISH	Allows an application to control whether activities are immediately finished when put in the background.
SET_ANIMATION_SCALE	Modify the global animation scaling factor.
SET_DEBUG_APP	Configure an application for debugging.
SET_PROCESS_LIMIT	Allows an application to set the maximum number of (not needed) application processes that can be running.
SET_TIME	Allows applications to set the system time directly.
SET_TIME_ZONE	Allows applications to set the system time zone directly.

	SIGNAL_PERSISTENT_PROCESSES	Allow an application to request that a signal be sent to all persistent processes.
	START_FOREGROUND_SERVICES_FROM_BACKGROUND	Allows an application to start foreground services from the background at any time.
	STATUS_BAR	Allows an application to open, close, or disable the status bar and its icons.
	UNINSTALL_SHORTCUT	Don't use this permission in your app.
	UPDATE_DEVICE_STATS	Allows an application to update device statistics.
	WRITE_APN_SETTINGS	Allows applications to write the apn settings and read sensitive fields of an existing apn settings like user and password.
	WRITE_GSERVICES	Allows an application to modify the Google service map.
	WRITE_SECURE_SETTINGS	Allows an application to read or write the secure system settings.
Dangerous	ACCEPT_HANDOVER	Allows a calling app to continue a call which was started in another app.
	ACCESS_BACKGROUND_LOCATION	Allows an app to access location in the background.
	ACCESS_COARSE_LOCATION	Allows an app to access approximate location.
	ACCESS_FINE_LOCATION	Allows an app to access precise location.
	ACCESS_MEDIA_LOCATION	Allows an application to access any geographic locations persisted in the user's shared collection.
	ACTIVITY_RECOGNITION	Allows an application to recognize physical activity.
	ADD_VOICEMAIL	Allows an application to add voicemails into the system.
	ANSWER_PHONE_CALLS	Allows the app to answer an incoming phone call.
	BLUETOOTH_ADVERTISE	Required to be able to advertise to nearby Bluetooth devices.
	BLUETOOTH_CONNECT	Required to be able to connect to paired Bluetooth devices.
	BLUETOOTH_SCAN	Required to be able to discover and pair nearby Bluetooth devices.

BODY_SENSORS	Allows an application to access data from sensors that the user uses to measure what is happening inside their body, such as heart rate.
BODY_SENSORS_BACKGROUND	Allows an application to access data from sensors that the user uses to measure what is happening inside their body, such as heart rate.
CALL_PHONE	Allows an application to access data from sensors that the user uses to measure what is happening inside their body, such as heart rate.
CAMERA	Required to be able to access the camera device.
GET_ACCOUNTS	Allows access to the list of accounts in the Accounts Service.
NEARBY_WIFI_DEVICES	Allows an instant app to create foreground services.
POST_NOTIFICATIONS	Allows interaction across profiles in the same profile group.
PROCESS_OUTGOING_CALLS	This constant was deprecated in API level 29. Applications should use CallRedirectionService instead of the Intent.ACTION_NEW_OUTGOING_CALL broadcast.
READ_CALENDAR	Allows an application to read the user's calendar data.
READ_CALL_LOG	Allows an application to read the user's call log.
READ_CONTACTS	Allows an application to read the user's contacts data.
READ_EXTERNAL_STORAGE	Allows an application to read from external storage.
READ_MEDIA_AUDIO	Allows an application to read audio files from external storage.
READ_MEDIA_IMAGES	Allows an application to read image files from external storage.
READ_MEDIA_VIDEO	Allows an application to read video files from external storage.
READ_PHONE_NUMBERS	Allows read access to the device's phone number(s).

READ_PHONE_STATE	Allows read only access to phone state, including the current cellular network information, the status of any ongoing calls, and a list of any PhoneAccounts registered on the device.
READ_SMS	Allows an application to read SMS messages.
RECEIVE_MMS	Allows an application to monitor incoming MMS messages.
RECEIVE_SMS	Allows an application to receive SMS messages.
RECEIVE_WAP_PUSH	Allows an application to receive WAP push messages.
RECORD_AUDIO	Allows an application to record audio.
SEND_SMS	Allows an application to send SMS messages.
USE_SIP	Allows an application to use SIP service.
UWB_RANGING	Required to be able to range to devices using ultra-wideband.
WRITE_CALENDAR	Allows an application to write the user's calendar data.
WRITE_CALL_LOG	Allows an application to write (but not read) the user's call log data.
WRITE_CONTACTS	Allows an application to write the user's contacts data.
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage.
ACCESS_BLOBS_ACROSS_USERS	Allows an application to access data blobs across users.
BATTERY_STATS	Allows an application to collect battery statistics
BIND_ACCESSIBILITY_SERVICE	Must be required by an AccessibilityService, to ensure that only the system can bind to it.
BIND_AUTOFILL_SERVICE	Must be required by a AutofillService, to ensure that only the system can bind to it.
BIND_CALL_REDIRECTION_SERVICE	Must be required by a CallRedirectionService, to ensure that only the system can bind to it.

BIND_CARRIER_MESSAGING_CLIENT_SERVICE	A subclass of CarrierMessagingClientService must be protected with this permission.
BIND_CARRIER_SERVICES	The system process that is allowed to bind to services in carrier apps will have this permission.
BIND_CHOOSER_TARGET_SERVICE	This constant was deprecated in API level 30. For publishing direct share targets, please follow the instructions in https://developer.android.com/training/sharing/receive.html#providing-direct-share-targets instead.
BIND_COMPANION_DEVICE_SERVICE	Must be required by any CompanionDeviceServices to ensure that only the system can bind to it.
BIND_CONDITION_PROVIDER_SERVICE	Must be required by a ConditionProviderService, to ensure that only the system can bind to it.
BIND_CONTROLS	Allows SystemUI to request third party controls.
BIND_DEVICE_ADMIN	Must be required by device administration receiver, to ensure that only the system can interact with it.
BIND_DREAM_SERVICE	Must be required by an DreamService, to ensure that only the system can bind to it.
BIND_INCALL_SERVICE	Must be required by a InCallService, to ensure that only the system can bind to it.
BIND_INPUT_METHOD	Must be required by an InputMethodService, to ensure that only the system can bind to it.
BIND_MIDI_DEVICE_SERVICE	Must be required by an MidiDeviceService, to ensure that only the system can bind to it.
BIND_NFC_SERVICE	Must be required by a HostApuService or OffHostApuService to ensure that only the system can bind to it.
BIND_NOTIFICATION_LISTENER_SERVICE	Must be required by an NotificationListenerService, to ensure that only the system can bind to it.

BIND_PRINT_SERVICE	Must be required by a PrintService, to ensure that only the system can bind to it.
BIND_QUICK_ACCESS_WALLET_SERVICE	Must be required by a QuickAccessWalletService to ensure that only the system can bind to it.
BIND_QUICK_SETTINGS_TILE	Allows an application to bind to third party quick settings tiles.
BIND_REMOTEVIEWS	Must be required by a RemoteViewsService, to ensure that only the system can bind to it.
BIND_SCREENING_SERVICE	Must be required by a CallScreeningService, to ensure that only the system can bind to it.
BIND_TELECOM_CONNECTION_SERVICE	Must be required by a ConnectionService, to ensure that only the system can bind to it.
BIND_TEXT_SERVICE	Must be required by a TextService (e.g. SpellCheckerService) to ensure that only the system can bind to it.
BIND_TV_INPUT	Must be required by a TvInputService to ensure that only the system can bind to it.
BIND_TV_INTERACTIVE_APP	Must be required by a TvInteractiveAppService to ensure that only the system can bind to it.
BIND_VISUAL_VOICEMAIL_SERVICE	Must be required by a link VisualVoicemailService to ensure that only the system can bind to it.
BIND_VOICE_INTERACTION	Must be required by a VoiceInteractionService, to ensure that only the system can bind to it.
BIND_VPN_SERVICE	Must be required by a VpnService, to ensure that only the system can bind to it.
BIND_VR_LISTENER_SERVICE	Must be required by an VrListenerService, to ensure that only the system can bind to it.
BIND_WALLPAPER	Must be required by a WallpaperService, to ensure that only the system can bind to it.

CHANGE_CONFIGURATION	Allows an application to modify the current configuration, such as locale.
CLEAR_APP_CACHE	Allows an application to clear the caches of all installed applications on the device.
DELETE_CACHE_FILES	Old permission for deleting an app's cache files, no longer used, but signals for us to quietly ignore calls instead of throwing an exception.
GET_ACCOUNTS_PRIVILEGED	Allows access to the list of accounts in the Accounts Service.
GLOBAL_SEARCH	This permission can be used on content providers to allow the global search system to access their data.
INSTANT_APP_FOREGROUND_SERVICE	Allows an instant app to create foreground services.
INTERACT_ACROSS_PROFILES	Allows interaction across profiles in the same profile group.
LAUNCH_MULTI_PANE_SETTINGS_DEEP_LINK	An application needs this permission for Settings.ACTION_SETTINGS_EMBED_DEEP_LINK_ACTIVITY to show its Activity embedded in Settings app.
LOADER_USAGE_STATS	Allows a data loader to read a package's access logs.
MANAGE_DOCUMENTS	Allows an application to manage access to documents, usually as part of a document picker.
MANAGE_EXTERNAL_STORAGE	Allows an application a broad access to external storage in scoped storage.
MANAGE_MEDIA	Allows an application to modify and delete media files on this device or any connected storage device without user confirmation.
MANAGE_ONGOING_CALLS	Allows to query ongoing call details and manage ongoing calls
PACKAGE_USAGE_STATS	Allows an application to collect component usage statistics
READ_ASSISTANT_APP_SEARCH_DATA	Allows an application to query over global data in AppSearch that's visible to the ASSISTANT role.

	READ_HOME_APP_SEARCH_DATA	Allows an application to query over global data in AppSearch that's visible to the HOME role.
	READ_PRECISE_PHONE_STATE	Allows read only access to precise phone state.
	READ_VOICEMAIL	Allows an application to read voicemails in the system.
	REQUEST_COMPANION_SELF_MANAGED	Allows an application to create a "self-managed" association.
	REQUEST_INSTALL_PACKAGES	Allows an application to request installing packages.
	SCHEDULE_EXACT_ALARM	Allows applications to use exact alarm APIs.
	START_VIEW_APP_FEATURES	Allows the holder to start the screen with a list of app features.
	START_VIEW_PERMISSION_USAGE	Allows the holder to start the permission usage screen for an app.
	SUBSCRIBE_TO_KEYGUARD_LOCKED_STATE	Allows an application to subscribe to keyguard locked (i.e., showing) state.
	SYSTEM_ALERT_WINDOW	Allows an app to create windows using the <code>type.WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY</code> , shown on top of all other apps.
	USE_ICC_AUTH_WITH_DEVICE_IDENTIFIER	Allows to read device identifiers and use ICC based authentication like EAP-AKA.
	WRITE_SETTINGS	Allows an application to read or write the system settings.
	WRITE_VOICEMAIL	Allows an application to modify and remove existing voicemails in the system.
Normal	ACCESS_LOCATION_EXTRA_COMMANDS	Allows an application to access extra location provider commands.
	BLUETOOTH	Allows applications to connect to paired bluetooth devices.
	BLUETOOTH_ADMIN	Allows applications to discover and pair bluetooth devices.
	BROADCAST_STICKY	Allows an application to broadcast sticky intents.

CALL_COMPANION_APP	Allows an app which implements the InCallService API to be eligible to be enabled as a calling companion app.
CHANGE_NETWORK_STATE	Allows applications to change network connectivity state.
CHANGE_WIFI_MULTICAST_STATE	Allows applications to enter Wi-Fi Multicast mode.
CHANGE_WIFI_STATE	Allows applications to change Wi-Fi connectivity state.
DELIVER_COMPANION_MESSAGES	Allows an application to deliver companion messages to system
DISABLE_KEYGUARD	Allows applications to disable the keyguard if it is not secure.
EXPAND_STATUS_BAR	Allows an application to expand or collapse the status bar.
FOREGROUND_SERVICE	Allows a regular application to use Service.startForeground.
GET_PACKAGE_SIZE	Allows an application to find out the space used by any package.
HIDE_OVERLAY_WINDOWS	Allows an app to prevent non-system-overlay windows from being drawn on top of it
HIGH_SAMPLING_RATE_SENSORS	Allows an app to access sensor data with a sampling rate greater than 200 Hz.
INSTALL_SHORTCUT	Allows an application to install a shortcut in Launcher.
INTERNET	Allows applications to open network sockets.
KILL_BACKGROUND_PROCESSES	Allows an application to call ActivityManager.killBackgroundProcesses(String).
MANAGE_OWN_CALLS	Allows a calling application which manages its own calls through the self-managedConnectionService APIs.
MODIFY_AUDIO_SETTINGS	Allows an application to modify global audio settings.
NFC	Allows applications to perform I/O operations over NFC.

NFC_PREFERRED_PAYMENT_INFO	Allows applications to receive NFC preferred payment service information.
NFC_TRANSACTION_EVENT	Allows applications to receive NFC transaction events.
QUERY_ALL_PACKAGES	Allows query of any normal app on the device, regardless of manifest declarations.
READ_BASIC_PHONE_STATE	Allows read only access to phone state with a non dangerous permission, including the information like cellular network type, software version.
READ_NEARBY_STREAMING_POLICY	Allows an application to read nearby streaming policy.
READ_SYNC_SETTINGS	Allows applications to read the sync settings.
READ_SYNC_STATS	Allows applications to read the sync stats.
RECEIVE_BOOT_COMPLETED	Allows an application to receive the Intent.ACTION_BOOT_COMPLETED that is broadcast after the system finishes booting.
REORDER_TASKS	Allows an application to change the Z-order of tasks.
REQUEST_COMPANION_PROFILE_WATCH	Allows app to request to be associated with a device via CompanionDeviceManager as a "watch"
REQUEST_COMPANION_RUN_IN_BACKGROUND	Allows a companion app to run in the background.
REQUEST_COMPANION_START_FOREGROUND_SERVICES_FROM_BACKGROUND	Allows a companion app to start a foreground service from the background.
REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	Allows a companion app to use data in the background.
REQUEST_DELETE_PACKAGES	Allows an application to request deleting packages.
REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	Permission an application must hold in order to use Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS.

	REQUEST_OBSERVE_COMPANION_DEVICE_PRESENCE	Allows an application to subscribe to notifications about the presence status change of their associated companion device
	REQUEST_PASSWORD_COMPLEXITY	Allows an application to request the screen lock complexity and prompt users to update the screen lock to a certain complexity level.
	SET_ALARM	Allows an application to broadcast an Intent to set an alarm for the user.
	SET_WALLPAPER	Allows applications to set the wallpaper.
	SET_WALLPAPER_HINTS	Allows applications to set the wallpaper hints.
	TRANSMIT_IR	Allows using the device's IR transmitter, if available.
	UPDATE_PACKAGES_WITHOUT_USER_ACTION	Allows an application to indicate via <code>PackageInstaller.SessionParams.setRequireUserAction(int)</code> that user action should not be required for an app update.
	USE_BIOMETRIC	Allows an app to use device supported biometric modalities.
	USE_EXACT_ALARM	Allows apps to use exact alarms just like with <code>SCHEDULE_EXACT_ALARM</code> but without needing to request this permission from the user.
	USE_FINGERPRINT	This constant was deprecated in API level 28. Applications should request <code>USE_BIOMETRIC</code> instead
	USE_FULL_SCREEN_INTENT	Required for apps targeting <code>Build.VERSION_CODES.Q</code> that want to use notification full screen intents.
	VIBRATE	Allows access to the vibrator.
	WAKE_LOCK	Allows using <code>PowerManager WakeLocks</code> to keep processor from sleeping or screen from dimming.
	WRITE_SYNC_SETTINGS	Allows applications to write the sync settings.
Deprecated	BIND_CARRIER_MESSAGING_SERVICE	This constant was deprecated in API level 23. Use <code>BIND_CARRIER_SERVICES</code> instead
	GET_TASKS	This constant was deprecated in API level 21. No longer enforced.

	PERSISTENT_ACTIVITY	This constant was deprecated in API level 15. This functionality will be removed in the future; please do not use. Allow an application to make its activities persistent.
	RESTART_PACKAGES	This constant was deprecated in API level 15. The <code>ActivityManager.restartPackage(String)</code> API is no longer supported.
	SET_PREFERRED_APPLICATIONS	This constant was deprecated in API level 15. No longer useful, see <code>PackageManager.addPackageToPreferred(String)</code> for details.
	SMS_FINANCIAL_TRANSACTIONS	This constant was deprecated in API level 31. The API that used this permission is no longer functional.

RuStore' Requirements to app permissions are comparable to Google Play Guidelines and are based on Google terms for developer's convenience.

For up-to-date permission categories refer to:

<https://developer.android.com/reference/android/Manifest.permission>

Publish your app

Publish your app to make it available to users on the RuStore.

1. Open the [RuStore Console](#).
2. Select the "Applications" tab.
3. Click "Add an app".
4. Enter the app name in the pop-up window.
5. Click "Add".
6. Select the added app.
7. Click "Submit for moderation".

Go to apps

screen_collect

- Information
- Versions
- Reviews
- RuStore Get **Beta**
- Monetization **Beta**
 - Subscriptions
 - In-App purchases
 - Payments management
 - Payment statistics
- Push notifications **Beta**
 - Projects
 - Push notification statistics

Versions > Uploading a new version Clear All

Version

Upload APK

2.0.apk 3.23 MB ✓ Uploaded

What's new 46/300

- Some bugs has been fixed
- New icon
- New in-app purchases

Comments for the moderator 36/136

Test credits are same with v.1

To avoid possible update errors, sign your apps with the same signatures as in other stores (for example, Google Play) before uploading them.

Let the users know why they need to update your app.

If the full functionality of your app is only available to registered users, please provide your login details. As well as any other information that will help the moderator check the app.

App Information

Name for the user 15/50

New_application

This is what the app name will look like in the window. You can always change it.

Type

Non-game app

Game

Cost, P 3/6

678

RuStore commission is 15%. For more information about paid apps, see the

Categories

Main

Ads and Services

To learn how to choose a category for an app, see the developer's guide.

Additional information

Education

Age category

3+

Rules for choosing an age category can be found in the developer's guide.

Description

Short description 22/30

Short text description

Description 24/4000

Enjoy our second version

Media files for the folder

Icon for the app catalog

2x5004e0e409-4a74-0ff5-e570c44b4b87.png ✓ Uploaded

Screen shot orientation

Landscape

Portrait

Choose whether the screenshots are oriented vertically or horizontally.

Screenshots of the app

6540173d4e5f0-4c24-8950-88721a465d3.png ✓ Uploaded

Upload up to 10 images with a maximum size of 1 MB (JPG, PNG format) (9-14, 185x120px - 2740x3940px) [Select an image](#)

Publishing version

Manually

Automatically after moderation

At the selected date and time

Date 15.03.2023 Time (GMT+3) 15:00

Deployment Parameters

Select the percentage of users that you want to deploy the updates to.

Percentage of users

100%

[Publish for moderation](#) [Cancel](#)

[Promote the app](#) [Learn more from advertising](#)

User agreement Privacy Policy Distribution Agreement Write to support

© Kowcools.ru Privacy

Fill in the required information fields

Upload an APK file to your app:

- An APK file should not exceed 2.5 GB;
- Make sure your file has an.apk extension;
- Generate a signed APK file;
- Use a unique package name;
- The app package must be checked and adjusted.

When downloading the second and subsequent versions:

- An APK file should not exceed 2.5 GB;
- Make sure your file has an.apk extension;
- Generate a signed APK file;
- The signature must match the previous one;
- The app package must match the previous one;
- The app package must be checked and adjusted;
- The version code must be greater than the previous one.

When submitting an APK file to the [RuStore Console](#), ensure that it carries the same signature as used in other app stores, such as Google Play. This will allow users to update apps installed on their devices whenever a newer version is available on the RuStore.

If you decide to use a new signing key, you will need to replace it in all stores as well, including Google Play. [How do I update my Google Play app signing?](#)

In case you need to replace the signing key on the RuStore, please reach out to support@rustore.ru

Note that if the signing keys do not match, users will be unable to update an app that was initially installed via a different store.

Fill in the required app information fields

1. Add the name of your app:
 - The app name should not exceed 30 characters;
 - The app name must be unique.
2. Choose your app category from the proposed list.
3. Specify an age rating according to the suggested guideline
4. Add a short description of your application (up to 40 characters).
5. Once there, add a detailed description of your application (up to 4000 characters. Always keep in mind the user's ability to minimize the description window for up to 2000 characters).

Upload app icon

- The image must not exceed 3 MB;
- Accepted image formats: .png or .jpg;
- The image resolution must be 512 x 512 px.

Upload app screenshots

You can add screenshots for mobile phones only. In this case, the same screenshots will be displayed on both device types: phones and tablets.

Besides, you can upload additional screenshots for tablets. Then phone screenshots will be displayed on devices with screens smaller than 7 inches, and tablet screenshots will appear in the application page on devices with 7-inch screens or larger.

Please make sure that you have uploaded screenshots for the mobile app version.

How to add screenshots

- Select vertical or horizontal orientation.
- Specify the device type: phone or tablet.
- Upload 1 to 10 screenshots.

Image Requirements

- The image size should not exceed 3 MB for phones and 5 MB for tablets;
- The image should not exceed 3 MB;
- Accepted image formats: .png or .jpg;
- Maximum resolution — 2160x3840 px;
- Recommended aspect ratio: 16:9;
- If the width or height is less than the recommended one, the screenshot will be stretched to 16:9;
- If the width or height is greater than the recommended one, the screenshot will be cropped to 16:9.

Add a link to a video

You can also add a link to a video hosted on VK Video. The video will be available in the application page for phones and tablets.

Video Requirements:


- it must be directly related to the application (for example, you can upload a trailer, gameplay recording, etc.);
- it must not contain inappropriate content (see Application Requirements);
- the video must be public;
- the page on which the video is posted must be open to all users. If you add a video from a page with limited privacy settings, then the video must be moved to a group or public page with open access.

The video duration cannot exceed 1 minute;
You can only add one video.

Link to VK Video (optional)

The video must be publicly accessible, horizontal orientation only

Video preview (optional)

 **Select files** or drag them here

Maximum image size and resolution: 3 MB 3840×2160px
Aspect ratio: 16:9, other sizes will be cropped
Horizontal orientation by default

Submit for moderation

Cancel

Enter a description and click "Submit for moderation".

Before submitting an application for moderation, make sure you checked the application for compliance with the App Review Guidelines.

To add a developer responsible for the app, the following conditions must be met:

- The developer should be logged in to the [RuStore Console](#);
- The developer should send the VK ID to the employee who uploaded the app (in digital format).

Uploading App Bundles

Uploading applications in AAB format is now available via the RuStore Console. This format streamlines the process, resulting in faster installations due to reduced file sizes for users downloading from RuStore.

How to upload App Bundles

1. Open RuStore Console.
2. Go to the **Applications** tab.
3. Click **Add an App**.
4. In the window that opens, enter the app name.
5. Click **Add**.
6. Select the added application.
7. Click **Upload app version**.
8. Upload your app signing keys in .aab format
9. Upload the application's AAB file.

Версия 1
Очистить все

Загрузка новой версии

Файлы

Вы можете загрузить несколько файлов, например:

- С разными подполками, чтобы избежать ошибок обновления у пользователей. [Подробнее](#)
- С флажком Mobile Beta для стабильной работы приложения на устройствах Nexus и Pixel на подэкранной версии Google.

Используется для загрузки версии приложения.
APK не более 2,5 GB или AAB не более 500 MB

[Выберите файл](#)

Поднять на загрузку

Для Android App Bundle требуется подпись. Если у вас нет KEK, загрузите версию в формате APK. [Подробнее](#)

[Загрузить](#)

Комментарий для модератора

Комментарий будет виден только модератору

Если полная функциональность вашего приложения доступна только зарегистрированным пользователям, укажите данные для входа в службу защиты. А также любую другую информацию, которая поможет модератору проверить приложение.

Безопасность данных пользователя

Категории и типы данных

Эти сведения помогут пользователям понять, как вы используете их данные и обеспечат безопасность. [Подробнее читайте в справочнике разработчика](#)

Типы данных, которые собирает ваше приложение

Информация о приложении

Название для пользователя

Так будет выглядеть название приложения на экране. Вы всегда можете его изменить.

Тип

Неигровое приложение

Игра

Категории

Основная

Как выбрать категорию для приложения читайте в [справочнике разработчика](#)

Дополнительная

Возрастная категория

Правила выбора возрастной категории читайте в [справочнике разработчика](#)

Описание

Краткое

Основное

Медиафайлы для каталога

Ресурсы для каталога в приложении

Изображение до 1 Mb, разрешение до 512x512

Ориентация изображения

Вертикальная

Горизонтальная

Скриншоты приложения для телефона

Изображение до 3 Mb, разрешение до 2160x1080px
Сопоставимый экран: 3:2, другие размеры будут обрезаны
Рекомендуем JPG, 100% (x2) DPI

[+ Добавить скриншоты для планшета](#)

Ссылка на видео с YouTube (необязательно)

Видео должно быть с публичным доступом, только горизонтальная ориентация

Обложка для видео (необязательно)

Изображение до 3 Mb, разрешение до 3840x2160px
Сопоставимый экран: 3:2, другие размеры будут обрезаны
Прямоугольная ориентация, no-embed

Публикация версии

Вручную

Автоматически
Включает одобренных модераторов

[Отправить на модерацию](#)
[Создать APK-файл](#)
[Отмена](#)

Requirements for uploading App Bundles:

App signing keys must be added separately before uploading the .aab file.

- The application file should not exceed 500 MB.
- Ensure that the package name is unique.
- Thoroughly test and configure the build.


Uploading app signing keys

For Android App Bundle to work, you need to upload the app signing key:



1. Go to the warning “Signature not uploaded” and click “Uploaded”.


Загрузка новой версии


Файлы

 Вы можете загрузить несколько файлов, например:

1. С разными подписями, чтобы исключить ошибки обновления у пользователей. [Подробнее](#)
2. С Huawei Mobile Services для стабильной работы приложения на устройствах Huawei и Honor, не поддерживающих сервисы Google

 1.0(9999)
2 GB 

 Выберите файлы или перетащите их сюда
APK не более 2,5 GB или AAB не более 5 GB

 Подпись не загружена

Для Android App Bundle требуется подпись, используемая в предыдущей версии. Если у вас её нет, загрузите версию в формате APK. [Подробнее](#)

[Загрузить](#)

2. In the new window, select **Upload** to acquire the PEPK tool.


3. Click **Copy** to duplicate the command along with your distinctive encryption key.

4. Customize the command by providing your data instead of the default values.
5. Run the tool to export and encrypt the private key using the modified command within a terminal. Substitute the parameters and input the vault and key passwords when prompted.

```
java -jar pepk.jar --keystore=your_key_storage.keystore  
--alias=имя_ключа --
```

6. Upload the ZIP archive created using the PEPK tool.
When migrating from .apk to .aab format, the signing certificate must match the signature fingerprint of the previous version.
7. Upload the upload key certificate that signed your AAB in .PEM.

Загрузка подписи приложения

 После публикации приложения в формате AAB, следующие версии должны содержать файл AAB. Подробнее о работе с подписями читайте [в справочнике разработчика](#)

1 Скачайте инструмент PERK

 Скачать

2 Запустите инструмент с помощью команды

С помощью указанной ниже команды запустите инструмент, который экспортирует и зашифрует закрытый ключ. Замените аргументы и, когда потребуется, введите пароли хранилища и ключа

```
java -jar perk.jar --  
keystore=ваше_хранилище_ключей.keystore --  
alias=имя_ключа --  
output=новый_путь_для_созданного_сертификата --...
```

Скопировать

3 Загрузите созданный ZIP-архив

Файл формата ZIP не более 100 КБ

Выберите файл

4 Загрузите сертификат загрузки

Файл формата PEM не более 100 КБ

Выберите файл

Отмена

Отправить подпись

Creating a signing key for app bundle

To generate a private key (sign key) for your Android App Bundle, which will be used to sign the .apk files distributed to users, follow these steps:

1. Create a new key pair (alias=sign) by executing the following command in the terminal:

```
keytool -keystore .keystore -genkey -alias sign_v1 -keyalg  
RSA -validity 36500
```

where sign is the name of the app signing key.

2. Copy the PEPK command from the second step of the signing key modal.
3. Obtain the PEPK utility from the first step of the modal and transfer it to the system folder of your PC.
4. Run the tool that exports and encrypts the private key using the modified command in a terminal. Subsequently, replace the arguments and input the vault and key passwords when prompted:

```
java -jar pepk.jar --keystore your_key_storage.keystore  
--alias key_name --output  
new_path_for_created_certificate/pepk_out.zip  
--encryptionkey=your_unique_encryption_key --include-cert
```

5. Generate the upload key that signs your .aab file using the command:

```
keytool -keystore .keystore -genkey -alias upload -keyalg RSA
```

6. Obtain the boot certificate from the previously generated boot key using the command below:

```
keytool -exportcert -alias upload -keystore .keystore -rfc  
-file uploadcert.pem
```

8. Upload the ZIP archive created using the PEPK tool and the download certificate obtained from steps 4 and 6 into modal options 3 and 4 respectively.

9. Go to the **App Signing Key** page and verify that the signature has been successfully uploaded and the appropriate information is displayed for it.

How to upload a new app version

Once published, you can upload new versions for your app.

1. Open the [RuStore Console](#).
2. Select the "Applications" tab.
3. Choose an application.
4. Click "Upload a new version".
5. Specify the app parameters.

Please make sure that you have uploaded screenshots for the mobile app version.

6. Select the app release options.
7. Click "Submit for moderation".

For paid apps, enter the price of your app. Developers who sell paid apps are subject to a 15% service fee.

When submitting an APK file to the [RuStore Console](#), ensure that it carries the same signature as used in other app stores, such as Google Play. This will allow users to update apps installed on their devices whenever a newer version is available on the RuStore.

If you decide to use a new signing key, you will need to replace it in all stores as well, including Google Play. [How do I update my Google Play app signing?](#)

In case you need to replace the signing key on the RuStore, please reach out to support@rustore.ru

Note that if the signing keys do not match, users will be unable to update an app that was initially installed via a different store.

Before submitting your app for moderation, be sure to check the application for compliance with the App Review Guidelines.

If you change the price when publishing a new app version, describe changes in the new app version in the "Description" section.

The application will be sent for moderation.

[Go to apps](#)

screen_collect

[Information](#)[Versions](#)[Reviews](#)[Native Dev](#) [Beta](#)[Monetization](#) [Beta](#)[Subscriptions](#)[In-app purchases](#)[Payments management](#)[Payment statistics](#)[Push notifications](#) [Beta](#)[Projects](#)[Push notification statistics](#)

Versions > 1.0(1)

[Clear All](#)

Version

Upload APK



To avoid possible update errors, sign your apps with the same signed APK as in other stores (for example, Google Play) before uploading them.

What's new 10/500

First version

Let the users know why they need to update your app.

Comment for the moderator 47/180

Credits for test
Login: example@example.com
password: qwerty

If the full functionality of your app is only available to registered users, please provide your login details. As well as any other information that will help the moderator check the app.

App information

Name for the user 10/50

New_application

This is what the app name will look like in the window. You can always change it.

Type

 Non-game app Game

Cost, ₽ 3/4

570

Users' commission is 15%. For more information about paid apps, see the

Categories

Main

Ads and Services

To learn how to choose a category for an app, see the [developer's guide](#).

Additional information

Education

Age category

3+

Rules for choosing an age category can be found in the [developer's guide](#).

Description

Short description 25/500

Short test description

Description 251/8000

Enjoy our first version

Media files for the folder

Icon for the app catalog



Screenshots orientation

 Landscape Portrait

Choose whether the screenshots are oriented vertically or horizontally.

Screenshots of the app



Upload up to 18 images with a maximum size of 1 MB
PNG, PSD format (716, 180x375px - 2140x3940px)

[Select an image](#)

Publishing version

 Manually Automatically after moderation At the selected date and time

Date 15.03.2023

Time (GMT+3) 14:30

Deployment Parameters

Select the percentage of users that you want to deploy the update to

Percentage of users

100%

[Promote the app](#)[Earn money from advertising](#)[Submit for moderation](#)[Cancel](#)

Adding an application to your TV device

When publishing an app, you must specify the type of device on which the application will be used.

1. Open RuStore Console.
2. Go to the "Applications" tab and click "Add an app".
3. Select the app type in the pop-up window: "Universal" or "Only for TV".

Select the "Universal" type if your app's APK is suitable for phone, tablet and TV.

Note. Only free applications are available for TV: if you download an apk file for TV, creating a paid universal application is not possible.

4. Enter the name of your app and click "Add".

****Restrictions:****

1. If different versions are uploaded, the `package_name` for TV must be different from the `package_name` of the mobile app.
2. APK apps can be downloaded for mobile and TV versions at the same time with the same `package_name` for the same version.

To download a new version of the application, in the “Versions” tab, click “Download Version” and fill in the following information:

****Download the APK file of the application and leave a comment for the moderator if necessary:****

- APK file size: no more than 2.5 GB;
- Valid format: ``.apk``;
- The apk signature must match the signature used in other app stores (for example, Google Play).

This will allow users to update applications installed on the device, for which a newer version is available in RuStore.

You can specify a new signing key, but then it needs to be replaced in all stores. [How to update your signature on Google Play?](https://support.google.com/googleplay/android-developer/answer/9842756?visit_id=638150905149654896-2147605919&rd=1#upgrade)

- The package name must be unique.

****Fill in the application information:****

1. Enter the name of the application:

- no more than 50 characters;
- the application name must be unique.

2. Select a category from the list provided.
3. Select an age limit from the list provided.
4. Enter a brief description of the application (up to 80 characters).
5. Enter a detailed description of the application (up to 4000 characters).

****Upload the app icon:****

- Image size: no more than 3 MB;
- Valid format: `.png` or `.jpg`;
- Image size: 512x512.<

****Upload at least 2 screenshots of the application:****

- Format: JPEG or PNG (24-bit without alpha channel);
- Recommended size: 1920 x 1080 pixels (16:9 ratio);
- Recommended volume: no more than 3 MB;
- Resolution: 320p to 4k.

The TV banner will be the ****first**** screenshot when publishing the application on TV.

****Configure application publication: “Automatically after moderation” or “Manually”.****

For more information about setting up app publishing, see [Automatically and manually publishing an app](#)

1. Click "Submit for moderation".

The app will be sent for moderation. After the moderation, the status of your application will be changed and the "Publish" button will appear.

2. Click Publish and your app will be available on TV.

ASO Recommendations

One of the main factors influencing the development of RuStore is the availability of applications. To catch the user's eye, it is important to describe the app in detail and wrap it in a compelling design. This will help to increase the number of installs and level up the app's positions in searches.

RuStore operates with several parameters to generate app search results. Here are some of them that we would like to draw attention to:

Rating. The higher the rating, the higher the position of the app in the list. It is important that the developer actively responds to user feedback. RuStore performs automatic and manual rating checks to make sure there is no tipping. You can increase the number of ratings by requesting them from users using [RuStore Reviews&Ratings SDK](/sdk/reviews-ratings).

Number of installations. The more there are, the more likely the app is to get to the top. However, you shouldn't use cheating, organic traffic is the only thing that matters here. To promote the application, the developer can place a download button on their resources or use advertising. This will help to attract an audience and improve positions.

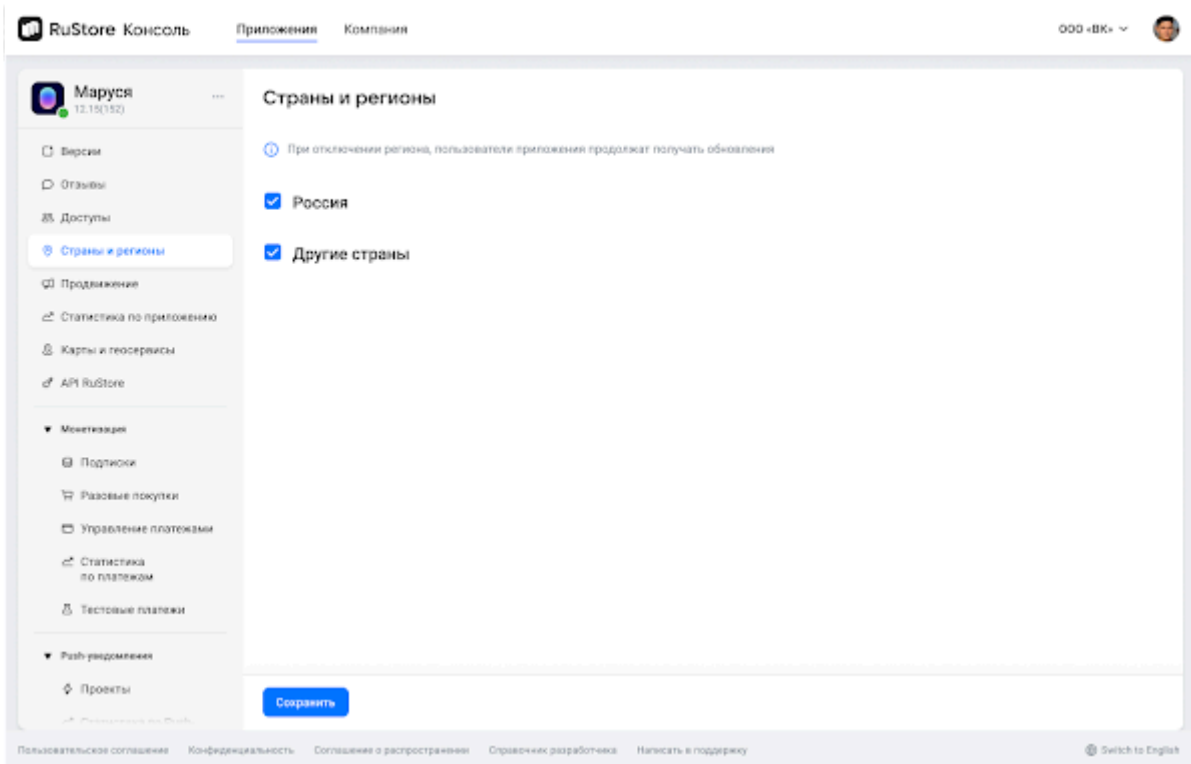
Number of conversions to installation. RuStore recommends that you use compelling design from the get-go to make sure conversion rate is high. It is vital to design screenshots properly, add a sufficient number of them to the description and update the "What's new?" section. This will help to increase the number of app downloads.

Number of ratings and their distribution. The more positive ratings, the better! RuStore recommends to respond to its users in time, refine the application constantly, boost its functionality, encourage users to rate and write comments. This will also improve the app's rendering performance. It is important to remember that RuStore monitors and suppresses the use of spoofing. [RuStore Reviews&Ratings SDK](/sdk/reviews-ratings) will simplify the work with reviews and natively encourage users to give their feedback right in your app.

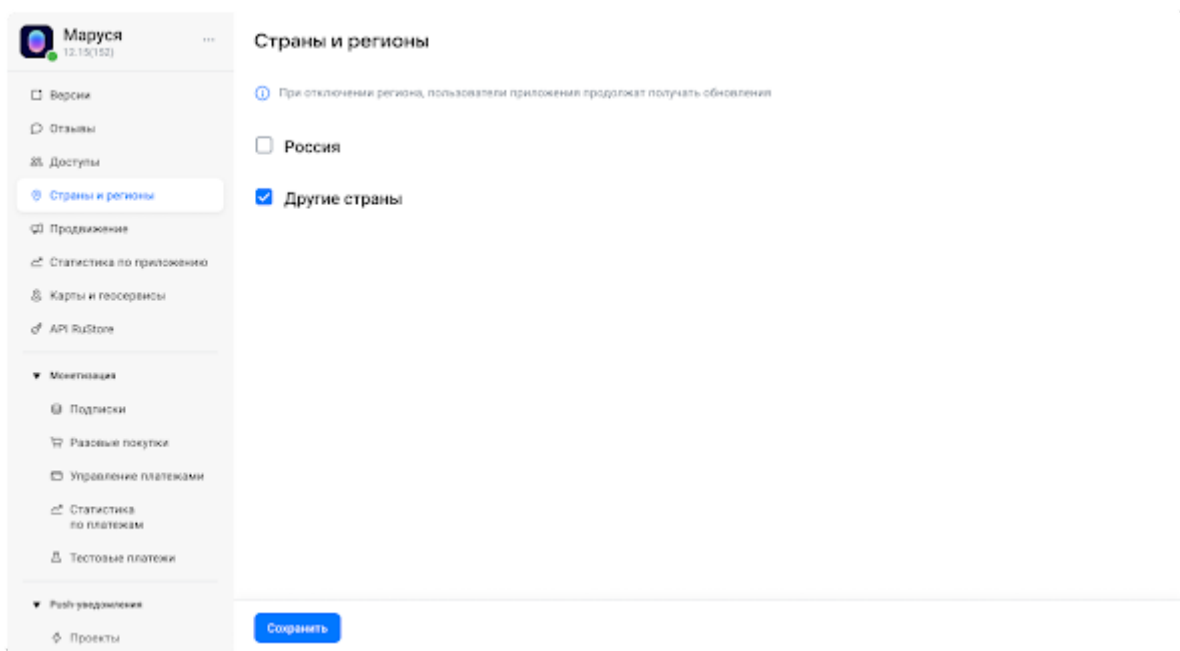
Manage availability for your app

Set up an available area where users can install your app on their devices. RuStore is now accessible in Russia and various other countries. At that, users can update previously downloaded apps regardless of their accessibility preferences.

1. Open [RuStore Console](#).
2. Go to Applications.
3. Select an app.
4. Click Countries and Regions from the left side menu.



5. Check the boxes next to areas where you want your app to be accessible. Uncheck the boxes to prevent your app from being visible in the chosen area.



Customize your app release options

Auto & manual app release

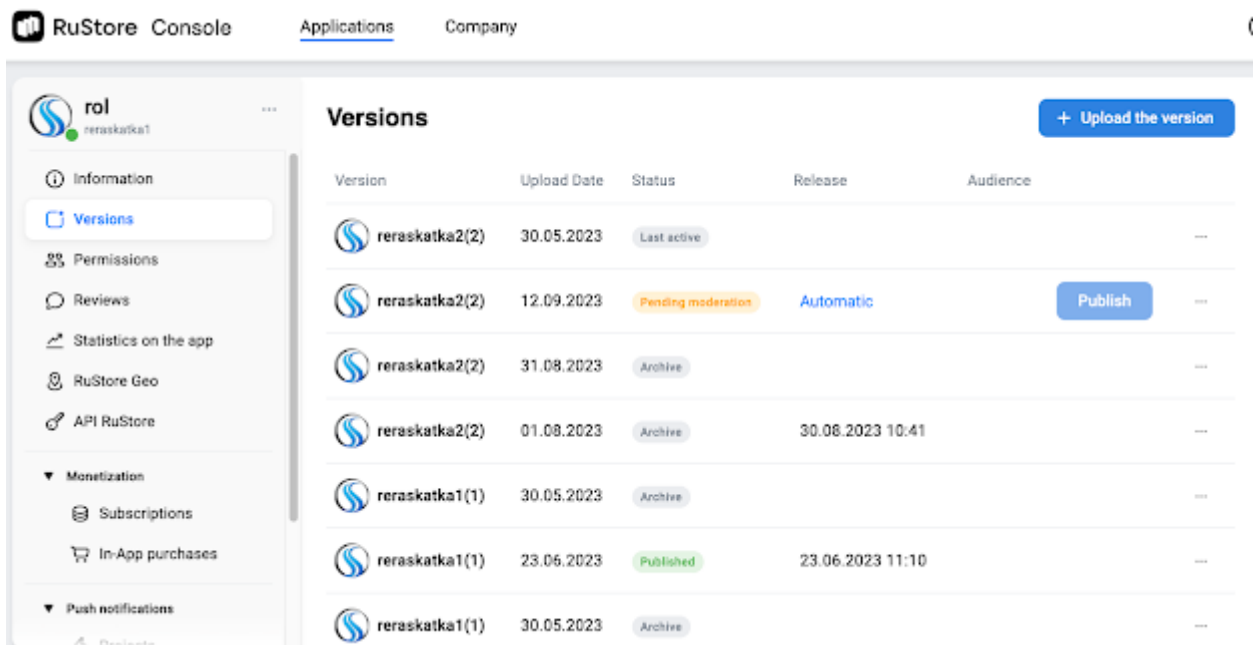
You can set up auto/instant app release while it undergoes moderation.

Instant app release is available to you when you first publish an app version and when you upgrade a previously uploaded version.

Auto/instant release

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an app with the "Waiting for moderation" status.
4. Click "Manage app release".
5. Select "Publish automatically after moderation".

Once moderation is completed, the app will be published on the RuStore within one hour. You will be notified about your app status by email.



You can change the release status from auto to manual as long as your app undergoes moderation.

Manual release




1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an app with the "Waiting for moderation" status.
4. Click "Manage app release".
5. Select "Manually".

After the moderation, the app status will change to "Ready for publication".

6. Click "Manage app release".
7. Select "Now".
8. Click "Apply".

Versions

[+ Upload the version](#)

Version	Upload Date	Status	Release	Audience
 ver2_test(2)	08.12.2022	Ready for publication	Manual	Publish ...
 ver1_test(1)	05.03.2023	Ready for publication	Manual	Publish ...
 ver1_test(1)	08.12.2022	Last active		...

You can also stop the current version. Click **Unpublish** to deactivate the published app version. The last active version will then be activated.

At that the users will continue to use your app if they installed a new app version before deactivation.

Delayed app release

You can delay your app release to a specific date and time.

The delayed release is only available when upgrading a previously uploaded app version.

Manage app release when uploading a new app version

1. Open the [RuStore Console](#).
2. Select the "Applications" tab.
3. Choose an application.
4. Upload a new app version.
5. Then go to the "Publish an app version" block.
6. Select "At the selected date and time".
7. Set the release date and time.
8. Next, click "Submit for moderation".

Your app will be sent for moderation.

Manage app release during moderation

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an app with the "Waiting for moderation" status.
4. Click "Manage app release".
5. Select "At the selected date and time".
6. Once there, select the release date and time.

Once moderation is completed, the app will be published on the RuStore at the selected date and time. You will be notified about your app status by email.

Manage app release after moderation

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an app with the "Ready for release" status.
4. Click "Edit description" and scroll down to the "Publishing version" section.
5. Select "At the selected date and time".
6. Once there, click "Submit for review".

Publishing version

- Manually
- Автоматически
Once approved by the moderator
- At the selected date and time
Audience settings will not be available

Date

29.09.2023



Time (GMT+3)

09:00



Submit for moderation

Cancel

Once moderation is completed, the app will be published on the RuStore at the selected date and time. You will be notified about your app status by email.

You can change your delayed release parameters if:

- your application undergoes moderation;
- your application has passed moderation, but the release date has not yet come.

Staged app release

You can publish your app in stages to make it available for a certain group of users. This will help you quickly track down bugs, issues, or negative reviews for a specific app version.

Staged release is only available when upgrading a previously uploaded version.

The following percentage values are available for a group of users:

- 5%
- 10%
- 25%
- 50%
- 75%
- 100%

You can also increase the percentage value. Staged release completion stands for 100% of users.

Manage app release when uploading a new version

1. Open the [RuStore Console](#).
2. Select the "Applications" tab.
3. Choose an application.
4. Upload a new app version. [How to do it?](#)
5. Then go to the "Publish an app version" block.
6. Specify the percentage of users.

You can select the percentage of users if you choose to publish your app automatically or either manually.

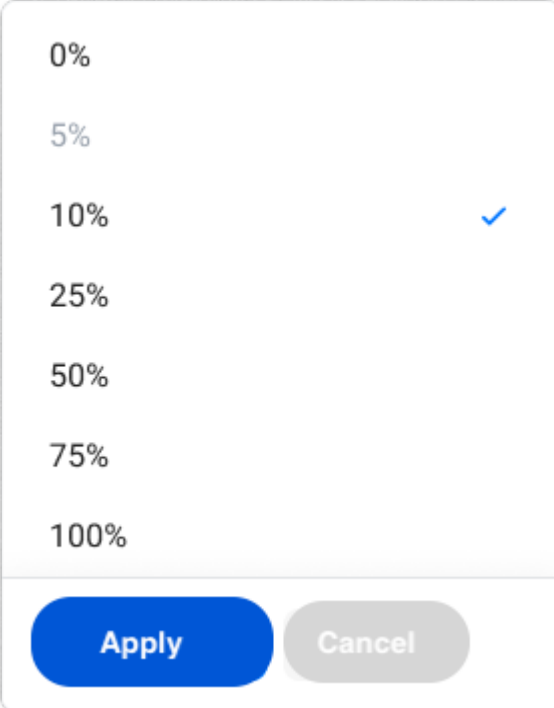
7. Click "Submit for moderation".

Your application will be sent for moderation.

Manage app release during moderation

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an app with the "Waiting for moderation" status.
4. Click "Manage app release".
5. Specify the percentage of users.
6. Once there, click "Apply".

If you change auto or manual release to delayed one, the percentage of users will automatically change to 100%.



0%

5%

10% ✓

25%

50%

75%

100%

Apply Cancel

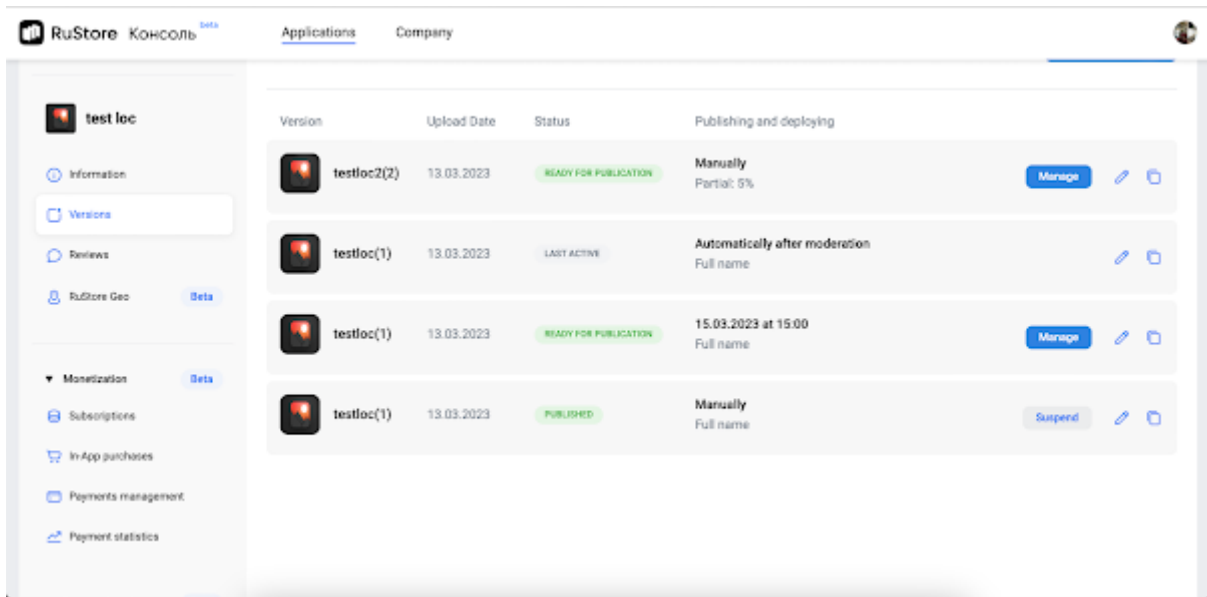
Manage app release after moderation

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an app with the "Ready for release" status.
4. Click "Manage app release".
5. Select "Now".
6. Specify the percentage of users.

You can only increase the percentage of users, decrease is not available.

7. Once there, click "Apply".

The app version will be published, and app version update will be available for randomly selected users.



You can also stop the staged release. Click "Manage app release" and select "Stop". The updated app version will be deactivated, the last active version will then be activated.

At that, the users will continue to use your app if they installed a new app version before deactivation.

How to manage APK signing keys

How to sign an Android app

RuStore supports one application format which is an APK file.

Each APK file must be signed with a digital certificate, which Android uses to identify the app owner. Please ensure safe storage of the signing key.

Check Android versions

Android compares digital fingerprints of each signed .APK file.

A digital fingerprint is a sequence of bytes created by applying a cryptographic hash function to a public key.

The digital fingerprint is represented as follows:

43:51:43:a1:b5:fc:8b:b7:0a:3a:a9:b1:0f:66:73:a8

What causes update errors?

One of the most common mistakes publishers make is using different signatures for an app published on the RuStore and other stores.

For example, the initially downloaded app version is signed with one certificate and the next version is signed by another one. Due to these differences, Android does not allow you to install updates for this application.

What are possible reasons:

1. The developer loses a certificate and then generates a new one to publish the application in the store.
2. Developers can publish the same application in different stores. For example, initially the developer published applications on Google Play and used one certificate. But after switching to RuStore, he started using a different certificate.

This results in dividing users into two categories:

- those who installed an app from Google Play,
- those who downloaded it from RuStore.

If a user has installed RuStore and wants to update an app that was previously downloaded from Google Play, he will not be able to do this due to different certificates.

How can this problem be solved?

1. We recommend using one certificate for all app versions to avoid problems with version updates from different sources.
2. In case of urgency, you can ask users to remove the "old" app versions that cannot be updated, and ask them to download new ones. But this method is associated with the risk of losing part of the audience.
3. You can also update the app signature with the help of RuStore technical support. The steps are listed below.

RuStore recommends using a locally stored certificate for more control over app releases. If you use Google Play App Signing, which allows Google Play to generate and store the signature on its own, you may find that you cannot use the certificate outside of Google Play.

Resolving Update Errors with RuStore Support

To update the signing key, send an email to support@rustore.ru

Specialists will initiate an appeal and check that the application belongs to the applicant. Upon successful identification, the support specialist will deactivate the old certificate. After that, the developer just needs to upload the new APK file with an updated signing certificate common to RuStore and other stores.

Paid apps

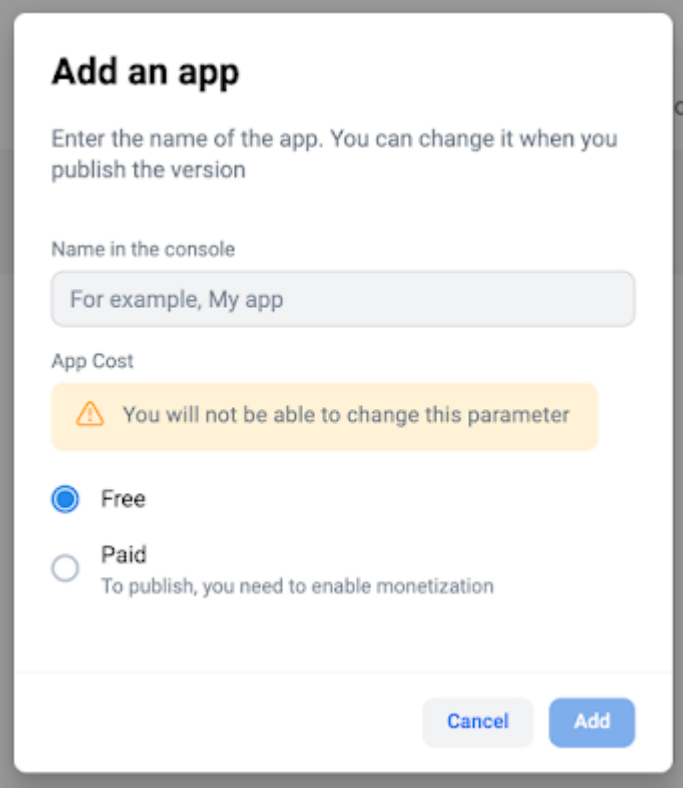
You can set your app as free or paid when adding a new app to the RuStore Console,

Enable monetization to upload paid apps.

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Click "Add an app".
4. Enter the name of your app in the pop-up window.
5. Select the app type: "Free" or "Paid".
6. Click "Add".

You cannot change a free app to a paid app on the RuStore.

Once you selected the "Free" app type, it cannot be changed to "Paid". If you want to charge for such an app, you need to re-upload it with a new name.




Add an app

Enter the name of the app. You can change it when you publish the version

Name in the console

For example, My app

App Cost

 You will not be able to change this parameter

Free

Paid
To publish, you need to enable monetization

Cancel Add

You can also update your app prices when editing or when uploading a new app version.

How to change the app price

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.

3. Select an application.
4. Click "Edit".
5. Enter the new app price.

RuStore service fee for paid apps is 15%. In case RuStore acts as the developer's tax agent, apart from service fee, taxes will be withheld in accordance with the legislation of the Russian Federation.

6. Click "Submit for moderation".

Before submitting your app for moderation, be sure to check the application for compliance with the [App Review Guidelines](#).

A new app version will be created and sent for moderation. After the moderation, the status of your application will be changed and the "Publish" button will appear.

Click "Publish" to make your app available to users. The new app price will be displayed on the RuStore.



← Go to apps

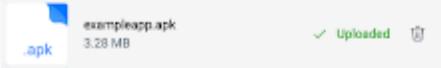
example_app

- Information
- Versions
- Reviews
- RuStore Geo [Beta](#)
- Monetization [Beta](#)
 - Subscriptions
 - In-App purchases
 - Payments management
 - Payment statistics
- Push notifications [Beta](#)
 - Projects
 - Push notification statistics

Versions > Uploading a new version Clear All

Version

Upload APK

 ✓ Uploaded

Comment for the moderator 11/180

example_app

To avoid possible update errors, sign your apps with the same signature as in other stores (for example, Google Play) before uploading them.

If the full functionality of your app is only available to registered users, please provide your login details. As well as any other information that will help the moderator check the app.

App Information

Name for the user 11/50

example_app

This is what the app name will look like in the window. You can always change it.

Type

Non game app

Game

Cost, ₽ 3/6

150

RuStore commission is 15%. For more information about paid apps, see the

Categories

Main

Adventure

To learn how to choose a category for an app, see [the developer's guide](#)

Additional information

Arcade

Age category

16+

Rules for choosing an age category can be found in [the developer's guide](#)

Description

Short description 11/80

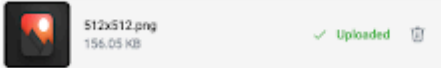
example_app

Description 11/4000

example_app

Media files for the folder

Icon for the app catalog

 ✓ Uploaded

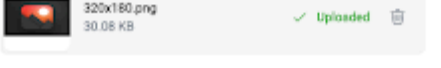
Screenshot orientation

Landscape

Portrait

Choose whether the screenshots are oriented vertically or horizontally.

Screenshots of the app

 ✓ Uploaded

Upload up to 10 images with a maximum size of 1 MB
JPG, PNG format (1:6-9, 320x180px - 3840x2160px) [Select an image](#)

[Promote the app](#)

[Earn money from advertising](#)

[Submit for moderation](#) [Cancel](#)

To set up in-app product pricing, see how to create paid in-app items.

To set up subscription pricing, see how to create an app subscription

App Review Guidelines

Before your app is published on the RuStore, it needs to pass mandatory review. Throughout this process you'll go through the entire user journey to make sure that your app works properly, corresponds to the declared category and age rating without neither posing any security risks nor threatening the laws and rights of other users. Non-compliance with the specified requirements may result in the application being declined for publication or, if violations are identified after review, the app may be removed from the store.

Follow a few steps below to double-check your app before publishing it on the RuStore:

1. Your app performance meets the following requirements:

- It runs and works steadily without failures and errors;
- All declared app features work correctly;
- We do not allow publishing apps that cause crashes, force close, freeze or work with errors;
- Your app must be self-sustaining and have original content. We discourage applications that are primarily designed to redirect to a website (WebView). An application may duplicate a website functionality or use a WebView to perform certain options, but it must be represented as a complete, standalone product. We reserve the right to block your app if we suspect that it is published solely to drive traffic to a website;
- Your app should not contain abundant advertising that will interfere with its usage (ads after each user action), or either with no options to remove ads from the screen;
- Each user should be able to successfully pass through authorization whenever so required without redirecting to third-party resources;
- For limited-audience apps only: specify test account data in a comment and send it for review;
- Do not forget to provide test authorization data to RuStore moderators in the "Comments for moderators" field to test the app functionality. Make sure that the demo mode shows all the app features.
- Make sure that your app is adapted to most modern mobile devices - no errors and incorrect layout of applications are allowed.

2. Your app content is designed as follows:

The application, its page and user content must be completely free of the following:

- any ethically inappropriate statements or ideas: overt discrimination, rabble-rousing, insults (except of playful jokes), etc.;
- offers for sale or distribution of illegal goods (prescription drugs, narcotic substances, weapons, tobacco and nicotine products to-be-delivered, etc.) and services (prostitution, forgery, hacking services, cheating, etc.) .);
- prohibited pornographic content (obscene images of minors, bestiality, dolcett, etc.), including drawings;
- shocking content and realistic images of people or animals being killed, maimed, tortured or tormented, except for those related to the relevant genre and age category (medical content)

- suicidal content in any form and in any context, including images or romanticization of self-harm;
- trademarks and third-party intellectual property, including misleading logos and names; Please note: if your application is an official partner of another service, please provide us with sufficient documents confirming this status. RuStore reserves the right to selectively check documents related to intellectual property rights.
- links to third party app stores or storefronts, phishing and irrelevant links.

Note. It is forbidden to publish apps related to holding and conduct of gambling, accepting bets and other games that involve real money.

If an app implies posting user-generated content, the developer ensures to provide timely and adequate pre-view of this content or post-moderation based on user complaints.

Applications with user-generated content or social networks must include:

- definition and criteria for inappropriate content;
- offensive content reporting options;
- access restrictions for those who fail to comply with the rules;
- support or app review team contact details.

Application reviewers must promptly respond to user problems and requests from RuStore technical support.

If an app includes user-generated content from an adult website, then it should be hidden by default and be only visible to users who have enabled it on the site.

3. Application security and user rights protection:

- Your app must not contain malicious or phishing links;
- If your app implies working with personal data, make sure that you are officially recognized as a personal data operator. You must also inform the user about the collection and use of their personal data and obtain their consent for data processing;
- If your app features in-purchase options, provide the user with detailed information about the payment process, as well as about products delivery and return. Please follow the Consumer Protection Act.

4. Privacy and device data:

While running, your app may access personal and sensitive user data stored on a device. Before you start, we recommend that you read the Declare app permissions section. RuStore accepts the following permission levels:

- prohibited;
- sensitive;
- safe;
- obsolete.

If prohibited permissions (protection level “not for use by third party applications”) are discovered in the app when uploading it to the RuStore Console, this version will be rejected. In this case the APK file needs to be finalized, excluding prohibited permissions, and re-uploaded. If sensitive permissions are found when uploading the application, the developer should provide sufficient reasons for each permission in the Sensitive Permissions section.

At this point, make sure that:

- data obtained with a user’s consent is not intended for sale, as well as for distribution and further sale;
- permissions are not granted to bypass device privacy settings;
- your application requests only those permissions that are essential for the provided service;
- If the user refuses to grant permission, you should not try to persuade him otherwise. You can provide a permission-free alternative to the service. Invite the user to find the nearest office on the map and not based on the device location;
- A user grants his consent as a certain active action. For example, an automatic window closing, intentional or accidental dialogue exit is not considered as a consent;
- information about requested permissions is available to users. You can embed this information into the app and post a link or data tag in the app description on the RuStore store;
- You have informed the users about all requested permissions and the purposes for which they are requested. In case of any inconsistencies we may request some more clarifications.

We ask you to take reasonable steps to prevent the use of your app for illegal purposes. If you don’t follow the above guidelines, the application and the developer account it may be blocked on the RuStore.

5. App type:

When releasing an app, make sure to label it as "Paid" if it lacks any free features or functionality.

Free apps can still offer paid subscriptions and in-app purchases.

For subscription-based apps, users should have access to certain app features and functionality before subscribing. It's crucial to clearly inform users about the availability and cost of a subscription in such cases.

To set up paid content for your app, refer to the guides "[How to Create a Paid In-App Item](#)" and "[How to Create an App Subscription](#)." Keep in mind that deliberately misleading users may lead to the removal of your application from RuStore.

6. App name, descriptions, icon and screenshots

6.1 App name and descriptions, including the revision history, must not contain:

- Avoid using clickbaits: do not use the words "best", "only", "forbidden", "secret", "most" in headlines and short descriptions. If you use the word "official" to describe your application, we will ask you to document this status;

We also ask you to refrain from other verbal and graphic ways to mislead the user.

- overt or direct insults, rabble-rousing, obscenities;
- trademarks and intellectual property of others, including logos and names;
- links to third party app stores or storefronts, phishing and irrelevant links.

Before submitting an application for review, use the spell checking services to correct the description.

6.2 Emoji and special characters are not allowed in the app name and short description unless they are part of the brand.

If the app name contains an acronym or professional terms and abbreviations, we recommend that you clearly indicate the purpose of the application in a brief description.

The application must have the same name both on the storefront and when installed on the user's device.

6.3 For an app description to be effective, it should adhere to the following guidelines: it must be written in Russian, clear, consistent, and easily understandable. However, there may be exceptions when it comes to terms and names. The description should accurately reflect the features and content that are currently available in the app at the time of publication.

To ensure a professional presentation, we advise against the use of emojis, emoticons, or excessive repetition of special characters in the detailed description. Additionally, it's recommended not to overuse keywords, capitalization, or any other methods that may excessively draw users' attention and appear intrusive.

6.4 The following guidelines should be observed for app icons:

- They must not imitate public application icons.
- They should not utilize trademarks or intellectual property belonging to others, including logos that could potentially mislead users.
- If the app does not have any unread messages or notifications within the application, it should not display an icon indicating otherwise.

Furthermore, it is crucial to ensure consistency in the app icons across both the storefront and the user's device once the app is installed.

6.5 Screenshots ought to accurately depict the mechanics and functionality of the app or game, rather than merely displaying the title, login page, splash screen, or artwork. It is considered inappropriate to curate a collection of screenshots that solely feature art or elements unrelated to the application interface.

When it comes to gaming applications, screenshots should faithfully represent the current graphics quality at the specific level.

It is important to refrain from including the following elements in screenshots:

- Interface elements from Android or other apps.
- Functionality that is either not yet implemented or not present in the app.

6.6 Video Requirements:

- it must be directly related to the application (for example, you can upload a trailer, gameplay recording, etc.);
- it must not contain inappropriate content (see Application Requirements);
- the video must be public;
- the page on which the video is posted must be open to all users. If you add a video from a page with limited privacy settings, then the video must be moved to a group or public page with open access.

The video duration cannot exceed 1 minute;

6.7 App layout:

- It is essential to assign the [correct category](#) to the application from the provided list. In the event of an incorrect category selection, the moderator has the authority to make the necessary changes.
- It is crucial to ensure that the application is labeled with the appropriate age category. If uncertain about the applicable age limits, please refer to the [guidelines for age classification](#).


Age ratings

Age ratings apply to all applications published on the RuStore. Those restrictions help users understand whether an app contains potentially inappropriate content for a certain age group.

Age ratings are not related to the app target group.

Specify an age rating when publishing an app or a new version of it if the update introduces potentially inappropriate content for a certain age group. This information will help RuStore users decide whether to install the application.

How to set an age rating

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Select an application.
4. Select a required app version.
5. Click .
6. Go to the "Age rating" block.
7. Select one of the restriction options from the drop-down list.
8. Click "Save".



← Go to apps

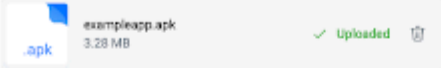
example_app

- Information
- Versions
- Reviews
- RuStore Geo [Beta](#)
- Monetization [Beta](#)
 - Subscriptions
 - In-App purchases
 - Payments management
 - Payment statistics
- Push notifications [Beta](#)
 - Projects
 - Push notification statistics

Versions > Uploading a new version Clear All

Version

Upload APK

 ✓ Uploaded

Comment for the moderator 11/180

example_app

To avoid possible update errors, sign your apps with the same signature as in other stores (for example, Google Play) before uploading them.

If the full functionality of your app is only available to registered users, please provide your login details. As well as any other information that will help the moderator check the app.

App Information

Name for the user 11/50

example_app

This is what the app name will look like in the window. You can always change it.

Type

Non game app

Game

Cost, ₽ 3/6

150

RuStore commission is 15%. For more information about paid apps, see the [developer's guide](#).

Categories

Main

Adventure

To learn how to choose a category for an app, see [the developer's guide](#).

Additional information

Arcade

Age category

16+

Rules for choosing an age category can be found in [the developer's guide](#).

Description

Short description 11/80

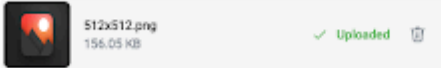
example_app

Description 11/4000

example_app

Media files for the folder

Icon for the app catalog

 ✓ Uploaded

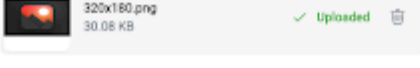
Screenshot orientation

Landscape

Portrait

Choose whether the screenshots are oriented vertically or horizontally.

Screenshots of the app

 ✓ Uploaded

Upload up to 10 images with a maximum size of 1 MB
JPG, PNG format (1:6-9, 320x180px - 3840x2160px) [Select an image](#)

[Promote the app](#) [Earn money from advertising](#)

[Submit for moderation](#) [Cancel](#)

How to choose an appropriate age rating

The RuStore uses the following age ratings:

	Description
3+	Inappropriate for children under 3 years old. Please note that not all apps with this age rating are designed specifically for children.
7+	Some violence or scary scenes. Not recommended for children under 7 years old. Please note that not all apps with this age rating are designed specifically for children.
12+	Moderate violence, scary scenes, obscene content. Not recommended for children under 12.
16+	Violence or sexual content. Not recommended for children under 16.
18+	High levels of violence, sexual content, references to drugs or alcohol. For adults only.

Age ratings have nothing to do with a game's complexity. We recommend that parents should be guided by the age ratings to make sure that the game content is suitable for the child.

How to choose an appropriate category for your app

You can choose a category when publishing your application on the RuStore. The categories you assign to your app help users discover it on the RuStore.

How to assign a category

1. Open the [RuStore Console](#).
2. Click the "Applications" tab.
3. Click "Add an App".
4. In the window that opens, enter the name of your application.
5. Select the type of your application in the "Type" section: a non-game app or a game. The non-game apps are located on the main page of the RuStore on the "Applications" tab, and the games are on the "Games" tab.
6. In the drop-down list, select the main category for your application, taking into account its features. The application will be placed in the selected category on the store.
7. Select an additional category whenever required. The application will be placed in two categories on the store at once.

Type

Non-game app

Game

Categories

Main

Select a category

To learn how to choose a category for an app, see the [developer's guide](#)

Additional information

None

Age category

Select a category

Rules for choosing an age category can be found in the [developer's guide](#)

Choose the category that is the most relevant to your app. This will make it easier for users to find the app.

Category	Examples
----------	----------

Health & Sports	<ul style="list-style-type: none"> ● Sports apps; ● Health and active lifestyle; ● Safety instructions; ● Sports news and commentary; ● Account tracking; ● Fantasy sports; ● Sport reviews
Food & Drink	<ul style="list-style-type: none"> ● Cafes and restaurants; ● Delivery services; ● Recipes; ● Wine catalogue; ● Gourmet apps
News	<ul style="list-style-type: none"> ● Newspapers; ● News aggregators; ● Magazines; ● Blogs
Transport	<ul style="list-style-type: none"> ● Navigation tools; ● GPS; ● Taxi; ● Public transport timetables and maps; ● Traffic Rules; ● Driving News
Entertainment	<ul style="list-style-type: none"> ● Interactive entertainment; ● Streaming video, movies and TV shows; ● Music apps; ● Players; ● Broadcasting; ● Reading apps; ● Directories; ● Textbooks; ● Dictionaries
Education	<ul style="list-style-type: none"> ● Study apps; educational materials; ● Dictionaries; ● Educational games; ● Language learning apps

Finance	<ul style="list-style-type: none"> • Banking and payment apps; • ATM locator apps, tax apps, stock portfolio management apps; • Financial news; • Tip calculators
Social	<ul style="list-style-type: none"> • Apps for couples and friends; • Social media • Check-in Apps
Shopping	<ul style="list-style-type: none"> • Shopping and auction apps • Gift coupons; • Price comparison and shopping list apps • Product review apps
Games	<ul style="list-style-type: none"> • Arcade; • Quizzes • Puzzles • Racing • For kids • AR-games • Indie • Casino • Casual • Cards • Music; • Board games • Adventures • RPG games • Family • Simulation • Conversation games • Sports • Strategies • Action
Medical	<ul style="list-style-type: none"> • Pharmacy apps • Medication guides; • Calculators; • Books for health workers • Magazines and news about medicine

Tools	<ul style="list-style-type: none"> • Browsers; • Email management tools; • Service personal accounts management tools; • Documents viewing and editing tools; • Parcel tracking and job search tools
Ads and services	<ul style="list-style-type: none"> • Search for real estate, mortgages, requests for renovation, interior design and housekeeping • Search for work, services and craftsmen • Booking services, searching for fellow travelers and taxis • Guidebooks • Maps • Local business info; • Travel planning and tour booking services
Public services	<ul style="list-style-type: none"> • Receiving public services; • Documents registration; • Doctor's appointment • Requests for certificates and extracts; • Settlement of taxes; • Check and payment of traffic police fines

RuStore Ratings & Reviews

Users can rate your app or write a review on the RuStore. One user can rate an app only once, though he is allowed to update his review at any time.

The [RuStore Console](#) makes it possible for you to track your app ratings and reviews. We will also add data analysis tools in a short while.

You can write a public response to a review. This option is given to you so that you can build better customer relationships. You can:

- thank the user for a good review;
- answer the questions;
- collect user feedback;
- find out about errors in your application.

How to write a response to a review

1. Stay on topic. The answer must be clear, truthful, useful and relevant to the review content.
2. Be polite. Your goal is to help users find a solution to a problem, not scare them away. Do not post insults, threats, disparaging or hate speech. Do not provoke users to provide such feedback.
3. Try to respond to all reviews. This is how you show users that you value their opinion. The app tends to get higher ratings if the developer responds to the reviews users post about it.
4. Avoid negative responses even if the user is extremely negative.
5. Try to speed up work with positive reviews, then you will have more time to solve complex user problems. For example, you can create quick reply templates on your local computer and paste a ready-made response. There are no built-in templates on the RuStore yet, but we are going to add such functions in a short while.
6. If you use quick reply templates, try to rephrase it from time to time. You should not sound like a robot.
7. Do not post advertisements and promo codes for other services or in-app purchases in the reviews as this is not relevant for users at all.
8. Avoid posting sexual content or obscene language.

If you don't follow these guidelines we may block your application or the developer account.

How to deal with negative reviews

Oftentimes there is a chance to change the user's opinion about your app for the better if you respond to his or her negative review correctly.

How to deal with negative reviews

1. Avoid one-word answers without apologies. It is next to impossible to comfort an angry user with a short reply, it will rather look rude and cause much more negative effects.
2. Avoid too much formality. Formal language often makes you sound like a robot.
3. It is important to show your empathy for the user. If there is a real problem in your app, you need to acknowledge the mistake.
4. Try to help the user in your answers. Avoid using quick reply templates when responding to negative reviews. You should always consider the context and see if the template reply is good enough for this very case.
5. Thank the user for error messages in your application. Feedback like this can keep users on your app.
6. Provide the user with details on every problem solution. Sometimes users don't understand why you removed their favorite feature. Reasonable answers can also help to cope with negative feedback.
7. Respond to reviews quickly. The sooner you help a user solve a problem, the more likely he decides to continue using the app and improve the app rating.

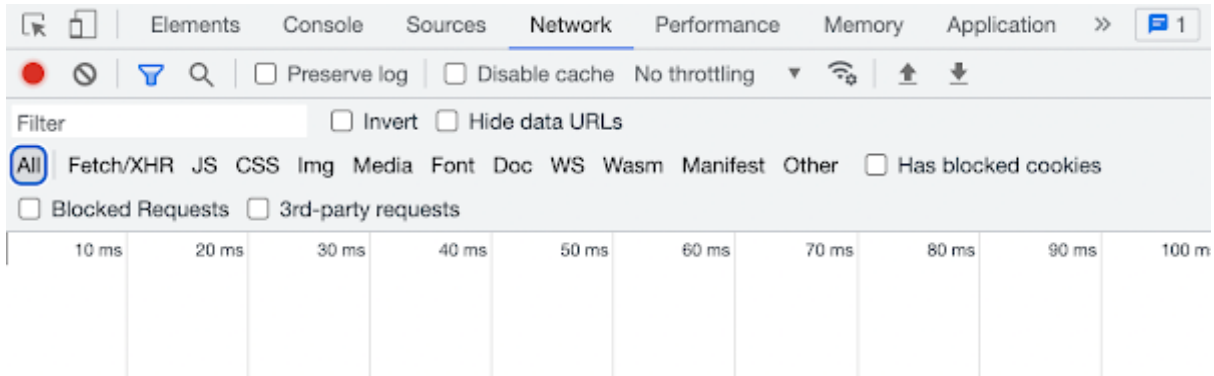
Please [contact us](#) if you find biases, swearing or insults in a comment or review. We will try our best to solve the problem.


How to capture HAR logs (Google Chrome)

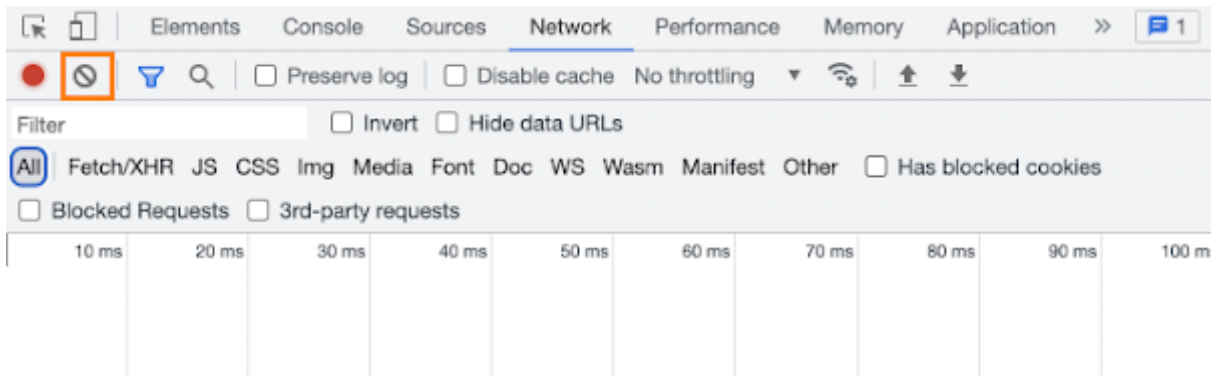
HAR is a log of a web browser's interaction with a site. They are required to solve problems when using the browser.

To capture HAR logs:

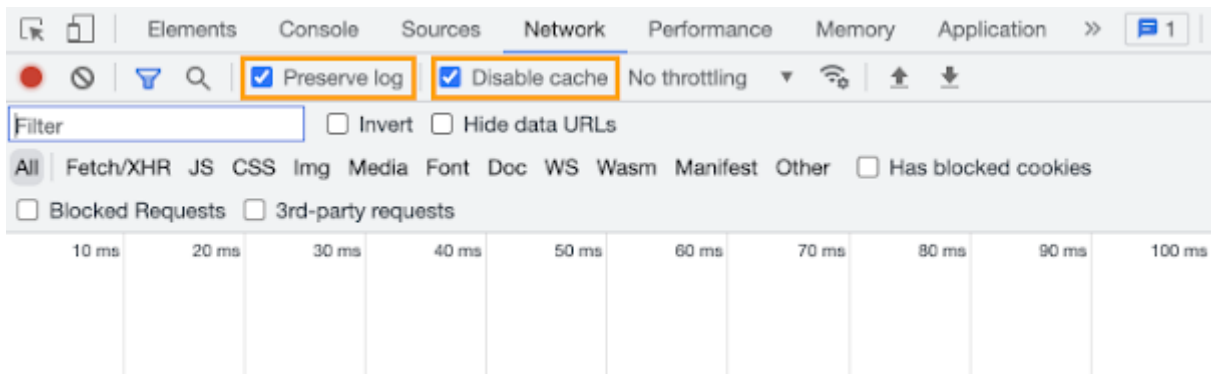
1. Press F12 on your keyboard to open the developer tools window.
2. Select the Network tab.




3. Click  to clear the network log.

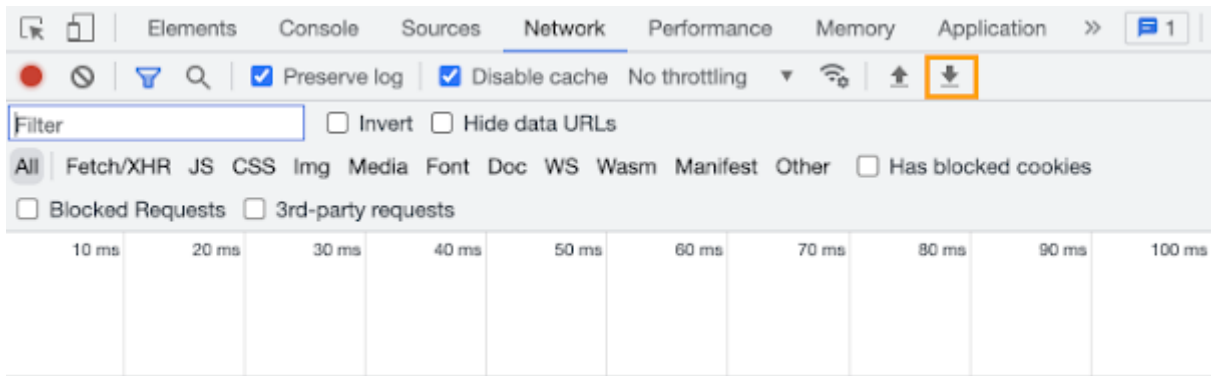


4. Check the boxes in the "Preserve log" and "Disable cache" fields.



5. Repeat the steps that caused the problem.

6. Click  to save HAR logs.



7. Select a destination folder to save the logs.
8. Click "Save".

HAR logs will be collected and saved to the specified folder.

Monetization

Follow the steps below to enable monetization:

1. Fill out an application for monetization.
2. Set up the RuStoreSDK to connect payments.
3. Create subscriptions or one-time purchases.

How to enable monetization on the RuStore

Enable monetization to sell in-app products and subscriptions on the RuStore store. Fill out a monetization request and sign it with EDS.

EDS is an electronic signature issued by the Federal Tax Service for your company. If you do not have an e-signature, contact a CEO, chief accountants, lawyers, HR specialists or procurement coordinators. Make sure that the employee with an EDS is entitled to sign financial documents. If in doubt, show them our [Terms of Service](#).

Monetization is available only to legal entities/individual entrepreneurs and application owners.

How to create and sign a request

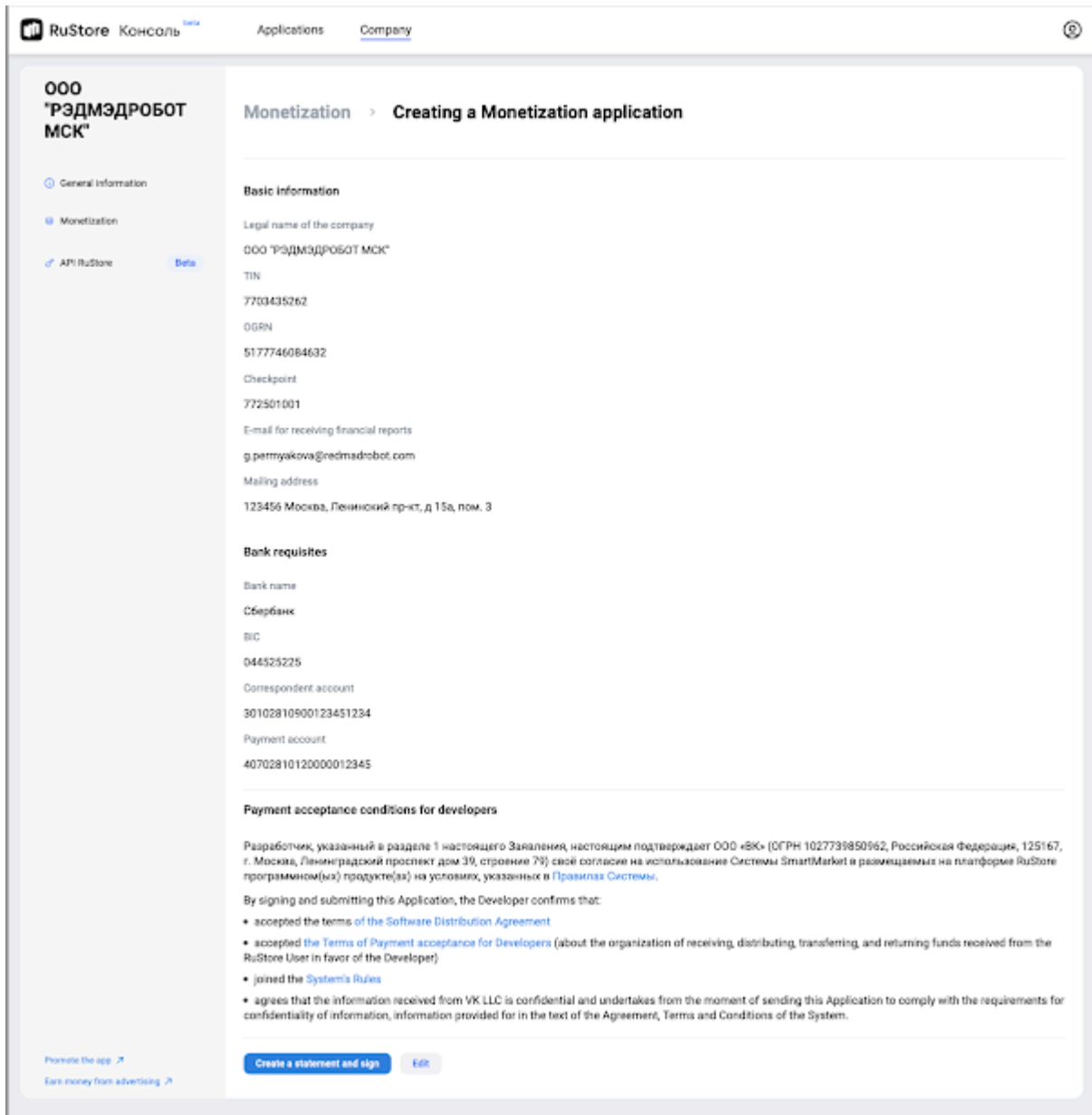
Install the CryptoPro plugin to sign the request.

1. Open the [RuStore Console](#).
2. Click the "Applications" tab at the top of your screen.
3. Select "Subscriptions" in the left side menu, then go to the "Monetization" section.
4. Click "Fill out the form".

If you accepted the terms before 12/22/2022, re-sign your application with an electronic signature.

5. Fill in the data in the pop-up window.
6. Click "Save and Continue".
7. Read the payment terms for developers.
8. Click "Create and Sign the request".

The screenshot shows the 'Monetization' section of the RuStore console, specifically the 'Editing a monetization request' form. The form is divided into two main sections: 'Basic information' and 'Bank requisites'. The 'Basic information' section includes fields for 'Legal name of the company' (000 'РЭДИМЭД'РОБЮТ МСК'), 'INN' (7703410242), 'OGRN' (5177148044632), 'Developer' (772001001), 'Email for financial reports' (g.demyanov@redimedit.com), and 'Mailing address' (125028 Москва, Ленинский просп., д. 15к, этаж: 3). The 'Bank requisites' section includes fields for 'Bank name' (Сбербанк), 'BIC' (044523223), 'Correspondent account' (30702810000123451234), and 'Payment account' (4070281012000012345). There are also two checkboxes: 'For security reasons, we do not pass on earlier entered values' and 'To change your TIN write to support'. At the bottom of the form, there are 'Save and continue' and 'Cancel' buttons. The footer of the console includes links for 'Privacy Policy', 'Distribution Agreement', and 'Write to support', along with the 'CryptoPro' logo.

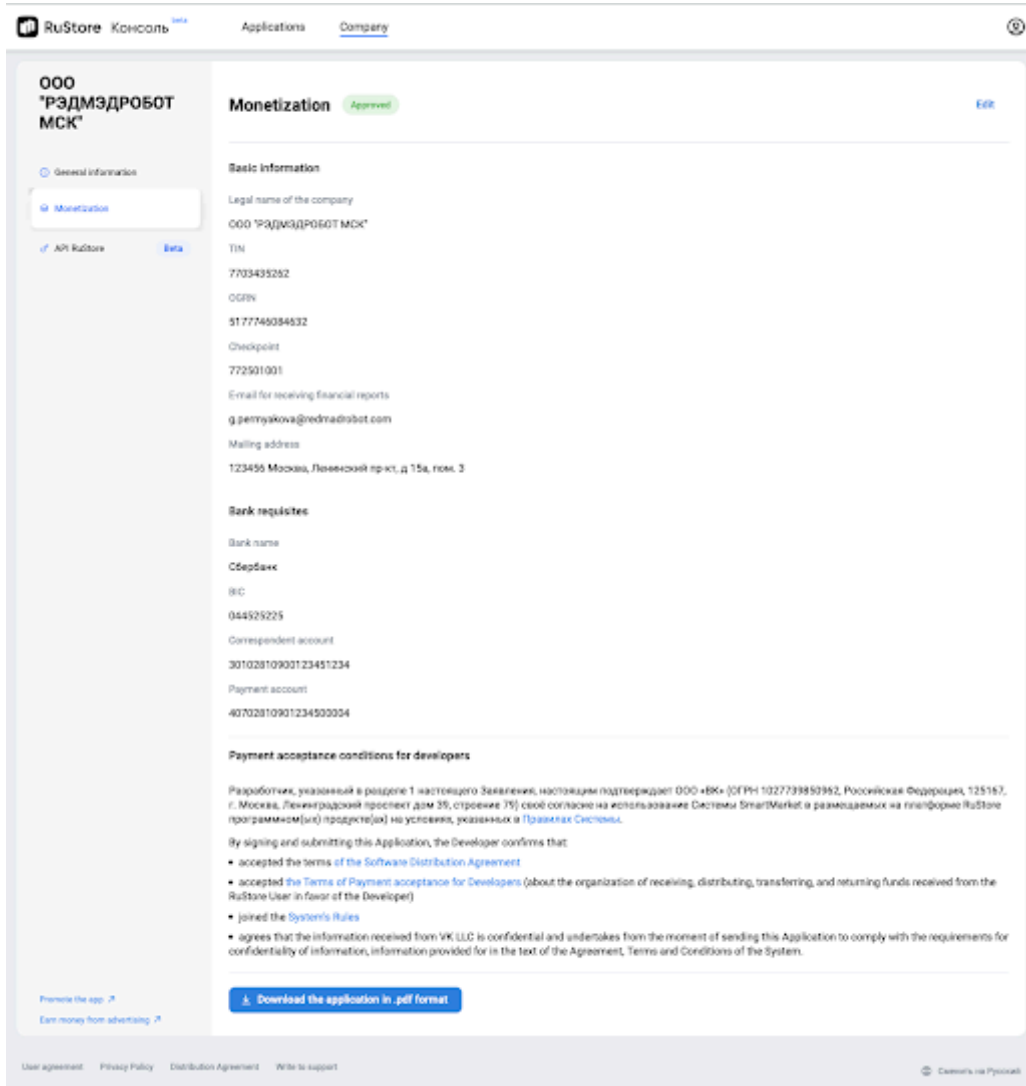
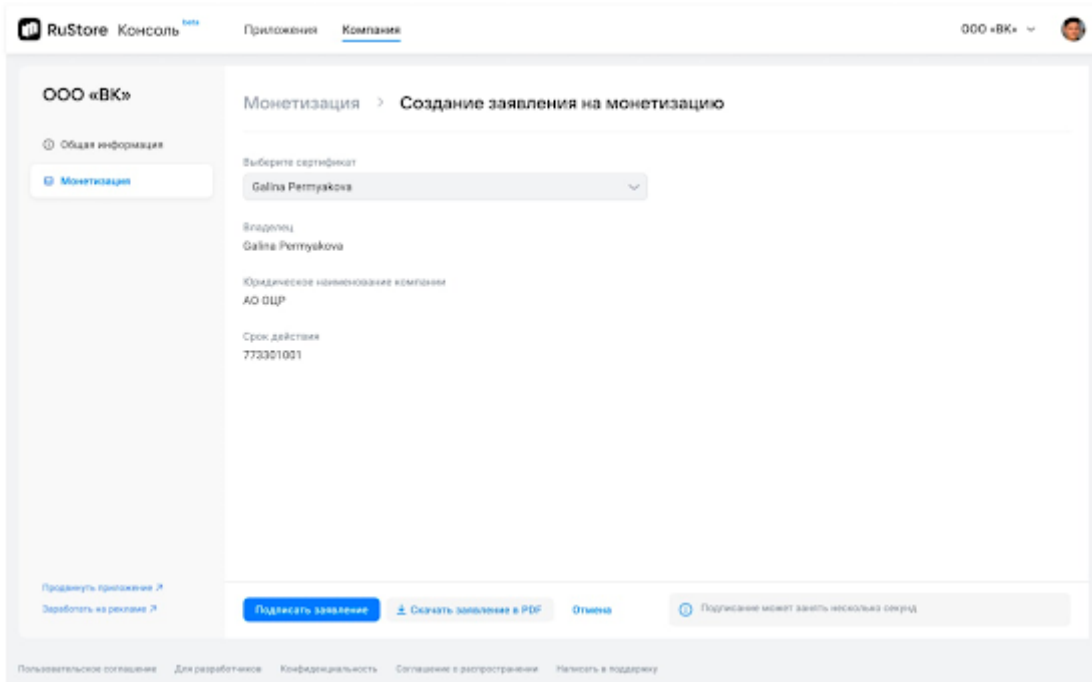


Select a signing certificate. The certificate must be issued on behalf of your company — we will check that the specified TIN matches the TIN of your company.

You can check your TIN in the certificate properties: in the operating system tools or in a certificate program (for example, CryptoPro).

Click "Sign a request". It may take a few seconds to sign your request.

Once there, you can download the approved monetization request. Click "Download a PDF-request". [Request example](#)



Test payments

Background

You can now test payments for subscriptions and in-app billing in your apps. The test purchase is carried out as a regular payment, though using test bank cards.

Functionality

The Test Payments function provides the following options:

- Test mode control via [RuStore Console](#).
- Testing of payment scenarios in your app, which is carried out in an isolated test environment using special test cards and without affecting the release app version.
- View test payments history in [RuStore Console](#).

Technical specifications

- You can perform testing before publishing an app version to all users.
- In test mode, you can purchase the same products and subscriptions that were added to the release app version. At that, it is not necessary to publish them.
- You can test the purchase of non-consumable items repeatedly (for example, access to the full app version or a skin in a game). To do that, you can return the payment via RuStore Console, and then make a purchase again in the application.
- In test mode, special parameters are applied to subscriptions. They are independent of and do not affect the settings you select for the release app version. These technical features are designed to simplify and speed up testing.

The parameters in the table below are the same for all subscriptions in test mode.

Parameter	Value
Maximum number of subscription charges. After completing the specified number of debits, the subscription is automatically closed	12
Period between subscription charges	10 mins
Frequency of resetting the free and starting period (purchase of subscriptions with a free and starting period). After the reset, you will again be able to purchase a subscription for free or at the initial price. This way you can test payment for a	Once in 3 hours

subscription several times under special conditions.	
------------------------------------------------------	--

Restrictions

- The application owner is the only user who can test payments. He has exclusive access to the Test Payments section. Options to add other testers are now under development.
- Testing of a paid app purchase is not available in RuStore.
- The RuStore app does not display the history of test purchases and test subscriptions. This means that test subscriptions cannot be canceled or renewed.
- You can pay for test purchases using test bank cards only.
- To test payments, at least the first app version must be moderated.

Before you start testing payments

- Integrate the SDK that supports the payment testing function into the tested app — **RuStore SDK billingclient 3.1.0 or higher**.
- Make sure that the Android device on which you plan to test your app has **RuStore version 1.29** or higher installed.
- Make sure that your company is not blocked, that it has monetization enabled, and supports payment functionality in the app. You can set up subscriptions and paid products in RuStore Console, but not publish them to users.

See also

- [How to enable monetization](#)
- [How to create a paid in-app product](#)
- [How to create an app subscription](#)

Enable Test Mode

You can test payments after moderation is completed and before you publish a new app version. You can also test purchases if an app has already been published.

Testing mode is enabled for the tester in the selected app only. At that, the same user will make real purchases in other applications. Besides, all other users in this application will make real purchases as well.

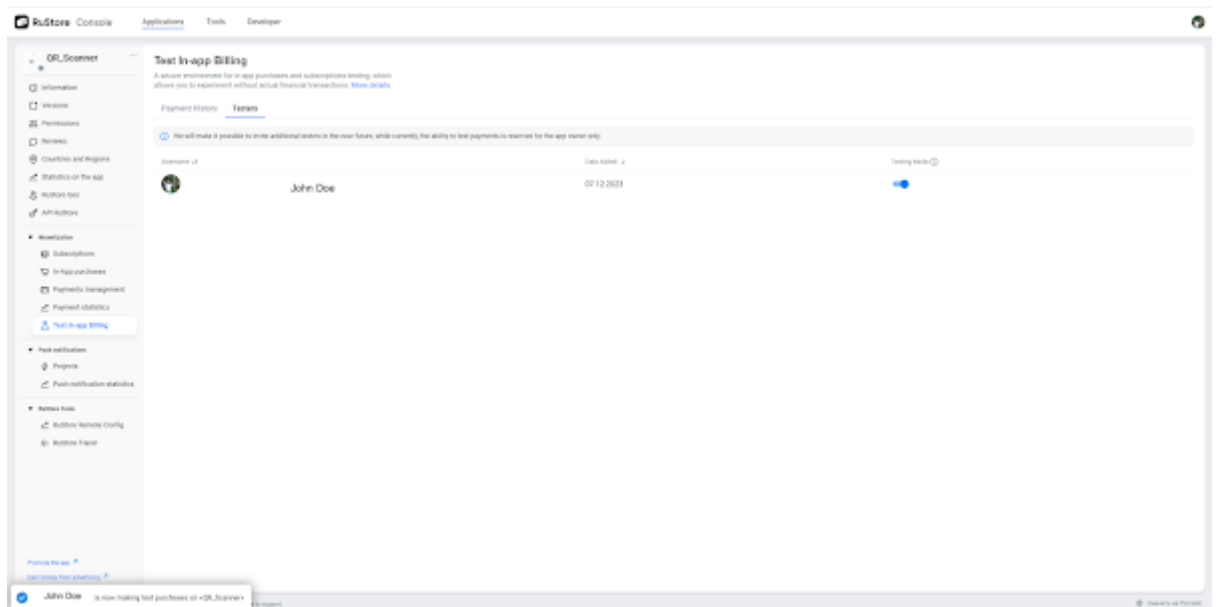
Test mode is disabled by default.

To enable test mode:

1. Open [RuStore Console](#).
2. Select **Applications** from the top menu.
3. Go to the **Monetization** → **Test Payments** tab.
4. Switch to the **Testers** tab.

If the tab displays the **Add yourself to the list of testers** window, click **Become a tester**.

5. Enable the switch in the **Test Mode** column.



Upon turning on the test mode, the tester will make test purchases in your application, and purchases of products that have not yet been published will become available.

Test in-app purchases

To apply the changes after switching test mode, you must either wait a few minutes or restart your app if it was open.

Follow the steps below to test payments:

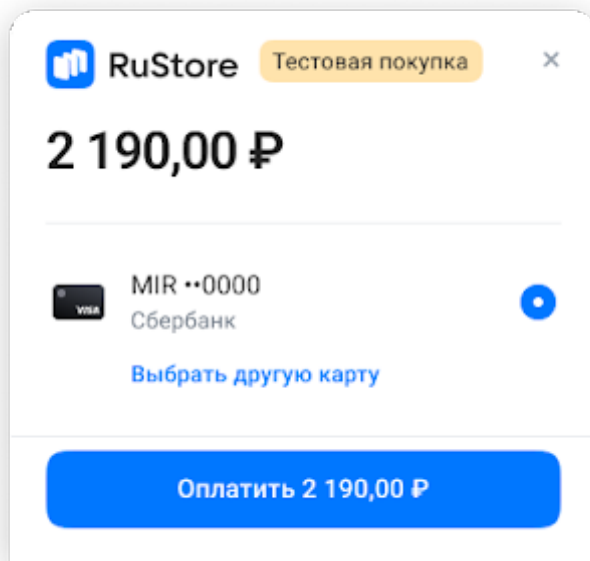
1. [Turn on test mode.](#)
For the changes to take effect, you must either wait a few minutes or restart your application if it was open.
2. If you are not yet authorized in RuStore on the current device, log in via a VK ID you used to sign up in RuStore Console as the app owner.

See also

- [How to view your VK ID in RuStore Console](#)
- [How to log in to RuStore app with a VK ID](#)

3. Open an app on your device, or install it if you haven't already.
4. Make a purchase. For example, subscribe.

During payment, a payment window with the **Test purchase** label will appear.



Note. If the Test Purchase label does not appear, it means that you are making a purchase not in the test environment, but in the release app version. Make sure that [test mode is enabled](#) and that you used the correct VK ID to log in to RuStore.

5. Add one of the [test bank cards](#) and select it for payment. Once added, this card will be saved in the test profile and further purchases with this card will be available in one click.

When your account is in test mode, use test cards only. If you try to pay for a purchase with a real bank card, an error occurs.

Note

- Use different test cards to test both successful and unsuccessful payment scenarios. For example, simulate an error when the payment amount exceeds the limit on a bank card.
- If you need to test the grace and hold periods, use a [test card with a balance of 1 kopeck](#). Using this card you can buy a subscription with a free period. After the end of the free period, there will be one unsuccessful charge attempt, and then the subscription will immediately switch to the grace or hold period, if they are preconfigured for this subscription.

For more details about periods, see [How to create an app subscription](#).

6. Click **Pay**.
7. Check if the payment was successful and if your app has processed the purchase.

If the payment was successful, but you still have no access to the purchase, check the implementation in your app. For other problems with testing scenarios, contact support support@rustore.ru.

Note. When Test Mode is enabled, 'Get Purchase Information' and 'Get Shopping List' methods will only return test purchases for that user.

Server validation of test payments and subscriptions

Using API requests, you can retrieve:

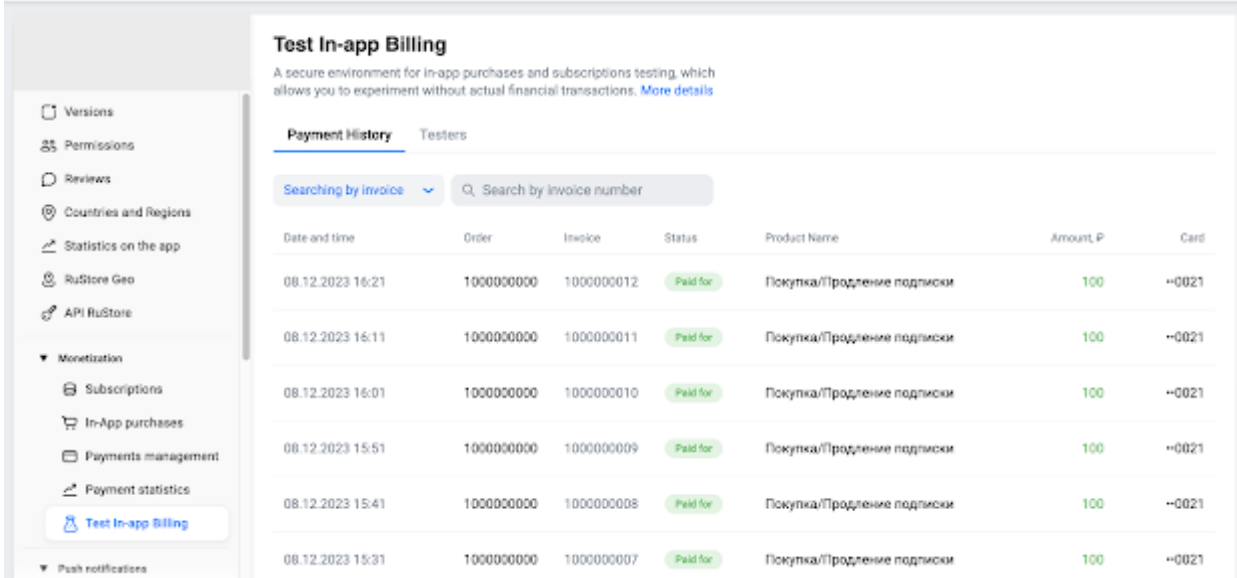
- Test payment details;
- Test subscription details;
- Test subscription status;
- Confirmation of subscription availability.

Before using RuStore API, create a new key and specify the methods for testing.

Test Payments History

To view the history of test payments:

1. Open [RuStore Console](#).
2. Select **Applications** from the top menu and open the app in which the test purchase was made.
3. Go to the **Monetization** → **Test Payments** tab.



Test In-app Billing
A secure environment for in-app purchases and subscriptions testing, which allows you to experiment without actual financial transactions. [More details](#)

Payment History Testers

Searching by invoice

Date and time	Order	Invoice	Status	Product Name	Amount, P	Card
08.12.2023 16:21	1000000000	1000000012	Paid for	Покупка/Продление подписки	100	****0021
08.12.2023 16:11	1000000000	1000000011	Paid for	Покупка/Продление подписки	100	****0021
08.12.2023 16:01	1000000000	1000000010	Paid for	Покупка/Продление подписки	100	****0021
08.12.2023 15:51	1000000000	1000000009	Paid for	Покупка/Продление подписки	100	****0021
08.12.2023 15:41	1000000000	1000000008	Paid for	Покупка/Продление подписки	100	****0021
08.12.2023 15:31	1000000000	1000000007	Paid for	Покупка/Продление подписки	100	****0021

The Payment History tab displays all payments made in test mode. In terms of functionality, this section is similar to [Payment Management](#), which displays user purchases made in the release app version.

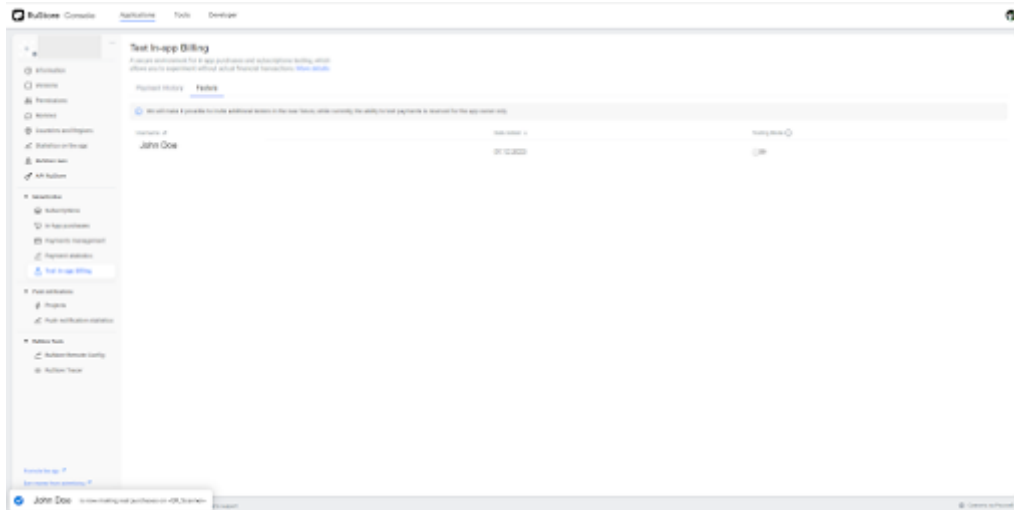
The Payment History tab features:

- Search for a test payment by account number and order number.
- Detailed information for each payment by clicking on it.
- Payment return to retest the purchase of a non-consumable item. To do this, click on the payment, and then on the **Return payment** button.

Disable Test Mode

Once payment testing is complete, you can disable test mode. This is not necessary; you can remain in test mode whenever so required.

1. Open [RuStore Console](#).
2. Select **Applications** from the top menu.
3. Go to the **Monetization** → **Test Payments** tab.
4. Switch to the **Testers** tab.
5. Deactivate the switch in the **Test Mode** section.



Once you disable the test mode:

- The tester starts making real in-app purchases.
- All payments made in test mode are saved in RuStore Console on the **Payment History** tab.
- Purchased paid products remain available to your account the next time you turn on test mode.
- Test subscriptions will be disabled automatically after 12 charges.

Further steps

Once you've tested payments to ensure they operate properly, you can publish subscriptions and paid items to your app, and then publish a new app version.

Test Bank Cards

To test purchases, you can use the following test cards.

Always successful

Card number	4111 1111 1111 1111
ECI	05
CVC\CVV	123
Expiration date	2024/12
3-D Secure verification code	12345678

Card number	5100 0000 0000 0008
ECI	05
CVC\CVV	123
Expiration date	2024/12
3-D Secure verification code	12345678

The buyer is offered to select the 3-D Secure verification result.

Card number	2201 3820 0000 0013
ECI	02
CVC\CVV	123
Expiration date	2024/12

Returns errors

After three unsuccessful payment attempts, the payment will be returned to the RuStore payment system with a payment error.

Card number	5100 0000 0000 0180
ECI	00
CVC\CVV	123
Expiration date	2024/12

Card number	4444 4444 4444 4422
CVC\CVV	123
Expiration date	2024/12

Card number	4444 4444 1111 1111
CVC\CVV	123
Expiration date	2024/12

Card with 1 kopeck balance

Card number	4954 8493 9714 9582
CVC\CVV	123
Expiration date	2024/12
3-D Secure verification code	12345678

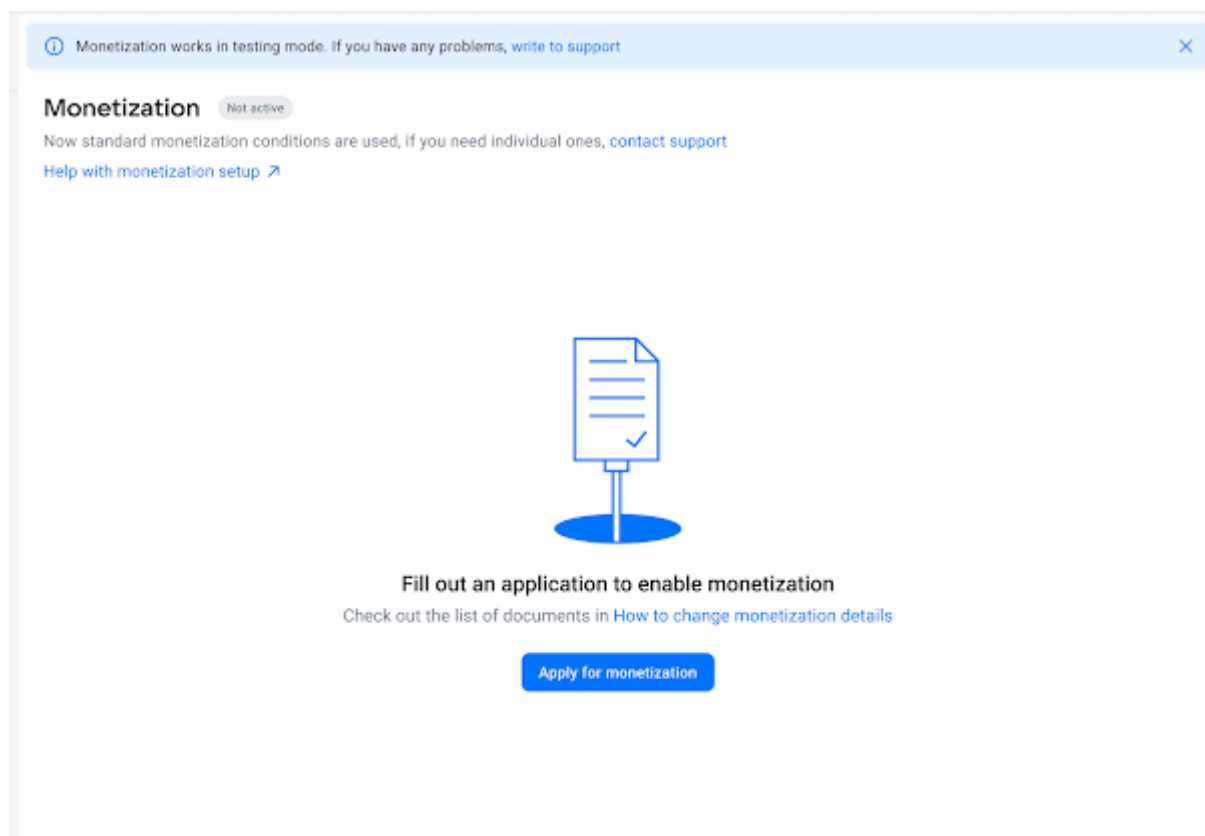
Monetization for foreign companies on RuStore

Foreign companies that are registered within RuStore can now enable monetization and receive income from both paid applications or in-app purchases, including subscriptions.

Monetization is available under the terms of a standard [Additional agreement](#). The actual settlement currency will be displayed in the form when submitting the monetization request.

Create and fill out a monetization request

1. Open the [RuStore Console](#).
2. Click the **Company** tab at the top of your screen.
3. Select **Monetization** in the left side menu.
4. Then click **Apply for monetization**.



5. Fill in the data in the new window.
 - **General info.** The **Trade and legal name** field is filled out by default according to your constituent documents. If the data is incorrect, contact support and submit a new request once you fill in the correct data.

Note. We may ask you to verify your rights to the specified trade name. You are not required to have your rights legally registered, however, in this case, we will ask you to ensure that the use of this trade name does not violate the rights of third parties.

ⓘ Monetization works in testing mode. If you have any problems, write to support✕

Apply for monetization Draft

Fill out the form, if you have any questions, [contact support](#) Clear all

[Help with monetization setup](#) ↗ [Supplementary Agreement](#) ↗

Main information

Trade company name
VK

Legal company name
VK LLC

1. Signatory differs from company owner.

If the representative is different from the company owner on RuStore, fill in the information according to the representative authority document. Attach the decision on appointment, power of attorney, etc. Make sure that the person who will sign the [Additional agreement](#) on behalf of your company is indicated here.

Signatory differs from company owner

In case the signatory differs from the company owner, please provide the required details



Position

For example, Product Manager

Name

Last name

Middle name (if included)

E-mail

ivanova@digitalforces.ru

Document establishing the signatory's authority 

Upload 1-3 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

[Choose file](#)

2. Company information

- Make sure that **Tax number** and **Legal address** match the information in your constituent documents.
- If the legal address does not match the actual one, you must also indicate the actual address where the company operates.

Company information

Tax number
1230572640

Legal address

305 E 75TH St New York NY 10021-3025 USA

Actual address differs from legal one



Actual address

1 Kuyper NY 10960-1016 USA

3. Contacts for user feedback

Note. The RuStore support service can send these contacts to users or publish them on the app page.

- When filling in the Contact person section, specify the up-to-date contact details so that users can get back to you quickly regarding app support, as well as financial claims (refund / payment cancellation requests, fraudulent payments notifications).
- The Phone Number and Email sections are intended for internal communications with RuStore and will be visible to users. Please enter the international phone number in the **Representative number** field.
- You can also specify the preferred method of communication and add a link to instant messengers, a feedback form, or another tool used to receive feedback from users (optional). We also recommend checking the availability of this tool for users from the Russian Federation.

Contacts for users regarding application support and complaints

E-mail

e-mail@address.com

You can mention several ones separated by a comma

Phone number (optional)

+7 (909) 860-13-33

Address (optional)

305 E 75TH St New York NY 10021-3025 USA

Links to Messengers, Feedback Forms (optional)

4. Contract information

- Company's chief accountant full name.
- Accounting statements: income statement, balance sheet.
- Documents confirming tax residency. A tax residency certificate is provided to avoid double taxation. Going forward, we will ask you to update this document annually.

Contract information

Chief accountant's name

Financial statements 

Upload 1-10 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

[Choose file](#)

Document confirming the tax residency of the company 

Upload 1-10 files, up to 10 MB each.
Format: DOC, DOCX, PDF, PNG, JPG, JPEG

[Choose file](#)

5. Bank details.

- Provide the company bank account details in either a Russian or foreign bank, where the settlements will be processed.

Note. If there is no currency listed which would be convenient for you to make payments, please contact support@rustore.ru and we will consider your offer.

Bank details

Settlement currency

Ruble (₽)



Company bank details

Beneficiary Bank name

Beneficiary Bank address

305 E 75TH St New York NY 10021-3025 USA

Beneficiary Bank account number (or IBAN if applicable)

For example: 30101810400000000712

BIC/SWIFT/ABA/Another Bank Identifier

For example: SGAZRU22

Beneficiary Bank correspondent account number

For example: 0473620564382556406754

- If you want to make payments through a specific correspondent bank, please specify the required details in this section. We will send your income to these details for further transfer to you. Otherwise, we will transfer your income to the details specified above through our standard correspondent bank.

Correspondent (Intermediary) Bank Details (optional)

Correspondent bank name

Correspondent bank account number

For example: 0473620564382556406754

Correspondent bank BIC/SWIFT/ABA/Another Bank Identifier

For example: SGAZRU22

Go to submit

 The data is automatically saved, you can return to filling in at any time

The [Additional agreement](#) outlines the process for currency conversion during payment.

Note. RuStore charges a fee on paid apps and in-app items (subscriptions, in-app purchases). The fee for non-residents is 15%. In cases where RuStore is required to be a tax agent of a non-resident, apart from the fee, the amount of taxes payable in accordance with the legislation of the Russian Federation will be withheld.

6. Click Submit a request.
7. Check the request text and click **Go to submit**. Your monetization request will be reviewed and approved by a moderator.

Note. You can withdraw the request at any stage and make the necessary changes whenever so required, and then re-submit for review.

Attention. In case of discrepancy, we will notify you by email and ask you to attach the correct documents, while we reserve the right to reject the request and / or either block the monetization option in exceptional circumstances.

Once you have submitted the moderation request, we will send the [Additional agreement](#) to your email. Read the agreement and sign it to enable monetization.

Note. Please note that the Supplementary Agreement must be signed by the person whose information you have indicated in the **Company Representative** section above.

Once monetization is enabled for you, you will be able to set up subscriptions, in-app products and publish paid apps. You will receive a monthly report on accepted payments. We will transfer payments to you in accordance with the [Additional agreement](#).

Payments are scheduled once a month at most, provided that the minimum amount has been reached since the previous payment.

Payments management

In RuStore Console you can monitor payments per monetization-enabled app.

Viewing payments information

1. Open [RuStore Console](#).
2. Select **Applications** in the top menu.
3. From the menu on the left select **Monetization > Payment management**.

Дата и время	Заказ	Счёт	Статус	Название товара	Сумма, Р	Карта
25.06.2022 12:23	110832	65085412	Оплачено	Диктатор	149,00	++1412
25.06.2022 12:23	110831	54729343	Оплачено	Как приручить дракона 3	299,00	++9821
11.10.2022 12:21	110835	45975466	Оплачено	Ежемесячная подписка на кино и сериалы	499,00	++3157
11.10.2022 12:03	110830	32058444	Возвращена	Пацаны	499,00	++2689
11.10.2022 11:57	110829	98867997	Оплачено	Дом дракона	649,00	++2366
11.10.2022 11:42	110828	05844724	Оплачено	Пес-самурай и город кошек	99,00	++3690
11.10.2022 11:38	110827	23947908	Оплачено	Ежегодная подписка на сериалы	3 499,00	++2378
11.10.2022 11:29	110826	06732278	Оплачено	Рик и Морти	549,00	++9917
11.10.2022 11:29	110826	06732566	Оплачено	Рик и Морти	549,00	++9917
11.10.2022 09:02	110825	55790466	Оплачено	Ежемесячная подписка на кино и сериалы	499,00	++3157

The payment information table is displayed only if the app was purchased at least once. The table contains:

- date and time of a purchase;
- invoice number and order number;
- payment card number and phone number from which the payment was made;
- purchase amount;
- invoice status.

Only subscription payments have order number. Order number is the same for all payments of a subscription.

Click on a payment to see the detailed information.

Счёт №54729343

25.06.2022 12:23

Оплачено

Номер заказа
110831

Название товара
Как приручить дракона 3

Описание
Безлимитный просмотр фильма в HD

Карта
**9821

Итого сумма платежа

299,00 Р

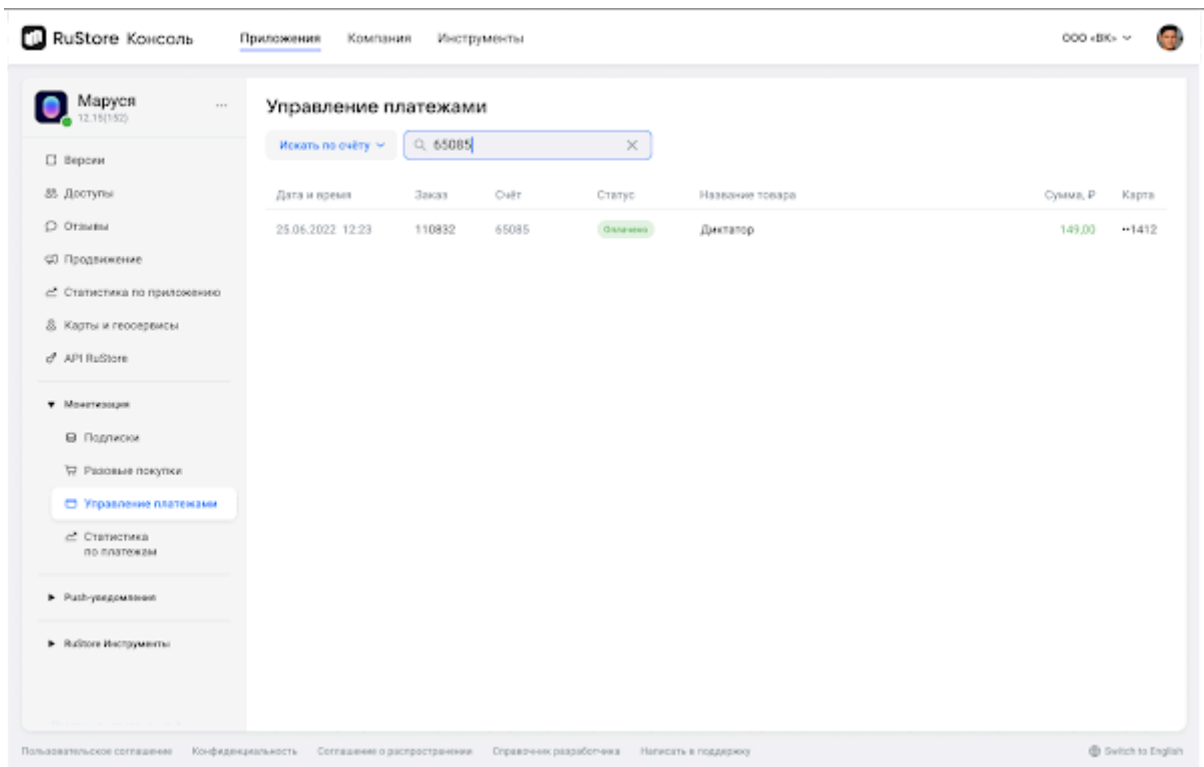
Закреть

Вернуть платеж

Payment search

To search and view payment information, follow these steps:

1. Open [RuStore Console](#).
2. Select **Applications** in the top menu.
3. From the menu on the left select **Monetization > Payment management**.
4. Enter the invoice number in the search field. The payment amount and status information will be displayed. Select one of the identifiers as a search criteria — invoice number or order number. After that, enter the invoice or order number in the search field accordingly. In the search results, you'll see all in-app payments that match the selected search criteria.



A payment can have one of the following statuses:

Processing: the money on the buyer's payment card is put on hold. The purchase is waiting to be confirmed.

Canceled: the money on the buyer's payment card is released from hold. The purchase is canceled.

Paid: transaction confirmed, the funds are debited from the buyer's account.

Refunded: the funds are returned to the buyer's account. The refund time may be up to 5 business days.

Payment return

To return a payment to the buyer, follow these steps:

1. Open [RuStore Console](#).
2. Select **Applications** in the top menu.
3. From the menu on the left select **Monetization > Payment management**.
4. Select the invoice that requires the payment return.

5. In the pop-up window, push **Return payment** to open the payment return page.

Счёт №54729343

25.06.2022 12:23 Оплачено

Номер заказа
110831

Название товара
Как приручить дракона 3

Описание
Безлимитный просмотр фильма в HD

Карта
**9821

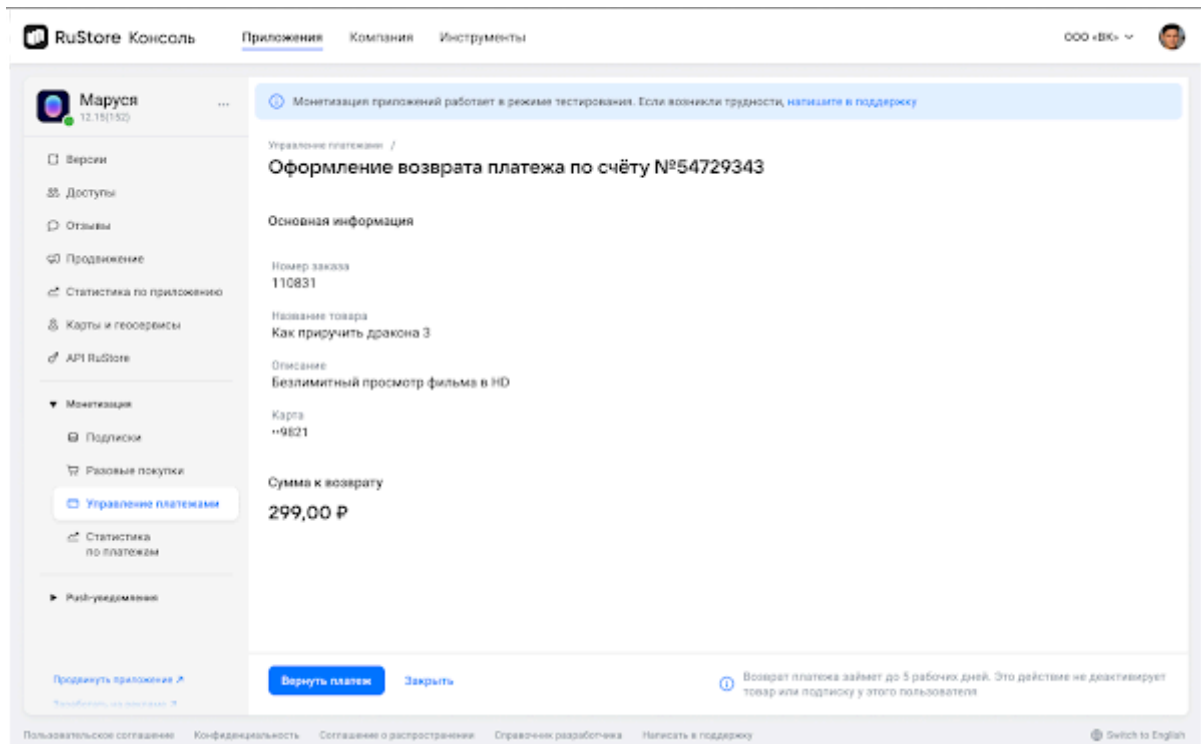
Итого сумма платежа
299,00 Р

[Заккрыть](#) [Вернуть платеж](#)

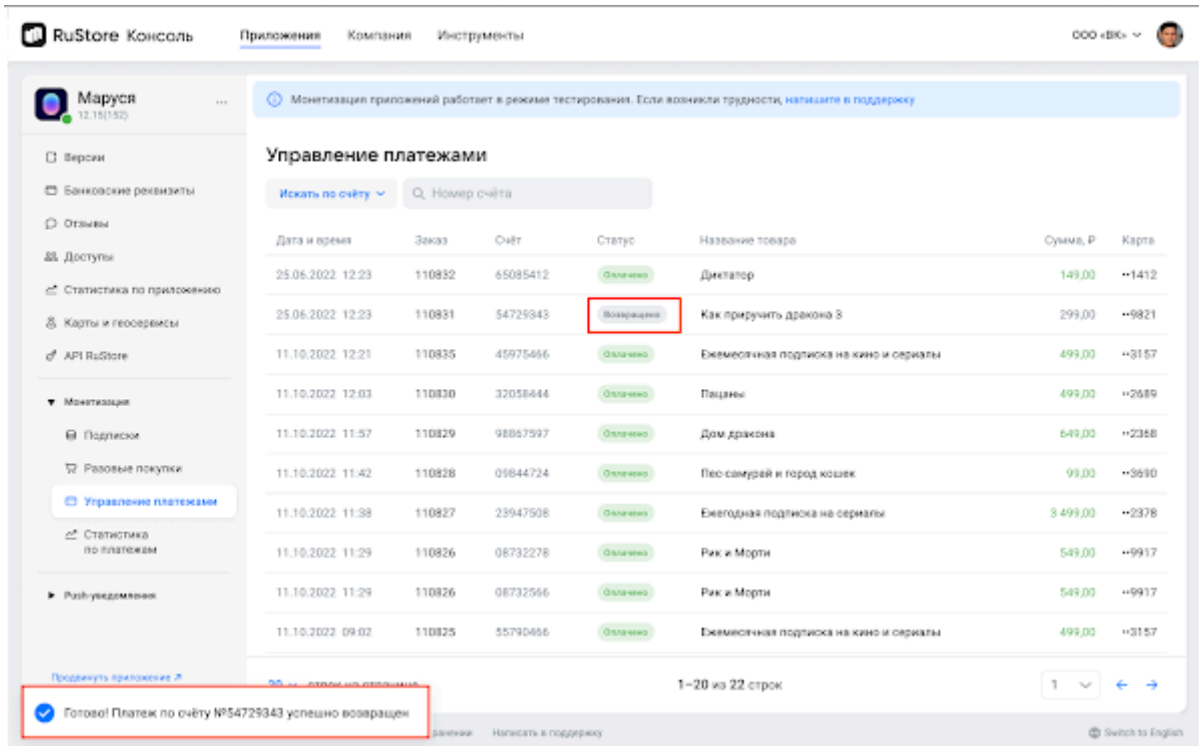
Review the information in the displayed window and select **Return payment** once more.

caution

Payment return does not deactivate the app or subscription.



After payment return is processed, the purchase status is changed to **Refunded**. The previously paid amount will be returned to the buyer's account.



App Statistics

RuStore Console features various statistics sections, providing insights into crucial data such as:

- app page views and downloads,
- amount of profit received and returns, and
- number of push notifications sent to users.

These statistics will help you gather essential information to boost your app's functionality and make sure it is user-friendly. Additionally, it allows you to assess sales dynamics and make informed decisions, i.e. to adjust the monetization strategy if needed.

Downloads statistics

You can view your app statistics on RuStore Console.

Note. A user should be logged in to RuStore Console to view statistics.

How to view your app download statistics?

1. Open [RuStore Console](#).
2. Go to **Applications**.
3. Click App Statistics in the side menu.

The statistics tab displays the following items:

- A table with general app statistics, containing:
 - Total number of app views in the RuStore interface (Web and mobile app);
 - Total number of app downloads in the RuStore interface;
- A table with statistics on OS versions of RuStore users, containing:
 - RuStore user OS version name;
 - Number of app downloads on a device with a specific OS version;
 - Percentage of app downloads on a device with a specific OS version;
- A table with statistics on device models of RuStore users, containing:
 - RuStore user device model name;
 - Number of app downloads on a particular device model;
 - Percentage of app downloads on a particular device model;

You can sort table elements with statistics by OS versions and device models whenever so required:

- Click on the sort icon once – all elements will be sorted from highest to lowest value;
- Click on the sort icon twice – all elements will be sorted from the lowest to the highest value;
- Click on the sort icon three times – the list of items will return to its default state.

To use sorting more efficiently, you can use several filters at the same time.

RuStore Консоль [Приложения](#) Компания 000 «ВК»

Маруся 12.18(152)

- Информация
- Версии
- Доступы
- Отзывы
- Продвижение
- Статистика по приложению**
- Карты и геосервисы [Beta](#)
- API RuStore [Beta](#)
- Монетизация [Beta](#)
- Push-уведомления [Beta](#)

Продвигать приложение [↗](#)
Заработать на рекламе [↗](#)

Пользовательское соглашение | Конфиденциальность | Соглашение о распространении | Написать в поддержку | [Switch to English](#)

Статистика по приложению с момента прохождения модерации первой версии

Просмотры и установки

Период	Уникальные пользователи на странице приложения в вебе	Уникальные пользователи на экране в приложении	Пользователей установило
За всё время	367	836	518

По модели устройства

Модель устройства ↑	Пользователей установило ↑	Процент от всех установок ↓
Samsung Galaxy S22 Plus ...	151	30%
Samsung Galaxy A53	110	20%
Google Pixel 6	90	18%
OnePlus 10 Pro	60	12%
Samsung Galaxy S22 Ultra	51	10%

[Смотреть все](#)

По версии OS Android

Версия ОС ↑	Пользователей установило ↑	Процент от всех установок ↓
12.1 Snow Cone	265	50%
11 Red Velvet Cake	121	24%
10 Queen Cake	60	12%
8.1 Oreo	41	8%
6 Marshmallow	30	6%

Monetization statistics

You can also view financial data to evaluate the dynamics of sales, including specific content or subscriptions. Revenue data is based on transaction volumes.

General requirements

To view monetization statistics, make sure to comply with the following conditions:

- at least one monetization type has been enabled and transactions have been made previously.
- the user is entitled to view statistics;
- the user must be logged in to RuStore Console.

How to view monetization statistics?

1. Open [RuStore Console](#).
2. Go to **Applications**.
3. Then go to **Monetization** → **Monetization Statistics**.

On the Monetization Statistics page you can view total income and returns from all sources. Sources refer to the type of monetization option enabled for the app:

1. Paid applications: income from paid app downloads;
2. In-app products: income from transactions with in-app products;
3. Subscriptions: income from recurring payments for using a subscription.

For returns and revenue statistics, the following data type is displayed:

- Amount, indicating currency;
- Number of transactions;
- Percentage of income.

Displayed if several monetization options enabled (for example, subscriptions + inn_app payments)

- Return percentage
- Period refers to a period of time for which you can view statistics. Available values:
 - Last day from 00:00 to 00:00;
 - Last 7 days including the current day;
 - Last 30 days including the current day;
 - All-time statistics = from the first transaction until the current day, inclusive.
- 2. The report is generated at the end of the selected period.



Маруся

12.15(152)

Версия

Доступы

Отзывы

Продвижение

Статистика по приложению

Карты и геосервисы [Beta](#)AR RuStore [Beta](#)Мониторинг [Beta](#)

Подписки

Разовые покупки

Управление платежами

Статистика по платежамРезультаты [Beta](#)

Проекты

Статистика по РИМ-устройствам

Мониторинг приложений работает в режиме тестирования. Если возникли трудности, напишите в поддержку

Статистика по платежам

Доходы

Транзакции

Тип	Вчера	7 дней	30 дней	Всё время
Подписки	7	14	31	52
Разовые покупки	4	8	16	28
Всего	22	22	47	80

Данные отображаются за период с 00:00 по 23:59 предыдущего дня

Проценты от дохода

Тип	Вчера	7 дней	30 дней	Всё время
Подписки	13%	27%	60%	67%
Разовые покупки	87%	73%	40%	33%

Суммы, ₽

Тип	Вчера	7 дней	30 дней	Всё время
Подписки	1 799,00	2 400,00	4 199,00	8 398,00
Разовые покупки	1 400,00	1 800,00	3 200,00	6 400,00
Всего	3 199,00	4 200,00	7 399,00	14 798,00

Возвраты

Транзакции

Тип	Вчера	7 дней	30 дней	Всё время
Подписки	0	0	0	0
Разовые покупки	0	2	2	2
Всего	0	2	2	2

Проценты возвратов от общего дохода

Тип	Вчера	7 дней	30 дней	Всё время
Подписки	0%	0%	0%	0%
Разовые покупки	0%	4%	4%	4%

Суммы, ₽

Тип	Вчера	7 дней	30 дней	Всё время
Подписки	0	0	0	0
Разовые покупки	0	499,00	499,00	499,00
Всего	0	499,00	499,00	499,00

[Продвигать приложение](#)[Заработать на рекламе](#)

Push Notifications Statistics

Here you can view push notifications statistics for all published applications.

Statistics are calculated based on data for each app that has push notifications configured.

How to view push notifications statistics?

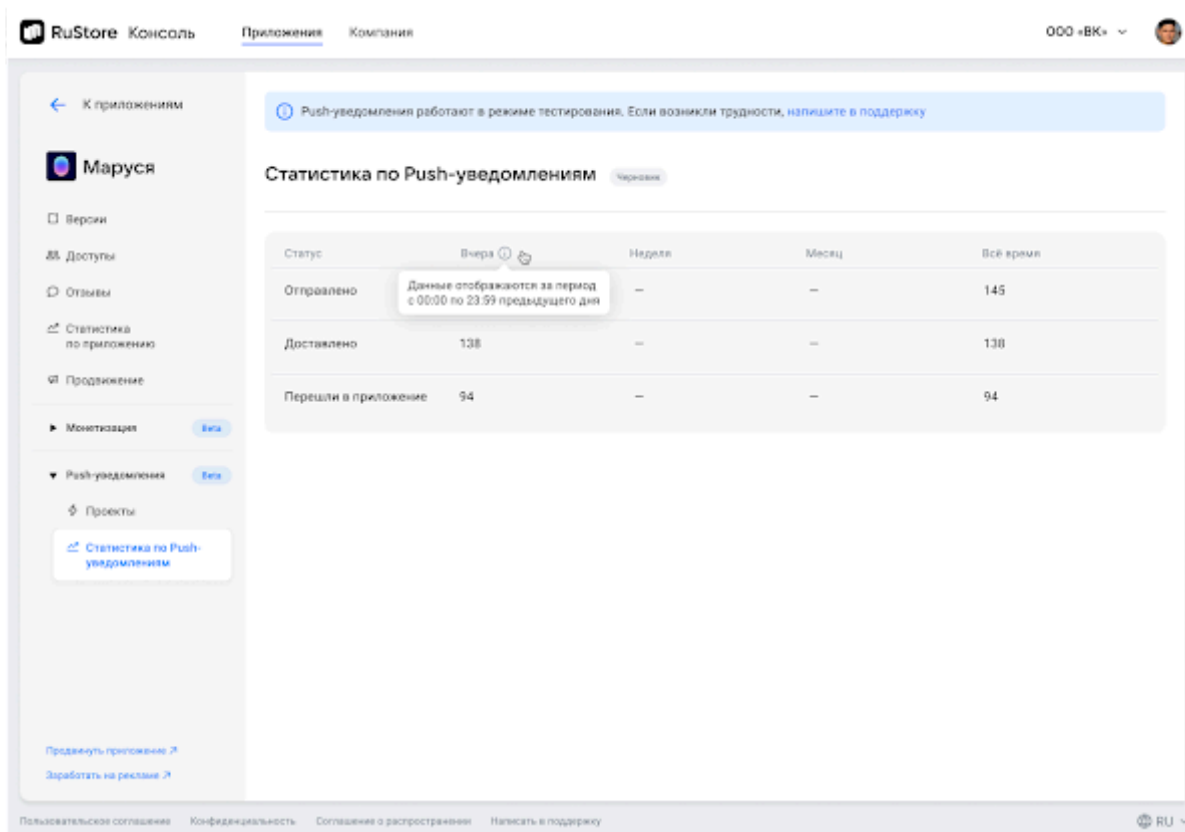
1. Open [RuStore Console](#).
2. Go to **Applications**.
3. Then go to **Push Notifications** → **Push Notifications Statistics**.

Statistics on push notifications are displayed one day after the publication of a new app version.

The statistics page displays the number of:

- all sent push notifications;
- delivered push notifications;
- users who accessed the app by clicking on a push notification.

This data is collected to be viewed by time period: daily, weekly, monthly, or all time.



Monetization report

Sample RuStore Software Distribution Agreement Completion Report

Companies that have enabled monetization receive a monthly monetization report by mail. [Report example.](#)

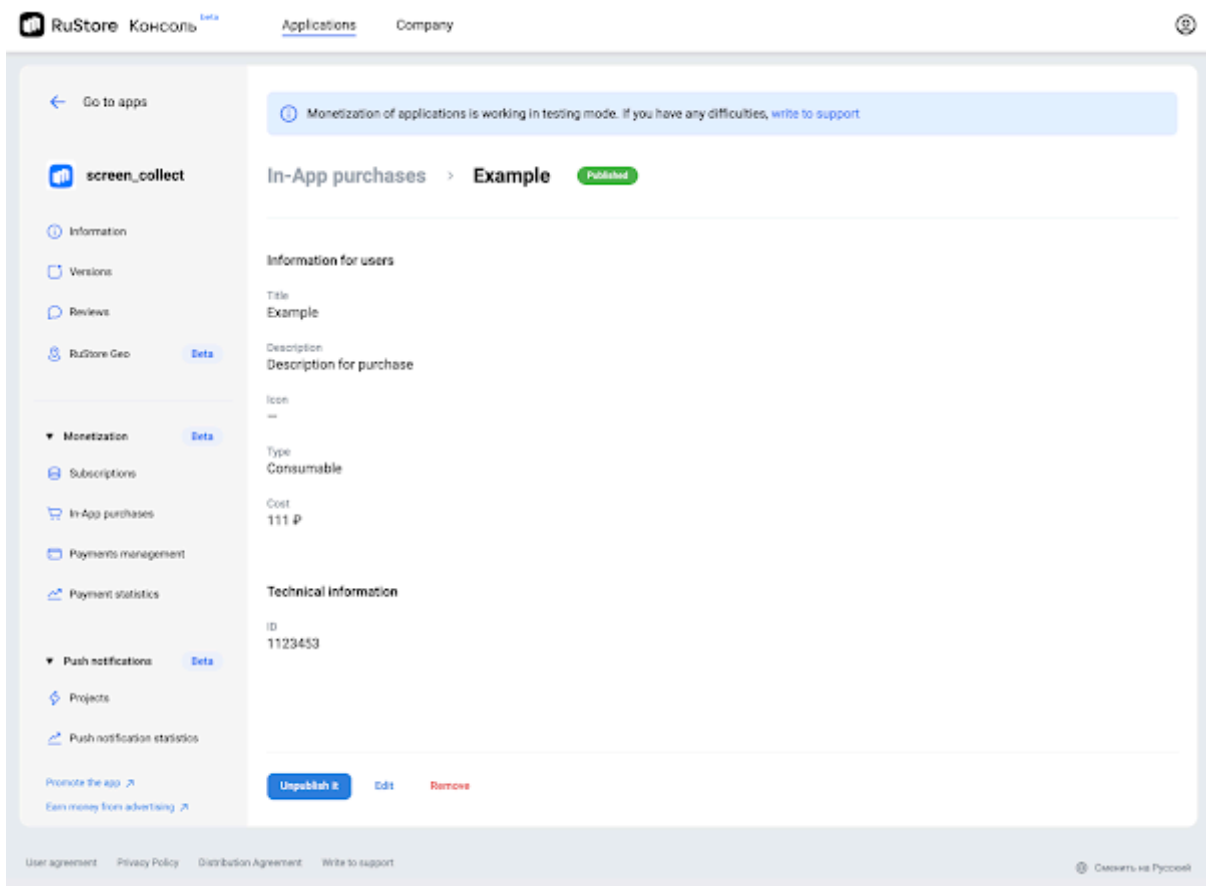
Report fields description

Field	Description
Company's debt as of the beginning of the reporting period, RUB	Payments accepted from your app users, but not received as calculated at the end of the previous reporting period
Debt payable to the Company as of the beginning of the reporting period, RUB	Always 0 because we compensate our agents for your debt to them in accordance with the settlement scheme. If you have a debt to the Company as of the previous report, we will send you an invoice to be paid within 3 business days from the date of receipt

Total amount of payments received from Users within the reporting period (including refunds), RUB	The amount received from your app users during the reporting period net of refunds made in the reporting period
Amount transferred within the reporting period, RUB	The amount of your revenue transferred to your bank account within the reporting period (user payments net of refunds and commissions payable to RuStore and agents)
Commissions payable by the Company within the reporting period, RUB	Commissions payable to RuStore and agents
Company's debt at the end of the reporting period, RUB	Payments accepted from your app users, but not received as of the end of the current reporting period
Amount payable at the end of the reporting period, RUB	Amount payable to the RuStore, if the refunds exceeded the amount of payments received from users as of the reporting date (revenue payment date). In accordance with the payment scheme, we will compensate the agents for the amount of your debt and send you an invoice to be paid within 3 business days from the date of receipt

How to create a paid in-app product

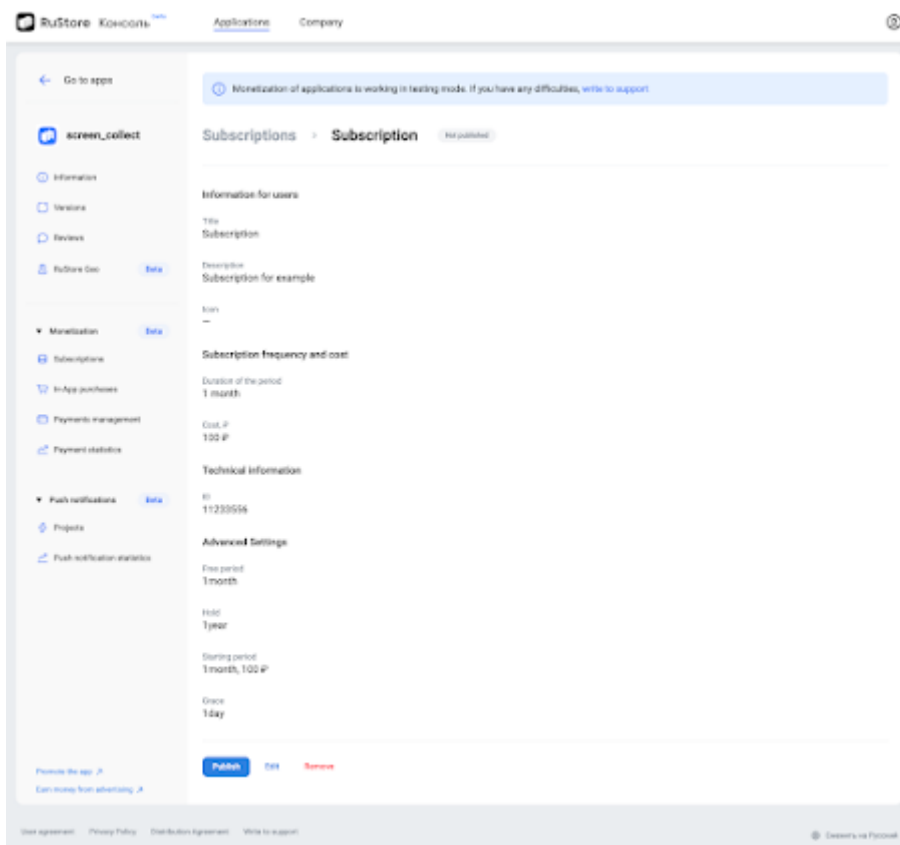
1. Open the [RuStore Console](#).
2. Click the "Applications" tab at the top of the screen.
3. Select "One-Time Purchases" in the left side menu of the "Monetization" section.
4. Click "New Item" in the center of your screen.
5. Fill in the user details and technical specifications about the product.
6. Click "Create" in the lower left corner of the screen.
7. Click "Publish". Click "Edit" to edit the product info at this step.
8. Once there, click "Publish" in the pop-up window.



How to create an app subscription

Here you can create a subscription so that users can buy a subscription to your application on the RuStore store:

1. Open the [RuStore Console](#).
2. Go to the "Applications" tab at the top of the screen.
3. Select "Subscriptions" in the left side menu of the "Monetization" section.
4. Click "New Subscription" on the middle of your screen.
5. Fill in user details and technical specifications about the subscription, as well as additional settings whenever so requested.
6. Click "Create" in the lower-left corner of the screen.
7. Click "Publish". Click "Edit" to edit the product info at this step.
8. Once there, click "Publish" in the pop-up window.



Recurring payments are debited within 8 hours before subscription ends. For example, if a user subscribed at 16:00, then subsequent regular payments are debited between 08:00 and 16:00. The terms of debiting should be regulated by the Developer documentation.

Ads and promotion

How to add a "Download from the RuStore" button

You can use the "Download from RuStore" button to tell users about your app.

Two-color logo version:

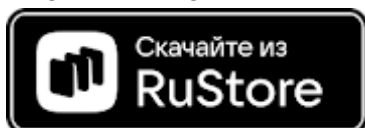


Download: [SVG](#), [PNG](#)



Download: [SVG](#), [PNG](#)

Single color logo version:

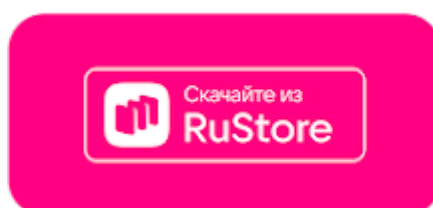
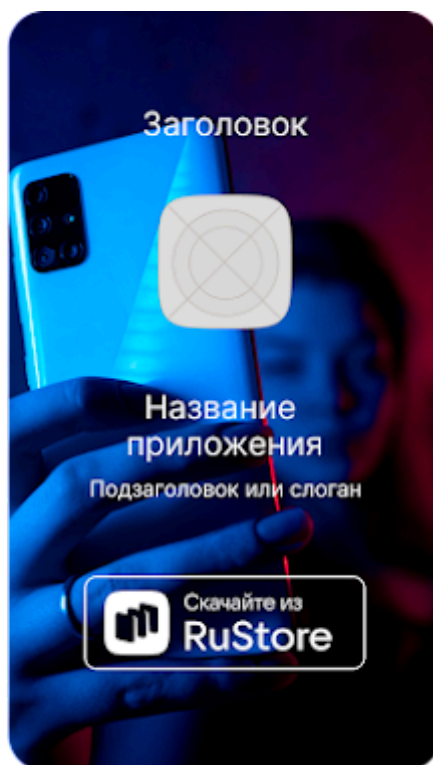
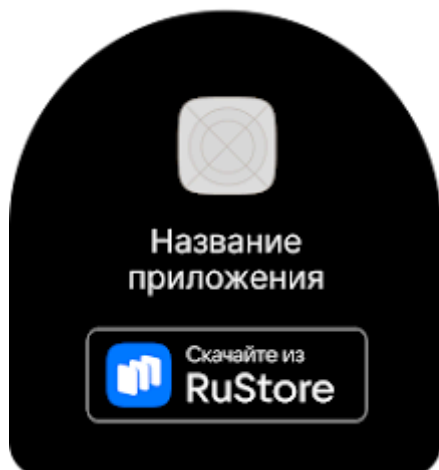
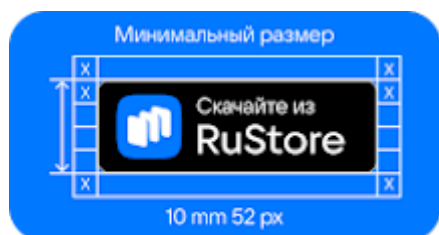


Download: [SVG](#), [PNG](#)



Download: [SVG](#), [PNG](#)

Use case:



Follow a few simple steps:

1. Download the "Download from the RuStore" logo.
2. Upload it to your server.
3. Find your application on the RuStore website and copy the bundle_id from the link.

Link example: https://trk.mail.ru/c/me10h4?bundle_id=com.vkontakte.android

4. Add code to the website:

```
<a href="#Link to your application from point 3" target="_blank">  
  
</a>
```

The recommended image size is 188 x 63 px.

Requests for in-app events

In- app events are a means to motivate users to install your app, for example: sales (for online stores), special offers or in-game events, and other limited offers.

RuStore Console enables you to submit a request for an event that will be displayed in your app. You can submit an event request only for an app with a published version.

To manage event requests log in to RuStore Console.

The processing time for submitted requests is up to 14 days. If we don't get in touch with you within 14 days, your request was not approved by RuStore editors.

The maximum amount of active event requests is 5.

Create request

To create an event request, follow these steps.

- In the navigation menu on the left side select Promotion > Event requests. Depending on whether you already have created requests or not the following page will open.
- No requests
- Submitted requests
- Click or tap .

The event creation form will be displayed (see the image below).

- Provide necessary data (please, refer to the table below).

Parameter	Description
Event name	Enter a name for your event (up to 34 characters). Use a short name that captures the essence of your event.
Event type	Select from the list: <ul style="list-style-type: none">● New season;● Special offer;● Tournament;● Sale;● Major update;● Special event;● Premiere;● Live broadcast.
Start date/Time	Start date and time of the event. Let the user seize the opportunity to take advantage of the offer instantly.

End date/Time	<p>End date and time of the event. Must be later than the current date and time.</p> <ul style="list-style-type: none"> • Maximum event duration is 90 days. • Minimum event duration is 24 hours. <p>The duration of the event shouldn't be too long so that the user does not postpone the opportunity.</p>
Brief summary	<p>Enter a brief description of the event — up to 32 characters.</p> <ul style="list-style-type: none"> • In this brief description we recommend you to call to action (e.g.: “Buy with 90% discount”/“Only in RuStore” etc.). • Be polite to users to be consistent with the tone of voice of RuStore. • If by taking part in an event the user gets special usage privileges, be creative to emphasize that — make it a call to action.
Description	<p>Provide a detailed event description — up to 448 characters.</p> <ul style="list-style-type: none"> • The description, besides the event details, should contain the context of the corresponding event. • The description should be spacious and include key advantages of the event or conditions of participation (how to activate promo code/in which section to look for a gift box/where to get the registration gift, etc.).
Cover	<p>Use the link to set the path to the file or drag and drop the file in the designated area:</p> <ul style="list-style-type: none"> • format: JPG or PNG; • size: up to 3 MB; • aspect ratio: 4:3; • resolution — 2880x2160 px. <p>The image must be of good quality. Instruction for designers.</p>
Cover color	<p>Specify a HEX color code for automatic gradient for the cover. The color matching the specified code will be displayed in the square below.</p> <p>We recommend you to specify the code that matches the color of your image.</p>

Text color	Specify text color — white or black. Please, make sure that the text is easily readable against background: if the background is black, then, select white for text, if the background is white, then, the text must be black.
Request processing priority	Select the processing priority for your request: <ul style="list-style-type: none"> • Standard — set by default standard processing priority; • High — high processing priority. High priority can be set only for 1 submitted event request for an upcoming event.
Phone number	Specify your contact phone number in the following format +79876543210 .
E-mail	Please, specify your email address so that we could contact you if your request is approved or to clarify possible questions.
Comment for moderator	If there is any additional information that has be taken into account regarding your event, specify it in this field.

- Click or tap .

After sending the request, on the specified address you will receive an email confirming that the request is received by the RuStore editors. If there's no email, please, check your SPAM folder.

An approved event will be visible on your app card at the specified time period. Also, your event may be included in «Interesting», «Games», or «Apps» selection if picked by the RuStore team.

View request

To view a created request, follow these steps.

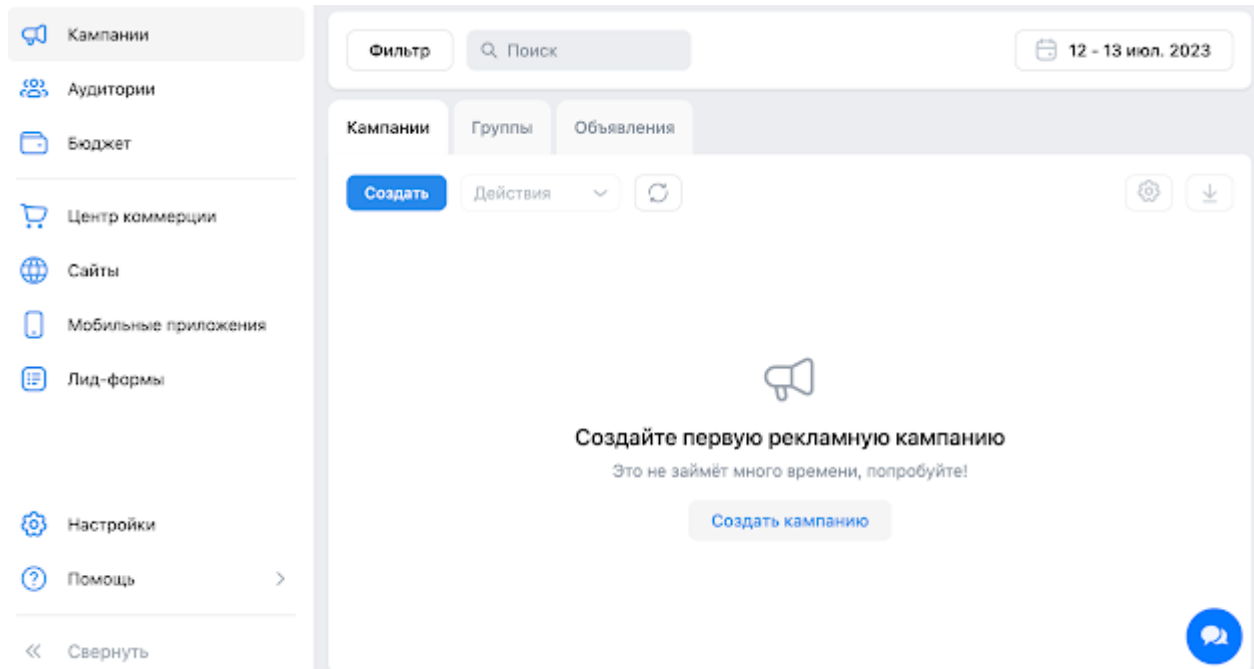
- From the navigation menu on the left, select Promotion > Events.
The list of the created requests will be displayed.
- Select the request you need.
The selected request page with the detailed information will be displayed.

VK Ads on RuStore

Now developers have the opportunity to promote their applications in RuStore with the help of advertising.

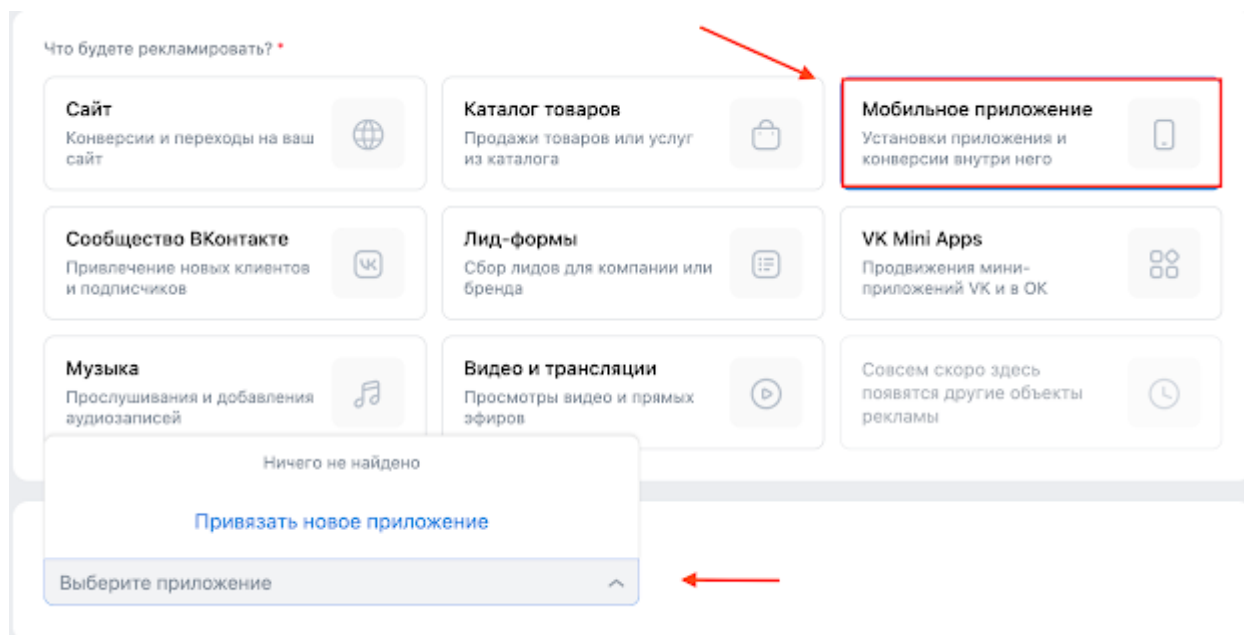
How to set up an advertising campaign via VK Ads

1. Open the [VK Ads](#) in your browser and log in.
2. Go to the **Campaigns** section.
3. Click **Create Campaign**.



4. Click **Mobile App**. Select an advertised app from the list or link a new one through VK Ads.

To get started for the first time, you need to add an app and set up integration with the tracker. For more details, see *How to add an application through VK Advertising* below.

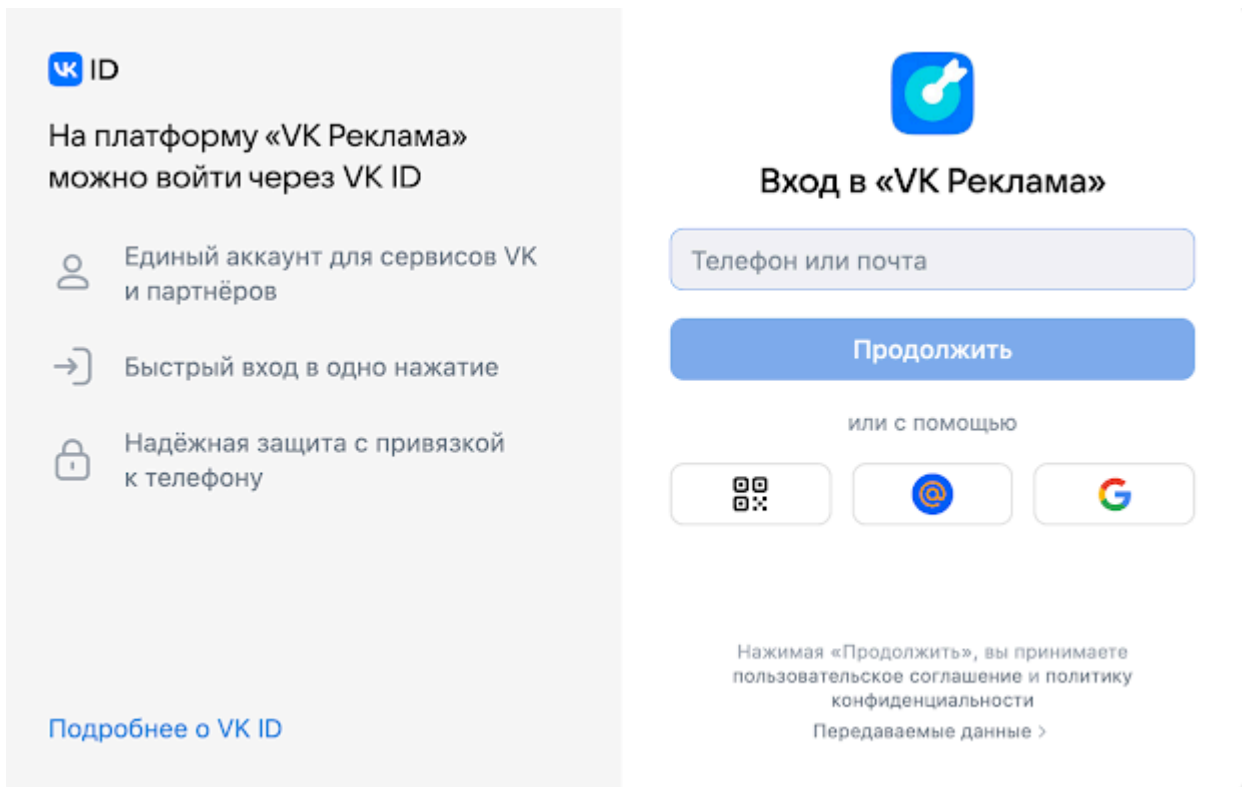


5. Set up an advertising campaign.

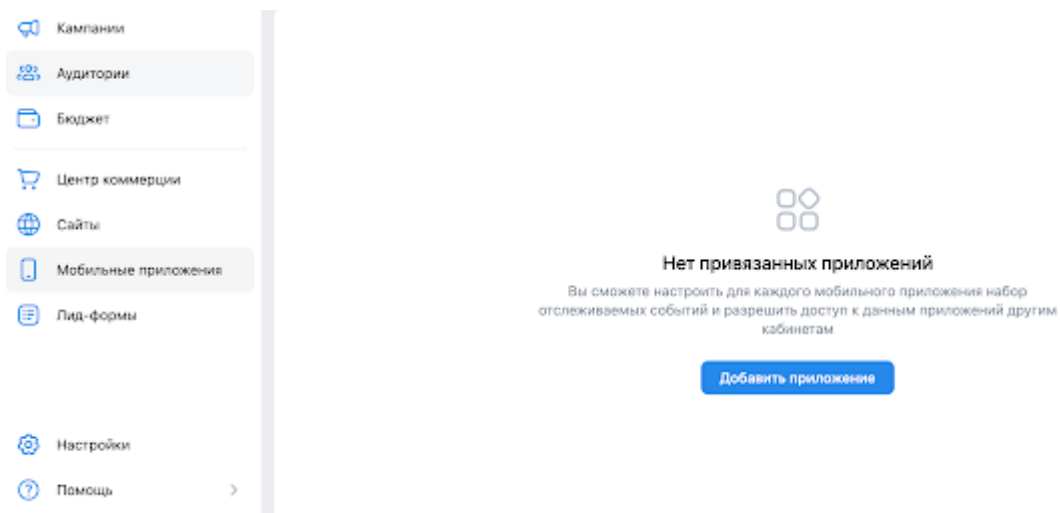
VK Ads provides many tools and technologies so that you can customize your advertising as accurately as possible. See the [Guide](#) for more details on setting up an advertising campaign.

How to add an app via VK Ads

1. Open [VK Ads](#).



2. Log in using one of available methods or [create a new account](#).
3. Go to **Mobile Applications** and click **Add Application**.



4. The add application form will open. Enter the link to the app in RuStore and click **Add**.

To find the link to your app, open the web version of the [RuStore storefront](#).

The application status will change to **Checking access**.

5. Start setting up integration with the tracker.

RuStore also allows launching advertising campaigns on VK Ads via direct links without any third-party tracking systems. To launch campaigns from RuStore, read the [Guide](#).

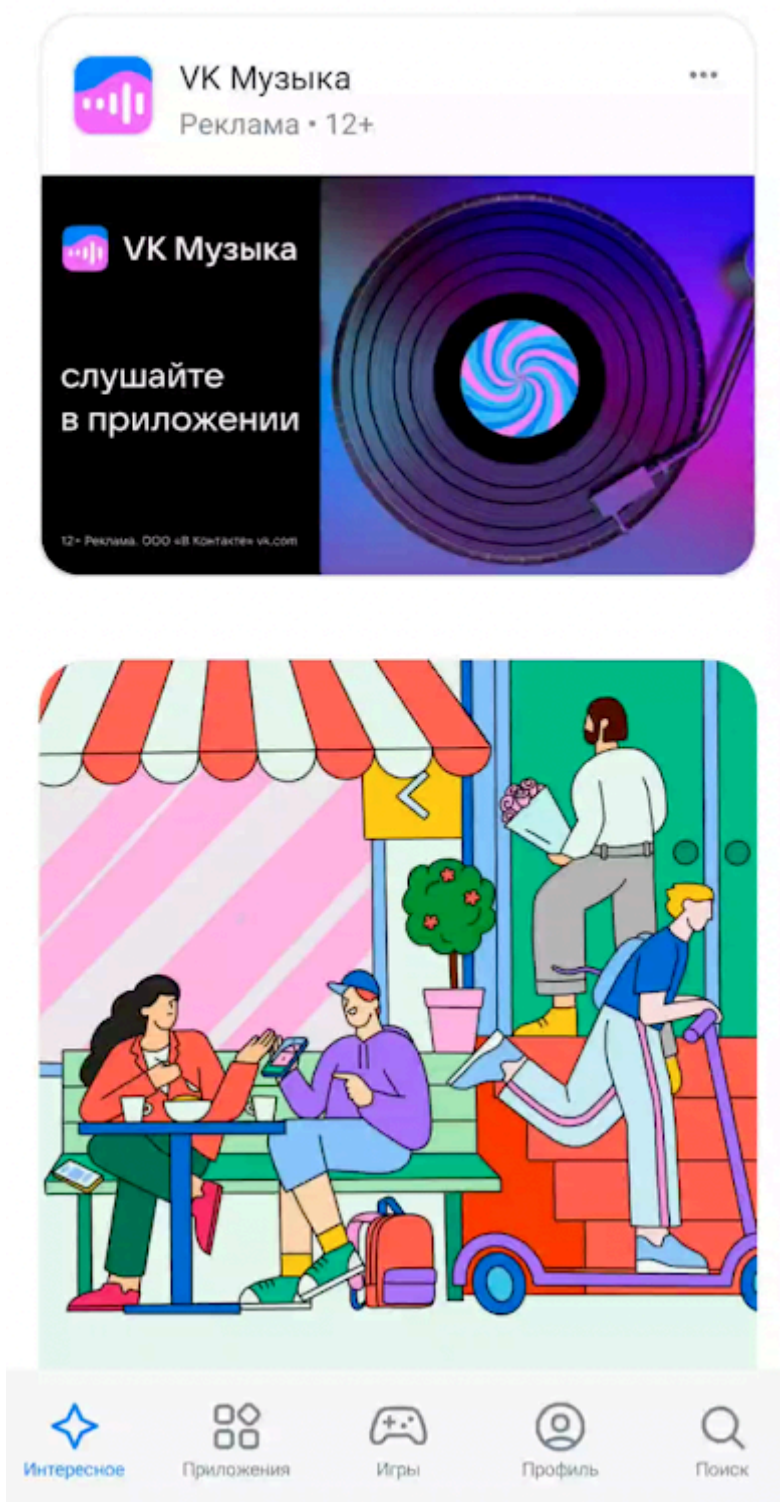
Where your ads can appear

Advertising applications are available in **Featuring**. At that, ads are displayed to both authorized and unauthorized users.

Featured Ads on RuStore:

An advertising banner appears when a user goes to the **Featuring** section. To go to the application page, the user can click on the banner.

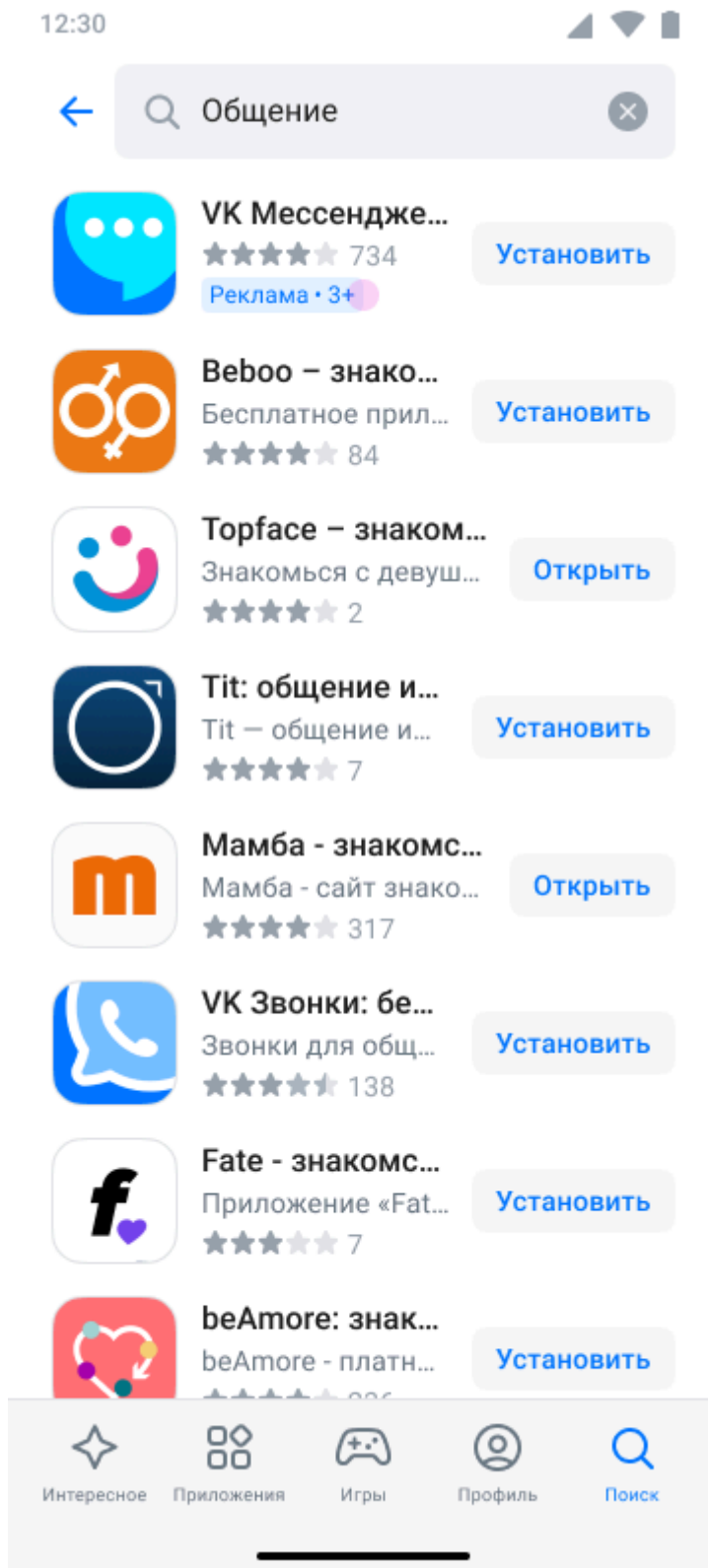
Интересное



Ads in Search results:

The ads will also appear at the top of the list in search results.

By clicking on **Install**, the advertising application starts downloading in the same way as other applications from the search results.



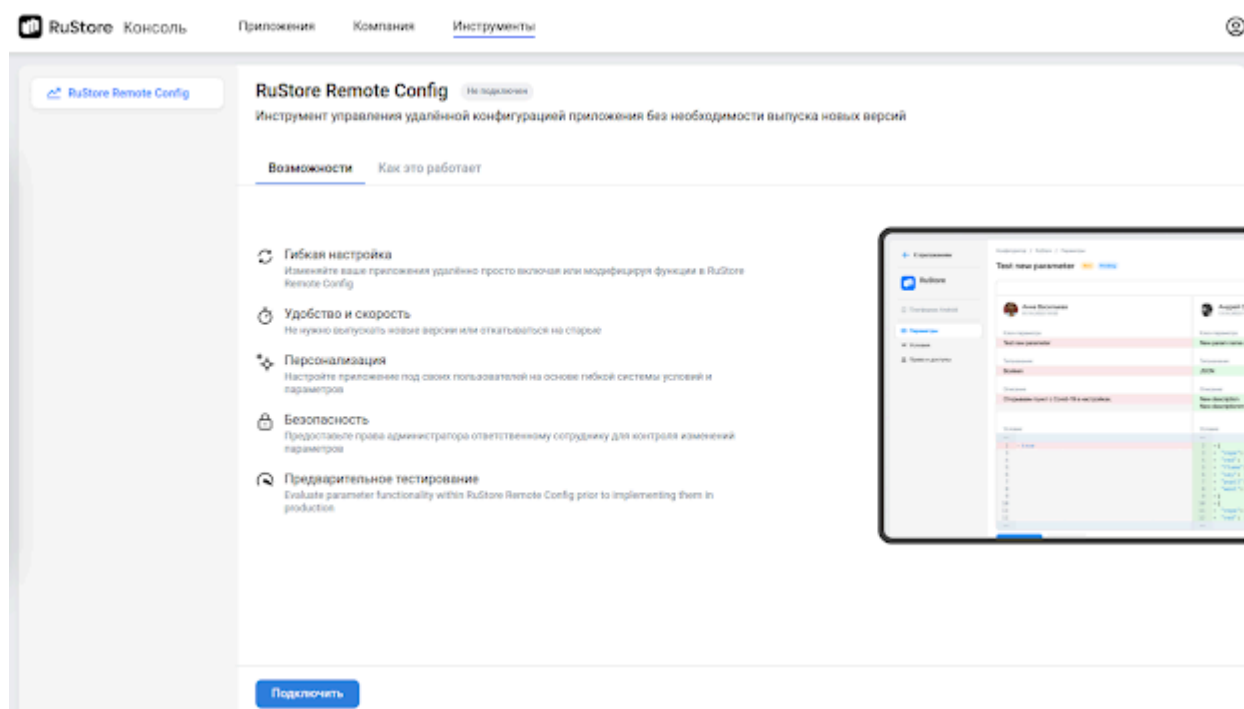
Tools

General information

Use “Tools” in [RuStore Console](#) to connect developer tools to your apps.

Tools provide developers with additional options for app management, error detection, and integration with various services.

Only users with owner permissions can connect a tool.



Connecting a tool

1. Select the “Tools” tab.
2. Select an instrument of your interest from the side menu and click “Connect”. The system will display the following result message.

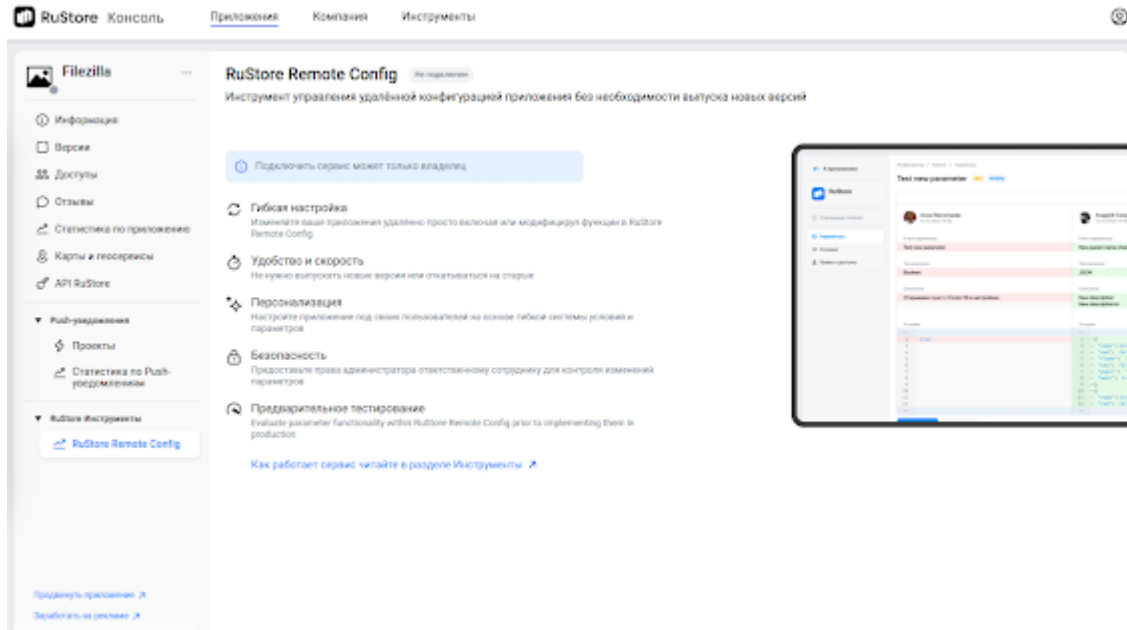
If an error occurs, contact our technical support.

From the main console page the selected tool will be connected to all of your apps that have at least one created version in [RuStore Console](#).

You can also connect a tool from the side menu of an app card. In this case the tool will be connected only to the selected app.

After connecting the tool you'll be able to see all connected apps in the tools section and switch directly to the app interface.

On connecting, an owner's account is created in a particular service in RuStore. To configure access for other employees of your organization after connecting a tool, switch to the service and configure access control in the tool interface.



After connecting the tool you'll be able to see all connected apps in the tools section and switch directly to the app interface.

On connecting, an owner's account is created in a particular service in RuStore. To configure access for other employees of your organization after connecting a tool, switch to the service and configure access control in the tool interface.

RuStore Remote Config

General information

RuStore Remote Config is a tool for managing mobile app configuration. It allows to implement feature toggles (or feature switchers) and manage them from a convenient GUI.

Info

You need a registered developer account to work with RuStore Remote Config.

Quickstart

[SDK installation and app configuration;](#)

[Creating conditions for enabling parameters;](#)

[Description of the Remote Config parameters.](#)

Additional information

[User permissions](#);

[How the service works](#).

How the service works

On creation of a new app its code needs to be configured in the way that will allow to change its parameters using the config settings. That is why Remote Config will not be able to work with just any app: you have to adapt your code for working with RuStore Remote Config.

On app start SDK sends a request to RuStore Remote Config and receives a response with a configuration in JSON that contains a set of parameters. Together with the configuration (parameters that define how the app behaves), the response includes the configuration hash. The app stores this hash and includes it in further requests to Remote Config as one of the parameters.

Remote Config compares the received hash with the hash of the configuration that needs to be returned for this request. If the hashes match, the service replies with the "not modified" response instead of the configuration. This is done to avoid sending unmodified configuration to the client.

After receiving the configuration the modified interface and new features obtained from RuStore Remote Config are added on the device.

Note

The app applies the received configuration only after restart. The users will not see the changes in real time.

SDK and app configuration

General information

SDK Remote Config is a cloud service that allows to change the behavior and look of your application without requiring the user to update the app. SDK encapsulates the request for configuration from the server, caching, and background update. It has a handy API for retrieving data.

Key features

SDK allows to select the most suitable mechanism for configuration update.

With it you can specify the percent of the app users that will receive the configuration.

You can pass additional information to narrow the list of users that will receive a particular configuration. You can even create configurations for particular users.

Has a set of SDK performance callbacks that you can use for analysis.

Has minimum external dependencies.

Connecting to project

Connect repository (see below).

build.gradle

```
repositories {
    maven {
        url = uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
    }
}
```

Connecting the dependency

Add the following code to your configuration file to add the dependency.

build.gradle

```
dependencies {
    implementation( "ru.rustore.sdk:remoteconfig:0.0.2" )
}
```

How to create and initialize RemoteConfigClient

RemoteConfigClient initialization must be done during Application.onCreate() as by the background sync start the SDK must be already initialized.

Creating RemoteConfigClient

```
val remoteConfigClient = RemoteConfigClientBuilder(appId = AppId("your_app_id"), context =
applicationContext).build()
```

Using RemoteConfigClientBuilder you can install [additional parameters](#) that can be used to retrieve a particular configuration.

Recalling RemoteConfigClientBuilder.build() will throw the [RemoteConfigClientAlreadyExist](#) error.****

On calling the RemoteConfigClientBuilder.build() method a RemoteConfigClient instance is created and a singleton is created. You can retrieve the created instance using the following method.

```
RemoteConfigClient.instance
```

Access through a static variable before the RemoteConfigClient instance is created via RemoteConfigClientBuilder.build() will throw the [RemoteConfigClientNotCreated](#) error.

RemoteConfigClientBuilder optional parameters

Parameter	Description
OsVersion	Condition in the configuration tool: Os Version Allows to compare

Parameter	Description
DeviceModel	<p>OsVersion against the value set in the interface. By default OsVersion is not passed, in this case the default configuration will be returned.</p> <p>Condition in the configuration tool: Device Model A DeviceModel Allows to compare DeviceModel against the value set in the interface. By default DeviceModel is not passed, in this case the default configuration will be returned.</p>
Language	<p>Condition in the configuration tool: Language Allows to compare Language against the value set in the interface. By default Language is not passed, In this case the default configuration will be returned. To pass Language, implement ConfigRequestParameterProvider.</p>
Account	<p>Condition in the configuration tool: Account Allows to compare account against the value set in the interface.</p> <p>Condition in the configuration tool: Account Percentile Allows to broadcast the configuration to a specified percent of users based on the account value.</p> <p>Condition in the configuration tool: Interval Account Percentile Allows to broadcast the configuration to a specified percent of users on a specified day based on the account value. To pass Account, implement ConfigRequestParameterProvider.</p>
DeviceId	<p>Condition in the configuration tool: DeviceID Allows to compare DeviceId against the value set in the interface.</p> <p>Condition in the configuration tool: DeviceID Percentile Allows to broadcast the configuration to a specified percent of devices based on a on the DeviceId value</p> <p>Condition in the configuration tool: Interval DeviceID Percentile Allows to broadcast the configuration to a specified percent of devices on a specified day based on the DeviceId value.</p>
AppVersion	<p>Condition in the configuration tool: App Version Allows to compare AppVersion against the value set in the interface.</p>
Environment	<p>Condition in the configuration tool: App Environment Allows to compare Environment with the value set in the interface Environment can take the following values: Alpha, Beta, Release. It is a convenient parameter for testing of the configuration on different app builds.</p>
AppBuild	<p>Condition in the configuration tool: App Build Allows to compare AppBuild against the value set in the interface.</p>

UpdateBehaviour

UpdateBehaviour — this parameter defines SDK behavior. The default value of the RemoteConfigClientBuilder instance upon creation is UpdateBehaviour.Default with a 15 minute sync interval.

You can set a different SDK behavior the following way.

```
remoteConfigClientBuilder.setUpdateBehaviour(UpdateBehaviour.Actual)
```

Difference UpdateBehaviour

UpdateBehaviour	Description
UpdateBehaviour.Actual	<p>With this type of initialization, every configuration request is made by a request to the server. This update type ensures the configuration is up to date, however, the speed of the configuration retrieval will depend on the network speed.</p> <p>This type of initialization cancels the background update.</p>
UpdateBehaviour.Default(minSyncInterval)	<p>With this type of initialization, the configuration request is made from the local store that is updated at specified intervals.</p> <p>If it is the first initialization (the local store is empty), a request to the server is made—the duration of this request depends on the network speed. Configuration access will wait for the initialization to complete.</p> <p>This type of configuration does not guarantee that during the process lifetime configuration retrieval will return the same result, as there can be configuration sync in the background.</p> <p>This initialization type starts background update.</p>
UpdateBehaviour.Snapshot(minSyncInterval)	<p>With this type of initialization, the configuration request is made from the local inMemory storage. inMemory-storage will receive the result of the request from the permanent store and save it until the end of the process lifetime.</p> <p>If it is the first initialization (the local store is empty), request to the server is made—the duration of this request depends on the network speed. Configuration access will wait until the initialization in complete.</p> <p>This type of configuration ensures that during the process lifetime configuration retrieval will return the same result.</p> <p>This type of initialization starts the background update process.</p>

ConfigRequestParameterProvider

ConfigRequestParameterProvider implementation allows to dynamically pass parameters to the server for configuration sync.

Supported parameters: Language and Account.

```
class ConfigRequestParameterProviderImpl : ConfigRequestParameterProvider {  
    override fun getConfigRequestParameter(): ConfigRequestParameter =  
        ConfigRequestParameter(  
            language = Language(Locale.getDefault().language),  
            account = Session.id,  
        )  
}
```

Implementation of the method must be fast and without time-consuming operations . If the getConfigRequestParameter method takes too long to execute, it may lead to high latency of the configuration retrieval.

The method call will be done on the RemoteConfig SDK threads.

Implement ConfigRequestParameterProvider the following way.

```
val provider = ConfigRequestParameterProviderImpl()  
remoteConfigClientBuilder.setConfigRequestParameterProvider(provider)
```

RemoteConfigClientEventListener

RemoteConfigClientEventListener implementation allows to receive callbacks on SDK performance, such as end of initialization and permanent storage updates. Listener's callbacks are called to MainThread.

Implement ConfigRequestParameterProvider the following way.

```
val listener = RemoteConfigClientEventListenerImpl()  
remoteConfigClientBuilder.setConfigRequestParameterProvider(provider)
```

RemoteConfigClient initialization

RemoteConfigClient initialization is done asynchronously—you can track the result via the [Task](#) class that is returned by the init() method or via the listener.

```
remoteConfigClient.init()
```

Retrieving configuration from RemoteConfigClient

Receiving configuration from RemoteConfigClient is done by calling the getRemoteConfig() method.

```
`remoteConfigClient.getRemoteConfig()`
```

If RemoteConfigClient hasn't been initialized, it will be initialized during the first getRemoteConfig() call—this may take some time. To faster retrieve the configuration, the RemoteConfigClient initialization must be done beforehand.

The getRemoteConfig() method returns [RemoteConfig](#) and is executed asynchronously. Task may fail. [SDK RemoteConfig errors](#)

Upon a successful Task completion, RemoteConfig will become available—this is a current set of all data received depending on the selected [update policy](#) and initialization parameters. With certain update policies the received configuration will depend not on the current initialization parameters but on the parameters that were used during the [background sync](#).

RemoteConfig class

A RemoteConfig instance is a current set of all data received based on the update policy selected during the initialization. This instance has a full set of keys that were received from the server based on the parameters set during the initialization.

Key availability check

```
remoteConfig.containsKey("my-key")
```

This method checks the availability of the key in the current RemoteConfig instance.

Receiving typed data

The RemoteConfig class has methods for fetching and typifying data.

```
getBoolean(key: String): Boolean  
getInt(key: String): Int  
getLong(key: String): Long  
getString(key: String): String  
getDouble(key: String): Double  
getFloat(key: String): Float
```

These methods may return a [RemoteConfigCastException](#) error if the data type doesn't match the selected method or the value corresponding to the passed key doesn't exist.

Configuration background sync

Configuration background sync is used with several types of update policies.

The result of a configuration background sync is the update of the current data in the permanent configuration store for further use depending on the selected update policy.

The minimum allowed interval between background syncs is 15 minutes.

For proper background sync performance, the SDK RemoteConfig initialization must be done using the Application.onCreate() method.

SDK RemoteConfig errors

All SDK RemoteConfig errors have the RemoteConfigException superclass.

RemoteConfigException descendants.

Error	Description
BackgroundConfigUpdateError	Thrown in case of an error during the background sync process.
FailedToReceiveRemoteConfig	Thrown in case of an error during the configuration retrieval call.
RemoteConfigCastException	Thrown in case of incorrect data retrieval by a key from the RemoteConfig class. The error may be due to impossibility of conversion to a type or absence of a value for the passed key.
RemoteConfigClientAlreadyExist	Thrown in case of creation of another RemoteConfigClient during the process lifetime.
RemoteConfigClientNotCreated	Thrown in case of an access to RemoteConfigClient through the static instance field before RemoteConfigClient creation.
RemoteConfigCommonException	General unexpected error.
RemoteConfigNetworkException	Thrown in case of a network error during the sync.

Conditions

Conditions define how your app will be updated. For example, “Android version 11 and later and the user interface language is “English”. A condition can be called “targeting” or “segment”. The creation of a condition itself doesn't affect anything, it must be applied to a parameter—[Remote Config parameters](#).

Conditions are used to apply parameters to a certain portion of the users. See **Conditions** for the full list of conditions. Conditions are also displayed in the drop-down list that is available during configuration of a parameter.

Warning

Do not edit conditions if it is unnecessary and unless you are sure that it will not affect the users. Especially, if conditions include parameters. It might negatively affect stable app performance.

Creating a new condition

To create a new condition click the **Add Condition** button.

Мое новое условие New Pending

Название
Мое новое условие

Описание
Описание условия

Атомарные условия

Тип	Операция	Значение
OS Version	Exactly matches	10

Подтвердить

Отклонить

Удалить

In the *Name* field set a name for the condition. in the *Description* field specify a descriptive comment.

Select **Atomic conditions**, that the condition being created consists of.

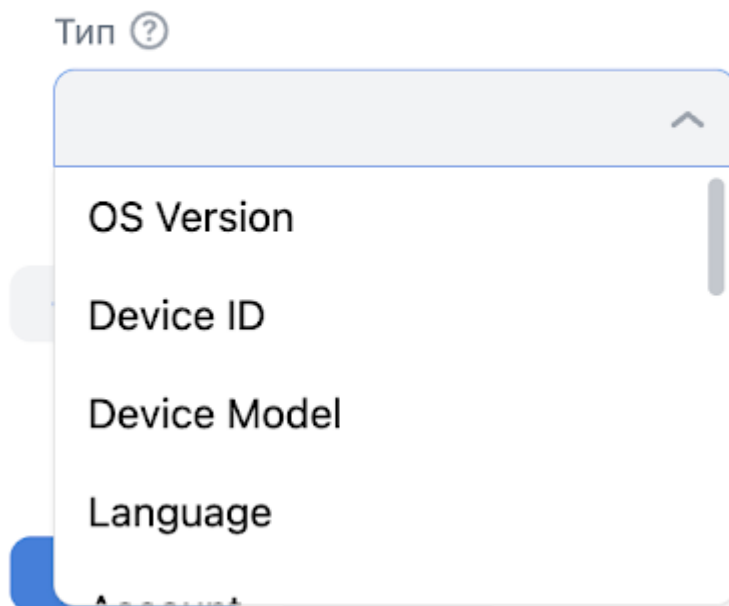
Note

All atomic conditions are joined by a logical AND operator. To trigger the condition, the user must match all of the atomic conditions.

Atomic conditions

Atomic conditions can be textual or numeric. The text field is filled in without quotes.

Атомарные условия



For textual conditions the following operators are available.

Exactly matches. Requires an exact match.

Example. The “OS Version exactly matches 9.0.0” will be matched by the `os-version=9.0.0` app query, but not by `os-version=9.0`.

Contains — contains the specified substring.

Does not contain — doesn't contain the specified substring.

Regexp — matches the specified regular expression.

Caution

When using regular expressions with Golang, check conditions [here](#) and during testing.

Presented in file — matches list elements from a file, maximum file size: 5 MB.

Not presented in file — matches all users that are not specified in an uploaded file.

For numeric conditions the following operators are available:

=;
!= (not equal);
>;
>=;
<;
<=.

List of atomic conditions

Atomic Condition	Description	Type	Usage examples and comments
OS Version	Operating system version	Textual + Numeric	OS Version <= 9.9 OS Version matches regex " [^] 1-8\$ ^9(.d(.d+))*\$"
Device Model	Model of the device	Textual	Samsung devices: Device Model Contains samsung
Language	device language	Textual	The Russian language on the device Language matches regex [^] ru
Account	User account,	Textual	Account exactly matches mrg.test@mail.ru In this case, if at least one of the connected accounts is mrg.test@mail.ru , the condition will be met.
App Version	Mobile application version	Textual	App Version exactly matches 1.5.3
App Environment	Alpha, beta, or release version	Available values: Alpha, Beta, Release.	This field's value is set during the SDK RuStore Remote Config initialization.
App Build Id	Mobile application build identifier	Numeric	Users that use builds 22563 and later App Build Id >= 22563
DeviceID Percentile	Percentile of DeviceID	Numeric	Used for publishing parameters per % of users. Percentile is a number from 0.00 to 99.99. The number is defined by a function of device-id. On every device-id RuStore Remote Config will return a value. There is an additional field: Salt — you

should us it whet it is necessary to apply the parameter to mutually exclusive groups of users.

Account Percentile	Same for accounts	Numeric	
DeviceID Interval Percentile	Calculated using DeviceID	Numeric	Allow to publish a feature in intervals. Example. You took 10% of the users, specified salt and a 7 day interval. The users that will see the update will be distributed across the whole interval: 1/7 of the users from the selected 10% will receive updates on every 7-th day.
Account Interval Percentile	Calculated based on Account	Numeric	Same as for DeviceID Interval Percentile .
Random Percent	A new percent value every time		
Show Time	Time of the config delivery		Specify the beginning, the end, and the time zone of the config delivery interval. Use it, for instance, if on Friday you need to schedule a launch for Monday or disable a feature on a certain date.

The Salt parameter

To distribute the users between different percentiles, the Salt parameter values must be different.

For example, if you want to enable new features for the same group of users while publishing using percentage, use the **deviceid percentile** atomic condition. If you need every update be available for different users, change the Salt parameter value.

Approving created conditions

A condition needs to be approved if it wasn't created the app owner or a user with approval permissions. [User roles](#).

Parameters

In RuStore Remote Config you can add parameters that will define your app's behavior. They can have different values depending on which [condition](#) is executed. During the app integration its code must be configured with SDK in a way, that this code contains the parameters that can be changed in the configuration.

Warning

Be very careful while editing parameters to avoid a negative effect on the users and app stability.

Creating a new parameter

Click the **Create parameter** button. A parameter creation screen will show up:

Приложения / null / Параметры

Новый параметр

Ключ параметра ?

Тип значения ?

Описание ?

Создать

Отмена

In the **Parameter key** field, specify its name. Add a comment or description.

Parameter types:

- Boolean — True/False;
- Numeric — digits and numbers;
- String — text (single line);
- Text — text (multiple lines);

JSON — sending configuration.


Warning

There can be only one type of a parameter value.

After a parameter type is selected, add to it previously created conditions

Each parameter can have different values based on different conditions. Conditions are calculated in the order defined in the parameter stored in RuStore Remote Config. For example, if the user matches condition “A”, further conditions are not taken into account. Therefore, using different combinations you can create conditions of any complexity.

Info

To move conditions, drag the  icon

When a parameter is created, it has the **New** label that indicates that the parameter is new. This parameter can be deleted without approval as it is in the draft state.

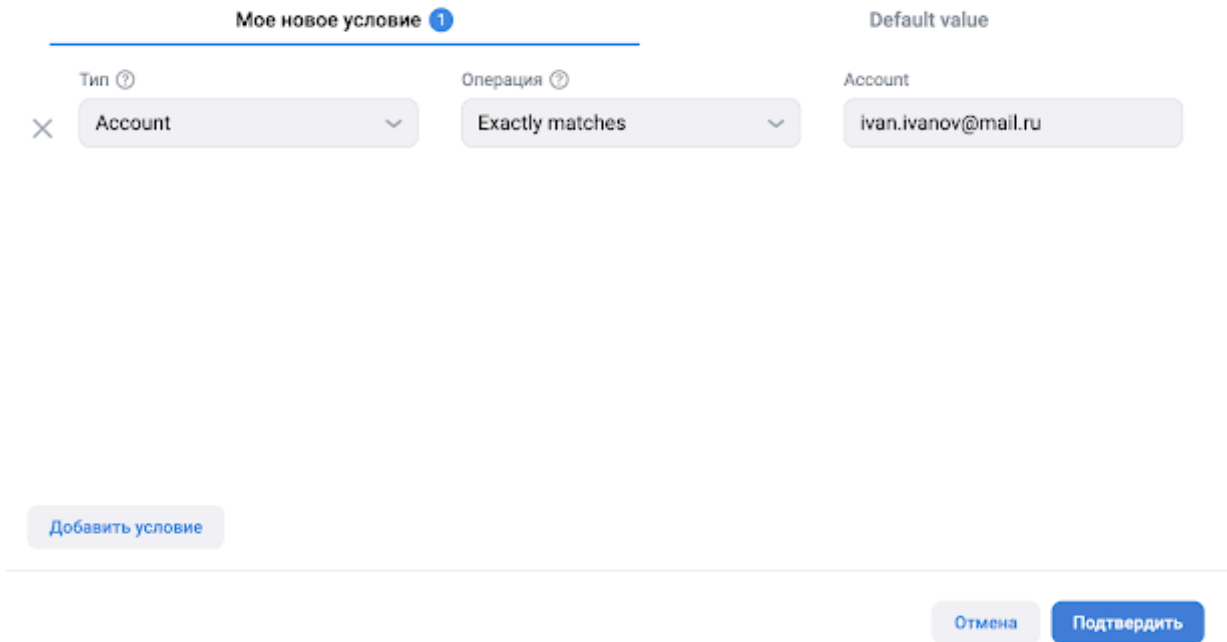
Testing a new parameter

Upon creation, editing, or deletion of a parameter its status is changed to **Pending**, that is, awaiting approval. In contrast to conditions though, you can test parameters before approving changes. The testing mechanism allows to apply a parameter on a single device or account to make sure it works correctly.

Testing is an important feature of the system that allows to avoid applying incorrect configuration values. Testing is based on applying custom atomic conditions to every possible version of the parameter.

Clicking the **Configure test** button displays the following window.

Тест параметра



Мое новое условие 1

Default value

Тип ⓘ

Account

Операция ⓘ

Exactly matches

Account

ivan.ivanov@mail.ru

Добавить условие

Отмена Подтвердить

Tabs at the top match added conditions and the default value of the changed parameter. On the screenshot above the tested value is the one that is received based on condition **My new condition**.

Clicking the **Run test** button changes the parameter status to **Testing**—in this status one you can confirm or reject changes.

Caution

The conditions being added fully replace atomic conditions of the **My new condition** parameter. This is done for testing complex scenarios that is difficult to reproduce on the user's device. For example, it can be a condition that utilizes the app version. You can replace the source conditions with custom ones and check how the interface looks after the **My new condition** condition is executed. In other words, the service allows to receive any parameter value on the device based on custom conditions.

The first condition is limited by the available choice of types and operations. Only the **DeviceID** and **Account** types are available together with the **Exactly matches** operation. This restriction is to prevent from using a too wide range of users for testing. There are no such restrictions starting from the second condition.

Note, that same **Account** or **DeviceID** is applied only to one condition or **Default value**. Otherwise, the system will not know exactly with what value to respond, thus, the value can be arbitrary and constantly changing.

During the deletion of the parameter there are not tabs and the selected atomic conditions are applied globally. Since the parameter is being deleted, it is important to

test the consequences of this action. In this case, particular parameter conditions are not important as they will be deleted with the parameter (see the example below).

Тест параметра

✕ Тип ⓘ Account Операция ⓘ Exactly matches Account ivan.ivanov@mail.ru

Добавить условие

Отмена

Подтвердить

Applying a new parameter

The parameter is enabled after approval by a user with the sufficient permissions.

User roles and permissions

Remote config provides flexible access management utilizing its permission distribution mechanism. Permissions are granted **per app**.

Permission list and description

Note

The company owner has the full permissions in all company apps.

Permission	Available activities
View	Viewing parameters, conditions and change log.
Record	Creation, editing, deletion, and rejection of parameters and conditions and parameter testing. This permission does not allow making approvals.
Approval	Approval or rejection of any changes in the systems.
Granting permissions	Granting permissions to any employee in the system for the current app.

Working with parameters and conditions

Remote Config protects users from unintentional changes of the parameters and conditions, that's why any change that might affect the app behavior will be sent for approval. Not all changes require approval. For example, changing a parameter or a condition description does not affect system behavior that's why such changes are applied immediately.

Access and permissions

It's important to note that permissions for making changes are divided — see [User roles](#).

Working with conditions

On condition creation it is in the **Pending** status, that is, awaiting approval.

The interface is different depending on the action type.

Creating a condition

Мое новое условие New Pending

Название
Мое новое условие

Описание
Описание условия

Атомарные условия

Тип	Операция	Значение
OS Version	Exactly matches	10

Подтвердить Отклонить Удалить

When a condition is created, it has the **New** label that indicates that the condition is new. This condition can be deleted without approval as it is in the draft state.

After approval you can use this condition in parameters.

Modifying a condition

Мое новое условие Pending

Название
Мое новое условие

Описание
Описание условия

Атомарные условия

Тип	Операция	Значение
OS Version	Exactly matches	10
Тип	Операция	Значение
OS Version	Exactly matches	11

Подтвердить

Отклонить

Удалить черновик

You can compare the old condition with a new one as well as discard changes by deleting the draft. All parameters will use the source version and ignore the draft version. On approval, all changes are applied and the user will receive all related parameters based on the new condition.

Deleting a condition

Мое новое условие Will be deleted Pending

Название
Мое новое условие

Описание
Описание условия

Атомарные условия

Тип	Операция	Значение
OS Version	Exactly matches	11

Подтвердить

Отклонить

Отменить удаление

When a condition removal is initiated the **Will be deleted** label is added to the condition name. Confirm action if you want to completely remove the condition from the system.

Rejecting changes

By rejecting changes you can adjust a condition if you made a mistake.

Мое новое условие Rejected

Название

Мое новое условие

Описание

Описание условия

Атомарные условия

Тип	Операция	Значение
OS Version	Exactly matches	11

Тип	Операция	Значение
OS Version	Exactly matches	12

Working with parameters

On parameter creation it is in the **Pending** status, that is, awaiting approval. In contrast to conditions though, you can test parameters before approving changes. The testing mechanism allows to apply a parameter on a single device or account to make sure it works correctly.

Creating a parameter

my_param New Pending

Ключ параметра
my_param

Тип значения
Numeric

Описание
Мой параметр

Условие Мое новое условие

1 42

Default value

1 10

Настроить тест

Запустить тест

Удалить

When a parameter is created, it has the New label. This parameter can be deleted without approval as it is in the draft state.

Modifying a parameter

my_param Pending

Ключ параметра my_param	Ключ параметра my_param
Тип значения Numeric	Тип значения Numeric
Описание Мой параметр	Описание Мой параметр
Мое новое условие 1 4 2	Мое новое условие 1 4 3
Значение по умолчанию 1 1 0	Значение по умолчанию 1 1 1

Настроить тест Запустить тест Удалить ⌵

You can compare the old parameter with a new one as well as discard changes by deleting the draft. All users receive the source parameter.

Deleting a parameter

Приложения / metida / Параметры

my_param Will be deleted Pending

Ключ параметра
my_param

Тип значения
Numeric

Описание
Мой параметр

Условие Мое новое условие

1 42

Default value

1 10

Настроить тест

Запустить тест

Отменить удаление

When a parameter removal is initiated the **Will be deleted** label is added to the parameter name.

Testing a parameter

Testing allows to avoid applying incorrect configuration values. Testing is based on applying custom atomic conditions to every possible version of the parameter.

Clicking the **Configure test** button displays the following window.

Тест параметра

Мое новое условие 1 Default value

Тип ⓘ Операция ⓘ

Account Exactly matches Account

ivan.ivanov@mail.ru

Добавить условие

Отмена Подтвердить

Tabs at the top match added conditions and the default value of the changed parameter. On the screenshot above the tested value is the one that is received based on condition **My new condition**.

Clicking the **Run test** button changes the parameter status to **Testing**—in this status one you can confirm or reject changes.

Warning

The conditions being added fully replace atomic conditions of the **My new condition** parameter. This is done for testing complex scenarios that is difficult to reproduce on the user's device. For example, it can be a condition that utilizes the app version. You can replace the source conditions with custom ones and check how the interface looks after the **My new condition** condition is executed. In other words, the service allows to receive any parameter value on the device based on custom conditions.

The first condition is limited by the available choice of types and operations. Only the **DeviceID** and **Account** types are available together with the **Exactly matches** operation. This restriction is to prevent from using a too wide range of users for testing. There are no such restrictions starting from the second condition.

Note, that same **Account** or **DeviceID** is applied only to one condition or **Default value**. Otherwise, the system will not know exactly with what value to respond, thus, the value can be arbitrary and constantly changing.

During the deletion of the parameter there are not tabs and the selected atomic conditions are applied globally. Since the parameter is being deleted, it is important to test the consequences of this action. In this case, particular parameter conditions are not important as they will be deleted with the parameter (see the example below).

Тест параметра

✕ Тип ⓘ Account Операция ⓘ Exactly matches Account ivan.ivanov@mail.ru

Добавить условие

Отмена

Подтвердить

Tracer

The Tracer service gathers and analyzes errors to send auto reports to the developer. This provides for fast and efficient error fixing. The service runs in iOS and Android mobile apps.

With Tracer you can:

- do profiling in production environment to determine what causes lags and freezes;
- find what causes memory leaks on the user's device in fully or partially automatic mode;
- detect forgotten and leaked files to reduce the risk of the app removal or user logout during the data cleaning.

Android

Quickstart

Registration and configuration

To start the procedure, perform the following actions.

1. Log in to the Tracer account.
2. Create or join organization.
3. Add an Android project (you must have administrator or owner permissions).

Connecting tracer dependencies to the project

In your <project>/settings.gradle.

```
pluginManagement {
    repositories {
        maven { url 'https://artifactory-external.vkpartner.ru/artifactory/maven/' }
    }
}
dependencyResolutionManagement {
    repositories {
        maven { url 'https://artifactory-external.vkpartner.ru/artifactory/maven/' }
    }
}
```

In your <project>/<app-module>/build.gradle.

```
plugins {
    id 'ru.ok.tracer' version '0.2.7'
}

tracer {
    defaultConfig {
        See in the "Settings" section
        pluginToken = "PLUGIN_TOKEN"
        appToken = "APP_TOKEN"
        // Enables mapping loading for the build. By default: disabled.
        uploadMapping = true
    }
    // You can also set configuration for every flavor, buildType, and buildVariant.
    // Configurations inherit defaultConfig.
    debug {
        // Parameters...
    }
    demoDebug {
        // Parameters...
    }
}

dependencies {
    // Plug-ins are independent of each other. You can connect only ones,
    // that are required at the moment.
    // Crashes and ANR gathering and analysis
    implementation "ru.ok.tracer:tracer-crash-report:0.2.7"
    // Native crashes gathering and analysis
    implementation "ru.ok.tracer:tracer-crash-report-native:0.2.7"
    // OOM heap dumps gathering and analysis
    implementation "ru.ok.tracer:tracer-heap-dumps:0.2.7"
    // Device disk usage analysis
    implementation "ru.ok.tracer:tracer-disk-usage:0.2.7"
    // Sampling profiler
    implementation "ru.ok.tracer:tracer-profiler-sampling:0.2.7"
    // Systrace
}
```

```
implementation "ru.ok.tracer:tracer-profiler-systrace:0.2.7"
}
```

Enabling and configuring tracer plug-ins in your project

Enable the HasTracerConfiguration interface in your Application.kt (see below).

```
class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            CoreTracerConfiguration.build {
                // tracer core options
            },
            CrashReportConfiguration.build {
                // crash collector options
            },
            CrashFreeConfiguration.build {
                // crash free counting options
            },
            HeapDumpConfiguration.build {
                // OOM heap dumps collector options
            },
            DiskUsageConfiguration.build {
                // disk usage analyzer options
            },
            SystraceProfilerConfiguration.build {
                // production systrace profiler options
            },
            SamplingProfilerConfiguration.build {
                // sampling profiler options
            },
        )
}
```

The HasTracerConfiguration.tracerConfiguration property will be called once on Application.onCreate process start but only after Application.attachBaseContext. In getter you can call the app context but it is early to call anything that will be initialized in onCreate.

Below is a detailed description of the options:

CoreTracerConfiguration — see below on this page.

CrashReportConfiguration and CrashFreeConfiguration — on page [Crash and ANR](#).

HeapDumpConfiguration — on page [Heap Dumps](#).

DiskUsageConfiguration — on page [Disk Usage](#).

SystraceProfilerConfiguration — on page [Systrace Profiler](#).

SamplingProfilerConfiguration — on page [Sampling Profiler](#).

CoreTracerConfiguration description

In a rare case you might want to configure the tracer core in your project, retrieve CoreTracerConfiguration from HasTracerConfiguration.

```
class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            CoreTracerConfiguration.build {
                // your options
            },
        )
}
```

Below are the CoreTracerConfiguration.Builder options.

setEnabled — not used and will be removed in version 0.3.x—tracer core is always enabled, however, inactive if there are no enabled plug-ins.

setHost, provideHost — tracer address change.

setStatHost, provideStatHost — tracer address change for the crash free feature.

setCustomAppKey, provideCustomAppKey — replaces sampleUploadToken from the gradle plug-in configuration.

Migrating to a new version

Migrating from 0.2.7 to 0.2.3

The Condition class and the SystraceProfiler.start(context, profileDuration, activeCondition) and SamplingProfiler.run(context, duration, condition) methods are deprecated and will soon be removed. For manual profiling use SystraceProfiler and SamplingProfiler API instead.

Where interestingEvents were set with TracerEvents.addEvent(), now call SystraceProfiler.commit() or SamplingProfiler.commit()—depending on the profiling type you used. Where you could catch interestingEvents, now call SystraceProfiler.start() or SamplingProfiler.start() respectively.

Migrating from 0.2.3 to 0.2.2

The TracerCrashReport.log(Throwable) method for sending non-fatals is now deprecated and will soon be removed. Use the TracerCrashReport.report(Throwable) method instead. This does not affect the TracerCrashReport.log(String) method for adding crash and non-fatal event logs—it will continue to work as usual. See [“Crash и ANR”](#) for up-to-date usage examples.

Starting from 0.2.3 the tracer-plugin connection method is changed. The dependency is changed from ru.ok.tracer:plugin:0.1.1 to ru.ok.tracer:tracer-plugin:0.2.3 (note, not only the version number is affected). Plugin id changes from ru.ok.tracer.mapping_plugin to ru.ok.tracer. Further on, the plug-in and runtime versions will use the same numbering and be published at the same time. See [“Quickstart”](#) for the up-to-date information.

Migrating from 0.2.2 to 0.1.15

Tracer.configure and Tracer.configureAsync methods. Instead of calling these methods, implement the HasTracerConfiguration interface in your Application class and retrieve everything that was previously passed in Tracer.configure. See [“Quickstart”](#) for the up-to-date information.

Now the tracer configuration is done at the current tracer start-up instead of the next tracer start-up.

Tracer Modules

Crash and ANR

Connecting dependencies to your project

In your <project>/<app-module>/build.gradle.

```
dependencies {
    implementation "ru.ok.tracer:tracer-crash-report:0.2.7"
}
```

Also, Tracer supports gathering and analysis of native crashes. If you want to gather crashes that occurred on the native code, connect the relevant dependency (see below).

```
dependencies {
    implementation "ru.ok.tracer:tracer-crash-report-native:0.2.7"
}
```

Warning

Currently, this feature is under active development. Crash logs gathering functions, however, native crashes are displayed without stack trace and other details. Now you can gather and count them while displaying and categorization are being tweaked.

For a detailed description of the dependencies see [“Quickstart”](#).

CrashReportConfiguration and CrashFreeConfiguration description

In your Application.kt.

```
class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            CrashReportConfiguration.build {
                // your options
            },
            CrashFreeConfiguration.build {
                // your options
            }
        )
}
```



```
}  
 )  
}
```

Below are the `CrashReportConfiguration.Builder` options:

`setEnabled` — enables/disables crash reporting. By default: enabled.
`setSendAnr` — disables ANR sending. By default: enabled.
`setNativeEnabled` — enables/disables native crash reporting. The default value depends on whether the `tracer-crash-report-native` dependency is connected. If the dependency is connected, then, reporting is by default enabled. If not, disabled. ⚠ You cannot enable native crash reporting without connecting the relevant dependency but you can disable reporting if the dependency is connected.

Below are the `CrashReportConfiguration.Builder` options that are deprecated or dangerous.

`setHost` — deprecated and will be removed in version 0.3.x. If you really need to change the host, use `CoreTracerConfiguration.setHost`;
`setSendLogs` — deprecated and does nothing, will be removed in version 0.3.x;
`setCountCrashFreeUsers` — deprecated and does nothing, will be removed in version 0.3.x; use `CrashFreeConfiguration.Builder.setEnabled` instead;
`setSendThreadsDump`, `setSendAsap`, `setMaxNonFatalExceptions` — deprecated and do nothing, will be deleted in version 0.3.x.

Below are the `CrashFreeConfiguration.Builder` options.

`setEnabled` — enables/disables crash free users count. By default: disabled!

Below are the `CrashFreeConfiguration.Builder` options that are deprecated or dangerous.

`setExperimentalMaxSessionsToUpload` — how many sessions to gather before sending a batch. By default: 10. For testing purposes only!
`setExperimentalMaxSessionTimeSpanToUpload` — for how long to gather sessions before sending a batch. By default: 4 hours (in milliseconds). For testing purposes only!
`setExperimentalUploadSessionsFromYesterday` — send a batch if yesterday's (or earlier) sessions are detected. By default: enabled.

TracerCrashReport description

To send non-fatals the `TracerCrashReport.report(throwable)` method is used.

```
// Log a non-fatal error.  
TracerCrashReport.report(new NonFatalException("I'll be ok soon"))
```

Crashes are grouped by common parts of a stack trace. By default, non-fatals, however, tweak this grouping approach.

You can gather all non-fatals in one group regardless of a stack trace using the report method with the issueKey parameter.

```
// Log a non-fatal error with the ISSUE-001 key
TracerCrashReport.report(NonFatalException("What a terrible failure"), issueKey = "ISSUE-001")
```

Warning

Currently, Tracer has a limit of 1 million events a day. That is why the excessive use of this method is not recommended.

You can also add [additional info](#) to an event.

Heap Dumps

Connecting dependencies to your project

In your <project>/<app-module>/build.gradle.

```
dependencies {
    implementation "ru.ok.tracer:tracer-heap-dumps:0.2.7"
}
```

For a detailed description of the dependencies see "[Quickstart](#)".

HeapDumpConfiguration description

In your Application.kt.

```
class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            HeapDumpConfiguration.builder {
                // your options
            },
        )
}
```

Below are the HeapDumpConfiguration.Builder options.

setEnabled — enables/disables OOM heap dumps gathering. By default: enabled.

Below are the HeapDumpConfiguration.Builder options that are deprecated or dangerous.

setProbability — deprecated and does nothing, will be removed in version 0.3.x;

setInterestingSize — deprecated and does nothing, will be removed in version 0.3.x;

Note

Heap dumps are sent at night, when the user doesn't use their device.

Disk Usage

Connecting dependencies to your project

In your project <project>/<app-module>/build.gradle.

```
dependencies {
    implementation "ru.ok.tracer:tracer-disk-usage:0.2.7"
}
```

For a detailed description of the dependencies see [“Quickstart”](#).

DiskUsageConfiguration description

In your Application.kt.

```
class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            DiskUsageConfiguration.build {
                // your options
            },
        )
}
```

Below are the DiskUsageConfiguration.Builder options.

- setEnabled — enable/disable plug-in. By default: enabled.
- setProbability — the probability (1/n) of daily background disk usage check for this user. By default: 0; that means that the plug-in is enabled.
- setInterestingSize — occupied space limit, on exceeding which the SDK will trigger an alarm and send report to Tracer. Measured in bytes. by default: 10 GB.
- setExcludePath — paths with known large files to be excluded from the check. Accepts only paths set by GlobalDirs.

Below is the description of GlobalDirs.

- GlobalDirs.INTERNAL_DATA.excludePath("foo/bar") — app internal store.
- GlobalDirs.EXTERNAL_DATA.excludePath("foo/bar") — app files on SD card.
- GlobalDirs.SRC.excludePath("foo/bar") — files of the app itself.

Systrace Profiler

Connecting dependencies to your project

In your <project>/<app-module>/build.gradle.

```
dependencies {
    implementation "ru.ok.tracer:tracer-profiler-systrace:0.2.7"
}
```

For a detailed description of the dependencies see [“Quickstart”](#).

SystraceProfilerConfiguration description

In your Application.kt.

```
class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            SystraceProfilerConfiguration.builder {
                // your options
            },
        )
}
```

Below are the SystraceProfilerConfiguration.Builder options.

setEnabled — enables/disables profiling. By default: enabled.

Below are the SystraceProfilerConfiguration.Builder options that are deprecated or dangerous.

setDurationMs — profiler performance time in milliseconds.

addCondition — adds a Condition that triggers profiling.

Description of Condition, TracerEvents, SystraceProfiler API, etc.

See [“Sampling Profiler”](#) for the up-to-date information.

Sampling Profiler

Connecting dependencies to your project

In your <project>/<app-module>/build.gradle.

```
dependencies {
    implementation "ru.ok.tracer:tracer-profiler-sampling:0.2.7"
}
```

For a detailed description of the dependencies see [“Quickstart”](#).

SamplingProfilerConfiguration description

In your Application.kt.

```

class MyApplication : Application(), HasTracerConfiguration {
    override val tracerConfiguration: List<TracerConfiguration>
        get() = listOf(
            SamplingProfilerConfiguration.build {
                // your options
            },
        )
}

```

Below are the SamplingProfilerConfiguration.Builder options.

setEnabled — enables/disables profiling. By default: enabled.

Below are the SamplingProfilerConfiguration.Builder options that are deprecated or dangerous.

setBufferSizeMb — see the android.os.Debug.startMethodTracingSampling description;

setSamplingIntervalUs — see the

android.os.Debug.startMethodTracingSampling description. By default: 5000.

setDurationMs — profiler performance time in milliseconds.

addCondition — adds a Condition that triggers profiling.

Condition.Deprecated description.

Condition is used to manage profiling start and send profiling results. Below is a usage example.

```

Condition.appStart(10_000, 7_000)

```

Start profiler with a 100% probability and send its performance result to the server if the app startup time exceeds a threshold of 7000 ms.

Creating own event.

```

val condition = Condition.build { // your options }

```

Below are the Condition.Builder options.

setTag("my_tag") — tag for the result to be loaded to Tracer.

setTagLimit(n) — maximum amount of the reports per day that will be accepted by the server.

setProbability(n) — the probability (1/n) of starting the profiler on startEvent occurrence.

setStartEvent("my_event") — the profiler will start on this event occurrence with the probability set above.

setInterestingEvent("my_other_event") — optional. If set, the profiling result will be sent to the backend only if the event occurs during profiler performance.

setInterestingDuration(n) — if the time between the start event and the interesting event exceeds this value, the profiling result will be sent to the backend.

Info

If an interesting event has its own counter, the comparison will be done against this counter. For example, app_freeze has a counter — the time the UI flow "hangs". Therefore, if interestingEvent == "app_freeze" and interestingDuration == 700, the report will be sent if there is a 700+ ms freeze during the profiler performance.

Manual profiling

There is an option to set profiling start/stop in code manually.

SamplingProfiler.start() — runs the profiler. Accepts the following parameters:

context: Context — App context;

tag: String — tag with which the result will be loaded to the tracer;

duration: Long — profiler performance time in milliseconds.

SamplingProfiler.abort() — stops the profiler and clears the result.

SamplingProfiler.commit() — stops the profiler and sends the result to the backend. If by the time of call the profiler is still active, the resulting tag will be <tag>_<tagSuffix>.

tagSuffix: String — suffix that will be added to the tag if the profiler stops prematurely (optional).

See the example below.

The profiler will start with the probability of 1/100000.

```
if (Random.nextInt(100_000) == 0) {
    SamplingProfiler.start(
        context = appContext,
        tag = "stream_request",
        duration = 10_000,
    )
}
// ... Code. For example, feed loading
SamplingProfiler.commit("loaded")
```

Description of "system" events

"System" events are used in the Condition class and manual profiling:

TracerEvents.EVENT_APP_START_BEGIN — "app_start_begin": app start.

TracerEvents.EVENT_APP_START_END — "app_start_end":

Application.onCreate() method stop.

TracerEvents.EVENT_FIRST_ACTIVITY_CREATED —

"app_first_activity_created": first activity created.

TracerEvents.EVENT_ACTIVITY_CREATED — "activity_created" — an activity is now in the created state.

TracerEvents.EVENT_FREEZE — "app_freeze" — UI flow froze for N ms. If this event is used as an interesting event, then, the interesting duration will be compared against N. For example, if N == 500, then, the profiling result will be sent if the UI flow freezes for more than 500 ms during the profiler performance.

EVENT_ANR — "app_anr": UI flow froze and didn't recover by the time the profiler stopped. N ms passed from freeze detection to the time the profiler stopped. If this event is used as an interesting event, then, the interesting duration will be compared against N. For example, if N == 5000, then, the profiling result will be sent if the UI flow freezes for more than 5000 ms during the profiler performance.

Adding user event (see below).

```
TracerEvents.addEvent(eventName, duration)
```

eventName — event name. Used in the Condition class and in the startEvent and interestingEvent methods.

duration — duration time of this event. If set, then, interestingDuration will not be calculated as difference between startEvent and interestingEvent, instead, it will be compared against the duration of this event.

Developer Documentation

RuStore Billing SDK

Kotlin

Quick Start	89
General Information	95
Payment functions availability	99
How to get up-to-date information on the product list	100
How to get the user's list of products	104
How to get purchase info	109
How to handle purchases	114
Server purchase validation	116
Purchase confirmation	117
Purchase cancellation	119
Consumption and cancellation scenario	121
Event Logging	122
Error handling	124
SDK payment error codes	125
Migration to Payments SDK v1.0.0 and higher	128
RuStore SDK payments Release Notes	133

Quick Start

Example of implementation

Please have a thorough look at the [application example](#) to learn how to integrate payments correctly.

Payment integration guideline

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your app user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

How to add a repository

Connect the repository:

```
repositories {
    maven {
        url =
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
    }
}
```

Dependency injection

Add the following code to your configuration file to inject the dependency:

```
dependencies {
    implementation("ru.rustore.sdk:billingclient:5.0.0")
}
```

How to initialize a library

Initialize the library before calling its methods.

Create RuStoreBillingClient by using RuStoreBillingClientFactory.create():

```
val billingClient: RuStoreBillingClient =
RuStoreBillingClientFactory.create(
    context = app,
```

```

consoleApplicationId = "111111",
deeplinkScheme = "yourappscheme",
themeProvider = null,
debugLogs = false,
externalPaymentLoggerFactory = null,
)

```

- context — Android context. Any context is allowed, applicationContext is used in the release version.
- consoleApplicationId — application code from the RuStore Developer Console (example: <https://console.rustore.ru/apps/111111>).
- deeplinkScheme — deeplink scheme required to return to your app upon payment via a third-party application (for example, SberPay or SBP). SDK generates its host for this scheme.
- themeProvider — interface that provides BillingClientTheme. There are 2 possible options of BillingClientTheme: light and dark. This interface is optional; the light team is set by default.
- externalPaymentLoggerFactory — interface that provides access to an external logger.
- debugLogs — flag that regulates logging (logs will be automatically disabled for Release builds).

The ApplicationId specified in build.gradle must match the ApplicationId of the apk file you published to the [RuStore Console](#).

The keystore signature must be the same as the one used to sign the application published to the [RuStore Console](#) system. Make sure that the buildType uses (eg: debug) uses the same signature as the published application (eg: release).

The library supports event logging, which is enabled separately when the library is initialized.

For details on library initialization, read the information above

Handling deeplinks in your app

To redirect a user back to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```

<activity
    android:name=".YourBillingActivity">

```

```

<intent-filter>

```

```

        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="yourappscheme" />
    </intent-filter>

</activity>

```

where "yourappscheme" — your deeplink scheme, it can be changed to another one. This scheme must match the `deeplinkScheme` parameter passed to `init()`.

You also need to add the following code to the application to ensure a successful return:

```

class YourBillingActivity: AppCompatActivity() {

    // Previously created with RuStoreBillingClientFactory.create()
    private val billingClient: RuStoreBillingClient =
        YourDependencyInjection.getBillingClient()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        if (savedInstanceState == null) {
            billingClient.onNewIntent(intent)
        }
    }

    override fun onNewIntent(intent: Intent?) {
        super.onNewIntent(intent)
        billingClient.onNewIntent(intent)
    }
}

```

To restore your app after deep linking, you need to add the following to `AndroidManifest.xml`:

```
android:launchMode="singleTask"
```

AndroidManifest.xml

```
<activity
    android:name=".YourBillingActivity"
    android:launchMode="singleTask"
    android:exported="true"
    android:screenOrientation="portrait"
    android:windowSoftInputMode="adjustResize">
```

General Information

You can see the "Quick Start" section to quickly integrate payments into the app.

Example of implementation

Please have a thorough look at the [application example](#) to learn how to integrate payments correctly.

Payment integration guideline

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your app user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

How to add a repository

Connect the repository:

```
repositories {
    maven {
        url =
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
    }
}
```

Dependency injection

Add the following code to your configuration file to inject the dependency:

```
dependencies {
    implementation("ru.rustore.sdk:billingclient:5.0.0")
}
```



How to initialize a library

Initialize the library before calling its methods.

Create `RuStoreBillingClient` by using `RuStoreBillingClientFactory.create()`:

```
val billingClient: RuStoreBillingClient =  
RuStoreBillingClientFactory.create(  
    context = app,  
    consoleApplicationId = "111111",  
    deeplinkScheme = "yourappscheme",  
    themeProvider = null,  
    debugLogs = false,  
    externalPaymentLoggerFactory = null,  
)
```

- `context` — Android context. Any context is allowed, `applicationContext` is used in the release version.
- `consoleApplicationId` — application code from the RuStore Developer Console (example: <https://console.rustore.ru/apps/111111>).
- `deeplinkScheme` — deeplink scheme required to return to your app upon payment via a third-party application (for example, SberPay or SBP). SDK generates its host for this scheme.
- `themeProvider` — interface that provides `BillingClientTheme`. There are 2 possible options of `BillingClientTheme`: light and dark. This interface is optional; the light theme is set by default.
- `externalPaymentLoggerFactory` — interface that provides access to an external logger.
- `debugLogs` — flag that regulates logging (logs will be automatically disabled for Release builds).

The `ApplicationId` specified in `build.gradle` must match the *applicationId* of the apk file published in RuStore Console.

The deeplink schema passed to `deeplinkScheme` must match the schema specified in `AndroidManifest.xml` in the Deeplink Processing section.

The keystore signature must match the one used to sign the app published in RuStore Console. Make sure that the `buildType` (eg debug) uses the same signature as the published app.

Handling deeplinks in your app

To redirect a user to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity
    android:name=".YourBillingActivity">

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="yourappscheme" />
    </intent-filter>

</activity>
```

where "yourappscheme" — your deeplink scheme, it can be changed to another one.

This scheme must match the deeplinkScheme parameter passed to init().

Next, add the following code to the Activity you need to return to after making the payment (your store page):

```
class YourBillingActivity: AppCompatActivity() {

    // Previously created with RuStoreBillingClientFactory.create()
    private val billingClient: RuStoreBillingClient =
        YourDependencyInjection.getBillingClient()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        if (savedInstanceState == null) {
            billingClient.onNewIntent(intent)
        }
    }

    override fun onNewIntent(intent: Intent?) {
        super.onNewIntent(intent)
        billingClient.onNewIntent(intent)
    }
}
```

```
}  
}
```

To restore your app after deep linking, you need to add the following to AndroidManifest.xml:

```
android:launchMode="singleTask"
```

```
<activity  
    android:name=".YourBillingActivity"  
    android:launchMode="singleTask"  
    android:exported="true"  
    android:screenOrientation="portrait"  
    android:windowSoftInputMode="adjustResize">
```

Payment functions availability

To check whether your app supports payment functions, call the `checkPurchasesAvailability` method. This method checks the following conditions:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all conditions are met, the method returns `FeatureAvailabilityResult.Available`. Otherwise, it returns `FeatureAvailabilityResult.Unavailable(val cause: RuStoreException)`, where *cause* indicates an unfulfilled condition. All possible `RuStoreException` errors are described in [Error Handling](#). Other errors (e.g. "No internet connection") are processed in `onFailure`.

```
RuStoreBillingClient.checkPurchasesAvailability(context)
    .addOnSuccessListener { result ->
        when (result) {
            FeatureAvailabilityResult.Available -> {
                // Process purchases available
            }
            is FeatureAvailabilityResult.Unavailable -> {
                // Process purchases unavailable
            }
        }
    }.addOnFailureListener { throwable ->
        // Process unknown error
    }
```

where `context` refers to Android context.

How to get up-to-date information on the product list

Use the `getProducts` method to get a list of products:

```
val productsUseCase: ProductsUseCase = billingClient.products
productsUseCase.getProducts(productIds = listOf("id1", "id2"))
    .addOnSuccessListener { products: List<Product> ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```


- `productIds`: `list<string>` — list of product identifiers. Maximum length is 2083 symbols in a list.

The method returns:

```
data class Product(
    val productId: String,
    val productType: ProductType?,
    val productStatus: ProductStatus,
    val priceLabel: String?,
    val price: Int?,
    val currency: String?,
    val language: String?,
    val title: String?,
    val description: String?,
    val imageUrl: Uri?,
    val promoImageUrl: Uri?,
    val subscription: ProductSubscription?,
)
```

- `productId` — product identifier;
- `productType` — product type;
- `productStatus` — product status;
- `priceLabel` — formatted product price, including the currency symbol in `[language]`;
- `price` — price in minor units (in kopecks);
- `currency` — ISO 4217 currency code;
- `language` — language specified with the BCP 47 encoding;
- `title` — product name in `[language]`;
- `description` — product description in `[language]`;
- `imageUrl` — link to the image;
- `promoImageUrl` — link to the promo image;
- `subscription` — subscription description, returned only for products with the subscription type.

Product structure

```
data class ProductsResponse(
    override val meta: RequestMeta?,
    override val code: Int,
```

```
        override val errorMessage: String?,
        override val errorDescription: String?,
        override val errors: List<DigitalShopGeneralError>?,
        val products: List<Product>?,
    ) : ResponseWithCode
```

- meta — additional request info;
- code — response code;
- errorMessage — error message;
- errorDescription — error description;
- errors — list of errors;
- products — list of products.

Subscription Structure:

```
data class ProductSubscription(
    val subscriptionPeriod: SubscriptionPeriod?,
    val freeTrialPeriod: SubscriptionPeriod?,
    val gracePeriod: SubscriptionPeriod?,
    val introductoryPrice: String?,
    val introductoryPriceAmount: String?,
    val introductoryPricePeriod: SubscriptionPeriod?
)
```

- subscriptionPeriod — subscription period;
- freeTrialPeriod — trial subscription period;
- gracePeriod — subscription grace period;
- introductoryPrice — formatted introductory subscription price, including the currency symbol, in product:language;
- introductoryPriceAmount — introductory price in minor units of currency (in kopecks);

- introductoryPricePeriod — calculated period of the introductory price.

Structure of the subscription period:

```
data class SubscriptionPeriod(  
    val years: Int,  
    val months: Int,  
    val days: Int,  
)
```

- years — number of years;
- months — number of months;
- days — number of days.

How to get the user's list of products

Use the `getPurchases` method to get the user's list of purchases

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.getPurchases()
    .addOnSuccessListener { purchases: List<Purchase> ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

Purchase Structure:

```
data class Purchase(
    val purchaseId: String?,
    val productId: String,
    val productType: ProductType?,
    val invoiceId: String?,
    val description: String?,
    val language: String?,
    val purchaseTime: Date?,
    val orderId: String?,
    val amountLabel: String?,
    val amount: Int?,
    val currency: String?,
    val quantity: Int?,
    val purchaseState: PurchaseState?,
    val developerPayload: String?,
    val subscriptionToken: String?
)
```

- `purchaseId` — purchase ID;
- `productId` — product identifier;
- `productType` — product type;
- `invoiceId` — invoice ID;
- `description` — purchase description;
- `language` — language specified with the BCP 47 encoding;
- `purchaseTime` — purchase time (in RFC 3339 format);
- `orderId` — unique payment identifier generated by the application (uuid);

- amountLabel — formatted purchase price, including the currency symbol in [language];
- amount — price in minor units of currency;
- currency — ISO 4217 currency code;
- quantity — number of products;
- purchaseState — purchase status:
 - possible values of the purchase condition:
 - CREATED — created;
 - INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;
 - CANCELLED — purchase canceled;
 - CONSUMED — purchase consumption is confirmed;
 - CLOSED — subscription is canceled.
- developerPayload — line specified by the developer that contains additional information about the order;
- subscriptionToken — token for validating a purchase on the server.

Product structure

```

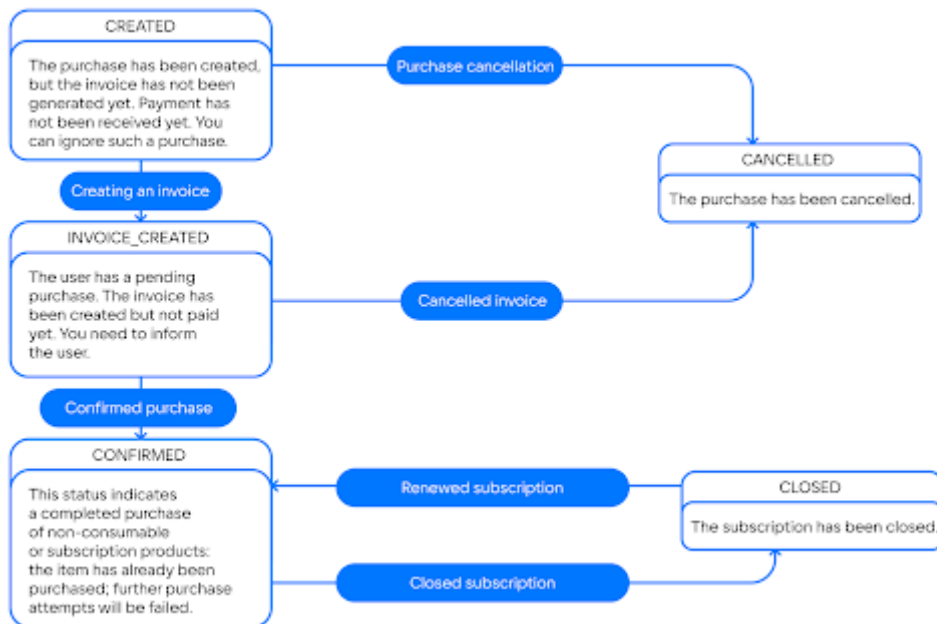
data class PurchasesResponse(
    override val meta: RequestMeta?,
    override val code: Int,
    override val errorMessage: String?,
    override val errorDescription: String?,
    override val errors: List<DigitalShopGeneralError>?,
    val purchases: List<Purchase>?,
) : ResponseWithCode

```

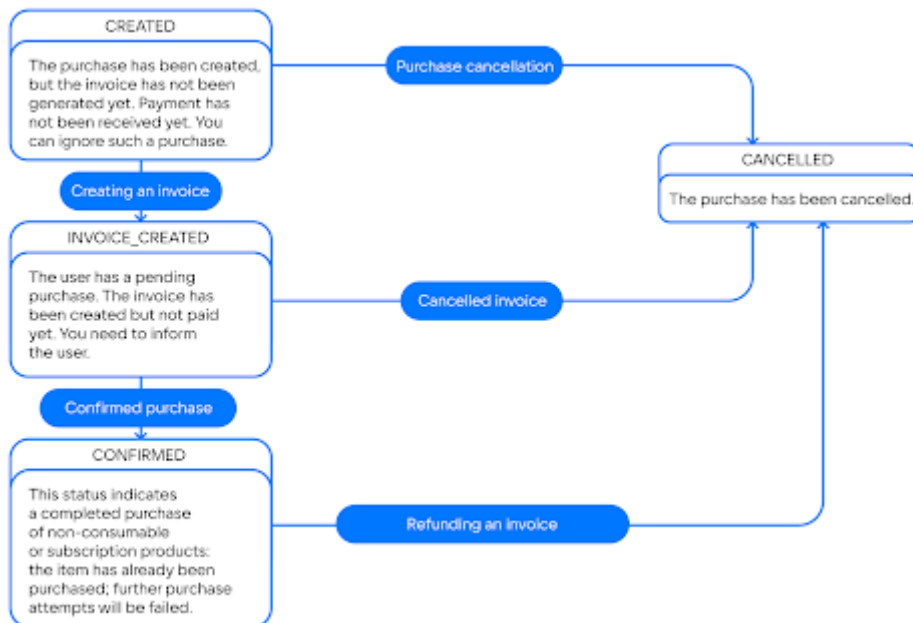
- meta — additional request info;
- code — response code;
- errorMessage — error message;
- errorDescription — error description;
- errors — list of errors;
- purchases — list of products.

The purchaseState model:

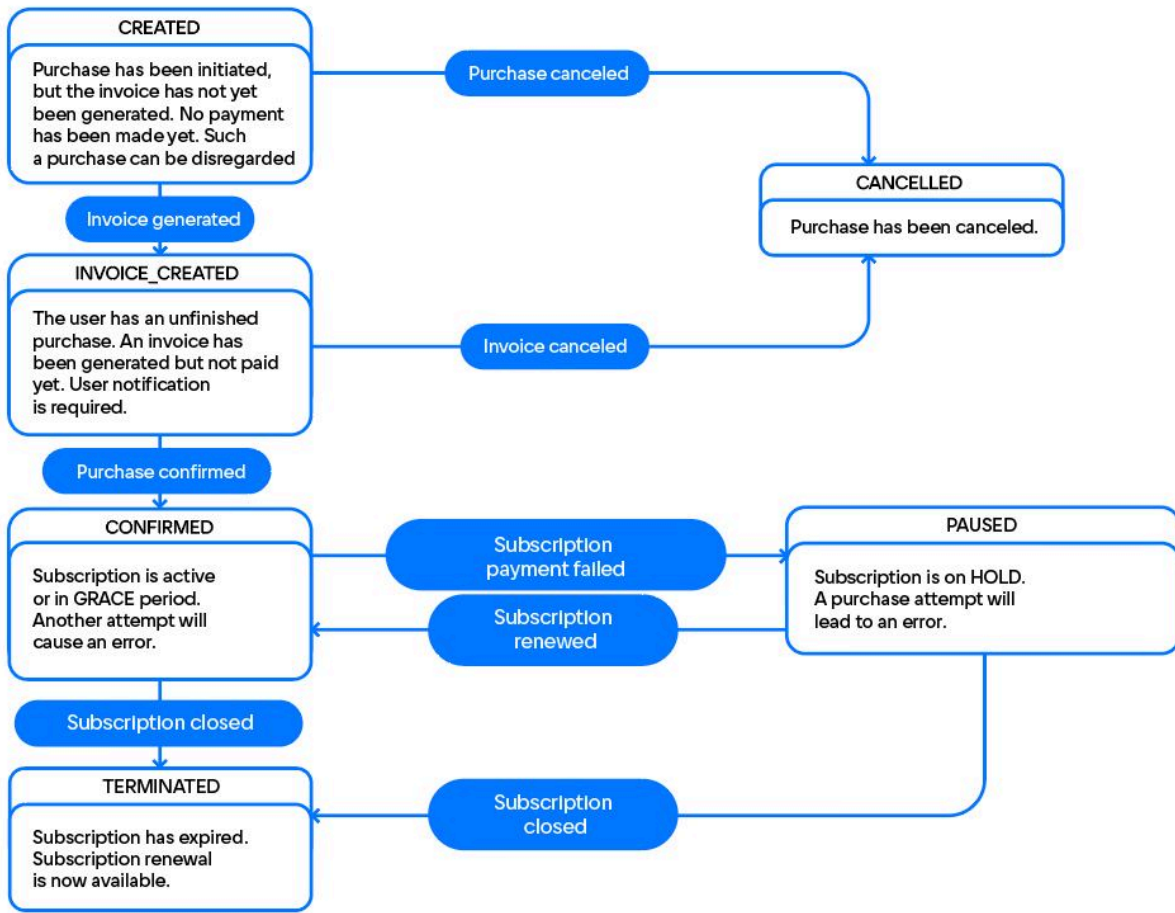
A status-based subscription purchase model (SUBSCRIPTIONS):



A status-based non-consumables subscription (NON-CONSUMABLES):



A status-based consumables subscription (CONSUMABLES):



How to get purchase info

Use the `getPurchaseInfo` method to get information about purchases:

```

val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.getPurchaseInfo("purchaseId")
    .addOnSuccessListener { purchase: Purchase ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
  
```

Purchase Structure:

```

data class Purchase(
  
```



```

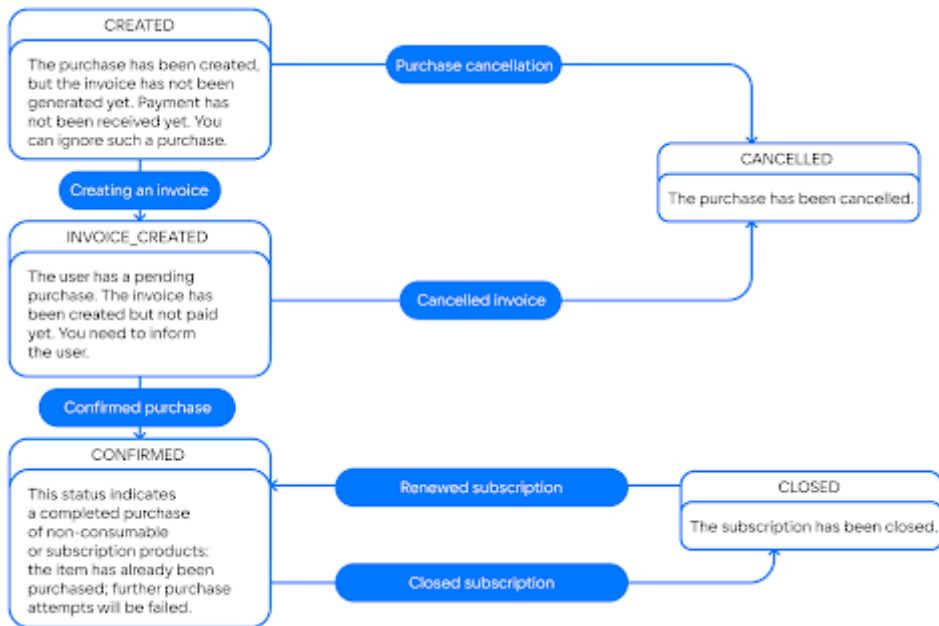
val purchaseId: String?,
val productId: String,
val productType: ProductType?,
val invoiceId: String?,
val description: String?,
val language: String?,
val purchaseTime: Date?,
val orderId: String?,
val amountLabel: String?,
val amount: Int?,
val currency: String?,
val quantity: Int?,
val purchaseState: PurchaseState?,
val developerPayload: String?,
val subscriptionToken: String?
)

```

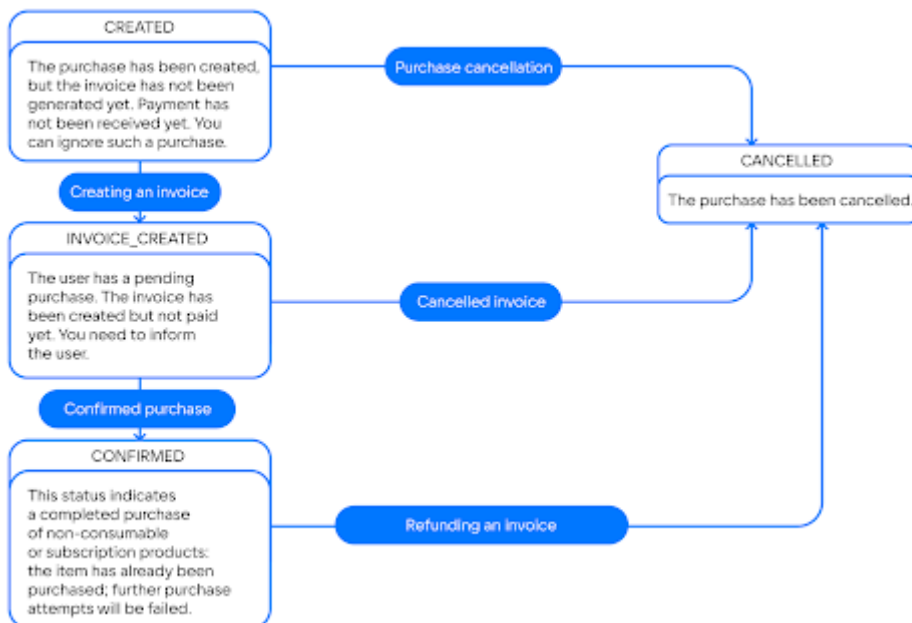
- purchaseId — purchase ID;
- productId — product identifier;
- productType — product type;
- invoiceId — invoice ID;
- description — purchase description;
- language — language specified with the BCP 47 encoding;
- purchaseTime — purchase time (in RFC 3339 format);
- orderId — unique payment identifier generated by the application (uuid);
- amountLabel — formatted purchase price, including the currency symbol in [language];
- amount — price in minor units of currency;
- currency — ISO 4217 currency code;
- quantity — number of products;
- purchaseState — purchase status:
 - possible values of the purchase condition:
 - CREATED — created;
 - INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;
 - CANCELLED — purchase canceled;
 - CONSUMED — purchase consumption is confirmed;
 - CLOSED — subscription is canceled.
- developerPayload — line specified by the developer that contains additional information about the order;
- subscriptionToken — token for validating a purchase on the server.

The purchaseState model:

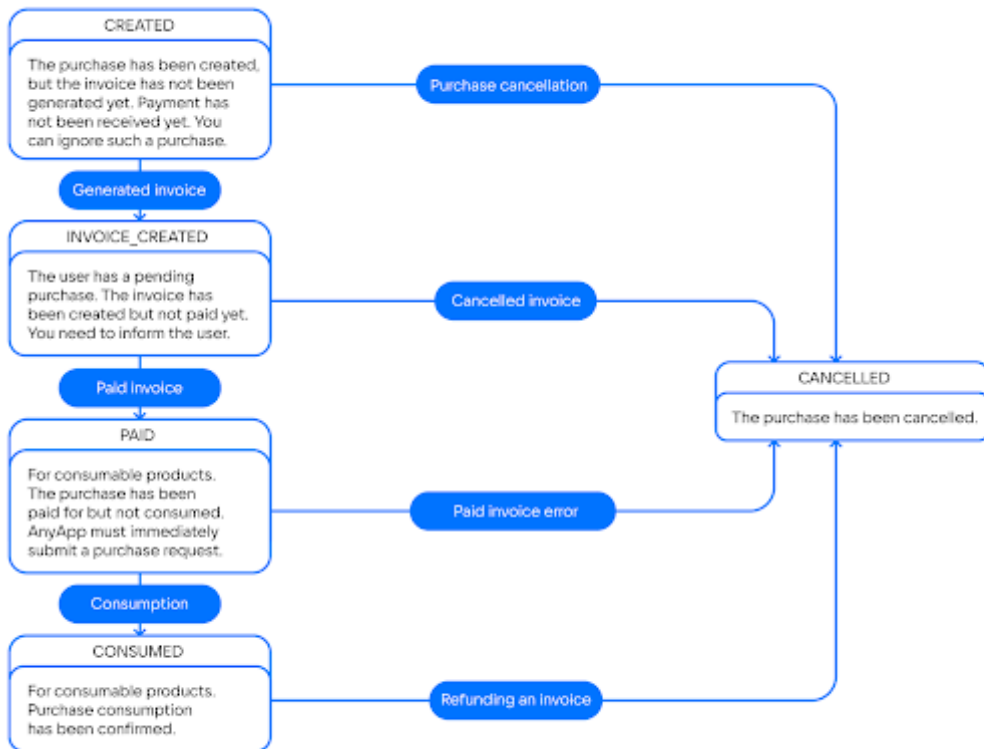
A status-based subscription purchase model (SUBSCRIPTIONS):



A status-based non-consumables subscription (NON-CONSUMABLES):



A status-based consumables subscription (CONSUMABLES):



How to handle purchases

Use the `purchaseProduct` method to call a product purchase:

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.getPurchaseInfo("purchaseId")
    .addOnSuccessListener { purchase: Purchase ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

- `productId`: String — product ID;
- `orderId`: String — order ID, generated by AnyApp (optional. If not specified, it is generated automatically);
- `quantity`: Int — number of products (optional);
- `developerPayload` — additional information from the AnyApp developer (optional).

Payment result structure:

```
public sealed interface PaymentResult {

    public data class Success(
        val orderId: String?,
        val purchaseId: String,
        val productId: String,
        val invoiceId: String,
        val subscriptionToken: String? = null,
    ) : PaymentResult

    public data class Cancelled(
        val purchaseId: String,
    ) : PaymentResult

    public data class Failure(
        val purchaseId: String?,
        val invoiceId: String?,
        val orderId: String?,
        val quantity: Int?,
    ) : PaymentResult
```

```

        val productId: String?,
        val errorCode: Int?,
    ) : PaymentResult

    public object InvalidPaymentState : PaymentResult()
}

```

- InvoiceResult — payments completed with a result;
- InvalidInvoice — payments completed without an invoice. Probably they were started with incorrect invoice (empty string, for example);
- PurchaseResult — successful purchase of a digital item;
- InvalidPurchase — failed payment for a digital product;
- InvalidPaymentState — no PaymentState when payments are completed.

Server purchase validation

For server purchase validation you can use the `subscriptionToken` in the `PurchaseResult` returned by `purchaseProduct` in case of a successful purchase.

`SubscriptionToken` consists of `invoiceId` of purchase and `userId` of `RuStore`, written with a dot: `"$invoiceId.$userId"`.

Getting `subscriptionToken` via purchase result

```

val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.purchaseProduct(productId).addOnSuccessListener {
    paymentResult ->
        if (paymentResult is PaymentResult.Success) {
            val subscriptionToken = paymentResult.subscriptionToken
            yourApi.validate(subscriptionToken)
        }
}

```

You can also get a `subscriptionToken` in the `Purchase` entity. The `Purchase` entity can be retrieved using the `getPurchases()` method:

Getting `subscriptionToken` via purchase result

```

val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.getPurchases().addOnSuccessListener { purchases ->
    purchases.forEach { purchase ->
        yourApi.validate(purchase.subscriptionToken)
    }
}

```

Purchase confirmation

The RuStore application consists of the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);
- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation if they are in the PurchaseState.PAID state.

You can use the confirmPurchase method to confirm the purchase:

```

val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.confirmPurchase(purchaseId = "purchaseId",
developerPayload = null)
    .addOnSuccessListener {
        // Process success
    }.addOnFailureListener { throwable: Throwable ->
        // Process error
    }

```

- purchaseId — purchase ID;
- developerPayload — line specified by the developer that contains additional information about the order (optional).

Purchase cancellation

You can cancel purchases in your app.

You can use the `deletePurchase` method to cancel the purchase:

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.deletePurchase(purchaseId = "purchaseId")
    .addOnSuccessListener {
        // Process success
    }.addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

- `purchaseId` — purchase ID.

Note. Use this method if your app logic is related to purchase cancellation. The purchase is canceled automatically after a 20-min timeout, or upon a second purchase from the same customer.

Error Handling

Possible errors:

- `RuStoreNotInstalledException()` — RuStore is not installed on the user's device;
- `RuStoreOutdatedException()` — RuStore, installed on the user's device, does not support payment processing functions;
- `RuStoreUserUnauthorizedException()` — user is not authorized on the RuStore;
- `RuStoreApplicationBannedException()` — your application is blocked on the RuStore;
- `RuStoreUserBannedException()` — user is blocked on the RuStore;
- `RuStoreException(message: String)` — basic RuStore error, from which all other errors are inherited.

When calling the `RuStoreBillingClient.purchases.purchaseProduct()` method, errors are handled automatically. You can use the `resolveForBilling` method to show an error dialog to the user:

Error handling

```
public fun RuStoreException.resolveForBilling(context: Context)
```


Consumption and cancellation scenario

Uncompleted payments must be processed by the AnyApp developer.

The purchase cancellation method should be used if:

1. The method of getting the list of products returned the purchase status as follows:
 - PurchaseState.CREATED;
 - PurchaseState.INVOICE_CREATED;

Note. In some cases, after paying through a banking app (SBP, SberPay, TinkoffPay, etc.), the purchase status may still return PurchaseState.INVOICE_CREATED when you subsequently return to AnyApp. This is caused by the purchase processing time by the bank. Therefore, the developer needs to correctly link the shopping list obtaining function to the life cycle on the screen.

To solve this problem, you can cancel a purchase in the PurchaseState.INVOICE_CREATED status only through user interaction with the application. For example, create a separate button for this purpose.

2. The purchase method (purchaseProduct) returned PaymentResult.Cancelled.
3. The purchase method (purchaseProduct) returned PaymentResult.Failure.

Use product consumption method (confirmPurchase) if the method the purchase obtaining method (getPurchases) returns a CONSUMABLE product and with the status PurchaseState.PAID.

Event Logging

If you want to log payment library events when calling `RuStoreBillingClient.init` add `externalPaymentLoggerFactory` and `debugLogs` parameters (these parameters are optional for initialization):

```
val billingClient: RuStoreBillingClient =
RuStoreBillingClientFactory.create(
    context = app,
    consoleApplicationId = "111111",
    deeplinkScheme = "yourappscheme",
    externalPaymentLoggerFactory = { tag -> PaymentLogger(tag) },
    debugLogs = true
)

class PaymentLogger(private val tag: String) : ExternalPaymentLogger
{
    override fun d(e: Throwable?, message: () -> String) {
        Log.d(tag, message.invoke(), e)
    }

    override fun e(e: Throwable?, message: () -> String) {
        Log.e(tag, message.invoke(), e)
    }

    override fun i(e: Throwable?, message: () -> String) {
        Log.i(tag, message.invoke(), e)
    }

    override fun v(e: Throwable?, message: () -> String) {
        Log.v(tag, message.invoke(), e)
    }

    override fun w(e: Throwable?, message: () -> String) {
        Log.w(tag, message.invoke(), e)
    }
}
```

Logging processing parameters:

- `externalPaymentLoggerFactory` — interface that allows you to create a logger that sends the library logs to the host application;

- debugLogs — enable logs (logs will be automatically disabled for Release builds).

Where PaymentLogger — example of payment event logging implementation.

Theme Changing

The SDK supports dynamic theme changing via the `BillingClientThemeProvider` provider interface:

```
val billingClient: RuStoreBillingClient =
RuStoreBillingClientFactory.create(
    context = app,
    consoleApplicationId = "111111",
    deeplinkScheme = "yourappscheme",
    themeProvider? = BillingClientThemeProviderImpl(),
)

class BillingClientThemeProviderImpl: BillingClientThemeProvider {

    override fun provide(): BillingClientTheme {
        val darkTheme = ....
        if(darkTheme){
            BillingClientTheme.Dark
        } else {
            BillingClientTheme.Light
        }
    }
}
```

Error handling

Possible errors:

- `RuStoreNotInstalledException()` — RuStore is not installed on the user's device;
- `RuStoreOutdatedException()` — RuStore, installed on the user's device, does not support payment processing functions;
- `RuStoreUserUnauthorizedException()` — user is not authorized on the RuStore;
- `RuStoreApplicationBannedException()` — your application is blocked on the RuStore;
- `RuStoreUserBannedException()` — user is blocked on the RuStore;
- `RuStoreException(message: String)` — basic RuStore error, from which all other errors are inherited.

When calling the `RuStoreBillingClient.purchases.purchaseProduct()` method, errors are handled automatically.

You can use the `resolveForBilling` method to show an error dialog to the user:

```
public fun RuStoreException.resolveForBilling(context: Context)
```

SDK payment error codes

Description of possible errors in the "code" field of the "ResponseWithCode" interface:

http code	code	Description
200	0	Successful request
400	40001	Incorrect request parameters: mandatory parameters are not filled in/incorrect parameters format
400	40003	No application found
400	40004	Inactive application status
400	40005	Product not found
400	40006	Inactive product status
400	40007	Invalid product type. Supported types: "consumables", "non-consumables", "subscription"
400	40008	A purchase with this "order_id" already exists
400	40009	The current client has a purchase of this product with the status "invoice_created". You are required to offer the client to pay for/cancel the purchase
400	40010	For "consumables" product type. The current customer already has purchased this product with the status "paid". First you need to confirm the purchase on the device, and then you can send the following purchase request for this product
400	40011	For "non-consumables" product type. The current client already has purchased this product with the status "pre_confirmed"/"confirmed". Such product has already been purchased. This product cannot be sold more than once
400	40012	For "subscription" product type. The current client has already purchased this product with the status "pre_confirmed"/"confirmed". Such product has already been purchased. The product cannot be sold more than once

400	40013	For "subscription" product type. When requesting the subscription service for a list of products "GET/products" ("serviceld", "user_id") data were not received
400	40014	The required attribute(s) was not received in the request
400	40015	Failed to change status when updating purchase (no transition allowed)
400	40016	When purchasing a subscription for a non-consumable product, the number > 1 is specified
400	40017	Product removed, no new purchases available
400	40018	You cannot consume a"product type" type product
401	40101	Invalid token
401	40102	Token lifetime has expired
403	40301	Access to the requested resource is denied (unauthorized)
403	40302	The current call is not authorized (method prohibited) for the token
403	40303	The application ID in the request does not match the one specified in the token
403	40305	Incorrect token type
404	40401	Not found
408	40801	The notification timeout period specified in the request has expired
500	50***	Internal payment service error

Migration to Payments SDK v2.2.0 and higher

General

In version 2.2.0, the purchase model in `PaymentResult` was significantly modified.

Follow the steps below to switch to the new SDK version smoothly.

Dependency update

To update the dependency, call the `billingclient` version in the dependencies block of your `build.gradle`:

`build.gradle`

```
dependencies {  
    implementation("ru.rustore.sdk:billingclient:2.2.0")  
}
```


Model update

Getting List of Products

The list of products model was significantly modified. From now on **getProducts()** returns the list of products:

```
val productsUseCase: ProductsUseCase = billingClient.products
productsUseCase.getProducts(productIds = listOf("id1", "id2"))
    .addOnSuccessListener { products: List<Product> ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

At that, the product and the error model remained unchanged.

Getting List of Purchases

The list of purchases model was significantly modified. From now on, **getPurchases()** returns the list of purchases:

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.getPurchases()
    .addOnSuccessListener { purchases: List<Purchase> ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

At that, the product and the error model remained unchanged.

Getting Product Info

The purchase info method was significantly modified. From now on, **getPurchaseInfo()** returns the purchase method:

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.getPurchaseInfo("purchaseId")
    .addOnSuccessListener { purchase: Purchase ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

The error model remained unchanged.

Purchases

In this version, the purchase model was also modified. The new model is represented as follows:

```

public sealed interface PaymentResult {

    public data class Success(
        val orderId: String?,
        val purchaseId: String,
        val productId: String,
        val invoiceId: String,
        val subscriptionToken: String? = null,
    ): PaymentResult

    public data class Cancelled(
        val purchaseId: String,
    ): PaymentResult

    public data class Failure(
        val purchaseId: String?,
        val invoiceId: String?,
        val orderId: String?,
        val quantity: Int?,
        val productId: String?,
        val errorCode: Int?,
    ): PaymentResult

    public object InvalidPaymentState : PaymentResult()
}

```

where:

- Success — digital product purchased successfully.
- Failure — product purchase error.
- Cancelled — product purchase canceled.
- InvalidPaymentState — SDK error. Returned in case of deeplink processing errors.

Please note that the product purchase and cancellation scenario were successfully modified.

Product purchase scenario

The product purchase scenario was significantly changed. The purchase method can now return an error:

```

val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.confirmPurchase(purchaseId = "purchaseId", developerPayload =
null)
    .addOnSuccessListener {
        // Process success
    }.addOnFailureListener { throwable: Throwable ->
        // Process error
    }

```

Product cancellation scenario

The product cancellation scenario was significantly changed. The cancellation method can now return an error:

```

val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.deletePurchase(purchaseId = "purchaseId")
    .addOnSuccessListener {
        // Process success
    }.addOnFailureListener { throwable: Throwable ->
        // Process error
    }

```

Product Purchase and Cancellation Scenario

Modifications in the product led to changes in the product purchase and cancellation scenario.

Use deletePurchase method if:

1. The getPurchases method returned an error with the following status:
 1. PurchaseState.CREATED.
 2. PurchaseState.INVOICE_CREATED.
2. The purchaseProduct method returned PaymentResult.Cancelled.
3. The purchaseProduct method returned PaymentResult.Failure.

Use confirmPurchase if getPurchases returned a CONSUMABLE type error with PurchaseState.PAID status.

1.x.x to 3.x.x Payments Migration

General

In version 3.0.0, the purchase model in `PaymentResult` was significantly modified.

Follow the steps below to switch to the new SDK version smoothly.

Dependency update

To update the dependency, call the `billingclient` version in the `dependencies` block of your `build.gradle`:

`build.gradle`

```
dependencies {  
    implementation("ru.rustore.sdk:billingclient:3.2.0")  
}
```

Model update

Getting List of Products

The list of products model was significantly modified. From now on **getProducts()** returns the list of products:

```
val productsUseCase: ProductsUseCase = billingClient.products
productsUseCase.getProducts(productIds = listOf("id1", "id2"))
    .addOnSuccessListener { products: List<Product> ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

At that, the product and the error model remained unchanged.

Getting List of Purchases

The list of purchases model was significantly modified. From now on, **getPurchases()** returns the list of purchases:

```
val purchasesUseCase: PurchasesUseCase =
    billingClient.purchases
    purchasesUseCase.getPurchases()
        .addOnSuccessListener { purchases: List<Purchase> ->
            // Process success
        }
        .addOnFailureListener { throwable: Throwable ->
            // Process error
        }
```

At that, the product and the error model remained unchanged.

Getting Product Info

The purchase info method was significantly modified. From now on, **getPurchaseInfo()** returns the purchase method:

```
val purchasesUseCase: PurchasesUseCase =
    billingClient.purchases
purchasesUseCase.getPurchaseInfo("purchaseId")
    .addOnSuccessListener { purchase: Purchase ->
        // Process success
    }
    .addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

The error model remained unchanged.

Purchases

In this version, the purchase model was also modified. The new model is represented as follows:

```

public sealed interface PaymentResult {

    public data class Success(
        val orderId: String?,
        val purchaseId: String,
        val productId: String,
        val invoiceId: String,
        val subscriptionToken: String? = null,
    ) : PaymentResult

    public data class Cancelled(
        val purchaseId: String,
    ) : PaymentResult

    public data class Failure(
        val purchaseId: String?,
        val invoiceId: String?,
        val orderId: String?,
        val quantity: Int?,
        val productId: String?,
        val errorCode: Int?,
    ) : PaymentResult

    public object InvalidPaymentState : PaymentResult()
}

```

where:

- Success — digital product purchased successfully.
- Failure — product purchase error.
- Cancelled — product purchase canceled.
- InvalidPaymentState — SDK error. Returned in case of deeplink processing errors.

Please note that the product purchase and cancellation scenario were successfully modified.

Product consumption scenario

The product purchase scenario was significantly changed. The purchase method can now return an error:

```
val purchasesUseCase: PurchasesUseCase =
    billingClient.purchases
    purchasesUseCase.confirmPurchase(purchaseId = "purchaseId",
    developerPayload = null)
        .addOnSuccessListener {
            // Process success
        }.addOnFailureListener { throwable: Throwable ->
            // Process error
        }
```

Product cancellation scenario

The product cancellation scenario was significantly changed. The cancellation method can now return an error:

```
val purchasesUseCase: PurchasesUseCase =
    billingClient.purchases
    purchasesUseCase.deletePurchase(purchaseId = "purchaseId")
        .addOnSuccessListener {
            // Process success
        }.addOnFailureListener { throwable: Throwable ->
            // Process error
        }
```

Product Purchase and Cancellation Scenario

Modifications in the product led to changes in the product purchase and cancellation scenario.

Use deletePurchase method if:

1. The `getPurchases` method returned an error with `PurchaseState.CREATED` or `PurchaseState.INVOICE_CREATED`.
2. The `purchaseProduct` method returned `PaymentResult.Cancelled`.
3. The `purchaseProduct` method returned `PaymentResult.Failure`.

Use `confirmPurchase` if `getPurchases` returned a `CONSUMABLE` type error with `PurchaseState.PAID` status.

Payment Errors FAQ

Q: How to fix the “Application is not verified yet” error?

A: This error occurs in the following cases:

- app review failed on [RuStore Console](#);
- a tested apk does not match the one uploaded on [RuStore Console](#).

The second point needs to be double-checked as follows:

- `applicationId` specified in `build.gradle` must match `applicationId` of the apk file that you published on [RuStore Console](#).
- keystore signature must match the signature that was used to sign the application published on [RuStore Console](#). Make sure that the `buildType` you use (eg `debug`) uses the same signature as the published application (eg `release`).

For payments to work properly, you need to publish the application in full, at that, it is not enough to pass the app review. Soon the logic will be redesigned so that review will be enough to test payments.

Q: How to fix the error “Previously created product purchase“...” in the amount of ... at the cost of ... rubles is paid in another session.”

A: The error occurs when attempting to purchase a product that has been terminated and not finalized using the `deletePurchase` and `confirmPurchase` methods.

This often occurs when a process was interrupted and `delete` or `consume` methods were not called in `purchaseProduct` due to the incorrectly terminated process.

For such cases, it is necessary to cancel or consume “suspended” purchases when starting the application or opening the store.

Below is an example of processing a shopping list. Run this code when the application starts or when the store screen opens:

```

val purchasesUseCase = billingClient.purchases

val purchases = purchasesUseCase.getPurchases().await().purchases.orEmpty()

purchases.forEach { purchase ->

    val purchaseId = purchase.purchaseId

    if (purchaseId != null) {

        when (purchase.purchaseState) {

            PurchaseState.CREATED, PurchaseState.INVOICE_CREATED -> {

                purchasesUseCase.deletePurchase(purchaseId).await()

            }

            PurchaseState.PAID -> {

                purchasesUseCase.confirmPurchase(purchaseId).await()

            }

            else -> Unit

        }

    }

}
}

```

You must also cancel or consume a purchase in `purchaseProduct()`.

Purchase processing in `purchaseProduct()`, which would terminate the purchase, confirming or canceling it, can be implemented as follows

```

private fun purchaseProduct(product: Product) {

    val purchasesUseCase = billingClient.purchases

    purchasesUseCase.purchaseProduct(product.productId)

        .addOnSuccessListener { paymentResult ->

            handlePaymentResult(paymentResult, product)

        }

        .addOnFailureListener {

            // Handle error

        }

}

private fun handlePaymentResult(paymentResult: PaymentResult, product: Product) {

    when (paymentResult) {

        is PaymentResult.InvalidPurchase -> {

            paymentResult.purchaseId?.let { deletePurchase(it) }

        }

        is PaymentResult.PurchaseResult -> {

            when (paymentResult.finishCode) {

                PaymentFinishCode.SUCCESSFUL_PAYMENT -> {

                    if (product.productType == ProductType.CONSUMABLE) {

```

```

        confirmPurchase(paymentResult.purchaseId)

    }

}

PaymentFinishCode.CLOSED_BY_USER,

PaymentFinishCode.UNHANDLED_FORM_ERROR,

PaymentFinishCode.PAYMENT_TIMEOUT,

PaymentFinishCode.DECLINED_BY_SERVER,

PaymentFinishCode.RESULT_UNKNOWN,

-> {

    deletePurchase(paymentResult.purchaseId)

}

}

}

else -> Unit

}

}

```

Read more in the “Consumption and purchase cancellation scenario” section.
 You can also view the status model of purchases in the “Getting a shopping list” section.

Q: How to perform server purchase validation?

A: First you need to get a subscriptionToken, which is a unique identifier for the user's purchase. Read more in the "Server purchase validation" section.

Next, you need to send the subscriptionToken to your backend, where you can request the purchase information using the "Get payments via its subscription token".

Q: How can I fix 404 when calling confirmPurchase or deletePurchase?

A: Make sure to pass the purchaseId as a parameter to the confirmPurchase and deletePurchase methods

```
val purchasesUseCase = billingClient.purchases

val purchases = purchasesUseCase.getPurchases().await().purchases.orEmpty()

purchases.forEach { purchase ->

    val purchaseId = purchase.purchaseId

    if (purchaseId != null) {

        when (purchase.purchaseState) {

            PurchaseState.CREATED, PurchaseState.INVOICE_CREATED -> {

                // purchasesUseCase.deletePurchase(purchaseId =
                purchase.productId).await() WRONG

                purchasesUseCase.deletePurchase(purchaseId = purchaseId).await() //
CORRECT

            }

            PurchaseState.PAID -> {

                // purchasesUseCase.confirmPurchase(purchaseId =
                purchase.productId).await() WRONG

                purchasesUseCase.confirmPurchase(purchaseId = purchaseId).await() //
CORRECT
```



```
    }  
    else -> Unit  
  }  
}  
}
```

Q: How to fix the "Method unavailable" error

A: consoleApplicationId must match the code on the RuStore Console (example: <https://console.rustore.ru/apps/111111>).

Q: How to cancel a subscription?

A: There is no method for canceling a subscription, you can only cancel auto-renewal in the RuStore app.

The subscription screen can be opened via deeplink:

```
startActivity(Intent(Intent.ACTION_VIEW, Uri.parse("rustore://profile/subscriptions")))
```

Below is a page with a list of deeplinks:

https://help.rustore.ru/rustore/for_developers/developer-documentation/RuStore_deeplinks

Q: Can I publish an app with RuStore SDK on Google Play, Huawei Store?

A: Since developers sign an application on Google Play with a Google signature, and on RuStore they sign it with their own one, the signing keys will always mismatch. Hence, RuStore SDK will not be available inside Google Play (or another application store). This means that for the RuStore SDK to work correctly, the user will need to download the application from RuStore and pay for a subscription there.

Q: What packageName does RuStore have?

A: ru.vk.store.

Q: How can I determine which store an app was installed from?

A: To do this, follow the steps below:

```
val installerPackage =  
packageManager.getInstallerPackageName(applicationInfo.packageName)
```

ru.vk.store is returned, but this function is unstable:

- This method will only work for applications originally installed from RuStore. If the application was originally installed from Google Play or other stores, then the source will be the standard package installer.
- If compatibility mode was used for installation (as on some Xiaomi models), then the installation source will be the Xiaomi system installer.
- If you delete RuStore, the installation source will be completely deleted. The installation source will not be returned even in case of subsequent installation.

We recommend creating a separate buildFlavor for RuStore.

Q: Why does the timeout drop in payment methods? (PayLibBackendFailure\$TimeoutError)

A: Payment via RuStore SDK is not available outside Russia. An enabled VPN may also interfere with the function.

Q: How to test payments? Can I use real cards?

A: You can only test payments using real cards. A sandbox is under development.

Q: Is there Java support?

A: Yes, there is. Kotlin is backward compatible with Java, but with some quirks.

For example, let's take the object entity from Kotlin, which is an analogue of the static class in Java:

```
RuStoreReviewManagerFactory.create(context)
```

Below is the request to RuStoreReviewManagerFactory in Java:

```
RuStoreReviewManagerFactory.INSTANCE.create(getContext());
```

RuStore SDK payments Release Notes

SDK 3.1.0

- Sandbox added;
- Internal SDK update

SDK 3.0.0

- Updates related to new dark mode settings;
- Fixed errors.

SDK version 2.2.0

- Added dynamic (light and dark) theme change function;
- Stabilized library performance;
- Fixed error related to payment via deeplink.

SDK version 2.1.2

- Fixed errors related to uninstalled RuStore app

SDK version 2.1.1

- Security practices updated

SDK version 2.1.0

- Modified response templates:
 - getting a list of products
 - getting a list of purchases
 - product purchase
 - purchase consumption
 - purchase cancellation
- Enhanced payment dialog appearance.

SDK version 1.1.0

- New payment method added: payment via TinkoffPay.
- New option: save card details during payment.
- Fixed: appearance and payment dialog improved.
- Fixed: unnecessary dependencies and uses-permissions removed.
- The PurchaseResult model supplemented with a new invoice ID field — invoiceId.

SDK version 1.0.0

- Moved from singleton to instance creation: RuStoreBillingClient.init() replaced with RuStoreBillingClientFactory.create().
- Singleton operation methods (init, products, purchases, getSingleton) marked as deprecated and are planned to be removed in further versions.
- To find out more about the changes, see the migration guide: "Migration to Payments SDK v1.0.0 and higher"
- The checkPurchasesAvailability() method became static — you can check for payments availability without creating a RuStoreBillingClient instance.

SDK version 0.1.7

- Internal SDK update.

SDK version 0.1.6

- Added the subscriptionToken field to the Purchase entity for server purchase validation.

SDK version 0.1.5

- Transition to Minciphra Certificates.
- Fixed payment button display via SBP when you quickly switch to an offer and back.
- The traceld: String field of the ResponseWithCode interface has been replaced by the meta: RequestMeta field, which contains traceld inside.

SDK version 0.1.4

- Added links to the offer for SBP and cell phone payment.
- Added additional disclaimer if payment confirmation takes more than 15 seconds.
- Fixed errors when working with OTP-codes and sending SMS.
- Fixed generation of orderId field for invoice on the RuStore if orderId is passed to purchaseProduct() as null.
- Added a blocking error message if the bank application does not support SBP deeplink.
- Added PurchaseResult.subscriptionToken, which ensures server purchase validation.

SDK version 0.1.3

- Added payment by phone number to purchaseProduct().

- In the initialization, the externalPaymentLogger parameter has been replaced by externalPayemtnLoggerFactory, intended for the ExternalPaymentLogger implementation (see Event Logging).

SDK version 0.1.2

- Added the Fast Payment System (SBP).
- Added the deeplinkScheme parameter to the init method.
- Removed deeplinkPrefix parameter from the init method.
- Fixed data loss bug in the card data entry form after minimizing the application.

SDK version 0.1.1

- Removed the "language" parameters for the following methods:
 - Getting a list of products — "getProducts".
 - Getting a list of purchases — "getPurchases".
 - Product confirmation — "confirmPurchase".
 - Product cancellation — "deletePurchase".
- Added the RuStoreBillingClient.isInitialized field, which returns the initialization status of the library.

SDK version 0.1.0

- The parameters of init have changed.
- Added check of payments availability — "checkPurchasesAvailability" method.
- In all methods, the "language" parameter has become optional.
- Suspend methods have been replaced by task API in the following methods:
 - Getting a list of products — "getProducts".
 - Getting a list of purchases — "getPurchases".
 - Product purchase — "purchaseProduct".
 - Product confirmation — "confirmPurchase".
 - Product cancellation — "deletePurchase".
- The "context" parameter has been removed from the "purchaseProduct" method.
- Removed the "resultObserver" method, now the purchase result is returned as "purchaseProduct".
- The "onFail" parameter is removed in the "resolveForBilling" method.

SDK version 0.0.9

- Added optional parameters "ExternalPaymentLogger" and "debugLogs" to the init function.

Godot

General Information

Example of implementation

Please have a thorough look at the [application example](#) to learn how to integrate payments correctly.

Payment integration guideline

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

Embed in your project

To connect SDK to your project, you need to download the following archive:
<https://gitflic.ru/project/rustore/godot-rustore-billing/release>.

Unpack the zip archive and add the files `rustore-billing.aar` and `RustoreBilling.gdap` to your project folder `/android/plugins`

```
example/  
  android/  
    plugins/  
      rustore-billing-release.aar  
      RustoreBilling.gdap
```

Handling deeplinks in your app

To redirect a user to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity
    android:name=".sample.MainActivity">

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE"
/>
        <data android:scheme="yourappscheme" />
    </intent-filter>

</activity>
```

where "yourappscheme" — your deeplink scheme, it can be changed to another one.

This scheme must match the deeplinkScheme parameter passed to init().

Exporting a project with an enabled plugin

To make a plugin available in your Godot project, it must be enabled in the export settings.

After all the settings are set, you can use the plugin in your project.

How to initialize the library

You must initialize the library to call its method. To do this, call the `init()` method:

```
if Engine.has_singleton("RustoreBilling"):  
    billing = Engine.get_singleton("RustoreBilling")  
  
    billing.init("123456", "yourappscheme://iamback")  
  
    billing.rustore_is_available.connect(_on_availability)  
    billing.rustore_purchase_product.connect(_on_purchase)  
    billing.rustore_delete_purchase.connect(_on_delete)  
    billing.rustore_confirm_purchase.connect(_on_confirm)  
    billing.rustore_get_purchases.connect(_on_get_purchases)  
    billing.rustore_get_purchase.connect(_on_get_purchase)  
    billing.rustore_get_products.connect(_on_get_products)
```

- 123456 — application code from RuStore Console (example: <https://console.rustore.ru/apps/123456>).
- `yourappscheme://iamback` — deeplink scheme required to return to your app page after payment through a third-party application (for example, SberPay or SBP). The SDK generates its own host for this scheme.

Make sure that the deeplink scheme passed to `deeplinkScheme` matches the scheme specified in `AndroidManifest.xml` in the "Deeplink Processing" section.

After the plugin is initialized, it will be connected to all available signals.

Payment functions availability

To check whether your app supports payment functions, the following conditions should be met:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all conditions are met, the `isAvailable()` method returns true.

```
func _availability():
    if billing != null:
        billing.isAvailable()
func _on_availability(data: Dictionary):
    if data['status'] == 'success':
        print('success')
        print(data['result'])
    elif data['status'] == 'failure':
        print('failure')
        print(data['message'])
```

The `_on_availability(data: Dictionary)` method is a `rustore_is_available` signal handler that receives a message about RuStore availability.

The `data['status']` key stores the request execution status. Possible values

- `success` — request successfully completed. In this case, the `data['result']` key will return true if RuStore is available and false if it is not available.
- `failure` — request error occurred. In the key `data['message']`

Getting the product list

Use the `getProducts` method to get a list of products:

```
func _get_products():
    if billing != null:
        billing.getProducts([
            "example1",
```

```

        "example2"
    ])

func _on_get_products(data: Dictionary):
    if data['status'] == 'success' and data.has('items'):
        var items = data['items']
        for key in items:
            print(items[key])
    elif data['status'] == 'failure':
        print('failure')
        print(data['message'])

```

The `_on_get_products(data: Dictionary)` method is a `rustore_get_products` signal handler that receives a message with a list of available products.

- `ids` — list of products IDs.

The `data['status']` key stores the request execution status. Possible values:

- `success` — request successfully executed. In this case, the `data['items']` key will store a list of available products.
- `failure` — request error occurred. The `data['message']` key stores the error message.

Available fields:

- `product_id` — product ID;
- `product_type` — product type;
- `product_status` — product status;
- `price_lable` — formatted product price, including currency symbol in `[language]`;
- `price` — price in minimum units (in kopecks);
- `currency` — currency code ISO 4217;
- `language` — language specified using BCP 47 encoding;
- `title` — product name in `[language]`;
- `description` — product description in `[language]`;
- `image_url` — image link
- `promo_image_url` — promotional image link;

- subscription — subscription description, returned only for subscription type products.

Fields available in subscription

- subscription_period — subscription period;
- free_trial_period — subscription trial period;
- grace_period — subscription grace period;
- introductory_price — formatted introductory subscription price, including currency sign, in product:language;
- introductory_price_amount — introductory price in minimum currency units (in kopecks);
- introductory_price_period — calculation period of the introductory price.

Fields available for subscription_period, free_trial_period, grace_period and introductory_price_period

- years — number of years;
- months — number of months;
- days — number of days.

How to get the user's list of products

Use the `getPurchases` method to get the user's list of purchases

```
func _get_purchases():
    if billing != null:
        billing.getPurchases()

func _on_get_purchases(data: Dictionary):
    if data['status'] == 'success' and data.has('items'):
        var items = data['items']
        for key in items:
            print(items[key])
    elif data['status'] == 'failure':
        print('failure')
        print(data['message'])
```

The `_on_get_purchases(data: Dictionary)` method is a `rustore_get_purchases` signal handler that receives a message with a list of available products.

The `data['status']` key stores the request execution status. Possible values:

- `success` — request successfully executed. In this case, the `data['items']` key will store a list of available products.
- `failure` — request error occurred. The `data['message']` key stores the error message.

Available fields:

- `purchase_id` — purchase ID;
- `product_id` — product ID;
- `product_type` — product type;
- `invoice_id` — account ID;
- `description` — purchase description;
- `language` — language specified using BCP 47 encoding;
- `purchase_time` — purchase time (in RFC 3339);
- `order_id` — unique payment ID generated by the application (uuid);
- `amount_table` — formatted purchase price, including currency symbol in `[language]`;
- `amount` — price in minimum currency units;

- currency — ISO 4217 currency code;
- quantity — quantity of product;
- purchase_state — purchase state;
- developer_payload — string specified by the developer containing additional information about the order;
- subscription_token — token for server purchase validation. For more information about validating a purchase on the server, see the “Server-based purchase validation” section.

Possible purchase status values:

- CREATED — created;
- INVOICE_CREATED — created, awaiting payment;
- CONFIRMED — confirmed;
- PAID — paid;
- CANCELLED — purchase canceled;
- CONSUMED — purchase confirmed;
- CLOSED — subscription canceled.

For more information about the purchasing status model, see the section “Getting a user’s shopping list.”

Getting purchase info

Use the `purchaseInfo()` method to get the user's list of purchases

```
func _purchase_info(id: String):
    if billing != null:
        billing.purchaseInfo(id)

func _on_get_purchase(data: Dictionary):
    if data['status'] == 'success':
        print('success')
        print(data['purchase'])
    elif data['status'] == 'failure':
        print('failure')
        print(data['message'])
```

The `_on_get_purchase(data: Dictionary)` method is a `rustore_get_purchase` signal handler that receives a message with a list of available products.

The `data['status']` key stores the request execution status. Possible values:

- `success` — request successfully executed. In this case, the `data['items']` key will store a list of available products.
- `failure` — request error occurred. The `data['message']` key stores the error message.

Available fields:

- `purchase_id` — purchase ID;
- `product_id` — product ID;
- `product_type` — product type;
- `invoice_id` — account ID;
- `description` — purchase description;
- `language` — language specified using BCP 47 encoding;
- `purchase_time` — purchase time (in RFC 3339);
- `order_id` — unique payment ID generated by the application (uuid);
- `amount_lable` — formatted purchase price, including currency symbol in `[language]`;

- amount — price in minimum currency units;
- currency — ISO 4217 currency code;
- quantity — quantity of product;
- purchase_state — purchase state;
- developer_payload — string specified by the developer containing additional information about the order;
- subscription_token — token for server purchase validation. For more information about validating a purchase on the server, see the “Server-based purchase validation” section.

Possible purchase status values:

- CREATED — created;
- INVOICE_CREATED — created, awaiting payment;
- CONFIRMED — confirmed;
- PAID — paid;
- CANCELLED — purchase canceled;
- CONSUMED — purchase confirmed;
- CLOSED — subscription canceled.

For more information about the purchasing status model, see the section “Getting a user’s shopping list.”

How to handle purchases

Use the `purchaseProduct(id)` method to call a product purchase:

```
func _purchase(id: String):
  var params = {
    "order_id": "1234",
    "quantity": 1,
    "payload": "example"
  }

  if billing != null:
    billing.purchaseProduct(id, params)
```

```
func _on_purchase(data: Dictionary):
  if data['status'] == 'cancelled':
    print('cancelled')
    if data['purchase'] != "":
      _delete(data['purchase'])
  elif data['status'] == 'success':
    print('success')
    print(data)
  elif data['status'] == 'failure':
    print('failure')
    print(data['message'])
```

The `_on_on_purchase(data: Dictionary)` method is a `rustore_purchase_product` signal handler that receives a message with a list of available products.

- `ids` — list of products IDs.
- `order_id` — order ID, created on the AnyApp side (optional. If not specified, it is generated automatically);
- `quantity` — number of products (optional);
- `payload` — additional information from the AnyApp developer (optional).

The `data['status']` key stores the request execution status. Possible values:

- `success` — request successfully executed. In this case, the `data['items']` key will store a list of available products.
- `failure` — request error occurred. The `data['message']` key stores the error message.
- `canceled` — user canceled the purchase.

Available fields:

- `product_id` — product ID;
- `order_id` — order ID, created on the AnyApp side (optional. If not specified, it is generated automatically);
- `invoice_id` — account ID;
- `product_id` — product ID;
- `quantity` — number of products (optional);
- `payload` — additional information from the AnyApp developer (optional).
- `error_code` — error code in case of failed request.

Purchase confirmation

The RuStore application consists of the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);
- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation if they are in the PurchaseState.PAID state.

You can use the `confirmPurchase(id)` method to confirm the purchase:

```
func _confirm(id: String):  
    var params = {  
        "payload": "123"  
    }  
  
    if billing != null:  
        billing.confirmPurchase(id, params)  
  
func _on_confirm(data: Dictionary):  
    print(data)  
    if data['status'] == 'success':  
        print('success')  
    elif data['status'] == 'failure':  
        print('failure')  
        print(data['message'])
```

The `_on_confirm(data: Dictionary)` method is a `rustore_get_purchase` signal handler that receives a message with purchase information.

- `id` — purchase ID.
- `payload` — string with additional information about the order (optional).

The data['status'] key stores the request execution status. Possible values:

- success — request successfully executed. In this case, the data['items'] key will store a list of available products.
- failure — request error occurred. The data['message'] key stores the error message.

Purchase cancellation

You can cancel purchases in your app.

You can use the `deletePurchase(id)` method to cancel the purchase:

```
func _delete(id: String):  
    if billing != null:  
        billing.deletePurchase(id)  
  
func _on_delete(data: Dictionary):  
    print(data)  
    if data['status'] == 'success':  
        print('success')  
    elif data['status'] == 'failure':  
        print('failure')  
        print(data['message'])
```

The `_on_delete(data: Dictionary)` method is a `rustore_get_purchase` signal handler that receives a message with purchase information.

- `id` — purchase ID.

The `data['status']` key stores the request execution status. Possible values:

- `success` — request successfully executed. In this case, the `data['items']` key will store a list of available products.
- `failure` — request error occurred. The `data['message']` key stores the error message.

Java

Quick Start	137
General Information	143
Payment functions availability	147
How to get the user's list of products	148
How to get the user's list of purchases	153
How to get purchase info	159
How to handle purchases	165
Server purchase validation	168
Purchase confirmation	169
Purchase cancellation	172
Consumption and cancellation scenario	175
Event Logging	176
Error handling	178
Migration to 1.0.0 purchase version	179

Quick Start

Payment integration guideline

Check out the [example application](#) (Kotlin) to learn how to properly integrate payments.

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. The user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions on work outside of Russia.

How to add a repository

Connect the repository:

```
repositories {
    maven {
        url
"https://artifactory-external.vkpartner.ru/artifactory/maven"
    }
}
```

Dependency injection

Add the following code to your configuration file to inject the dependency:

```
dependencies {
    implementation("ru.rustore.sdk:billingclient:5.0.0")
}
```

How to initialize a library

Initialize the library before calling its methods.

Create RuStoreBillingClient by using RuStoreBillingClientFactory.create():

```
final Context context = getContext();
```

```

final String consoleApplicationId = "111111";
final String deeplinkScheme = "yourappscheme";

final BillingClientThemeProvider themeProvider = null;
final boolean debugLogs = false;
final ExternalPaymentLoggerFactory externalPaymentLoggerFactory =
null;

RuStoreBillingClient billingClient =
RuStoreBillingClientFactory.INSTANCE.create(
    context,
    consoleApplicationId,
    deeplinkScheme,
    themeProvider,
    debugLogs,
    externalPaymentLoggerFactory
);

```

- context — Android context. Any context is allowed, applicationContext is used in the release version.
- consoleApplicationId — application code from the RuStore Developer Console (example: <https://console.rustore.ru/apps/111111>).
- deeplinkScheme — deeplink scheme required to return to your app upon payment via a third-party application (for example, SberPay or SBP). SDK generates its host for this scheme.
- themeProvider — interface that provides BillingClientTheme. There are 2 possible implementations of BillingClientTheme: light and dark. This interface is optional; the default is a light theme.
- externalPaymentLoggerFactory — interface that provides access to an external logger.
- debugLogs — flag that regulates logging (logs will be automatically disabled for Release builds).

The ApplicationId specified in build.gradle must match the applicationId of the apk file you published to the [RuStore Console](#).

The keystore signature must be the same as the one used to sign the application published to the [RuStore Console](#) system. Make sure that the buildType uses (eg: debug) uses the same signature as the published application (eg: release).

The library supports event logging, which is enabled separately when the library is initialized.

For details on library initialization, read the above information.

Handling deeplinks in your app

To redirect a user back to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity
    android:name=".YourBillingActivity">

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE"
/>
        <data android:scheme="yourappscheme" />
    </intent-filter>

</activity>
```

where "yourappscheme" — your deeplink scheme, it can be changed to another one. This scheme must match the deeplinkScheme parameter passed to init().

You also need to add the following code to the application to ensure a successful return:

```
public class YourBillingActivity extends AppCompatActivity {

    // Previously created with RuStoreBillingClientFactory.create();
    RuStoreBillingClient billingClient =
    YourDependencyInjection.getBillingClient();

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
```



```

        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
            billingClient.onNewIntent(getIntent());
        }
    }

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

```

How to get a list of purchases

RuStore Billing SDK requires proper processing of purchases to provide the best-case scenario. For example, purchased items need to be consumed and unfinished purchases need to be canceled to ensure that a user can start a new one all over again.

```

PurchasesUseCase purchasesUseCase = billingClient.getPurchases();
purchasesUseCase.getPurchases().addOnSuccessListener(purchases -> {
    for (Purchase purchase: purchases) {
        if (purchase.getPurchaseId() != null) {
            if (purchase.getPurchaseState() == PurchaseState.CREATED
|| purchase.getPurchaseState() == PurchaseState.INVOICE_CREATED) {

purchasesUseCase.deletePurchase(purchase.getPurchaseId());
                } else if (purchase.getPurchaseState() ==
PurchaseState.PAID) {

purchasesUseCase.confirmPurchase(purchase.getPurchaseId());
                    }
                }
            }
        });
}

```

How to handle purchases

Process the purchase result as follows:

```

private void purchaseProduct(Product product) {

```

```

        PurchasesUseCase purchasesUseCase =
billingClient.getPurchases();
        purchasesUseCase.purchaseProduct(product.getProductId())
            .addOnSuccessListener(paymentResult ->
handlePaymentResult(paymentResult, product))
            .addOnFailureListener(throwable -> { /* Handle error */
});
    }

```

```

private void handlePaymentResult(PaymentResult paymentResult,
Product product) {
    PurchasesUseCase purchasesUseCase =
billingClient.getPurchases();
    if (paymentResult instanceof PaymentResult.Cancelled) {
        String purchaseId = ((PaymentResult.Cancelled)
paymentResult).getPurchaseId();
        purchasesUseCase.deletePurchase(purchaseId);
    } else if (paymentResult instanceof PaymentResult.Success) {
        PaymentResult.Success purchaseResult =
((PaymentResult.Success) paymentResult);

purchasesUseCase.confirmPurchase(purchaseResult.getPurchaseId());
    } else if (paymentResult instanceof PaymentResult.Failure) {
        String purchaseId = ((PaymentResult.Failure)
paymentResult).getPurchaseId();
        if (purchaseId != null) {
            purchasesUseCase.deletePurchase(purchaseId);
        }
    }
}

```

General Information

You can see the "Quick Start" section to quickly integrate payments into the app.

Example of implementation

Please have a thorough look at the [application example](#) to learn how to integrate payments correctly.

Payment integration guideline

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your app user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

How to add a repository

Connect the repository:

```
repositories {
    maven {
        url
"https://artifactory-external.vkpartner.ru/artifactory/maven"
    }
}
```

Dependency injection

Add the following code to your configuration file to inject the dependency:

```
dependencies {
    implementation("ru.rustore.sdk:billingclient:5.0.0")
}
```

How to initialize a library

Initialize the library before calling its methods.

Create RuStoreBillingClient by using RuStoreBillingClientFactory.create():

```
final Context context = getContext();
```

```

final String consoleApplicationId = "111111";
final String deeplinkScheme = "yourappscheme";

final BillingClientThemeProvider themeProvider = null;
final boolean debugLogs = false;
final ExternalPaymentLoggerFactory externalPaymentLoggerFactory =
null;

RuStoreBillingClient billingClient =
RuStoreBillingClientFactory.INSTANCE.create(
    context,
    consoleApplicationId,
    deeplinkScheme,
    themeProvider,
    debugLogs,
    externalPaymentLoggerFactory
);

```

- context — Android context. Any context is allowed, applicationContext is used in the release version.
- consoleApplicationId — application code from the RuStore Developer Console (example: <https://console.rustore.ru/apps/111111>).
- deeplinkScheme — deeplink scheme required to return to your app upon payment via a third-party application (for example, SberPay or SBP). SDK generates its host for this scheme.
- themeProvider — interface that provides BillingClientTheme. There are 2 possible implementations of BillingClientTheme: light and dark. This interface is optional; the default is a light theme.
- externalPaymentLoggerFactory — interface that provides access to an external logger.
- debugLogs — flag that regulates logging (logs will be automatically disabled for Release builds).

Make sure that the deeplink scheme passed to deeplinkScheme matches the one specified in AndroidManifest.xml under "Handling deeplinks in your app".

The library supports event logging which is plugged in separately when the library is initialized.

Handling deeplinks in your app

To redirect a user to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity
    android:name=".YourBillingActivity">

    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="yourappscheme" />
    </intent-filter>

</activity>
```

where "yourappscheme" — your deeplink scheme, it can be changed to another one.

This scheme must match the deeplinkScheme parameter passed to init().

Next, add the following code to the Activity you need to return to after making the payment (your store page):

```
public class YourBillingActivity extends AppCompatActivity {

    // Previously created with RuStoreBillingClientFactory.create();
    RuStoreBillingClient billingClient =
    YourDependencyInjection.getBillingClient();

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (savedInstanceState == null) {
            billingClient.onNewIntent(getIntent());
        }
    }

    @Override
```

```
protected void onNewIntent(Intent intent) {  
    super.onNewIntent(intent);  
    billingClient.onNewIntent(intent);  
}  
}
```

To restore your app after deep linking, you need to add the following to AndroidManifest.xml:

```
android:launchMode="singleTask"
```

```
<activity  
    android:name=".YourBillingActivity"  
    android:launchMode="singleTask"  
    android:exported="true"  
    android:screenOrientation="portrait"  
    android:windowSoftInputMode="adjustResize">
```

Payment functions availability

To check whether your app supports payment functions, call the `checkPurchasesAvailability` method. This method checks the following conditions:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. The app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all conditions are met, the method returns `FeatureAvailabilityResult.Available`. Otherwise, it returns `FeatureAvailabilityResult.Unavailable(val cause: RuStoreException)`, where *cause* indicates an unfulfilled condition. All possible `RuStoreException` errors are described in [Error Handling](#). Other errors (e.g. "No internet connection") are processed in `onFailure`.

```
RuStoreBillingClientExtKt.checkPurchasesAvailability(RuStoreBilling
Client.Companion, getContext())
    .addOnSuccessListener(result -> {
        if (result instanceof
FeatureAvailabilityResult.Available) {
            // Handle purchases available
        } else {
            RuStoreException exception =
((FeatureAvailabilityResult.Unavailable) result).getCause();
            // Handle purchases unavailable
        }
    })
    .addOnFailureListener(error -> {
        // Handle error
    });
```

How to get the user's list of products

Use the `getProducts` method to get the user's list of products

```

ProductsUseCase productsUseCase = billingClient.getProducts();
productsUseCase.getProducts(Arrays.asList("id1",
"id2")).addOnCompleteListener(new
OnCompleteListener<List<Product>>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(List<Product> products) {
        // Process success
    }
});

```

- productIds: List<String> - list of product identifications.

The method returns:

```

interface Product {
    public String getProductId();

    @Nullable
    public ProductType getProductType();

    public ProductStatus getProductStatus();

    @Nullable
    public String getPriceLabel();

    @Nullable
    public Integer getPrice();

    @Nullable
    public String getCurrency();

    @Nullable

```



```

public String getLanguage();

@Nullable
public String getTitle();

@Nullable
public String getDescription();

@Nullable
public Uri getImageUrl();

@Nullable
public Uri getPromoImageUrl();

@Nullable
public ProductSubscription getProductSubscription();
}

```

- `getProductId()` — product identifier;
- `getProductType()` — product type;
- `getProductStatus()` — product status;
- `getPriceLabel()` — formatted product price, including currency symbol in `[language]`;
- `getPrice()` — price in minimum units (kopecks);
- `getCurrency()` — ISO 4217 currency code;
- `getLanguage()` — language specified using BCP 47 encoding;
- `getTitle()` — product title in `[language]`;
- `getDescription()` — product description in `[language]`;
- `getImageUrl()` — link to the image;
- `getPromoImageUrl()` — link to the promotional image;
- `getSubscription()` — subscription's description, returned only for products with the subscription type.

Subscription structure:

```

interface ProductSubscription {
    @Nullable
    public SubscriptionPeriod getSubscriptionPeriod();

    @Nullable

```

```

public SubscriptionPeriod getFreeTrialPeriod();

@Nullable
public SubscriptionPeriod getGracePeriod();

@Nullable
public String getIntroductoryPrice();

@Nullable
public String getIntroductoryPriceAmount();

@Nullable
public String getIntroductoryPricePeriod();
}

```

- `getsubscriptionPeriod` — subscription period;
- `getfreeTrialPeriod` — trial subscription period;
- `getgracePeriod` — subscription grace period;
- `getintroductoryPrice` — formatted introductory subscription price, including the currency symbol, in `product:language`;
- `getintroductoryPriceAmount` — introductory price in minor currency units (in kopecks);
- `getintroductoryPricePeriod` — introductory price settlement period.

Subscription period structure:

```

interface SubscriptionPeriod {
    int getYears();

    int getMonths();

    int getDays();
}

```

- `years` — number of years;
- `months` — number of months;
- `days` — number of days.

How to get the user's list of purchases

Use the `getPurchases` method to get the user's list of purchases:

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();
purchasesUseCase.getPurchases().addOnCompleteListener(new
OnCompleteListener<PurchasesResponse>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(PurchasesResponse purchasesResponse) {
        // Process success
    }
});
```

Purchase Structure:

```
interface Purchase {
    @Nullable
    public String getPurchaseId();

    public String getProductId();

    @Nullable
    public ProductType getProductType();

    @Nullable
    public String getInvoiceId();

    @Nullable
    public String getDescription();

    @Nullable
    public String getLanguage();

    @Nullable
```

```

public Date getPurchaseTime();

@Nullable
public String getOrderId();

@Nullable
public String getAmountLabel();

@Nullable
public Integer getAmount();

@Nullable
public String getCurrency();

@Nullable
public Integer getQuantity();

@Nullable
public PurchaseState getPurchaseState();

@Nullable
public String getDeveloperPayload();

@Nullable
public String getSubscriptionToken();
}

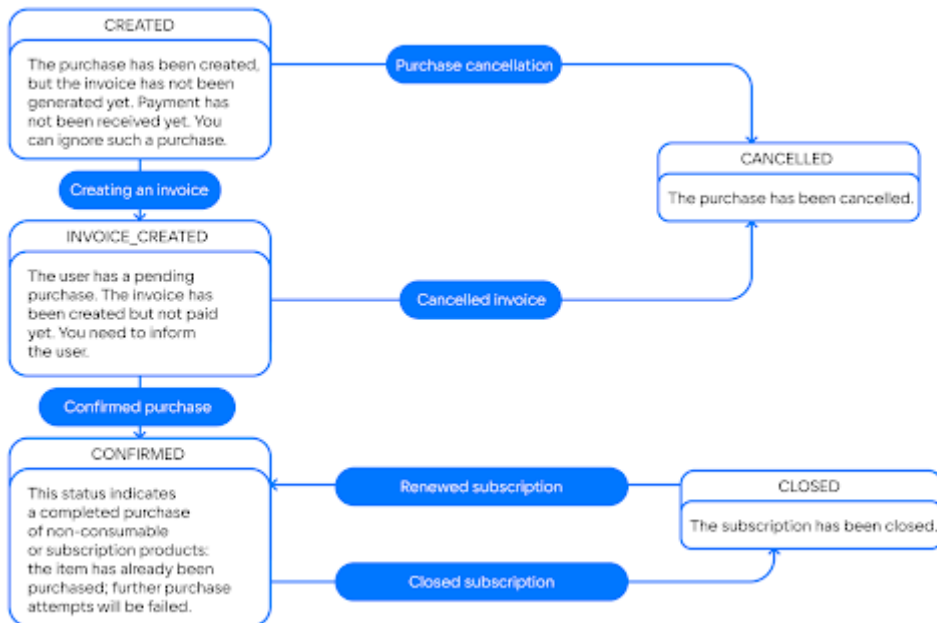
```

- `getPurchaseId()` — purchase ID;
- `getPurchaseId()`— product identifier;
- `getProductType()` — product type;
- `getInvoiceId()` — invoice ID;
- `getDescription()` — purchase description;
- `getLanguage()` — language specified with the BCP 47 encoding;
- `getPurchaseTime()` — purchase time (in RFC 3339 format);
- `getOrderId()` — unique payment identifier generated by the application (uuid);
- `getAmountLabel()` — formatted purchase price, including the currency symbol in [language];
- `getAmount()` — price in minor units of currency;
- `getCurrency()` — ISO 4217 currency code;
- `getQuantity()` — number of products;
- `getPurchaseState()` — purchase status:

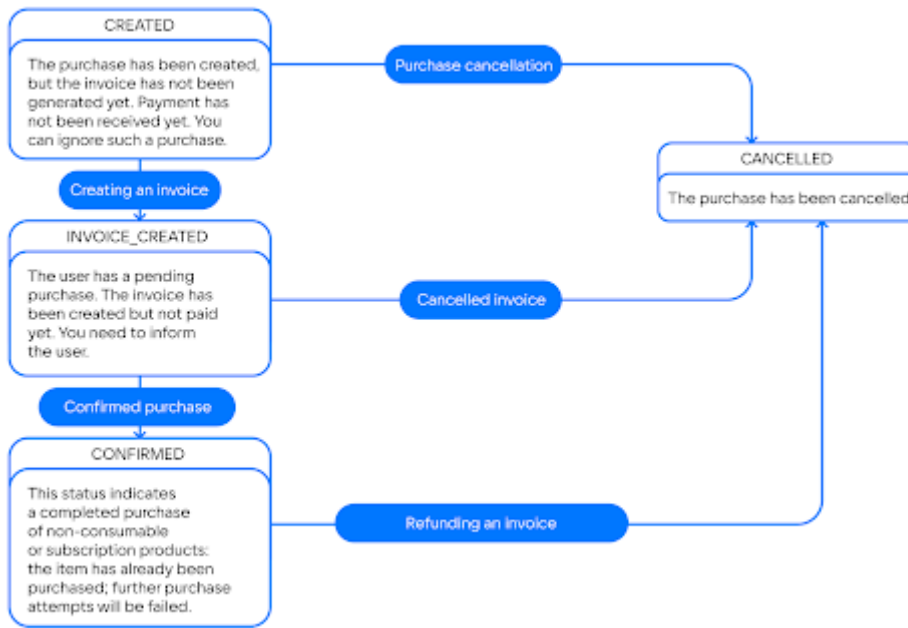
- possible values of the purchase condition:
 - CREATED — created;
 - INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;
 - CANCELLED — purchase canceled;
 - CONSUMED — purchase consumption is confirmed;
 - CLOSED — subscription is canceled.
- `getDeveloperPayload()` — line specified by the developer that contains additional information about the order;
- `getSubscriptionToken()` — token for validating a purchase on the server.

The purchaseState model:

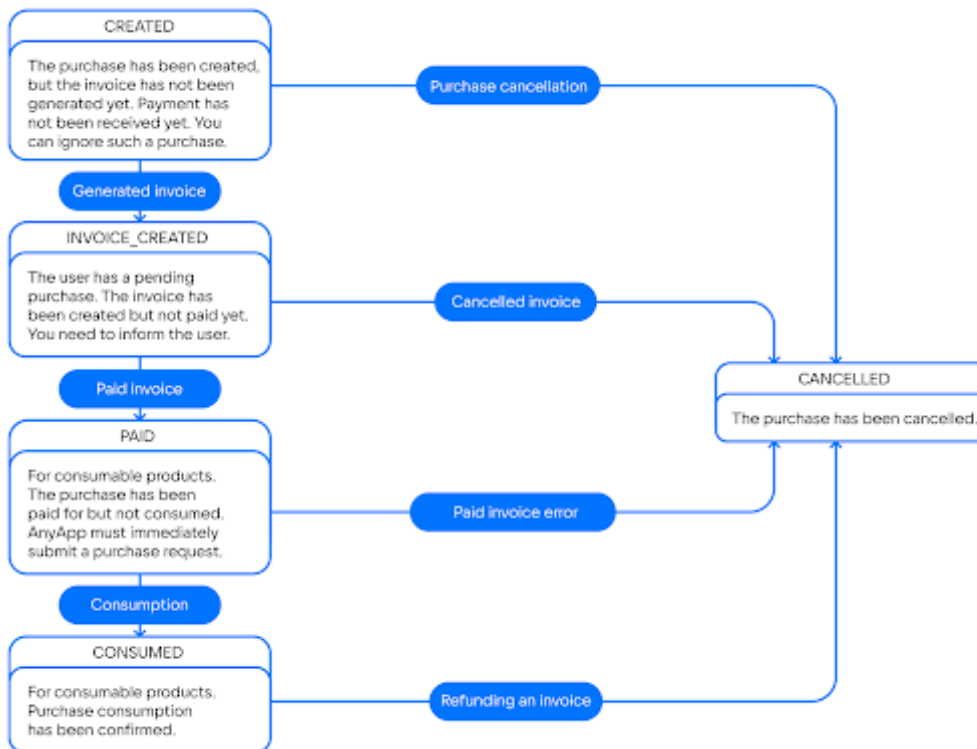
A status-based subscription purchase model (SUBSCRIPTIONS):



A status-based non-consumables subscription (NON-CONSUMABLES):



A status-based consumables subscription (CONSUMABLES):



How to get purchase info

Use the `getPurchaseInfo` method to get information about purchases:

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();
purchasesUseCase.getPurchaseInfo("purchaseId").addOnCompleteListener(
    new OnCompleteListener<PurchaseInfoResponse>() {
        @Override
        public void onFailure(@NonNull Throwable throwable) {
            // Process error
        }

        @Override
        public void onSuccess(PurchaseInfoResponse result) {
            // Process success
        }
    }
);
```

Product Structure:

```
interface Purchase {
    @Nullable
    public String getPurchaseId();

    public String getProductId();

    @Nullable
    public ProductType getProductType();

    @Nullable
    public String getInvoiceId();

    @Nullable
    public String getDescription();

    @Nullable
    public String getLanguage();

    @Nullable
    public Date getPurchaseTime();
}
```

```

@Nullable
public String getOrderId();

@Nullable
public String getAmountLabel();

@Nullable
public Integer getAmount();

@Nullable
public String getCurrency();

@Nullable
public Integer getQuantity();

@Nullable
public PurchaseState getPurchaseState();

@Nullable
public String getDeveloperPayload();

@Nullable
public String getSubscriptionToken();
}

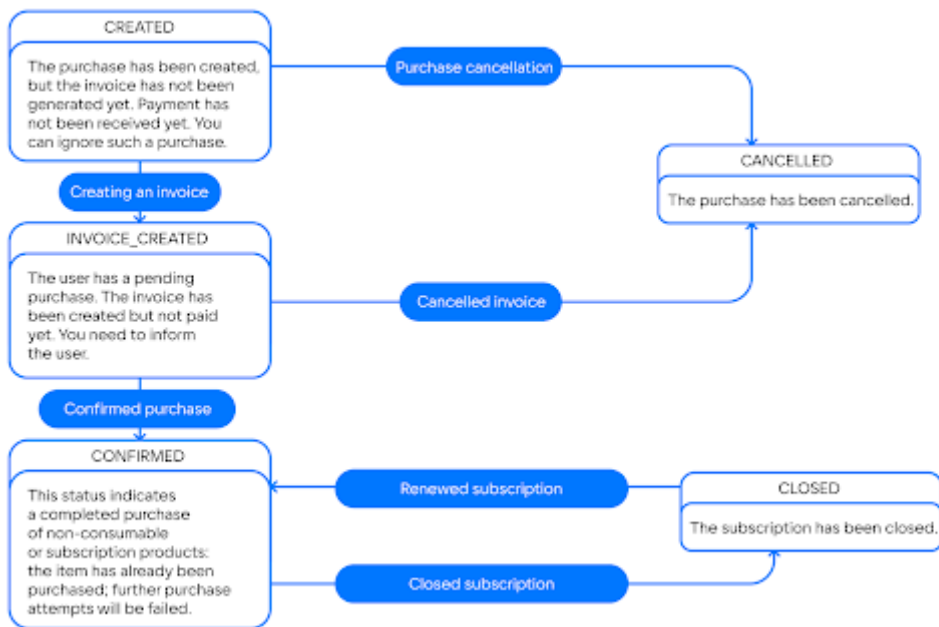
```

- `getPurchaseId()` — purchase ID;
- `getProductId()` — product identifier;
- `getProductType()` — product type;
- `getInvoiceId()` — invoice ID;
- `getDescription()` — purchase description;
- `getLanguage()` — language specified with the BCP 47 encoding;
- `getPurchaseTime()` — purchase time (in RFC 3339 format);
- `getOrderId()` — unique payment identifier generated by the application (uuid);
- `getAmountLabel()` — formatted purchase price, including the currency symbol in [language];
- `getAmount()` — price in minor units of currency;
- `getCurrency()` — ISO 4217 currency code;
- `getQuantity()` — number of products;
- `getPurchaseState()` — purchase status:
 - possible values of the purchase condition:
 - `CREATED` — created;

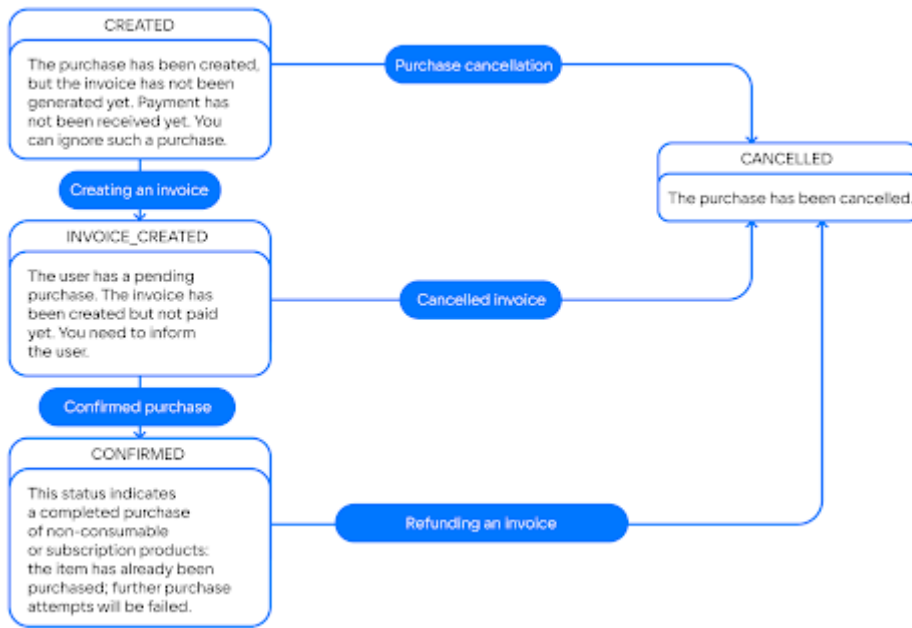
- INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;
 - CANCELLED — purchase canceled;
 - CONSUMED — purchase consumption is confirmed;
 - CLOSED — subscription is canceled.
- getDeveloperPayload() — line specified by the developer that contains additional information about the order;
 - getDeveloperPayload() — token for server purchase validation.

The purchaseState model:

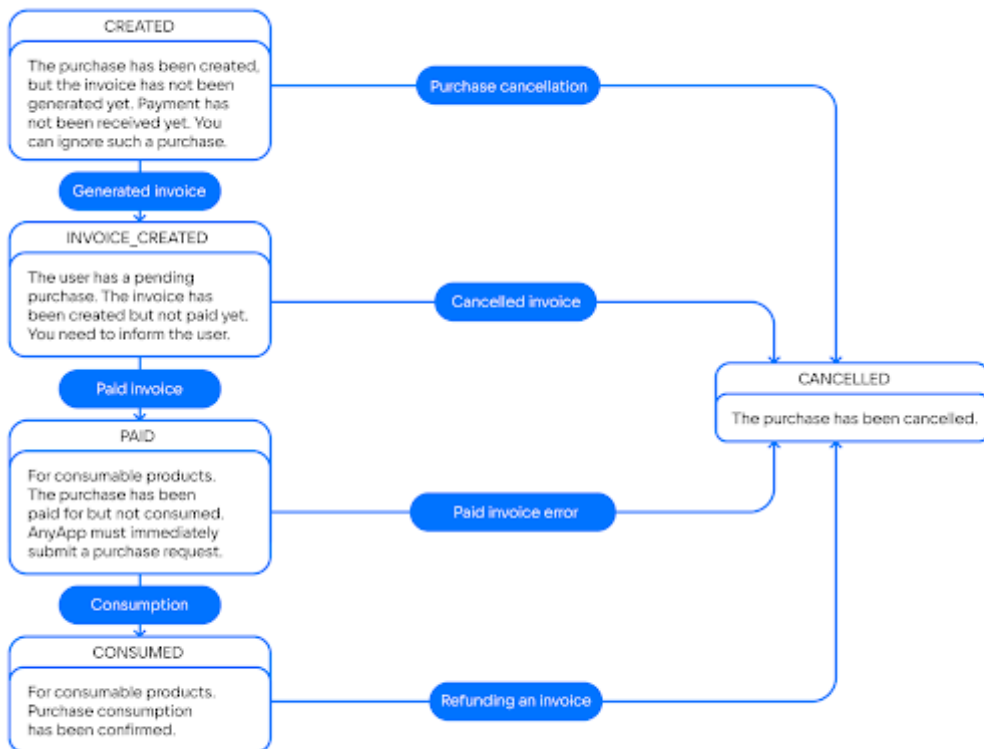
A status-based subscription purchase model (SUBSCRIPTIONS):



A status-based non-consumables subscription (NON-CONSUMABLES):



A status-based consumables subscription (CONSUMABLES):



How to handle purchases

Use the `purchaseProduct` method to call a product purchase:

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();
purchasesUseCase.purchaseProduct("productId").addOnCompleteListener
(new OnCompleteListener<PaymentResult>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(PaymentResult paymentResult) {
        // Process PaymentResult
    }
});
```

- `productId`: String — product ID;
- `orderId`: String — order ID, generated by AnyApp (optional. If not specified, it is generated automatically);
- `quantity`: Int — number of products (optional);
- `developerPayload` — additional information from the AnyApp developer (optional).

Payment result structure:

```
interface PaymentResult {

    interface Success extends PaymentResult {
        @Nullable
        public String getOrderId();

        public String getPurchaseId();

        public String getProductId();

        public String getInvoiceId();

        @Nullable
```

```

        public String getSubscriptionToken();
    }

    interface Failure extends PaymentResult {
        @Nullable
        public String getPurchaseId();
        @Nullable
        public String getInvoiceId();
        @Nullable
        public String getOrderId();
        @Nullable
        public Integer getQuantity();
        @Nullable
        public String getProductId();
        @Nullable
        public Integer getErrorCode();
    }

    interface Cancelled extends PaymentResult {
        public String getPurchaseId();
    }

    interface InvalidPaymentState extends PaymentResult {}
}

```

- Success — product successfully purchased;
- Failure — product purchase error;
- Canceled — purchase cancelled;
- InvalidPaymentState — payment SDK error. May occur in case of incorrect reverse deeplink.

Server purchase validation

For server purchase validation you can use the subscriptionToken in the PurchaseResult returned by purchaseProduct in case of a successful purchase.

SubscriptionToken consists of invoiceId of purchase and userId of RuStore, written with a dot: "\$invoiceId.\$userId"

```

RuStoreBillingClient.INSTANCE.getPurchases().purchaseProduct(productId).addOnSuccessListener(paymentResult -> {
    if (paymentResult instanceof PaymentResult.PurchaseResult) {
        String subscriptionToken = ((PaymentResult.PurchaseResult)
paymentResult).getSubscriptionToken();
        yourApi.validate(subscriptionToken);
    }
});

```

You can also get a subscriptionToken in the Purchase entity. The Purchase entity can be retrieved using the getPurchases() method:

```

RuStoreBillingClient.INSTANCE.getPurchases().getPurchases().addOnSuccessListener(purchasesResponse -> {
    for (Purchase purchase : purchasesResponse.getPurchases()) {
        yourApi.validate(purchase.getSubscriptionToken());
    }
});

```

Purchase confirmation

RuStore consists of the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);
- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation if they are in the PurchaseState.PAID state.

You can use the confirmPurchase method to confirm the purchase:

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();
purchasesUseCase.confirmPurchase("purchaseId",
"developerPayload").addOnCompleteListener(new
OnCompleteListener<Unit>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(Unit result) {
        // Process success
    }
});
```

- purchaseId — purchase ID;
- developerPayload — line specified by the developer that contains additional information about the order (optional).

Purchase cancellation

You can use the deletePurchase method to cancel the purchase:

```
PurchasesUseCase purchasesUseCase = billingClient.getPurchases();
purchasesUseCase.deletePurchase("purchaseId").addOnCompleteListener
(new OnCompleteListener<Unit>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(Unit result) {
        // Process success
    }
});
```

- purchaseId — purchase ID.

Note. Use this method if your app logic is related to purchase cancellation. The purchase is canceled automatically after a 20-min timeout, or upon a second purchase from the same customer.

Consumption and cancellation scenario

Uncompleted payments must be processed by the AnyApp developer.

The purchase cancellation method (`deletePurchase`) should be used if:

1. The method of getting the list of products (`getPurchases`) returned the purchase status as follows:
 - `PurchaseState.CREATED`;
 - `PurchaseState.INVOICE_CREATED`;

Note. In some cases, after paying through a banking app (SBP, SberPay, TinkoffPay, etc.), the purchase status may still return `PurchaseState.INVOICE_CREATED` when you subsequently return to AnyApp. This is caused by the purchase processing time by the bank. Therefore, the developer needs to correctly link the shopping list obtaining function to the life cycle on the screen.

To solve this problem, you can cancel a purchase in the `PurchaseState.INVOICE_CREATED` status only through user interaction with the application. For example, create a separate button for this purpose.

2. The purchase method (`purchaseProduct`) returned `PaymentResult.Cancelled`.
3. The purchase method (`purchaseProduct`) returned `PaymentResult.Failure`.

Use product consumption method (`confirmPurchase`) if the method the purchase obtaining method (`getPurchases`) returns a `CONSUMABLE` product and with the status `PurchaseState.PAID`.

Event Logging

If you want to log payment library events add `externalPaymentLoggerFactory` and `debugLogs` parameters (these parameters are optional for initialization) when calling `RuStoreBillingClient.init`:

```
public class App extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        final Application application = this;
        final String consoleApplicationId = "111111";
        final String deeplinkScheme = "yourappscheme";
        final ExternalPaymentLoggerFactory
externalPaymentLoggerFactory = (tag) -> new PaymentLogger(tag);
        final boolean debugLogs = true;
        RuStoreBillingClient.INSTANCE.init(application,
consoleApplicationId, deeplinkScheme, externalPaymentLoggerFactory,
debugLogs);
    }

    public class PaymentLogger implements ExternalPaymentLogger {

        private final String tag;

        public PaymentLogger(String tag) {
            this.tag = tag;
        }

        @Override
        public void d(@Nullable Throwable throwable, @NonNull
Function0<String> function0) {
            Log.d(tag, function0.invoke());
        }

        @Override
        public void e(@Nullable Throwable throwable, @NonNull
Function0<String> function0) {
            Log.e(tag, function0.invoke());
        }
    }
}
```

```

        @Override
        public void i(@Nullable Throwable throwable, @NonNull
Function0<String> function0) {
            Log.i(tag, function0.invoke());
        }

        @Override
        public void v(@Nullable Throwable throwable, @NonNull
Function0<String> function0) {
            Log.v(tag, function0.invoke());
        }

        @Override
        public void w(@Nullable Throwable throwable, @NonNull
Function0<String> function0) {
            Log.w(tag, function0.invoke());
        }
    }
}

```

Logging processing parameters:

- externalPaymentLoggerFactory — interface that allows you to create a logger that sends the library logs to the host application;
- debugLogs — enables logs (logs will be automatically disabled for Release builds).

PaymentLogger — example of payment event logging implementation.

Theme Changing

The SDK supports dynamic theme changing through the `BillingClientThemeProvider` provider interface:

```
final Context context = getContext();
final String consoleApplicationId = "111111";
final String deeplinkScheme = "yourappscheme";
final BillingClientThemeProvider themeProvider = BillingClientThemeProviderImpl();

RuStoreBillingClient billingClient = RuStoreBillingClientFactory.INSTANCE.create(
    context,
    consoleApplicationId,
    deeplinkScheme,
    themeProvider
);

public class BillingClientThemeProviderImpl implements BillingClientThemeProvider {

    @NonNull
    @Override
    public BillingClientTheme provide() {
        boolean darkTheme = ...;
        if (darkTheme) {
            return BillingClientTheme.Dark;
        } else {
            return BillingClientTheme.Light;
        }
    }
}
```

Error handling

Possible errors:

- `RuStoreNotInstalledException()` — RuStore is not installed on the user's device;
- `RuStoreOutdatedException()` — RuStore, installed on the user's device, does not support payment processing functions;
- `RuStoreUserUnauthorizedException()` — user is not authorized on the RuStore;
- `RuStoreApplicationBannedException()` — your application is blocked on the RuStore;
- `RuStoreUserBannedException()` — user is blocked on the RuStore;
- `RuStoreException(message: String)` — basic RuStore error, from which all other errors are inherited.

When calling the `RuStoreBillingClient.purchases.purchaseProduct()` method, errors are handled automatically.

You can use the `resolveForBilling` method to show an error dialog to the user:

```
BillingRuStoreExceptionExtKt.resolveForBilling(exception,  
getContext());
```

Migration to Payments SDK from v.1.x.x to v3.x.x

General

In version 3.0.0, the purchase model in `PaymentResult` was significantly modified.

Follow the steps below to switch to the new SDK version smoothly.

Dependency update

To update the dependency, call the `billingclient` version in the `dependencies` block of your `build.gradle`:

`build.gradle`

```
dependencies {  
    implementation("ru.rustore.sdk:billingclient:3.0.0")  
}
```

Model update

Getting List of Products

The products list model was significantly modified. From now on **getProducts()** returns the list of products:

```
ProductsUseCase productsUseCase = billingClient.getProducts();
productsUseCase.getProducts(Arrays.asList("id1",
"id2")).addOnCompleteListener(new
OnCompleteListener<List<Product>>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(List<Product> products) {
        // Process success
    }
});
```

At that, the product and the error model remained unchanged.

Getting List of Purchases

The list of purchases model was significantly modified. From now on, **getPurchases()** returns the list of purchases:

```
PurchasesUseCase purchasesUseCase =
billingClient.getPurchases();
purchasesUseCase.getPurchases().addOnCompleteListener(new
OnCompleteListener<List<Purchase>>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(List<Purchase> purchases) {
        // Process success
    }
});
```

At that, the product and the error model remained unchanged.

Getting Product Info

The purchase info method was significantly modified. From now on, **getPurchaseInfo()** returns the purchase method:

```
PurchasesUseCase purchasesUseCase =
billingClient.getPurchases();
purchasesUseCase.getPurchaseInfo("purchaseId").addOnCompleteListener(
new OnCompleteListener<Purchase>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(Purchase purchase) {
        // Process success
    }
});
```

The error model remained unchanged.

Purchases

In this version, the purchase model was also modified. The new model is represented as follows:


```
interface PaymentResult {

    interface Success extends PaymentResult {
        @Nullable
        public String getOrderId();

        public String getPurchaseId();

        public String getProductId();

        public String getInvoiceId();

        @Nullable
        public String getSubscriptionToken();
    }

    interface Failure extends PaymentResult {
        @Nullable
        public String getPurchaseId();

        @Nullable
        public String getInvoiceId();

        @Nullable
        public String getOrderId();

        @Nullable
        public Integer getQuantity();

        @Nullable
        public String getProductId();

        @Nullable
        public Integer getErrorCode();
    }

    interface Cancelled extends PaymentResult {
        public String getPurchaseId();
    }
}
```

```
interface InvalidPaymentState extends PaymentResult {}  
}
```

where:

- Success — digital product purchased successfully.
- Failure — product purchase error.
- Cancelled — product purchase canceled.
- InvalidPaymentState — SDK error. Returned in case of deeplink processing errors.

Please note that the product purchase and cancellation scenario were successfully modified.

Product consumption scenario

The product purchase scenario was significantly changed. The purchase method can now return an error:

```
PurchasesUseCase purchasesUseCase =  
billingClient.getPurchases();  
purchasesUseCase.confirmPurchase("purchaseId",  
"developerPayload").addOnCompleteListener(new  
OnCompleteListener<Unit>() {  
    @Override  
    public void onFailure(@NonNull Throwable throwable) {  
        // Process error  
    }  
  
    @Override  
    public void onSuccess(Unit result) {  
        // Process success  
    }  
});
```

Product cancellation scenario

The product cancellation scenario was significantly changed. The cancellation method can now return an error:

```
PurchasesUseCase purchasesUseCase =
billingClient.getPurchases();
purchasesUseCase.deletePurchase("purchaseId").addOnCompleteListener(
new OnCompleteListener<Unit>() {
    @Override
    public void onFailure(@NonNull Throwable throwable) {
        // Process error
    }

    @Override
    public void onSuccess(Unit result) {
        // Process success
    }
});
```

Product Purchase and Cancellation Scenario

Modifications in the product led to changes in the product purchase and cancellation scenario.

Use deletePurchase method if:

1. The getPurchases method returned an error with the following status:
PurchaseState.CREATED or PurchaseState.INVOICE_CREATED.

Note. In some cases, after paying through a banking app (SBP, SberPay, TinkoffPay, etc.), the purchase status may still return PurchaseState.INVOICE_CREATED when you subsequently return to AnyApp. This is caused by the purchase processing time by the bank. Therefore, the developer needs to correctly link the shopping list obtaining function to the life cycle on the screen.

2. The purchaseProduct method returned PaymentResult.Cancelled.

3. The purchaseProduct method returned PaymentResult.Failure.

Use confirmPurchase if getPurchases returned a CONSUMABLE type error with PurchaseState.PAID status.

RuStore SDK payments Release Notes

SDK version 2.2.0

- Added dynamic theme change functionality (light and dark);
- Stabilized library;
- Fixed deeplink payment problem.

SDK version 2.1.1

- Security updates.

SDK version 2.1.0

- Changed response models:
 - getting a list of products
 - getting a shopping list
 - product purchase
 - purchases consumption
 - purchases cancellation
- Improved payment dialog.

SDK version 1.1.1

- Fixed await() method for Task API.

SDK version 1.1.0

- Payment via TinkoffPay.
- Saving card details during payments.
- Improved appearance and behavior of the payment dialog.
- Removed unnecessary dependencies and uses-permissions
- PurchaseResult model supplemented with a new invoiceId field.

SDK Version 1.0.0

- Transition from singleton to instance creation: RuStoreBillingClient.init() replaced by RuStoreBillingClientFactory.create().
- Singleton operation methods (init, products, purchases, getSingleton) are marked as deprecated and will be removed in future versions.

Unity

General Information	185
Payment functions availability	188
How to get the user's list of products	189
How to get the user's list of purchases	192
How to handle purchases	197
Purchase confirmation	200
Purchase cancellation	202
Confirmation and cancellation scenario	204
Error handling	205
Unity plug-in revision history	207

General Information

Download the [Unity plug-in](#) to integrate payments in your app.

Comply with the terms below to ensure proper payment integration:

1. The RuStore app must be installed on the user's device.
2. The user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

How to integrate it in your project

To get started, you need to download the [RuStore Billing SDK](#) and import it into your project (Assets → Import Package → Custom Package). Dependencies are included automatically using the External Dependency Manager (included in the SDK).

To correctly handle SDK dependencies, you must set the following settings:

- Edit -> Project Settings -> Player Settings -> Publishing Settings, then enable Custom Main Gradle Template and Custom Gradle Properties Template
- Assets -> External Dependencies Manager -> Android Resolver -> Settings, then enable Use Jetifier, Patch mainTemplate.gradle, Patch gradleTemplate.properties

After setting up, be sure complete Assets -> External Dependencies Manager -> Android Resolver -> Force Resolve.

Minimum API level must be set to at least 24. Application minification (ProGuard/R8) is not currently supported; it must be disabled in the project settings (File → Build Settings → Player Settings → Publishing Settings → Minify).

Handling deeplinks in your app

To redirect a user back to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity
    android:name="ru.rustore.unitysdk.RuStoreUnityActivity"
    android:theme="@style/UnityThemeSelector" android:exported="true">

    <intent-filter>
```

```

        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE"
/>
        <data android:scheme="yourappscheme" />
    </intent-filter>

</activity>

```

where "yourappscheme" — your deeplink scheme, it can be changed to another one.

Next, extend the UnityPlayerActivity class and add incoming intent processing to onNewIntent:

```

package ru.rustore.unitysdk;

import android.os.Bundle;
import android.content.Intent;
import ru.rustore.unitysdk.billingclient.RuStoreUnityBillingClient;
import com.unity3d.player.UnityPlayerActivity;

public class RuStoreUnityActivity extends UnityPlayerActivity {

    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (savedInstanceState == null) {
            RuStoreUnityBillingClient.onNewIntent(getIntent());
        }
    }

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

```

```
}  
}
```

The Java file with UnityPlayerActivity extension code must be placed in the Assets folder of the project. If you already have your own UnityPlayerActivity extension, you need to transfer the code of the onCreate and onNewIntent functions to it.

How to initialize a library

You must initialize the library before calling its methods. The library initialization parameters are configured in the Unity editor. In the editor menu select Window → RuStoreSDK → Settings → Billing Client.

```
RuStoreBillingClient.Instance.Init();
```

Once you're required other settings, you can pass them directly from the code:

```
var config = new RuStoreBillingClientConfig() {  
    consoleApplicationId = "11111",  
    deeplinkPrefix = "yourappscheme",  
    allowNativeErrorHandling = true,  
    enableLogs = true  
};  
  
RuStoreBillingClient.Instance.Init(config);
```

- consoleApplicationId — an application code from the [RuStore Console](https://console.rustore.ru/apps/111111). (Example: <https://console.rustore.ru/apps/111111>);
- deeplinkPrefix — an url used for deeplink. Make sure your use a unique name, for exampl: yourappscheme);
- allowNativeErrorHandling — allows error handling in a native SDK (see more in the [Error handling](#) section);
- enableLogs — enable logs.

Make sure that the deeplink scheme passed to deeplinkScheme matches the one specified in AndroidManifest.xml under "Handling deeplinks in your app".

If you need to check that the library is initialized, use RuStoreBillingClient.isInialized, which returns true if the library is initialized and false if the init function has not been called yet.

```
var isInialized = RuStoreBillingClient.Instance.IsInialized;
```


Payment functions availability

To check whether your app supports payment functions, call the `checkPurchasesAvailability` method. This method checks the following conditions:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. The app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all conditions are met, `onSuccess` displays `FeatureAvailabilityResult.isAvailable == true`. Otherwise, it returns `FeatureAvailabilityResult.isAvailable == false`, where `FeatureAvailabilityResult.cause` indicates an unfulfilled condition. All possible `RuStoreException` errors are described in [Error Handling](#). Other errors (e.g. "No internet connection") are processed by `onFailure`.

```
RuStoreBillingClient.Instance.CheckPurchasesAvailability(  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: (response) => {  
        if (response.isAvailable) {  
            // Process purchases available  
        } else {  
            // Process purchases unavailable  
        }  
    })  
});
```

How to get the user's list of products

Use the `getProducts` method to get the user's list of products

```
RuStoreBillingClient.Instance.GetProducts(productIds,
    onFailure: (error) => {
        // Process error
    },
    onSuccess: (response) => {
        // Process response
    });
```

- `string[] productIds` — list of product IDs.

The method returns:

- `List<Product> response` — list of products.

Product Structure:

```
public class Product {

    public enum ProductStatus {

        ACTIVE,
        INACTIVE
    }

    public enum ProductType {

        NON_CONSUMABLE,
        CONSUMABLE,
        SUBSCRIPTION
    }

    public string productId;
    public ProductType productType;
    public ProductStatus productStatus;
    public string priceLabel;
    public int price;
```

```

public string currency;
public string language;
public string title;
public string description;
public string imageUrl;
public string promoImageUrl;
public ProductSubscription subscription;
}

```

- productId — product identifier;
- productType — product type;
- productStatus — product status;
- priceLabel — formatted product price, including the currency symbol in [language];
- price — price in minor units (in kopecks);
- currency — ISO 4217 currency code;
- language — language specified with the BCP 47 encoding;
- title — product name in [language];
- description — product description in [language];
- imageUrl — link to the image;
- promoImageUrl — link to the promo image;
- subscription — subscription description, returned only for products with the subscription type.

Subscription structure:

```

public class ProductSubscription {

    public SubscriptionPeriod subscriptionPeriod;
    public SubscriptionPeriod freeTrialPeriod;
    public SubscriptionPeriod gracePeriod;
    public string introductoryPrice;
    public string introductoryPriceAmount;
    public SubscriptionPeriod introductoryPricePeriod;
}

```

- subscriptionPeriod — subscription period;
- freeTrialPeriod — trial subscription period;
- gracePeriod — grade subscription period;

- introductoryPrice — formatted introductory subscription price, including the currency symbol in product language;
- introductoryPriceAmount — introductory price in minor currency units (in kopecks);
- introductoryPricePeriod — introductory price settlement period.

Subscription period structure:

```
public class SubscriptionPeriod {

    public int years;
    public int months;
    public int days;
}
```

- years — number of years;
- months — number of months;
- days — number of days.

How to get the user's list of purchases

Use the getPurchases method to get the user's list of purchases:

```
RuStoreBillingClient.Instance.GetPurchases(
    onFailure: (error) => {
        // Process error
    },
    onSuccess: (response) => {
        // Process response
    });
```

The method returns:

- List<Purchase> response — list of products.

Purchase Structure:

```
public class Purchase {

    public enum PurchaseState
    {
```

```

    CREATED,
    INVOICE_CREATED,
    CONFIRMED,
    PAID,
    CANCELLED,
    CONSUMED,
    CLOSED
}

public string purchaseId;
public string productId;
public string description;
public string language;
public DateTime purchaseTime;
public string orderId;
public string amountLabel;
public int amount;
public string currency;
public int quantity;
public PurchaseState purchaseState;
public string developerPayload;
public string subscriptionToken;
}

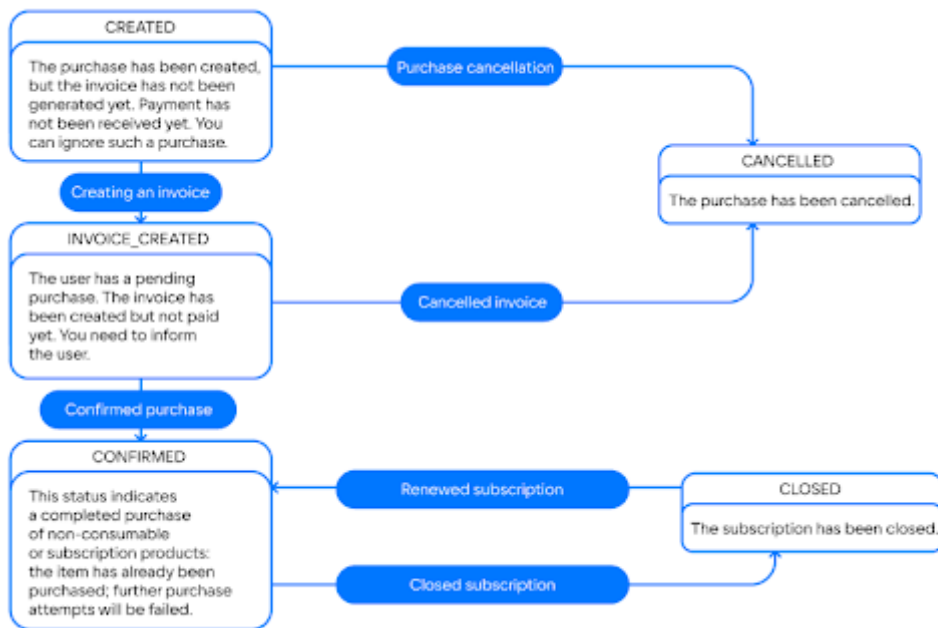
```

- purchaseId — purchase ID;
- productId — product identifier;
- productType — product type;
- invoiceId — invoice ID;
- description — purchase description;
- language — language specified with the BCP 47 encoding;
- purchaseTime — purchase time (in RFC 3339 format);
- orderId — unique payment identifier generated by the application (uuid);
- amountLabel — formatted purchase price, including the currency symbol in [language];
- amount — price in minor units of currency;
- currency — ISO 4217 currency code;
- quantity — number of products;
- purchaseState — purchase status:
 - possible values of the purchase condition:
 - CREATED — created;
 - INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;

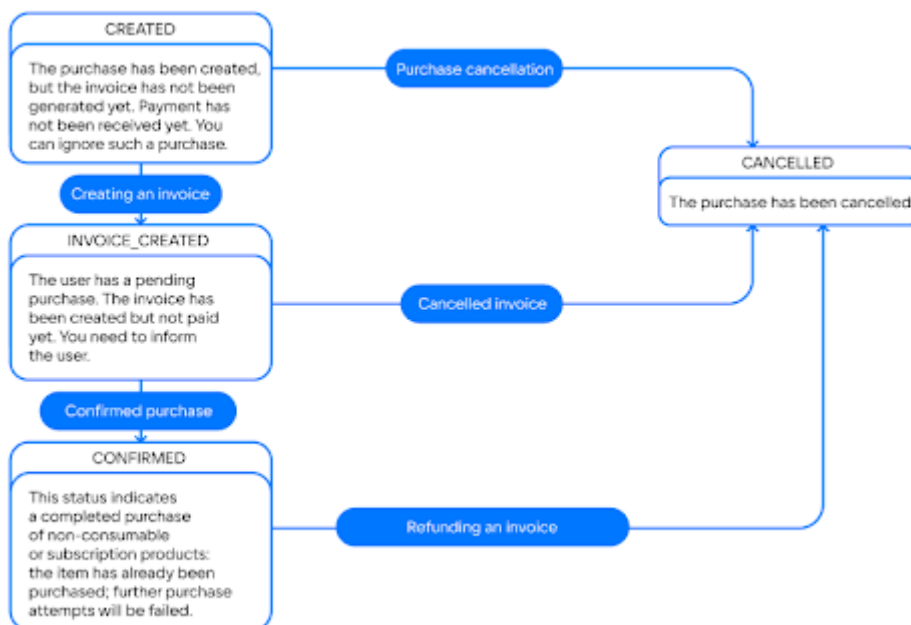
- CANCELLED — purchase canceled;
- CONSUMED — purchase consumption is confirmed;
- CLOSED — subscription is canceled.
- developerPayload — line specified by the developer that contains additional information about the order;

The purchaseState model:

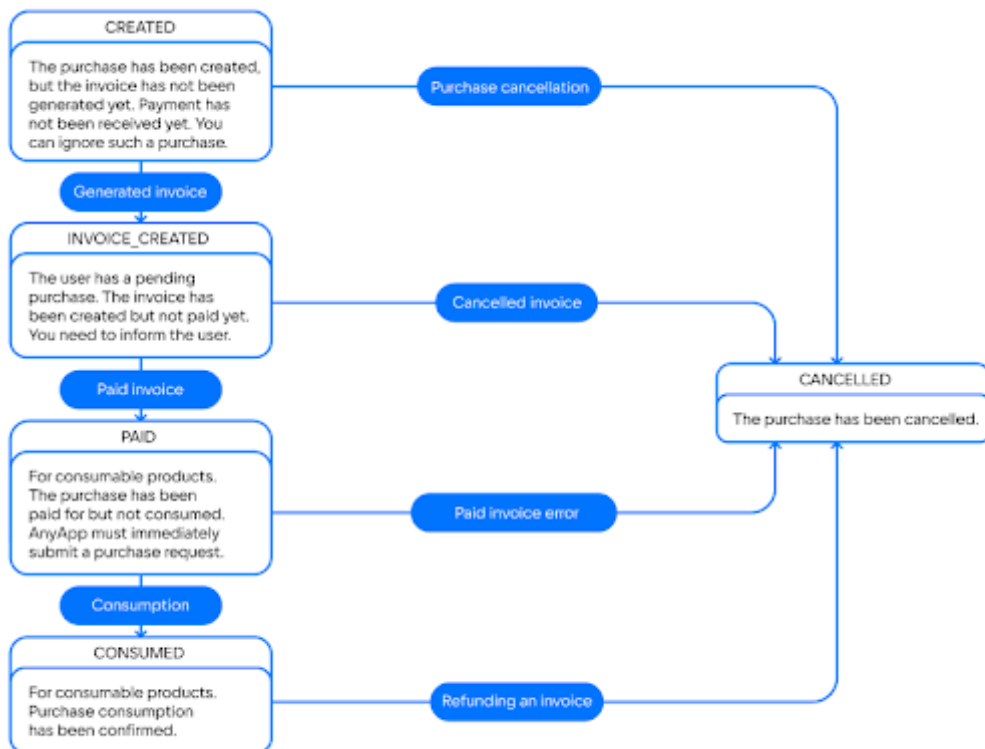
A status-based subscription purchase model (SUBSCRIPTIONS):



A status-based non-consumables subscription (NON-CONSUMABLES):



A status-based consumables subscription (CONSUMABLES):



How to handle purchases

Use the `purchaseProduct` method to call a product purchase:

```
RuStoreBillingClient.Instance.PurchaseProduct(  
    productId: "productId",  
    quantity: 1,  
    developerPayload: "",  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: (response) => {  
        switch (response) {  
            case PaymentSuccess paymentSuccess:  
                // Process PaymentSuccess  
                break;  
            case PaymentCancelled paymentCancelled:  
                // Process PaymentCancelled  
                break;  
            case PaymentFailure paymentFailure:  
                // Process PaymentFailure  
                break;  
            case InvalidPaymentState invalidPaymentState:  
                // Process InvalidPaymentState  
                break;  
        }  
    });
```

- `string productId` — product identifier;
- `string orderId` — order identifier, generated by the AnyApp;
- `int quantity` — products quantity;
- `string developerPayload` — additional information received from the AnyApp developer.

Payment result structure:

```
public class PaymentResult {  
}
```

```

public class PaymentSuccess : PaymentResult {

    public string orderId;
    public string purchaseId;
    public string productId;
    public string invoiceId;
    public string subscriptionToken;
}

public class PaymentCancelled : PaymentResult {

    public string purchaseId;
}

public class PaymentFailure : PaymentResult {

    public string purchaseId;
    public string invoiceId;
    public string orderId;
    public int quantity;
    public string productId;
    public int errorCode;
}

public class InvalidPaymentState : PaymentResult {
}

```

- PaymentSuccess — product successfully purchased;
- PaymentCancelled — purchase canceled;
- PaymentFailure — product purchase error;
- InvalidPaymentState — payment SDK error. May occur in case of an incorrect reverse deeplink.

Purchase confirmation

The RuStore application consists of the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);

- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation once they get the PurchaseState.PAID state.

You can use the ConfirmPurchase method to consume the purchase:

```
RuStoreBillingClient.Instance.ConfirmPurchase(  
    purchaseId: "purchaseId",  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: (response) => {  
        // Process success  
    }  
);
```

- purchaseId — purchase ID;

Purchase cancellation

You can use the `DeletePurchase` method to cancel the purchase:

```
RuStoreBillingClient.Instance.DeletePurchase(  
    purchaseId: "purchaseId",  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: () => {  
        // Process success  
    }  
);
```

- `purchaseId` — purchase ID.

Note. Use this method if your app logic is related to purchase cancellation. The purchase is canceled automatically after a 20-min timeout, or upon a second purchase from the same customer.

Confirmation and cancellation scenario

Uncompleted payments must be processed by the AnyApp developer.

The purchase cancellation method should be used if:

1. The method of getting the list of products returned the purchase status as follows:
 - PurchaseState.CREATED;
 - PurchaseState.INVOICE_CREATED;
2. If purchaseProduct returned PaymentResult.InvalidPurchase.
3. If purchaseProduct returned PaymentResult.PurchaseResult that contains the following PaymentFinishCode:
 - CLOSED_BY_USER — canceled by the user;
 - UNHANDLED_FORM_ERROR — unknown error;
 - PAYMENT_TIMEOUT — timeout payment error;
 - DECLINED_BY_SERVER — rejected by server;
 - RESULT_UNKNOWN — unknown payment status.

The confirmPurchase method should be used if:

1. The getPurchases method returned the purchase status as follows:
 - PurchaseState.PAID.
2. The purchaseProduct method returned PaymentResult.PurchaseResult that contains the following PaymentFinishCode:
 - SUCCESSFUL_PAYMENT — successful payment.

Error handling

All the errors that may occur are processed by `onFailure`.

Error structure:

```
public class RuStoreError {  
  
    public string name;  
    public string description;  
}
```

- `name` — error name;
- `description` — error description.

Possible errors:

- `RuStoreNotInstalledException` — RuStore is not installed on the user's device;
- `RuStoreOutdatedException` — RuStore, installed on the user's device, does not support payment processing functions;
- `RuStoreUserUnauthorizedException` — user is not authorized on the RuStore;
- `RuStoreApplicationBannedException` — your application is blocked on the RuStore;
- `RuStoreUserBannedException` — user is blocked on the RuStore;
- `RuStoreException` — basic RuStore error, from which all other errors are inherited.

When calling the `PurchaseProduct` method, errors are handled automatically.

If `allowNativeErrorHandling == true` occurs during SDK initialization, it not only calls the `onFailure` method but also returns the error to the `resolveForBilling` method in native SDK in order to demonstrate the error to the user:

```
public fun RuStoreException.resolveForBilling(context: Context)
```

Once initialized, it can be changed by setting the `AllowNativeErrorHandling` property:

```
RuStoreBillingClient.Instance.AllowNativeErrorHandling = false;
```

Unity plug-in revision history

Plug-in version 2.2.0

- Internal update.

Plug-in version 2.1.1

- Changed return value in GetProducts() for getting a list of products.
- Changed return value in GetPurchases() for getting a shopping list.
- Changed return value in GetPurchaseInfo() for getting purchase information.
- Changed value format in ConfirmPurchase().
- Changed return value in DeletePurchase() for deleting a purchase.
- Changed return value format in PurchaseProduct().

Plug-in version 1.1.1

- Added SDK configuration functionality in the Unity editor.
- orderId in PurchaseProduct() made optional.
- Added invoiceId field, invoice identifier to the Purchase and PurchaseResult classes.
- Fixed SDK initialization from Activity.

Plug-in version 0.1.9

- RuStoreBillingClientConfig.enableLogs flag added to enable logging.
- CallbackHandler initialization error fixed.

Plug-in version 0.1.8

- subscriptionToken field added to Purchase for server-side purchase validation.

Plug-in version 0.1.7

- Internal plug-in update.

Flutter

General Information	209
Payment functions availability	212
How to get the list of products	213
How to get the user's list of purchases	217
How to handle purchases	220
Purchase confirmation	223

General Information

Example of implementation

Please have a thorough look at the [application example](#) to learn how to integrate payments correctly.

Payment integration guideline

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your app user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

How to add the package to your project

Run the command below to add the application package to your project:

```
flutter pub add flutter_rustore_billing
```

This command adds the following line to pubspec.yaml.

```
dependencies:  
  flutter_rustore_billing: ^3.0.0
```

Handling deeplinks in your app

To redirect a user back to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity  
  android:name=".sample.MainActivity">  
  
  <intent-filter>  
    <action android:name="android.intent.action.MAIN" />  
    <category android:name="android.intent.category.LAUNCHER" />  
  </intent-filter>  
  
  <intent-filter>  
    <action android:name="android.intent.action.VIEW" />  
    <category android:name="android.intent.category.DEFAULT" />  
    <category android:name="android.intent.category.BROWSABLE" />  
  </intent-filter>  
</activity>
```

```
        <data android:scheme="yourappscheme" />
    </intent-filter>
```

```
</activity>
```

where "yourappscheme" — your deeplink scheme, it can be changed to another one.

This scheme must match the deeplinkScheme parameter passed to initialize().

How to initialize a library

Initialize the library before calling its methods. You can initialize the library using the `RustoreBillingClient.initialize()` method as follows:

```
RustoreBillingClient.initialize(
    "123456",
    "yourappscheme://iamback",
).then((value) {
    print("initialize success: $value");
}, onError: (err) {
    print("initialize err: $err");
});
```

- 123456 — application code from the RuStore Console (example: <https://console.rustore.ru/apps/123456>).
- yourappscheme://iamback — deeplink scheme which is used to redirect a user back to your app after he completes payment via a this-party app (for example, SberPay or FPS). Therefore SDK generates its host to the deeplink scheme.

The deeplink scheme passed to deeplinkScheme must correspond to the scheme specified in the Handling deeplinks section of the `AndroidManifest.xml` file.

Payment functions availability

Please ensure compliance with the conditions below to check whether your app supports payment functions.

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all the above conditions are met, the `RustoreBillingClient.available()` method returns true.

```
RustoreBillingClient.available().then((value) {  
    print("available success $value");  
}, onError: (err) {  
    print("available err: $err");  
});
```

How to get the list of products

Use the `RustoreBillingClient.products(ids)` method to get a list of products:

```
RustoreBillingClient.products(ids).then((response) {
    for (final product in response.products) {
        print(product?.productId);
    }
}, onError: (err) {
    print("products err: $err");
});
```

- `ids: List<String?>` — list of product identifiers.

The method returns `ProductsResponse`:

```
class ProductsResponse {
    int code;
    String? errorMessage;
    String? errorDescription;
    String? traceId;
    List<Product?> products;
    List<DigitalShopGeneralError?> errors;
}
```

- `code` — response code;
- `errorMessage` — error method;
- `errorDescription` — error description;
- `traceId` — error ID;
- `errors` — list of errors;
- `products` — list of products.

Error structure `DigitalShopGeneralError`:

```
class DigitalShopGeneralError {
    String? name;
    int? code;
}
```

```
String? description
}
```

- name — error name;
- code — error code;
- description — error description.

Product Structure:

```
class Product {
    String productId;
    String? productType;
    String productStatus;
    String? priceLabel;
    int? price;
    String? currency;
    String? language;
    String? title;
    String? description;
    String? imageUrl;
    String? promoImageUrl;
    Subscription? subscription;
}
```

- productId — product ID;
- productType — product type;
- productStatus — product status;
- priceLabel — formatted product price, including the currency symbol in [language];
- price — price in minor units (in kopecks);
- currency — ISO 4217 currency code;
- language — language specified with the BCP 47 encoding;
- title — product name in [language];
- description — product description in [language];
- imageUrl — link to the image;
- promoImageUrl — link to the promo image;
- subscription — subscription description, returned only for products with the subscription type.

Subscription Structure:

```
class Subscription {
```

```
SubscriptionPeriod? subscriptionPeriod;  
SubscriptionPeriod? freeTrialPeriod;  
SubscriptionPeriod? gracePeriod;  
String? introductoryPrice;  
String? introductoryPriceAmount;  
SubscriptionPeriod? introductoryPricePeriod;  
}
```

- subscriptionPeriod — subscription period;
- freeTrialPeriod — trial subscription period;
- gracePeriod — subscription grace period;
- introductoryPrice — formatted introductory subscription price, including the currency symbol, in product:language;
- introductoryPriceAmount — introductory price in minor units of currency (in kopecks);
- introductoryPricePeriod — calculated period of the introductory price.

SubscriptionPeriod structure:

```
class SubscriptionPeriod {  
    int years;  
    int months;  
    int days;  
}
```

- years — number of years;
- months — number of months;
- days — number of days.

How to get the user's list of purchases

Use the `RustoreBillingClient.purchases()` method to get the user's list of purchases:

```
RustoreBillingClient.purchases().then((response) {
    for (final product in response.purchases) {
        print(product?.purchaseId);
    }
}, onError: (err) {
    print("purchases err: $err");
});
```

The method returns `PurchasesResponse`:

```
class PurchasesResponse {
    int code;
    String? errorMessage;
    String? errorDescription;
    String? traceId;
    List<Purchase?> purchases;
    List<DigitalShopGeneralError?> errors;
}
```

- `code` — response code;
- `errorMessage` — error method;
- `errorDescription` — error description;
- `traceId` — error ID;
- `errors` — list of errors;
- `purchases` — list of purchases.

`DigitalShopGeneralError` error structure:

```
class DigitalShopGeneralError {
    String? name;
    int? code;
    String? description;
}
```

- `name` — error name;
- `code` — error code;
- `description` — error description.

Purchase Structure:

```
class Purchase {
    String? purchaseId;
    String? productId;
    String? description;
    String? language;
    String? purchaseTime;
    String? orderId;
    String? amountLabel;
    int? amount;
    String? currency;
    int? quantity;
    String? purchaseState;
    String? developerPayload;
    String? invoiceId;
    String? subscriptionToken;
}
```

- purchaseId — purchase ID;
- productId — product ID;
- description — purchase description;
- language — language specified with the BCP 47 encoding;
- purchaseTime — purchase time (in RFC 3339 format);
- orderId — unique payment identifier generated by the application (uuid);
- amountLabel — formatted purchase price, including the currency symbol in [language];
- amount — price in minor units of currency;
- currency — ISO 4217 currency code;
- quantity — number of products;
- purchaseState — purchase status:
 - possible values of the purchase condition:
 - CREATED — created;
 - INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;
 - CANCELLED — purchase canceled;
 - CONSUMED — purchase consumption is confirmed;
 - CLOSED — subscription is canceled.
- developerPayload — line specified by the developer that contains additional information about the order.
- invoiceId — invoice ID;

- `subscriptionToken` — server validation token.

How to handle purchases

Use the `RustoreBillingClient.purchase(id)` method to call a product purchase:

```
RustoreBillingClient.purchase(id).then((response) {  
    print("purchase success: $response");  
}, onError: (err) {  
    print("purchase err: $err");  
});
```

- `id` — product ID.

Structure `PaymentResult`:

```
class PaymentResult {  
    SuccessInvoice? successInvoice;  
    InvalidInvoice? invalidInvoice;  
    SuccessPurchase? successPurchase;  
    InvalidPurchase? invalidPurchase;  
}
```

Structure `SuccessInvoice`:

```
class SuccessInvoice {  
    String invoiceId;  
    String finishCode;  
}
```

Structure `InvalidInvoice`:

```
class InvalidInvoice {  
    String? invoiceId;  
}
```

Structure `SuccessPurchase`:

```
class SuccessPurchase {
    String finishCode;
    String? orderId;
    String purchaseId;
    String productId;
}
```

InvalidPurchase Structure:

```
class InvalidPurchase {
    String? purchaseId;
    String? invoiceId;
    String? orderId;
    int? quantity;
    String? productId;
    int? errorCode;
}
```

- SuccessInvoice — payment successfully completed;
- InvalidInvoice — payment completed without an invoice. It may possibly be caused by an error in the invoice (such as an empty line);
- SuccessPurchase — product has been paid for successfully;
- InvalidPurchase — failed to complete payment.

finishCode may return one the following statuses:

- SUCCESSFUL_PAYMENT — successful payment;
- CLOSED_BY_USER — cancelled by the user;
- UNHANDLED_FORM_ERROR — unknown error;
- PAYMENT_TIMEOUT — timeout payment error;
- DECLINED_BY_SERVER — rejected by server;
- RESULT_UNKNOWN — unknown payment status.

Purchase confirmation

The RuStore app features the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);
- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation once they are in the PurchaseState.PAID state.

You can use the confirmPurchase method to confirm the purchase:

```
val purchasesUseCase: PurchasesUseCase = billingClient.purchases
purchasesUseCase.confirmPurchase(purchaseId = "purchaseId",
developerPayload = null)
    .addOnSuccessListener {
        // Process success
    }.addOnFailureListener { throwable: Throwable ->
        // Process error
    }
```

- purchaseId is a purchase ID;
- developerPayload refers to a line with additional info (optional).

Unreal Engine

General Information	1
Calling CheckPurchasesAvailability	5
Getting the List of Purchases	16
Getting Purchase Info	23
How to handle purchases	26
Consumption and cancellation scenario	45
Error handling	46

General Information

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

Embed in your project

To have the Billing SDK enabled, you need to download the Unreal Engine plugins “RuStoreCore” and “RuStoreBilling” from the official RuStore repository on gitflic. Then place them in the “Plugins” folder inside the project. Once there, the “RuStore Core” and “RuStore Billing” plugins will appear in the list of plugins (Edit → Plugins → Project → Mobile). Next, connect the “RuStoreCore” and “RuStoreBilling” modules in the `*YourProject*.Build.cs` file (`PublicDependencyModuleName`).

When building an Android app, the Minimum API level must be set to at least 24. Application minification (ProGuard/R8) is not currently supported. All necessary gradle settings and project dependencies are written in “RuStoreCore_UPL_Android.xml” and “RuStoreBilling_UPL_Android.xml”.

Handling deeplinks in your app

To redirect a user to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in `AndroidManifest.xml`:

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  <data android:scheme="yourscheme" />
</intent-filter>
```

where “yourscheme” is your deeplink scheme. It can be changed to another, and must match the deeplinkScheme value specified when initializing the billing client library.

You can change “yourscheme” to your own scheme in the “RuStoreCore_UPL_Android.xml” file.

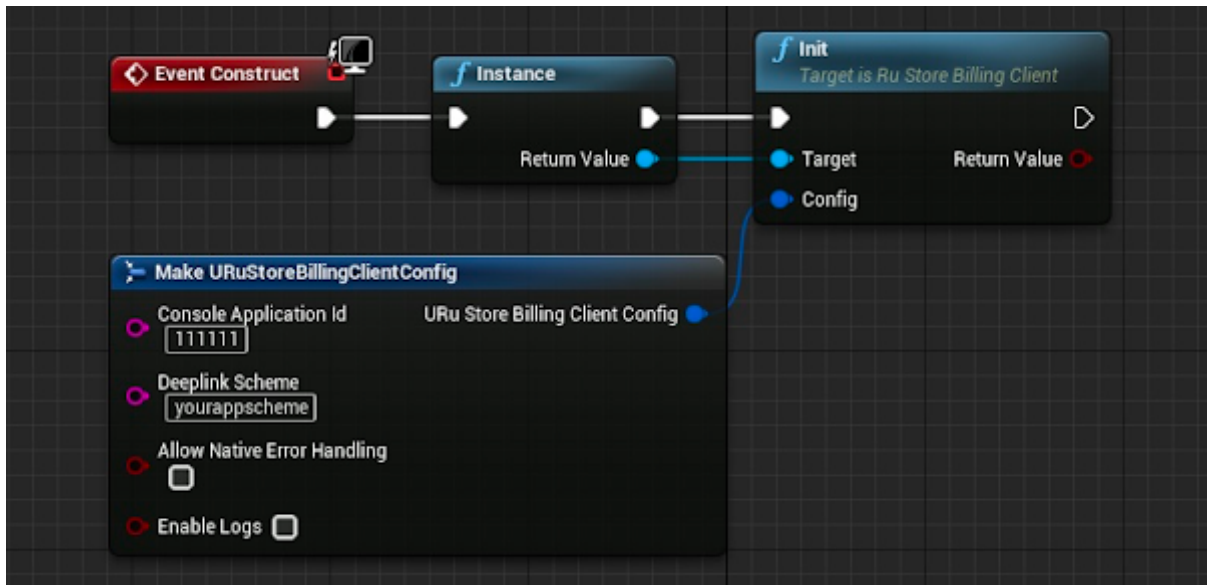
How to initialize the library

You must initialize the library to call its method. To do this, call the `FURuStoreBillingClientConfig` method:

Initialization

```
try {
  RustoreBillingClient.initialize({
    consoleApplicationId: 'appId',
    deeplinkScheme: 'scheme',
  });
  console.log(`initialize success: ${result}`);
} catch (err) {
  console.log(`initialize err: ${err}`);
}
```

All customer transactions are also accessible from Blueprints. Initialization example:



- consoleApplicationId — an application code from the [RuStore Console](https://console.rustore.ru/apps/111111). (Example: <https://console.rustore.ru/apps/111111>);
- deeplinkPrefix — an url used for deeplink. Make sure your use a unique name, for exampl: yourappscheme);
- allowNativeErrorHandling — allows error handling in a native SDK (see more in the [Error handling](#) section):
- enableLogs — enable logs.

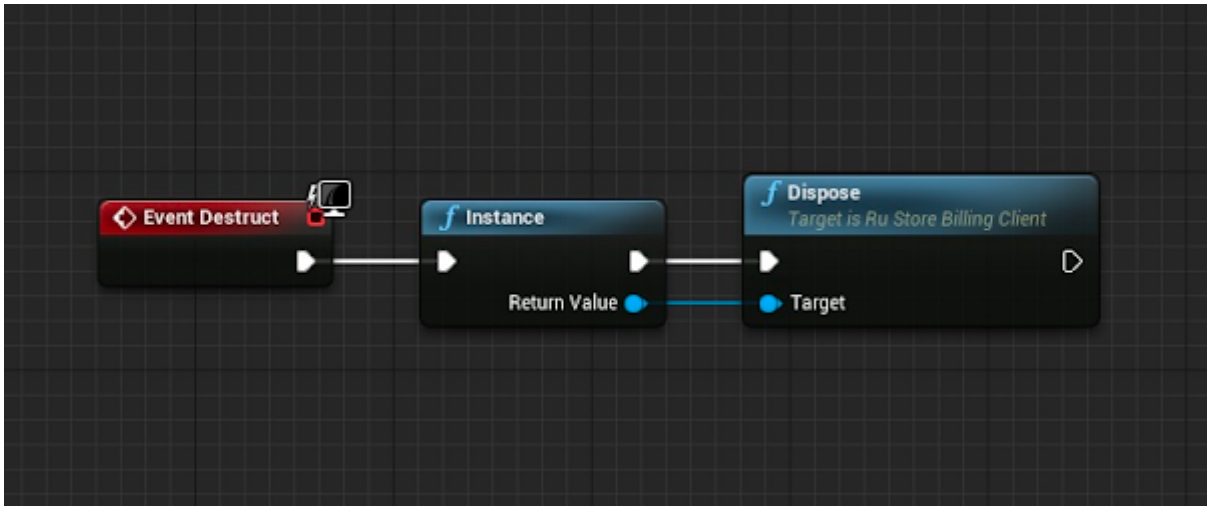
Note:

1. Make sure that the deeplink scheme passed to deeplinkScheme matches the one specified in AndroidManifest.xml under "Handling deeplinks in your app".
2. When calling Init(), it binds the object to the scene root, and if no further work is planned on the object, the Dispose() method must be called to free up memory.

When calling Dispose(), it unbinds the object from the root and safely completes all requests sent.

Deinitialization

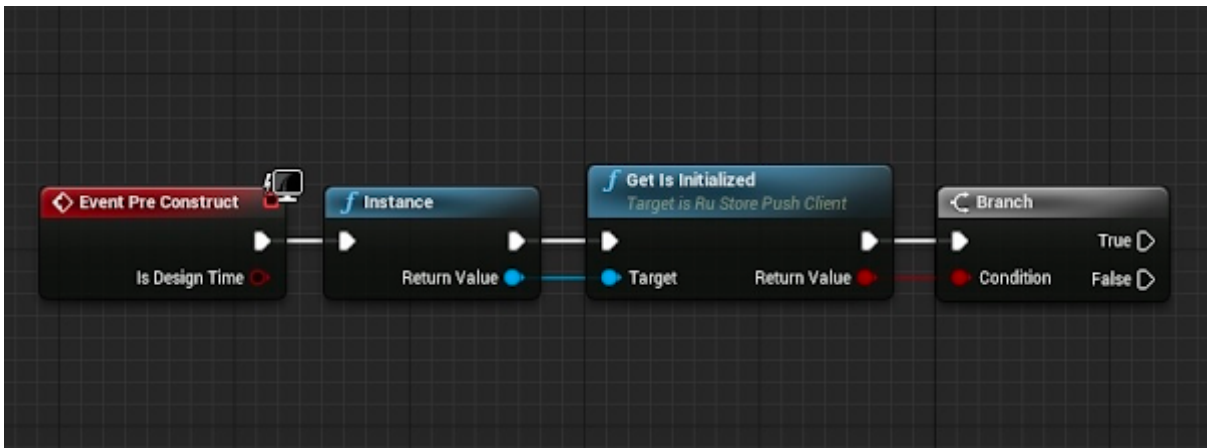
```
URuStoreBillingClient::Instance()->Dispose();
```

If you need to check whether the library has been initialized, use `URuStoreBillingClient::Instance()->getIsInitialized()`. It returns true if the library is initialized, and false if Init has not yet been called.

Initialization check

```
bool isInitialized =
URuStoreBillingClient::Instance()->IsIninialized();
```



Payment functions availability

Use `CheckPurchasesAvailability` to check whether your app supports payment functions. At that, the following conditions should be met:

1. The RuStore app must be installed on the user's device.

2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all conditions are met, `onSuccess` displays `FeatureAvailabilityResult.isAvailable == true`. Otherwise, it returns `FeatureAvailabilityResult.isAvailable == false`, where `FeatureAvailabilityResult.cause` indicates an unfulfilled condition. All possible `RuStoreException` errors are described in [Error Handling](#). Other errors (e.g. "No internet connection") are processed by `onFailure`.

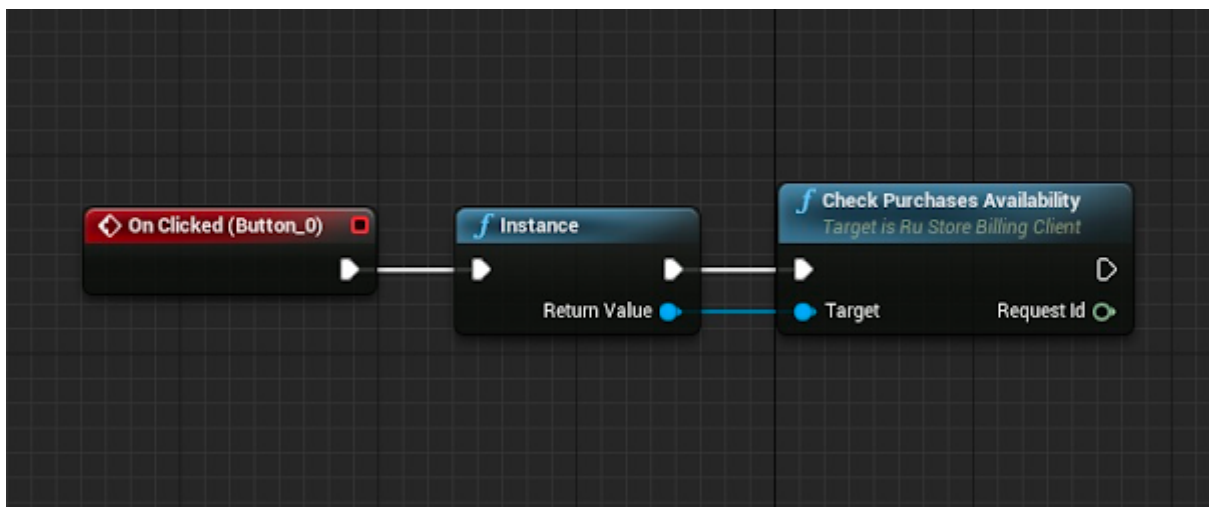
Each request receives a unique `requestId` within a single application launch. At that, each event returns `requestId` that generated the event.

Calling CheckPurchasesAvailability

```

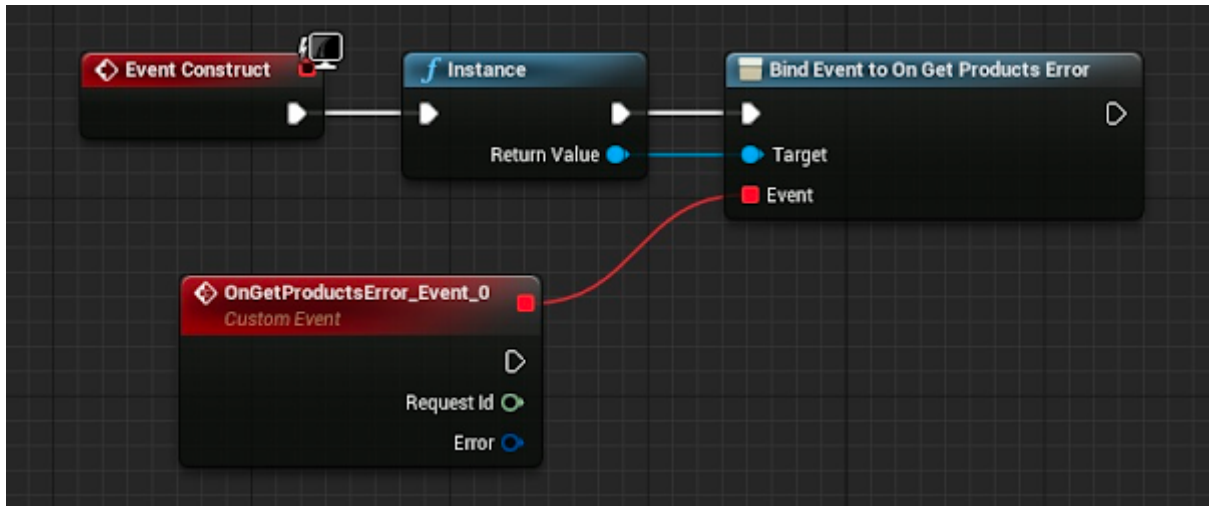
long requestId =
URuStoreBillingClient::Instance()->CheckPurchasesAvailability(
    [](long requestId,
    TSharedPtr<FURuStoreFeatureAvailabilityResult,
    ESPMode::ThreadSafe> response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);

```

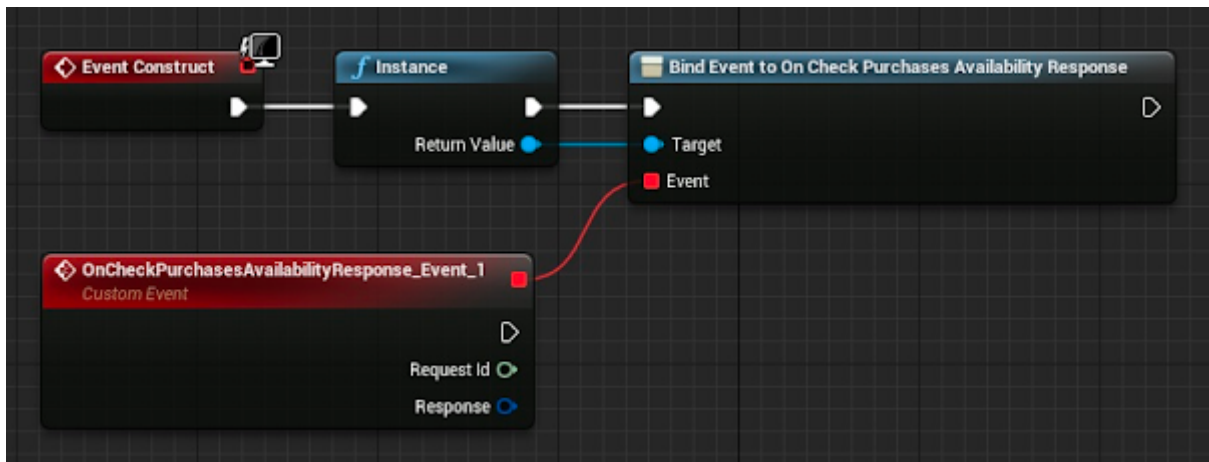


When working with Blueprints, you must subscribe to the `OnCheckPurchasesAvailabilityResponse` and `OnCheckPurchasesAvailabilityError` events to process the response.

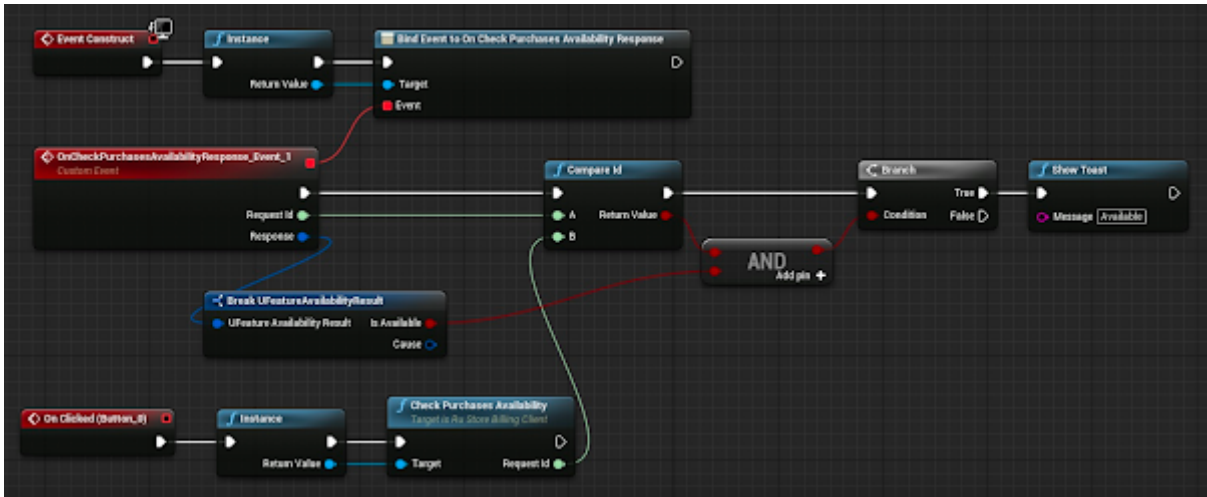
Example of subscribing to the `OnCheckPurchasesAvailabilityError` event is as follows:



Example of subscribing to the `OnCheckPurchasesAvailabilityResponse` event:



All Blueprint-events in the plugin are broadcast. A given request can be filtered using the `requestId` parameter. Each call to the `CheckPurchasesAvailability` method returns a unique `requestId` and each Blueprint-event returns `requestId` of the method that generated it.



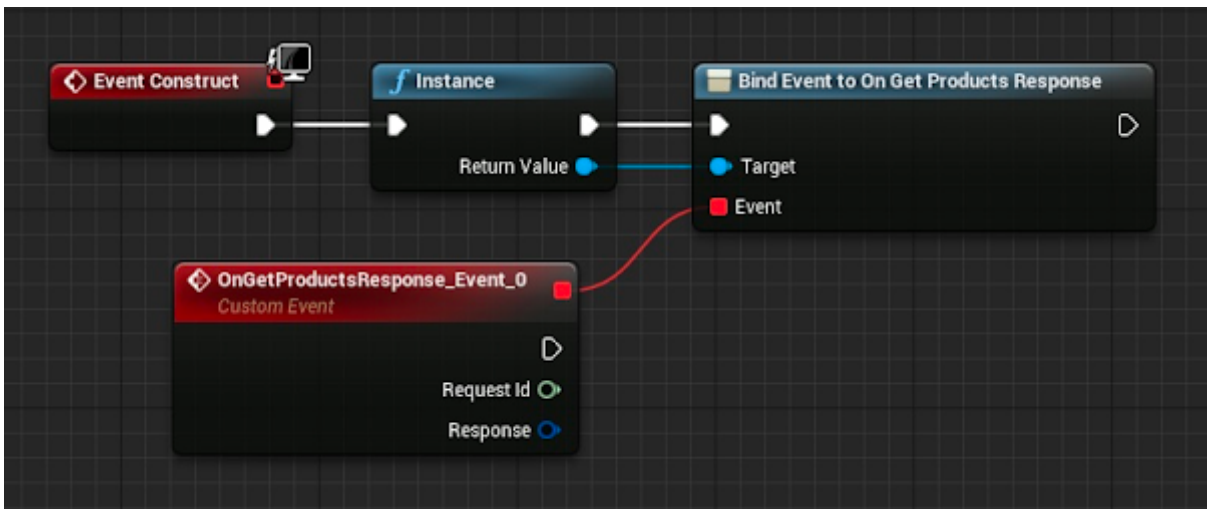
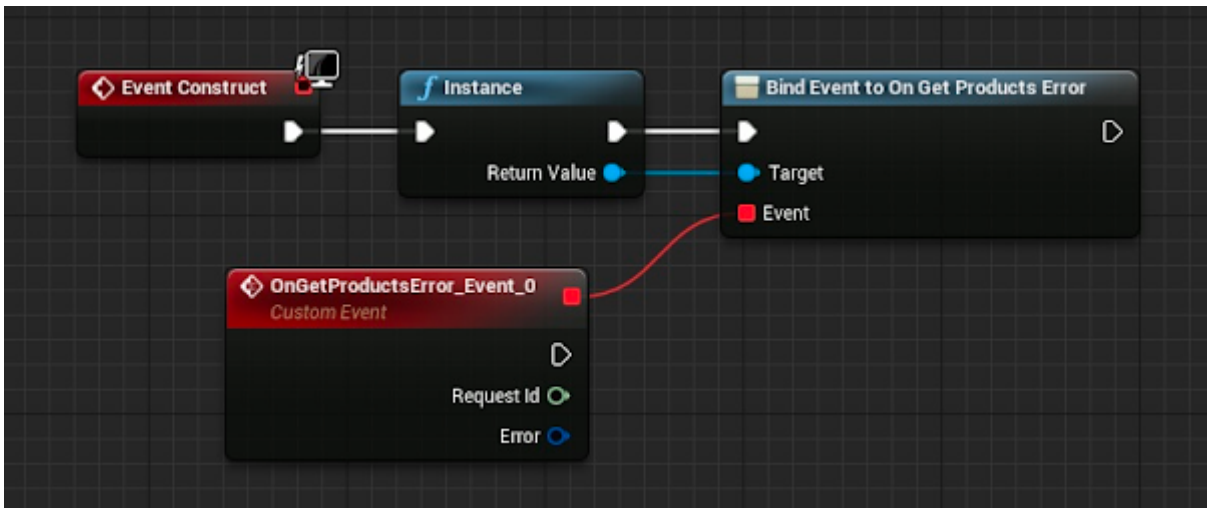
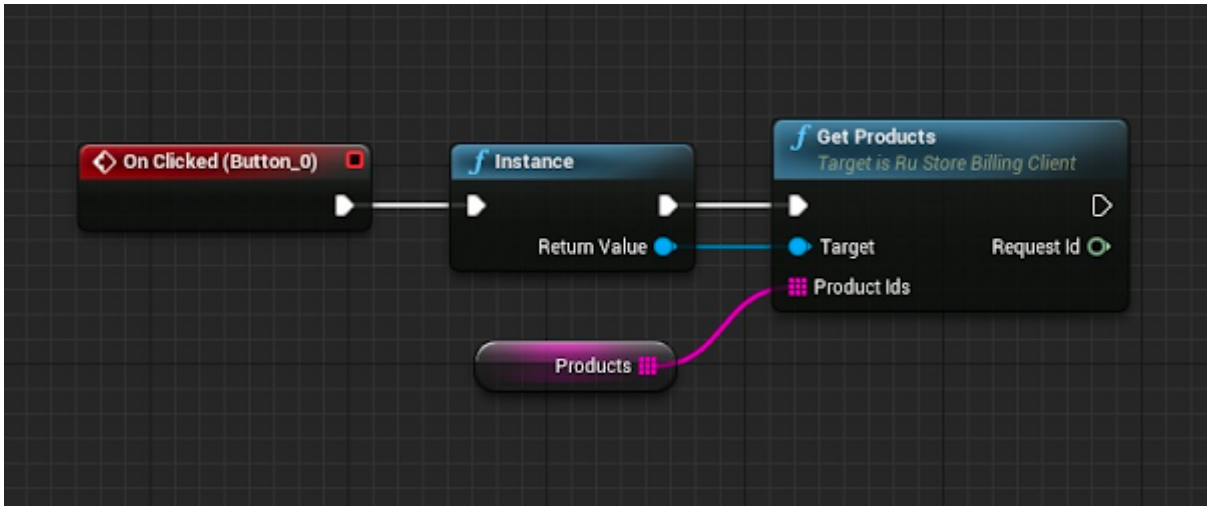
Getting the List of Products

Use the GetProducts method to get a list of products:

```
long requestId = URuStoreBillingClient::Instance()->GetProducts(
    productIds,
    [](long requestId, TSharedPtr<FURuStoreProductsResponse,
    ESPMode::ThreadSafe> response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```

- TArray<FString> productIds — list of products IDs.

Blueprint implementation



This method returns:
GetProducts response

```

USTRUCT(BlueprintType)
struct FURuStoreProductsResponse : public FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStoreProduct> products;
};

```

- products — list of products.

Basic response class

```

USTRUCT(BlueprintType)
struct FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    FURuStoreResponseWithCode()
    {
        int code = 0;
        errorMessage = "";
        errorDescription = "";
    }

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString errorMessage;

    UPROPERTY(BlueprintReadOnly)
    FString errorDescription;

    UPROPERTY(BlueprintReadOnly)
    TArray<FUDigitalShopGeneralError> errors;
};

```

- code — response code.

- errorMessage — error message.
- errorDescription — error description.
- errors — list of errors.

Error structure

```

USTRUCT(BlueprintType)
struct FURuStoreDigitalShopGeneralError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreDigitalShopGeneralError()
    {
        name = "";
        code = 0;
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    int code;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name — error name.
- code — error code.
- description — error description.

Product structure

```

USTRUCT(BlueprintType)
struct FURuStoreProduct
{
    GENERATED_USTRUCT_BODY()

    FURuStoreProduct()
    {
        productId = "";
        productType = EURuStoreProductType::CONSUMABLE;
        productStatus = EURuStoreProductStatus::INACTIVE;
        priceLabel = "";
        price = 0;
        currency = "";
        language = "";
        title = "";
        description = "";
        imageUrl = "";
        promoImageUrl = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString productId;

    UPROPERTY(BlueprintReadOnly)
    EURuStoreProductType productType;

    UPROPERTY(BlueprintReadOnly)
    EURuStoreProductStatus productStatus;

    UPROPERTY(BlueprintReadOnly)
    FString priceLabel;

    UPROPERTY(BlueprintReadOnly)
    int price;

```



```

UPROPERTY(BlueprintReadOnly)
FString currency;

UPROPERTY(BlueprintReadOnly)
FString language;

UPROPERTY(BlueprintReadOnly)
FString title;

UPROPERTY(BlueprintReadOnly)
FString description;

UPROPERTY(BlueprintReadOnly)
FString imageUrl;

UPROPERTY(BlueprintReadOnly)
FString promoImageUrl;

UPROPERTY(BlueprintReadOnly)
FURuStoreProductSubscription subscription;
};

```

- product_id — product ID;
- product_type — product type;
- product_status — product status;
- price_lable — formatted product price, including currency symbol in [language];
- price — price in minimum units (in kopecks);
- currency — currency code ISO 4217;
- language — language specified using BCP 47 encoding;
- title — product name in [language];
- description — product description in [language];
- image_url — image link
- promo_image_url — promotional image link;
- subscription — subscription description, returned only for subscription type products.

Subscription structure:

```
USTRUCT(BlueprintType)
struct FURuStoreProductSubscription
{
    GENERATED_USTRUCT_BODY()

    FURuStoreProductSubscription()
    {
        introductoryPrice = "";
        introductoryPriceAmount = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod subscriptionPeriod;

    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod freeTrialPeriod;

    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod gracePeriod;

    UPROPERTY(BlueprintReadOnly)
    FString introductoryPrice;

    UPROPERTY(BlueprintReadOnly)
    FString introductoryPriceAmount;

    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod introductoryPricePeriod;
};
```

- subscriptionPeriod — subscription period;
- freeTrialPeriod — subscription trial period;
- gracePeriod — subscription grace period;
- introductoryPrice — formatted introductory subscription price, including currency sign, in product:language;
- introductoryPriceAmount — introductory price in minimum currency units (in kopecks);
- introductoryPricePeriod — calculation period of the introductory price.

SubscriptionPeriod structure. Structure relates to subscription period terms

```
USTRUCT(BlueprintType)
struct FURuStoreSubscriptionPeriod
{
    GENERATED_USTRUCT_BODY()

    FURuStoreSubscriptionPeriod()
    {
        years = 1970;
        months = 1;
        days = 1;
    }

    UPROPERTY(BlueprintReadOnly)
    int years;

    UPROPERTY(BlueprintReadOnly)
    int months;

    UPROPERTY(BlueprintReadOnly)
    int days;
};
```

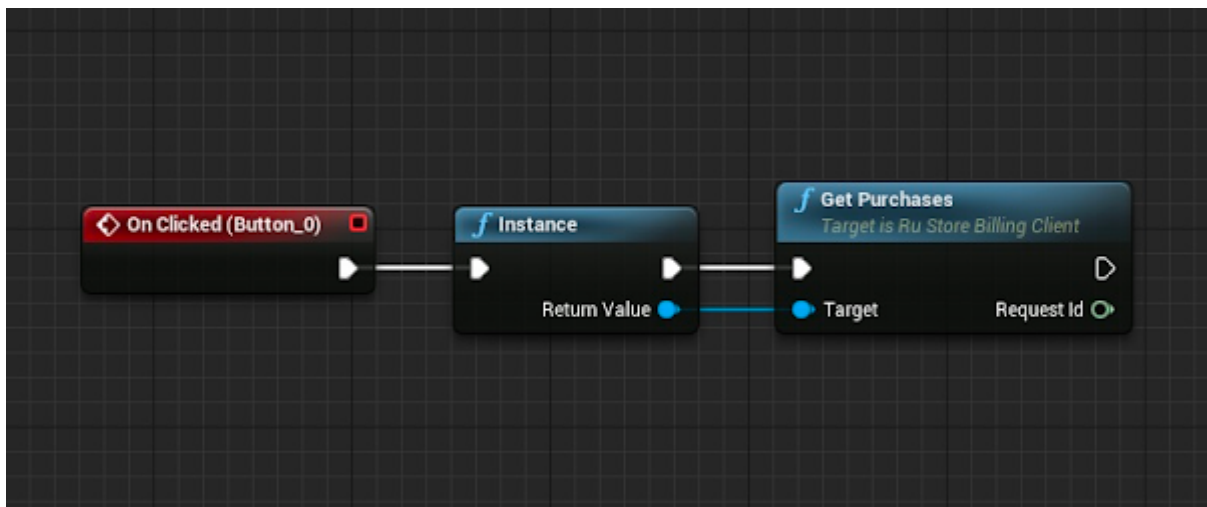
- years — number of years;
- months — number of months;
- days — number of days.

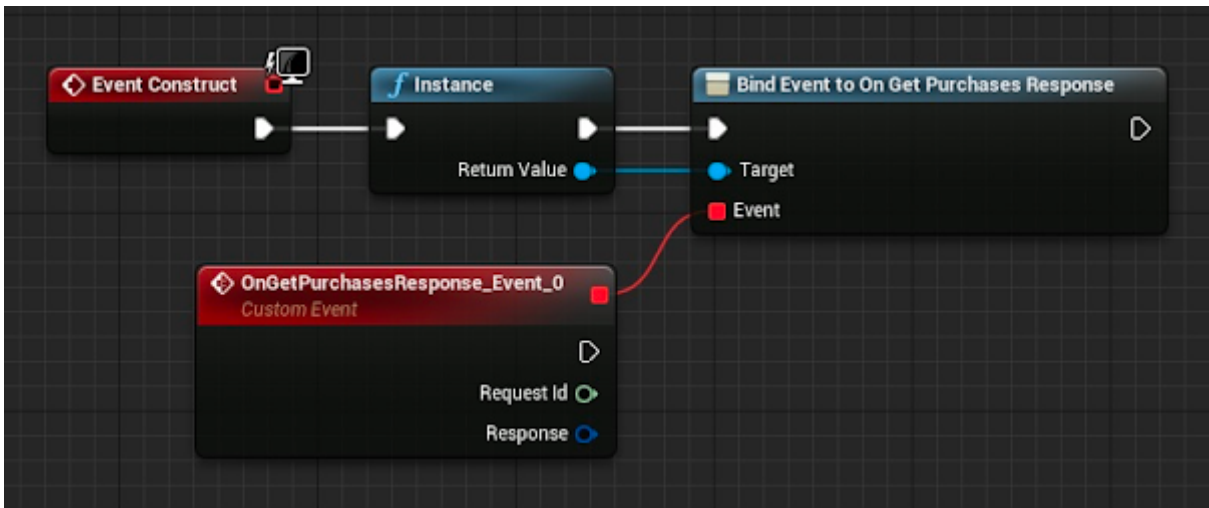
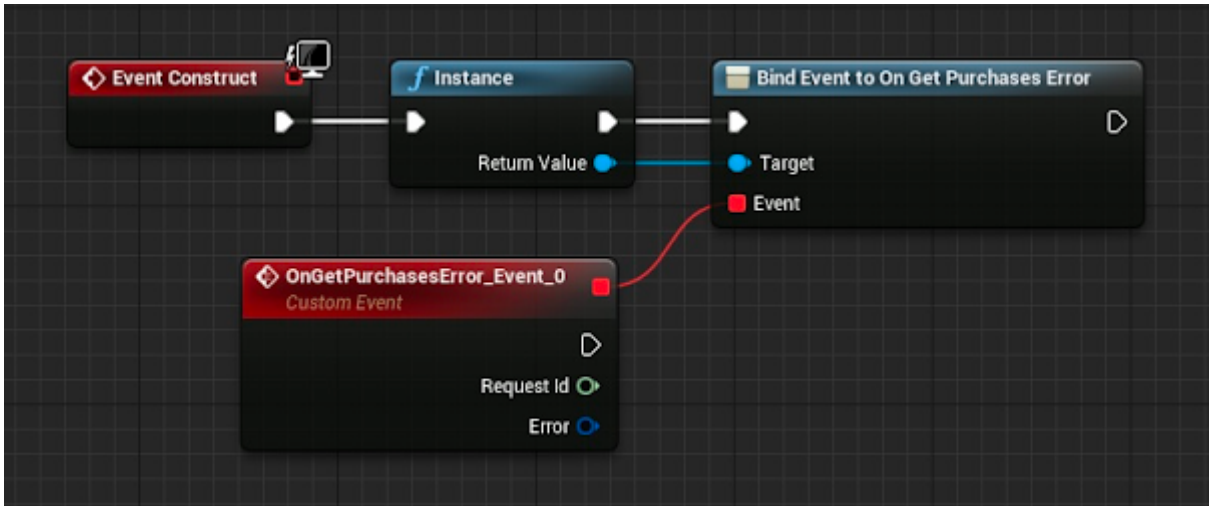
Getting the List of Purchases

Use the `GetPurchases` method to get the user's list of purchases

```
try {  
  const purchases = await RustoreBillingClient.getPurchases();  
  for (const purchase of purchases) {  
    console.log(purchase?.purchaseId);  
  }  
} catch (err) {  
  console.log(`purchase err: ${err}`);  
}
```

Blueprint implementation





This method returns:
GetProducts response

```

USTRUCT(BlueprintType)
struct FURuStorePurchasesResponse : public FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStorePurchase> purchases;
};

```

- products — list of products.

Basic response class

```

USTRUCT(BlueprintType)
struct FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    FURuStoreResponseWithCode()
    {
        code = 0;
        errorMessage = "";
        errorDescription = "";
    }

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString errorMessage;

    UPROPERTY(BlueprintReadOnly)
    FString errorDescription;

    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStoreDigitalShopGeneralError> errors;
};

```

- code — response code.
- errorMessage — error message.
- errorDescription — error description.
- errors — list of errors.

Error structure

```

USTRUCT(BlueprintType)
struct FURuStoreDigitalShopGeneralError
{
    GENERATED_USTRUCT_BODY()

    FURuStoreDigitalShopGeneralError()
    {
        name = "";
        code = 0;
        description = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString name;

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name — error name.
- code — error code.
- description — error description.

Product structure (with purchase info)

```

USTRUCT(BlueprintType)
struct FURuStorePurchase
{
    GENERATED_USTRUCT_BODY()

    FURuStorePurchase()
    {
        purchaseId = "";
        productId = "";
        invoiceId = "";
        description = "";
        language = "";
        purchaseTime = FDateTime(0);
        orderId = "";
        amountLabel = "";
        amount = 0;
        currency = "";
        quantity = 0;
        purchaseState = EURuStorePurchaseState::CANCELLED;
        developerPayload = "";
        subscriptionToken = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;

    UPROPERTY(BlueprintReadOnly)
    FString productId;

    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;

    UPROPERTY(BlueprintReadOnly)
    FString description;

```



```
UPROPERTY(BlueprintReadOnly)
FString language;

UPROPERTY(BlueprintReadOnly)
FDateTime purchaseTime;

UPROPERTY(BlueprintReadOnly)
FString purchaseTimeLabel;

UPROPERTY(BlueprintReadOnly)
FString orderId;

UPROPERTY(BlueprintReadOnly)
FString amountLabel;

UPROPERTY(BlueprintReadOnly)
int amount;

UPROPERTY(BlueprintReadOnly)
FString currency;

UPROPERTY(BlueprintReadOnly)
int quantity;

UPROPERTY(BlueprintReadOnly)
EURuStorePurchaseState purchaseState;

UPROPERTY(BlueprintReadOnly)
FString developerPayload;

UPROPERTY(BlueprintReadOnly)
FString subscriptionToken;
};
```

- purchaseId — purchase ID;
- productId — product identifier;
- description — product description;
- invoiceId — invoice ID;
- language — language specified with the BCP 47 encoding;
- purchaseTime — purchase time (in RFC 3339 format);
- purchaseTimeLabel — purchase time (DD.MM.YYYY HH:MM:SS)
- orderId — unique payment identifier generated by the application (uuid);
- amountLabel — formatted purchase price, including the currency symbol in [language];
- amount — price in minor units of currency;
- currency — ISO 4217 currency code;
- quantity — number of products;
- purchaseState — purchase status:
 - possible values of the purchase condition:
 - CREATED — created;
 - INVOICE_CREATED — created, waiting for payment;
 - CONFIRMED — confirmed;
 - PAID — paid for;
 - CANCELLED — purchase canceled;
 - CONSUMED — purchase consumption is confirmed;
 - CLOSED — subscription is canceled.
- developerPayload — line specified by the developer that contains additional information about the order;
- subscriptionToken — token for server purchase validation.

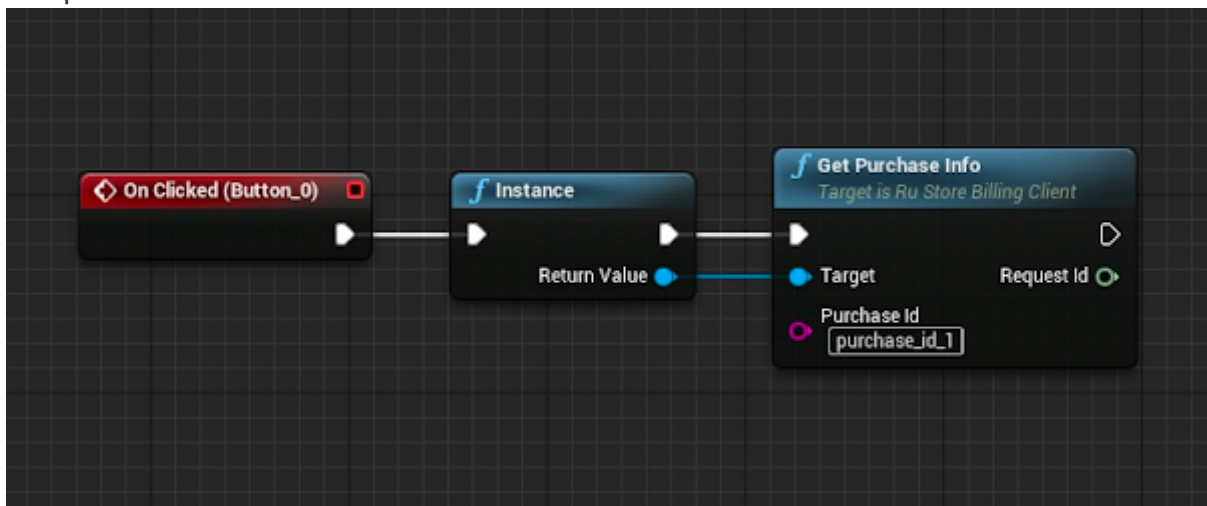
Getting Purchase Info

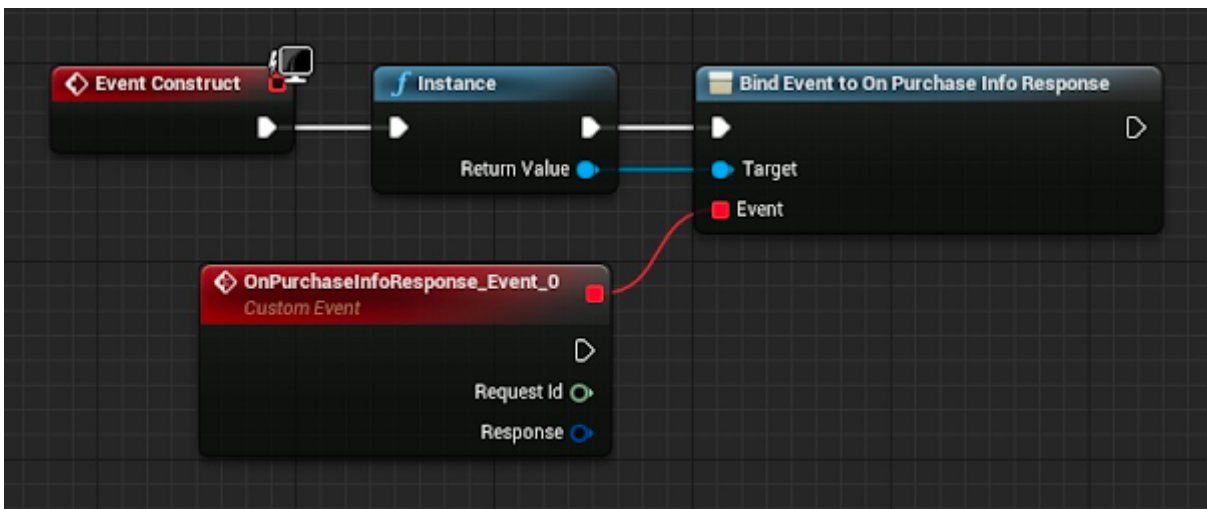
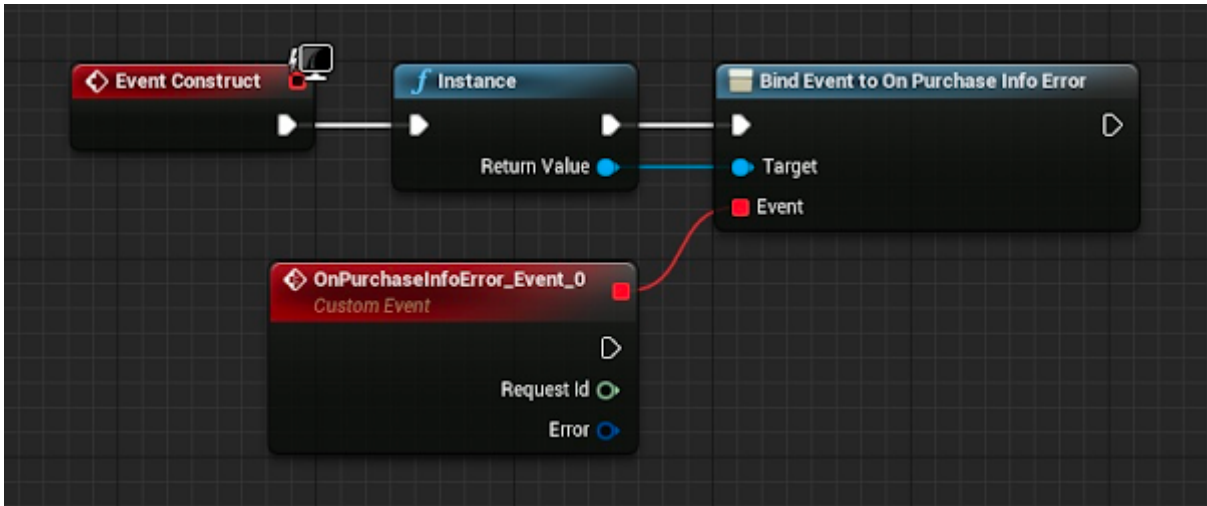
To get a specific purchase info, you must use the `GetPurchaseInfo` method:

```
long requestId =  
URuStoreBillingClient::Instance()->GetPurchaseInfo(  
    purchaseId,  
    [](long requestId, TSharedPtr<FURuStorePurchaseInfoResponse,  
    ESPMode::ThreadSafe> response) {  
        // Process response  
    },  
    [](long requestId, TSharedPtr<FURuStoreError,  
    ESPMode::ThreadSafe> error) {  
        // Process error  
    }  
);
```

- `purchaseId` — purchase ID.

Blueprint:





This method returns:

DeletePurchase response

```

USTRUCT(BlueprintType)
struct FURuStorePurchaseInfoResponse : public FUResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    UPROPERTY(BlueprintReadOnly)
    FURuStoreRequestMeta meta;

    UPROPERTY(BlueprintReadOnly)
    FURuStorePurchase purchase;
};

```

- meta — additional meta information about the purchase.
- purchase — purchase info.

Meta information structure:

DeletePurchase response

```

USTRUCT(BlueprintType)
struct FURuStoreRequestMeta
{
    GENERATED_USTRUCT_BODY()

    FURuStoreRequestMeta()
    {
        traceId = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString traceId;
};

```

- traceId — additional meta information about the purchase.

Basic response class

```

USTRUCT(BlueprintType)
struct FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    FURuStoreResponseWithCode()
    {
        code = 0;
        errorMessage = "";
        errorDescription = "";
    }

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString errorMessage;

    UPROPERTY(BlueprintReadOnly)
    FString errorDescription;

    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStoreDigitalShopGeneralError> errors;
};

```

- code — response code.
- errorMessage — error message.
- errorDescription — error description.
- errors — list of errors.

Error structure:

```

USTRUCT(BlueprintType)
struct FURuStoreDigitalShopGeneralError
{
    GENERATED_USTRUCT_BODY()

    FURuStoreDigitalShopGeneralError()
    {
        name = "";
        code = 0;
        description = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString name;

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name — error name.
- code — error code.
- description — error description.

How to handle purchases

Use the `PurchaseProduct` method to call a product purchase:

```

long requestId = URuStoreBillingClient::Instance()->PurchaseProduct(
    productId,
    orderId,
    quantity,
    developerPayload,
    [](long requestId, TShardPtr<FURuStorePaymentResult,
    ESPMode::ThreadSafe> response) {
        // Process response
    },

```

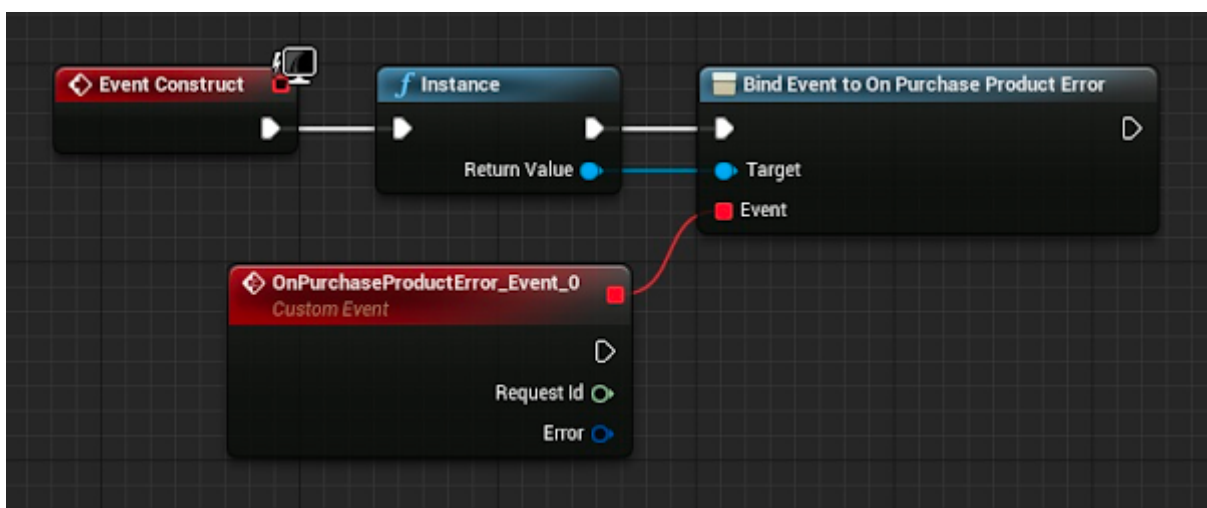
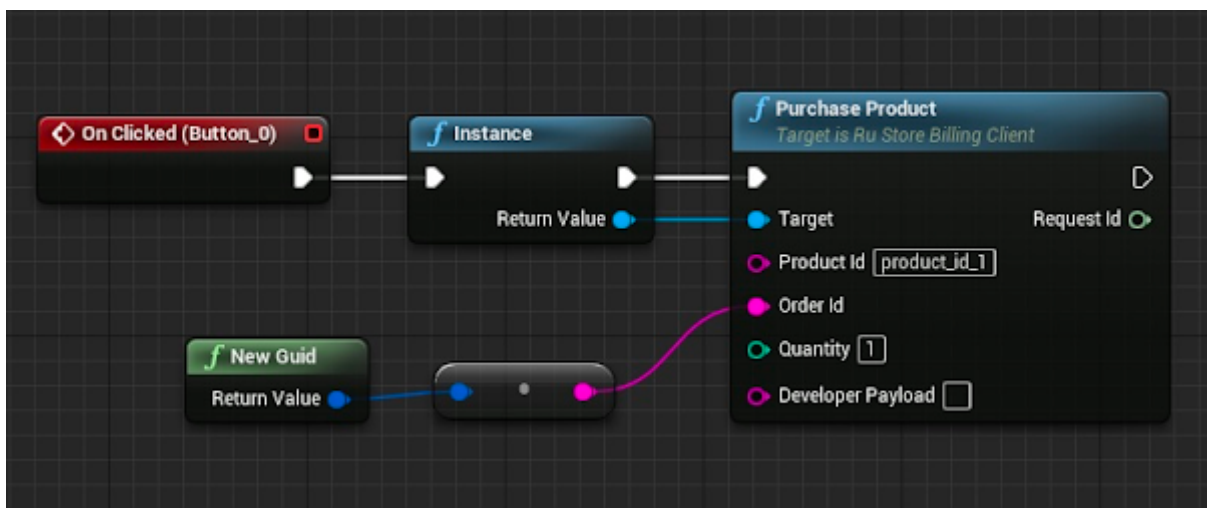
```

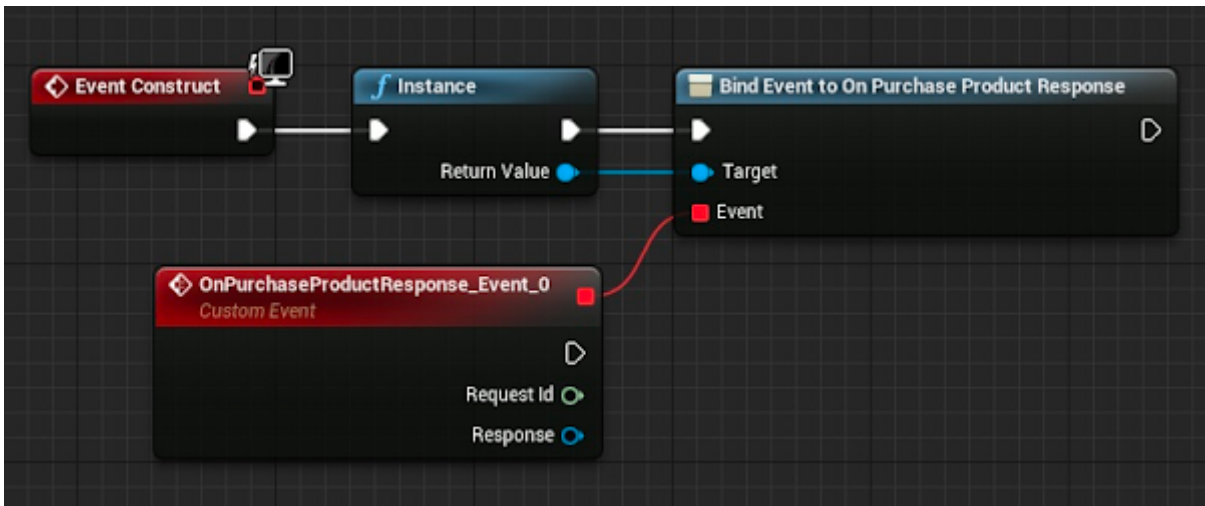
    [(long requestId, TSharedPtr<FURuStoreError,
ESPMODE::ThreadSafe> error) {
        // Process error
    }
];

```

- string productId — product identifier.
- int quantity — number of products.
- string developerPayload — additional information from the AnyApp developer.

Blueprint implementation:





Purchase result classes:

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStorePaymentResultBase : public
UObject
{
    GENERATED_BODY()
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()

    virtual ~FURuStorePaymentResult() {}

    virtual FString GetTypeName() { return
"FURuStorePaymentResult"; }
};
    
```

Purchase result structure


```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreInvoiceResult : public
URuStorePaymentResultBase
{
    GENERATED_BODY()

public:
    UPROPERTY(BlueprintReadOnly)
    FURuStoreInvoiceResult value;
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStoreInvoiceResult : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()

public:
    FURuStoreInvoiceResult()
    {
        invoiceId = "";
        finishCode =
EURuStorePaymentFinishCode::RESULT_UNKNOWN;
    }

    virtual ~FURuStoreInvoiceResult() {}

    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;

    UPROPERTY(BlueprintReadOnly)
    EURuStorePaymentFinishCode finishCode;
}

```

```

    virtual FString GetTypeName() override { return
"EURuStoreInvoiceResult"; }
};

UENUM(BlueprintType)
enum class EURuStorePaymentFinishCode : uint8
{
    SUCCESSFUL_PAYMENT UMETA(DisplayName =
"SUCCESSFUL_PAYMENT"),
    CLOSED_BY_USER UMETA(DisplayName = "CLOSED_BY_USER"),
    UNHANDLED_FORM_ERROR UMETA(DisplayName =
"UNHANDLED_FORM_ERROR"),
    PAYMENT_TIMEOUT UMETA(DisplayName = "PAYMENT_TIMEOUT"),
    DECLINED_BY_SERVER UMETA(DisplayName =
"DECLINED_BY_SERVER"),
    RESULT_UNKNOWN UMETA(DisplayName = "RESULT_UNKNOWN")
};

```

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreInvalidInvoice : public
URuStorePaymentResultBase
{
    GENERATED_BODY()

public:
    UPROPERTY(BlueprintReadOnly)
    FURuStoreInvalidInvoice value;
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStoreInvalidInvoice : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()

    FURuStoreInvalidInvoice()
    {
        invoiceId = "";
    }

    virtual ~FURuStoreInvalidInvoice() {}

    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;

    virtual FString GetTypeName() override { return
"FURuStoreInvalidInvoice"; }
};

```

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStorePurchaseResult : public
URuStorePaymentResultBase
{
    GENERATED_BODY()

public:
    UPROPERTY(BlueprintReadOnly)
    FURuStorePurchaseResult value;
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStorePurchaseResult : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()

    FURuStorePurchaseResult()
    {
        finishCode =
EURuStorePaymentFinishCode::RESULT_UNKNOWN;
        orderId = "";
        purchaseId = "";
        productId = "";
        subscriptionToken = "";
    }

    virtual ~FURuStorePurchaseResult() {}

    UPROPERTY(BlueprintReadOnly)
    EURuStorePaymentFinishCode finishCode;

    UPROPERTY(BlueprintReadOnly)
    FString orderId;

```

```

    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;

    UPROPERTY(BlueprintReadOnly)
    FString productId;

    UPROPERTY(BlueprintReadOnly)
    FString subscriptionToken;

    virtual FString GetTypeName() override { return
"FURuStorePurchaseResult"; }
};

UENUM(BlueprintType)
enum class EURuStorePaymentFinishCode : uint8
{
    SUCCESSFUL_PAYMENT UMETA(DisplayName =
"SUCCESSFUL_PAYMENT"),
    CLOSED_BY_USER UMETA(DisplayName = "CLOSED_BY_USER"),
    UNHANDLED_FORM_ERROR UMETA(DisplayName =
"UNHANDLED_FORM_ERROR"),
    PAYMENT_TIMEOUT UMETA(DisplayName = "PAYMENT_TIMEOUT"),
    DECLINED_BY_SERVER UMETA(DisplayName =
"DECLINED_BY_SERVER"),
    RESULT_UNKNOWN UMETA(DisplayName = "RESULT_UNKNOWN")
};

```

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreInvalidPurchase : public
URuStorePaymentResultBase
{
    GENERATED_BODY()

public:
    UPROPERTY(BlueprintReadOnly)
    FURuStoreInvalidPurchase value;
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStoreInvalidPurchase : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()

public:
    FURuStoreInvalidPurchase()
    {
        purchaseId = "";
        invoiceId = "";
        orderId = "";
        quantity = 0;
        productId = "";
        errorCode = 0;
    }

    virtual ~FURuStoreInvalidPurchase() {}

    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;

    UPROPERTY(BlueprintReadOnly)

```

```

    FString invoiceId;

    UPROPERTY(BlueprintReadOnly)
    FString orderId;

    UPROPERTY(BlueprintReadOnly)
    int quantity;

    UPROPERTY(BlueprintReadOnly)
    FString productId;

    UPROPERTY(BlueprintReadOnly)
    int errorCode;

    virtual FString GetTypeName() override { return
"FUruStoreInvalidPurchase"; }
};

```

Purchase structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreInvalidPaymentState : public
URuStorePaymentResultBase
{
    GENERATED_BODY()

public:
    UPROPERTY(BlueprintReadOnly)
    FUruStoreInvalidPaymentState value;
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FUruStoreInvalidPaymentState : public
FUruStorePaymentResult
{
    GENERATED_USTRUCT_BODY()
}

```

```
virtual ~FURuStoreInvalidPaymentState() {}

virtual FString GetTypeName() override { return
"FURuStoreInvalidPaymentState"; }
};
```

- URuStoreInvoiceResult — payments completed with results.
- URuStoreInvalidInvoice — payments completed without an invoice. It is probably related to an incorrect invoice (an empty line, for example).
- URuStorePurchaseResult — purchase successfully completed.
- URuStoreInvalidPurchase — payment error.
- URuStoreInvalidPaymentState — PaymentState is missing when completing payments.

EUPaymentFinishCode statuses:

- SUCCESSFUL_PAYMENT — payment successfully completed.
- CLOSED_BY_USER — payment cancelled.
- UNHANDLED_FORM_ERROR — unknown error.
- PAYMENT_TIMEOUT — timeout error.
- DECLINED_BY_SERVER — server rejected.
- RESULT_UNKNOWN — unknown payment status.

Purchase confirmation

The RuStore application consists of the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);
- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation if they are in the PurchaseState.PAID state.

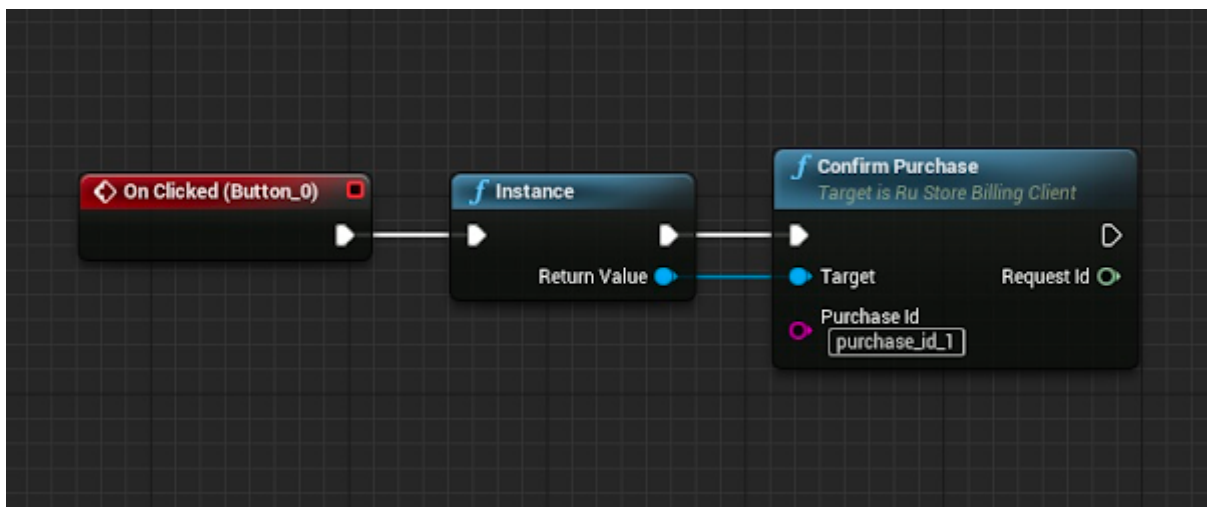
You can use the ConfirmPurchase method to confirm the purchase:

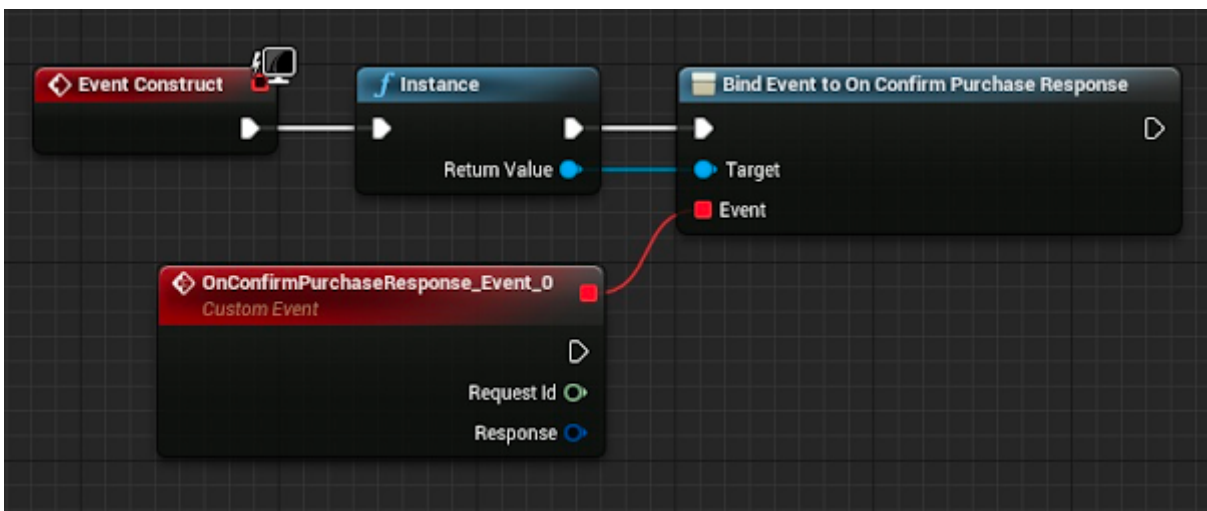
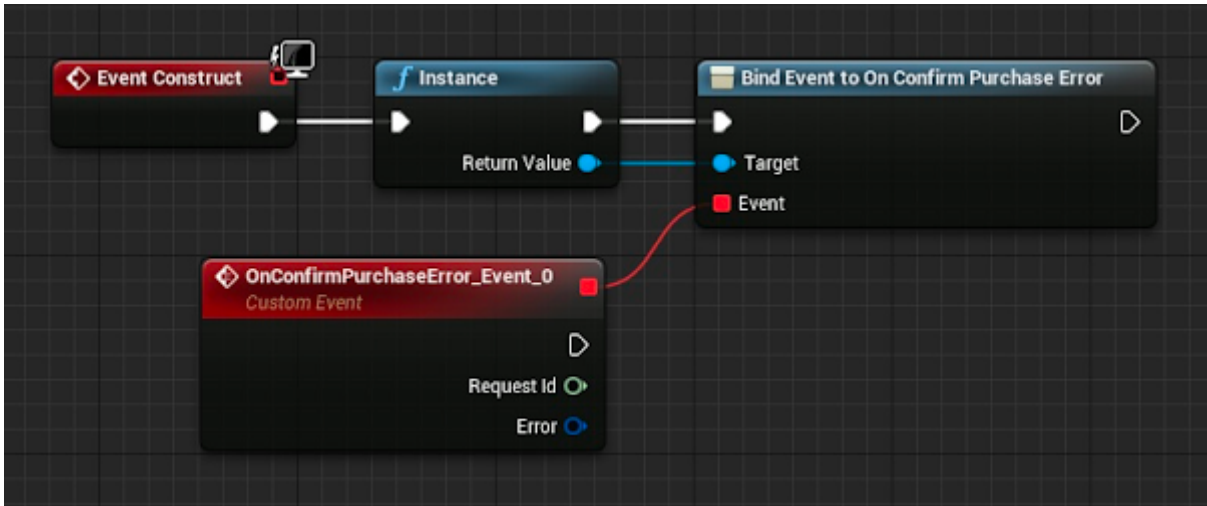
Calling confirmation method

```
long requestId =  
RuStoreBillingClient::Instance()->ConfirmPurchase(  
    purchaseId,  
    [](long requestId, TSharedPtr<FURuStoreError,  
ESPMODE::ThreadSafe> error) {  
        // Process error  
    },  
    [](long requestId,  
TSharedPtr<FURuStoreConfirmPurchaseResponse,  
ESPMODE::ThreadSafe> response) {  
        // Process response  
    }  
);
```

- purchaseld — purchase ID.

Blueprint implementation:





This method returns

ConfirmPurchase response

```

USTRUCT(BlueprintType)
struct FURuStoreConfirmPurchaseResponse : public
FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()
};

```

Basic response class

```

USTRUCT(BlueprintType)
struct FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    FURuStoreResponseWithCode()
    {
        code = 0;
        errorMessage = "";
        errorDescription = "";
    }

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString errorMessage;

    UPROPERTY(BlueprintReadOnly)
    FString errorDescription;

    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStoreDigitalShopGeneralError> errors;
};

```

- code — response code.
- errorMessage — error message.
- errorDescription — error description.
- errors — list of errors.

Error structure

```

USTRUCT(BlueprintType)
struct FURuStoreDigitalShopGeneralError
{
    GENERATED_USTRUCT_BODY()

    FURuStoreDigitalShopGeneralError()
    {
        name = "";
        code = 0;
        description = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString name;

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name — error name.
- code — error code.
- description — error description.

Purchase cancellation

You can use the **DeletePurchase** method to cancel the purchase:

```

long requestId =
URuStoreBillingClient::Instance()->DeletePurchase(
    purchaseId,
    [(long requestId,
TSharedPtr<FURuStoreDeletePurchaseResponse,
ESPMODE::ThreadSafe> response) {
    // Process response

```

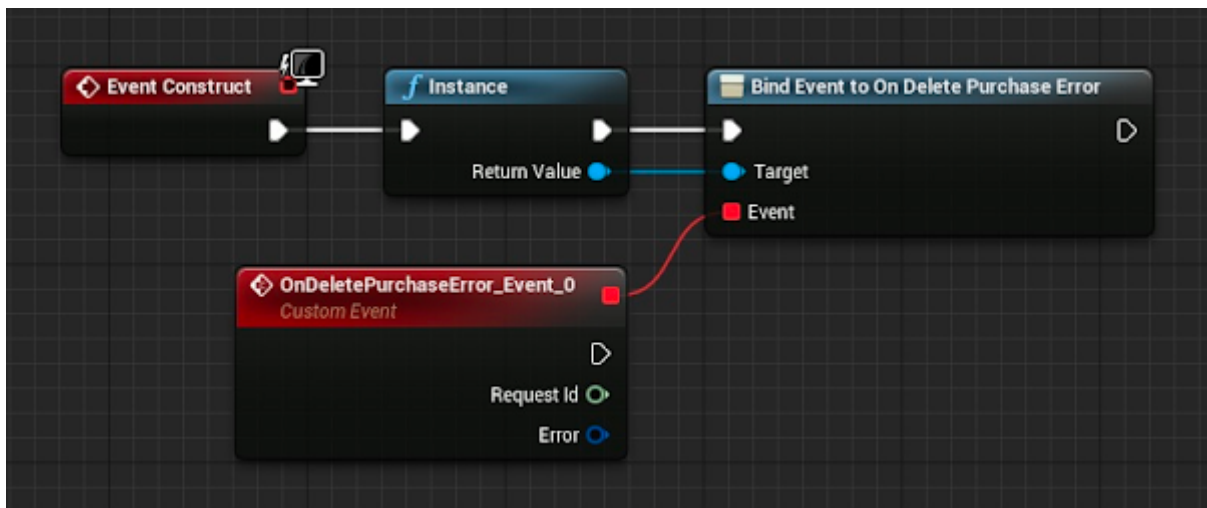
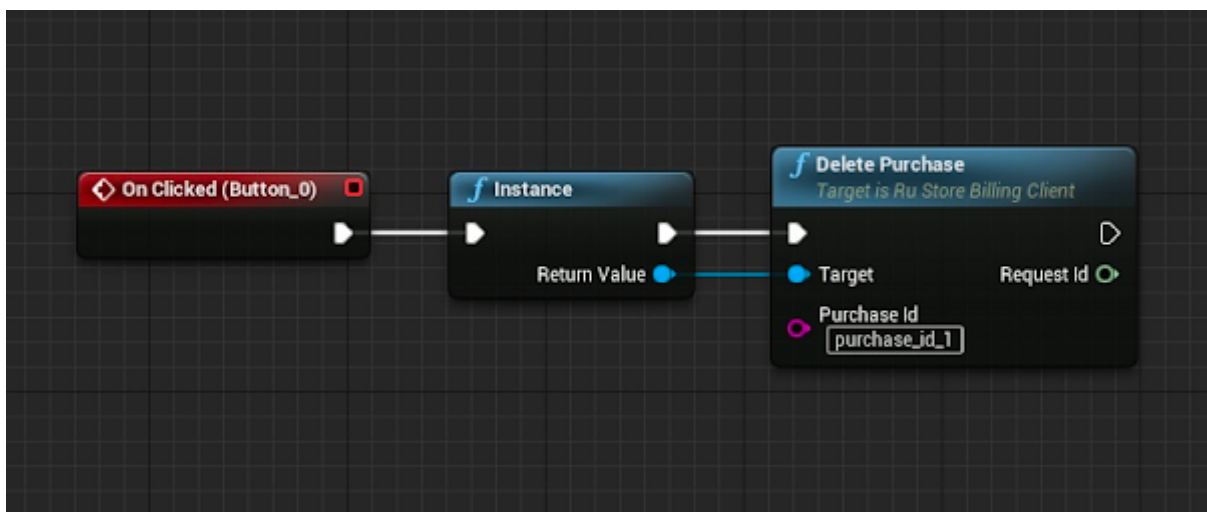
```

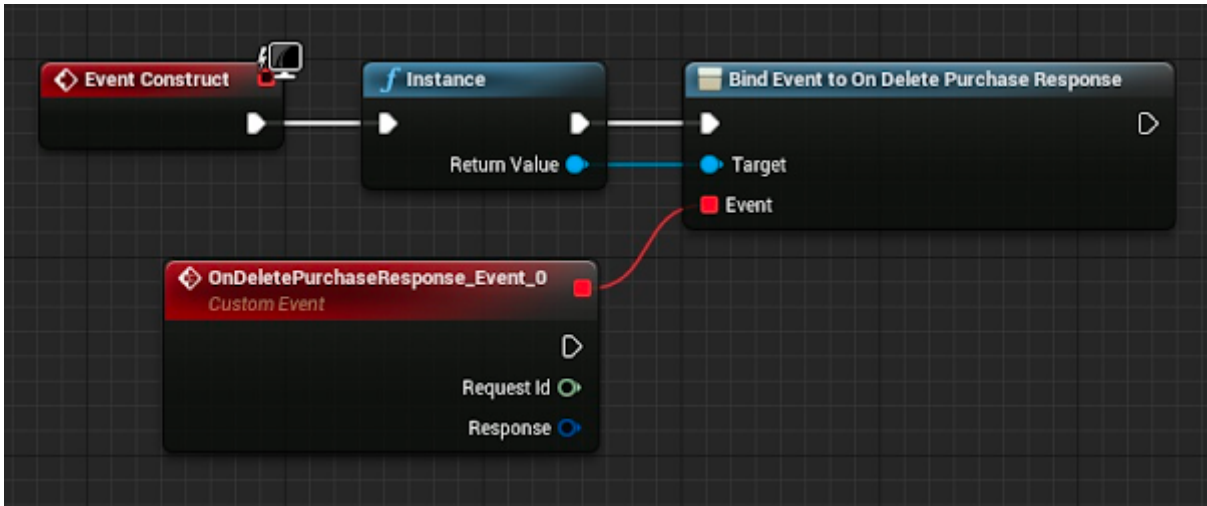
    },
    [(long requestId, TSharedPtr<FUruStoreRuStoreError,
ESPMode::ThreadSafe> error) {
        // Process error
    }
];

```

- purchaseld — purchase ID.

Blueprint implementation





This method returns:

DeletePurchase response

```

USTRUCT(BlueprintType)
struct FURuStoreDeletePurchaseResponse : public
FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()
};

```

Basic response class

```

USTRUCT(BlueprintType)
struct FURuStoreResponseWithCode
{
    GENERATED_USTRUCT_BODY()

    FURuStoreResponseWithCode()
    {
        code = 0;
        errorMessage = "";
        errorDescription = "";
    }

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString errorMessage;

    UPROPERTY(BlueprintReadOnly)
    FString errorDescription;

    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStoreDigitalShopGeneralError> errors;
};

```

- code — response code.
- errorMessage — error message.
- errorDescription — error description.
- errors — list of errors.

Error structure

```

USTRUCT(BlueprintType)
struct FURuStoreDigitalShopGeneralError
{
    GENERATED_USTRUCT_BODY()

    FURuStoreDigitalShopGeneralError()
    {
        name = "";
        code = 0;
        description = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString name;

    UPROPERTY(BlueprintReadOnly)
    int code;

    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name — error name.
- code — error code.
- description — error description.

Consumption and cancellation scenario

Uncompleted payments must be processed by the AnyApp developer.

The purchase cancellation method should be used if:

1. The method of getting the list of products returned the purchase status as follows:
 - PurchaseState.CREATED;
 - PurchaseState.INVOICE_CREATED;
2. If purchaseProduct returned PaymentResult.InvalidPurchase.
3. If purchaseProduct returned PaymentResult.PurchaseResult that contains the following PaymentFinishCode:
 - CLOSED_BY_USER — canceled by the user;

- UNHANDLED_FORM_ERROR — unknown error;
- PAYMENT_TIMEOUT — timeout payment error;
- DECLINED_BY_SERVER — rejected by server;
- RESULT_UNKNOWN — unknown payment status.

The confirmPurchase method should be used if:

1. The getPurchases method returned the purchase status as follows:
 - PurchaseState.PAID.
2. The purchaseProduct method returned PaymentResult.PurchaseResult that contains the following PaymentFinishCode:
 - SUCCESSFUL_PAYMENT — successful payment.

Error handling

All possible errors are processed by onFailure handler of SDK methods.

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreRuStoreError
{
    GENERATED_USTRUCT_BODY()

    FURuStoreRuStoreError()
    {
        name = "";
        description = "";
    }

    UPROPERTY(BlueprintReadOnly)
    FString name;

    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name — error name.
- description — error description.

Possible errors:

- `RuStoreNotInstalledException` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException` — RuStore installed on a user's device doesn't support push notifications.
- `RuStoreUserUnauthorizedException` — the user is not logged in to the RuStore.
- `RuStoreFeatureUnavailableException` — RuStore app is not allowed to run in the background.
- `RuStoreException` — RuStore basic error from which all the other errors are inherited.

When calling the `PurchaseProduct` method, errors are handled automatically.

If the `allowNativeErrorHandling == true` parameter was passed during SDK initialization, it is passed to the `resolveForBilling` method apart from calling the corresponding `onFailure` handler.

Error handling

```
public fun RuStoreException.resolveForBilling(context: Context)
```

You can change this behavior after initialization by setting the `AllowNativeErrorHandling` property:

Restriction from native error handling

```
RuStoreBillingClient::Instance()->AllowNativeErrorHandling =  
false;
```


React Native

General Information	1
Embed in your project	1
How to initialize the library	3
Payment functions availability	3
Getting the List of Products	4
Getting the List of Purchases	8
Getting Specific Purchase Info	9
How to handle purchases	10
Purchase confirmation	13
Purchase cancellation	14
Consumption and cancellation scenario	14

General Information

Example of implementation

Please have a thorough look at the [application example](#) to learn how to integrate payments correctly.

Payment integration guideline

Comply with the terms below to ensure proper payment integration in your app:

1. The RuStore app must be installed on the user's device.
2. Your app user must be authorized on the RuStore.
3. The user and the application must not be blocked on the RuStore.
4. The [RuStore Console](#) shopping option must be enabled for the application.

The service has some restrictions to work outside of Russia.

Embed in your project

To connect SDK to your project, you need to run the following code:

```
// HTTPS
npm install
git+https://git@gitflic.ru:rustore/react-native-rustore-billing-s
dk.git
```

```
// SSH
npm install
git+ssh://git@gitflic.ru:rustore/react-native-rustore-billing-sdk
.git
```

Handling deeplinks in your app

To redirect a user to your app after payment via third-party apps (the Faster Payments System (SBP), SberPay and others), you need to properly implement deep linking in your app. Specify the intent-filter with the scheme in AndroidManifest.xml:

```
<activity
  android:name=".sample.MainActivity">

  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>

  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE"
  />
    <data android:scheme="yourappscheme" />
  </intent-filter>

</activity>
```

where "yourappscheme" — your deeplink scheme, it can be changed to another one.

This scheme must match the deeplinkScheme parameter passed to init().

How to initialize the library

You must initialize the library to call its method. To do this, call the `RustoreBillingClient.initialize()` method:

```
try {
    RustoreBillingClient.initialize({
        consoleApplicationId: 'appId',
        deeplinkScheme: 'scheme',
    });
    console.log(`initialize success: ${result}`);
} catch (err) {
    console.log(`initialize err: ${err}`);
}
```

- `consoleApplicationId` — application code from RuStore Console (example: <https://console.rustore.ru/apps/123456>).
- `deeplinkScheme` — deeplink scheme required to return to your app page after payment through a third-party application (for example, SberPay or SBP). The SDK generates its own host for this scheme.

Make sure that the deeplink scheme passed to `deeplinkScheme` matches the scheme specified in `AndroidManifest.xml` in the "Deeplink Processing" section.

Payment functions availability

To check whether your app supports payment functions, the following conditions should be met:

1. The RuStore app must be installed on the user's device.
2. Your RuStore app should support the payment processing function.
3. Your app user must be authorized on the RuStore.
4. The user and the application must not be blocked on the RuStore.
5. The [RuStore Console](#) shopping option must be enabled for the application.

If all conditions are met, the `RustoreBillingClient.checkPurchasesAvailability()` method returns `true`.

```
try {
  const isAvailable = await
RustoreBillingClient.checkPurchasesAvailability();
  console.log(`available success ${isAvailable}`);
} catch (err) {
  console.log(`available error ${err}`);
}
```

Getting the List of Products

Use the `RustoreBillingClient.getProducts(productIds)` method to get a list of products:

```
try {
  const products = await
RustoreBillingClient.getProducts(productIds);
  for (const product of products) {
    console.log(product?.productId);
  }
} catch (err) {
  console.log(`products err: ${err}`);
}
```

- `productIds` — list of products IDs.

This method returns `Product[]`. Below is the product pattern:

```
interface Product {
```

```

    productId: string;
    productType?: ProductType;
    productStatus: ProductStatus;
    priceLabel?: string;
    price?: number;
    currency?: string;
    language?: string;
    title?: string;
    description?: string;
    imageUrl?: string;
    promoImageUrl?: string;
    subscription?: ProductSubscription;
}

```

- product_id — product ID;
- product_type — product type;
- product_status — product status;
- price_label — formatted product price, including currency symbol in [language];
- price — price in minimum units (in kopecks);
- currency — currency code ISO 4217;
- language — language specified using BCP 47 encoding;
- title — product name in [language];
- description — product description in [language];
- image_url — image link
- promo_image_url — promotional image link;
- subscription — subscription description, returned only for subscription type products.

Subscription structure:

```

interface ProductSubscription {
    subscriptionPeriod?: SubscriptionPeriod;
    freeTrialPeriod?: SubscriptionPeriod;
    gracePeriod?: SubscriptionPeriod;
    introductoryPrice?: string;
    introductoryPriceAmount?: string;
    introductoryPricePeriod?: SubscriptionPeriod;
}

```



```
}
```

- `subscriptionPeriod` — subscription period;
- `freeTrialPeriod` — subscription trial period;
- `gracePeriod` — subscription grace period;
- `introductoryPrice` — formatted introductory subscription price, including currency sign, in `product:language`;
- `introductoryPriceAmount` — introductory price in minimum currency units (in kopecks);
- `introductoryPricePeriod` — calculation period of the introductory price.

SubscriptionPeriod structure:

```
interface SubscriptionPeriod {  
    years: number;  
    months: number;  
    days: number;  
}
```

- `years` — number of years;
- `months` — number of months;
- `days` — number of days.

Getting the List of Purchases

Use the `RustoreBillingClient.getPurchases()` method to get the user's list of purchases

```
try {
  const purchases = await RustoreBillingClient.getPurchases();
  for (const purchase of purchases) {
    console.log(purchase?.purchaseId);
  }
} catch (err) {
  console.log(`purchase err: ${err}`);
}
```

This method returns `Product[]`. Below is the product pattern:

```
interface Purchase {
  purchaseId?: string;
  productId: string;
  productType?: ProductType;
  invoiceId?: string;
  description?: string;
  language?: string;
  purchaseTime?: string;
  orderId?: string;
  amountLabel?: string;
  amount?: number;
  currency?: string;
  quantity?: number;
  purchaseState?: PurchaseState;
  developerPayload?: string;
  subscriptionToken?: string;
}
```

- `purchase_id` — purchase ID;
- `product_id` — product ID;
- `product_type` — product type;
- `invoice_id` — account ID;
- `description` — purchase description;
- `language` — language specified using BCP 47 encoding;
- `purchase_time` — purchase time (in RFC 3339);
- `order_id` — unique payment ID generated by the application (uuid);
- `amount_lable` — formatted purchase price, including currency symbol in [language];
- `amount` — price in minimum currency units;
- `currency` — ISO 4217 currency code;
- `quantity` — quantity of product;
- `purchase_state` — purchase state;
- `developer_payload` — string specified by the developer containing additional information about the order;
- `subscription_token` — token for server purchase validation. For more information about validating a purchase on the server, see the “Server-based purchase validation” section.

Possible purchase status values:

- `CREATED` — created;
- `INVOICE_CREATED` — created, awaiting payment;
- `CONFIRMED` — confirmed;
- `PAID` — paid;
- `CANCELLED` — purchase canceled;
- `CONSUMED` — purchase confirmed;
- `CLOSED` — subscription canceled.

Getting Specific Purchase Info

To get a specific purchase, you must use the `RustoreBillingClient.getPurchaseInfo(purchaseId)` method:

```

try {
  const purchase = await
RustoreBillingClient.getPurchaseInfo('purchaseId');
  console.log(purchase?.purchaseId);
} catch (err) {
  console.log(`purchase err: ${err}`);
}

```

- purchaseId — purchase ID.

This method returns Purchase, which is described above.

How to handle purchases

Use the `RustoreBillingClient.purchaseProduct({...})` method to call a product purchase:

```

try {
  const response = await RustoreBillingClient.purchaseProduct({
    productId: 'productId',
    orderId: 'orderId',
    quantity: 0,
    developerPayload: 'developerPayload'
  });
  console.log(`purchase success: ${response}`);
} catch (err) {
  console.log(`purchase err: ${err}`);
}

```

- product_id — product ID;
- order_id — order ID, created on the AnyApp side (optional. If not specified, it is generated automatically);
- invoice_id — account ID;
- product_id — product ID;
- quantity — number of products (optional);
- payload — additional information from the AnyApp developer (optional).
- error_code — error code in case of failed request.

The purchase result can be represented as one of the following interfaces:
SuccessPayment, CanceledPayment or FailurePayment:

```
enum PaymentResult {
  SUCCESS = 'SUCCESS',
  CANCELLED = 'CANCELLED',
  FAILURE = 'FAILURE',
}

interface SuccessPaymentResult {
  orderId?: string;
  purchaseId: string;
  productId: string;
  invoiceId: string;
  subscriptionToken?: string;
}

interface SuccessPayment {
  type: PaymentResult.SUCCESS;
  result: SuccessPaymentResult;
}

interface CancelledPaymentResult {
  purchaseId: string;
}

interface CancelledPayment {
  type: PaymentResult.CANCELLED;
  result: CancelledPaymentResult;
}

interface FailurePaymentResult {
  purchaseId?: string;
  invoiceId?: string;
  orderId?: string;
  quantity?: number;
  productId?: string;
  errorCode?: number;
}

interface FailurePayment {
  type: PaymentResult.FAILURE;
  result: FailurePaymentResult;
}
```

- SuccessPayment — product successfully purchased.
- FailurePayment — purchase failed.
- CancelledPayment — purchase canceled.

Purchase confirmation

The RuStore application consists of the following types of products:

- CONSUMABLE — consumables (multiple-time purchases, such as crystals in the app);
- NON_CONSUMABLE — non-consumables (one-time purchases, such as disabling ads in an app);
- SUBSCRIPTION — subscription (can be purchased for a period of time, such as a streaming service subscription).

Only CONSUMABLE type products require confirmation if they are in the PurchaseState.PAID state.

You can use the `RustoreBillingClient.confirmPurchase({...})` method to confirm the purchase:

```
try {
  const isConfirmed = await RustoreBillingClient.confirmPurchase({
    purchaseId: 'purchaseId',
    developerPayload: 'developerPayload'
  })
  console.log(`confirm success: ${isConfirmed}`);
} catch (err) {
  console.log(`confirm err: ${err}`);
}
```

- `purchaseId` — purchase ID.
- `developerPayload` — developer-specified string containing additional information.

Provided that all conditions are met, the `RustoreBillingClient.confirmPurchase()` method returns true.

Purchase cancellation

You can use the `RustoreBillingClient.deletePurchase(purchaseId)` method to cancel the purchase:

```
try {
  const isDeleted = await
RustoreBillingClient.deletePurchase(purchaseId)
  console.log(`delete success: ${isDeleted}`);
} catch (err) {
  console.log(`delete err: ${err}`);
}
```

- `purchaseId` stands for the purchase ID.

Provided that all conditions are met, the `RustoreBillingClient.deletePurchase()` method returns `true`.

Note. Use this method if your app logic is related to purchase cancellation. The purchase is canceled automatically after a 20-min timeout, or upon a second purchase from the same customer.

Consumption and cancellation scenario

Uncompleted payments must be processed by the AnyApp developer.

The purchase cancellation method (`deletePurchase`) should be used if:

1. The method of getting the list of products (`getPurchases`) returned the purchase status as follows:
 - `PurchaseState.CREATED`;
 - `PurchaseState.INVOICE_CREATED`;
2. The purchase method (`purchaseProduct`) returned `PaymentResult.Cancelled`.
3. The purchase method (`purchaseProduct`) returned `PaymentResult.Failure`.

Use product consumption method (`confirmPurchase`) if the method the purchase obtaining method (`getPurchases`) returns a `CONSUMABLE` product and with the status `PurchaseState.PAID`.

Unreal

Unreal in-app payments plug-in (3.0)

General information

For in-app payments to work, the following requirements must be met:

1. RuStore is installed on the user's device.
2. The user is authorized in RuStore.
3. The user and the app are not banned in RuStore.
4. For the app, the purchase option is enabled in [RuStore Console](#).
5. Unreal Engine 4.26 or later.

Connecting to project

1. Copy the contents of the “*Plugins*” folder from the official RuStore repository on [gitflic](#) to the “*Plugins*” folder of your project. Restart Unreal Engine, then, in (Edit → Plugins → Project → Mobile) check plug-ins “RuStoreBilling” and “RuStoreCore”.
2. In the “*YourProject.Build.cs*” file in the PublicDependencyModuleNames list connect modules “RuStoreCore” and “RuStoreBilling”.
3. In the project settings (Edit → Project Settings → Android) set the Minimum SDK Version parameter to 24 or later and the Target SDK Version parameter: 31 or later.

Processing deeplink

When paying via SberPay, the user is forwarded to the Sber payment app. On payment completion, the user should be returned to the initial app via deeplink. To process deeplink, plug-in RuStore Billing automatically adds the AndroidManifest.xml attribute android:exported="true" for the main activity and the next intent-filter:

AndroidManifest.xml

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="yourappscheme" />
</intent-filter>
```

where “yourappscheme” — is the scheme of your deeplink, can be replaced by another one and must match the deeplinkScheme value specified during the billing client library initialization.

You can replace “yourappscheme” with yours in the “RuStoreCore_UPL_Android.xml” file.

Initialization

Initialize the library before calling its methods. Initialization parameters are configured with the help of the FURuStoreBillingClientConfig structure.

Initialization

```
FURuStoreBillingClientConfig config;
config.consoleApplicationId = "111111";
config.deeplinkScheme = "yourappscheme";
URuStoreBillingClient::Instance()->Init(config);
```

All operations with the client are also accessible from Blueprints. Initialization example:

- consoleApplicationId - app code from RuStore Console (example: <https://console.rustore.ru/apps/111111>).
- deeplinkScheme - URL for using deeplink. You can use any unique name (example: yourappscheme).
- allowNativeErrorHandling - allow native error handling (see "[Error handling](#)" for more details)
- enableLogs - enable event logging.

Important:

1. The deeplink scheme that is passed in deeplinkPrefix must match the scheme specified in AndroidManifest.xml (see "[Processing deeplink](#)" for more details).
2. The Init() call ties the object to the root of the scene. If no further work with the object is needed, execute the Dispose() method to free memory. The Dispose() method call will untie the object from root and securely complete all sent requests.

Deinitialization

```
URuStoreBillingClient::Instance()->Dispose();
```

Blueprint implementation:

If you need to check whether the library is initialized, use the GetIsInitialized() method. The method will return true if the library is initialized and false if Init hasn't been called yet.

Initialization check

```
bool isInitialized =
URuStoreBillingClient::Instance()->IsIninialized();
```

Blueprint implementation:

Payments availability check

To check whether payments are available, use the CheckPurchasesAvailability() method. On calling, the following conditions are checked:

1. RuStore is installed on the user's device.
2. RuStore supports payments.
3. The user is authorized in RuStore.
4. The user and the app are not banned in RuStore.
5. In-app purchases for the app are enabled in [RuStore Console](#).

Each CheckPurchasesAvailability() request returns a requestId that is unique per app launch. Each event returns requestId of the request that triggered this event.

CheckPurchasesAvailability request

```

long requestId =
URuStoreBillingClient::Instance()->CheckPurchasesAvailability(
    [](long requestId,
TSharedPtr<FURuStoreFeatureAvailabilityResult, ESPMode::ThreadSafe>
response) {
    // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
    // Process error
    }
);

```

Blueprint implementation:

The Success callback returns the FURuStoreFeatureAvailabilityResult structure in the Response parameter:

CheckPurchasesAvailability response

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreFeatureAvailabilityResult
{
    GENERATED_USTRUCT_BODY()
    FURuStoreFeatureAvailabilityResult()
    {
        isAvailable = false;
    }
    UPROPERTY(BlueprintReadWrite)
    bool isAvailable;

    UPROPERTY(BlueprintReadWrite)
    FURuStoreError cause;
};

```

isAvailable - whether payment conditions are met.

cause - error information.

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the [“Error handling”](#) section.

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)

```

```

    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

Working with products

Retrieving products list

To retrieve the products list, use the `GetProducts()` method.

GetProducts request

```

long requestId = URuStoreBillingClient::Instance()->GetProducts(
    productIds,
    [](long requestId, TSharedPtr<FURuStoreProductsResponse,
    ESPMode::ThreadSafe> response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);

```

- `TArray<FString> productIds` - list of product identifiers.

Blueprint implementation:

The Success callback returns the `FURuStoreProductsResponse` structure in the Response parameter:

GetProducts response

```

USTRUCT(BlueprintType)
struct FURuStoreProductsResponse
{
    GENERATED_USTRUCT_BODY()
    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStoreProduct> products;
};

```

- products - products list.

Product structure

```

USTRUCT(BlueprintType)
struct FURuStoreProduct
{
    GENERATED_USTRUCT_BODY()
    FURuStoreProduct()
    {

```

```

        productId = "";
        productType = EURuStoreProductType::NON_CONSUMABLE;
        productStatus = EURuStoreProductStatus::INACTIVE;
        priceLabel = "";
        price = 0;
        currency = "";
        language = "";
        title = "";
        description = "";
        imageUrl = "";
        promoImageUrl = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString productId;
    UPROPERTY(BlueprintReadOnly)
    EURuStoreProductType productType;
    UPROPERTY(BlueprintReadOnly)
    EURuStoreProductStatus productStatus;
    UPROPERTY(BlueprintReadOnly)
    FString priceLabel;
    UPROPERTY(BlueprintReadOnly)
    int price;
    UPROPERTY(BlueprintReadOnly)
    FString currency;
    UPROPERTY(BlueprintReadOnly)
    FString language;
    UPROPERTY(BlueprintReadOnly)
    FString title;
    UPROPERTY(BlueprintReadOnly)
    FString description;
    UPROPERTY(BlueprintReadOnly)
    FString imageUrl;
    UPROPERTY(BlueprintReadOnly)
    FString promoImageUrl;
    UPROPERTY(BlueprintReadOnly)
    FURuStoreProductSubscription subscription;
};

```

- productId - product identifier.
- productType - product type.
- productStatus - product status.
- priceLabel - formatted product price, including currency symbol in language FURuStoreProduct::language.
- price - price in minor currency units.
- currency - ISO 4217 currency code.
- language - language specified by BCP 47 encoding.
- title - product name in language FURuStoreProduct::language.
- description - product description in language FURuStoreProduct::language.

- imageUrl - image URL.
- promoImageUrl - promo image URL.
- subscription - subscription description, returns only for products with type subscription.

Product type

```
UENUM(BlueprintType)
enum class EURuStoreProductType : uint8
{
    NON_CONSUMABLE UMETA(DisplayName = "NON_CONSUMABLE"),
    CONSUMABLE UMETA(DisplayName = "CONSUMABLE"),
    SUBSCRIPTION UMETA(DisplayName = "SUBSCRIPTION")
};
```

Product status

```
UENUM(BlueprintType)
enum class EURuStoreProductStatus : uint8
{
    ACTIVE UMETA(DisplayName = "ACTIVE"),
    INACTIVE UMETA(DisplayName = "INACTIVE")
};
```

Subscription structure

```
USTRUCT(BlueprintType)
struct FURuStoreProductSubscription
{
    GENERATED_USTRUCT_BODY()
    FURuStoreProductSubscription()
    {
        introductoryPrice = "";
        introductoryPriceAmount = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod subscriptionPeriod;
    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod freeTrialPeriod;
    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod gracePeriod;
    UPROPERTY(BlueprintReadOnly)
    FString introductoryPrice;
    UPROPERTY(BlueprintReadOnly)
    FString introductoryPriceAmount;
    UPROPERTY(BlueprintReadOnly)
    FURuStoreSubscriptionPeriod introductoryPricePeriod;
};
```

- subscriptionPeriod - subscription period.
- freeTrialPeriod - free trial period.
- gracePeriod - grace period.
- introductoryPrice - formatted introductory subscription price including currency symbol, in language FURuStoreProduct::language.

- introductoryPriceAmount - initial price in minor currency units.
- introductoryPricePeriod - introductory price period.

Subscription period structure

```
USTRUCT(BlueprintType)
struct FURuStoreSubscriptionPeriod
{
    GENERATED_USTRUCT_BODY()
    FURuStoreSubscriptionPeriod()
    {
        years = 1970;
        months = 1;
        days = 1;
    }
    UPROPERTY(BlueprintReadOnly)
    int years;
    UPROPERTY(BlueprintReadOnly)
    int months;
    UPROPERTY(BlueprintReadOnly)
    int days;
};
```

- years - number of years.
- months - number of years.
- days - number of days.

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};
```

- name - error name.
- description - error description.

Working with purchases

Retrieving purchases list

To retrieve the purchases list, use the `GetPurchases()` method.

Retrieving the user's purchases list

```
long requestId = URuStoreBillingClient::Instance()->GetPurchases(
    [](long requestId, TSharedPtr<FURuStorePurchasesResponse,
    ESPMode::ThreadSafe> response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreRuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```

Blueprint implementation:

The Success callback returns the `FURuStorePurchasesResponse` structure in the Response parameter:

GetPurchases response

```
USTRUCT(BlueprintType)
struct FURuStorePurchasesResponse
{
    GENERATED_USTRUCT_BODY()
    UPROPERTY(BlueprintReadOnly)
    TArray<FURuStorePurchase> purchases;
};
```

- purchases - list of purchases.

Purchases information structure

```
USTRUCT(BlueprintType)
struct FURuStorePurchase
{
    GENERATED_USTRUCT_BODY()
    FURuStorePurchase()
    {
        purchaseId = "";
        productId = "";
        invoiceId = "";
        description = "";
        language = "";
        purchaseTime = FDateTime(0);
        orderId = "";
        amountLabel = "";
        amount = 0;
        currency = "";
        quantity = 0;
        purchaseState = EURuStorePurchaseState::CANCELLED;
        developerPayload = "";
    }
};
```

```

        subscriptionToken = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;
    UPROPERTY(BlueprintReadOnly)
    FString productId;
    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;
    UPROPERTY(BlueprintReadOnly)
    FString description;
    UPROPERTY(BlueprintReadOnly)
    FString language;
    UPROPERTY(BlueprintReadOnly)
    FDateTime purchaseTime;
    UPROPERTY(BlueprintReadOnly)
    FString purchaseTimeLabel;
    UPROPERTY(BlueprintReadOnly)
    FString orderId;
    UPROPERTY(BlueprintReadOnly)
    FString amountLabel;
    UPROPERTY(BlueprintReadOnly)
    int amount;
    UPROPERTY(BlueprintReadOnly)
    FString currency;
    UPROPERTY(BlueprintReadOnly)
    int quantity;
    UPROPERTY(BlueprintReadOnly)
    EURuStorePurchaseState purchaseState;
    UPROPERTY(BlueprintReadOnly)
    FString developerPayload;
    UPROPERTY(BlueprintReadOnly)
    FString subscriptionToken;
};

```

- purchaseId - purchase identifier.
- productId - product identifier.
- description - purchase description.
- invoiceId - invoice identifier.
- language - language specified by BCP 47 encoding.
- purchaseTime - purchase time
- purchaseTimeLabel - purchase time in the DD.MM.YYYY HH:MM:SS format
- orderId - payment identifier generated by the app (uuid).
- amountLabel - formatted purchase price, including currency symbol, in language EURuStorePurchase::language.
- amount - price in minor currency units.
- currency - ISO 4217 currency code.
- quantity - quantity of the product.
- purchaseState - purchase state.

- developerPayload - a text string specified by the developer that contains additional information about the order.
- subscriptionToken - token for the server validation of the purchase.

Purchase state

```
UENUM(BlueprintType)
enum class EURuStorePurchaseState : uint8
{
    CREATED UMETA(DisplayName = "CREATED"),
    INVOICE_CREATED UMETA(DisplayName = "INVOICE_CREATED"),
    CONFIRMED UMETA(DisplayName = "CONFIRMED"),
    PAID UMETA(DisplayName = "PAID UMETA"),
    CANCELLED UMETA(DisplayName = "CANCELLED"),
    CONSUMED UMETA(DisplayName = "CONSUMED"),
    CLOSED UMETA(DisplayName = "CLOSED")
};
```

- CREATED - purchase created.
- INVOICE_CREATED - purchase created, awaiting payment.
- CONFIRMED - purchase confirmed.
- PAID - purchase paid.
- CANCELLED - purchase canceled.
- CONSUMED - purchase consumed.
- CLOSED - subscription closed.

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};
```

- name - error name.
- description - error description.

Product purchase

To make a product purchase, use the `PurchaseProduct()` method.

Purchase product request

```
long requestId = URuStoreBillingClient::Instance()->PurchaseProduct(
    productId,
    orderId,
    quantity,
    developerPayload,
    [](long requestId, TShardPtr<FURuStorePaymentResult,
    ESPMode::ThreadSafe> response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```

- string `productId` - product identifier.
- int `quantity` - quantity of products.
- string `developerPayload` - additional information from the AnyApp developer.

Blueprint implementation:

The `Success` callback returns a managed UE pointer (doesn't require manual removal) for the `URuStorePaymentResultClass` class in the `Response` parameter:

Purchase result structure

```
UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStorePaymentResultClass : public UObject
{
    GENERATED_BODY()
};
USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()
    virtual ~FURuStorePaymentResult() {}
    virtual FString GetTypeName() { return "FURuStorePaymentResult";
}
};
```

Purchase result structure

```
UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreSuccess : public
URuStorePaymentResultClass
{
    GENERATED_BODY()
public:
    UPROPERTY(BlueprintReadOnly)
```

```

    FURuStoreSuccess value;
};
USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStoreSuccess : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()
    FURuStoreSuccess()
    {
        orderId = "";
        purchaseId = "";
        productId = "";
        invoiceId = "";
        subscriptionToken = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString orderId;
    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;
    UPROPERTY(BlueprintReadOnly)
    FString productId;
    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;
    UPROPERTY(BlueprintReadOnly)
    FString subscriptionToken;
    virtual FString GetTypeName() override { return
"FURuStoreSuccess"; }
};

```

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreCancelled : public
URuStorePaymentResultClass
{
    GENERATED_BODY()
public:
    UPROPERTY(BlueprintReadOnly)
    FURuStoreCancelled value;
};
USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStoreCancelled : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()
    FURuStoreCancelled()
    {
        purchaseId = "";
    }
    UPROPERTY(BlueprintReadOnly)

```

```

    FString purchaseId;
    virtual FString GetTypeName() override { return
"FUruStoreCancelled"; }
};

```

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreFailure : public
URuStorePaymentResultClass
{
    GENERATED_BODY()
public:
    UPROPERTY(BlueprintReadOnly)
    FUruStoreFailure value;
};

USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FUruStoreFailure : public
FUruStorePaymentResult
{
    GENERATED_USTRUCT_BODY()
public:
    FUruStoreFailure()
    {
        purchaseId = "";
        invoiceId = "";
        orderId = "";
        quantity = 0;
        productId = "";
        errorCode = 0;
    }
    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;
    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;
    UPROPERTY(BlueprintReadOnly)
    FString orderId;
    UPROPERTY(BlueprintReadOnly)
    int quantity;
    UPROPERTY(BlueprintReadOnly)
    FString productId;
    UPROPERTY(BlueprintReadOnly)
    int errorCode;
    virtual FString GetTypeName() override { return
"FUruStoreFailure"; }
};

```

Purchase result structure

```

UCLASS(BlueprintType)
class RUSTOREBILLING_API URuStoreInvalidPaymentState : public
URuStorePaymentResultBase

```

```

{
    GENERATED_BODY()
public:
    UPROPERTY(BlueprintReadOnly)
    FURuStoreInvalidPaymentState value;
};
USTRUCT(BlueprintType)
struct RUSTOREBILLING_API FURuStoreInvalidPaymentState : public
FURuStorePaymentResult
{
    GENERATED_USTRUCT_BODY()

    virtual FString GetTypeName() override { return
"FURuStoreInvalidPaymentState"; }
};

```

- Success - result of a successful purchase completion.
- Failure - digital product purchase failure result.
- Cancelled - digital product purchase cancellation result.
- InvalidPaymentState - SDK payments error. May occur if the deeplink is processed incorrectly.

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the “[Error handling](#)” section.

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

Purchase consumption (approval)

RuStore deals with the following types of products:

- CONSUMABLE - consumable products (can be purchased numerous times, for example: crystals in the app).

- NON_CONSUMABLE - non-consumable products (can be purchased only once, for example: disabling ads in the app).
- SUBSCRIPTION - product subscription (can be purchased for a period of time, for example: a streaming service subscription).

Only CONSUMABLE products need consumption (approval)—if they are in the PurchaseState.PAID state.

To consume (approve) a purchase, use the ConfirmPurchase method:

ConfirmPurchase request

```
long requestId = RuStoreBillingClient::Instance()->ConfirmPurchase(
    purchaseId,
    [](long requestId) {
        // Process error
    },
    [](long requestId, TSharedPtr<FURuStoreConfirmPurchaseResponse,
    ESPMode::ThreadSafe> response) {
        // Process response
    }
);
```

- purchaseId - purchase identifier.

Blueprint implementation:

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};
```

- name - error name.
- description - error description.

Purchase cancellation

To cancel a purchase, use the DeletePurchase method:

Purchase cancellation request

```
long requestId = URuStoreBillingClient::Instance()->DeletePurchase(
    purchaseId,
    [](long requestId) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreRuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```

- purchaseId - purchase identifier.

Important

Use this method only if your business logic implies manual purchase cancellation.

An unpaid purchase is canceled automatically after 20 minute timeout or on a subsequent purchase of the same customer.

Blueprint implementation:

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};
```

- name - error name.
- description - error description.

Purchase information

To retrieve purchase information, use the GetPurchaseInfo method:

Purchase cancellation request

```
long requestId = URuStoreBillingClient::Instance()->GetPurchaseInfo(
```

```

        purchaseId,
        [](long requestId, TSharedPtr<FURuStorePurchase,
ESPMode::ThreadSafe> response) {
            // Process response
        },
        [](long requestId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
            // Process error
        }
    };
);

```

- purchaseId - purchase identifier.

Blueprint implementation:

The Success callback returns the FURuStorePurchase structure in the Response parameter:

Purchases information structure

```

USTRUCT(BlueprintType)
struct FURuStorePurchase
{
    GENERATED_USTRUCT_BODY()
    FURuStorePurchase()
    {
        purchaseId = "";
        productId = "";
        invoiceId = "";
        description = "";
        language = "";
        purchaseTime = FDateTime(0);
        orderId = "";
        amountLabel = "";
        amount = 0;
        currency = "";
        quantity = 0;
        purchaseState = EURuStorePurchaseState::CANCELLED;
        developerPayload = "";
        subscriptionToken = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString purchaseId;
    UPROPERTY(BlueprintReadOnly)
    FString productId;
    UPROPERTY(BlueprintReadOnly)
    FString invoiceId;
    UPROPERTY(BlueprintReadOnly)
    FString description;
    UPROPERTY(BlueprintReadOnly)

```

```

FString language;
UPROPERTY(BlueprintReadOnly)
FDateTime purchaseTime;
UPROPERTY(BlueprintReadOnly)
FString purchaseTimeLabel;
UPROPERTY(BlueprintReadOnly)
FString orderId;
UPROPERTY(BlueprintReadOnly)
FString amountLabel;
UPROPERTY(BlueprintReadOnly)
int amount;
UPROPERTY(BlueprintReadOnly)
FString currency;
UPROPERTY(BlueprintReadOnly)
int quantity;
UPROPERTY(BlueprintReadOnly)
EURuStorePurchaseState purchaseState;
UPROPERTY(BlueprintReadOnly)
FString developerPayload;
UPROPERTY(BlueprintReadOnly)
FString subscriptionToken;
};

```

The Failure callback returns the `FURuStoreError` structure with the error information. All possible `FURuStoreException` errors are described in the [“Error handling”](#).

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

Error handling

Errors that occur are passed to the `onFailure` handler of the SDK methods.

Error structure

```

USTRUCT(BlueprintType)

```

```

struct RUSTORECORE_API FURuStoreRuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreRuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - name of the error.
- description - error description.

Possible errors:

- RuStoreNotInstalledException - RuStore is not installed on the user's device.
- RuStoreOutdatedException - the RuStore app installed on the user's devices doesn't support payments.
- RuStoreUserUnauthorizedException - the user is not authorized in RuStore.
- RuStoreApplicationBannedException - the app is banned in RuStore.
- RuStoreUserBannedException - the user is banned in RuStore.
- RuStoreException - base RuStore error from which all other errors are inherited.

On PurchaseProduct call, errors are handled automatically.

If allowNativeErrorHandling == true was passed during the SDK initialization and an error occurs, aside from calling the onFailure handler, this error is passed to the resolveForBilling method of the native SDK to show the error dialog to the user:

Error handling

```
public fun RuStoreException.resolveForBilling(context: Context)
```

You can change this behavior after the initialization by setting AllowNativeErrorHandling property:

Deny native error handling

```
RuStoreBillingClient::Instance()->SetAllowNativeErrorHandling(false);
```

Theme change

To dynamically change theme, use the SetTheme method:

SetTheme request

```
EURuStoreTheme theme = EURuStoreTheme::DARK;
URuStoreBillingClient::Instance()->SetTheme(theme);
```

- theme - theme type from the EURuStoreTheme enumeration.

Blueprint implementation:

To retrieve the information about the set theme, use the `GetTheme` method:

SetTheme request

```
EURuStoreTheme theme = URuStoreBillingClient::Instance()->GetTheme();
```

Blueprint implementation:

Theme type

```
UENUM(BlueprintType)
enum class EURuStoreTheme : uint8
{
    DARK UMETA(DisplayName = "DARK"),
    LIGHT UMETA(DisplayName = "LIGHT")
};
```

Purchase confirmation and cancellation scenario

Due to the change of the product purchase model result, the business logic of the purchase confirmation and cancellation was also changed.

Use the `DeletePurchase()` method if:

1. The `GetPurchases()` method returned a purchase with the following status:
 1. `FURuStorePurchaseState::CREATED`.
 2. `FURuStorePurchaseState::INVOICE_CREATED`.
2. The `PurchaseProduct()` methods returned `FURuStorePaymentResult::CANCELLED`.
3. The `PurchaseProduct()` method returned `FURuStorePaymentResult::FAILURE`.

Use the `ConfirmPurchase()` method if the `GetPurchases()` method returned a `CONSUMABLE` purchase with the `FURuStorePurchaseState::PAID` status.

Test data

[Test payment cards.](#)

Push notifications SDK

Learn how to enable push notifications in your app depending on your development environment.

Kotlin

General	231
Checking ability to receive push notification	235
Methods for working with push token	236
Getting data from RuStore SDK	237
Notification structure	239
Event logging	241

Error handling	243
RuStore SDK for revision history	244

General

Implementation example

See the [example app](#) to learn how to enable push notifications correctly.

Push notifications enabling conditions

For push notifications to be enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.
5. App signature must match the one added to the RuStore Console.

How to add a repository

Enable local repository:

```
repositories {
    maven {
        url =
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
    }
}
```

Dependency injection

To enable dependency, add the following code to your configuration file:

```
dependencies {
    implementation("ru.rustore.sdk:pushclient:1.2.0")
}
```

Editing your app's manifest

Declare service extending RuStoreMessagingService:

```
<service
```

```

        android:name=".MyRuStoreMessagingService"
        android:exported="true"
        tools:ignore="ExportedService">
        <intent-filter>
            <action
android:name="ru.rustore.sdk.pushclient.MESSAGING_EVENT" />
            </intent-filter>
        </service>

```

You can add the following metadata if you wish to change icon or color of standard notification:

```

<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_icon"
    android:resource="@drawable/ic_baseline_android_24" />
<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_color"
    android:resource="@color/your_favorite_color" />

```

You can add the following metadata to redefine notification channel:

```

<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_channel_id"
    android:value="@string/pushes_notification_channel_id" />

```

When adding your own push notification channel, you have to create the channel yourself.

Initialization

For initialization, you will need a project ID, which can be obtained in the [RuStore Console](#). To do this, on the application page, go to the "Push notifications" section and select "Projects".

ID проекта

wANNRz5N [redacted] 

Отпечаток подписи SHA-256

B4:74:02:45:29:E5:89:A2:C2:F6:1E [redacted]
DF:61

Сервисные токены

Создано: 1/5

[+ Создать](#)



Htdbx-mZfl-hs [redacted]
tT7f7i8wA



```
class App : Application() {  
  
    override fun onCreate() {  
        super.onCreate()  
        RuStorePushClient.init(  
            application = this,  
            projectId = "i5UTx96jw6c1C9Lvd1E4cdNrWHMnyRBt",  
            logger = DefaultLogger()  
        )  
    }  
}
```

Add to your project's 'Application' the following code for initialization:

- application — instance of 'Application' class;
- projectId — your project's identifier in VKPNS system;
- logger — logger, output to logcat is used by default.

Checking ability to receive push notification

For push notifications to be enabled, all the conditions must be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.

You can use `RuStorePushClient.checkPushAvailability` method to check fulfillment of these conditions:

```
RuStorePushClient.checkPushAvailability()
    .addOnCompleteListener(object :
OnCompleteListener<FeatureAvailabilityResult> {
    override fun onSuccess(result: FeatureAvailabilityResult) {
        when (result) {
            FeatureAvailabilityResult.Available -> {
                // Process push available
            }

            is FeatureAvailabilityResult.Unavailable -> {
                result.cause.resolveForPush(requireContext())
            }
        }
    }

    override fun onFailure(throwable: Throwable) {
        // Process error
    }
})
```

- context — context in the app.

Methods for working with push token

Getting user's push token

After initialization of the library, you can use `RuStorePushClient.getToken()` method to get the user's current push token.

If the user has no push token, the method will create and return a new push token.

```
RuStorePushClient.getToken().addOnCompleteListener(object :  
OnCompleteListener<String?> {  
    override fun onFailure(throwable: Throwable) {  
        // Process error  
    }  
  
    override fun onSuccess(result: String?) {  
        // Process success  
    }  
})
```

Deleting user's push token

You can use `RuStorePushClient.deleteToken()` method to delete the user's current push token.

```
RuStorePushClient.deleteToken().addOnCompleteListener(object :  
OnCompleteListener<Unit> {  
    override fun onFailure(throwable: Throwable) {  
        // Process error  
    }  
  
    override fun onSuccess(result: Unit) {  
        // Process success  
    }  
})
```

Methods for working with push topics

Getting user's push topic

After initialization of the library, you can use

`RuStorePushClient.subscribeToTopic("your_topic_name")` method to get the user's current push topics.

```
RuStorePushClient.subscribeToTopic("your_topic_name").addOnCompleteListener(object : OnCompleteListener<Unit> {  
    override fun onFailure(throwable: Throwable) {  
        // Process subscribe error  
    }  
  
    override fun onSuccess(result: Unit) {  
        // Process subscribe success  
    }  
})
```

Unsubscribing from user's push topic

You can use `RuStorePushClient.unsubscribeToTopic("your_topic_name")` method to delete the user's current push topic.

```
RuStorePushClient.unsubscribeFromTopic("your_topic_name").addOnCompleteListener(object : OnCompleteListener<Unit> {  
    override fun onFailure(throwable: Throwable) {  
        // Process unsubscribe error  
    }  
  
    override fun onSuccess(result: Unit) {  
        // Process unsubscribe success  
    }  
})
```

Getting data from RuStore SDK

To get data from RuStoreSDK, create your service which is inherited from RuStoreMessagingService:

```
class MessagingService: RuStoreMessagingService() {  
  
    override fun onNewToken(token: String) {  
    }  
  
    override fun onMessageReceived(message: RemoteMessage) {  
    }  
  
    override fun onDeletedMessages() {  
    }  
  
    override fun onError(errors:  
List<RuStorePushClientException>) {  
    }  
}
```

Methods

1. **onNewToken** — will be called when a new push token is received. After this method is called, your app will be responsible for delivering the new push token to its server.
2. **onMessageReceived** — will be called when a new push notification is received. If there is data in 'notification' object, RuStoreSDK will display the notification itself. If you don't want RuStoreSDK to display notification itself, use 'data' object, and leave 'notification' object empty. However, onMessageReceived will be called in any case. Push notification's payload (Map<String, String>) may be obtained from message.data field.
3. **onDeletedMessages** — will be called if one or more push notifications have not been delivered to device. This may happen, for example, due to expiry of notification's lifetime before it is delivered to device. When this method is called, synchronizing with your server is recommended to avoid missing any data.
4. **onError** — will be called if an error occurs at time of initialization.

Possible errors

- `UnauthorizedException` — the user has not logged in to the RuStore.
- `HostAppNotInstalledException` — user's device doesn't have RuStore app installed.
- `HostAppBackgroundWorkPermissionNotGranted` — RuStore app is not allowed to run in the background.

All the methods will be called in the background.

Notification structure

Full notification structure

```
public data class RemoteMessage(  
    val messageId: String?,  
    val priority: Int,  
    val ttl: Int,  
    val collapseKey: String?,  
    val data: Map<String, String>,  
    val rawData: ByteArray?,  
    val notification: Notification?  
)
```

- `messageId` — unique ID of message. It is the identifier of each message;
- `priority` — returns priority value (not taken into account at the moment). Possible values:
 - 0 — UNKNOWN;
 - 1 - HIGH;
 - 2 - NORMAL.
- `ttl` — returns `Int` type push notification's lifetime in seconds;
- `collapseKey` — identifier of a group of notifications (not taken into account at the moment);
- `data` — a dictionary to which additional data for notification can be sent;
- `rawData` — 'data' dictionary in the form of a binary array;
- `notification` — notification object.

Structure of Notification object

```
public data class Notification(  
    val title: String?,  
    val body: String?,  
    val channelId: String?,  
    val imageUrl: Uri?,  
    val color: String?,  
    val icon: String?,  
    val clickAction: String?  
)
```

- `title` — notification's title;
- `body` — notification's body;
- `channelId` — option to create the channel to which notification will be sent (for Android 8.0 or later);

- imageUrl — a direct link to image for insertion to notification (maximum size 1 MB);
- color — notification's color (Notification.color). Color needs to be sent in hex format, as a line (Example: #A52A2A);
- icon — notification's icon. Icon should be located in the app's resources (res/drawable). Parameter's value is a line that matches the resource's name:
 - small_icon.xml icon is located in res/drawable, which is accessible in code via R.drawable.small_icon. For this icon to be displayed in notification, the server should place 'small_icon' value to 'icon' parameter.
- clickAction — 'intent action' with which activity is opened when a notification is pressed on.

Creating channel for sending notification

The following order of priority is used for the channel to which notification will be sent:

1. If there is 'channelId' field in push notification, then RuStoreSDK will send notification to this channel. Note that your app is responsible for creating this channel in advance.
2. If there is no 'channelId' field in push notification, but your app has specified parameter with channel in AndroidManifest.xml, then channel from AndroidManifest.xml will be used. Your app is responsible for creating the channel.
3. If there is no 'channelId' field in push notification, and no default channel has been set in your app's AndroidManifest.xml, then RuStoreSDK will create channel itself and will send notification to it. All further notifications with no channel specified will be sent to this channel.

Opening Activity when notification is pressed on

By default, RuStoreSDK opens activity with 'android.intent.action.MAIN' action whenever a notification is pressed on. If 'clickAction' field is present, RuStoreSDK will open activity which falls under 'Intent filter' with specified 'action'.

For activity to open in RuStoreSDK when a notification is pressed on (this also applies to default activity), add <category android:name="android.intent.category.DEFAULT" /> line in the corresponding activity's <intent-filter> element in app's manifest. Activity will not open in RuStoreSDK without this line.

Event logging

If you wish to log events of push notification library, add 'logger' parameter to RuStorePushClient.init call (this parameter is not required for initialization).

To do this, 'Logger' interface needs to be implemented:

```
interface Logger {  
  
    fun verbose(message: String, throwable: Throwable? = null)  
    fun debug(message: String, throwable: Throwable? = null)  
    fun info(message: String, throwable: Throwable? = null)  
    fun warn(message: String, throwable: Throwable? = null)  
    fun error(message: String, throwable: Throwable? = null)  
  
    fun createLogger(tag: String): Logger  
}
```

If Logger has not been sent, then default implementation with AndroidLog will be used.

```
public class DefaultLogger(  
    private val tag: String? = null,  
) : Logger {  
    override fun verbose(message: String, throwable: Throwable?) {  
        Log.v(tag, message, throwable)  
    }  
  
    override fun debug(message: String, throwable: Throwable?) {  
        Log.d(tag, message, throwable)  
    }  
  
    override fun info(message: String, throwable: Throwable?) {  
        Log.i(tag, message, throwable)  
    }  
  
    override fun warn(message: String, throwable: Throwable?) {  
        Log.w(tag, message, throwable)  
    }  
  
    override fun error(message: String, throwable: Throwable?) {  
        Log.e(tag, message, throwable)  
    }  
  
    override fun createLogger(tag: String): Logger {  
        val newTag = if (this.tag != null) {
```

```
        "${this.tag}:$tag"
    } else {
        tag
    }
    return DefaultLogger(newTag)
}
}
```

Error handling

Possible errors:

- `RuStoreNotInstalledException()` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException()` — RuStore installed on a user's device doesn't support push notifications.
- `RuStoreUserUnauthorizedException()` — the user is not logged in to the RuStore.
- `RuStoreFeatureUnavailableException()` — RuStore app is not allowed to run in the background.
- `RuStoreException(message: String)` — RuStore basic error from which all the other errors are inherited.

If you wish to use UI interface for error handling, then use `resolveForPush()` method:

```
fun RuStoreException.resolveForPush(context: Context)
```

E2E Testing of Push Notifications SDK

To have testing enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.

Enable the test mode to start testing the SDK:

```
RuStorePushClient.init(  
    application = this,  
    projectId = "some_project_id",  
    testModeEnabled = true  
)
```

In test mode, a test push token is generated and test push notifications will be delivered using the following method only:

```
val testNotificationPayload = TestNotificationPayload(
    title = "Test notification title",
    body = "Test notification message",
    imgUrl = "some_image_http_url",
    data = mapOf("some_key" to "some_value")
)

RuStorePushClient.sendTestNotification(testNotificationPayload).addOnCompleteListener(object : OnCompleteListener<Unit> {
    override fun onFailure(throwable: Throwable) {
        // Process send test push error
    }

    override fun onSuccess(result: Unit) {
        // Process send test push success
    }
})
```

RuStore SDK for revision history

SDK version 1.0.0

- Updated sdk libraries: core and analytics up to v.1.0.0
- Divided pushclient into several modules which should be connected transitively:
 - ru.rustore.sdk:push-common
 - ru.rustore.sdk:push-core
 - ru.rustore.sdk:push-core-network

SDK version 0.7.0

- Fixed "checkPushAvailability" method bugs
- Marked "checkPushAvailability(context: Context)" as deprecated to remove it in the future. Use "checkPushAvailability()" instead without arguments.
- Added analyticsCallback to "RuStorePushClient.init": AnalyticsCallback?. It is required to operate with the future function Targeting mailing.
- Bugfix.

SDK version 0.6.0

- Bugfix.

SDK version 0.5.0

- Bugfix.

SDK version 0.4.0

- Updated push notifications methods. You no longer need to add the ProGuard -keep public class com.vk.push.** extends android.os.Parcelable rule, it can be removed. All the required parameters are now provided with the SDK;
- Modified obfuscation structure. Now obfuscated classes are kept amid the root Push SDK package.

SDK version 0.3.0

- Bugfix;
- Added E2E Testing of Push Notifications SDK.

SDK version 0.2.0

- Bugfix;
- Added methods for working with push topics.

SDK version 0.1.9

- Internal SDK update.

SDK version 0.1.8

- Fixed await() method for Task API.

SDK version 0.1.7

- Internal SDK update.

SDK version 0.1.6

- Internal SDK update.

SDK version 0.1.5

- Internal SDK update.

SDK version 0.1.4

- Intent Redirection vulnerability fixed which prevented from publishing the app on Google Play.

SDK version 0.1.3

- Internal SDK update.

SDK version 0.1.2

- Internal SDK update.

SDK version 0.1.1

- Internal SDK update.
- In case there is no push token, the "getToken" method will create and return a new push token.
- The "deleteToken" method no longer creates a new push token after the current one is removed.

SDK version 0.1.0

- The Suspend methods are replaced with task API as follows:
 - Push notification availability check— «checkPushAvailability».
 - Obtaining a push token — «getToken».
 - Removing a push token — «deleteToken».
- The "checkPushAvailability" method now returns a "FeatureAvailabilityResult" object.

SDK version 0.0.9

- Internal SDK update.

Unreal

Client SDK: push notifications manual (RuStoreSDK) (1.0)

General information

For push notifications to work, the following requirements must be met:

1. RuStore is installed on the user's device.
2. RuStore supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user is authorized in RuStore.
5. The app signature fingerprint must match the fingerprint specified [in RuStore Console](#).
6. Unreal Engine 4.26 or later.

Connecting to project

1. Copy the contents of the “*Plugins*” folder from the official RuStore repository on [gitflit](#) to the “*Plugins*” folder of your project. Restart Unreal Engine, in the plug-in list (Edit → Plugins → Project → Mobile) select “RuStorePush” and “RuStoreCore”.
2. In the “*YourProject.Build.cs*” file of the PublicDependencyModuleNames list connect the “RuStoreCore” and “RuStorePush” modules.
3. In the project settings (Edit → Project Settings → Android) set the Minimum SDK Version parameter to 24 or later and the Target SDK Version parameter to 31 or later.

Editing app manifest

The “RuStorePush” plug-in will declare the RuStoreUnityMessagingService service:

AndroidManifest.xml

```
<service
    android:name="ru.rustore.unitysdk.pushclient.RuStoreUnityMessagingService"
    android:exported="true"
    tools:ignore="ExportedService">
    <intent-filter>
        <action
            android:name="ru.rustore.sdk.pushclient.MESSAGING_EVENT" />
        </intent-filter>
    </service>
```

You can add the following metadata if you want to change the default notification icon or color:

AndroidManifest.xml

```
<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_icon"
    android:resource="@drawable/ic_baseline_android_24" />
</meta-data>
```

```
android:name="ru.rustore.sdk.pushclient.default_notification_color"
    android:resource="@color/your_favorite_color" />
```

You can add the following metadata:

AndroidManifest.xml

```
<meta-data
```

```
android:name="ru.rustore.sdk.pushclient.default_notification_channel_
id"
    android:value="@string/pushes_notification_channel_id" />
```

To add a channel for push notifications, create this channel yourself.

Initialization

Before calling library methods, initialize the library:

Initialization

```
FURuStorePushClientConfig config;
config.allowNativeErrorHandling = true;
config.messagingServiceListener = pushMessagingServiceListener;
config.logListener = pushLogListener;
FURuStorePushClient::Instance()->Init(config);
```

All operations with the client are also accessible from Blueprints. Initialization example:

- allowNativeErrorHandling - allow native error handling (see “Error handling” for details)
- messagingServiceListener - class object that implements IRuStoreMessagingServiceListenerInterface
- logListener - class object that implements IRuStoreLogListenerInterface
- projectId - your project identifier in RuStore Console.

The URuStoreMessagingServiceListener and URuStoreLogListener classes implement IRuStoreMessagingServiceListenerInterface and IRuStoreLogListenerInterface interfaces respectively. It allows you to handle interface events directly from blueprint.

Initialization of URuStoreMessagingServiceListener and URuStoreLogListener together with URuStorePushClient:

IRuStoreMessagingServiceListenerInterface

```
class RUSTOREPUSH_API IRuStoreMessagingServiceListenerInterface
{
    GENERATED_BODY()
```

```
public:
```

```
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Messaging Service Listener Interface")
    void NewTokenResponse(int64 requestId, FString& token);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Messaging Service Listener Interface")
```



```

    void MessageReceivedResponse(int64 requestId,
FURuStoreRemoteMessage& message);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Messaging Service Listener Interface")
    void DeletedMessagesResponse(int64 requestId);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Messaging Service Listener Interface")
    void ErrorResponse(int64 requestId, TArray<FURuStoreError>&
errors);
};

```

- **NewTokenResponseResponse** - will be called on receiving the push token. After calling this method your app is responsible for passing new push token to its server.
- **MessageReceivedResponse** - will be called on getting a new push notification.
- **DeletedMessagesResponse** - will be called if one or several push notifications are not delivered to the device. It may happen, for example, if notification lifetime expires before the notification is delivered. When calling this method, it is recommended to sync with your server to avoid missing data.
- **ErrorResponse** - will be called in case of an error during the initialization.

Possible errors:

- **UnauthorizedException** - the user is not authorized in RuStore.
- **HostAppNotInstalledException** - RuStore is not installed on the user's device.
- **HostAppBackgroundWorkPermissionNotGranted** - RuStore doesn't have permissions to work in the background.

Notification structure.

Full notification structure

```

USTRUCT(BlueprintType)
struct RUSTOREPUSH_API FURuStoreRemoteMessage
{
    GENERATED_USTRUCT_BODY()
public:
    UPROPERTY()
    FString collapseKey;

    UPROPERTY()
    TMap<FString, FString> data;

    UPROPERTY()
    FString messageId;

    UPROPERTY()
    FURuStoreNotification notification;

    UPROPERTY()
    int priority;

```

```

    char* rawData;

    UPROPERTY()
    int ttl;
};

```

- `messageId` - message identifier. Is added to each message.
- `priority` - (**currently, ignored**) returns priority value. For now, the following variants are proposed:
 - 0 - UNKNOWN.
 - 1 - HIGH.
 - 2 - NORMAL.
- `ttl` - time to live of a push notification with the `Int` type in seconds.
- `collapseKey` - (**currently, ignored**) notifications group identifier.
- `data` - dictionary to which additional notification data can be passed.
- `rawData` - byte array data dictionary.
- `notification` - notification object.

Notification object structure

```

USTRUCT(Blueprintable)
struct RUSTOREPUSH_API FURuStoreNotification
{
    GENERATED_USTRUCT_BODY()
public:
    FURuStoreNotification()
    {
        title = "0";
        body = "0";
        channelId = "0";
        imageUrl = "0";
        color = "0";
        icon = "0";
        clickAction = "0";
    }
    UPROPERTY()
    FString title;
    UPROPERTY()
    FString body;
    UPROPERTY()
    FString channelId;
    UPROPERTY()
    FString imageUrl;
    UPROPERTY()
    FString color;
    UPROPERTY()
    FString icon;
    UPROPERTY()

```

```

    FString clickAction;
};

```

- title - notification title.
- body - notification body.
- channelId - allows you to set a channel to which a notification will be sent (relevant for Android 8.0 and later).
- imageUrl - direct URL of an image to be included in the notification (the image size must not exceed 1 MB).
- color - notification color (Notification.color). The color needs to be passed as a HEX string. Example: "#A52A2A".
- icon - notification icon. The icon must be placed in the app resources (res/drawable). Parameter value is a string that matches the resource name. Example: in res/drawable there is a small_icon.xml icon that is available in the code via R.drawable.small_icon. To display this icon in a notification the server must place the "small_icon" value in the "icon" parameter.
- clickAction - intent action that will open activity when a notification is clicked on.

IRuStoreLogListenerInterface

```

class RUSTOREPUSH_API IRuStoreLogListenerInterface
{
    GENERATED_BODY()
public:
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Log Listener Interface")
    void LogResponse(int64 requestId, FString& logString);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Log Listener Interface")
    void LogWarningResponse(int64 requestId, FString& logString);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Log Listener Interface")
    void LogErrorResponse(int64 requestId, FString& logString);
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore Log Listener Interface")
    void LogExceptionResponse(int64 requestId, FURuStoreError&
error);
};

```

- LogResponse - is called whenever regular event log records are created.
- LogWarningResponse - is called whenever warnings are created in the event log.
- LogErrorResponse - is called whenever error messages are created in the event log.
- LogExceptionResponse - is called whenever exception records are created in the event log.

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
};

```

```

    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

Calling `Init()` for `URuStorePushClient`, `URuStoreMessagingServiceListener`, `URuStoreLogListener` ties objects to the scene root. If no further work with the objects is needed, run the `Dispose()` method to free memory. The `Dispose()` method call untie the objects from root and securely complete all sent requests.

Deinitialization

```

URuStorePushClient::Instance()->Dispose();
URuStoreMessagingServiceListener::Instance()->Dispose();
URuStoreLogListener::Instance()->Dispose();

```

Push notification availability check

To check whether push notifications are available, use the `CheckPushAvailability()` method. On calling, the following conditions are checked:

1. RuStore is installed on the user's device.
2. RuStore supports push notifications.
3. The user is authorized in RuStore.
4. The user and the app are not banned in RuStore.
5. The app signature fingerprint must match the fingerprint specified in [RuStore Console](#).

Each `CheckPushAvailability()` request returns `requestId` that is unique per app launch. Each event returns `requestId` of the request that triggered this event.

RuStorePushClient.CheckPushAvailability implementation example

```

long responseId =
URuStorePushClient::Instance()->CheckPushAvailability(
    [](long responseId, TSharedPtr<FUFeatureAvailabilityResult,
ESPMode::ThreadSafe> response) {
        // Process response
    },
    [](long responseId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
        // Process error
    }
);

```

Blueprint implementation:

The Success callback returns the `FURuStoreFeatureAvailabilityResult` structure in the Response parameter:

CheckPushAvailability response

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreFeatureAvailabilityResult
{
    GENERATED_USTRUCT_BODY()
    FURuStoreFeatureAvailabilityResult()
    {
        isAvailable = false;
    }
    UPROPERTY(BlueprintReadWrite)
    bool isAvailable;

    UPROPERTY(BlueprintReadWrite)
    FURuStoreError cause;
};
```

`isAvailable` - whether payment conditions are met.

`cause` - error information.

The Failure callback returns the `FURuStoreError` structure with the error information in the Error parameter. All possible `FURuStoreException` errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};
```

- `name` - error name.
- `description` - error description.

Methods for working with push tokens

Retrieving the user's push token

After the library is initialized you can use `URuStorePushClient::Instance()->GetToken()` to retrieve the user's push token. If the user doesn't have a push token, the method will create one and return it.

`RuStorePushClient.GetToken` implementation example

```

long responseId = URuStorePushClient::Instance()->GetToken(
    [](long responseId, FString response) {
        // Process response
    },
    [](long responseId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);

```

Blueprint implementation:

The Success callback returns the token string in the Response parameter:

GetProducts response

pFString response

- response - current push token.

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the [“Error handling”](#) section.

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

User's push token deletion

After the library is initialized you can use RuStorePushClient::Instance()->DeleteToken() to delete the user's current push token.

RuStorePushClient.DeleteToken implementation example

```

long responseId = URuStorePushClient::Instance()->DeleteToken(
    [](long responseId) {
        // Process success
    },

```

```

        [](long responseId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
            // Process error
        }
    };

```

Blueprint implementation:

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the “[Error handling](#)” section.

Error structure

```

USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

Methods for working with push topics

Topic-based subscription for push notifications

After the library is initialized you can use URuStorePushClient::Instance()->SubscribeToTopic() to subscribe to a topic.

RuStorePushClient.GetToken implementation example

```

long requestId = URuStorePushClient::Instance()->SubscribeToTopic(
    topicName,
    [](long requestId) {
        // Process error
    },
    [](long requestId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
        // Process error
    }
);

```

Blueprint implementation:

The Failure callback returns the `FURuStoreError` structure with the error information in the `Error` parameter. All possible `FURuStoreException` errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};
```

- name - error name.
- description - error description.

Unsubscribing from topic-based push notifications

After the library is initialized you can use `RuStorePushClient::Instance()->UnsubscribeToTopic()` to unsubscribe from a topic.

RuStorePushClient.DeleteToken implementation example

```
long responseId =
URuStorePushClient::Instance()->UnsubscribeFromTopic(
    [(long responseId) {
        // Process success
    }],
    [(long responseId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
        // Process error
    }
]);
```

Blueprint implementation:

The Failure callback returns the `FURuStoreError` structure with the error information in the `Error` parameter. All possible `FURuStoreException` errors are described in the “[Error handling](#)” section.

Error structure

```
USTRUCT(BlueprintType)
struct RUSTORECORE_API FURuStoreError
{
    GENERATED_USTRUCT_BODY()
    FURuStoreError()
    {
```



```

        name = "";
        description = "";
    }
    UPROPERTY(BlueprintReadOnly)
    FString name;
    UPROPERTY(BlueprintReadOnly)
    FString description;
};

```

- name - error name.
- description - error description.

Creation of a notification channel

For a channel to which a notification will be sent, the following priority order is used:

1. If a push notification contains the `channelId` field, the SDK will send the notification to this channel. Note, that your application is responsible for creating this channel in advance.
2. If the `channelId` field is absent in a push notification, but your app specified a parameter with the channel in `AndroidManifest.xml`, then, the channel from `AndroidManifest.xml` is used. Your app is responsible for channel creation.
3. If the `channelId` field is absent in a push notification and the default channel is not set in `AndroidManifest.xml` of your application, then, SDK will create the channel and send the notification to it. Further on, all notifications without set channel will be passed to this channel.

Opening Activity on notification tap

By default, on notification tap the SDK opens an activity with the “`android.intent.action.MAIN`” action. If the `clickAction` field is available, then, the SDK opens the activity that matches Intent filter with the specified action.

For the SDK to open an activity on notification tap (this is also relevant to the default activity), in the app manifest add `<category android:name="android.intent.category.DEFAULT" />` in the relevant `<intent-filter>` element of the activity. Without this string the SDK will be unable to open an activity.

Error handling

Possible errors:

1. `RuStoreNotInstalledException` - RuStore is not installed on the user's device.
2. `RuStoreOutdatedException` - the RuStore app installed on the user's device doesn't support push notifications.
3. `RuStoreUserUnauthorizedException` - the user is not authorized in RuStore.
4. `RuStoreFeatureUnavailableException` - RuStore isn't allowed to run in the background.
5. `RuStoreException` - base RuStore error from which all other errors are inherited.

If `allowNativeErrorHandling == true` was passed during the SDK initialization and an error occurs, aside from calling the `onFailure` handler, this error is passed to the `resolveForBilling` method of the native SDK to show the error dialog to the user:

resolveForPush

```
fun RuStoreException.resolveForPush(context: Context)
```

You can change this behavior after the initialization by setting `AllowNativeErrorHandling` property:

Deny native error handling

```
URuStorePushClient::Instance()->SetAllowNativeErrorHandling(false);
```

Unity

General

Implementation example

See the [example app](#) to learn how to enable push notifications correctly.

Push notifications enabling conditions

For push notifications to be enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.
5. App signature must match the one added to the RuStore Console.

Connecting to the project

To have the SDK enabled, you need to download the RuStore Push SDK and import it into (Assets → Import Package → Custom Package). Dependencies are included automatically using the External Dependency Manager (included in the SDK).

Minimum API level must be set to at least 24. Application minification (ProGuard/R8) is not currently supported; it must be disabled in the project settings (File → Build Settings → Player Settings → Publishing Settings → Minify).

Editing your app's manifest

Declare service extending RuStoreMessagingService:

```
<service
android:name="ru.rustore.unitysdk.pushclient.RuStoreUnityMessagingS
ervice"
    android:exported="true"
    tools:ignore="ExportedService">
    <intent-filter>
```

```
        <action
android:name="ru.rustore.sdk.pushclient.MESSAGING_EVENT" />
        </intent-filter>
</service>
```

You can add the following metadata if you wish to change icon or color of standard notification:

```
<meta-data

android:name="ru.rustore.sdk.pushclient.default_notification_icon"
    android:resource="@drawable/ic_baseline_android_24" />
<meta-data

android:name="ru.rustore.sdk.pushclient.default_notification_color"
    android:resource="@color/your_favorite_color" />
```

You can add the following metadata to redefine notification channel:

```
<meta-data

android:name="ru.rustore.sdk.pushclient.default_notification_channel_id"
    android:value="@string/pushes_notification_channel_id" />
```

When adding your own push notification channel, you have to create the channel yourself.

Initialization

For initialization, you will need a project ID, which can be obtained in the [RuStore Console](#). To do this, on the application page, go to the "Push notifications" section and select "Projects".

```
package ru.rustore.unitysdk;

import android.app.Application;
import ru.rustore.unitysdk.pushclient.RuStoreUnityPushClient;
```

```
public class RuStoreUnityApplication extends Application {

    @Override public void onCreate() {
        super.onCreate();
        RuStoreUnityPushClient.init(
            application = this
        );
    }
}
```

- application — Application class instance.

Specify this class in AndroidManifest.xml:

```
<application
    android:name="ru.rustore.unitysdk.RuStoreUnityApplication">
```

The library initialization parameters are configured in the Unity editor. Select Window → RuStoreSDK → Settings → Push Client in the editor menu:

- VKPNS Project Id — project ID on the RuStore Console;
- Allow Native Error Handling — allow error handling in the native SDK.

You need to initialize the library before calling its methods from C# code:

```
var config = new RuStorePushClientConfig() {
    allowNativeErrorHandling = true,
    messagingServiceListener = pushServiceListener,
    logListener = pushLogListener
};

RuStorePushClient.Instance.Init(config);
```

allowNativeErrorHandling — allow error handling in the native SDK;
messagingServiceListener — object of a class that implements the IMessagingServiceListener interface;
logListener — object of a class that implements the ILogListener interface.

Checking ability to receive push notification

For push notifications to be enabled, all the conditions must be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.

You can use `RuStorePushClient.checkPushAvailability` method to check fulfillment of these conditions:

```
RuStorePushClient.Instance.CheckPushAvailability(  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: (response) => {  
        if (!response.isAvailable) {  
            // Process push unavailable  
        }  
    });
```

Push Token Management Methods

Getting user's push token

After initialization of the library, you can use `RuStorePushClient.getToken()` method to get the user's current push token.

```
RuStorePushClient.Instance.GetToken(  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: (token) => {  
        // Process success  
    });
```

Deleting user's push token

You can use `RuStorePushClient.deleteToken()` method to delete the user's current push token.

```
RuStorePushClient.Instance.DeleteToken(  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: () => {  
        // Process success  
    });
```

Push Topics Management Methods

Getting user's push topic

After initialization of the library, you can use `SubscribeToTopic("your_topic_name")` method to get the user's current push topics.

```
RuStorePushClient.Instance.SubscribeToTopic(  
    topicName: "your_topic_name",  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: () => {  
        // Process success  
    });
```

Unsubscribing from user's push topic

You can use `UnsubscribeFromTopic("your_topic_name")` method to delete the user's current push topic.

```
RuStorePushClient.Instance.UnsubscribeFromTopic(  
    topicName: "your_topic_name",  
    onFailure: (error) => {  
        // Process error  
    },  
    onSuccess: () => {  
        // Process success  
    });
```

RuStoreMessagingService

```
public interface IMessagingServiceListener {  
  
    public void OnNewToken(string token);  
    public void OnMessageReceived(RemoteMessage message);  
    public void OnDeletedMessages();  
    public void OnError(List<RuStoreError> errors);  
}
```

onNewToken — will be called when a new push token is received. After this method is called, your app will be responsible for delivering the new push token to its server.

onMessageReceived — will be called when a new push notification is received. If there is data in 'notification' object, RuStoreSDK will display the notification itself. If you don't want RuStoreSDK to display notification itself, use 'data' object, and leave 'notification' object empty. However, onMessageReceived will be called in any case. Push notification's payload (Map<String, String>) may be obtained from message.data field.

onDeletedMessages — will be called if one or more push notifications have not been delivered to device. This may happen, for example, due to expiry of notification's lifetime before it is delivered to device. When this method is called, synchronizing with your server is recommended to avoid missing any data.

onError — will be called if an error occurs at time of initialization.

Possible errors

- `UnauthorizedException` — the user has not logged in to the RuStore.
- `HostAppNotInstalledException` — user's device doesn't have RuStore app installed.
- `HostAppBackgroundWorkPermissionNotGranted` — RuStore app is not allowed to run in the background.

All the methods will be called in the background.

Notification structure

Full notification structure

```
public class RemoteMessage {  
  
    public string collapseKey;  
    public Dictionary<string, string> data;  
    public string messageId;  
    public Notification notification;  
    public int priority;  
    public sbyte[] rawData;  
    public int ttl;  
}
```

- `messageId` — unique ID of message. It is the identifier of each message;
- `priority` — returns priority value (not taken into account at the moment). Possible values:
 - 0 — UNKNOWN;
 - 1 - HIGH;
 - 2 - NORMAL.
- `ttl` — returns `Int` type push notification's lifetime in seconds;
- `collapseKey` — identifier of a group of notifications (not taken into account at the moment);
- `data` — a dictionary to which additional data for notification can be sent;
- `rawData` — 'data' dictionary in the form of a binary array;
- `notification` — notification object.

Structure of Notification object

```
public class Notification {  
  
    public string title;  
    public string body;  
    public string channelId;  
    public string imageUrl;  
    public string color;  
    public string icon;  
    public string clickAction;  
}
```

- `title` — notification's title;

- body — notification's body;
- channelId — option to create the channel to which notification will be sent (for Android 8.0 or later);
- imageUrl — a direct link to image for insertion to notification (maximum size 1 MB);
- color — notification's color (Notification.color). Color needs to be sent in hex format, as a line (Example: #A52A2A);
- icon — notification's icon. Icon should be located in the app's resources (res/drawable). Parameter's value is a line that matches the resource's name:
 - small_icon.xml icon is located in res/drawable, which is accessible in code via R.drawable.small_icon. For this icon to be displayed in notification, the server should place 'small_icon' value to 'icon' parameter.
- clickAction — 'intent action' with which activity is opened when a notification is pressed on.

Creating channel for sending notification

The following order of priority is used for the channel to which notification will be sent:

1. If there is 'channelId' field in push notification, then RuStoreSDK will send notification to this channel. Note that your app is responsible for creating this channel in advance.
2. If there is no 'channelId' field in push notification, but your app has specified parameter with channel in AndroidManifest.xml, then channel from AndroidManifest.xml will be used. Your app is responsible for creating the channel.
3. If there is no 'channelId' field in push notification, and no default channel has been set in your app's AndroidManifest.xml, then RuStoreSDK will create channel itself and will send notification to it. All further notifications with no channel specified will be sent to this channel.

Opening Activity when notification is pressed on

By default, RuStoreSDK opens activity with 'android.intent.action.MAIN' action whenever a notification is pressed on. If 'clickAction' field is present, RuStoreSDK will open activity which falls under 'Intent filter' with specified 'action'.

For activity to open in RuStoreSDK when a notification is pressed on (this also applies to default activity), add <category android:name="android.intent.category.DEFAULT" /> line in the corresponding activity's <intent-filter> element in app's manifest. Activity will not open in RuStoreSDK without this line.

Error handling

Possible errors:

- `RuStoreNotInstalledException()` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException()` — RuStore installed on a user's device doesn't support push notifications.
- `RuStoreUserUnauthorizedException()` — the user is not logged in to the RuStore.
- `RuStoreFeatureUnavailableException()` — RuStore app is not allowed to run in the background.
- `RuStoreException(message: String)` — RuStore basic error from which all the other errors are inherited.

If the `allowNativeErrorHandling == true` parameter was passed during SDK initialization, when an error occurs, it is passed to the `resolveForPush` method of the native SDK and calls the corresponding `onFailure` handler,:

```
fun RuStoreException.resolveForPush(context: Context)
```

You can change this behavior after initialization by setting the `AllowNativeErrorHandling` property:

```
RuStorePushClient.Instance.AllowNativeErrorHandling = false;
```

Flutter

General	241
Checking ability to receive push notification	243
How to work with push tokens and push notifications	244
Notification structure	246

General

Implementation example

See the [example app](#) to learn how to integrate package for push notifications handling correctly.

Push notifications enabling conditions

For push notifications to be enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.

Enabling in project

To enable package in project, execute the following command:

```
flutter pub add flutter_rustore_push
```

This command will add a line to pubspec.yaml file.

```
dependencies:
```

```
  flutter_rustore_push: ^1.0.0
```

Initialization

To initialize push notifications service, add a value to 'values' of your android project:

```
<resources>
  <string name="flutter_rustore_push_project"
  translatable="false">xxx</string>
</resources>
```

xxx — project's identifier. The field's name in [the RuStore Console](#), in 'Push notifications -> Projects' section, is 'Project ID'.

To start push notification service, Application class inherited from FlutterRustoreApplication needs to be added.

An example of how to do this in Kotlin:

```
package ru.rustore.flutter_rustore_push_example

import ru.rustore.flutter_rustore_push.FlutterRustoreApplication

open class Application: FlutterRustoreApplication() {
}
```

Specify the following class in AndroidManifest.xml:

```
<application
    android:label="flutter_rustore_push_example"
    android:name=".Application"
    android:icon="@mipmap/ic_launcher">
    // ...
</application>
```

Configuring ProGuard

To configure ProGuard, add the following rule:

```
-keep public class com.vk.push.** extends android.os.Parcelable
```

Add the following code in android/app/build.gradle:

```
buildTypes {
    release {
        // ...

        proguardFiles
getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
    }

    // ...
}
```

Checking ability to receive push notification

For push notifications to be enabled, all the conditions must be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.

You can use `RustorePushClient.available()` method to check if these conditions are met:

```
RustorePushClient.available().then((value) {  
    print("available success: ${value}");  
}, onError: (err) {  
    print("available error: ${err}");  
});
```


How to work with push tokens and push notifications

Getting user's push token

After initialization of the library, you can use `RustorePushClient.getToken()` method to get the user's current push token.

If the user has no push token, the method will create and return a new push token.

```
RustorePushClient.getToken().then((value) {
    print("get token success: ${value}");
}, onError: (err) {
    print("get token error: ${err}");
})
```

Deleting user's push token

You can use `RustorePushClient.deleteToken()` method to delete user's current push token.

```
RustorePushClient.deleteToken().then(() {
    print("delete success:");
}, onError: (err) {
    print("delete error: ${err}");
})
```

Token change events

An old token can become invalid from time to time. Token can be issued anew. To see if a new token has been issued, use `RustorePushClient.onNewToken()` callback.

```
RustorePushClient.onNewToken((value) {
    print("on new token success: ${value}");
}, error: (err) {
    print("on new token err: ${err}");
});
```

Working with push notification

To get information from a push notification, add `RustorePushClient.onMessageReceived()` callback.

```
RustorePushClient.onMessageReceived((value) {
```

```
        print("on message received success: id=${value.messageId},
data=${value.data}, notification.body:
${value.notification?.body}");
    }, error: (err) {
        print("on message received error: ${err}");
    });
```

Deleting push notification

To delete push notification, add `RustorePushClient.onDeletedMessages()` callback.

```
RustorePushClient.onDeletedMessages(() {
    print("deleted messages");
}, error: (err) {
    print("on message received error: ${err}");
});
```

Error handling

Use `RustorePushClient.onError()` callback for error handling.

```
RustorePushClient.onError((err) {
    print("on error: ${err}");
});
```

Notification structure

Notification structure

```
class Message {
    String? messageId;
    int priority;
    int ttl;
    String? collapseKey;
    Map<String?, String?> data;
    Notification? notification;
}
```

- `messageId` — unique ID of message. It is the identifier of each message;
- `priority` — returns priority value. The following options are currently available:
 - 0 — UNKNOWN.
 - 1 — HIGH.
 - 2 — NORMAL.
- `ttl` — returns `Int` type push notification's lifetime in seconds;
- `collapseKey` — identifier of a group of notifications;
- `data` — a dictionary to which additional data for notification can be sent;
- `notification` — notification object.

Structure 'Notification'

```
class Notification {
    String? title;
    String? body;
    String? channelId;
    String? imageUrl;
    String? color;
    String? icon;
    String? clickAction;
}
```

- `title` — notification's title;
- `body` — notification's body;
- `channelId` — option to create the channel to which notification will be sent (for Android 8.0 or later);
- `imageUrl` — a direct link to image for insertion to notification (maximum size 1 MB);
- `color` — notification's color (`Notification.color`). Color needs to be sent in hex format, as a line (Example: `#A52A2A`);
- `icon` — notification's icon. Icon should be located in the app's resources (`res/drawable`). Parameter's value is a line that matches the resource's name:

- `small_icon.xml` icon is located in `res/drawable`, which is accessible in code via `R.drawable.small_icon`. For this icon to be displayed in notification, the server should place `'small_icon'` value to `'icon'` parameter.
- `clickAction` — `'intent action'` with which activity is opened when a notification is pressed on.

Sending push notifications

The API was developed to provide a drop-in replacement for the Firebase.

To send a push notification, use the POST method
`https://vkpns.rustore.ru/v1/projects/$project_id/messages:send`.

Fill in "Project ID" and "Service Token" to send a push notification. You can also get these values on RuStore Console. To do this, go to the "Push Notifications" section and select "Projects".

Проекты > prod PROD

ID проекта

wANNRz5N- 

Отпечаток подписи SHA-256

B4:74:02:45:29:E5:89:A2:C2:F6:10-
DF:61

Сервисные токены

Создано: 1/5

[+ Создать](#)



Htdbx-mZfl-hs-
tT7f7i8wA



The service token must be specified in the "Authorization: Bearer {service-token}" header.

Request

Parameter	Type	Description
validate_only	bool	Validate request without sending push notifications
message	object (message)	Push notifications structure

message

Parameter	Type	Description
token	string	Push user token received in the app
data	map	Object which contains «key»: value
notification	object (message.notification)	Basic notification template to be used on all platforms
android	object (message.android)	Special Android parameters for messages

*in hms it is required.

message.notification

Parameter	Type	Description
title	string	Notification title
body	string	Notification body
image	string	Contains URL images that will be displayed in the notification

message.android

Parameter	Type	Description
ttl	string (duration format)	How long (in seconds) the message should be stored. Example: "3.5s"
notification	object (message.android.notification)	Notification to Android devices

message.android.notification

Parameter	Type	Description
-----------	------	-------------

title	string	Notification title
body	string	Notification body
icon	string	Notification icon
color	string	Notification icon color in #rrgggb
image	string	Contains URL images that will be displayed in the notification
channel_id	string	Notification channel ID
click_action	string	Action related to the user's access to the notification

* hms defaults to type 1 (intent).

At this point, message structure only supports the above fields.

Successful response

Parameter	Type	Description
-	-	In case of a successful response, a message with the "OK" status is returned.

Error response

Parameter	Type	Description
error	object (error)	Error

error

Parameter	Type	Description
code	int	Numerical error code (404, 400, 403, 401, ...)
status	string	Detailed error description
errors	array (string)	Error code per provider or validation error

HTTP status corresponds to the code field.

Possible errors when sending a message:

- INVALID_ARGUMENT—incorrect request parameters.
- INTERNAL—internal service error.
- TOO_MANY_REQUESTS—exceeded number of attempts to send a message.
- PERMISSION_DENIED—incorrect service key.
- NOT_FOUND — incorrect user's push token.

Message validation algorithm

1. If there is a non-empty payload message.data (at least one pair of the key-value inside), then the message is valid. The sections message.notification and message.android may be empty.
2. If there are no message.data fields, then there must be a notification. In this case, the presence of either the field message.notification or message.android.notification is checked. At least one of these fields should be present, but also they both may be present (if both are present, then some fields will be rewritten).

Restrictions

1. If there is no ttl field or it is equal to 0, then the default value will be included equal to 4 weeks. If there is no message.android section, then it will be added with the ttl field.
2. The maximum message size is 4096 bytes.

Push notification examples

Successful request example

POST

https://vkpns.rustore.ru/v1/projects/myproject-b5ae1/messages:s
end HTTP/2

Content-Type: application/json

Authorization: Bearer \$ss_token

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "body":"This is a notification message!",
      "title":"Message",
      "image":"https://image-hosting.org/284239234.jpeg"
    }
  }
}
```

Response

HTTP/2 200

```
{}
```

Invalid provider example

```
POST
https://vkpns.rustore.ru/v1/projects/U95076bdd5KDJ3LjYkNp91o05Y
6LkfQk/messages:send HTTP/2
Content-Type: application/json
Authorization: Bearer
Fw9FgDx9FQtya6k-7UkS0nzhPHYhDq0SQY4-8QKJ6wKZI90UPiCCYyNmS-CV2-ZQ
5

{
  "message": {
    "token": "bad-push-token",
    "notification": {
      "body": "This is a notification message!",
      "title": "Message",
      "image": "https://image-hosting.org/284239234.jpeg"
    }
  }
}
```

Response

```
HTTP/2 400

{
  "error": {
    "code": 400,
    "message": "The registration token is not a valid FCM
registration token",
    "status": "INVALID_ARGUMENT"
  }
}
```

Invalid message example

```
POST
https://vkpns.rustore.ru/v1/projects/U95076bdd5KDJ3LjYkNp91o05Y
6LkfQk/messages:send HTTP/2
Content-Type: application/json
Authorization: Bearer
Fw9FgDx9FQtya6k-7UkS0nzpHYhDq0SQY4-8QKJ6wKZI90UPiCCYyNmS-CV2-ZQ
5

{
  "message": {
    "token": "eH4tgqKEFFKqH6cMJ2WLttVibgQ09hfn",
    "notification": {
      "body": "This is a notification message!",
      "title": "Message",
      "image": "https://image-hosting.org/284239234.jpeg"
    }
  }
}
```

Response

```
HTTP/2 404

{
  "error": {
    "code": 404,
```

```
    "message": "Requested entity was not found.",

    "status": "NOT_FOUND"

}

}
```

Sending push notifications by topic

To work with topics you will need the following artifacts:

- project_id (project ID), ss_token (service token) — These values can be obtained in the RuStore Console. To do this, go to the “Push Notifications” section, open the “Projects” tab on the app page.
- push_token — Device Push token(s) for subscribing to / unsubscribing from mailing list by topic.
- topic — Project Topic.

Sending Push Notification to topic

Header is required. Authorization: Authorization: Bearer \$ss_token.

Request

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>data</i>	<i>map</i>	Object containing «key»: value

notification	object (message.notification)	Basic notification template to be used on all platforms
android	object (message.android)	Android specific messaging options

Successful response

Parameter	Type	Description
message	text	Message about successful push sending

Error

Parameter	Type	Description
code	int	Error code
status	text	Status
message	text	Details

Subscribing to push notifications by topic

Header is required Authorization: Authorization: Bearer \$ss_token.

Request

Parameter	Type	Description
------------------	-------------	--------------------

<i>push_tokens</i>	<i>array (string)</i>	<i>Push tokens that need to be subscribed to a topic</i>
---------------------------	------------------------------	-----------------------------------------------------------------

Successful response

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>message</i>	<i>text</i>	<i>Message about successful subscription or errors</i>
<i>errors</i>	<i>object array (error response by token)</i>	<i>Error by token</i>

Error

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>push_token</i>	<i>text</i>	<i>Push token</i>
<i>error</i>	<i>object (error response)</i>	<i>Error</i>

Error response

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>code</i>	<i>int</i>	<i>Error code</i>
<i>status</i>	<i>text</i>	<i>Status</i>
<i>message</i>	<i>text</i>	<i>Details</i>

Unsubscribing from push notifications by topic

Header is required Authorization: Authorization: Bearer \$ss_token.

Request

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>push_tokens</i>	<i>string array</i>	<i>Push tokens that need to be unsubscribed from topic</i>

Successful response

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>message</i>	<i>text</i>	<i>Message about successful subscription or errors</i>
<i>errors</i>	<i>object array (error response by token)</i>	<i>Error by token</i>

Error by token

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>push_token</i>	<i>text</i>	<i>Push token</i>
<i>error</i>	<i>object (error response)</i>	<i>Error</i>

Error response

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
<i>code</i>	<i>int</i>	<i>Error code</i>
<i>status</i>	<i>text</i>	<i>Status</i>
<i>message</i>	<i>text</i>	<i>Details</i>

Java

General

Implementation example

See the [example app](#) to learn how to enable push notifications correctly.

Push notifications enabling conditions

For push notifications to be enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.
5. App signature must match the one added to the RuStore Console.

How to add a repository

Enable local repository:

```
repositories {  
    maven {  
        url =  
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")  
    }  
}
```

Dependency injection

To enable dependency, add the following code to your configuration file:

```
dependencies {  
    implementation("ru.rustore.sdk:pushclient:2.0.0")  
}
```

Editing your app's manifest

Declare service extending RuStoreMessagingService:

```
<service  
    android:name=".MyRuStoreMessagingService"
```

```

        android:exported="true"
        tools:ignore="ExportedService">
        <intent-filter>
            <action
android:name="ru.rustore.sdk.pushclient.MESSAGING_EVENT" />
            </intent-filter>
</service>

```

You can add the following metadata if you wish to change icon or color of standard notification:

```

<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_icon"
    android:resource="@drawable/ic_baseline_android_24" />
<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_color"
    android:resource="@color/your_favorite_color" />

```

You can add the following metadata to redefine notification channel:

```

<meta-data
    android:name="ru.rustore.sdk.pushclient.default_notification_channel_id"
    android:value="@string/pushes_notification_channel_id" />

```

When adding your own push notification channel, you have to create the channel yourself.

Initialization

For initialization, you will need a project ID, which can be obtained in the [RuStore Console](#). To do this, on the application page, go to the "Push notifications" section and select "Projects".

Проекты > prod PROD

ID проекта

wANNRz5N ██████████

Отпечаток подписи SHA-256

B4:74:02:45:29:E5:89:A2:C2:F6:1E
DF:61 ██████████

Сервисные токены

Создано: 1/5

[+ Создать](#)



Htbdx-mZfl-hs ██████████
tT7f7i8wA



```
class App : Application() {  
  
    @Override  
    public void onCreate() {  
        super.onCreate()  
        RuStorePushClient.INSTANCE.init(  
            this,  
            "i5UTx96jw6c1C9Lvd1E4cdNrWHMnyRBt",  
            DefaultLogger()  
        );  
    }  
}
```

Add to your project's 'Application' the following code for initialization:

- application — instance of 'Application' class;
- projectId — your project's identifier in VKPNS system;
- logger — logger, output to logcat is used by default.

Methods for working with push topics

Getting user's push topic

After initialization of the library, you can use `RuStorePushClient.subscribeToTopic("your_topic_name")` method to get the user's current push topics.

```
RuStorePushClient.INSTANCE.subscribeToTopic("your_topic_name").addOnCompleteListener(new OnCompleteListener<Void>() {
    @Override
    public void onComplete(Task<Void> task) {
        if (task.isSuccessful()) {
            // Process subscribe success
        } else {
            Exception exception = task.getException();
            // Process subscribe error
        }
    }
});
```

Unsubscribing from user's push topic

You can use `RuStorePushClient.unsubscribeToTopic("your_topic_name")` method to delete the user's current push topic.

```
RuStorePushClient.INSTANCE.unsubscribeFromTopic("your_topic_name").addOnCompleteListener(new OnCompleteListener<Void>() {
    @Override
    public void onComplete(Task<Void> task) {
        if (task.isSuccessful()) {
            // Process unsubscribe success
        } else {
            Exception exception = task.getException();
            // Process unsubscribe error
        }
    }
});
```

RuStoreMessagingService

```
public class MessagingService extends RuStoreMessagingService {  
  
    @Override  
    public void onNewToken(String token) {  
  
    }  
  
    @Override  
    public void onMessageReceived(RemoteMessage message) {  
  
    }  
  
    @Override  
    public void onDeletedMessages() {  
  
    }  
  
    @Override  
    public void onError(List<RuStorePushClientException> errors) {  
  
    }  
}
```

onNewToken — will be called when a new push token is received. After this method is called, your app will be responsible for delivering the new push token to its server.

onMessageReceived — will be called when a new push notification is received. If there is data in 'notification' object, RuStoreSDK will display the notification itself. If you don't want RuStoreSDK to display notification itself, use 'data' object, and leave 'notification' object empty. However, onMessageReceived will be called in any case. Push notification's payload (Map<String, String>) may be obtained from message.data field.

onDeletedMessages — will be called if one or more push notifications have not been delivered to device. This may happen, for example, due to expiry of notification's lifetime before it is delivered to device. When this method is called, synchronizing with your server is recommended to avoid missing any data.

onError — will be called if an error occurs at time of initialization.

Possible errors

- `UnauthorizedException` — the user has not logged in to the RuStore.
- `HostAppNotInstalledException` — user's device doesn't have RuStore app installed.
- `HostAppBackgroundWorkPermissionNotGranted` — RuStore app is not allowed to run in the background.

All the methods will be called in the background.

Notification structure

Full notification structure

```
public RemoteMessage(  
    String messageId,  
    int priority,  
    int ttl,  
    String collapseKey,  
    Map<String, String> data,  
    byte[] rawData,  
    Notification notification  
) {  
    this.messageId = messageId;  
    this.priority = priority;  
    this.ttl = ttl;  
    this.collapseKey = collapseKey;  
    this.data = data;  
    this.rawData = rawData;  
    this.notification = notification;  
}
```

- `messageId` — unique ID of message. It is the identifier of each message;
- `priority` — returns priority value (not taken into account at the moment). Possible values:
 - 0 — UNKNOWN;
 - 1 - HIGH;
 - 2 - NORMAL.
- `ttl` — returns `Int` type push notification's lifetime in seconds;
- `collapseKey` — identifier of a group of notifications (not taken into account at the moment);
- `data` — a dictionary to which additional data for notification can be sent;
- `rawData` — 'data' dictionary in the form of a binary array;
- `notification` — notification object.

Structure of Notification object

```
public Notification(  
    String title,  
    String body,
```

```

    String channelId,
    Uri imageUrl,
    String color,
    String icon,
    String clickAction
) {
    this.title = title;
    this.body = body;
    this.channelId = channelId;
    this.imageUrl = imageUrl;
    this.color = color;
    this.icon = icon;
    this.clickAction = clickAction;
}

```

- title — notification's title;
- body — notification's body;
- channelId — option to create the channel to which notification will be sent (for Android 8.0 or later);
- imageUrl — a direct link to image for insertion to notification (maximum size 1 MB);
- color — notification's color (Notification.color). Color needs to be sent in hex format, as a line (Example: #A52A2A);
- icon — notification's icon. Icon should be located in the app's resources (res/drawable). Parameter's value is a line that matches the resource's name:
 - small_icon.xml icon is located in res/drawable, which is accessible in code via R.drawable.small_icon. For this icon to be displayed in notification, the server should place 'small_icon' value to 'icon' parameter.
- clickAction — 'intent action' with which activity is opened when a notification is pressed on.

Creating channel for sending notification

The following order of priority is used for the channel to which notification will be sent:

1. If there is 'channelId' field in push notification, then RuStoreSDK will send notification to this channel. Note that your app is responsible for creating this channel in advance.
2. If there is no 'channelId' field in push notification, but your app has specified parameter with channel in AndroidManifest.xml, then channel from AndroidManifest.xml will be used. Your app is responsible for creating the channel.
3. If there is no 'channelId' field in push notification, and no default channel has been set in your app's AndroidManifest.xml, then RuStoreSDK will create

channel itself and will send notification to it. All further notifications with no channel specified will be sent to this channel.

Opening Activity when notification is pressed on

By default, RuStoreSDK opens activity with 'android.intent.action.MAIN' action whenever a notification is pressed on. If 'clickAction' field is present, RuStoreSDK will open activity which falls under 'Intent filter' with specified 'action'.

For activity to open in RuStoreSDK when a notification is pressed on (this also applies to default activity), add `<category android:name="android.intent.category.DEFAULT" />` line in the corresponding activity's `<intent-filter>` element in app's manifest. Activity will not open in RuStoreSDK without this line.

Event logging

If you wish to log events of push notification library, add 'logger' parameter to RuStorePushClient.init call (this parameter is not required for initialization).

To do this, 'Logger' interface needs to be implemented:

```
public interface Logger {  
  
    void verbose(String message, Throwable throwable);  
    void debug(String message, Throwable throwable);  
    void info(String message, Throwable throwable);  
    void warn(String message, Throwable throwable);  
    void error(String message, Throwable throwable);  
  
    Logger createLogger(String tag);  
}
```

If Logger has not been sent, then default implementation with AndroidLog will be used.

```
public class PushLogger implements Logger {  
  
    private final String tag;  
  
    public PushLogger(String tag) {  
        this.tag = tag;  
    }  
  
    @Override  
    public void debug(@NonNull String message, Throwable throwable) {  
        Log.d(tag, message, throwable);  
    }  
  
    @Override  
    public void error(@NonNull String message, Throwable throwable) {  
        Log.e(tag, message, throwable);  
    }  
  
    @Override  
    public void info(@NonNull String message, @Nullable Throwable  
throwable) {  
        Log.i(tag, message, throwable);  
    }  
}
```

```
    @Override
    public void verbose(@NonNull String message, @Nullable Throwable
throwable) {
        Log.v(tag, message, throwable);
    }

    @Override
    public void warn(@NonNull String message, @Nullable Throwable
throwable) {
        Log.w(tag, message, throwable);
    }

    @NonNull
    @Override
    public Logger createLogger(@NonNull String newTag) {
        String combinedTag = (tag != null) ? tag + ":" + newTag :
newTag;
        return new PushLogger(combinedTag);
    }
}
```

Error handling

Possible errors:

- `RuStoreNotInstalledException()` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException()` — RuStore installed on a user's device doesn't support push notifications.
- `RuStoreUserUnauthorizedException()` — the user is not logged in to the RuStore.
- `RuStoreFeatureUnavailableException()` — RuStore app is not allowed to run in the background.
- `RuStoreException(message: String)` — RuStore basic error from which all the other errors are inherited.

If you wish to use UI interface for error handling, then use `resolveForPush()` method:

```
public class RuStoreExceptionExtension {
    public static void resolveForPush(RuStoreException exception,
    Context context) {
        // Your implementation here to resolve the exception for
    push
    }
}
```

E2E Testing of Push Notifications SDK

To have testing enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.

Enable the test mode to start testing the SDK:

```
RuStorePushClient.INSTANCE.init(
    this,
    "some_project_id",
    true
)
```

In test mode, a test push token is generated and test push notifications will be delivered using the following method only:

```
TestNotificationPayload testNotificationPayload = new
TestNotificationPayload(
    "Test notification title",
    "Test notification message",
    "some_image_http_url",
    data
);

RuStorePushClient.INSTANCE.sendTestNotification(testNotificationP
ayload)
    .addOnCompleteListener(new OnCompleteListener<Void>() {
        @Override
        public void onComplete(Task<Void> task) {
            if (task.isSuccessful()) {
                // Process send test push success
            } else {
                Exception exception = task.getException();
                // Process send test push error
            }
        }
    });
```

Defold

General

Implementation example

See the [example app](#) to learn how to enable push notifications correctly.

Push notifications enabling conditions

For push notifications to be enabled, the following conditions need to be met:

1. The RuStore app is installed on the user's device.
2. The RuStore app supports push notifications.
3. The RuStore app is allowed to run in the background.
4. The user has logged in to the RuStore.
5. App signature must match the one added to the RuStore Console.

Connecting to the project

To connect a package to your project, you need to add it as a dependency:

<https://gitflic.ru/project/rustore/defold-extension-rustore-push/file/downloadAll?branch=master>

We recommend that you point to a specific release

<https://gitflic.ru/project/rustore/defold-extension-rustore-push/release/>

Initialization

To initialize the Push Notification SDK, you need to add a value to the `game_project` in your project. Open it with any convenient text editor and add the parameter to the `[android]` section

```
[android]
rustore_project_id = %your project id%
package = %your package%
```

`%your project id%` is the project ID,

`%your package%` is the Android package.

In the RuStore Console, the field is called “Project ID” and “Android Package Name” on the “Push Notifications” -> “Projects” page.

Default settings for Push Message Header and Body

You can add default values for the push header and body if they are not in the message data

```
[android]
push_field_title = default push title
push_field_text = default push body
```

The push notification service will be launched automatically on Android devices when `ruStorePush.set_on_token()` is called

Push token and message methods

Getting a user's push token

To receive push tokens, you need to add a *callback* to the listener `ruStorePush.set_on_token()`

If the user does not have a push token, the method will return a new push token.

```
local function new_token(self, token, error)
    if token then
        print(token)
    else
        print(error.error)
    end
end

local function push_android()
    ruStorePush.set_on_token(new_token)
end
```

Token replacement

From time to time, the old token may become invalid. At that, it can be issued again. These events come to the callback `ruStorePush.set_on_token()`

Working with a push message

To receive information from a push notification, you need to add a callback `ruStorePush.set_on_message()`

```
local function listener(self, payload, activated)
    -- The payload arrives here.
    pprint(payload)
end

local function push_android()
    ruStorePush.set_on_message(listener)
end
```


Listener structure

- payload → data field for push notification (lua table)
- activated → whether the user open an app by clicking on the push (bool)

Full example

```
local function listener(self, payload, activated)
    -- The payload arrives here.
    pprint(payload)
end

local function new_token(self, token, error)
    if token then
        print(token)
    else
        print(error.error)
    end
end

local function push_android()
    ruStorePush.set_on_token(new_token)
    ruStorePush.set_on_message(listener)

    print("Rustore pushes registered")
end

function init(self)
    local sysinfo = sys.get_sys_info()
    if sysinfo.system_name == "Android" then
        push_android()
    else
        print("Notifications work only Android")
    end

    msg.post(".", "acquire_input_focus")
end
```

Note!

Do not send push messages with notification and android.notification.title. They will be processed by RuStore and don't operate properly, you can't use them to go to the app.

Send data push with the following fields

- **title** — for push title;
- **message** — for push body.

Testing push notification integration

NOTE

This section covers information related to the following programming languages:

- Kotlin;
- Java.

For testing to be enabled, the following conditions need to be met:

- The RuStore app must be installed on the user's device.
- Your RuStore app should support the payment processing function.
- The RuStore app is allowed to run in the background.
- Your app user must be authorized on the RuStore.

To start testing the SDK, you need to enable the testing mode:

- Kotlin
 - Java
-

Enabling test mode example

```
RuStorePushClient.init(  
    application = this ,
```

```
    projectId = "your_push_project_id" ,
    testModeEnabled = true
)
```

During testing mode, only test push notifications sent through the specified method will be delivered, using a test push token generated for this purpose.

- Kotlin
- Java

Example of sending a test push notification

```
val testNotificationPayload = TestNotificationPayload(
    title = "Test notification title",
    body = "Test notification message",
    imgUrl = "some_image_http_url",
    data = mapOf("some_key" to "some_value")
)
```

```
RuStorePushClient.sendTestNotification(testNotificationPa
ayload).addOnCompleteListener(object :
OnCompleteListener<Unit> {
    override fun onFailure(throwable: Throwable) {
        // Process send test push error
    }

    override fun onSuccess(result: Unit) {
        // Process send test push success
    }
})
```

User-group focused Push Notifications API

Often, you need to send messages to specific groups of users. You can do this easily with RuStore and MyTracker. Just follow the instructions in the documentation to

integrate SDK Push and MyTracker to your app. Then, you can send messages to different groups of users quickly and easily.

A segment is a group of users chosen based on specific criteria you set. For instance, it could include users who generate the most revenue or those using older versions of Android. In MyTracker, a segment is a powerful tool for analyzing your product's audience. It lets you:

- Evaluate the size and changes within user groups.
- Target specific groups for advertising.
- Share device and user identifiers from the segment with external analytics systems or partners.
- Create detailed reports for one or more segments without cluttering your statistics with irrelevant data.

General

To manage segments effectively, you'll require the following items:

- `project_id` (project ID), `ss_token` (service token). These values can be obtained in the RuStore Console. To do this, go to the Push Notifications > Projects section on the application page (see Sending push notifications).
- `push_token`.
- Push tokens needed to subscribe or unsubscribe devices from newsletters on specific topics.
- `mt_api_user_id`. API User ID from myTracker.
- `mt_secret_key`. API secret key from myTracker.
- `mt_segment_id`. Segment ID from myTracker.
- `mt_app_id`. Application ID.
- `export_project_id(uuid)`. Export project ID.
- `export_segment_id(uuid)`. Segment ID to export.

Creating an Export Project

POST https://vkpns-segments.rustore.ru/v1/export_settings/project/create

Request body

Parameter	Type	Description
-----------	------	-------------

<code>project</code>	text	Project ID push from RuStore Console
<code>secret_key</code>	text	API MyTracker secret key
<code>api_user_id</code>	text	MyTracker user ID required to download segments
<code>app_id</code>	int	MyTracker App ID

Тело успешного ответа

Parameter	Type	Description
<code>export_project_id</code>	text	Export Project ID

Тело ответа ошибки

Parameter	Type	Description
<code>code</code>	int	Error code
<code>message</code>	text	Error message
<code>status</code>	text	Status

Getting Export Project Settings

POST https://vkpns-segments.rustore.ru/v1/export_settings/project/get

Header required a Authorization: Authorization: Bearer \$ss_token.

Request body

Parameter	Type	Description
<code>id</code>	text	Export Project ID

Successful return

Parameter	Type	Description
<code>app_id</code>	int	MyTracker App ID
<code>id</code>	text	Export Project ID
<code>project</code>	text	Project ID push from RuStore Console

Error response

Parameter	Type	Description
code	int	Error code
status	text	Error message
message	text	Status

Creating a segment for export

POST

https://vkpns-segments.rustore.ru/v1/export_settings/project/<export_project_id>/segments/create

Header required Authorization: Authorization: Bearer \$ss_token.

Request body

Parameter	Type	Description
period	int	Unload frequency in hours
segment	text	Segment ID from MyTracker

Successful return

Parameter	Type	Description
export_segment_id	text	Segment ID to upload

Error response

Parameter	Type	Description
code	int	Error code
status	text	Error message
message	text	Status

Getting segment data for export

POST

https://vkpns-segments.rustore.ru/v1/export_settings/project/<export_project_id>/segments/get

Header required Authorization: Authorization: Bearer \$ss_token.

Successful return

Parameter	Type	Description
segments	array<Segment>	Aegments array

Segment

Parameter	Type	Description
id	text	Segment ID to upload
segment	text	Segmeter ID from MyTracker
period	int	Upload frequency in hours
is_enabled	boolean	Indicator showing whether this segment is enabled or disabled.

Error response

Parameter	Type	Description
code	int	Error code
status	text	Error message
message	text	Status

Sending push notifications to a segment

POST

https://vkpns-segments.rustore.ru/v1/projects/<project_id>/segments/<mt_segment_id>/publish

Header required Authorization: Authorization: Bearer \$ss_token.

Request body

Parameter	Type	Description
-----------	------	-------------

message object (message) Message

Successful response

Parameter	Type	Description
message	text	Successful push message

Error response

Parameter	Type	Description
code	int	Error code
status	text	Error message
message	text	Status

Examples

Creating a project with export settings

```
curl --location
'https://vkpns-segments.rustore.ru/v1/export_settings/project/create' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer <ss_token>' \
--data '{
  "project": "<project_id>",
  "api_user_id": "<mt_api_user_id>",
  "secret_key": "<mt_secret_key>",
  "app_id": <mt_app_id>
}';
```

Successful return

HTTP/2 200

```
{
  "id": "b04b48ab-3125-444f-94eb-aad511c074e7"
};
```

Invalid s2s token

HTTP/2 400

```
{
  "code": 2000,
  "status": "BAD_REQUEST",
  "message": "Invalid S2S token"
};
```

Getting export project settings

```
curl --location
'https://vkpns-segments.rustore.ru/v1/export_settings/project/get' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer <ss_token>' \
--data '{
  "id": "<export_project_id>"
}';
```

Successful return

HTTP/2 200

```
{
  "id": "<export_project_id>",
  "app_id": <mt_app_id>,
  "project": "<project_id>"
};
```

Creating an export segment

```
curl --location
'https://vkpns-segments.rustore.ru/v1/export_settings/project/<export_project_id>/segments/create' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer <ss_token>' \
--data '{
```

```
"period": 24,  
"segment": "<mt_segment_id>"  
}';
```

Successful return

HTTP/2 200

```
{  
  "id": "<export_segment_id>"  
};
```

Getting export segment data

```
curl --location --request POST  
'https://vkpns-segments.rustore.ru/v1/export_settings/project/<e  
xport_project_id>/segments/get' \  
--header 'Accept: application/json' \  
--header 'Authorization: Bearer <ss_token>';
```

Successful return

HTTP/2 200

```
{  
  "segments": [  
    {  
      "id": "<export_segment_id>",  
      "segment": "<mt_segment_id>",  
      "period": 24,  
      "is_enabled": true  
    }  
  ]  
};
```

Sending messages to chosen segments

```
curl --location
'https://vkpns-segments.rustore.ru/v1/projects/<project_id>/segments/<mt_segment_id>/publish' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer <ss_token>' \
--data '{
  "message": {
    "notification": {
      "body": "This is a notification message!",
      "title": "Message",
      "image": "https://image-hosting.org/284239234.jpeg"
    }
  }
};
```

Successful return

HTTP/2 200

```
{
  "message": "payload has been successfully published to segment
<mt_segment_id>"
};
```

Managing segments in MyTracker

Link MyTracker to your app

To send push notifications based on segments, you must integrate both the MyTracker SDK and Push Notification SDK into your app.

Establish segments

RuStore's push notifications enable targeting different user segments. To activate this functionality, integration with MyTracker is necessary.

- Install and configure MyTracker SDK within your app.
- Enable segment collection through the MyTracker interface.











A segment is a group of users chosen based on specific criteria you set. For instance, it could include users who generate the most revenue or those using older versions of Android. In MyTracker, a segment is a powerful tool for analyzing your product's audience. It lets you:

- Evaluate the size and changes within user groups.
- Target specific groups for advertising.
- Share device and user identifiers from the segment with external analytics systems or partners.
- Create detailed reports for one or more segments without cluttering your statistics with irrelevant data.

See the MyTracker documentation to get more details about segments. MyTracker figures out segment sizes based on what you choose. It updates these numbers every day, so it's easy to see how things change on the segment pages over time. The List of Segments shows important details like sizes, when they were last calculated, and which apps they're linked to.

List of segments

Список сегментов

Сегмент	Аккаунт	Размер сегмента	Дата создания	Дата расчёта			
Заходят ежедневно на час ^d Автор: Иван Синичкин	myTrackerAccount	37.86% 3 695 733 из 9 760	13 Мар 2023	14 Мар 2023			
Достижение: 10 уровень ^u Автор: Иван Воробьев	myTrackerAccount myProject Приложения:  My application	28.64% 51 567 из 180 078	13 Мар 2023	14 Мар 2023			
Мужчины 18-45 ^d Автор: Андрей Орлов	Main account	0.25% 271 из 110 174	13 Мар 2023	14 Мар 2023			

To add a new segment, go to the Segment List page and click **Add**.

You can use an existing segment as a starting point. Simply open the segment page and click **Duplicate**.

Next, fill out the form with the following details:

- Name: The name you want to give to the segment, which will appear in MyTracker lists and reports.
- Audience type: Specify whether the segment will be built based on physical devices or user accounts.
- Account*: Choose the account to which the segment will be added. If you have only one account, it will be selected automatically. Refer to the Account section for more details.
- Projects: Select one or more projects whose applications will be used to form the segment. Check the Project section for further information.
- Applications: Choose one or more applications whose audience will be used to create the segment.

Connecting MyTracker API

To use segments, you must connect to the MyTracker API. Obtain a token from the user's profile page to proceed.

[ГЛАВНАЯ](#) · [НАСТРОЙКИ ПРОФИЛЯ](#)

Настройки профиля

Личные данные

[Уведомления](#)

[Отчёты](#)

[Безопасность](#)

Имя *

Artem

Фамилия *

Kovardin

Email

artem-kovardin@yandex.ru

Номер телефона

+ 79818463973

Учётные данные Export API

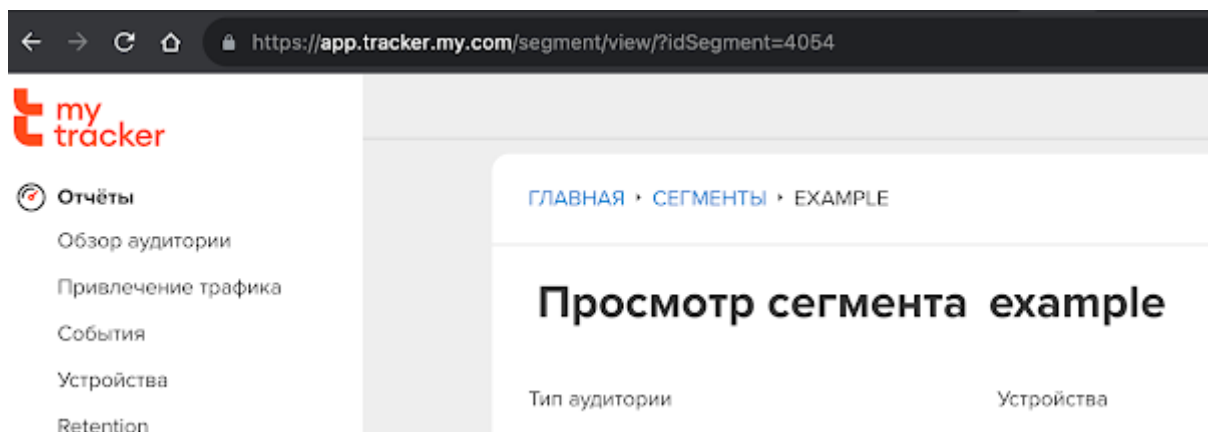
ПОКАЗАТЬ

СОХРАНИТЬ

ОБНОВИТЬ СЕКРЕТНЫЙ КЛЮЧ

Click **Show** to view the token. Copy the token and paste it into the "secret_key" field when configuring settings in the push notification API.

Once the project is created, you can begin adding new segments for export. To do this, you'll need the segment ID, which you can find in the URL after navigating to the segment page.



Copy the idSegment from the url and use it in the request:

```
curl --location
'https://segments-vkpns.rustore.ru/v1/export_settings/project/<export_project_id>/segments/create' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--header 'Authorization: Bearer <ss_token>' \
--data '{
  "period": 24,
  "segment": "<mt_segment_id>"
}';
```

export_project_id: This is the project ID obtained during the initial setup.

mt_segment_id: This is the segment ID acquired from MyTracker.

ss_token: This is the service token used for authorization, obtained from the RuStore Console on the push notifications project page.

RuStore General Push Notifications SDK

Kotlin

General	256
Google Play	260
Huawei Mobile Services	262
Initialization	265
Push notifications functional check	267
Tokens	268
Topics	270
Handling notifications	271
If HMS/FCM are already in use	274

General

RuStore General Push Notifications SDK is a set of packages which are required to operate with push notifications. You can send and receive messages via multiple channels:

- FCM — Firebase Cloud Messaging;
- HMS — Huawei Mobile Services;
- RuStore.

Use SDK together with pre-configured HMS and FCM.

Implementation example

Check out the [example app](#) to learn how to integrate universal push notifications properly.

Conditions for correct operation of SDK

For general push notifications SDK to operate correctly, the following conditions need to be met:

- The RuStore app must be installed on the user's device.
- The RuStore app must support push notifications.
- The RuStore app is allowed access to work in the background.
- The user has logged in to the RuStore.
- The app signature must match the one added to the Developer Console.

Setting up the app

To initialize your app, you will need a project ID, which can be obtained from the RuStore Console. On the app page, go to Push Notifications and then select Projects.

ID проекта

wANNRz5N- 

Отпечаток подписи SHA-256

B4:74:02:45:29:E5:89:A2:F6:10-
DF:61

Сервисные токены

Создано: 1/5

[+ Создать](#)



Htdbx-mZfl-hs-
tT7f7i8wA



Importing SDK to your project

Connect the repository in settings.gradle:

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven {url = uri("https://developer.huawei.com/repo/")}
        maven {
            url =
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
        }
    }
}
```

Dependency injection

Add the following code to your build.gradle to inject the dependency at the app level:

```
dependencies {
```

```

implementation 'ru.rustore.sdk:universalpush:1.0.0'
implementation 'ru.rustore.sdk:universalrustore:1.0.0'
implementation 'ru.rustore.sdk:universalhms:1.0.0'
implementation 'ru.rustore.sdk:universalfcm:1.0.0'
}

```

Add the following rule when using the SDK in your app along with ProGuard:

```
-keep public class com.vk.push.** extends android.os.Parcelable
```

Follow the steps below to work with FCM and HMS:

- add to build.gradle at the app level

```

plugins {
    // ...

    // required for fcm
    id 'com.google.gms.google-services'
    // required for hms
    id 'com.huawei.agconnect'
}

```

- add to build.gradle at the root level

```

dependencies {
    // for fcm
    classpath 'com.google.gms:google-services:4.3.15'
    // for hms
    classpath 'com.huawei.agconnect:agcp:1.6.0.300'
    classpath 'com.android.tools.build:gradle:7.4.0'
}

```

- add to settings.gradle at the root level

```

pluginManagement {
    repositories {
        google()
        mavenCentral()
        gradlePluginPortal()
        // required for hms
        maven {url = uri("https://developer.huawei.com/repo/")}
    }
}

```

}

Google Play

Setting up the app

To publish an app on Google Play, you only need to add dependencies which are required to send push notifications through FCM and RuStore. To get started with FCM, set up a project in [Firebase](#).

1. Create a new project in the Firebase Console.
2. In the Firebase Console, select the project you plan to enable push notifications for.
3. In the left menu, next to the project name, click the gear icon and go to Project Settings.
4. Go to Your apps and download google-services.json.
5. Once downloaded, move google-services.json to app/google-services.json.

Importing to your project

To publish your app on Google Play Store, you are required to use `universalcm`, `universalrustore`, and `universalpush` packages only.

Connect the repository in `settings.gradle` at the root level:

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        // required for rustore
        maven {
            url =
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
        }
    }
}
```

Dependency injection

Add the following code to your `build.gradle` to inject the dependency at the app level:

```
dependencies {
    implementation('ru.rustore.sdk:universalpush:0.1.1')
    implementation('ru.rustore.sdk:universalrustore:0.1.0')
    implementation('ru.rustore.sdk:universalcm:0.1.0')
}
```

Add the following rule when using the SDK in your app along with ProGuard:

```
-keep public class com.vk.push.** extends android.os.Parcelable
```

Follow the steps below to work with FCM:

- add to build.gradle at the app level

```
plugins {  
    // ...  
  
    // required for fcm  
    id 'com.google.gms.google-services'  
}
```

- add to build.gradle at the root level

```
dependencies {  
    // required for fcm  
    classpath 'com.google.gms:google-services:4.3.15'  
}
```

Setting up the app

To publish an app on AppGallery, you only need to add dependencies which are required to send push notifications through HMS and RuStore. To get started with HMS, set up a project in developer.huawei.com.

1. Create a new project in developer.huawei.com.
2. In AppGallery Connect, select the project you plan to enable push notifications for.
3. Go to Project Settings - Main.
4. Then go to App data and download agconnect-services.json.
5. Once downloaded, move agconnect-services.json to app/agconnect-services.json.

Importing to your project

To publish your app on AppGallery, you are required to use universalcm, universalrustore, and universalpush packages only.

Connect the repository in settings.gradle:

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
        maven {url = uri("https://developer.huawei.com/repo/")}
        maven {
            url =
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")
        }
    }
}
```

Dependency injection

Add the following code to your build.gradle to inject the dependency at the app level:

```
dependencies {
    implementation('ru.rustore.sdk:universalpush:0.1.1')
    implementation('ru.rustore.sdk:universalrustore:0.1.0')
    implementation('ru.rustore.sdk:universalhms:0.1.0')
}
```

Add the following rule when using the SDK in your app along with ProGuard:

```
-keep public class com.vk.push.** extends android.os.Parcelable
```

Follow the steps below to work with HMS:

- add to build.gradle at the app level

```
plugins {  
    // ...  
  
    // required for hms  
    id 'com.huawei.agconnect'  
}
```

- add to build.gradle at the root level

```
dependencies {  
    // for hms  
    classpath 'com.huawei.agconnect:agcp:1.6.0.300'  
    classpath 'com.android.tools.build:gradle:7.4.0'  
}
```

- add to settings.gradle at the root level

```
pluginManagement {  
    repositories {  
        google()  
        mavenCentral()  
        gradlePluginPortal()  
        // required for hms  
        maven {url = uri("https://developer.huawei.com/repo/")}  
    }  
}
```

Initialization

To initialize your app, you need to add the following code to App.kt:

```
import android.app.Application
import ru.rustore.sdk.universalpush.RuStoreUniversalPushClient
import
ru.rustore.sdk.universalpush.firebase.provides.FirebasePushProvider
import ru.rustore.sdk.universalpush.hms.providers.HmsPushProvider
import ru.rustore.sdk.universalpush.rustore.logger.DefaultLogger
import
ru.rustore.sdk.universalpush.rustore.providers.RuStorePushProvider

class App: Application() {

    private val tag = "UniversalPushExampleApp"

    override fun onCreate() {
        super.onCreate()

        RuStoreUniversalPushClient.init(
            context = this,
            rustore = RuStorePushProvider(
                application = this,
                projectId = "m3Id6aPeXq36mp...",
                logger = DefaultLogger(tag = tag),
            ),
            firebase = FirebasePushProvider(
                context = this,
            ),
            hms = HmsPushProvider(
                context = this,
                appid = "108003365",
            ),
        )
    }
}
```

You need to initialize those push notification providers that are currently in use.

```
RuStoreUniversalPushClient.init(
    context = this,
    rustore = RuStorePushProvider(
        application = this,
```



```
        projectId = "m3Id6aPeXq36mp...",
        logger = DefaultLogger(tag = tag),
    ),
    hms = HmsPushProvider(
        context = this,
        appid = "108003365",
    ),
)
```

Push notifications functional check

To check whether push notification providers are available, call the `checkAvailability(context)` method.

```
RuStoreUniversalPushClient.checkAvailability(this)
    .addOnCompleteListener(object :
OnCompleteListener<Map<String, Boolean>> {
    override fun onSuccess(result: Map<String,
Boolean>) {
        Log.w(tag, "get availability success
${result}")
    }

    override fun onFailure(throwable: Throwable) {
        Log.e(tag, "get tokens err: ${throwable}")
    }
})
```

result — dictionary with keys:

```
public const val UNIVERSAL_FCM_PROVIDER: String = "firebase"
public const val UNIVERSAL_HMS_PROVIDER: String = "hms"
public const val UNIVERSAL_RUSTORE_PROVIDER: String = "rustore"
```

To check the availability of a specific push notification provider:

```
if (result[UNIVERSAL_HMS_PROVIDER] ?: false) {
    // hms provider is available
}
```

Tokens

Receiving tokens

Call the `getTokens()` method to get a list of tokens for all providers.

```
RuStoreUniversalPushClient.getTokens()
    .addOnCompleteListener(object :
OnCompleteListener<Map<String, Boolean>> {
    override fun onSuccess(result: Map<String,
Boolean>) {
        Log.w(tag, "get availability success
${result}")
    }

    override fun onFailure(throwable: Throwable) {
        Log.e(tag, "get tokens err: ${throwable}")
    }
})
```

`result` — dictionary with keys:

```
public const val UNIVERSAL_FCM_PROVIDER: String = "firebase"
public const val UNIVERSAL_HMS_PROVIDER: String = "hms"
public const val UNIVERSAL_RUSTORE_PROVIDER: String = "rustore"
```

To get a specific token, use the code:

```
result[UNIVERSAL_FCM_PROVIDER].orEmpty()
```

Deleting tokens

To delete tokens, you need to call the `deleteTokens(token)` method and pass a dictionary with a list of tokens.

```
RuStoreUniversalPushClient.deleteTokens(
    mapOf(
        UNIVERSAL_RUSTORE_PROVIDER to "xxx",
        UNIVERSAL_FCM_PROVIDER to "yyy",
        UNIVERSAL_HMS_PROVIDER to "zzz"
```

```
)  
)
```

where xxx, yyy, zzz — tokens from different push notification providers.

Topics

Subscribing to a topic

To subscribe to a topic, call the `subscribeToTopic("topic")` method.

```
RuStoreUniversalPushClient.subscribeToTopic("some_topic")
```

Unsubscribing from a topic

To unsubscribe from a topic, call the `unsubscribeFromTopic("topic")` method.

```
RuStoreUniversalPushClient.unsubscribeFromTopic("some_topic")
```

Handling events

Handling onDeletedMessages Events

To manage the onDeletedMessages event, you should integrate an OnDeletedMessagesListener callback in the App class after initializing RuStoreUniversalPushClient. When the onDeletedMessages event is triggered, the callback will be invoked, providing you with the providerType parameter. This parameter enables you to determine which push provider triggered the event.

```
RuStoreUniversalPushClient.setOnDeletedMessagesListener { providerType ->
    // process event
}
```

Handling onNewToken Events

To manage the onNewToken event, it is necessary to include an OnNewTokenListener callback within the App class after initializing RuStoreUniversalPushClient. When the onNewToken event is triggered, a callback function will be invoked, and it will provide the following parameters:

- providerType — determines which push provider triggered the event.
- token — new push token.

```
RuStoreUniversalPushClient.setOnNewTokenListener { providerType, token ->
    // process event
}
```

Handling notifications

To receive notifications, it's essential to incorporate an OnMessageReceiveListener callback within the App class once you have initialized RuStoreUniversalPushClient. If notifications are delivered via the universal API, they will undergo deduplication on the client side, and the notification reception callback will be triggered just once.

In cases where the notification object contains data, RuStoreSDK will automatically handle the notification display. However, if you prefer RuStoreSDK not to handle the notification display, you can utilize the data object while keeping the notification object empty. Nevertheless, the OnMessageReceiveListener callback will still be invoked. You can retrieve the push notification payload (Map<String, String>) from the remoteMessage.data field.

```
RuStoreUniversalPushClient.setOnMessageReceiveListener { remoteMessage ->
    // process message
}
```

Handling push provider errors

To manage errors, it's essential to include an `OnPushClientErrorListener` callback in the `App` class following the initialization of `RuStoreUniversalPushClient`. This callback will be invoked with specific parameters when errors occur:

- `providerType` — determines which push provider triggered the event..
- `errors` — list of errors.

```
RuStoreUniversalPushClient.setOnPushClientErrorListener { providerType, error ->
    // process error
}
```

If HMS/FCM are already in use

If you use FCM/HMS services in your applications, add additional code to the services.

FCM

Add the following code to the service for FCM:

```
import
ru.rustore.sdk.universalpush.firebase.messaging.toNotificationPayload
ad

class MyFirebaseMessagingService: FirebaseMessagingService() {
    override fun onMessageReceived(message: RemoteMessage) {
        super.onMessageReceived(message)

        RuStoreUniversalPushManager.processMessage(message.toNotificationPayload())
    }
}
```

```

    }

    override fun onNewToken(token: String) {
        super.onNewToken(token)

        RuStoreUniversalPushManager.processToken(token)
    }
}

```

HMS

Add the following code to the service for HMS:

```

import
ru.rustore.sdk.universalpush.hms.messaging.toNotificationPayload

class MyMessagePushService: HmsMessageService() {
    override fun onMessageReceived(msg: RemoteMessage?) {
        super.onMessageReceived(msg)

        RuStoreUniversalPushManager.processMessage(msg.toNotificationPayload())
    }

    override fun onNewToken(token: String?) {
        super.onNewToken(token)

        RuStoreUniversalPushManager.processToken(token)
    }
}

```

SDK Release Notes

SDK version 1.0.0

- Added deduplication of push notifications
- Added callbacks for:
 - Handling notifications
 - Handling onDeletedMessages events
 - Handling onNewToken events

- Handling push provider errors

App Feedback and Rating SDK

Android (Kotlin/Java)

General	277
Importing SDK to your project	279
Creating RuStoreReviewManager	280
Getting ReviewInfo object	281
Starting app rating	282
Possible errors	283
RuStore Change History Feedback and Rating SDK	286

General

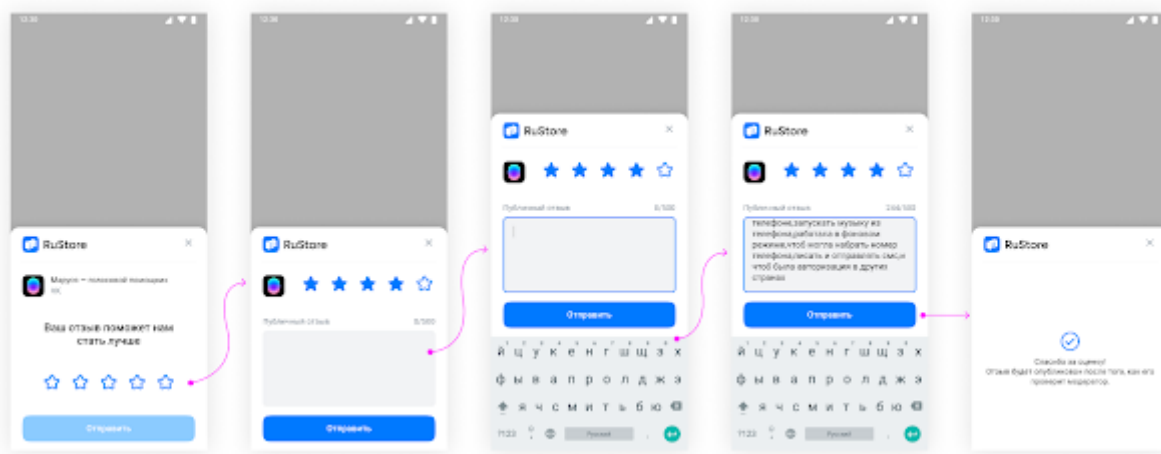
RuStore In-app Review SDK prompts the user to rate your app and leave feedback on the RuStore without exiting the app.

Rating and feedback user scenarios may be run at any time throughout the user's path in your app. The user can rate your app from 1 to 5 and leave feedback. Feedback is optional.

Implementation example

See the [example app](#) to learn how to integrate rating and feedback SDK correctly.

Use case example



Conditions for correct operation of SDK

For rating and feedback SDK to operate correctly, the following conditions need to be met:

1. Android 6.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The user has logged in to the RuStore.
5. The app should be already installed on RuStore.

When to ask to rate and leave feedback

Use the tips below to decide when to ask the user to rate and leave feedback:

- Start the process once the user has been using your app for long enough.
- Avoid starting it too often as this will impair your app's user experience and limit the use or SDK ratings.
- Avoid using calls to action like "Rate App" button as the user could have already reached the process starting limit.

- Your app should not ask the user any questions before the start or while the process is running, including their opinion (“Do you like the app?”) or predictive questions (“Would you give this app 5 stars?”).

Design tips

Use the tips below to decide how to integrate the process:

- Display the process as is, without any intervention or modification of existing design, including size, opacity, shape and other properties.
- Add nothing on top or on sides of the process.
- The process should open on top of all layers. Don't close the process after starting. The process will close by itself after an express action by the user.

Importing SDK to your project

Connect the repository:

```
repositories {  
    maven {  
        url =  
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")  
    }  
}
```

Dependency injection

Add the following code to your configuration file to inject the dependency:

```
dependencies {  
    implementation("ru.rustore.sdk:pushclient:2.0.0")  
}
```

Creating RuStoreReviewManager

To manage the rating process, you need to create RuStoreReviewManager using RuStoreReviewManagerFactory:

```
val manager = RuStoreReviewManagerFactory.create(context)
```

Getting ReviewInfo object

Call `requestReviewFlow()` in advance before calling `launchReviewFlow(reviewInfo)` to prepare necessary information to display.

`ReviewInfo` has a lifetime — about five minutes.

```
manager.requestReviewFlow().addOnCompleteListener(object :  
OnCompleteListener<ReviewInfo> {  
    override fun onFailure(throwable: Throwable) {  
        // Handle error  
    }  
  
    override fun onSuccess(result: ReviewInfo) {  
        // Save reviewInfo  
    }  
})
```

If `onSuccess` response is received, save `ReviewInfo` locally to call `launchReviewFlow(reviewInfo)` later.

If `onFailure` is received, displaying the error to the user is not recommended because the user didn't start this process.

Starting app rating

To start the app rating and feedback form on the user's side, call `launchReviewFlow(reviewInfo)` method using `ReviewInfo` received earlier.

```
manager.launchReviewFlow(reviewInfo).addOnCompleteListener(object :
OnCompleteListener<Unit> {
    override fun onFailure(throwable: Throwable) {
        // Review flow has finished, continue your app flow.
    }

    override fun onSuccess(result: Unit) {
        // Review flow has finished, continue your app flow.
    }
})
```

Wait for notification on form completion by the user in `onSuccess` or `onFailure` to continue operation of the app.

Displaying any additional forms related with rating and feedback after completion of the rating form is not recommended, whatever the result is (onSuccess or onFailure).

Frequently calling `launchReviewFlow` will not result in the rating form being displayed to the user because permitted display is controlled by the `RuStore`.

Possible errors

Possible errors you can get in `onFailure`:

- `RuStoreNotInstalledException()` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException()` — RuStore installed on a user's device doesn't support the start of the rating and feedback process.
- `RuStoreUserUnauthorizedException()` — the user is not logged in to RuStore.
- `RuStoreUserBannedException()` — the user is blocked in RuStore.
- `RuStoreApplicationBannedException()` — the app is blocked in RuStore.
- `RuStoreRequestLimitReached()` — too little time elapsed since the last display of the process.
- `RuStoreReviewExists()` — this user has already rated your app.
- `RuStoreInvalidReviewInfo()` — problems with `ReviewInfo`.
- `RuStoreException(message: String)` — RuStore basic error from which all the other errors are inherited.

RuStore Change History Feedback and Rating SDK

SDK Version 1.0.1

- Internal SDK update

SDK Version 1.0.0

- Internal SDK update

SDK Version 0.2.0

- Class packages have been brought to a unified form ru.rustore.**

SDK Version 0.1.6

- Modified the await() method for the Task API.

SDK Version 0.1.5

- Fixed the Intent Redirection vulnerability that prevented the app from being published on Google Play.

SDK Version 0.1.4

- This is an internal update of SDK.

SDK Version 0.1.3

- Method launchReviewFlow(activity, reviewInfo) marked as deprecated. Use launchReviewFlow(reviewInfo).

Unreal

RuStore In-app Review SDK (1.0) manual

General information

RuStore In-app Review SDK enables users to rate your app without closing it.

The rate prompt can be started at any time the user is in your app. The user can evaluate your app within a scale from 1 to 5 and leave a comment (comments are optional).

User scenario example

Figure 1. Rate and comment scenario example.

Conditions for proper SDK performance

For In-app Review SDK to work properly, the following requirements must be met:

1. OS Android 7.0 or later.
2. RuStore is installed on the user's device.
3. The RuStoreApp version on the device is up-to-date.
4. The user is authorized in RuStore.
5. The app is published in RuStore.

When to ask users to rate and comment?

Follow these recommendations to decide when to prompt the users to rate and comment your app:

- Start the flow when the user has experienced your app for quite enough time.
- Do not start the flow too often—this will negatively affect the user's experience and limit the SDK rating usage.
- Do not call the user for action, for example, do not use the "Rate this app" button, as the user might have already exceeded the flow start limit.
- Your app shouldn't ask the user any questions before the flow starts or during the flow including questions about their opinion (example: "Do you like the app?") or forecast questions (example: "Would you give this app 5 stars?").

Design recommendations

Follow these recommendations to make a decision on how to implement your flow:

- Display your flow as is, without altering the design—including size, opacity, form, and other properties.
- Do not add anything at the front or at the borders of the flow.
- The flow should open on top of all layers. Do not close the flow after it starts. The flow will stop by itself after overt user action.

Connecting to project

1. Copy the contents of the “*Plugins*” folder from the official RuStore repository on [gitflic](#) to the “*Plugins*” folder of your project. Restart Unreal Engine, in the plug-in list (Edit → Plugins → Project → Mobile) select “RuStoreReview” and “RuStoreCore”.

2. In the “*YourProject.Build.cs*” file of the `PublicDependencyModuleNames` list connect the “`RuStoreCore`” and “`RuStoreReview`” modules.
3. In th project settings (Edit → Project Settings → Android) set the Minimum SDK Version parameter to 24 or later, Target SDK Version – to 31 or later.

RuStoreReviewManager initialization

To work with rates and reviews, initialize `RuStoreReviewManager`:

Initialization

```
URuStoreReviewManager::Instance()->Init();
```

Blueprints implementation example:

Important:

1. The `Init()` call ties the object to the scene root, and, If no further work with the object is needed, execute the `Dispose()` method to free memory.

The `Dispose` method call will untie the object from root and securely complete all sent requests.

Initialization

```
URuStoreReviewManager::Instance()->Dispose();
```

If you need to check whether the library is initialized or not, use the `getIsInitialized()`, its value is true if the library is initialized and false if `Init` hasn't been called yet.

Preparing the app for launch

Call `RequestReviewFlow()` in advance, before calling [LaunchReviewFlow\(\)](#), to prepare the necessary information to display.

RequestReviewFlow

```
long requestId =
URuStoreReviewManager::Instance()->RequestReviewFlow(
    [](long requestId) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```

Blueprint implementation:

If the Success callback is received, then, in approximately 5 minuses you can start the rate and review flow—`LaunchReviewFlow()`.

The Failure callback returns the `FURuStoreError` structure with the error information in the `Error` parameter. All possible `FURuStoreException` errors are described in the “[Error handling](#)” section.

Launching rating flow

Call the `LaunchReviewFlow()` method to launch the user's rating and review flow (Figure 1.).

Each request returns `requestId` that is unique per app launch. Each event returns `requestId` of the request that triggered this event.

LaunchReviewFlow

```
long requestId = URuStoreReviewManager::Instance()->LaunchReviewFlow(  
    [](long requestId) {  
        // Process response  
    },  
    [](long requestId, TSharedPtr<FURuStoreError,  
    ESPMode::ThreadSafe> error) {  
        // Process error  
    }  
);
```

Blueprint implementation:

To continue the app flow, wait for the notification after the user stops the flow in `onSuccess` or `onFailure`.

After the rate flow is finished, we do not recommend you to display any additional flows related to rating or review—regardless of the result (`onSuccess` or `onFailure`).

Frequent `LaunchReviewFlow` calls will not start rate flow for the user, as the display limit is configured on `RuStore` side.

Error handling

Errors that occur are passed to the `onFailure` handler of the SDK methods.

Error structure:

Error structure

```
USTRUCT(BlueprintType)  
struct RUSTORECORE_API FURuStoreError  
{  
    GENERATED_USTRUCT_BODY()  
    FURuStoreError()  
    {  
        name = "";  
        description = "";  
    }  
    UPROPERTY(BlueprintReadOnly)  
    FString name;  
    UPROPERTY(BlueprintReadOnly)  
    FString description;  
};
```

- `name` - name of the error.
- `description` - error description.

Possible errors you can receive in `onFailure`:

- `RuStoreNotInstalledException` — RuStore is not installed on the user's device.
- `RuStoreOutdatedException` — the RuStore app installed on the user's device does not support the rate and review flow.
- `RuStoreUserUnauthorizedException` — the user is not authorized in RuStore.
- `RuStoreRequestLimitReached` - too little time has passed since the latest flow display.
- `RuStoreReviewExists` — this user already rated your app.
- `RuStoreInvalidReviewInfo` — something is wrong with `ReviewInfo`.
- `RuStoreException` — base RuStore error from which all other errors are inherited.

Unity

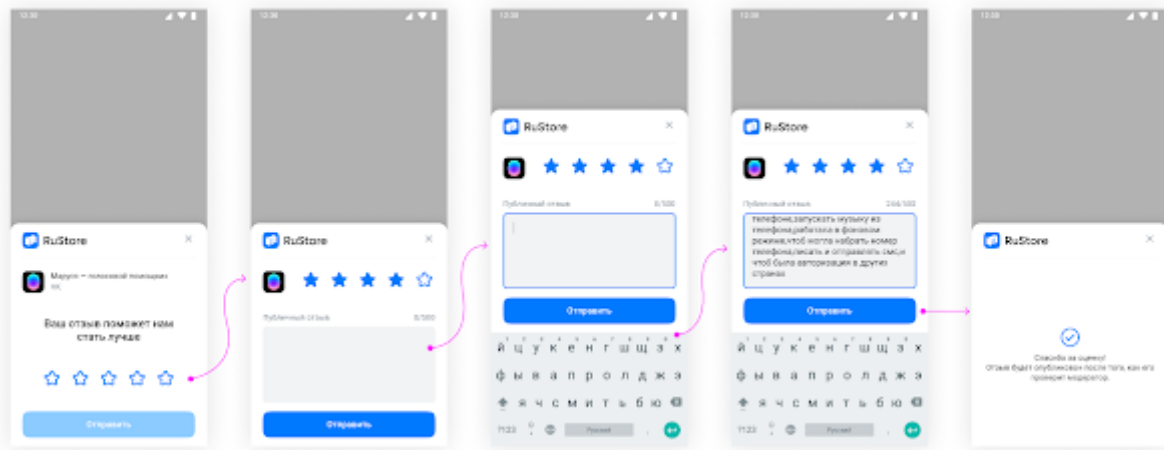
General	286
Importing SDK to your project	288
Creating <code>RuStoreReviewManager</code>	289
App release preparation	290
Starting app rating	291
Handling errors	292

General

RuStore In-app Review SDK prompts the user to rate your app and leave feedback on the RuStore without exiting the app.

Rating and feedback user scenario may be run at any time throughout the user's path in your app. The user can rate your app from 1 to 5 and leave feedback. Feedback is optional.

Use case example



Conditions for correct operation of SDK

For rating and feedback SDK to operate correctly, the following conditions need to be met:

1. Android 7.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The user has logged in to the RuStore.
5. The app should be already published on RuStore.

When to ask to rate and leave feedback

Use the tips below to decide when to ask the user to rate and leave feedback:

- Start the process once the user has been using your app for long enough.
- Avoid starting it too often as this will impair your app's user experience and limit the use or SDK ratings.
- Avoid using calls to action like "Rate App" button as the user could have already reached the process starting limit.
- Your app should not ask the user any questions before the start or while the process is running, including their opinion ("Do you like the app?") or predictive questions ("Would you give this app 5 stars?").

Design tips

Use the tips below to decide how to integrate the process:

- Display the process as is, without any intervention or modification of existing design, including size, opacity, shape and other properties.
- Add nothing on top or on sides of the process.
- The process should open on top of all layers. Don't close the process after starting. The process will close by itself after an express action by the user.

Importing SDK to your project

To get started, you need to download the [RuStore Review SDK](#) and import it to your project (Assets → Import Package → Custom Package). Dependencies are added automatically using the External Dependency Manager (included in SDK).

Minimum API level must be set to at least 24. Application minification (ProGuard/R8) is currently not supported, it must be disabled in the project settings (File → Build Settings → Player Settings → Publishing Settings → Minify).

Creating RuStoreReviewManager

To manage the rating process, you need to create RuStoreReviewManager using RuStoreReviewManagerFactory:

```
RuStoreReviewManager.Instance.Init();
```

App release preparation

Call `RequestReviewFlow()` in advance before calling `LaunchReviewFlow()` to prepare the essential information to display.

```
RuStoreReviewManager.Instance.RequestReviewFlow(  
    onFailure: (error) => {  
        // Handle error  
    },  
    onSuccess: () => {  
        // Handle success  
    });
```

If `onSuccess` is received, you can start requesting app feedback and rating by calling `LaunchReviewFlow()` within about five minutes.

If `onFailure` is received, we do not recommend displaying the user's error, since the user did not start this process.

Starting app rating

To start the app rating and feedback form on the user's side, call `launchReviewFlow(reviewInfo)` method.

```
RuStoreReviewManager.Instance.LaunchReviewFlow(  
    onFailure: (error) => {  
        // Handle error  
    },  
    onSuccess: () => {  
        // Handle success  
    });
```

Wait for notification on form completion by the user in `onSuccess` or `onFailure` to continue operation of the app.

Displaying any additional forms related with rating and feedback after completion of the rating form is not recommended, whatever the result is (onSuccess or onFailure).

Frequently calling `launchReviewFlow` will not result in the rating form being displayed to the user because permitted display is controlled by the `RuStore`.

Handling errors

The errors are processed by `onFailure` using SDK methods.

Error structure:

```
public class RuStoreError {  
  
    public string name;  
    public string description;  
}
```

- name — error name;
- description — error description.

Possible errors you can get in `onFailure`:

- `RuStoreNotInstalledException` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException` — RuStore installed on a user's device doesn't support the start of the rating and feedback process.
- `RuStoreUserUnauthorizedException` — the user is not logged in to RuStore.
- `RuStoreRequestLimitReached` — too little time elapsed since the last display of the process.
- `RuStoreReviewExists` — this user has already rated your app.
- `RuStoreInvalidReviewInfo` — problems with `ReviewInfo`.
- `RuStoreException` — RuStore basic error from which all the other errors are inherited.

Flutter

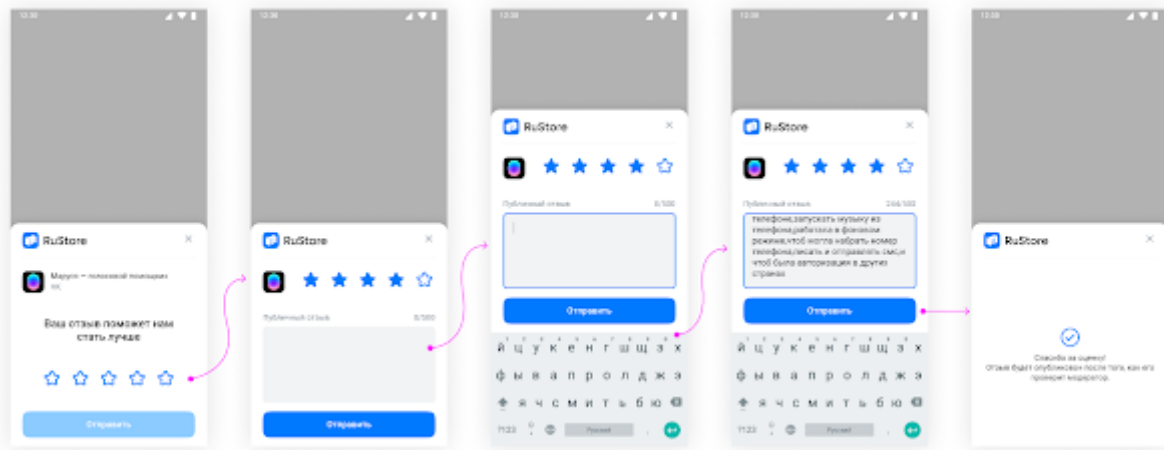
General	294
Importing SDK to your Project	296
Feedback request	297

General

RuStore In-app Review SDK prompts the user to rate your app and leave feedback on the RuStore without exiting the app.

Rating and feedback user cases may be run at any time throughout the user's path in your app. The user can rate your app from 1 to 5 and leave feedback. Feedback is optional.

Use case example



Conditions for correct operation of SDK

For rating and feedback SDK to operate correctly, the following conditions need to be met:

1. Android 7.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The user has logged in to the RuStore.
5. The app should be published on RuStore.

When to ask to rate and leave feedback

Use the tips below to decide when to ask the user to rate and leave feedback:

- Start the process once the user has been using your app for long enough.
- Avoid starting it too often as this will impair your app's user experience and limit the use or SDK ratings.
- Avoid using calls to action like "Rate App" button as the user could have already reached the process starting limit.
- Your app should not ask the user any questions before the start or while the process is running, including their opinion ("Do you like the app?") or predictive questions ("Would you give this app 5 stars?").

Design tips

Use the tips below to decide how to integrate the process:

- Display the process as is, without any intervention or modification of existing design, including size, opacity, shape and other properties.
- Add nothing on top or on sides of the process.
- The process should open on top of all layers. Don't close the process after starting. The process will close by itself after an express action by the user.

Importing SDK to your Project

Run the following command to import the package to your project:

```
flutter pub add flutter_rustore_review
```

Dependency injection

This command adds a line to pubspec.yaml.

```
dependencies:  
  flutter_rustore_review: ^0.0.3
```

Feedback request

To display a feedback and rating window, it is required to perform the plugin initialization:

```
RustoreReviewClient.initialize();
```

Once initialization is completed, you can make a request and display a form.

```
RustoreReviewClient.request().then((value) {  
  RustoreReviewClient.review().then((value) {  
    print("success review");  
  }, onError: (err) {  
    print("on err ${err}");  
  });  
});
```

Gogot

General

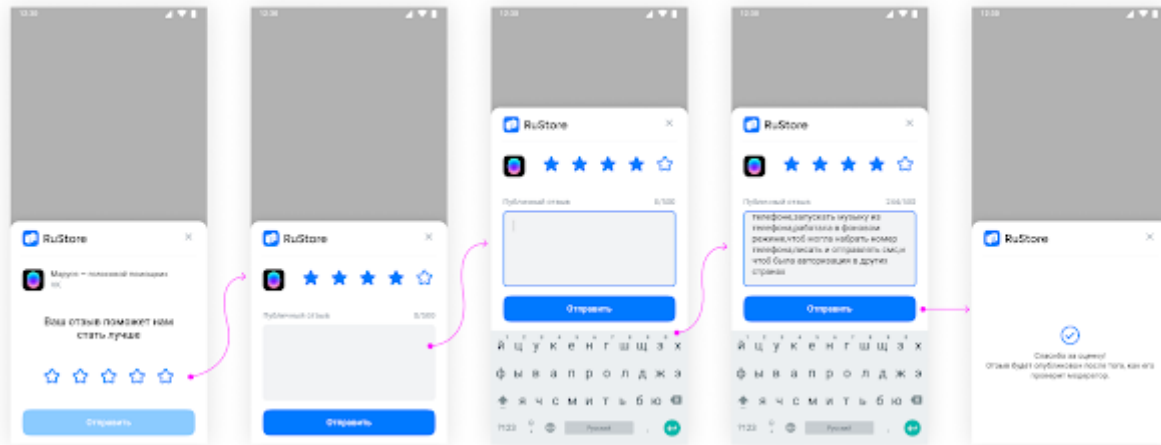
RuStore In-app Review SDK prompts the user to rate your app and leave feedback on the RuStore without exiting the app.

Rating and feedback user scenarios may be run at any time throughout the user's path in your app. The user can rate your app from 1 to 5 and leave feedback. Feedback is optional.

Implementation example

See the [example app](#) to learn how to integrate rating and feedback SDK correctly.

Use case example



Conditions for correct operation of SDK

For rating and feedback SDK to operate correctly, the following conditions need to be met:

6. Android 6.0 or later.
7. The RuStore app is installed on the user's device.
8. The current RuStoreApp version is installed on the user's device.
9. The user has logged in to the RuStore.
10. The app should be already installed on RuStore.

When to ask to rate and leave feedback

Use the tips below to decide when to ask the user to rate and leave feedback:

- Start the process once the user has been using your app for long enough.
- Avoid starting it too often as this will impair your app's user experience and limit the use or SDK ratings.
- Avoid using calls to action like "Rate App" button as the user could have already reached the process starting limit.
- Your app should not ask the user any questions before the start or while the process is running, including their opinion ("Do you like the app?") or predictive questions ("Would you give this app 5 stars?").

Design tips

Use the tips below to decide how to integrate the process:

- Display the process as is, without any intervention or modification of existing design, including size, opacity, shape and other properties.
- Add nothing on top or on sides of the process.
- The process should open on top of all layers. Don't close the process after starting. The process will close by itself after an express action by the user.

Importing SDK to your project

1. Copy the plugin projects from the official RuStore repository on GitFlic.
2. Open the Android project from the `godot_plugin_libraries` folder in your IDE.
3. Place the package `godot-lib.xxx.yyy.template_release.aar`, where `xxx.yyy` is the version of your Godot Engine edition, into the `godot_plugin_libraries / libs` folder.
4. Build the project with the `gradle assemble` command.

If the assembly is successful, files will be created in the `godot_example / android / plugins` folder:

RuStoreGodotReview.gdap

RuStoreGodotReview.aar

RuStoreGodotCore.gdap

RuStoreGodotCore.aar

WARNING

Please note that plugin libraries must be built for your version of Godot Engine.

5. Copy the contents of the `godot_example/android/plugins` folder to your `_project/android/plugins` folder.
6. In the Plugins list tick the RuStore Godot Review and RuStore Godot Core plugins in your Android build preset

Feedback management

Before you start

To work with feedback SDK, you need to create an instance of `RuStoreGodotReviewManager`.

Initialisation

```
Unset
var _review_client: RuStoreGodotReviewManager = null

func _ready:
    _review_client = RuStoreGodotReviewManager.get_instance()
```

Getting started

Call `request_review_flow` in advance of calling `launch_review_flow` to prepare the necessary information for the screen display. The lifetime of `ReviewInfo` is about five minutes.

You must subscribe to events once before using the method:

Unset

```
on_request_review_review_flow_success;  
on_request_review_flow_failure.  
Subscription to events  
func _ready():  
    # Initialisation of _review_client  
  
    _review_client.on_request_review_flow_success.connect(_on_request_review_flow_s  
uccess)  
  
    _review_client.on_request_review_flow_failure.connect(_on_request_review_flow_  
failure)  
  
func _on_request_review_review_flow_success():  
    pass  
  
func _on_request_review_review_flow_failure(error: RuStoreError):  
    pass
```

Unset

```
Calling the request_review_flow method  
_review_client.request_review_flow()
```

The callback `on_request_review_review_flow_failure` returns a `RuStoreError` object with error information.

Launching SDK

To launch Feedback SDK for your app, call the `launch_review_flow` method using the previously retrieved `ReviewInfo`.

You must subscribe to the events once before using the method:

```
Unset
on_request_review_flow_success;
on_request_review_review_flow_failure.
Subscription to events
func _ready():
    # Initialisation of _review_client

    _review_client.on_launch_review_flow_success.connect(_on_launch_review_flow_success)

    _review_client.on_launch_review_review_flow_failure.connect(_on_launch_review_flow_failure)

func _on_launch_review_review_flow_success():
    pass

func _on_launch_review_review_flow_flow_failure(error: RuStoreError):
    pass
```

```
Unset

Calling the launch_review_flow method
_review_client.launch_review_flow()
```

Wait for a notification that the user has completed the form in `on_launch_review_flow_success` or `on_launch_review_flow_failure` to continue the application.

The callback `on_launch_review_flow_flow_failure` returns a `RuStoreError` object with error information.

Error handling

The errors that occur can be retrieved in `*_failure` events.

Error structure

Unset

```
class_name RuStoreError extends Object

var description: String

func _init(json: String = ""):
    if json == "":
        description = ""
    else:
        var obj = JSON.parse_string(json)
        description = obj["detailMessage"]
```

- description – error description.

Possible errors

Possible errors you can get in `onFailure`:

- `RuStoreNotInstalledException()` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException()` — RuStore installed on a user's device doesn't support the start of the rating and feedback process.
- `RuStoreUserUnauthorizedException()` — the user is not logged in to RuStore.
- `RuStoreUserBannedException()` — the user is blocked in RuStore.
- `RuStoreApplicationBannedException()` — the app is blocked in RuStore.
- `RuStoreRequestLimitReached()` — too little time elapsed since the last display of the process.
- `RuStoreReviewExists()` — this user has already rated your app.
- `RuStoreInvalidReviewInfo()` — problems with `ReviewInfo`.
- `RuStoreException(message: String)` — RuStore basic error from which all the other errors are inherited.

React Native

General

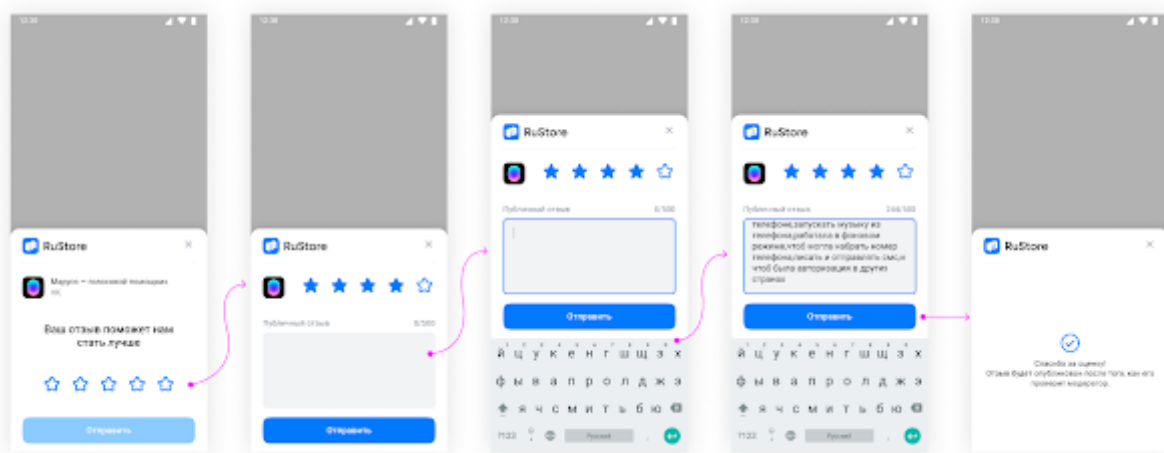
RuStore In-app Review SDK prompts the user to rate your app and leave feedback on the RuStore without exiting the app.

Rating and feedback user scenarios may be run at any time throughout the user's path in your app. The user can rate your app from 1 to 5 and leave feedback. Feedback is optional.

Implementation example

See the [example app](#) to learn how to integrate rating and feedback SDK correctly.

Use case example



Conditions for correct operation of SDK

For rating and feedback SDK to operate correctly, the following conditions need to be met:

11. Android 6.0 or later.
12. The RuStore app is installed on the user's device.
13. The current RuStoreApp version is installed on the user's device.
14. The user has logged in to the RuStore.
15. The app should be already installed on RuStore.

When to ask to rate and leave feedback

Use the tips below to decide when to ask the user to rate and leave feedback:

- Start the process once the user has been using your app for long enough.
- Avoid starting it too often as this will impair your app's user experience and limit the use or SDK ratings.

- Avoid using calls to action like “Rate App” button as the user could have already reached the process starting limit.
- Your app should not ask the user any questions before the start or while the process is running, including their opinion (“Do you like the app?”) or predictive questions (“Would you give this app 5 stars?”).

Design tips

Use the tips below to decide how to integrate the process:

- Display the process as is, without any intervention or modification of existing design, including size, opacity, shape and other properties.
- Add nothing on top or on sides of the process.
- The process should open on top of all layers. Don’t close the process after starting. The process will close by itself after an express action by the user.

Getting started

Run the following command to connect the package to the project.

```
Unset
// HTTPS.
npm install
git+https://git@gitflic.ru/project/rustore/react-native-rustore-review-sdk.git

// SSH
npm install
git+ssh://git@gitflic.ru/project/rustore/react-native-rustore-review-sdk.git
```

Feedback management

Preparing to work with evaluations

To display the evaluation window and review form, the plugin needs to be initialized.

Unset

```
RustoreReviewClient.init();
```

Start the evaluation of the application

After initialisation, you can query and display the form.

```
try {  
  
    const isRequested = await RustoreReviewClient.requestReviewFlow();  
  
    if (isRequested) {  
  
        await RustoreReviewClient.launchReviewFlow();  
  
    }  
  
} catch (err) {  
  
    console.log(err);  
  
}
```

Possible errors

Possible errors you can get in onFailure:

- `RuStoreNotInstalledException()` — user's device doesn't have RuStore installed.
- `RuStoreOutdatedException()` — RuStore installed on a user's device doesn't support the start of the rating and feedback process.
- `RuStoreUserUnauthorizedException()` — the user is not logged in to RuStore.
- `RuStoreUserBannedException()` — the user is blocked in RuStore.
- `RuStoreApplicationBannedException()` — the app is blocked in RuStore.
- `RuStoreRequestLimitReached()` — too little time elapsed since the last display of the process.
- `RuStoreReviewExists()` — this user has already rated your app.
- `RuStoreInvalidReviewInfo()` — problems with ReviewInfo.

- `RuStoreException(message: String)` — `RuStore` basic error from which all the other errors are inherited.

App Update SDK

Kotlin/Java

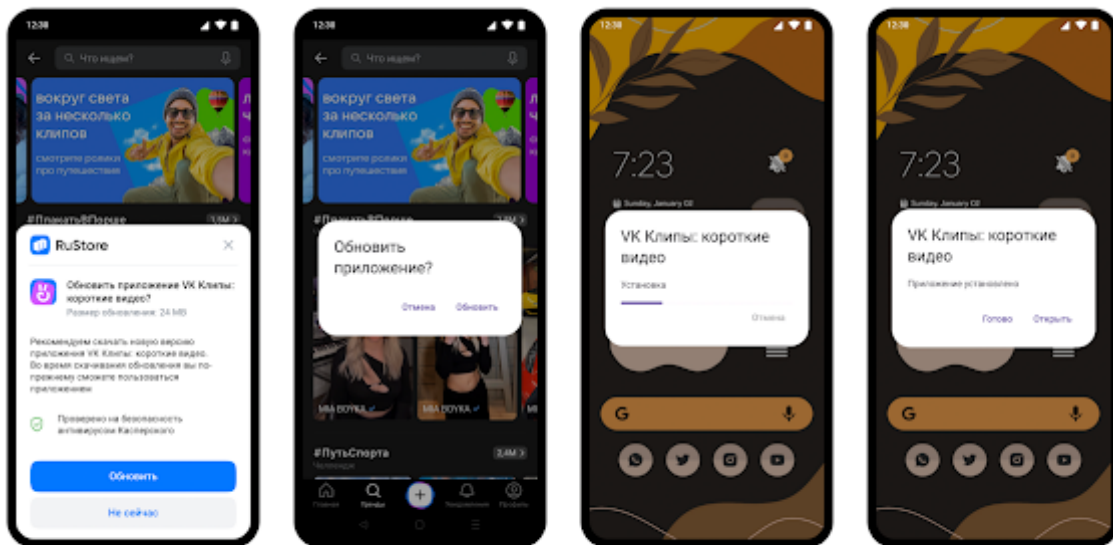
General Information	299
Importing SDK to your project	300
Creating an Update Manager	301
Checking for updates	302
Downloading updates	304
Installing updates	306
Possible errors	307

General Information

RuStore In-app updates SDK enable users to be kept up to date with the latest app version on their device. This allows them to stay informed about any performance enhancements or bug fixes that have been implemented.

Additionally, the SDK offers the ability to notify users of a new version and provide an option to install it. The installation process can occur in the background, while the user can track the progress of the update.

Use case example



Conditions for correct operation of SDK

For RuStore In-app updates SDK to operate correctly, the following conditions need to be met:

1. Android 6.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The RuStore app is allowed to install applications.

Importing SDK to your project

Connect the repository:

```
repositories {  
    maven {  
        url =  
uri("https://artifactory-external.vkpartner.ru/artifactory/maven")  
    }  
}
```

Dependency injection

Add the following code to your configuration file to inject the dependency:

```
dependencies {  
    implementation("ru.rustore.sdk:appupdate:2.0.0")  
}
```


Creating an Update Manager

Before you start, create an update manager using the Factory method:

```
val updateManager = RuStoreAppUpdateManagerFactory.create(context)
```

Checking for updates

Before requesting an update, check if it is available for your application. To check for updates, call the `getAppUpdateInfo()` method. When this method is called, the following conditions will be verified:

1. The RuStore app is installed on the user's device.
2. The current RuStoreApp version is installed on the user's device.
3. The user and the app should not be blocked on the RuStore.

Upon calling this method, an `AppUpdateInfo` object will be returned which contains information regarding any required updates. It is recommended to request and cache this object in advance, ensuring a prompt and convenient update download process for the user.

```
var appUpdateInfo: AppUpdateInfo? = null
updateManager
    .getAppUpdateInfo()
    .addOnSuccessListener { info ->
        appUpdateInfo = info
    }
    .addOnFailureListener { throwable ->

    }
```

The `updateInfo` object contains a set of parameters needed to determine if an update is available:

- `updateAvailability` — update availability:
 - UNKNOWN (Int == 0) — default status;
 - UPDATE_NOT_AVAILABLE (Int == 1) — no update required
 - UPDATE_AVAILABLE (Int == 2) — update needs to be downloaded or it has already been downloaded to the user's device.
 - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS — update is already being downloaded or installation is already running.
- `installStatus` — update installation status, if the user has already started the update installation at the time:
 - UNKNOWN (Int == 0) — by default;
 - DOWNLOADED (Int == 1) — downloaded;
 - DOWNLOADING (Int == 2) — being downloaded;;
 - FAILED (Int == 3) — error;
 - INSTALLING (Int == 4) — being installed;
 - PENDING (Int == 5) — waiting for download.

An update download can only be triggered if the `updateAvailability` field contains `UPDATE_AVAILABLE`.

This method may return an error.

Update scenario

Forced update

Availability check

Once `AppUpdateInfo` is received, you can check whether a push update is available.

`registerListener()` example

```
if (appUpdateInfo.isUpdateTypeAllowed(IMMEDIATE)) {  
    TODO()  
}
```

We recommend that you use the result of `isUpdateTypeAllowed` to decide whether to run a forced update. However this result does not affect the script's ability to run. Your internal app logic determines the necessity to run the update script.

Running the script

`startUpdateFlow()` example

```
updateManager  
    .startUpdateFlow(appUpdateInfo,  
AppUpdateOptions.Builder().appUpdateType(IMMEDIATE).build())  
    .addOnSuccessListener { resultCode ->  
  
    }  
    .addOnFailureListener { throwable ->  
  
    }
```

`resultCode (Int)` :

- `Activity.RESULT_OK (-1)` — update completed, the code may not be received as the app terminates at the time of update.
- `Activity.RESULT_CANCELED (0)` — flow interrupted by user or an error occurred. When you receive this code, you are expected to exit the app.
- `ActivityResult.ACTIVITY_NOT_FOUND (2)` — `RuStore` is not installed, or an installed version does not support forced updating (`RuStore versionCode < 191`)

`throwable` — error starting update script.

Delayed update

Downloading the Update

It is advisable to create a custom interface for managing the update process.

Once you've verified the update's availability, you can initiate the silent update script. The initial phase involves discreetly downloading the update in the background.

To monitor the download progress, you should employ the `registerListener()` method to add a listener.

`registerListener()` example

```
updateManager.registerListener { state ->
    if (state.installStatus == InstallStatus.DOWNLOADED) {
        // Update is ready to install
    }
}
```

The State object describes the current update download status. The object contains:

- `installStatus` — update installation status if the update is being installed at the moment:
 - `DOWNLOADED` (1) — successfully downloaded.
 - `DOWNLOADING` (2) — currently being downloaded.
 - `FAILED` (3) — error.
 - `INSTALLING` (4) — currently being installed.
 - `PENDING` (5) — awaiting update.
 - `UNKNOWN` (0) — by default.
- `bytesDownloaded` — number of bytes downloaded.
- `totalBytesToDownload` — total number of bytes that need to be downloaded.
- `installErrorCode` — error code during download.

If you no longer need a listener.

To discontinue listener functionality, employ the `unregisterListener()` method. This method enables you to remove a listener by providing the previously registered listener as an argument.

`unregisterListener()` example

```
updateManager.unregisterListener(listener)
```

To initiate the download of an app update, you need to invoke the `startUpdateFlow()` method, passing the `AppUpdateInfo` obtained from the `getAppUpdateInfo()` method, while also specifying the update type as `SILENT` within `AppUpdateOptions`.

It's important to note that the `AppUpdateInfo` object becomes obsolete after a single use. Therefore, to call the `startUpdateFlow()` method again, you should obtain a fresh `AppUpdateInfo` instance by making another request to the `getAppUpdateInfo()` method.

`startUpdateFlow()` example

```
updateManager
    .startUpdateFlow(appUpdateInfo,
AppUpdateOptions.Builder().build())
    .addOnSuccessListener { resultCode ->
        }
    .addOnFailureListener { throwable ->
        }
```

When the `onSuccessListener` is triggered with a `resultCode` of `Activity.RESULT_OK`, a task for downloading the update will be scheduled.

In this particular situation, either the `onSuccessListener` with `resultCode = Activity.RESULT_OK` or the `onFailureListener` can be invoked. For a comprehensive list of potential errors, please refer to the [Possible Errors](#) section.

Once the "DOWNLOADED" status is received, you can proceed to invoke the update installation method within the listener. It is advisable to notify the user that the update is prepared for installation at this point.

Installing the Update

To start the installation, use the `completeUpdate()` method.

`completeUpdate()` example

```
updateManager
    .completeUpdate()
    .addOnFailureListener { throwable ->

}
```

The update is carried out through the native android tool. If the update is successfully installed, the application will be closed.

This method can return an error.

Silent Update

Downloading the Update

Once you've verified the availability of the update, the next step is to commence the deferred update script. The initial action in this process involves silently downloading the update in the background. To monitor the download progress effectively, you'll need to employ the `registerListener()` method to add a listener.

`registerListener()` example

```
updateManager.registerListener { state ->
    if (state.installStatus == InstallStatus.DOWNLOADED) {
        // Update is ready to install
    }
}
```

The State object describes the current update download status. The object contains:

- `installStatus` — update installation status if the update is being installed at the moment:
 - `DOWNLOADED` (1) — successfully downloaded.
 - `DOWNLOADING` (2) — currently being downloaded.
 - `FAILED` (3) — error.
 - `INSTALLING` (4) — currently being installed.
 - `PENDING` (5) — awaiting update.
 - `UNKNOWN` (0) — by default.
- `bytesDownloaded` — number of bytes downloaded.
- `totalBytesToDownload` — total number of bytes that need to be downloaded.
- `installErrorCode` — error code during download.

If you no longer need a listener, use the `unregisterListener()` method to remove a listener, passing a previously registered listener to the method.

`unregisterListener()` example

```
updateManager.unregisterListener(listener)
```

To initiate the download of an app update, execute the `startUpdateFlow()` method while passing the `AppUpdateInfo` object acquired from the `getAppUpdateInfo()` method.

Keep in mind that the `AppUpdateInfo` object loses its validity after its initial use. To invoke the `startUpdateFlow()` method a second time, you must obtain a fresh `AppUpdateInfo` object using the `getAppUpdateInfo()` method.

Here's an example demonstrating the use of the `startUpdateFlow()` method:

```
val appUpdateOptions =
AppUpdateOptions.Builder().appUpdateType(SILENT).build()

updateManager
    .startUpdateFlow(appUpdateInfo, appUpdateOptions)
    .addOnSuccessListener { resultCode ->

    }
    .addOnFailureListener { throwable ->

    }
```

Upon the user's decision to download the update, the `resultCode` variable should be set to `Activity.RESULT_OK`. Conversely, if the user declines, `resultCode` should be set to `Activity.RESULT_CANCEL`.

Errors may occur during the process, so it's crucial to be aware of potential pitfalls.

Once the "DOWNLOADED" status is received, consider invoking the update installation method within the listener. It is advisable to inform the user that the update is ready for installation.

To commence the update installation, use the `completeUpdate()` method.

Here's an example of how to call the `completeUpdate()` method:

```
updateManager
    .completeUpdate()
    .addOnFailureListener { throwable ->
```



```
}
```

The update is carried out through the native android tool. If the update is successfully installed, the application will be closed.

This method can return an error.

Possible errors

If `onFailure` is received, we still do not recommend displaying an error to the user on your own as it can negatively impact the user experience.

Possible errors during the basic RuStore check.

- `RuStoreNotInstalledException()` — RuStore is not installed on the user's device;
- `RuStoreOutdatedException()` — RuStore is installed on the user's device but does not support application updates;
- `RuStoreUserUnauthorizedException()` — user is not logged in on the RuStore;
- `RuStoreException(message: String)` — basic RuStore error which gives rise to all other errors;
- `RuStoreInstallException()` — download and installation error.

Download and installation error codes:

- `ERROR_UNKNOWN` — unknown error;
- `ERROR_DOWNLOAD` — error while downloading;
- `ERROR_BLOCKED` — installation is blocked by the system;
- `ERROR_INVALID_APK` — invalid update APK;
- `ERROR_CONFLICT` — conflict with the current app version;
- `ERROR_STORAGE` — insufficient device storage;
- `ERROR_INCOMPATIBLE` — incompatible with device
- `ERROR_APP_NOT_OWNED` — application has not been purchased;
- `ERROR_INTERNAL_ERROR` — internal error;
- `ERROR_ABORTED` — user refused to install the update;
- `ERROR_APK_NOT_FOUND` — installation APK was not found;
- `ERROR_EXTERNAL_SOURCE_DENIED` — update is prohibited. For example, the first method responses that an update is not available, but the user calls the second method.

RuStore Update SDK Release Notes

SDK version 1.0.1

- Internal update

SDK version 1.0.0

- Internal update

SDK version 0.2.0

- Added flow force updates.
- Enhanced VersionCode Transfer with Long Type Support

SDK version 0.1.2

- Added Silent update functionality.

SDK version 0.1.1

- Fixed the `await()` method for the Task API.

Unreal

RuStore In-app updates plug-in 1.0 manual

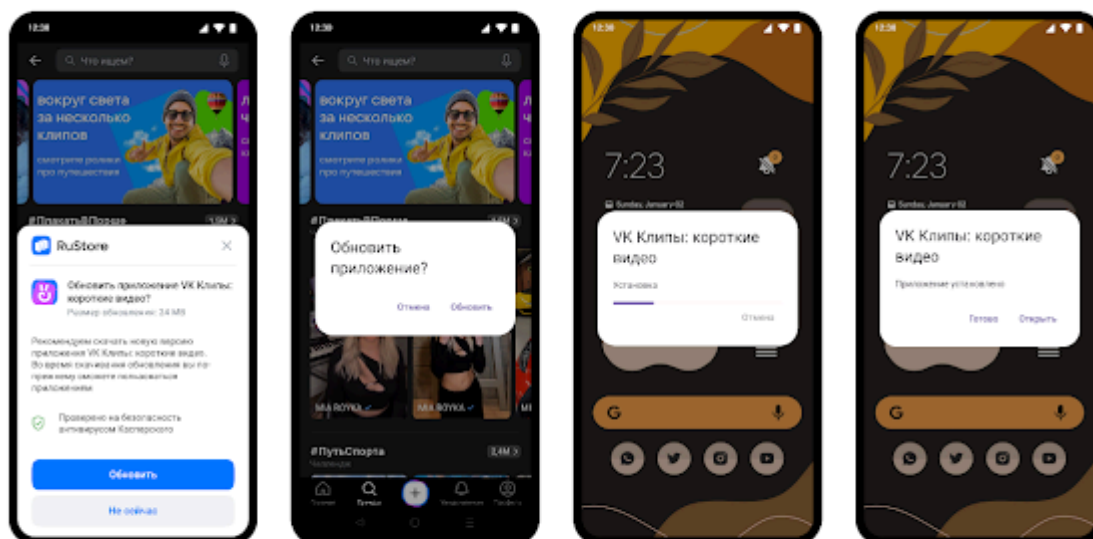
General information

The plug-in helps you to keep the up-to-date version of your app on the user's device.

When the users have an up-to-date version of the app, they can have a taste of new features and enjoy improved performance and bug fixes.

You can use RuStore In-app updates to display the app update process that ensures background download and update installation with status monitoring. For example, you can remind the user that there is a new version of your app available and suggest that they update. The user will be able to use your app during the update download.

User scenario example



Conditions for correct plug-in work

For RuStore In-app updates to work flawlessly, the following requirements must be met:

1. OS Android 7.0 or later.
2. RuStore is installed on the user's device.
3. The RuStoreApp version on the device is up-to-date.
4. The user is authorized in RuStore.
5. The RuStore app is allowed to install apps.

Connecting to project

1. Copy the contents of the “*Plugins*” folder from the official RuStore repository on [gitflie](https://github.com) to the “*Plugins*” folder of your project. Restart Unreal Engine, in the plug-in list (Edit → Plugins → Project → Mobile) select “RuStoreAppUpdate” and “RuStoreCore”.

2. In the “*YourProject.Build.cs*” file of the `PublicDependencyModuleNames` list connect the “RuStoreCore” and “RuStoreAppUpdate” modules.
3. In the project settings (Edit → Project Settings → Android) set the Minimum SDK Version parameter to level 24 or later and the Target SDK Version parameter to 31 or later.

Creating update manager

Create update manager before calling library methods.

```
URuStoreAppUpdateManager::Instance()->Init();
```

All operations with the manager are also available from Blueprints.

Blueprint implementation:

Important:

1. The `Init()` call ties the object to the scene root, and, If no further work with the object is needed, execute the `Dispose()` method to free memory.

The `Dispose()` method call will untie the object from root and securely complete all sent requests.

Deinitialization

```
URuStoreAppUpdateManager::Instance()->Dispose();
```

Blueprint implementation:

If you need to check whether the library is initialized or not, use

`URuStoreBillingClient::Instance()->getIsInitialized()`, its value is true if the library is initialized and false if `Init` hasn't been called yet.

Initialization check

```
URuStoreAppUpdateManager::Instance()->getIsInitialized();
```

Blueprint implementation:

Checking whether there are updates available

Before calling an update, check whether it is available for your app. To check the update availability, first call the `GetAppUpdateInfo()` method. When this method is called, the following conditions are checked:

1. RuStore is installed on the user's device.
2. The RuStoreApp version on the device is up-to-date.
3. The user is authorized in RuStore.
4. The user and the app are not banned in RuStore.
5. The RuStore app is allowed to install apps.

In response to this method, you will receive the `AppUpdateInfo` object that will contain the information whether it is necessary to update. Request the information in advance to request the user to download without delay and in a convenient time.

GetAppUpdateInfo() call example

```
long requestId = GetAppUpdateInfo(
```

```

        [(long requestId, TSharedPtr<FURuStoreAppUpdateInfo,
ESPMode::ThreadSafe> response) {
            // Process response
        },
        [(long requestId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
            // Process error
        }
    ];
};

```

Blueprint implementation:

The Success callback returns the AppUpdateInfo structure in the Response parameter:

The AppUpdateInfo object contains a set of parameters required to determine whether an update is available:

- updateAvailability - update availability:
 - UPDATE_NOT_AVAILABLE - no need for the update.
 - UPDATE_AVAILABLE - the update needs to be downloaded or is already downloaded to the user's device.
 - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS - the update is being downloaded or installed.
 - UNKNOWN - the default status.
- installStatus - the update installation status: if the user is in the process of installing the update:
 - DOWNLOADED - the update is downloaded.
 - DOWNLOADING - the update is being downloaded.
 - FAILED - error.
 - INSTALLING - the update is being installed.
 - PENDING - the update is pending.
 - UNKNOWN - the default status.

The update download is only available if the updateAvailability field has the UPDATE_AVAILABLE value.

The Failure callback returns the FURuStoreError structure that contains the error information in the Error parameter. All possible errors FURuStoreException are described in the “[Error handling](#)” section.

After receiving AppUpdateInfo you can check the availability of the immediate update using the CheckIsImmediateUpdateAllowed() method.

CheckIsImmediateUpdateAllowed() call example

```

bool available =
URuStoreAppUpdateManager::Instance()->CheckIsImmediateUpdateAllowed()
;

```

Blueprint implementation:

Downloading update

To start downloading the app update, call the StartUpdateFlow() method.

There are three update scenarios available:

EURuStoreAppUpdateOptions::DELAYED - delayed update.

EURuStoreAppUpdateOptions::SILENT - silent update.

EURuStoreAppUpdateOptions::IMMEDIATE - immediate update.

Important! To call the StartUpdateFlow() method again request AppUpdateInfo again using the GetAppUpdateInfo() method.

StartUpdateFlow() call example

```
EURuStoreAppUpdateOptions appUpdateOptions =
EURuStoreAppUpdateOptions::DELAYED;
long requestId = StartUpdateFlow(
    appUpdateOptions,
    [](long requestId, EURuStoreUpdateFlowResult response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```

Blueprint implementation:

The Failure callback returns the FURuStoreError structure with the error information in the Error parameter. All possible FURuStoreException errors are described in the [“Error handling”](#) section.

The Success callback returns the EURuStoreUpdateFlowResult value in the Response parameter:

- EURuStoreUpdateFlowResult::RESULT_OK (-1) - the update is complete; the code may fail to be sent as the app stops during the update.
- EURuStoreUpdateFlowResult::RESULT_CANCELED (0) - the flow was interrupted by the user or an error occurred. It is expected, that the app should be stopped on receiving this code.
- EURuStoreUpdateFlowResult::ACTIVITY_NOT_FOUND (2) - RuStore not installed, or the installed version does not support immediate updates (RuStore versionCode < 191)

After calling the StartUpdateFlow() method you can monitor the download status in the OnStateUpdatedInstanceEvent event.

OnStateUpdatedInstanceEvent

```
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FRuStoreOnStateUpdatedInstanceDelegate, int64, listenerId, FURuStoreInstallState, state);
UPROPERTY(BlueprintAssignable, Category = "RuStore AppUpdate Manager")
FRuStoreOnStateUpdatedInstanceDelegate OnStateUpdatedInstanceEvent;
```

Blueprint implementation:

On receiving `EURuStoreInstallStatus::DOWNLOADED`, in the `installStatus` field, you can call the update installation method.

Update installation

After downloading the APK update file, you can start the update installation. To start the update installation call the `CompleteUpdate()` method.

CompleteUpdate() call example

```
requestId = CompleteUpdate(  
    [](long requestId, TSharedPtr<FURuStoreError,  
    ESPMode::ThreadSafe> error) {  
        // Process error  
    }  
);
```

Blueprint implementation:

The update is done via the native Android tool. If the update is successful, the app will close.

The Failure callback returns the `FURuStoreError` structure with the error information in the `Error` parameter. All possible `FURuStoreException` errors are described in the “[Error handling](#)” section.

Possible errors

We do not recommend display an error to the user if you receive Failure in response. It can negatively affect the user experience.

Error structure:

Error structure

```
USTRUCT(BlueprintType)  
struct RUSTORECORE_API FURuStoreRuStoreError  
{  
    GENERATED_USTRUCT_BODY()  
  
    FURuStoreRuStoreError()  
    {  
        name = "";  
        description = "";  
    }  
  
    UPROPERTY(BlueprintReadOnly)  
    FString name;  
  
    UPROPERTY(BlueprintReadOnly)  
    FString description;  
};
```

List of possible errors:

- `RuStoreNotInstalledException` — RuStore is not installed on the user's device.
- `RuStoreOutdatedException` — the RuStore app installed on the user's device doesn't support updates.
- `RuStoreUserUnauthorizedException` — the user is not authorized in RuStore.

- RuStoreException — base RuStore error from which all other errors are inherited.
- RuStoreInstallException - Download and installation error.

RuStoreInstallException error codes:

- ERROR_UNKNOWN - Unknown error.
- ERROR_DOWNLOAD - Download error.
- ERROR_BLOCKED - Installation blocked by the systems.
- ERROR_INVALID_APK - Invalid update APK.
- ERROR_CONFLICT - There is a conflict with the current app version.
- ERROR_STORAGE - Not enough memory on the device.
- ERROR_INCOMPATIBLE - incompatible with the device.
- ERROR_APP_NOT_OWNED - The app is not purchased.
- ERROR_INTERNAL_ERROR - Internal error.
- ERROR_ABORTED - The user aborted the initialization.
- ERROR_APK_NOT_FOUND - APK for installation not found.
- ERROR_EXTERNAL_SOURCE_DENIED - Installation start denied. For example, the first request returned response that the update is unavailable, despite that, the user calls the second method.
- ERROR_ACTIVITY_SEND_INTENT - Error while sending intent for opening an activity.
- ERROR_ACTIVITY_UNKNOWN - Unknown error on activity opening.

Unity

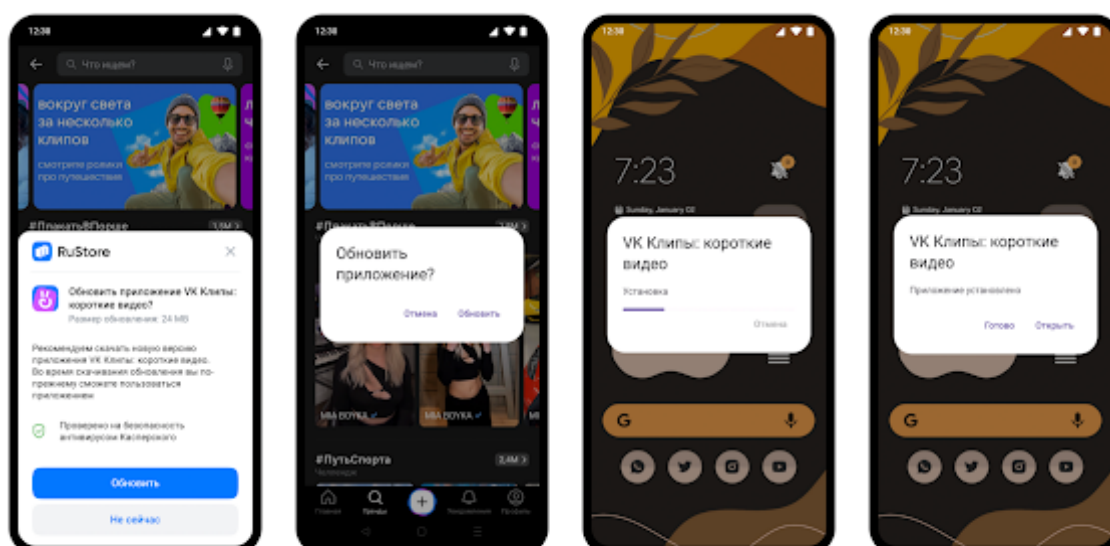
General Information	309
Importing SDK to your project	310
Creating an Update Manager	311
Checking for updates	312
Downloading updates	314
Installing updates	317
Handling errors	318

General Information

RuStore In-app updates SDK enable users to be kept up to date with the latest app version on their device. This allows them to stay informed about any performance enhancements or bug fixes that have been implemented.

Moreover, the SDK offers the ability to notify users of a new version and provide an option to install it. The installation process can occur in the background, while the user can track the progress of the update.

Use case example



Conditions for correct operation of SDK

For RuStore In-app updates SDK to operate correctly, the following conditions need to be met:

1. Android 7.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The user has logged in to the RuStore.
5. The RuStore app is allowed to install applications.

Importing SDK to your project

To connect the SDK, you need to download [RuStore AppUpdate SDK](#) and import it into the project. To do this, go to Assets → Import Package → Custom Package). Dependencies are connected automatically using the External Dependency Manager (included in the SDK).

The minimum API level must be set to at least 24. Application minification (ProGuard/R8) is currently not supported, it must be disabled in the project settings. Go to File → Build Settings → Player Settings → Publishing Settings → Minify.

Creating an Update Manager

Before you start, create an update manager using the Factory method:

```
RuStoreAppUpdateManager.Instance.Init();
```

Checking for updates

Before requesting an update, check if it is available for your application. To check for updates, call the `getAppUpdateInfo()` method. When this method is called, the following conditions will be verified:

1. The RuStore app is installed on the user's device.
2. The current RuStoreApp version is installed on the user's device.
3. The user has logged in to the RuStore.
4. The user and the app should not be blocked on the RuStore.
5. The RuStore app is allowed to install applications.

Upon calling this method, an `AppUpdateInfo` object will be returned which contains information regarding any required updates. It is recommended to request and cache this object in advance, ensuring a prompt and convenient update download process for the user.

GetAppUpdateInfo() method example

```
RuStoreAppUpdateManager.Instance.GetAppUpdateInfo(onFailure:
(error) => {
    // Handle error
},
onSuccess: (info) => {
    // Process update info
});
```

`AppUpdateInfo` contains a set of parameters needed to determine whether there is an update available:

- `updateAvailability` — update availability:
 - `UNKNOWN` — default status;
 - `UPDATE_NOT_AVAILABLE` — no update required
 - `UPDATE_AVAILABLE` — update needs to be downloaded or it has already been downloaded to the user's device.
 - `DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS` — update is already being downloaded or installation is already running.
- `installStatus` — update installation status, if the user has already started the update installation at the time:
 - `DOWNLOADED` — downloaded;
 - `DOWNLOADING` — being downloaded;;
 - `FAILED` — error;
 - `INSTALLING` — being installed;
 - `PENDING` — waiting for download;
 - `UNKNOWN` — by default;

An update download can only be triggered if the `updateAvailability` field contains `UPDATE_AVAILABLE`.

This method may return an error. For detailed information about possible errors, refer to [Handling errors](#).

Downloading updates

Once an update is available, you can request the user to download the update, but before doing so, you must start the update download status listener using the `registerListener()` method.

RegisterListener() method example

```
RuStoreAppUpdateManager.Instance.RegisterListener(listener);
```

`listener` — an object that implements an interface `IInstallStateUpdateListener`.

IInstallStateUpdateListener

```
public interface IInstallStateUpdateListener {  
  
    public void OnStateUpdated(InstallState state);  
  
}
```

The state object describes the current update download status. The object contains:

- `installStatus` — update installation status, if the user has already started update installation at the time:
 - `DOWNLOADED` — downloaded;
 - `DOWNLOADING` — being downloaded;
 - `FAILED` — error;
 - `INSTALLING` — being installed;
 - `PENDING` — waiting for download;
 - `UNKNOWN` — by default.
- `bytesDownloaded` — number of bytes downloaded.
- `totalBytesToDownload` — total number of bytes to be downloaded.
- `installErrorCode` — error code during the download. Possible errors
 - `ERROR_UNKNOWN` — Unknown error.
 - `ERROR_DOWNLOAD` — Download error.
 - `ERROR_BLOCKED` — Installation blocked by the system.
 - `ERROR_INVALID_APK` — Invalid update APK.
 - `ERROR_CONFLICT` — Conflict with the current app version.
 - `ERROR_STORAGE` — Insufficient storage.
 - `ERROR_INCOMPATIBLE` — Incompatible with the device.
 - `ERROR_APP_NOT_OWNED` — App purchase has not been completed.
 - `ERROR_INTERNAL_ERROR` — Internal error.

- `ERROR_ABORTED` — Update refused to the user.
- `ERROR_APK_NOT_FOUND` — No APK file found.
- `ERROR_EXTERNAL_SOURCE_DENIED` — Update prohibited. For example, the first method responds that an update is not available, but the user calls the second method.

If the listener is no longer required, then use the `unregisterListener()` method to remove the listener, by passing the previously registered listener to the method.

UnregisterListener() method example

```
RuStoreAppUpdateManager.Instance.UnregisterListener(listener);
```

Call the `startUpdateFlow()` method to start downloading an app update.

Once used, an `AppUpdateInfo` object becomes invalid. To call the `startUpdateFlow()` method again, request `AppUpdateInfo` using the `getAppUpdateInfo()` method again.

StartUpdateFlow() method example

```
RuStoreAppUpdateManager.Instance.StartUpdateFlow(
    onFailure: (error) => {
        // Handle error
    },
    onSuccess: (resultCode) => {
        // Handle flow result
    });
```

If the user confirmed the update download, use `resultCode = UpdateFlowResult.RESULT_OK`, if it was refused, then use `resultCode = UpdateFlowResult.RESULT_CANCELED`.

After calling the method, you can monitor the update download status in `Listener`. If you received the `DOWNLOADED` status in `Listener`, you can call the update install method. It is recommended to inform the user when the update is ready for installation.

This method may return an error.

Installing updates

Once you have downloaded the update APK file, you can start installing the update. To start the update installation, call the `CompleteUpdate()` method.

CompleteUpdate() method example

```
RuStoreAppUpdateManager.Instance.CompleteUpdate(  
    onFailure: (error) => {  
        // Handle error  
    });
```

The update is processed through the native android tool. If the update is successful, the application will be closed.

Errors may occur at this step. For more information, refer to [Handling errors](#)

Handling errors

If `onFailure` is received, we still do not recommend displaying an error to the user on your own as it can negatively impact the user experience.

Error structure:

```
public class RuStoreError {  
  
    public string name;  
    public string description;  
}
```

The list of possible errors:

- `RuStoreNotInstalledException` — RuStore is not installed on the user's device;
- `RuStoreOutdatedException` — RuStore is installed on the user's device but does not support application updates;
- `RuStoreUserUnauthorizedException` — user is not logged in on the RuStore;
- `RuStoreException` — basic RuStore error which gives rise to all other errors;
- `RuStoreInstallException` — download and installation error.

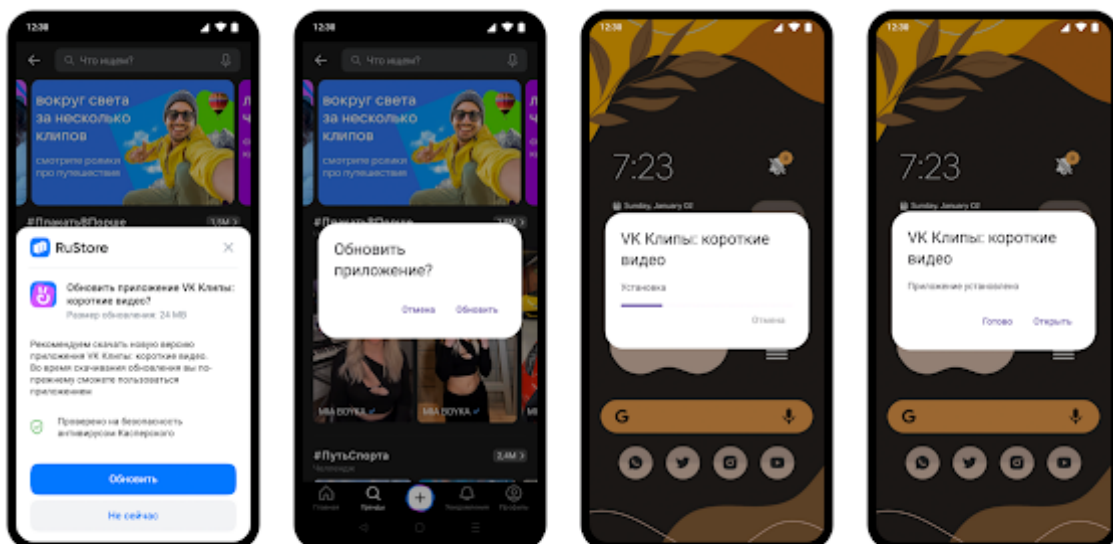
Flutter

General Information

RuStore In-app updates SDK enable users to be kept up to date with the latest app version on their device. This allows them to stay informed about any performance enhancements or bug fixes that have been implemented.

Additionally, the SDK offers the ability to notify users of a new version and provide an option to install it. The installation process can occur in the background, while the user can track the progress of the update.

Use case example



Conditions for correct operation of SDK

For RuStore In-app updates SDK to operate correctly, the following conditions need to be met:

1. Android 6.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The RuStore app is allowed to install applications.

Importing SDK to your project

Run the script below to connect the SDK:

```
flutter pub add flutter_rustore_update
```

This command adds a line in pubspec.yaml: :

```
dependencies:  
  flutter_rustore_update: ^0.0.2
```

Checking for updates

Before requesting an update, check if it is available for your application. To check for updates, call the `info()`. When this method is called, the following conditions will be verified:

1. The RuStore app is installed on the user's device.
2. The current RuStoreApp version is installed on the user's device.
3. The user and the app should not be blocked on the RuStore.

Upon calling this method, an `Info` object will be returned which contains information regarding any required updates.

```
RustoreUpdateClient.info().then((info) {  
    print(info);  
}).catchError((err) {  
    print(err);  
});
```

The `updateInfo` object contains a set of parameters needed to determine if an update is available:

- `updateAvailability` — update availability:
 - `UPDATE_AILABILITY_NOT_AVAILABLE` — no update required
 - `UPDATE_AILABILITY_AVAILABLE` — update needs to be downloaded or it has already been downloaded to the user's device.
 - `UPDATE_AILABILITY_IN_PROGRESS` — update is already being downloaded or installation is already running.
 - `UPDATE_AILABILITY_UNKNOWN` — default status.
- `installStatus` — update installation status, if the user has already started the update installation at the time:
 - `INSTALL_STATUS_DOWNLOADED` — downloaded.
 - `INSTALL_STATUS_DOWNLOADING` — being downloaded.
 - `INSTALL_STATUS_FAILED` — error.
 - `INSTALL_STATUS_INSTALLING` — being installed.
 - `INSTALL_STATUS_PENDING` — waiting for download.
 - `INSTALL_STATUS_UNKNOWN` — by default.

An update download can only be triggered if the `updateAvailability` field contains `UPDATE_AVAILABLE`.

This method may return an error.

Downloading the Update

Upon confirming the availability of an update, you can prompt the user to initiate the download. However, prior to proceeding, it's essential to initialize a listener for tracking the download status of the update using the `listener()` method.

```
RustoreUpdateClient.listener((value) {
    print("listener installStatus ${value.installStatus}");
    print("listener bytesDownloaded ${value.bytesDownloaded}");
    print("listener totalBytesToDownload ${value.totalBytesToDownload}");
    print("listener installErrorCode ${value.installErrorCode}");

    if (value.installStatus == INSTALL_STATUS_DOWNLOADED) {
    }
});
```

The *state* object describes the current update download status. It contains:

- `installStatus` — update installation status, if the user has already started the update installation at the time:
 - `INSTALL_STATUS_DOWNLOADED` — downloaded.
 - `INSTALL_STATUS_DOWNLOADING` — being downloaded.
 - `INSTALL_STATUS_FAILED` — error.
 - `INSTALL_STATUS_INSTALLING` — being installed.
 - `INSTALL_STATUS_PENDING` — waiting for download.
 - `INSTALL_STATUS_UNKNOWN` — by default.
- `bytesDownloaded` — number of bytes downloaded.
- `totalBytesToDownload` — total number of bytes that need to be downloaded.
- `installErrorCode` — error code during download.

Downloading with RuStore UI

To start downloading an application update, call the `download()` method.

```
RustoreUpdateClient.download().then((value) {
    print("download code ${value.code}");
}).catchError((err) {
    print("download err ${err}");
});
```

When the user confirms the download of the update, the `value.code` should be set to `ACTIVITY_RESULT_OK`. Conversely, if the user declines, the `value.code` should be set to `ACTIVITY_RESULT_CANCELED`.

After invoking the method, you can track the update download status through the listener. If the listener reports the `INSTALL_STATUS_DOWNLOADED` status, you can proceed to execute the

update installation method. It is advisable to inform the user that the update is ready for installation.

Forced Update

To start downloading a mandatory app update, call the `immediate()` method.

```
RustoreUpdateClient.immediate().then((value) {  
    print("immediate code ${value.code}");  
}).catchError((err) {  
    print("immediate err ${err}");  
});
```

Once the user confirms the download of the update, set `value.code` to `ACTIVITY_RESULT_OK`. In the event of a refusal, set `value.code` to `ACTIVITY_RESULT_CANCELED`.

Subsequent to invoking the method, it's possible to keep track of the update download status through the listener. If the listener reports the `INSTALL_STATUS_DOWNLOADED` status, you should proceed to invoke the update installation method. We strongly recommend notifying the user that the update is prepared for installation.

Silent update

To start downloading an application update without the RuStore interface, call the `silent()` method.

```
RustoreUpdateClient.silent().then((value) {  
    print("silent code ${value.code}");  
}).catchError((err) {  
    print("silent err ${err}");  
});
```

Once you've triggered the method, you can actively monitor the update download status via the listener. If you receive the `INSTALL_STATUS_DOWNLOADED` status within the listener, you should proceed to initiate the update installation method. It is advisable to promptly inform the user that the update is now ready for installation.

Installing the Update

Once the update apk file is downloaded, you can start installing the update. To start installing the update, call the `complete()` method.

```
RustoreUpdateClient.complete().catchError((err) {  
  print("complete err ${err}");  
});
```

The update process is facilitated using the native Android tool. Upon a successful update, the application will automatically close.

Possible errors

All errors in the plugin are implemented using constants. Description of constants is listed in `const.dart`.

If you receive an `onFailure` response, we do not recommend displaying the error to the user as it may negatively impact the user experience.

List of possible errors:

-
- `UPDATE_ERROR_DOWNLOAD` — Error while downloading.
- `UPDATE_ERROR_BLOCKED` — Installation is blocked by the system.
- `UPDATE_ERROR_INVALID_APK` — Incorrect APK update.
- `UPDATE_ERROR_CONFLICT` — Conflict with the current app version.
- `UPDATE_ERROR_STORAGE` — Not enough device space.
- `UPDATE_ERROR_INCOMPATIBLE` — Incompatible with the device.
- `UPDATE_ERROR_APP_NOT_OWNED` — Application has not been purchased.
- `UPDATE_ERROR_INTERNAL_ERROR` — Internal error.
- `UPDATE_ERROR_ABORTED` — Update referred by user.
- `UPDATE_ERROR_APK_NOT_FOUND` — APK not found.
- `UPDATE_ERROR_EXTERNAL_SOURCE_DENIED` — Running the update is prohibited. For example, the first method returns a response stating that the update is not available, but the user calls the second method.

RuStore Update SDK Release Notes

SDK version 0.0.3

- **Internal update**

SDK version 0.0.2

- **Internal update**

SDK version 0.0.1

- **Update functionality implemented**

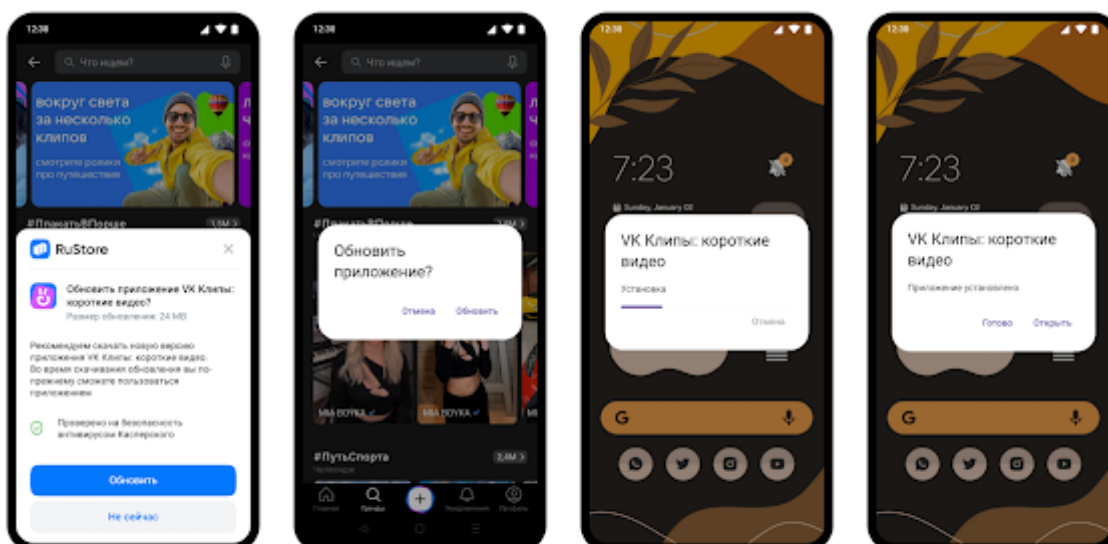
Unreal

General Information

RuStore In-app updates SDK enable users to be kept up to date with the latest app version on their device. This allows them to stay informed about any performance enhancements or bug fixes that have been implemented.

Moreover, the SDK offers the ability to notify users of a new version and provide an option to install it. The installation process can occur in the background, while the user can track the progress of the update.

Use case example



Conditions for correct operation of SDK

For RuStore In-app updates SDK to operate correctly, the following conditions need to be met:

1. Android 7.0 or later.
2. The RuStore app is installed on the user's device.
3. The current RuStoreApp version is installed on the user's device.
4. The user has logged in to the RuStore.
5. The RuStore app is allowed to install applications.

Importing SDK to your project

To connect the SDK, you need to download the Unreal Engine plugins RuStoreCore and RuStoreAppUpdate from the official RuStore [gitflic](#) repository, then place them in the Plugins folder inside the project. The RuStoreCore and RuStoreAppUpdate plugins will appear in the list (Edit → Plugins → Project → Mobile). Once where you need to connect the RuStoreCore and RuStoreAppUpdate modules in PublicDependencyModuleNames list of the *YourProject*.Build.cs file.

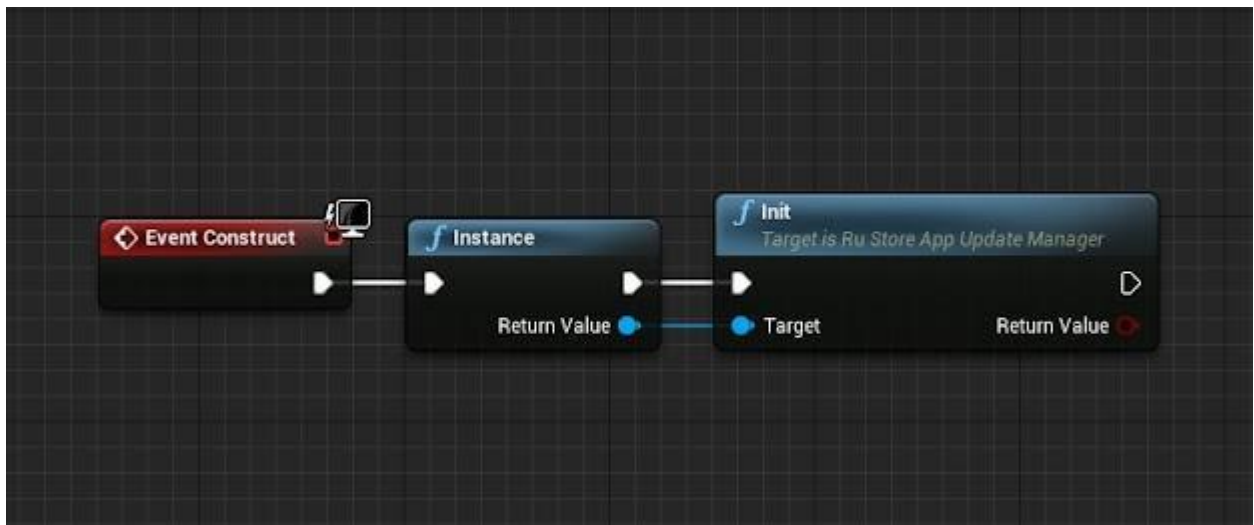
When building an Android application, Minimum SDK Version must be set to level no lower than 24, Target SDK Version must be set to no lower than 31. Application minification (ProGuard/R8) is not currently supported. All necessary gradle settings and project dependencies are specified in the RuStoreCore_UPL_Android.xml and RuStoreAppUpdate_UPL_Android.xml files.

Creating an Update Manager

Before you start, create an update manager:

```
URuStoreAppUpdateManager::Instance()->Init();
```

All manager operations are also accessible via Blueprints. Initialization example:

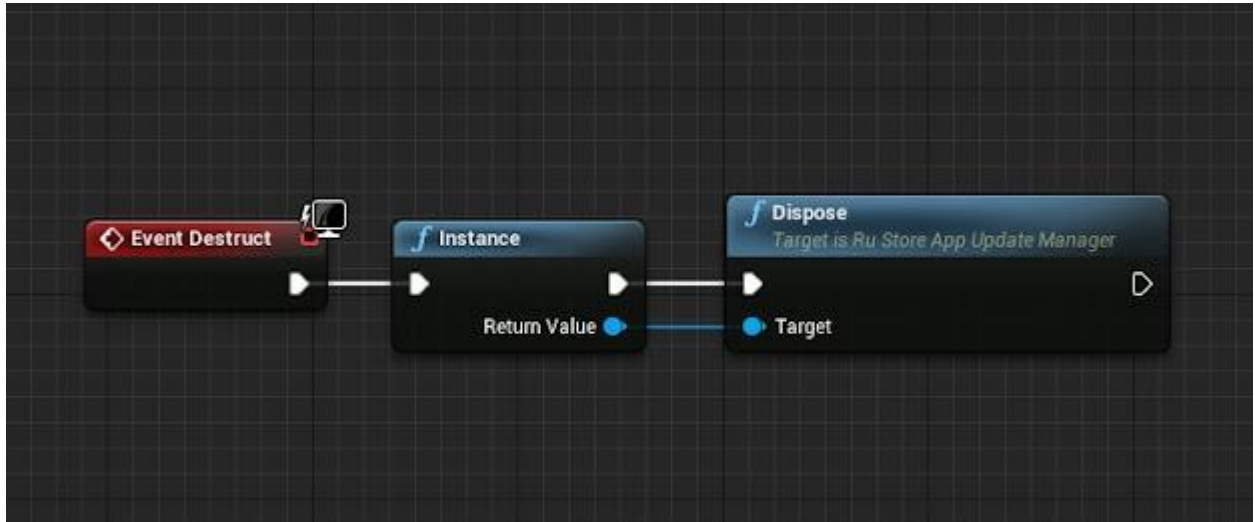


The Init() call binds the object to the scene root, and if no further work is planned on the object, the Dispose() method must be called to free up memory.

Calling the Dispose() method will unbind the object from the root and safely complete all requests sent.

Deinitialization

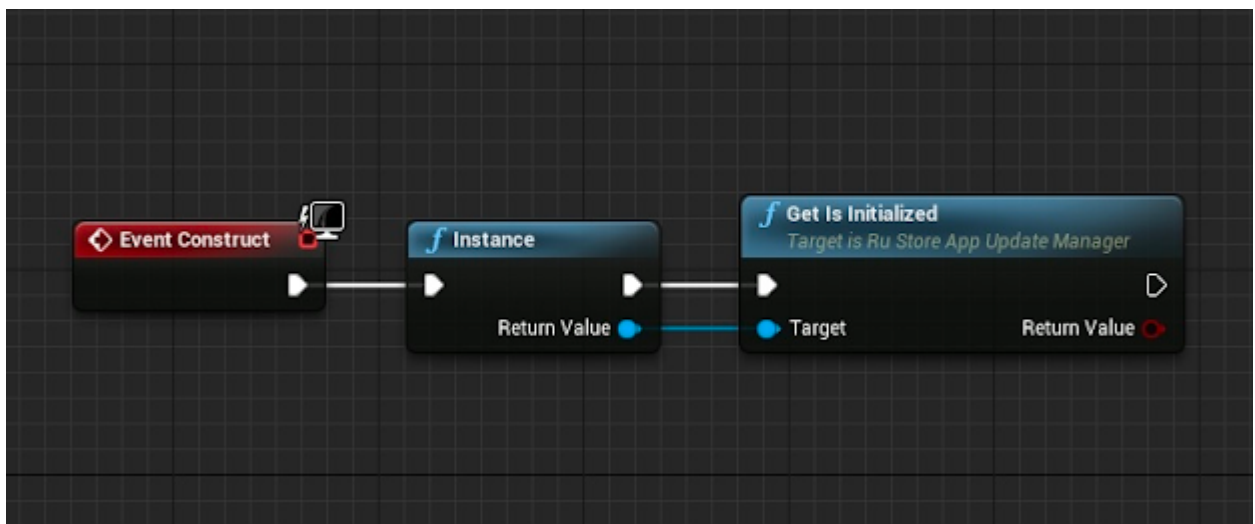
```
URuStoreAppUpdateManager::Instance()->Dispose();
```



If you need to check whether the library has been initialized, use the `URuStoreBillingClient::Instance()->getIsInitialized()` property. Its value is true if the library is initialized, and false if Init has not yet been called.

Initialization check

```
URuStoreAppUpdateManager::Instance()->getIsInitialized();
```



Checking for updates

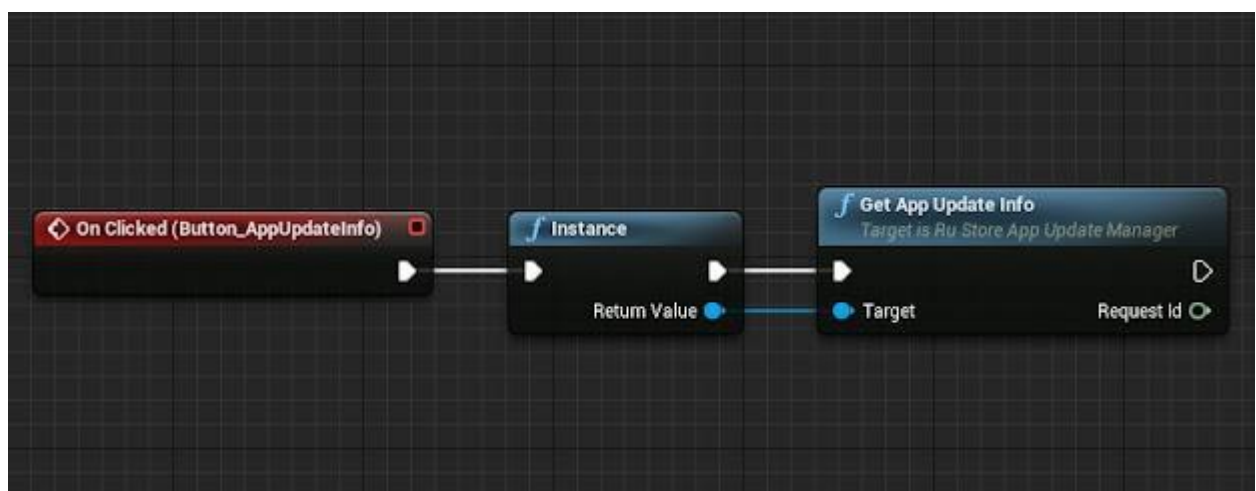
Before requesting an update, check if it is available for your application. To check for updates, call the `getAppUpdateInfo()` method. When this method is called, the following conditions will be verified:

1. The RuStore app is installed on the user's device.
2. The current RuStoreApp version is installed on the user's device.
3. The user has logged in to the RuStore.
4. The user and the app should not be blocked on the RuStore.
5. The RuStore app is allowed to install applications.

Upon calling this method, an `AppUpdateInfo` object will be returned which contains information regarding any required updates. It is recommended to request and cache this object in advance, ensuring a prompt and convenient update download process for the user.

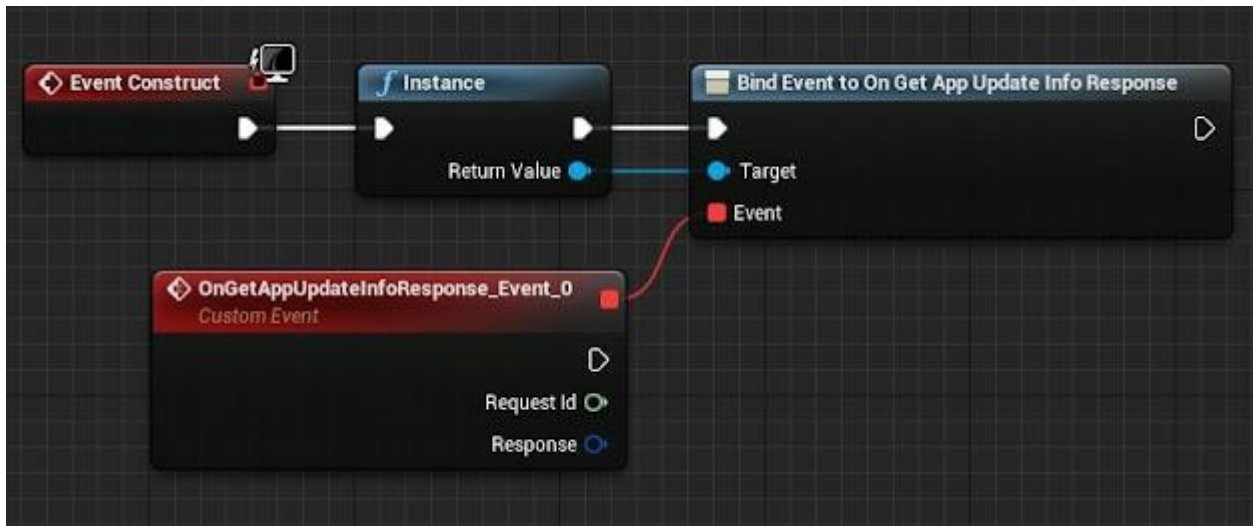
GetAppUpdateInfo() method example

```
long requestId = GetAppUpdateInfo(  
    [](long requestId, TSharedPtr<FURuStoreAppUpdateInfo,  
    ESPMode::ThreadSafe> response) {  
        // Process response  
    },  
    [](long requestId, TSharedPtr<FURuStoreError,  
    ESPMode::ThreadSafe> error) {  
        // Process error  
    }  
);
```

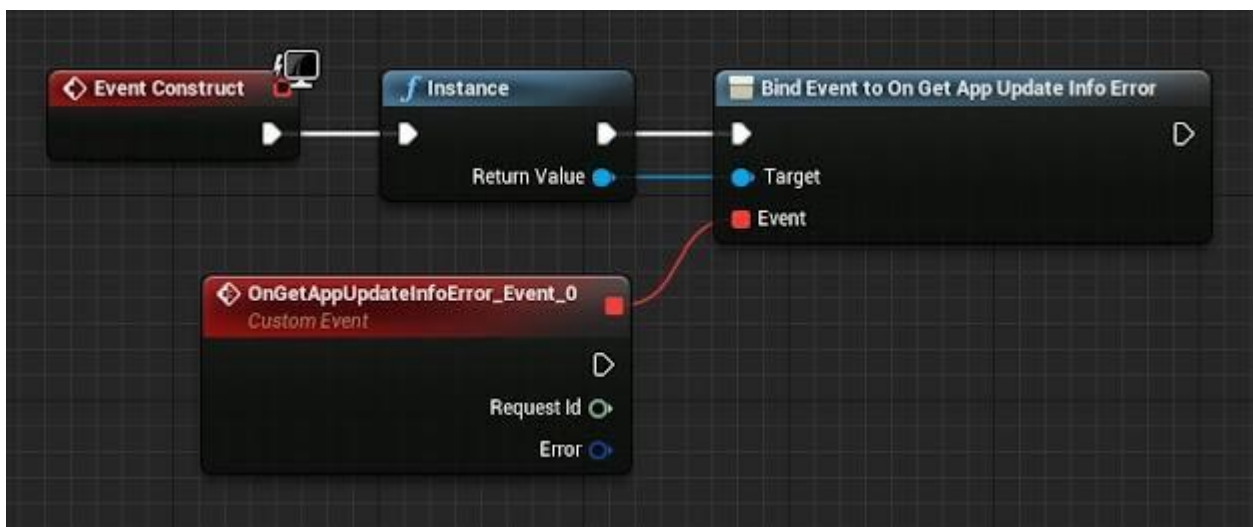


To process a response using Blueprints, you must subscribe to OnGetAppUpdateInfoResponse and OnGetAppUpdateInfoError events.

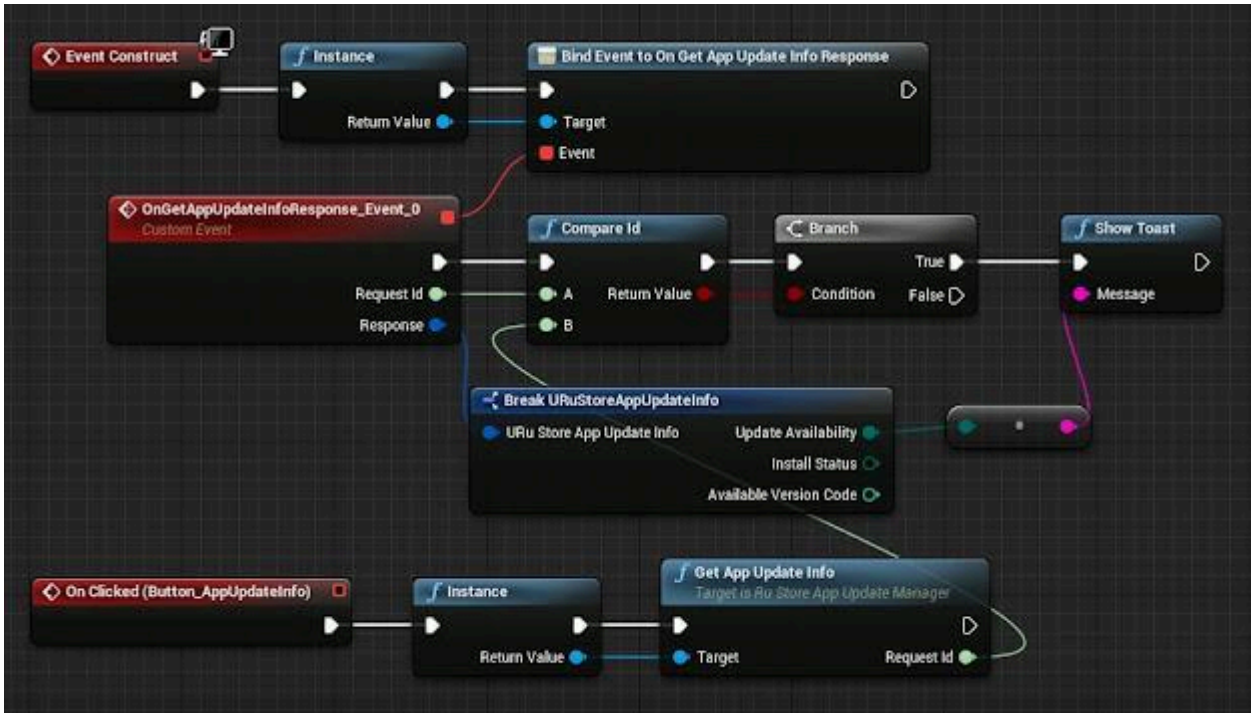
OnGetAppUpdateInfoResponse event subscription example:



OnGetAppUpdateInfoError event subscription example:



All Blueprint-events in the plugin are broadcast. A given request can be filtered using the requestId parameter. Each call to the GetAppUpdateInfo() method returns a unique requestId, Blueprint-event returns the requestId of the method that generated it.



AppUpdateInfo contains a set of parameters needed to determine whether there is an update available:

- updateAvailability — update availability:
 - UNKNOWN — default status;
 - UPDATE_NOT_AVAILABLE — no update required
 - UPDATE_AVAILABLE — update needs to be downloaded or it has already been downloaded to the user's device.
 - DEVELOPER_TRIGGERED_UPDATE_IN_PROGRESS — update is already being downloaded or installation is already running.
- installStatus — update installation status, if the user has already started the update installation at the time:
 - DOWNLOADED — downloaded;
 - DOWNLOADING — being downloaded;;
 - FAILED — error;
 - INSTALLING — currently being installed.
 - PENDING — pending.
 - UNKNOWN — by default.

An update download can only be triggered if the updateAvailability field contains UPDATE_AVAILABLE.

This method may return an error. For detailed information about possible errors, refer to Handling errors.

Downloading updates

Once an update is available, you can request the user to download the update, but before doing so, you must start the update download status listener using the `registerListener()` method.

RegisterListener() method example

```
URuStoreAppUpdateManager::Instance()->RegisterListener(listener);
```

`listener` — an object that implements an interface `IInstallStateUpdateListener`.

IInstallStateUpdateListener

```
UINTERFACE(Blueprintable)
class RUSTOREAPPUPDATE_API
URuStoreInstallStateUpdateListenerInterface : public UInterface
{
    GENERATED_BODY()
};

class IRuStoreInstallStateUpdateListenerInterface
{
    GENERATED_BODY()

public:
    UFUNCTION(BlueprintCallable, BlueprintNativeEvent, Category =
"RuStore InstallStateUpdate Listener Interface")
    void OnStateUpdated(int64 listenerId, FURuStoreInstallState&
state);
};
```

The state object describes the current update download status. The object contains:

- `installStatus` — update installation status, if the user has already started update installation at the time:
 - `DOWNLOADED` — downloaded;
 - `DOWNLOADING` — being downloaded;

- FAILED — error;
- INSTALLING — being installed;
- PENDING — waiting for download;
- UNKNOWN — by default.
- bytesDownloaded — number of bytes downloaded.
- totalBytesToDownload — total number of bytes to be downloaded.
- installErrorCode — error code during the download. Possible errors
 - ERROR_UNKNOWN — Unknown error.
 - ERROR_DOWNLOAD — Download error.
 - ERROR_BLOCKED — Installation blocked by the system.
 - ERROR_INVALID_APK — Invalid update APK.
 - ERROR_CONFLICT — Conflict with the current app version.
 - ERROR_STORAGE — Insufficient storage.
 - ERROR_INCOMPATIBLE — Incompatible with the device.
 - ERROR_APP_NOT_OWNED — App purchase has not been completed.
 - ERROR_INTERNAL_ERROR — Internal error.
 - ERROR_ABORTED — Update refused to the user.
 - ERROR_APK_NOT_FOUND — No APK file found.
 - ERROR_EXTERNAL_SOURCE_DENIED — Update prohibited. For example, the first method responds that an update is not available, but the user calls the second method.

If the listener is no longer required, then use the `unregisterListener()` method to remove the listener, by passing the previously registered listener to the method.

UnregisterListener() method example

```
URuStoreAppUpdateManager::Instance()->UnregisterListener(listener);
```

Call the `startUpdateFlow()` method to start downloading an app update.

To call the `startUpdateFlow()` method again, request `AppUpdateInfo` using the `getAppUpdateInfo()` method again.

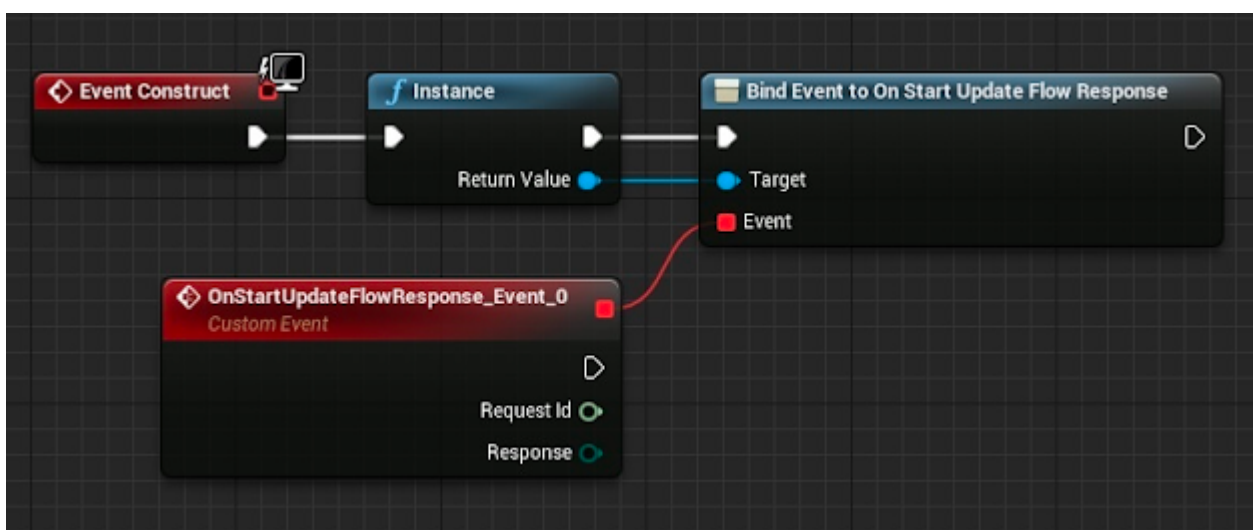
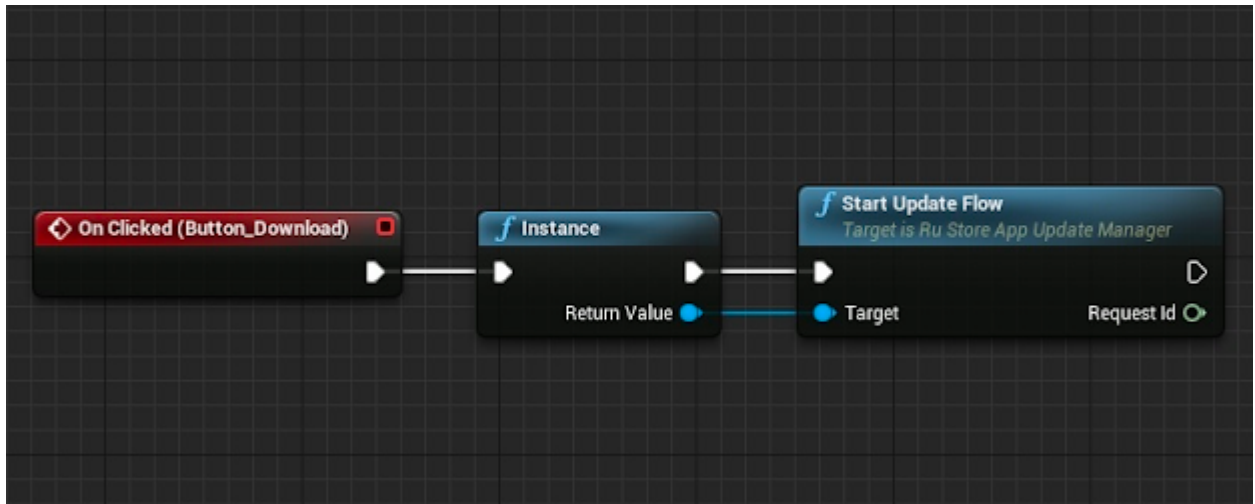
StartUpdateFlow() method example

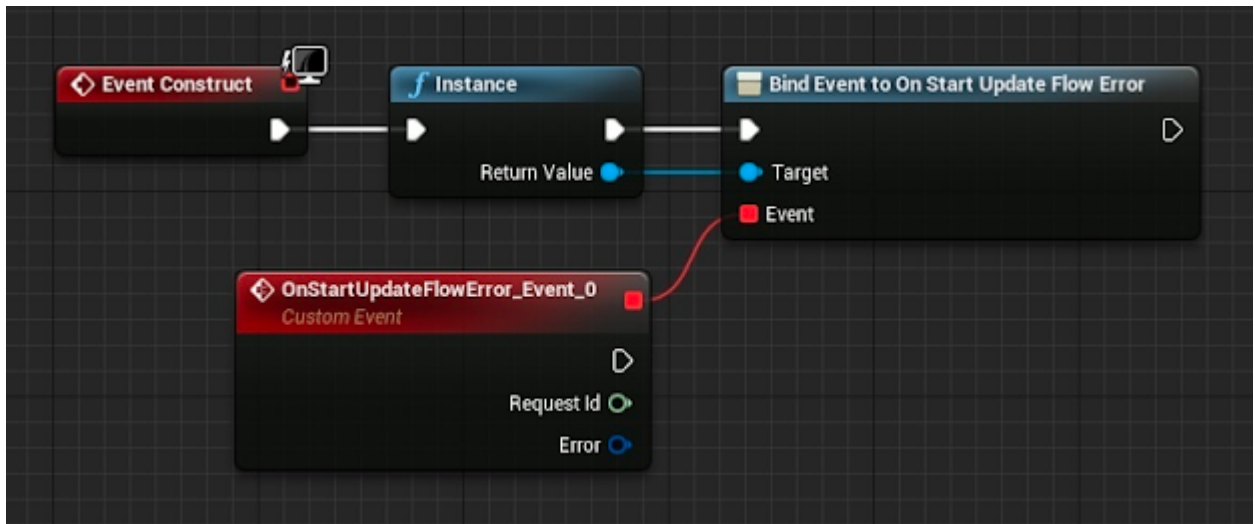
```
long requestId = StartUpdateFlow(
```

```

    [](long requestId, EURuStoreUpdateFlowResult response) {
        // Process response
    },
    [](long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
};

```





If the user confirmed the update download, use `resultCode = UpdateFlowResult.RESULT_OK`, if it was refused, then use `resultCode = UpdateFlowResult.RESULT_CANCELED`.

After calling the method, you can monitor the update download status in Listener. If you received the DOWNLOADED status in Listener, you can call the update install method. It is recommended to inform the user when the update is ready for installation.

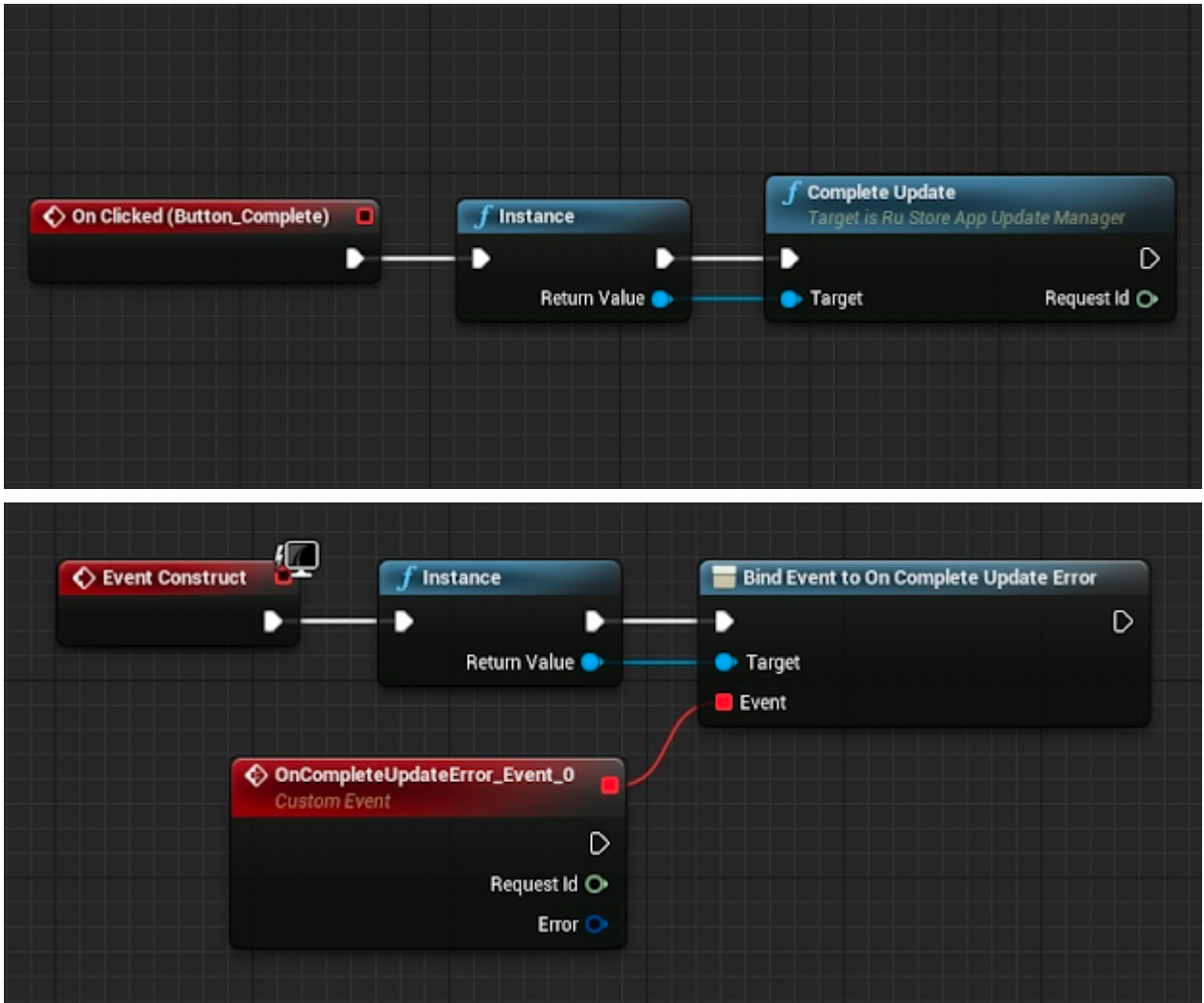
This method may return an error.

Installing updates

Once you have downloaded the update APK file, you can start installing the update. To start the update installation, call the `CompleteUpdate()` method.

CompleteUpdate() method example

```
requestId = CompleteUpdate(
    [(long requestId, TSharedPtr<FURuStoreError,
    ESPMode::ThreadSafe> error) {
        // Process error
    }
);
```



The update is processed through the native android tool. If the update is successful, the application will be closed.

Errors may occur at this step. For more information, refer to Handling errors

Handling errors

If onFailure is received, we still do not recommend displaying an error to the user on your own as it can negatively impact the user experience.

Error structure:

```

    USTRUCT(BlueprintType)
    struct RUSTORECORE_API FURuStoreRuStoreError
    {
        GENERATED_USTRUCT_BODY()

        FURuStoreRuStoreError()
    }

```

```
{
    name = "";
    description = "";
}

UPROPERTY(BlueprintReadOnly)
FString name;

UPROPERTY(BlueprintReadOnly)
FString description;
};
```

The list of possible errors:

- RuStoreNotInstalledException — RuStore is not installed on the user's device;
- RuStoreOutdatedException — RuStore is installed on the user's device but does not support application updates;
- RuStoreUserUnauthorizedException — user is not logged in on the RuStore;
- RuStoreException — basic RuStore error which gives rise to all other errors;
- RuStoreInstallException — download and installation error.

List of App Update Dependencies

- ru.rustore.sdk:core:0.1.10 — GNU Lesser General Public License v3.0;
- ru.rustore.sdk:analytics:0.1.5 — GNU Lesser General Public License v3.0;
- org.jetbrains.kotlin:kotlin-stdlib-jdk8:1.7.20 — The Apache Software License, Version 2.0;
- org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4 — The Apache Software License, Version 2.0;
- androidx.core:core-ktx:1.9.0 — The Apache Software License, Version 2.0;
- androidx.appcompat:appcompat:1.5.1 — The Apache Software License, Version 2.0;
- androidx.activity:activity:1.5.1 — The Apache Software License, Version 2.0.

List of available SDK

Platform	In-App Purchases SDK	Push Notifications SDK	General Push Notifications SDK	App Feedback and Rating SDK	App Update SDK
Kotlin	Documentation Example	Documentation Example	Documentation Example	Documentation Example	Documentation Example
Java	Documentation Example	Documentation Example	Documentation Example	Documentation Example	Documentation Example
Unity	Documents Example	Documentation		Documentation	Documentation
Flutter	Documentation Example	Documentation Example		Documentation Example	Documentation Example

Unreal Engine	Documentation Example				
Godot	Documentation Example				
React Native	Documentation Example				

Compatible SDK versions

You should use compatible SDK versions only.

RuStore Billing SDK	App Feedback and Rating SDK	App Update SDK	Push notifications SDK
3.0.0	1.0.0	1.0.0	1.0.0
4.0.0	2.0.0	2.0.0	2.0.0

When using one of the above versions, version of other used SDK must comply to the table (must not be lower)

RuStore Geo

RuStore Geo provides cartographic products and high-load services on the basis of OpenStreetMap open data.

Terms of Use of RuStore Geo Functionality on the RuStore

The following Terms of Use of RuStore Geo functionality on the RuStore (the “**Terms**”) set forth general terms and conditions of your (hereinafter — “**You**” or “**Developer**”) use of RuStore Geo services and functionality, including software for mobile or other platforms which may be provided by VK LLC (the “**Company**”), tools and any other RuStore Geo applications, services, software, components, and functions provided by the Company (hereinafter collectively referred to as “**RuStore Geo**”).

Enabling and using RuStore Geo functionality, You expressly agree to these Terms.

1. The Company operates RuStore Geo on the basis of OpenStreetMap open license. Terms of use of OpenStreetMap license are available at <http://www.openstreetmap.org/copyright/en>.
2. The Company does not make decisions concerning status of a territory (names of locations, borders, other features) on maps in OpenStreetMap database, and if the User has any questions regarding the status of any territory, relevant information is available via this link: <https://wiki.osmfoundation.org/w/images/d/d8/DisputedTerritoriesInformation.pdf>.
3. The Company hereby grants the Developer access to the use of software (including any mobile applications) provided to You by the Company (the “**Software**”) on the basis of these Terms.
4. The Company hereby represents and warrants that services of provision of access to RuStore Geo are provided 24 hours per day, seven days per week (24/7), with availability rate of not less than 99.9%.
5. The Company reserves the right to interrupt operation of RuStore Geo for necessary routine maintenance, including on business days. Such maintenance breaks shall not be considered interruptions of services and/or availability of RuStore Geo.
6. The Company shall not be responsible for unavailability of RuStore Geo if it is due to: 1) a breach of these Terms by the Developer; 2) any other Developer’s acts; 3) force majeure; 4) suspension of RuStore Geo operation requested by a governmental or municipal authority when and as provided for by applicable legislation of the Russian Federation.
7. RuStore Geo technical support is provided on the basis of a message/request sent by the Developer to RuStore Support.
8. The Company and its licensors own all intellectual property with regard to RuStore Geo and Company’s proprietary databases, including, but not limited to, RuStore Geo components and algorithms, and also access to RuStore Geo server complex, except for the intellectual property made available to the Company under OpenStreetMap license.
9. The Developer shall not have the right to modify, publish, transfer, display, participate in the transfer or sale of, create derivative works from or otherwise use the content of RuStore Geo or

any part thereof. Unless copyright law expressly provides otherwise, the Developer may not copy, disseminate, publish, display or use for commercial purposes any materials which are the property of the Company without express consent of the Company.

10. These Terms do not operate to give the Developer, whether directly or indirectly, any right to use any trademark, service mark, name or logo of the Company, except where the Developer places and uses RuStore Geo or a part of its functionality inside its Product, in which case the logo of the Company/RuStore Geo, information about the owner of RuStore Geo and <https://www.rustore.ru> link shall be always required to be added.

11. The Developer agrees and acknowledges that the sole purpose of placement of Developer's content by the Company is to enable the Developer to use RuStore Geo, and hereby grants the Company a non-exclusive, worldwide, royalty-free, fully paid, transferable, assignable and capable of sub-licensing right and license to use, copy, cache, publish, display, disseminate, alter, create derivative works from and store Developer's content within RuStore Geo. This right and license shall allow the Company to place and copy Developer's content exclusively within RuStore Geo functionality. The User hereby represents, acknowledges and agrees that they have the right to grant the above rights to the Company.

12. Except for Developer's own content, all the content displayed on RuStore Geo or available through RuStore Geo, including text, images, maps, software or source code, shall be and remain property of the Company or third parties and shall be protected by provisions of applicable law of the Russian Federation and international intellectual property laws. Logos and product names appearing on RuStore Geo or in relation therewith shall be and remain property of the Company or our licensors. The Developer may not delete any property right notices or identification labels of products from RuStore Geo.

13. Unless the Company provides information which is accessed by the Developer using RuStore Geo functionality, such information is provided, input or published by other developers and is neither controlled nor verified nor confirmed by the Company in any case. The Company has the right, but is at no obligation, to verify and track the content placed via RuStore Geo functionality, and to determine whether it conforms to these Terms and any other rules of RuStore Service which the Company may introduce from time to time. With regard to user content/information, though it does not control such content/information, the Company may use certain automatic computer systems and filters for scanning such content/messages to limit spam or other undesirable content which may be sent via RuStore Geo functionality. Such content/messages shall be sole responsibility of the posters or senders thereof.

14. RuStore Geo functionality is provided by the Company and operates "as is" and with no guarantees, whether express or implied, including, but not limited to any implied guarantee of commercial value or fitness for any specific use.

15. The Company does not guarantee that RuStore Geo and its functional capabilities will be safe or error-free, or that defects of RuStore Geo operation will be corrected.

16. The Company gives no guarantee and makes no statement as to the use or results of the use of the services or any third party websites referred to on RuStore Geo with regard to the correctness, accuracy, reliability or any other property thereof.

17. If RuStore Geo functionality is used in a Developer's Product for the travel of or for driving any vehicle, such use of RuStore Geo shall be exclusively at the user's own risk. The information being provided on RuStore Geo is not intended to replace road information (for example, provided by traffic lights, road signs or traffic police officers), and when such road information differs from the information provided on RuStore Geo, the user shall not rely on RuStore Geo data. The User shall abide by any and all laws and road traffic rules when using RuStore Geo.

18. The Developer shall be solely responsible and liable to third parties for Developer's actions related with the use of RuStore Geo functionality, including when such actions lead to infringement of rights and legitimate interests of third parties, and shall always comply with the law when using RuStore Geo.

19. When using RuStore Geo functionality, the Developer shall not have the right to:

- a. upload, send, transmit or otherwise place and/or disseminate content which is illegal, harmful, slanderous, injures morality, demonstrates (or promotes) violence and cruelty, infringes intellectual property rights, encourages hate and/or discrimination on grounds of race, ethnicity, sex, religion or social status, contains insults against any person or entity, contains elements of (or promotes) pornography, child porn, is an advertisement of (or promotes) sexual services (including under the guise of other services), explains how to make, utilize or otherwise use narcotic substances or equivalents thereof, explosives or other weapons;
- b. infringe rights of third parties, including young persons, and/or cause them harm in any form;
- c. pretend to be another person or act as a representative of an organization and/or a community without having sufficient rights to, including as an employee of the Company, as a message board moderator, as a website owner, and also use any other form or method of illegal representation of others on the Web, and also confuse users or the Company regarding properties and characteristics of any subject or object;
- d. upload, send, transmit or otherwise place and/or disseminate content without having rights to according to any law or contract;
- e. upload, send, transmit or otherwise place and/or disseminate any information of an advertisement nature that is not expressly authorized, spam (including search spam), lists or others' e-mail addresses, plans of "pyramids schemes", multi-level (network) marketing (MLM), Internet earning systems and e-mail businesses, chain e-mails, and also use Company's services to engage in such activities, or use Company's services solely to redirect users to pages of other domains;
- f. upload, send, transmit or otherwise place and/or disseminate any materials containing viruses or other computer codes, files or programs intended for disrupt, destroy or limit functionality of any computer or telecommunication equipment or programs, or to have unauthorized access, and also serial numbers for commercial software products and tools for generation thereof, logins, passwords and other means for obtaining unauthorized access to paid Internet resources, and also place links to any of the above information;
- g. collect and store others' personal data without being authorized to;
- h. disrupt normal operation of websites, RuStore service and RuStore Geo functionality;
- i. assist any action aimed at breach of any limitations and restrictions imposed by these Terms;

- j. otherwise infringe provisions of legislation of the Russian Federation, including provisions of international law.

20. The Developer shall not have the right to temporarily store (cache) any results of operation of RuStore Geo and/or information obtained from RuStore Geo, and also to alter the contents of results of queries sent to RuStore Geo provided to the Developer in response to a query.

21. The Developer shall always display all the attribution information the Company provides to the Developer for using RuStore Geo functionality (including branding, logos, and also copyright and trademark notices).

22. The Company may modify these Terms from time to time, including by way of posting a new revision of amended Terms at:

https://help.rustore.ru/rustore/for_developers/developer-documentation/RuStore_Geo, and it shall be Developer's sole responsibility to check the Terms for any changes. Amendments to the Terms made by the Company shall come into effect on the day immediately following the day of publishing the new revision of the Terms on the above website. Continued use of RuStore Geo functionality after amendment of the Terms shall constitute Developer's acceptance of and agreement with such amendments. If the Developer disagrees with any amendments to the Terms, the Developer shall cease to use (disable) RuStore Geo functionality.

23. The Developer shall have the right to use e-mail address support@rustore.ru for all matters related with these Terms.

Revision of December 20, 2022

General

You need to use an [access key](#) to access RuStore Geo products.

Supported languages

We are committed to creating fully functional maps of the whole world with maximum support for languages expected by our clients. Availability of a particular language depends on a geographic region. Main supported languages are Russian, English, and region's local language.

Request rate limitation

By default, all access keys have the same RPS limitation. You can increase or decrease the request rate for a specific key when necessary. To increase the request rate, you need to contact your client account manager at RuStore Geo.

Standard limitation of service call rate: 10 RPS (10 SPS for map display, and 10 EPS for distance matrix calculation and traveling salesman problem resolution services).

Access to services

To use RuStore Geo products you should use a valid access key which provides positive identification of the client. The access key is used in your web products and mobile products when geoservices are called, and it is also used for tracing calls to RuStore Geo products which are related with your project.

All RuStore Geo products are available at <https://geo.rustore.ru/>. To access the desired service, specify its version, name, your access key and request body in the address.

Service requests should be made as follows:

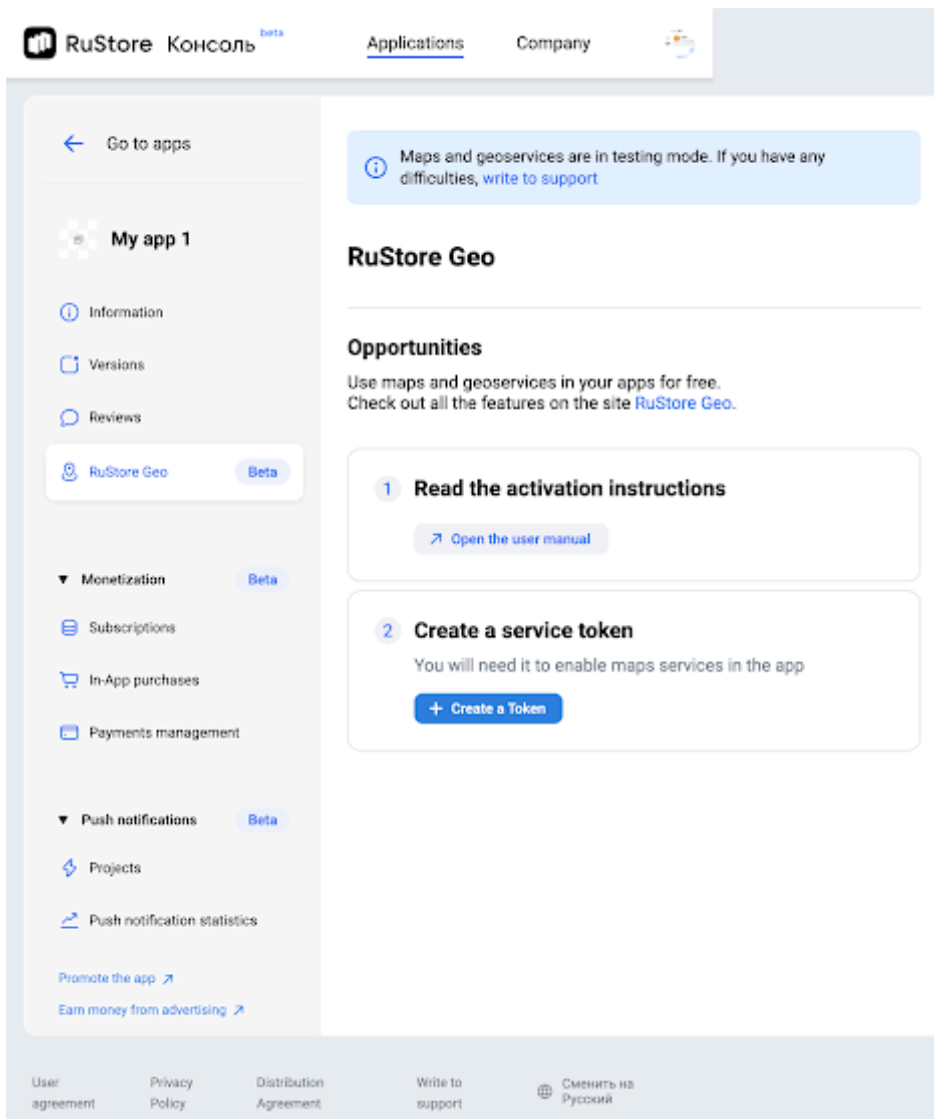
[https://geo.rustore.ru/api/\[endpoint\]?\[api_key\]&\[query\]](https://geo.rustore.ru/api/[endpoint]?[api_key]&[query])

- *endpoint* — service call point;
- *query* — request body with parameters;
- *api_key* — your access key.

To get access key:

1. Open [RuStore Console](#).
2. Go to “Applications” tab.
3. Select an application.
4. Find “Maps and geoservices” section.
5. Click “Create a token”.

Only one key for using maps can be requested per one application.



A key with validity period will be displayed. You can copy and paste it to your application. Click



Key renewal occurs automatically when an app update is released. Key's validity is limited to 90 calendar days.

Validity period and limitations applicable to a key are provided for by [Software Products Distribution Agreement](#).

If you have a problem with a key, please [contact us](#). We will do our best to resolve the problem.

Search and geocoding services

Search for Places of Interest

Search for Places of Interest allows you to search for nearby landmarks and find out more about them.

/places — service call point which is used to search for places of interest and get more information about them.

Request

Required request parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to services”	<code>api_key=fa749bace6d8a3b1..</code> <code>..</code>
q	string	Request body. Use one of the below search options: <ul style="list-style-type: none">• positioning data in lat,lon, where:<ul style="list-style-type: none">○ lat — latitude of the desired point in degrees (6 decimal places are used);○ lon — longitude of the desired point in degrees (6 decimal places are used);• text string (may contain the desired place name or address).	<code>q=rustore.ru</code> <code>q=subway</code> <code>q=55.797041,37.537830</code>

Extra request parameters

Field name	Format	Description	Example
------------	--------	-------------	---------

fields	fieldname1,fieldname2,...fieldnameN	<p>Select fields that will be passed in a response.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● name (default) — name of a place of interest; ● place_details — details on a place of interest (currently unavailable); ● address_details (default) — details of a discovered address (Postal code included) without positioning data; ● address — single-line address of a place of interest; ● pin (default) — positioning data (latitude and longitude); ● bbox — place location area; ● geometry — boundaries of a place; ● type — object type. 	<p><code>fields=pin,bbox,name</code></p>
--------	-------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------

types	string	<p>Currently unavailable.</p> <p>Searchable object types.</p> <p>By default, the search is performed on all types of places of interest.</p> <p>The guide is constantly updated. Refer to the <code>/v3/places/types</code> service to get a complete and up-to-date type reference</p>	<code>types=hotel,metro</code>
lang	2 character language code	<p>Response language in one of the available languages.</p> <p>The language of the object region is used by default.</p>	<code>lang=en</code>
location	string	<p>Relative search coordinate.</p> <p>Positioning data in lat,lon, where:</p> <ul style="list-style-type: none"> • lat — latitude of the desired point in degrees (6 decimal places are used); • lon — longitude of the desired point in degrees (6 decimal places are used); <p><i>The search results will be sorted by distance from the specified coordinate.</i></p>	<code>location=55.796743,37.537354</code>

radius	integer	<p>Search radius in meters relative to the location parameter.</p> <p><i>The location parameter is required when using the radius parameter.</i></p> <p><i>Please note that the object search may fail if it is performed within a small radius.</i></p>	<code>radius=500</code>
limit	integer	<p>Limited object number in a response. The number should be from 1 to 100.</p> <p>By default: 5</p> <p><i>Large values may cause longer response time.</i></p>	<code>limit=10</code>
isocode	2char	<p>2-character code according to ISO 3166-1 alpha-2</p>	<code>isocode=RU</code>

Response

The response should answer the request and be provided in JSON.

JSON format

Field name	Format	Description	Example
request	string	Request	<pre>"request": "/v3/places?location=55.7967432,37.5373542&a pi_key=internal_&q=prime&fields=name,place_d etails,address,address_details,pin,bbox,geom etry,type"</pre>
results	list	Discovered result	<pre>"results": [{ "address": "Russia, Moscow, Northern Administrative District, Moscow, Airport district, Leningradsky Prospekt, 72 apt4", "address_details": { "building": "72 apt4", "country": "Russia", "isocode": "RU", "locality": "Moscow", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Airport district" }, "bbox": null, "geometry": { "coordinates": [37.519963, 55.805396], "type": "Point" }, "name": "Prime", "pin": [37.519963, 55.805396], "type": "food/cafe" }]</pre>

address	string	Full object address	<pre>"address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b14"</pre>
address_details	list	Detailed object address	<pre>"address_details": { "building": "39 c80", "country": "Russia", "isocode": "RU", "locality": "Moscow", "postal_code": "125167", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Khoroshevsky" }</pre>
pin	list	Object coordinates (latitude and longitude)	<pre>"pin": [37.538851, 55.796731]</pre>
bbox	list	Object location boundaries for map positioning	<pre>"bbox": [37.538253, 55.796405, 37.539368, 55.79694]</pre>
geometry	list	Object geometry	<pre>"geometry": { "coordinates": [44.795525, 41.775552], "type": "Point" }</pre>

name	string	Object name	<code>"name": "Airport subway"</code>
type	string	Object type	<code>"type": "public_transport/station_train"</code>

address_details fields description

Field name	Format	Description	Example
country	string	Country	<code>"country": "Russia"</code>
isocode	2char	2-character code according to ISO 3166-1 alpha-2	<code>"isocode": "RU"</code>
region	string	Region	<code>"region": "Moscow"</code>
subregion	string	Subregion	<code>"subregion": "Northern Administrative District"</code>
locality	string	Locality	<code>"locality": "Moscow"</code>
sublocality	string	Sublocality or community	<code>"sublocality": "Airport"</code>
street	string	Street	<code>"street": "Leningradsky Prospekt"</code>
building	string	House, building	<code>"building": "39 bld 80"</code>

suburb	string	Suburb/neighborhood	"suburb": "Khoroshevsky"
postal_code	string	Postal code	"postal_code": "125167"

geometry field description

Field name	Format	Description	Example
type	string	Geometry type: <ul style="list-style-type: none">● Point — point;● MultiPoint — multiple points;● Linestring — line;● MultiLineString — multiple lines;● Polygon — polygon;● MultiPoligon — multiple polygons.	<pre>"type": "Polygon"</pre>
coordinates	list	Point array (longitude and latitude) that describe the object geometry	<pre>"coordinates": [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]]</pre>

Example

Request

```
https://geo.rustore.ru/api/places?api_key=<YOUR_API_KEY>&q=prime&fields=name,place_details,address,address_details,pin,bbox,geometry,type&location=55.7967432,37.5373542&limit=1
```

Geocoding service

Geocoder — service that provides direct and reverse geocoding.

Direct geocoding — process of converting addresses into geographic coordinates (latitude and longitude) that can be used to locate markers on a map or to locate a map on a display.

Reverse geocoding — process of converting geographic coordinates (latitude and longitude) to an address or part of an address (country, city, region, etc.).

All geocoding services are available throughout the world in all the supported languages.

/search — single direct or reverse geocoding point.

Request

Required request parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to services”	<code>api_key=fa749bace6d8a3b1....</code>
q	string	<p>Search request body.</p> <p>For reverse geocoding lat and lon coordinates are used, where:</p> <ul style="list-style-type: none">• lat — latitude of the desired point in degrees (6 decimal places are used);• lon — longitude of the desired point in degrees (6 decimal places are used). <p>A text string is used for direct geocoding.</p>	<code>q=Moscow Leningradsky 39b80</code> <code>q=Munich</code> <code>q=Austria</code> <code>q=55.7967432,37.5373542</code>

General request parameters

Field name	Format	Description	Example
lang	2-character language code	Response language in one of the available languages. The language of the object region is used by default.	<code>lang=en</code>
limit	unsigned integer	Limited object number in a response. The number should be from 1 to 100. By default: 5	<code>limit=10</code>
fields	fieldname1,fieldname2,...,fieldnameN	Select fields that will be passed in a response. Possible values: <ul style="list-style-type: none"> ● address_details (default) — details of a discovered address (Postal code included) without positioning data; ● address — single-line address; ● pin (default) — object positioning data (latitude and longitude); ● bbox — object location; ● geometry — object boundaries; ● type — object type. 	<code>fields=geometry,address_details</code>

admin_level	admin level	<p>Admin level response limitations.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● 1 — country; ● 2 — region; ● 3 — locality; ● 4 — street; ● 5 — house. <p>The default value is not set. The search will be performed at all admin levels.</p>	<code>admin_level=2</code>
-------------	-------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------

Extra request parameters for direct geocoding

Field name	Format	Description	Example
location	string	<p>Relative search coordinate.</p> <p>Positioning data in lat, lon, where:</p> <ul style="list-style-type: none"> ● lat — latitude of the desired point in degrees (6 decimal places are used); ● lon — longitude of the desired point in degrees (6 decimal places are used); <p><i>The search results will be sorted by distance from the specified coordinate.</i></p>	<code>location=55.7967432,37.5373542</code>
isocode	2char	2-character code according to ISO 3166-1 alpha-2	<code>isocode=RU</code>

Extra request parameters for reverse geocoding

Field name	Format	Description	Example
radius	integer	Search radius in meters relative to the location parameter. <i>Please note that the object search may fail if it is performed within a small radius.</i>	<code>radius=300</code>

Response

The response should answer the request and be provided in JSON which was designed specially for Geocoder v.3.

JSON format

Field name	Format	Description	Example
request	string	Request	<pre>"request" : "/v3/search?api_key=demo_demo_main&limit =1&q=55.796668,37.538871&fields=address_ details,address, pin, bbox, geometry, type",</pre>

results	list	Discovered result	<pre> "results": [{ "address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b14", "address_details": { "building": "39 b14", "country": "Russia", "postal_code": "125167", "isocode": "RU", "locality": "Moscow", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Khoroshevsky" }, "bbox": [37.538253, 55.796405, 37.539368, 55.79694], "geometry": { "coordinates": [[[37.538253, 55.796822], [37.539204, 55.796405], [37.539368, 55.796523], [37.539105, 55.796639], [37.539082, </pre>
---------	------	-------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

			<pre> 55.796674], [37.539052, 55.796705], [37.53901, 55.796737], [37.538958, 55.796763], [37.538899, 55.796784], [37.538835, 55.796801], [37.538772, 55.796808], [37.538714, 55.79681], [37.538416, 55.79694], [37.538253, 55.796822]]], "type": "Polygon" }, "pin": [37.538851, 55.796731], "type": "building" </pre>
--	--	--	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

			<pre> }] </pre>
address	string	Full address	<pre> "address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b14", </pre>
address_details	list	Detailed address info	<pre> "address_details": { "building": "39 b80", "country": "Russia", "isocode": "RU", "locality": "Moscow", "postal_code": "125167", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Khoroshevsky" } </pre>
pin	list	Object coordinates (latitude and longitude)	<pre> "pin": [37.538851, 55.796731] </pre>
bbox	list	Object boundaries for map positioning	<pre> "bbox": [37.538253, 55.796405, 37.539368, 55.79694] </pre>

geometry	list	Object geometry	<pre> "geometry": { "coordinates": [[[37.538253, 55.796822], [37.539204, 55.796405], [37.539368, 55.796523], [37.539105, 55.796639], [37.539082, 55.796674], [37.539052, 55.796705], [37.53901, 55.796737], [37.538958, 55.796763], [37.538899, 55.796784], [37.538835, 55.796801], [37.538772, 55.796808]]] } </pre>
----------	------	-----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

			<pre>], [37.538714, 55.79681], [37.538416, 55.79694], [37.538253, 55.796822]]], "type": "Polygon" } </pre>
type	string	Object type	"type": "building"

address_details fields description

Field name	Format	Description	Example
country	string	Country	"country": "Russia"
isocode	2char	2-character code according to ISO 3166-1 alpha-2	"isocode": "RU"
region	string	Region	"region": "Moscow"
subregion	string	Subregion	"subregion": "Northern Administrative District"
locality	string	Locality	"locality": "Moscow"

sublocality	string	Sublocality or community	<code>"sublocality": "Airport"</code>
street	string	Street	<code>"street": "Leningradsky Prospekt"</code>
building	string	House, building	<code>"building": "39 b80"</code>
suburb	string	Suburb/neighborhood	<code>"suburb": "Khoroshevsky"</code>
postal_code	string	Postal code	<code>"postal_code": "125167"</code>

geometry fields description

Field name	Format	Description	Example
type	string	Geometry type: <ul style="list-style-type: none"> ● Point — point; ● MultiPoint — multiple points; ● Linestring — line; ● MultiLineString — multiple lines; ● Polygon — polygon; ● MultiPoligon — multiple polygons. 	<code>"type": "Polygon"</code>
coordinates	list	Point array (longitude and latitude) that describe the object geometry	<pre> "coordinates": [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]] </pre>

In case of zero request result, the response will show the following:

```
{
  "results": [],
  "request": "/v3/search?limit=1&q=ascec"
}
```

Example

Direct geocoding

Request

```
https://geo.rustore.ru/api/search?api_key=<YOUR_API_KEY>q=Moscow%20Len
ingradsky%2039%20c14&fields=address_details,address, pin, bbox, geometry,
type&limit=1
```

Response

```
{
  "request": "/search?api_key=demo_demo_main&limit=1&q=Moscow
Leningradsky 39
c14&fields=address_details,address, pin, bbox, geometry, type",
  "results": [
    {
      "address": "Russia, Moscow, Northern Administrative District,
Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b14",
      "address_details": {
        "building": "39 c14",
        "country": "Russia",
        "postal_code": "125167",
```

```
"isocode": "RU",  
  
"locality": "Moscow",  
  
"region": "Moscow",  
  
"street": "Leningradsky Prospekt",  
  
"subregion": "Northern Administrative District",  
  
"suburb": "Khoroshevsky"  
  
},  
  
"bbox": [  
  
  37.538253,  
  
  55.796405,  
  
  37.539368,  
  
  55.79694  
  
],  
  
"geometry": {  
  
  "coordinates": [  
  
    [  
  
      37.538253,  
  
      55.796822  
  
    ],  
  
    [  

```

37.539204,
55.796405
],
[
37.539368,
55.796523
],
[
37.539105,
55.796639
],
[
37.539082,
55.796674
],
[
37.539052,
55.796705
],
[

37.53901,

55.796737

],

[

37.538958,

55.796763

],

[

37.538899,

55.796784

],

[

37.538835,

55.796801

],

[

37.538772,

55.796808

],

[

37.538714,

```
        55.79681
      ],
      [
        37.538416,
        55.79694
      ],
      [
        37.538253,
        55.796822
      ]
    ]
  ],
  "type": "Polygon"
},
"pin": [
  37.538851,
  55.796731
],
"type": "building"
}
```



```
]
}
```

Reverse geocoding

Request

```
https://geo.rustore.ru/api/search?api_key=<YOUR_API_KEY>q=55.796668,37.538871&fields=address_details,address,pin,bbox,geometry,type&limit=1
```

Response

```
{
  "request":
  "/v3/search?api_key=demo_demo_main&limit=1&q=55.796668,37.538871&fields=address_details,address,pin,bbox,geometry,type",
  "results": [
    {
      "address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b14",
      "address_details": {
        "building": "39 b14",
        "country": "Russia",

        "isocode": "RU",
        "locality": "Moscow",
        "region": "Moscow",
        "street": "Leningradsky Prospekt",
        "subregion": "Northern Administrative District",
        "suburb": "Khoroshevsky"
      },
      "bbox": [
        37.538253,
        55.796405,
        37.539368,
        55.79694
      ],
      "geometry": {
        "coordinates": [
          [

```

```
[
  37.538253,
  55.796822
],
[
  37.539204,
  55.796405
],
[
  37.539368,
  55.796523
],
[
  37.539105,
  55.796639
],
[
  37.539082,
  55.796674
],
[
  37.539052,
  55.796705
],
[
  37.53901,
  55.796737
],
[
  37.538958,
  55.796763
],
[
  37.538899,
  55.796784
],
[
  37.538835,
  55.796801
],
[
```

```
        37.538772,  
        55.796808  
    ],  
    [  
        37.538714,  
        55.79681  
    ],  
    [  
        37.538416,  
        55.79694  
    ],  
    [  
        37.538253,  
        55.796822  
    ]  
    ]  
    ],  
    "type": "Polygon"  
},  
"pin": [  
    37.538851,  
    55.796731  
],  
"type": "building"  
}  
]  
}
```

Address and Place Autocomplete

Prompter — service that offers suggestions to auto complete addresses and/or places when entering character by character in the search bar.

/suggest — call point of the address and place autocomplete when entering character by character.

Request

Request request parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to services”	<code>api_key=fa749bace6d8a3b1....</code>
q	string	Search request body	<code>q=Russia Moscow</code> <code>q=Moscow Leningradsky</code> <code>q=Moscow Leningradsky Prospekt</code> <code>q=Moscow Leninsky д. 1</code> <code>q=subway Airp</code>

Extra request parameters

Field name	Format	Description	Example
fields	fieldname1,fieldname2,.... fieldnameN	<p>Select fields that will be passed in a response.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● name (default) — object name; ● address_details — address details (Postal code included) without coordinates; ● address (default) — single-line address; ● ref — object ID; ● type (default) — object type. 	<code>fields=name</code>
types	string	<p>Search is performed by the following object type:</p> <ul style="list-style-type: none"> ● address (default) — address; ● place — place name. 	<code>type=address,place</code>
lang	2-character language code	<p>Response language in one of available languages.</p> <p>The language of the object region is used by default.</p>	<code>lang=en</code>

location	string	<p>Relative search coordinate.</p> <p>Positioning data in lat,lon, where:</p> <ul style="list-style-type: none"> ● lat — latitude of the desired point in degrees (6 decimal places are used); ● lon — longitude of the desired point in degrees (6 decimal places are used); <p><i>The search results will be sorted by distance from the specified coordinate.</i></p>	<p>location=55.796743,37.537354</p>
radius	integer	<p>Search radius in meters relative to the location parameter.</p> <p>The location parameter is required when using the radius parameter.</p> <p>Please note that the object search may fail if it is performed within a small radius.</p>	<p>radius=500</p>
admin_level	rank number	<p>Admin level response limitations. To be used with types=address.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● 1 — country; ● 2 — region; ● 3 — locality; ● 4 — street; ● 5 — house. <p>The default value is not set. The search will be performed at all admin levels.</p>	<p>admin_level=3</p>

limit	integer	Limited object number in a response. The number should be from 1 to 100. By default: 5 Large values may cause longer response time.	<code>limit=10</code>
-------	---------	---------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------

Response

The response should answer the request and be provided in JSON which was designed specially for Autocomplete v.3.

JSON format

Field name	Format	Description	Example
request	string	Request	<code>"request": "/v3/suggest?limit=3&q=Moscow Leningradsky 39&fields=name,address_details,address, ref,type"</code>
results	list	Result	<code>"results": [{ "address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 39 c79", "address_details": { "building": "39 c79", "country": "Russia", "isocode": "RU", "locality": "Moscow", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Khoroshevsky" }, "ref": "1000000C4D63818", "type": "building"</code>

			<pre> "name": "Leningradsky Prospekt, 39 b79", }] </pre>
address	string	Full address	<pre> "address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b79", </pre>
address_details	list	Detailed address info	<pre> "address_details": { "building": "39 b79", "country": "Russia", "isocode": "RU", "locality": "Moscow", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Khoroshevsky" } </pre>
ref	hex	Object ID Object ID is not fixed and can be changed from time to time	<pre> "ref": "1000000C4D63818" </pre>
type	string	Object type	<pre> "type": "building" </pre>
name	string	Object name	<pre> "name": "Leningradsky Prospekt, 39 b79" </pre>

address_details fields description

Field name	Format	Description	Example
------------	--------	-------------	---------

country	string	Country	"country": "Russia"
isocode	2char	2-character code according to ISO 3166-1 alpha-2	"isocode": "RU"
region	string	Region	"region": "Moscow"
subregion	string	Subregion	"subregion": "Northern Administrative District"
locality	string	Locality	"locality": "Moscow"
sublocality	string	Sublocality or community	"sublocality": "Airport"
street	string	Street	"street": "Leningradsky"
building	string	House, building	"building": "39 b80"
suburb	string	Suburb/neighborhood	"suburb": "Khoroshevsky"
postal_code	string	Postal code	"postal_code": "125167"

Example

Request

https://geo.rustore.ru/api/suggest?api_key=<YOUR_API_KEY>&limit=3&q=Moscow%20Leningradsky%2039

Response

```
{
  "request": "/suggest?limit=2&q=Moscow Leningradsky
39&api_key=demo_demo_main",
  "results": [
    {
      "address": "Russia, Moscow, Northern Administrative District,
Moscow, Khoroshevsky, Leningradsky Prospekt, 39 79",
      "name": "Leningradsky Prospekt, 39 79",
      "type": "building"
    },
    {
      "address": "Russia, Moscow, Northern Administrative District,
Moscow, Khoroshevsky, Leningradsky Prospekt, 39 b80",
      "name": "Leningradsky Prospekt, 39 b80",
      "type": "building"
    },
    {
      "address": "Russia, Moscow, Northern Administrative District,
Moscow, Khoroshevsky, Leningradsky Prospekt, 39 3",
      "name": "Alexander Gomelsky Universal Sports Hall CSKA",
      "type": "building"
    }
  ]
}
```

Map display services

Map display services are designed to help you embed an interactive and static map into your products.

An interactive map — service that allows you to embed a map on a website or app, enabling the user to change the scale, angle and position of the display, as well as to add various elements to the map using JS: pins, curves, highlighting, etc.

A static map — service that allows you to get a map image with pins placed on it as a PNG or JPG image.

Services

Call point	Description
	Interactive map
/staticmap	Static map

Interactive map

Quick start	368
1. Map	379
2. Properties and options	447
3. Tags and controls	457
4. Geography and geometry	477
5. Handlers	487
6. Sources	496
7. Events	504
8. How to use the library in React applications	508
Description of additional map objects	515

The following components are required to display an interactive map:

- display data (tiles);
- display style;
- map display code.

The display data is represented as square-shaped mosaic pieces — tiles. The whole set of tiles is a pyramid. There is one tile at the zero pyramid scale. Further, the number of tiles at each next scale is $4n$, where n is the scale number. Each tile contains geometric and attribute information.

Map display styles are a json object with a description of where to get data and how to draw certain elements.

To embed maps in your project, you need to add a layout element to the page where the map will be displayed.

How to create an interactive map

To simply display the map, you need to add JS & CSS files from SDK, as well as add the map initialization code, having specified the current API access key.

For initialization, you must specify the required style:

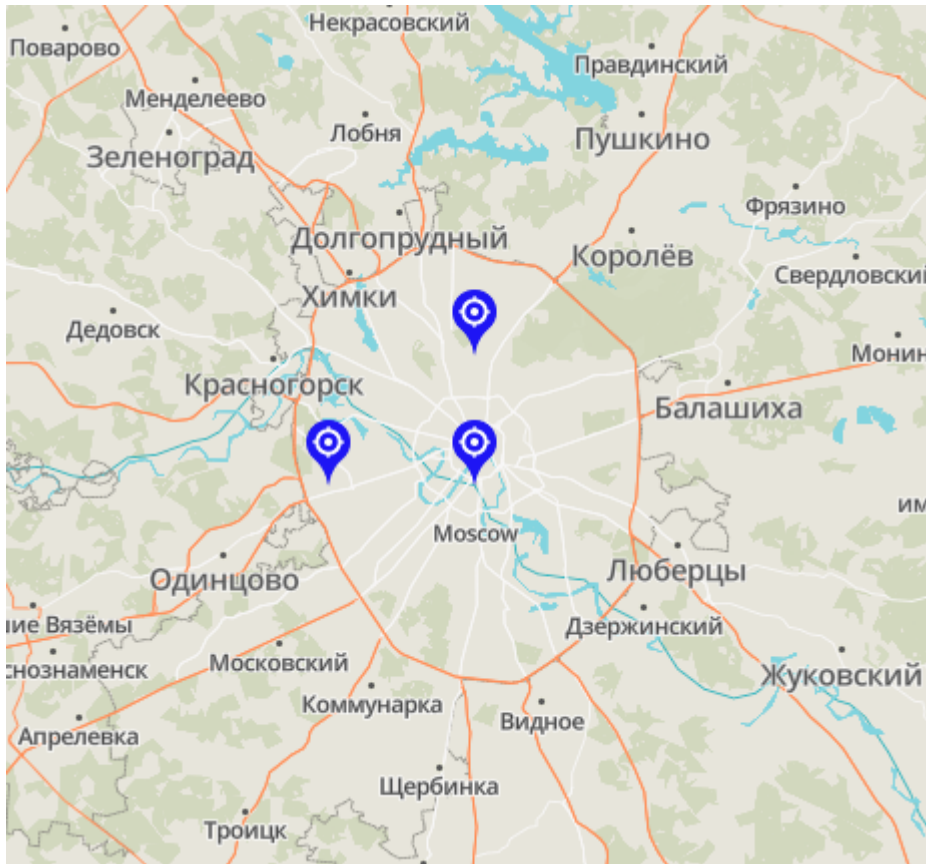
- `light_style.json` — `mmr://api/styles/light_style.json`
- `mapsme_style.json` — `mmr://api/styles/mapsme_style.json`
- `main_style.json` — `mmr://api/styles/main_style.json`

Adding objects to the map

To add your objects to the map, first you need to describe their coordinates using [GeoJson](#).

Adding a pin point

Follow the steps below to add a separate layer with icons:



Create a list of points to add to the map:

```
let userPointData = {
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [37.6165, 55.7505]
    }
  }, {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [37.4165, 55.7505]
    }
  }, {
    "type": "Feature",
    "geometry": {
```

```

        "type": "Point",
        "coordinates": [37.6165, 55.8505]
    }
}
];

```

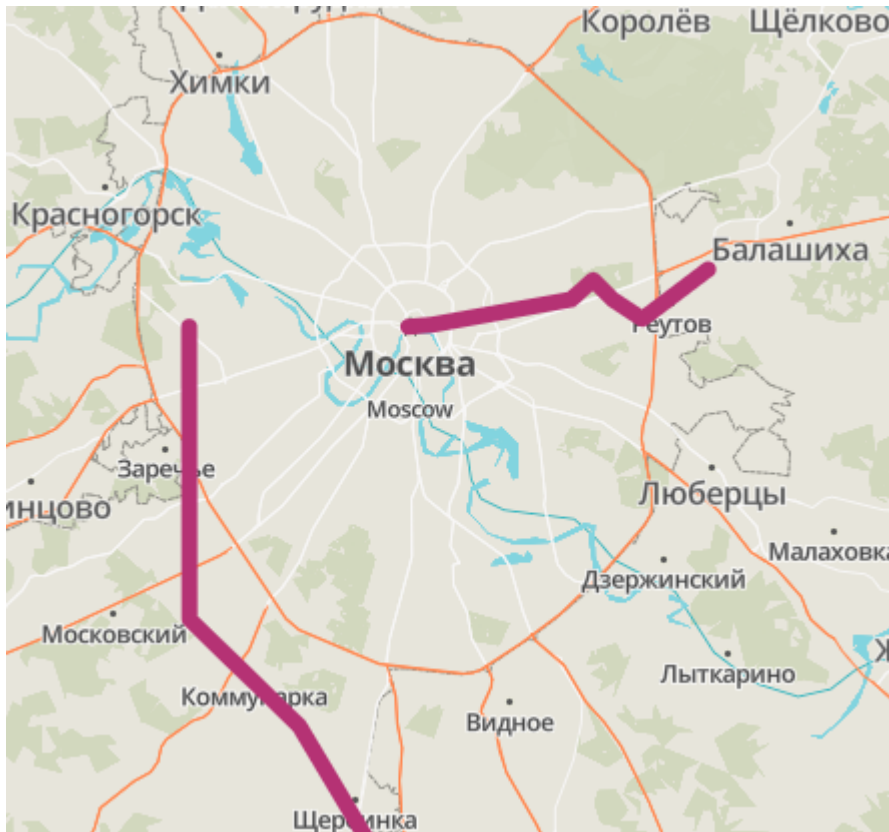
The next step is to describe the point icon and add a layer describing that the points should be drawn with this icon. This should be done upon the map loading event.

```

map.on('load', function () {
    map.loadImage(
        'https://geo.rustore.ru/api/styles/pins/blue_target.png',
        function (error, image) {
            if (error) throw error;
            map.addImage('custom_pin', image);
            map.addLayer({
                "id": "points",
                "type": "symbol",
                "source": {
                    "type": "geojson",
                    "data": userPointData
                },
                "layout": {
                    "icon-image": "custom_pin",
                    "icon-size": 1
                }
            });
        }
    );
});

```

Adding a line



Line objects are added to the map in the same way as point objects — you need to prepare the data and describe it.

Just like in the previous example, you need to prepare the data and present it in two lines, for example.

```
let userLineData = {
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "geometry": {
      "type": "LineString",
      "coordinates": [
        [37.6165, 55.7505],
        [37.6375, 55.7515],
        [37.6375, 55.7515],
        [37.665, 55.7545],
        [37.765, 55.7645],
        [37.785, 55.7745],
        [37.803, 55.7645],
        [37.83, 55.7545],
        [37.89, 55.78]
      ]
    }
  }]
}
```

```

        ]
      }
    }, {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [37.4165, 55.7505],
          [37.4166, 55.7505],
          [37.4169, 55.6],
          [37.5169, 55.545],
          [37.6169, 55.45],
          [37.5169, 55.334],
          [37.3169, 55.211]
        ]
      }
    }
  ]
};

```

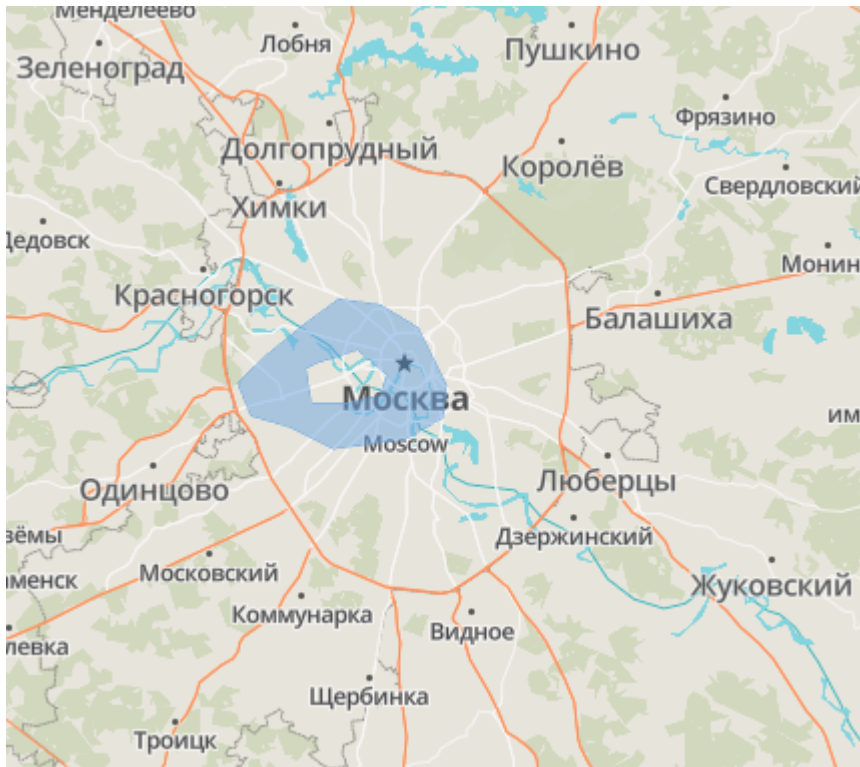
Next, add a layer description. Also add a new layer to the loading event of the main map.

```

map.on('load', function () {
  map.addLayer({
    'id': 'route',
    'type': 'line',
    'source': {
      'type': 'geojson',
      'data': userLineData
    },
    'layout': {
      'line-join': 'round',
      'line-cap': 'round'
    },
    'paint': {
      'line-color': '#AE3478',
      'line-width': 8
    }
  });
});

```

Adding a polygon



Adding a layer with polygons is similar to creating a layer with lines. Polygons can be either solid or hatched. For example, add a complex polygon to the map:

```
let userPolygonData = {
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "geometry": {
      "type": "Polygon",
      "coordinates": [
        [
          [37.427278, 55.756486],
          [37.387117, 55.734843],
          [37.405653, 55.709126],
          [37.4592, 55.70042],
          [37.518239, 55.683585],
          [37.617439, 55.690939],
          [37.669614, 55.707578],
          [37.674419, 55.73291],
          [37.635975, 55.777345],
          [37.583457, 55.795105],
          [37.525791, 55.799544],
          [37.478765, 55.780049],
          [37.427278, 55.756486]
        ]
      ]
    }
  }]
}
```

```

        ], [
            [37.495585, 55.749917],
            [37.481511, 55.745666],
            [37.48872, 55.718796],
            [37.536432, 55.718989],
            [37.582428, 55.724983],
            [37.592382, 55.740835],
            [37.551535, 55.759964],
            [37.495585, 55.749917]
        ]
    ]
}
}]
};

```

And describe how to draw it on the map:

```

map.on('load', function () {
    map.addLayer({
        'id': 'maine',
        'type': 'fill',
        'source': {
            'type': 'geojson',
            'data': userPolygonData
        },
        'layout': {},
        'paint': {
            'fill-color': '#6ea5e8',
            'fill-opacity': 0.5
        }
    });
});

```

Quick start

MMR GL JS — JavaScript library that uses WebGL to render interactive maps.

Quick start

How to connect to a html page

It is required to add `<script>` and `<link>` in `<head>` tag

```
<head>
  ...
  <script
src="https://geo.rustore.ru/sdk/js/<version>/mmr-gl.js"></script>
  <link href="https://geo.rustore.ru/sdk/js/<version>/mmr-gl.css"
rel="stylesheet">
  ...
</head>
```

It is recommended to use the latest SDK version. To get the latest SDK version, you can specify 0 or 0.0 instead of the exact version, which will load the latest up-to-date version with the number 0.x.x or 0.0.x respectively.

To initialize the map, add the following code:

```
<div id="map" style="width: 800px; height: 600px;"></div>
<script>
  mmrgl.accessToken = 'Token';
  var map = new mmrgl.Map({
    container: 'map',
    zoom: 8,
    center: [37.6165, 55.7505],
    style: 'mmr://api/styles/main_style.json',
    hash: true
  });
</script>
```

Create a token

To install `@geors/maps-sdk-js`, create a token at [github](#):

1. Go to user settings.
2. Find the Developer settings button at the bottom left.
3. Create a Personal access token with reading access to packages (read:packages).
4. Create a `.npmrc` file at the root of the project.
5. Add two lines to the created file.

```
@geors:registry=https://npm.pkg.github.com
//npm.pkg.github.com/:_authToken=ACCESS_TOKEN
```

Replace ACCESS_TOKEN to the created one.

Install a package

Navigate to the project root to package.json and use npm or yarn to install the package

```
yarn add @geors/maps-sdk-js
or
```

```
npm install @geors/maps-sdk-js
```

To initialize the map, add the following code (example in React):

```
import mmrgl from '@geors/maps-sdk-js';
import { useEffect } from 'react'

import '@geors/maps-sdk-js/dist/mmr-gl.css';

export function Map() {
  useEffect( () => {
    mmrgl.accessToken = 'accessToken';

    const map = new mmrgl.Map({
      container: 'map',
      zoom: 8,
      center: [37.6165, 55.7505],
      style: 'mmr://api/styles/main_style.json',
      hash: true,
    })

    return () => {
      if (map) map.remove();
    }
  }

  return <div id="map" style={{ width: '800px', height: '600px'}} />
}
```


1. Map

The map on the page is associated with a Map object. It provides methods and properties that allow you to modify the map from code. When an MMR GL JS map is created, it initializes it and returns a Map object.

Parameters

Name	Type	Description
container	HTMLElement string	The HTML element or block id into which MMR GL JS will render the map. The specified element must not contain dependent elements.
minZoom	number default: 0	Minimum zoom level (0-20).
maxZoom	number default: 20	Maximum zoom level (0-20).
minPitch	number default: 0	Minimum map tilt level (0-85).
maxPitch	number default: 85	Maximum map tilt level (0-85).
style	object string	Map style in JSON or style reference. This is a JSON object composed according to the style rules. <ul style="list-style-type: none">● light_style.json — mmr://api/styles/light_style.json● dark_style.json — mmr://api/styles/dark_style.json● main_style.json — mmr://api/styles/main_style.json
hash	boolean string default: false	If true, all page parameters (zoom, latitude, longitude and pitch) will be synced to the URL via parameter #.
interactive	boolean default: false	If false, the map will not respond to any control (mouse, screen, keyboard).
bearingSnap	number default: 7	Boundary, measured in degrees, determines when the map's bearing will be locked to the north.

pitchWithRotate	boolean default: true	If false, "drag and rotate" map tilt control will be disabled.
clickTolerance	number default: 3	Maximum number of pixels the user is able to point during a valid click (as opposed to dragging the mouse).
attributionControl	boolean default: true	If true, AttributionControl will be added to the map.
customAttribution	string Array<string>	A string or strings to be displayed in AttributionControl. Only possible if attributionControl = true.
failIfMajorPerformanceCaveat	boolean default: false	If true, map initialization will fail in case the map's performance is not acceptable.
preserveDrawingBuffer	boolean default: false	If true, the map canvas can be exported to a PNG image using <code>map.getCanvas().toDataURL()</code> . Set to false by default to improve performance.
antialias	boolean default: false	If true, the gl context will be created using MSAA anti-aliasing, which can be useful for anti-aliasing custom layers. Set to false by default to improve performance.
refreshExpiredTiles	boolean default: true	If false, then the map will not request tiles after they expire, according to the <code>cacheControl / expires</code> headers.
maxBounds	LngLatBoundsLike	If set, the map will be limited to the given boundaries.
scrollZoom	boolean object default: true	If true, then scroll zoom works. The object value is passed as parameters to ScrollZoomHandler.
boxZoom	boolean default: true	If true, "box zoom" interaction is enabled (see BoxZoomHandler for details)
dragRotate	boolean default: true	If true, "drag to rotate" interaction is enabled (learn more about DragRotateHandler)

dragPan	boolean object default: true	If true, "drag to pan" interaction is enabled (learn more about DragPanHandler)
keyboard	boolean default: true	If true, you can use the keyboard and keyboard shortcuts to interact with the map (learn more about KeyboardHandler)
doubleClickZoom	boolean default: true	If true, "double click to zoom" interaction is enabled (learn more about DoubleClickZoomHandler)
touchZoomRotate	boolean object default: true	If true, "pinch to rotate and zoom" interaction is enabled (learn more about TouchZoomRotateHandler)
touchPitch	boolean object default: true	If true, drag to pitch interaction is enabled (learn more about TouchPitchHandler)
trackResize	boolean default: true	If true, the map will automatically resize itself when the window changes.
center	LngLatLike default: [0,0]	Initial point on the map. If the center is not specified in the constructor parameters, MMR GL JS will search for it in the map style object. If it is also not specified in the style, it will default to [0, 0]. Note: coordinate order of longitude, latitude (as opposed to latitude, longitude) is used to match GeoJSON.
zoom	number default: 0	Initial scale level of the map. If the zoom level is not specified in the constructor parameters, MMR GL JS will search for it in the map style object. If it is also not specified in the style, it will default to 0.
bearing	number default: 0	Initial bearing (rotation) of the map, measured in degrees counterclockwise from north. If the bearing is not specified in the constructor parameters, MMR GL JS will search for it in the map style object. If it is also not specified in the style, it will default to 0.

pitch	number default: 0	Initial step (tilt) of the map, measured in degrees (0-85). If the step is not specified in the constructor parameters, MMR GL JS will search for it in the map style object. If it is also not specified in the style, it will default to 0.
bounds	LngLatBoundsLike	Initial map boundaries. If boundaries are given, it defines center and zoom.
fitBoundsOptions	object	Map#fitBounds Object parameters should only be used when fitting the original bounds above.
renderWorldCopies	boolean default: true	If true, then multiple copies will be rendered side by side beyond -180 and +180 degrees of longitude. If set to false: <ul style="list-style-type: none"> • when the map is enlarged so that no image of the world fills the entire map container, there will be empty space beyond +180 and -180 degrees of longitude. • features that intersect +180 and -180 degrees of longitude will be cut in two (one on the right edge and one on the left) at each zoom level.
maxTileCache	number default: null	Maximum number of tiles stored in the cache for this source. If this parameter is omitted, the cache size will be dynamically determined based on the current viewport.
localIdeographFontFamily	string default: sans-serif	Specifies a CSS font-family to locally redefine glyph generation in 'CJK Unified Ideographs', 'Hiragana', 'Katakana' and 'Hangul Syllables'. In these language frameworks, the font settings from the map style will be ignored, except for font-weight (light/regular/medium/bold). Set to false to enable the font settings from the map style for these glyphs.

localFontFamily	string default: false	Define a CSS font-family to redefine the generation of all glyphs. Font settings from the map style will be ignored, except for font-weight (light/regular/medium/bold). If this option is set, it overrides the setting in localIdeographFontFamily.
transformRequest	RequestTransformFunction default: null	This function is executed before the URL request is executed. It can be used to change URL, set headers, or credentials for cross-domain requests.
collectResourceTiming	boolean default: false	If true, Resource Timing API will be available for requests made by GeoJSON and Vector Tile (this information is not normally available from the main JavaScript thread). The information will be returned in ResourceTiming.
fadeDuration	number default: 300	Animation duration controls fade-in/fade-out animation for caption collisions in milliseconds. This setting does not affect the duration of style transitions during runtime or crossfading of bitmap tiles.
crossSourceCollisions	boolean default: true	If true, then characters from multiple sources may collide with each other during collision detection. If false, collision detection is performed separately for symbols in each source.
accessToken	string default: null	If specified, then this token will be used instead of the one specified in mmrgl.accessToken

locale	object default: null	A patch to apply to the default localization table for UI strings, such as control hints. The locale object correlates UI string identifiers in the namespace to translated strings in the target language; see <code>src/ui/default_locale.js</code> for all supported string identifiers for example. An object can specify all user interface strings (thus adding new translation available) or only a subset of strings (thus fixing the default translation table).
--------	-------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example

```

var map = new mmrgl.Map({
  container: 'map',
  center: [37.6165, 55.7505],
  zoom: 8,
  style: styleObject,
  hash: true,
  transformRequest: (url, resourceType)=> {
    if(resourceType === 'Source' &&
url.startsWith('http://myHost')) {
      return {
        url: url.replace('http', 'https'),
        headers: { 'my-custom-header': true},
        credentials: 'include' // Include cookies for
cross-origin requests
      }
    }
  }
});

```

Methods

Name	Description
addControl(control, position?)	<p>Adds an IControl to the map by calling control.onAdd(this)</p> <p>Options: control:iControl IControl to add position:string position on the map ('top-left' , 'top-right' , 'bottom-left' , and 'bottom-right'). The default is 'top-right'.</p>
addImage(id, image, options)	<p>Add an image to the style. This image can be displayed on the map like any other icon in the sprite, using an image ID with icon-image, background-pattern, fill-pattern, or line-pattern.</p> <p>The Map.event:error event will be called if there is not enough space in the sprite to add this image.</p> <p>Options: id:string Image ID image:HTMLImageElement ImageBitmap image data {width: number, height: number, data: (Uint8Array Uint8ClampedArray)} StyleImageInterface An image as an HTMLImageElement , ImageData , ImageBitmap , or an object with width, height, and data properties in the same format as ImageData options</p> <pre> { pixelRatio is the pixel ratio in an image to physical pixels on the screen. sdf whether to interpret the image as an SDF image. stretchX [[x1, x2], ...] if icon-text-fit is used on this image layer, this option specifies the image portion(s) that can be stretched horizontally. stretchY [[y1, y2], ...] if icon-text-fit is used on this image layer, this option specifies the image portion(s) that can be stretched vertically. content [x1, y1, x2, y2] if icon-text-fit is used on this image layer, this option specifies the image portion(s) that can be covered by the text field content. } </pre> <pre> // If the style's sprite does not already contain an image with ID 'cat', // add the image 'cat-icon.png' to the style's sprite with the ID 'cat'. map.loadImage('https://upload.wikimedia.org/wikipedia/commons/thumb/6/60/Cat_silhouette.svg/400px-Cat_silhouette.png', function(error, image) { if (error) throw error; if (!map.hasImage('cat')) map.addImage('cat', image); }); </pre> <pre> // Add a stretchable image that can be used with `icon-text-fit` // In this example, the image is 600px wide by 400px high. </pre>

```
map.loadImage('https://upload.wikimedia.org/wikipedia/commo
ns/8/89/Black_and_White_Boxed_%28bordered%29.png',
function(error, image) {
    if (error) throw error;
    if (!map.hasImage('border-image')) {
        map.addImage('border-image', image, {
            content: [16, 16, 300, 384], // place text over
left half of image, avoiding the 16px border
            stretchX: [[16, 584]], // stretch everything
horizontally except the 16px border
            stretchY: [[16, 384]], // stretch everything
vertically except the 16px border
        });
    }
});
```

addLayer(layer, beforeId?)

Adds a style layer to the map style.

The layer determines how the data from the specified source will be styled.

Options:

layer ((Object | CustomLayerInterface)) the layer to add, matches either the style specification or the CustomLayerInterface specification. Description of the Layer object:

Name	Description
id:string	Unique layer ID
type:string	Layer type (for example, fill or symbol). A list of layer types is available in the Style Specification. (It can also be customized. Learn more about CustomLayerInterface)
source:(string object)	Layer data source. A reference to a source that has already been identified using a unique ID. Refer directly to the new source using the original object (as defined in the style specification). This is required for all layer.type parameters except custom.
sourceLayer	(optional) layer name in the specified layer.source to be used for the style layer. This only applies to vector tiles and is required if layer.source is of type vector.
filter:array	(optional) An expression that specifies the conditions for the objects. Only those objects that match the filter are displayed. The style specification contains additional information about the limitations of the filter parameter and a complete list of available expressions. If no filter is provided, all objects will be displayed.
layer:object	(optional) Layer properties. The available properties depend on the layer type. A complete list of properties for each layer type is available in Style Specification. If no layout properties are set, the default values will be used.
maxzoom:number	(optional) Maximum layer zoom level. Zoom levels equal to or greater than maxzoom will hide the layer. The value can be any number between 0 and 24 (inclusive). If maxzoom is not specified, the layer will be visible at all zoom levels for which tiles are available.

minzoom:number	(optional) Minimum layer zoom level. At zoom levels less than minzoom, the layer will be hidden. The value can be any number between 0 and 24 (inclusive). If minzoom is not provided, the layer will be visible at all zoom levels as long as tiles are available.
metadata:object	(optional) Arbitrary properties that are useful for tracking with the layer, but do not affect rendering.
renderingMode:string	For custom layers only. Learn more about CustomLayerInterface.

beforeId:string new layer inserts before the existing layer by ID, causing the new layer to visually appear below the existing layer. If this argument is not specified, the layer will be added to the end of the layers array and will be displayed above all other layers.

```
// Add a circle layer with a vector source
map.addLayer({
  id: 'points-of-interest',
  source: {
    type: 'vector',
    url: 'path-to-source'
  },
  'source-layer': 'poi_label',
  type: 'circle',
  paint: {
    // Style Specification paint properties
  },
  layout: {
    // Style Specification layout properties
  }
});
```

```
// Define a source before using it to create a new layer
map.addSource('state-data', {
  type: 'geojson',
  data: 'path/to/data.geojson'
});
```

```
map.addLayer({
  id: 'states',
  // References the GeoJSON source defined above
  // and does not require a `source-layer`
  source: 'state-data',
```

```
    type: 'symbol',
    layout: {
      // Set the label content to the
      // feature's `name` property
      text-field: ['get', 'name']
    }
  });

// Add a new symbol layer before an existing layer
map.addLayer({
  id: 'states',
  // References a source that's already been defined
  source: 'state-data',
  type: 'symbol',
  layout: {
    // Set the label content to the
    // feature's `name` property
    text-field: ['get', 'name']
  }
  // Add the layer before the existing `cities` layer
}, 'cities');
```


<p><code>addSource(id, source)</code></p>	<p>Adds a source to the map style.</p> <p>Options: id:string added source ID. Should not conflict with existing sources. source:object source object that conforms to the source specification or <code>CanvasSourceOptions</code>.</p> <pre>map.addSource('my-data', { type: 'vector', url: 'path-to-source' }); map.addSource('my-data', { "type": "geojson", "data": { "type": "Feature", "geometry": { "type": "Point", "coordinates": [-77.0323, 38.9131] }, "properties": { "title": "title", "marker-symbol": "monument" } } });</pre>
<p><code>areTilesLoaded()</code></p>	<p>Returns a Boolean value indicating whether all tiles in the viewport are loaded from all style sources.</p> <pre>var tilesLoaded = map.areTilesLoaded();</pre>
<p><code>boxZoom</code></p>	<p><code>BoxZoomHandler</code> of the map, which implements zooming with a Shift-drag gesture. For more information and examples of how to use <code>boxZoom</code> refer to <code>BoxZoomHandler</code></p>

<p>cameraForBounds(bounds, options?)</p>	<p>Options: bounds:LngLatBoundsLike calculates the bounds center on the window using the highest zoom level up to <code>Map#getMaxZoom()</code> that fits on the window. <code>LngLatBounds</code> is a box that is always axis aligned with bearing 0.</p> <table border="1" data-bbox="443 344 1493 824"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>padding:number PaddingOptions</td> <td>Amount of padding in pixels, to add to the given bounds.</td> </tr> <tr> <td>bearing:number</td> <td>Desired map bearing at the end of the animation, in degrees.</td> </tr> <tr> <td>offset:PointLike</td> <td>Center of the specified boundaries relative to the map center, measured in pixels.</td> </tr> <tr> <td>maxZoom:number</td> <td>Maximum zoom level allowed when the camera moves within the specified boundaries.</td> </tr> </tbody> </table> <pre data-bbox="443 896 1401 1048"> var bbox = [[-79, 43], [-73, 45]]; var newCameraTransform = map.cameraForBounds(bbox, { padding: {top: 10, bottom:25, left: 15, right: 5} }); </pre>	Name	Description	padding:number PaddingOptions	Amount of padding in pixels, to add to the given bounds.	bearing:number	Desired map bearing at the end of the animation, in degrees.	offset:PointLike	Center of the specified boundaries relative to the map center, measured in pixels.	maxZoom:number	Maximum zoom level allowed when the camera moves within the specified boundaries.
Name	Description										
padding:number PaddingOptions	Amount of padding in pixels, to add to the given bounds.										
bearing:number	Desired map bearing at the end of the animation, in degrees.										
offset:PointLike	Center of the specified boundaries relative to the map center, measured in pixels.										
maxZoom:number	Maximum zoom level allowed when the camera moves within the specified boundaries.										
<p>doubleClickZoom</p>	<p>DoubleClickZoomHandler of a map that allows the user to zoom in by a double click. For more information and examples of using doubleClickZoom refer to DoubleClickZoomHandler</p>										
<p>dragPan</p>	<p>DragPanHandler of a map which implements dragging the map with a mouse or touch gesture. For more information and examples of how to use dragPan, see DragPanHandler</p>										
<p>dragRotate</p>	<p>DragRotateHandler of a map, which implements the map rotation when dragging with right-click or while holding down the control key. For more information and examples of how to use dragRotate, see DragRotateHandler.</p>										
<p>easeTo(options, eventData?)</p>	<p>Changes any combination of center, zoom, bearing, pitch and padding with an animated transition between previous and new values. The map will retain its current values for any details not specified in the options.</p> <p><i>The transition will happen instantly if the user has enabled the reduced motion feature, unless the options include essential: true.</i></p> <p>Options: options parameters which describe destination and transition animation. Accepts CameraOptions and AnimationOptions. eventData are additional properties added to event objects raised by this method.</p>										

fitBounds(bounds, options?, eventData?)

Moves and scales the map so that it contains the viewable area within the given geographic boundaries. This function will also reset the map bearing to 0 if the bearing is not zero. If padding is set on the map, the boundaries will match the inset.

Options:

bounds:LngLatBoundsLike centralizes these boundaries in the window and uses the highest zoom level up to the Map#getMaxZoom() that matches them in the window.

options:Object options support all properties from AnimationOptions and CameraOptions in addition to the fields below.

Name	Description
padding:number PaddingOptions	Number of padding in pixels to be added to the given boundaries.
linear:boolean	If true, then the map is transitioned using Map#easeTo. If false, then the map is navigated using Map#flyTo. These functions and AnimationOptions are used to get information about available options.
easing:function?	Animated transition easing function. See AnimationOptions
offset:PointLike	Center of the specified boundaries relative to the map center, measured in pixels.
maxZoom:number	Maximum zoom level allowed when the camera moves within the specified boundaries.

eventData represents additional properties added to event objects raised by this method.

```
var bbox = [[-79, 43], [-73, 45]];
map.fitBounds(bbox, {
  padding: {top: 10, bottom:25, left: 15, right: 5}
});
```

fitScreenCoordinates(
p0, p1, bearing,
options?,
eventData?)

Pans, rotates and scales the map to fit the box made by p0 and p1 once the map is rotated to the specified bearing. To zoom in without rotation, transmit the current bearing of the map.

Options:

p0:PointLike first point on the screen in pixel coordinates

p1:PointLike second point on the screen in pixel coordinates

bearing:number desired map bearing at the end of the animation, in degrees.

This value is ignored if the map has a non-zero step.

options:

Name	Description
padding:number Padding Options	Number of padding in pixels to be added to the given boundaries.
linear:boolean	If true, then the map is transitioned using Map#easeTo. If false, then the map is navigated using Map#flyTo. These functions and AnimationOptions are used to get information about available options.
easing:function?	Animated transition easing function. See AnimationOptions
offset:PointLike	Center of the specified boundaries relative to the map center, measured in pixels.
maxZoom:number	Maximum zoom level allowed when the camera moves within the specified boundaries.

eventData represents additional properties added to event objects raised by this method.

```
var p0 = [220, 400];  
var p1 = [500, 900];  
map.fitScreenCoordinates(p0, p1, map.getBearing(), {  
  padding: {top: 10, bottom:25, left: 15, right: 5}  
});
```

flyTo(options, eventData?)

Changes any combination of center, zoom, bearing and pitch, creating an animated transition along a curve that triggers movement. The animation seamlessly incorporates zooming and panning to preserve user's orientation even after traveling a long distance.

The animation will be skipped, and this will act equivalent to jumpTo if the user has enabled the reduced motion feature in his operating system, unless "options" include essential: true.

Name	Description
curve:number default: 1.42	"Curve" scaling that will be performed along the motion path. A high value maximizes scaling for an exaggerated animation, while a low value minimizes scaling resulting in an effect close to Map#easeTo. 1.42 is an average value chosen by user study participants discussed in van Wijk (2003). $\text{Math.pow}(6, 0.25)$ would be equivalent to the rms average speed. A value of 1 will produce a circular motion.
minZoom:number	Zero zoom level at the peak of the motion path. If options.curve is specified, then this option is ignored.
speed:number default: 1.2	Average animation speed is determined in relation to options.curve. 1.2 speed means that the map appears to be moving along its flight path 1.2 times faster. Screenful is the visible map span. It does not correspond to a fixed physical distance, but varies depending on the zoom level.
screenSpeed:number?	Average animation speed is measured in screen seconds, assuming a linear time curve. If the options.speed parameter is specified, then this parameter is ignored.
maxDuration:number?	Maximum animation duration, measured in milliseconds. If the duration exceeds the maximum value, it is reset to 0.

eventData represents additional properties added to event objects that are called by this method.

```
// fly with default options to null island
map.flyTo({center: [0, 0], zoom: 9});
// using flyTo options
map.flyTo({
  center: [0, 0],
  zoom: 9,
  speed: 0.2,
```

	<pre> curve: 1, easing(t) { return t; } }); </pre>
getBearing()	Returns the current map bearing. Bearing — compass direction, which is "up"; for example, bearing 90° rotates the map so that east is up.
getBounds()	Returns the geographic map boundaries. When the bearing or pitch is non-zero, the viewable area is not an axis-aligned rectangle, and the result is the smallest boundary enclosing the viewable area. If padding is specified on the map, then the boundaries will be relative to the insert. <pre> var bounds = map.getBounds(); </pre>
getCanvas()	Returns <canvas> map element.
getCanvasContainer())	Returns an HTML element containing the map <canvas> element. If you want to add non-GL overlays to the map, you must add them to this element. This is the element to which events are bound for map interactivity (such as pan and zoom). It will receive events from dependent elements such as <canvas> but not map controls.
getCenter()	Returns the geographic map center point. <pre> // return a LngLat object such as {lng: 0, lat: 0} var center = map.getCenter(); // access longitude and latitude values directly var {longitude, latitude} = map.getCenter(); </pre>
getContainer()	Returns an HTML element containing the map.

<p>getFeatureState(feature)</p>	<p>Returns the object state. The object state is a set of user-defined key-value pairs that are assigned to an object at run time. Features are defined via the feature.id attribute, which can be any number or string.</p> <p><i>To access object values for object styling purposes, use feature expressions.</i></p> <p>Options: feature:Object object identifier. Feature objects returned from Map#queryRenderedFeatures or event handlers can be used as object identifiers.</p> <p>options:</p> <table border="1" data-bbox="443 613 1497 1061"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>id:number string</td> <td>Unique feature identifier. Can be an integer or a string, but only supports string values when the promoteId parameter is applied to the source or either the string can be set to an integral value.</td> </tr> <tr> <td>source:string</td> <td>Vector or GeoJSON source ID for the feature.</td> </tr> <tr> <td>sourceLayer:string</td> <td>(optional) Sources of vector tiles require sourceLayer.</td> </tr> </tbody> </table> <pre data-bbox="443 1106 1437 1525"> // When the mouse moves over the `my-layer` layer, // get the feature state for the feature under the mouse map.on('mousemove', 'my-layer', function(e) { if (e.features.length > 0) { map.getFeatureState({ source: 'my-source', sourceLayer: 'my-source-layer', id: e.features[0].id }); } }); </pre>	Name	Description	id:number string	Unique feature identifier. Can be an integer or a string, but only supports string values when the promoteId parameter is applied to the source or either the string can be set to an integral value.	source:string	Vector or GeoJSON source ID for the feature.	sourceLayer:string	(optional) Sources of vector tiles require sourceLayer.
Name	Description								
id:number string	Unique feature identifier. Can be an integer or a string, but only supports string values when the promoteId parameter is applied to the source or either the string can be set to an integral value.								
source:string	Vector or GeoJSON source ID for the feature.								
sourceLayer:string	(optional) Sources of vector tiles require sourceLayer.								
<p>getFilter(layerId)</p>	<p>Returns the filter applied to the specified style layer.</p>								
<p>getFreeCameraOptions()</p>	<p>Returns position and orientation of the camera object.</p>								
<p>getLayer(id)</p>	<p>Returns the layer with the specified map style ID.</p> <p>Options: id:string layer ID to be obtained.</p> <pre data-bbox="443 1939 1289 1973"> var stateDataLayer = map.getLayer('state-data'); </pre>								

getLayoutProperty(layerId, name)	<p>Returns the layout property value on the specified style layer.</p> <p>Options: layerId:string layer ID from which the layout property will be obtained. name:string is the layer property name to be obtained.</p>
getLight()	Returns the light object value .
getMaxBounds()	<p>Returns the maximum geographic boundaries the map is bound to, or null if they are not set.</p> <pre>var maxBounds = map.getMaxBounds();</pre>
getMaxPitch()	Returns the maximum allowable tilt of the map.
getMaxZoom()	<p>Returns the maximum acceptable zoom level for the map.</p> <pre>var maxZoom = map.getMaxZoom();</pre>
getMinPitch()	Returns the minimum allowable map step.
getMinZoom()	<p>Returns the minimum acceptable zoom level for the map.</p> <pre>var minZoom = map.getMinZoom();</pre>
getPadding()	Returns the current padding applied around the map viewport.
getPaintProperty(layerId, name)	<p>name:string paint name to be obtained.</p> <p>Returns the paint property value on the specified style layer.</p> <p>Options: layerId:string layer ID from which the paint property will be derived. name:string paint property name to be obtained.</p>
getPitch()	Returns the current tilt of the map.

<p>getRenderWorldCopies()</p>	<p>Returns the state of renderWorldCopies. If this is true, then multiple copies of the world will be rendered side by side beyond -180 and +180 degrees of longitude. If set to false:</p> <ul style="list-style-type: none"> • when the map is zoomed so that no image of the world fills the entire map container, there will be empty space beyond +180 and -180 degrees of longitude. • features that intersect +180 and -180 degrees of longitude will be divided in two (one on the right edge of the map and one on the left) at each zoom level. <pre>var worldCopiesRendered = map.getRenderWorldCopies();</pre>
<p>getSource(id)</p>	<p>Returns the source with the specified map-style ID.</p> <p>This method is often used to update a source with instance elements for the appropriate source type defined in sources. For example, to set the data for a GeoJSON source, or update the URL and coordinates of an image source.</p> <p>Options: id:string source ID to be obtained.</p> <pre>var sourceObject = map.getSource('points');</pre>
<p>getStyle()</p>	<p>Returns the map style object, a JSON object that can be used to recreate the map style.</p> <pre>map.on('load', function() { var styleJson = map.getStyle(); });</pre>
<p>getTerrain()</p>	<p>Returns the terrain specification, or null if terrain is not defined on the map.</p>
<p>getZoom()</p>	<p>Returns the current map zoom level.</p> <pre>map.getZoom();</pre>
<p>hasControl(control)</p>	<p>Verifies whether there is a control on the map.</p> <p>Options: control:IControl IControl for navigation control</p> <pre>// Define a new navigation control. var navigation = new mmrgl.NavigationControl(); // Add zoom and rotation controls to the map. map.addControl(navigation); // Check that the navigation control exists on the map. const added = map.hasControl(navigation);</pre>

	<pre>// added === true</pre>
hasImage(id)	<p>Checks if an image with a specific ID exists in the style. Validates both the images in the original style sprite and any images added using the Map#addImage method.</p> <p>Options: id:string image ID.</p> <pre>// Check if an image with the ID 'cat' exists in // the style's sprite. var catIconExists = map.hasImage('cat');</pre>
isMoving()	<p>Returns true if the map is moved, scaled, rotated, or tilted via a camera animation or user gesture.</p> <pre>var isMoving = map.isMoving();</pre>
isRotating()	<p>Returns true if the map is rotating via a camera animation or user gesture.</p> <pre>map.isRotating();</pre>
isSourceLoaded(id)	<p>Returns a Boolean value indicating whether the source has been loaded. Returns true if the source with the given map style ID has no network requests, otherwise returns false.</p> <p>Options: id:string source ID to be verified</p> <pre>var sourceLoaded = map.isSourceLoaded('bathymetry-data');</pre>
isStyleLoaded()	<p>Returns a Boolean value indicating whether the map style is fully loaded.</p> <pre>var styleLoadStatus = map.isStyleLoaded();</pre>
isZooming()	<p>Returns true if the map is zoomed via a camera animation or user gesture.</p> <pre>var isZooming = map.isZooming();</pre>

<p>jumpTo(options, eventData?)</p>	<p>Changes any combination of center, zoom, bearing and pitch without an animated transition. The map will retain its current values for any details not specified in the options.</p> <p>Options: options:CameraOptions option object eventData are additional properties added to event objects called by this method.</p> <pre>// jump to coordinates at current zoom map.jumpTo({center: [0, 0]}); // jump with zoom, pitch, and bearing options map.jumpTo({ center: [0, 0], zoom: 8, pitch: 45, bearing: 90 });</pre> <p>Copy</p>
<p>keyboard</p>	<p>KeyboardHandler of the map, which allows the user to zoom, rotate, and pan the map using keyboard shortcuts. For more information refer to KeyboardHandler</p>
<p>listImages()</p>	<p>Returns a string array containing the IDs of all images currently available on the map. This includes both images from the original style sprite and any images added using the Map#addImage method.</p> <pre>var allImages = map.listImages();</pre>
<p>loaded()</p>	<p>Returns a Boolean value indicating whether the map is fully loaded. Returns false if the style has not yet been fully loaded, or if there have been changes to the sources or style that are not yet fully loaded.</p>
<p>loadImage(url, callback)</p>	<p>Uploads an image from an external URL to use with Map#addImage. External domains must support CORS.</p> <p>Options: url:string The image file URL. The image file must be in png, webp or jpg. callback:Function wait callback(error, data). Called when an image is loaded, or with an error argument in case of an error.</p> <pre>// Load an image from an external URL. map.loadImage('http://placekitten.com/50/50', function(error, image) { if (error) throw error;</pre>

	<pre> // Add the loaded image to the style's sprite with the ID 'kitten'. map.addImage('kitten', image); }); </pre>
<p>moveLayer(id, beforeId?)</p>	<p>Moves the layer to a different z-position.</p> <p>Options:</p> <p>id:string layer ID to be moved.</p> <p>beforeId:string layer ID before which the new layer will be inserted. When viewing the map, the layer ID will appear below the BeforeID layer. If the BeforeID parameter is omitted, then the layer will be added to the end of the layers array and appear above all other layers on the map.</p> <pre> // Move a layer with ID 'polygon' before the layer with ID 'country-label'. The `polygon` layer will appear beneath the `country-label` layer on the map. map.moveLayer('polygon', 'country-label'); </pre>
<p>off(type, listener)</p>	<p>Removes the event listener previously added with Map#on.</p> <p>Options:</p> <p>type:string The event type previously used to set the listener.</p> <p>listener:Function is the function previously set as the listener.</p>
<p>off(type, layerId, listener)</p>	<p>Removes the event listener for specific layer events previously added with Map#on.</p> <p>Options:</p> <p>type:string The event type previously used to set the listener.</p> <p>layerId:string The layer ID previously used to set the listener.</p> <p>listener:Function is the function previously set as the listener.</p>

on(type, layerId,
listener)

Adds a listener for events of the specified type, optionally limited to objects in the specified style layer.

Options:

type:string event type for the listener. Events compatible with the optional layerId parameter are called when the cursor enters the visible portion of the specified layer outside that layer or outside the map canvas.

Event	Compatible with layerId
mousedown	yes
mouseup	yes
mouseover	yes
mouseout	yes
mousemove	yes
mouseenter	yes (required)
mouseleave	yes (required)
click	yes
dblclick	yes
contextmenu	yes
touchstart	yes
touchend	yes
touchcancel	yes
wheel	
resize	
remove	
touchmove	
movestart	
move	
moveend	

	dragstart	
	drag	
	dragend	
	zoomstart	
	zoom	
	zoomend	
	rotatestart	
	rotate	
	rotateend	
	pitchstart	
	pitch	
	pitchend	
	boxzoomstart	
	boxzoomend	
	boxzoomcancel	
	webglcontextlost	
	webglcontextrestored	
	load	
	render	
	idle	
	error	
	data	
	styledata	
	sourcedata	

dataloading	
styledataloading	
sourcedataloading	
styleimagemissing	

layerId:string (optional) style layer ID. The event will only be called if its location is within a visible object in that layer. The event will have features which contain an array of relevant objects. If layerId is not specified, the event will not have features. Note that many event types are incompatible with layerId.

listener:Function The function to be triggered by an event.

```
// Set an event listener that will fire
// when the map has finished loading
map.on('load', function() {
  // Once the map has finished loading,
  // add a new layer
  map.addLayer({
    id: 'points-of-interest',
    source: {
      type: 'vector',
      url: 'path-to-source'
    },
    'source-layer': 'poi_label',
    type: 'circle',
    paint: {
      // Style Specification paint properties
    },
    layout: {
      // Style Specification layout properties
    }
  });
});

// Set an event listener that will fire
// when a feature on the countries layer of the map is
// clicked
map.on('click', 'countries', function(e) {
  new mmrgl.Popup()
    .setLngLat(e.lngLat)
    .setHTML(`Country name:
    ${e.features[0].properties.name}`)
    .addTo(map);
});
```

<p>once(type, listener)</p>	<p>Adds a listener that will only be called once for the specified event type.</p> <p>Options: layerId:string (optional) style layer ID. The event will only be called if its location is within a visible object in that layer. The event will have a features property containing an array of relevant objects. If layerId is not specified, the event will not have a features property. Note that many event types are incompatible with the optional layerId parameter. listener:Function. This function is called when the event is triggered.</p>
<p>once(type, layerId, listener)</p>	<p>Adds a listener that will be called only once for the specified event type that occurs within objects in the specified style layer.</p> <p>Options: type:string event type for the listener; available values: 'mousedown' , 'mouseup' , 'click' , 'dblclick' , 'mousemove' , 'mouseenter' , 'mouseleave' , 'mouseover' , 'mouseout' , 'contextmenu' , 'touchstart' , 'touchend' or 'touchcancel' . The mouseenter and mouseover events are called when the cursor enters the visible part of the specified layer outside that layer or outside the map canvas. The mouseleave and mouseout events are called when the cursor leaves the visible part of the specified layer or leaves the map canvas. layerId:string style layer ID. Only events within the visible object in the current layer will trigger the listener. The event will have a features property containing an array of relevant objects. listener:Function — this function is called when the event is triggered.</p>
<p>panBy(offset, options?, eventData?)</p>	<p>Map panning at a given offset.</p> <p>Options: offset:PointLike x and y coordinates to pan the map. options:AnimationOptions options object eventData are additional properties added to event objects triggered by this method.</p>
<p>panTo(Inglat, options?, eventData?)</p>	<p>Moves the map to the specified location via an animated transition.</p> <p>Options: Inglat:LngLatLike a place to pan the map. options:AnimationOptions an options object eventData are additional properties added to event objects triggered by this method.</p> <pre>map.panTo([-74, 38]); // Specify that the panTo animation should last 5000 milliseconds. map.panTo([-74, 38], {duration: 5000});</pre>

project(Inglat)	<p>Returns a point that represents the pixel coordinates, relative to the map container, corresponding to the specified geographic location.</p> <p>When the map is tilted and Inglat is completely behind the camera, there are no pixel coordinates corresponding to that location. In this case, x and y components of the returned point will be Number.MAX_VALUE.</p> <p>Inglat:LngLatLike project geographic location.</p> <pre>var coordinate = [-122.420679, 37.772537]; var point = map.project(coordinate);</pre>
-----------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

queryRenderedFeatures(geometry? options?)

Returns an array of GeoJSON Feature objects representing the visible features that match the request parameters.

Options:

geometry: PointLike | Array<PointLike> Area geometry in pixels, either a single point or bottom left and top right points describing the bounding box where the origin is at the top left. When omitting this parameter (i.e. calling Map#queryRenderedFeatures with zero arguments or with only options argument) it would be equivalent to passing a bounding box that spans the entire map viewport.

options:

Name	Description
layers:Array<string>	A style layer IDs array to validate the request. Only objects within those layers will be returned. If this parameter is not defined, then all layers will be checked.
filter:Array	Filter to limit request results.
validate:boolean	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.

It returns:

Array<Object>: Objects array GeoJSON.

The property value of each returned object contains the properties of its original object. For GeoJSON sources, only string and numeric property values are supported (i.e., null, Array, and Object values are not supported).

Each object includes top-level layer, source, and source layer properties. The layer property is an object that represents the style layer to which the object belongs. The layout and paint properties on this object contain values that are fully verified for the given zoom level and object.

Only features that are currently being visualized are enabled. Some features will not be enabled, such as:

- objects from layers with "none" visibility property.
- objects from layers whose zoom range excludes the current zoom level.
- character features that were hidden due to text or icon collisions.

Objects from all other layers are included, as well as objects that may not have a visible result.

The topmost rendered object appears first in the returned array, and subsequent objects are sorted in descending z-order. Objects that are rendered multiple times (due to anti-meridian wrapping at low zoom levels) are only returned once. Hence, objects come from vector tiles or GeoJSON data that is converted to tiles, objects' geometry can be split or duplicated across tile boundaries. Thus, objects can appear multiple times in the request results. Let's have a look at the

below example: there is a highway passing through the request bounding box. The request result will include those parts of the highway that are located inside the map tiles covering the bounding box, even if the highway extends into other tiles, and the part of the highway inside each map tile will be returned as a separate object. Similarly, a point feature near a tile boundary can appear in multiple tiles due to tile buffering.

```
// Find all features at a point
var features = map.queryRenderedFeatures(
  [20, 35],
  { layers: ['my-layer-name'] }
);

// Find all features within a static bounding box
var features = map.queryRenderedFeatures(
  [[10, 20], [30, 50]],
  { layers: ['my-layer-name'] }
);

// Find all features within a bounding box around a point
var width = 10;
var height = 20;
var features = map.queryRenderedFeatures([
  [point.x - width / 2, point.y - height / 2],
  [point.x + width / 2, point.y + height / 2]
], { layers: ['my-layer-name'] });

// Query all rendered features from a single layer
var features = map.queryRenderedFeatures({ layers:
['my-layer-name'] });
```

<p>querySourceFeatures (sourceId, parameters?)</p>	<p>Returns an array of GeoJSON Feature objects that represent the features within the specified vector tile or GeoJSON source that comply with the requested parameters.</p> <p>Options: sourceId:string vector tiles or GeoJSON source IDs for a request. parameters:</p> <table border="1" data-bbox="443 448 1492 963"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>sourceLayer:string</td> <td>Source layer request name. For vector tile sources, this parameter is mandatory. For GeoJSON sources, it is ignored.</td> </tr> <tr> <td>filter:Array</td> <td>Request results imitation filter.</td> </tr> <tr> <td>validate:boolean</td> <td>You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.</td> </tr> </tbody> </table> <p>Returns: Array<Object>: GeoJSON objects array</p> <p>Unlike Map#queryRenderedFeatures, this function returns all features that match the requested parameters, whether they are rendered in the current style (i.e., visible) or not. The request domain includes all currently loaded vector tiles and source GeoJSON tiles: this function does not check for tiles beyond the currently visible viewport.</p> <p>Hence, objects come from vector tiles or GeoJSON data that is converted to tiles, objects' geometry can be split or duplicated across tile boundaries. Thus, objects can appear multiple times in the request results. Let's have a look at the below example: there is a highway passing through the request bounding box. The request result will include those parts of the highway that are located inside the map tiles covering the bounding box, even if the highway extends into other tiles, and the part of the highway inside each map tile will be returned as a separate object. Similarly, a point feature near a tile boundary can appear in multiple tiles due to tile buffering.</p> <pre data-bbox="443 1608 1485 1765"> // Find all features in one source layer in a vector source var features = map.querySourceFeatures('your-source-id', { sourceLayer: 'your-source-layer' }); </pre>	Name	Description	sourceLayer:string	Source layer request name. For vector tile sources, this parameter is mandatory. For GeoJSON sources, it is ignored.	filter:Array	Request results imitation filter.	validate:boolean	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.
Name	Description								
sourceLayer:string	Source layer request name. For vector tile sources, this parameter is mandatory. For GeoJSON sources, it is ignored.								
filter:Array	Request results imitation filter.								
validate:boolean	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.								
<p>remove()</p>	<p>Clean up and free all internal resources associated with the map. This includes DOM elements, event bindings, web workers, and WebGL resources.</p> <p>Use this method when you're done using the map and want to make sure it's no longer consuming browser resources. After that, you should not call any other methods on the map.</p>								

removeControl(control)

Removes control elements from the map.

Options:

control: **IControl** an IControl element to be deleted

```
// Define a new navigation control.  
var navigation = new mmrgl.NavigationControl();  
// Add zoom and rotation controls to the map.  
map.addControl(navigation);  
// Remove zoom and rotation controls from the map.  
map.removeControl(navigation);
```

removeFeatureState(
feature, key)

Removes the object state, returning it to its default behavior. If only feature.source is specified, it will remove the state for all features from that source. If feature.id is also specified, then it will remove all keys for the state of that object. If a key is also specified, it removes only that key from the object state. Features are identified by feature.id, which can be represented as any number or string.

Options:

feature:Object object identifier. Feature objects returned from Map#queryRenderedFeatures or event handlers can be used as object IDs.

Name	Description
id:number string	Unique feature identifier. Can be an integer or a string, but only supports string values when the promoteId parameter is applied to the source or either the string can be set to an integral value.
source:string	Vector or GeoJSON source ID for the feature.
sourceLayer:string	(optional) Sources of vector tiles require sourceLayer.

key:string (optional) a key in the object state to be reset.

```
// Reset the entire state object for all features  
// in the `my-source` source  
map.removeFeatureState({  
  source: 'my-source'  
});
```

```
// When the mouse leaves the `my-layer` layer,  
// reset the entire state object for the  
// feature under the mouse  
map.on('mouseleave', 'my-layer', function(e) {  
  map.removeFeatureState({  
    source: 'my-source',  
    sourceLayer: 'my-source-layer',  
    id: e.features[0].id  
  });  
});
```

```
// When the mouse leaves the `my-layer` layer,  
// reset only the `hover` key-value pair in the  
// state for the feature under the mouse  
map.on('mouseleave', 'my-layer', function(e) {  
  map.removeFeatureState({  
    source: 'my-source',  
    sourceLayer: 'my-source-layer',  
    id: e.features[0].id
```

	<pre> }, 'hover'); });</pre>
removeImage(id)	<p>Removes an image from the style. This can be an image from the original style sprite, or any images added via the Map#addImage method.</p> <p>Options: id:string image ID.</p> <pre>// If an image with the ID 'cat' exists in // the style's sprite, remove it. if (map.hasImage('cat')) map.removeImage('cat');</pre>
removeLayer(id)	<p>Removes a layer with the given ID from the map style. If no such layer exists, an error event is called.</p> <p>Options: id:string layer ID to be removed.</p> <pre>// If a layer with ID 'state-data' exists, remove it. if (map.getLayer('state-data')) map.removeLayer('state-data');</pre>
removeSource(id)	<p>Removes a source from the map style.</p> <p>Options: id:string source ID to be removed.</p> <pre>map.removeSource('bathymetry-data');</pre>
repaint	<p>Gets and sets a Boolean value indicating whether the map will continuously be redrawn. This information is useful for performance analysis.</p>
resetNorth(options?, eventData?)	<p>Rotates the map so that north is up (bearing 0°), with an animated transition.</p> <p>Options: options:AnimationOptions options object eventData additional properties added to event objects that are called by this method.</p>
resetNorthPitch(options?, eventData?)	<p>Rotates and tilts the map so that north is up (azimuth 0°) and pitch is 0°, with an animated transition.</p> <p>Options: options:AnimationOptions options object eventData additional properties added to event objects that are called by this method.</p>

<p>resize(eventData?)</p>	<p>Resizes the map to fit its container element.</p> <p>Checks whether the map container size has changed and updates the map in this case. This method should be called after the map's container is changed programmatically, or when the map is displayed after being initially hidden with CSS.</p> <p>Options: eventData: Object additional properties passed to the movestart , move , resize , and moveend events triggered by map size updates. This can be useful for differentiating the event source (for example, user-triggered or programmatically-triggered events).</p> <pre>// Resize the map when the map container is shown // after being initially hidden with CSS. var mapDiv = document.getElementById('map'); if (mapDiv.style.visibility === true) map.resize();</pre>
<p>rotateTo(bearing, options?, eventData?)</p>	<p>Rotates the map to the specified bearing with an animated transition. Bearing is the compass direction "up"; for example, a bearing of 90° sets the map so that east is up.</p> <p>Options: bearing: number desired bearing options: AnimationOptions options object eventData additional properties added to event objects that are called by this method.</p>
<p>scrollZoom</p>	<p>The map's ScrollZoomHandler that implements zooming in and out using the scroll wheel or trackpad. For more information, see ScrollZoomHandler</p>
<p>setBearing(bearing, eventData?)</p>	<p>Sets the map bearing (rotation). Bearing is the compass direction s "up"; for example, a bearing of 90° sets the map so that east is up.</p> <p>Equivalent to jumpTo({bearing: bearing}).</p> <p>Options: bearing: number desired bearing eventData additional properties added to event objects that are called by this method.</p> <pre>// rotate the map to 90 degrees map.setBearing(90);</pre>
<p>setCenter(center, eventData?)</p>	<p>Sets the geographic center point of the map. Equivalent to jumpTo({center: center}).</p> <p>Options: center: LngLatLike central point</p>

eventData additional properties added to event objects that are called by this method.

```
map.setCenter([-74, 38]);
```

setFeatureState(feature, state)

Sets the state of an object. The state of an object is a set of user-defined key-value pairs that are assigned to an object. When using this method, the state is concatenated with any existing key-value pairs in the object's state. Features are identified through the feature.id attribute, which can be any number or string.

This method can only be used with sources that have a feature.id attribute. The Feature.id attribute can be defined in three ways:

- for vector or GeoJSON sources, including the ID attribute in the original data file.
- for vector sources or GeoJSON sources, use the promoteId option when defining the source.
- for GeoJSON sources, use the generateId option to automatically assign an ID based on the feature's index in the source data. If you change feature data using map.getSource('some id').setData(..), you may need to reapply the state with the updated ID values.

Options:

feature: Object object ID. Feature objects returned from Map#queryRenderedFeatures or event handlers can be used as object identifiers.

Name	Description
id:number string	Unique feature identifier. Can be an integer or a string, but only supports string values when the promoteId parameter is applied to the source or either the string can be set to an integral value.
source:string	Vector or GeoJSON source ID for the feature.
sourceLayer:string	(optional) Sources of vector tiles require sourceLayer.

state: Object a set of key-value pairs. Values must be valid JSON types.

```
// When the mouse moves over the `my-layer` layer, update
// the feature state for the feature under the mouse
map.on('mousemove', 'my-layer', function(e) {
  if (e.features.length > 0) {
    map.setFeatureState({
      source: 'my-source',
      sourceLayer: 'my-source-layer',
      id: e.features[0].id,
    }, {
      hover: true
    });
  }
});
```

```

    }
  });

```

`setFilter(layerId, filter, options = {})`

Sets the filter for the specified style layer.

The filters define which objects in a style layer are rendered from its source. Any object for which the filter expression evaluates to true will be displayed on the map. Those that are false will be hidden.

Use `setFilter` to display a subset of the original data.

To clear the filter, pass null or undefined as the second parameter.

Options:

layerId:string layer ID to which the filter will be applied.

filter: Array | null | undefined a filter that matches the style specification's filter definition. If set to null or undefined, the function removes any existing filter from the layer.

options:object default {} options object

Name	Description
validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.

```

// display only features with the 'name' property 'USA'
map.setFilter('my-layer', ['==', ['get', 'name'], 'USA']);

```

```

// display only features with five or more
'available-spots'
map.setFilter('bike-docks', ['>=', ['get',
'available-spots'], 5]);

```

```

// remove the filter for the 'bike-docks' style layer
map.setFilter('bike-docks', null);

```

`setFreeCameraOptions(options, eventData?)`

FreeCameraOptions provides direct access to the underlying camera object. For backwards compatibility, a state set using this API must also be represented using CameraOptions. Parameters are clamped into the valid range or discarded as invalid if the conversion to pitch and bearing representation is ambiguous. For example, an orientation might be invalid if it causes the camera to be upside down, the quaternion to have zero length, or the stride to exceed the maximum stride limit.

Options:

options:FreeCameraOptions options object

	<p>eventData additional properties added to event objects that are called by this method.</p>				
<p>setLayerZoomRange(layerId, minzoom, maxzoom)</p>	<p>Sets the zoom level for the specified style layer. The zoom level includes the minimum zoom level and the maximum zoom level at which the layer will be displayed.</p> <p><i>For style layers that use vector sources, the style layers cannot be drawn at zoom levels below the source layer's minimum zoom level because the data does not exist at those zoom levels. If the source layer minimum zoom level is higher than the value defined in the style layer, the style layer will not display at all zoom levels within the zoom range.</i></p> <p>Options: layerId:string layer ID to which the zoom level will be applied. minzoom:number minimum zoom level (0-24). maxzoom:number maximum zoom level (0-24).</p> <pre>map.setLayerZoomRange('my-layer', 2, 5);</pre>				
<p>setLayoutProperty(layerId, name, value, options = {})</p>	<p>Sets the layer property value on the specified style layer.</p> <p>Options: layerId:string layer ID to which the zoom level will be applied. name:string property name value:any property value. Must be of a type that matches this property, as defined in the style specification. options:object default {} options object</p> <table border="1" data-bbox="443 1205 1497 1473"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>validate:boolean default: true</td> <td>You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.</td> </tr> </tbody> </table> <pre>map.setLayoutProperty('my-layer', 'visibility', 'none');</pre>	Name	Description	validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.
Name	Description				
validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.				

<p>setLight(light, options = {})</p>	<p>Sets any combination of light values.</p> <p>Options: light:LightSpecification lighting properties. Must match the style specification. options:object default {} options object</p> <table border="1" data-bbox="443 376 1497 651"> <thead> <tr> <th data-bbox="443 376 727 450">Name</th> <th data-bbox="732 376 1497 450">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="443 456 727 651">validate:boolean default: true</td> <td data-bbox="732 456 1497 651">You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.</td> </tr> </tbody> </table> <pre data-bbox="443 689 1497 763">var layerVisibility = map.getLayoutProperty('my-layer', 'visibility');</pre>	Name	Description	validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.
Name	Description				
validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.				
<p>setMaxBounds(bounds)</p>	<p>Sets or clears the geographic boundaries of the map.</p> <p>Pan and zoom operations are limited by the above values. In case pan or zoom display areas beyond these boundaries, the map instead displays the position and zoom level as close to the operation request as possible while remaining within these boundaries.</p> <p>Options: bounds:LngLatBoundsLike null undefined maximum limits to set. If null or undefined, this function removes the map's maximum bounds.</p> <pre data-bbox="443 1205 1497 1514">// Define bounds that conform to the `LngLatBoundsLike` object. var bounds = [[-74.04728, 40.68392], // [west, south] [-73.91058, 40.87764] // [east, north]]; // Set the map's max bounds. map.setMaxBounds(bounds);</pre>				
<p>setMaxPitch(maxPitch)</p>	<p>Sets or clears the maximum map tilt. If the current map tilt is higher than the new maximum value, the map will tilt towards the new maximum value.</p> <p>Options: maxPitch:number null undefined maximum tilt to set. If null or undefined, the function removes the current maximum slope (sets it to 85)</p>				

<p>setMaxZoom(maxZoom)</p>	<p>Sets or clears the maximum map zoom level.</p> <p>If the current map zoom level is higher than the new maximum value, the map will zoom in to the new maximum value.</p> <p>Options: maxZoom: number null undefined is the maximum zoom level to set. If null or undefined, the function removes the current maximum scale (sets it to 22).</p> <pre>map.setMaxZoom(18.75);</pre>
<p>setMinPitch(minPitch)</p>	<p>Sets or clears the minimum map pitch.</p> <p>If the current chart pitch is below the new low value, the chart will tilt towards the new low value.</p> <p>Options: minPitch: number null undefined is the minimum pitch to set (0-85). If null or undefined, the function removes the current minimum pitch (i.e. sets it to 0)</p>
<p>setMinZoom(minZoom)</p>	<p>Sets or clears the minimum map zoom level. If the current map zoom level is below the new minimum value, the map zooms in to the new minimum value. It is not always possible to zoom out and reach the set minZoom. Other factors, such as map height, may limit zoom. For example, if the map is 512 pixels high, then zooming below zoom 0 will be impossible, no matter what minZoom is set to.</p> <p>Options: minZoom: number null undefined is the minimum zoom level to set (-2 - 24). If null or undefined, the function removes the current minimum scale (i.e., sets it to -2).</p> <pre>map.setMinZoom(12.25);</pre>
<p>setPadding(padding, eventData?)</p>	<p>Sets the padding in pixels around the screen.</p> <p>Equivalent to <code>jumpTo({padding: padding})</code>.</p> <p>Options: padding: PaddingOptions desired padding. Format: { left: number, right: number, top: number, bottom: number } eventData are additional properties added to event objects called by this method.</p> <pre>// Sets a left padding of 300px, and a top padding of 50px map.setPadding({ left: 300, top: 50 });</pre>

<p>setPaintProperty(layerId, name, value, options = {})</p>	<p>Sets the value of the paint property on the specified style layer.</p> <p>Options: layerId:string layer ID to which the zoom level will be applied. name:string property name value:any property value. Must be of a type that matches this property, as defined in the style specification. options:object default {} options object</p> <table border="1" data-bbox="443 510 1493 790"> <thead> <tr> <th data-bbox="443 510 703 589">Name</th> <th data-bbox="703 510 1493 589">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="443 589 703 790">validate:boolean default: true</td> <td data-bbox="703 589 1493 790">You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.</td> </tr> </tbody> </table> <pre data-bbox="443 824 1493 869">map.setPaintProperty('my-layer', 'fill-color', '#faafee');</pre>	Name	Description	validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.
Name	Description				
validate:boolean default: true	You should check whether it matches the style specification. Validation disabling stands for performance optimization that should only be used if you have previously validated the values to be passed to this function.				
<p>setPitch(pitch, eventData?)</p>	<p>Sets the tilt height of the map. Equivalent to jumpTo({pitch: pitch}).</p> <p>Options: pitch:number pitch is measured in degrees (0-60°). eventData additional properties to be added to event objects of events called by this method.</p>				
<p>setRenderWorldCopies(renderWorldCopies)</p>	<p>Sets the state of renderWorldCopies.</p> <p>Options: renderWorldCopies:boolean If true, then multiple copies will be rendered side by side beyond -180 and +180 degrees of longitude. If set to false:</p> <ul data-bbox="491 1339 1493 1507" style="list-style-type: none"> • when the map is enlarged so that no image of the world fills the entire map container, there will be empty space beyond +180 and -180 degrees of longitude. • features that intersect +180 and -180 degrees of longitude will be cut in two (one on the right edge and one on the left) at each zoom level. <p>undefined is regarded as true, null — as false.</p> <pre data-bbox="443 1574 991 1619">map.setRenderWorldCopies(true);</pre>				

<p>setStyle(style, options?)</p>	<p>Updates the map style object with the new value.</p> <p>When using this option, if a style is already set and options.diff is set to true, the map renderer will attempt to compare the given style with the current map state and make only those changes necessary to make the map style match the desired state. Changes to sprites (images used for icons and patterns) and glyphs (fonts for label text) cannot be distinguished. If the sprites or fonts used in the current style and this style differ in any way, the map renderer will force a full update, removing the current style and creating this one from scratch.</p> <p>Options: style:StyleSpecification string null a JSON object corresponding to the scheme described in the style specification, or a URL to such a JSON object. options:Object options object</p> <table border="1" data-bbox="443 680 1497 1225"> <thead> <tr> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>diff:boolean default: true</td> <td>If false, force a full update, removing the current style and creating this one instead of trying to update it based on the diff.</td> </tr> <tr> <td>localIdeographFontFamily:string default: 'sans-serif'</td> <td>Defines a CSS font family for locally overriding glyph generation within CJK Unified Ideographs, Hiragana, Katakana, and Hangul Syllables. Within this framework, the font settings from the map style will be ignored, except for font-weight keywords (light/regular/medium/bold). Set to false to enable the font settings from the map style for these glyph ranges. It forces a full update.</td> </tr> </tbody> </table> <p><code>map.setStyle("path-to-style");</code></p>	Name	Description	diff:boolean default: true	If false, force a full update, removing the current style and creating this one instead of trying to update it based on the diff.	localIdeographFontFamily:string default: 'sans-serif'	Defines a CSS font family for locally overriding glyph generation within CJK Unified Ideographs, Hiragana, Katakana, and Hangul Syllables. Within this framework, the font settings from the map style will be ignored, except for font-weight keywords (light/regular/medium/bold). Set to false to enable the font settings from the map style for these glyph ranges. It forces a full update.
Name	Description						
diff:boolean default: true	If false, force a full update, removing the current style and creating this one instead of trying to update it based on the diff.						
localIdeographFontFamily:string default: 'sans-serif'	Defines a CSS font family for locally overriding glyph generation within CJK Unified Ideographs, Hiragana, Katakana, and Hangul Syllables. Within this framework, the font settings from the map style will be ignored, except for font-weight keywords (light/regular/medium/bold). Set to false to enable the font settings from the map style for these glyph ranges. It forces a full update.						
<p>setTerrain(terrain)</p>	<p>Sets the style's terrain property.</p> <p>Options: terrain:TerrainSpecification properties to be set. Must match the style specification. If set to null or undefined, the function removes the terrain.</p> <pre>map.addSource('mmr-dem', { 'type': 'raster-dem', 'url': 'path-to-source', 'tileSize': 512, 'maxzoom': 14 }); // add the DEM source as a terrain layer with exaggerated height map.setTerrain({ 'source': 'mmr-dem', 'exaggeration': 1.5 });</pre>						

setZoom(zoom, eventData?)	<p>Sets the map zoom level. Equivalent to <code>jumpTo({zoom: zoom})</code>.</p> <p>Options: zoom: number zoom level to set (0-20). eventData: Object additional properties that will be added to event objects called by this method.</p> <pre>// Zoom to the zoom level 5 without an animated transition map.setZoom(5);</pre>
showCollisionBoxes	<p>Gets and sets a Boolean value that specifies whether the map will display margins around all symbols in the data source, indicating which symbols have been rendered or which have been hidden due to collisions. This information is useful for debugging.</p>
showPadding	<p>Gets and sets a Boolean value indicating whether the map will render padding.</p>
showTerrainWireframe	<p>Gets and sets a Boolean value indicating whether the map will display wireframe on top of the rendered terrain. Useful for debugging.</p> <p>The wireframe is always red and is drawn only when the terrain is active.</p> <pre>map.showTerrainWireframe = true;</pre>
showTileBoundaries	<p>Gets and sets a Boolean value indicating whether the map will draw an outline around each tile and the tile ID. These tile borders are useful for debugging.</p> <p>The uncompressed file size of the first vector source is displayed in the upper left corner of each tile next to the tile ID.</p> <pre>map.showTileBoundaries = true;</pre>
snapToNorth(options?, eventData?)	<p>Snaps the map so that north is up (bearing 0°) if the current bearing is close enough to it (i.e. within the bearingSnap threshold).</p> <p>Options: options: AnimationOptions eventData additional properties added to event objects that are called by this method.</p>
stop()	<p>Stops any animated transition.</p>
touchPitch	<p>TouchPitchHandler of the map, which allows the user to pass the map using touch gestures. For more information, see TouchPitchHandler.</p>
touchZoomRotate	<p>TouchZoomRotateHandler of the map, which allows the user to zoom or rotate the map using touch gestures. For more information, see TouchZoomRotateHandler</p>

triggerRepaint()	<p>Starts rendering one frame. Use this method with custom layers to redraw the map when the layer changes. Only one frame will be rendered, even if this method is called multiple times before visualizing the next frame.</p> <pre>map.triggerRepaint();</pre>
unproject(point)	<p>Returns LngLat representing the geographic coordinates corresponding to the specified pixel coordinates. If the horizon is visible and the specified pixel is above the horizon, returns LngLat which corresponds to the point on the horizon closest to the point.</p> <p>Options: point:PointLike pixel coordinates for unproject</p> <pre>map.on('click', function(e) { // When the map is clicked, get the geographic coordinate. var coordinate = map.unproject(e.point); });</pre>
updateImage(id, image)	<p>Update an existing style image. This image can be displayed on the map like any other icon in a style sprite using an image ID with an icon-image, background-pattern, fill-pattern, or line -pattern).</p> <p>Options: id:string image ID image:(HTMLImageElement ImageBitmap ImageData {width: number, height: number, data: (Uint8ClampedArray Uint8ClampedArray)} StyleImageInterface) an image as HTMLImageElement, ImageData, ImageBitmap, or an object with width, height, and data properties in the same format as ImageData.</p> <pre>// If an image with the ID 'cat' already exists in the style's sprite, // replace that image with a new image, 'other-cat-icon.png'. if (map.hasImage('cat')) map.updateImage('cat', './other-cat-icon.png');</pre>
version	MMR GL JS version as specified in package.json
zoomIn(options?, eventData?)	<p>Increases the map scale by 1.</p> <p>Options: options:AnimationOptions options object eventData additional properties added to event objects that are called by this method.</p>

	<pre>// zoom the map in one level with a custom animation duration map.zoomIn({duration: 1000});</pre>
<p>zoomOut(options?, eventData?)</p>	<p>Reduces the map scale by 1.</p> <p>Options: options:AnimationOptions options object eventData additional properties added to event objects that are called by this method.</p> <pre>// zoom the map out one level with a custom animation offset map.zoomOut({offset: [80, 60]});</pre>
<p>zoomTo(zoom, options, eventData?)</p>	<p>Zooming the map to a given zoom level with an animated transition.</p> <p>Options: zoom:number zoom level for the transition. options:AnimationOptions options object eventData additional properties added to event objects that are called by this method.</p> <pre>// Zoom to the zoom level 5 without an animated transition map.zoomTo(5); // Zoom to the zoom level 8 with an animated transition map.zoomTo(8, { duration: 2000, offset: [100, 50] });</pre>

Events

Name	Description
boxzoomcancel	<p>Triggered when the user cancels the "box zoom" interaction or when the bounding box does not meet the minimum size threshold. See BoxZoomHandler</p> <pre data-bbox="472 517 1337 779">// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // the user cancels a "box zoom" interaction. map.on('boxzoomcancel', function() { console.log('A boxzoomcancel event occurred.');</pre>
boxzoomend	<p>Triggered when "box zoom" is finished. See BoxZoomHandler</p> <pre data-bbox="472 994 1286 1256">// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just after a "box zoom" interaction ends. map.on('boxzoomend', function() { console.log('A boxzoomend event occurred.');</pre>
boxzoomstart	<p>Triggered when «box zoom» is started. See BoxZoomHandler</p> <pre data-bbox="472 1464 1321 1727">// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just before a "box zoom" interaction starts. map.on('boxzoomstart', function() { console.log('A boxzoomstart event occurred.');</pre>

click	<p>Triggered when pressed.</p> <p><i>This event is compatible with the optional layerId parameter. If layerId is included as the second argument to Map#on, the event listener will only trigger when the point contains the visible part of the specified layer.</i></p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('click', function(e) { console.log('A click event has occurred at ' + e.lngLat); }); // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener for a specific layer map.on('click', 'poi-label', function(e) { console.log('A click event has occurred on a visible portion of the poi-label layer at ' + e.lngLat); });</pre>
contextmenu	<p>Triggered when the right mouse button or context menu key is pressed inside the map.</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when the right mouse button is // pressed within the map. map.on('contextmenu', function() { console.log('A contextmenu event occurred.');</pre>

data	<p>Triggered when loading or changing any map data</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when map data loads or changes. map.on('data', function() { console.log('A data event occurred.');</pre>
dataloading	<p>Triggered when any map data (style, source, tile, etc.) starts loading or changing asynchronously. All dataloadings are followed by a data or error event.</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when any map data begins loading // or changing asynchronously. map.on('dataloading', function() { console.log('A dataloading event occurred.');</pre>

<p>dblclick</p>	<p>Triggered when pressed twice at the same point on the map in quick succession.</p> <p><i>This event is compatible with the optional layerId parameter. If layerId is included as the second argument to Map#on, the event listener will only trigger when the double-clicked point contains the visible part of the specified layer.</i></p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('dblclick', function(e) { console.log('A dblclick event has occurred at ' + e.lngLat); }); // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener for a specific layer map.on('dblclick', 'poi-label', function(e) { console.log('A dblclick event has occurred on a visible portion of the poi-label layer at ' + e.lngLat); }); </pre>
<p>drag</p>	<p>Triggered repeatedly during a drag-to-pan interaction. See DragPanHandler</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // repeatedly during a "drag to pan" interaction. map.on('drag', function() { console.log('A drag event occurred.');</pre>

dragend	<p>Triggered when a drag-to-pan interaction ends. See DragPanHandler</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when a "drag to pan" interaction ends. map.on('dragend', function() { console.log('A dragend event occurred.');</pre>
dragstart	<p>Triggered when a drag-to-pan interaction begins. See DragPanHandler</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when a "drag to pan" interaction starts. map.on('dragstart', function() { console.log('A dragstart event occurred.');</pre>
error	<p>Triggered when an error occurs. This is the main GL JS error reporting mechanism. We use event instead of throw to better adapt to asynchronous operations. If the listener is not bound to the error event, then the error will be printed to the console.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when an error occurs. map.on('error', function() { console.log('A error event occurred.');</pre>

idle	<p>Triggered after the last frame drawn before the map enters the "idle" state:</p> <ul style="list-style-type: none">• no camera transitions occur• all currently requested tiles have been loaded• all fade/transition animations have been completed <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just before the map enters an "idle" state. map.on('idle', function() { console.log('A idle event occurred.');</pre>
load	<p>Triggered immediately after all the necessary resources were loaded and the map was visually rendered for the first time.</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when the map has finished loading. map.on('load', function() { console.log('A load event occurred.');</pre>

<p>mousedown</p>	<p>Triggered when you click inside the map.</p> <p><i>This event is compatible with the optional layerId parameter. If layerId is included as the second argument to Map#on, the event listener will only trigger when the cursor is clicked inside the visible part of the specified layer.</i></p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('mousedown', function() { console.log('A mousedown event has occurred.');</pre> <pre>});</pre> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener for a specific layer map.on('mousedown', 'poi-label', function() { console.log('A mousedown event has occurred on a visible portion of the poi-label layer.');</pre> <pre>});</pre>
<p>mouseenter</p>	<p>Triggered when the cursor enters the visible part of the specified layer beyond that layer or beyond the map canvas.</p> <p><i>This event can only be listened for when Map#on includes three arguments, where the second argument specifies the desired layer.</i></p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('mouseenter', 'water', function() { console.log('A mouseenter event occurred on a visible portion of the water layer.');</pre> <pre>});</pre>

<p>mouseleave</p>	<p>Triggered when the cursor leaves the visible part of the specified layer or leaves the map canvas.</p> <p><i>This event can only be listened for when Map#on includes three arguments, where the second argument specifies the desired layer.</i></p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when the pointing device leaves // a visible portion of the specified layer. map.on('mouseleave', 'water', function() { console.log('A mouseleave event occurred.');</pre>
<p>mousemove</p>	<p>Triggered when the cursor moves while the cursor is inside the map. When moving the cursor around the map, the event will trigger every time the cursor changes its position on the map.</p> <p><i>This event is compatible with the optional layerId parameter. If layerId is included as the second argument to Map#on, the event listener will only trigger when the cursor is inside the visible part of the specified layer.</i></p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('mousemove', function() { console.log('A mousemove event has occurred.');</pre> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener for a specific layer map.on('mousemove', 'poi-label', function() { console.log('A mousemove event has occurred on a visible portion of the poi-label layer.');</pre>

<p>mouseout</p>	<p>Triggered when the mouse cursor leaves the map canvas.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when the pointing device leave's // the map's canvas. map.on('mouseout', function() { console.log('A mouseout event occurred.');</pre>
<p>mouseover</p>	<p>Triggered when the mouse cursor moves inside the map. When moving the cursor over a web page containing a map, the event will trigger every time it enters the map or any child elements.</p> <p><i>This event is compatible with the optional layerId parameter. If layerId is included as the second argument to Map#on, the event listener will only trigger when the cursor moves inside the visible part of the specified layer.</i></p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('mouseover', function() { console.log('A mouseover event has occurred.');</pre> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener for a specific layer map.on('mouseover', 'poi-label', function() { console.log('A mouseover event has occurred on a visible portion of the poi-label layer.');</pre>

<p>mouseup</p>	<p>Triggered when the mouse cursor is "released" within the map.</p> <p><i>This event is compatible with the optional layerId parameter. If layerId is included as the second argument to Map#on, the event listener will only trigger when the cursor is released while inside the visible part of the specified layer.</i></p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener map.on('mouseup', function() { console.log('A mouseup event has occurred.');</pre> <pre> }); // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener for a specific layer map.on('mouseup', 'poi-label', function() { console.log('A mouseup event has occurred on a visible portion of the poi-label layer.');</pre> <pre> }); </pre>
<p>move</p>	<p>Triggered repeatedly during an animated transition from one view to another as a result of user interaction or methods such as Map#flyTo.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // repeatedly during an animated transition. map.on('move', function() { console.log('A move event occurred.');</pre> <pre> }); </pre>

<p>moveend</p>	<p>Triggered immediately after the map completes a transition from one view to another, either as a result of user interaction or methods such as Map#jumpTo.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just after the map completes a transition. map.on('moveend', function() { console.log('A moveend event occurred.');</pre>
<p>movestart</p>	<p>Triggered just before the map starts transitioning from one view to another, either as a result of user interaction or methods such as Map#jumpTo.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just before the map begins a transition // from one view to another. map.on('movestart', function() { console.log('A movestart` event occurred.');</pre>
<p>pitch</p>	<p>Triggered repeatedly during map pitch (tilt) animation between one state and another as a result of user interaction or methods such as Map#flyTo.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // repeatedly during a pitch (tilt) transition. map.on('pitch', function() { console.log('A pitch event occurred.');</pre>

pitchend	<p>Triggered immediately after the pitch (tilt) of the map ends as a result of user interaction or methods such as Map#flyTo.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just after a pitch (tilt) transition ends. map.on('pitchend', function() { console.log('A pitchend event occurred.');</pre>
pitchstart	<p>Triggered whenever the pitch (tilt) of the map starts to change as a result of user interaction or methods such as Map#flyTo.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just before a pitch (tilt) transition starts. map.on('pitchstart', function() { console.log('A pitchstart event occurred.');</pre>
remove	<p>Triggered immediately after the map is removed using Map.event:remove.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just after the map is removed. map.on('remove', function() { console.log('A remove event occurred.');</pre>

render	<p>Triggered whenever the map is drawn on the screen and cause:</p> <ul style="list-style-type: none"> • changing map position, zoom, pitch or bearing • changing map style • changing the GeoJSON source • loading a vector tile, GeoJSON file, glyph or sprite <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // whenever the map is drawn to the screen. map.on('render', function() { console.log('A render event occurred.');</pre>
resize	<p>Triggered immediately after the map is resized.</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // immediately after the map has been resized. map.on('resize', function() { console.log('A resize event occurred.');</pre>
rotate	<p>Triggered repeatedly during a drag to rotate interaction. See DragRotateHandler</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // repeatedly during "drag to rotate" interaction. map.on('rotate', function() { console.log('A rotate event occurred.');</pre>

rotateend	<p>Triggered when a drag-to-rotate interaction ends. See DragRotateHandler</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just after a "drag to rotate" interaction ends. map.on('rotateend', function() { console.log('A rotateend event occurred.');</pre>
rotatestart	<p>Triggered when a drag-to-rotate interaction starts. See DragRotateHandler</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just before a "drag to rotate" interaction starts. map.on('rotatestart', function() { console.log('A rotatestart event occurred.');</pre>
sourcedata	<p>Triggered when loading or changing one of the map sources, including when loading or changing a tile belonging to the source.</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when one of the map's sources loads or changes. map.on('sourcedata', function() { console.log('A sourcedata event occurred.');</pre>

sourcedataloading	<p>Triggered when one of the map sources starts loading or changing asynchronously. All sourcedataloading events are followed by sourcedata or error event. For more information, see MapDataEvent</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // map's sources begin loading or // changing asynchronously. map.on('sourcedataloading', function() { console.log('A sourcedataloading event occurred.');</pre>
styledata	<p>Triggered when the map style is loaded or changed. For more information, see MapDataEvent</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when the map's style loads or changes. map.on('styledata', function() { console.log('A styledata event occurred.');</pre>
styledataloading	<p>Triggered when the map style starts to load or change asynchronously. All styledataloading events are followed by a styledata or error event. For more information, see MapDataEvent</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // map's style begins loading or // changing asynchronously. map.on('styledataloading', function() { console.log('A styledataloading event occurred.');</pre>

styleimagemissing	<p>Triggered when there is no icon or template required by the style. A missing image can be added using Map#addImage to prevent the image from being skipped. This event can be used to dynamically generate icons and templates.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // an icon or pattern is missing. map.on('styleimagemissing', function() { console.log('A styleimagemissing event occurred.');</pre>
touchcancel	<p>Triggered when a touchcancel event occurs on the map.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when a touchcancel event occurs within the map. map.on('touchcancel', function() { console.log('A touchcancel event occurred.');</pre>
touchend	<p>Triggered when a touchend event occurs on the map.</p> <pre> // Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when a touchstart event occurs within the map. map.on('touchstart', function() { console.log('A touchstart event occurred.');</pre>

touchmove	<p>Triggered when a touchmove event occurs on the map.</p> <pre> // Initialize the map var map = new mmr.gl.Map({ // map options }); // Set an event listener that fires // when a touchmove event occurs within the map. map.on('touchmove', function() { console.log('A touchmove event occurred.');</pre>
touchstart	<p>Triggered when a touchstart event occurs on the map.</p> <p>Parameters: data:MapMouseEvent</p> <pre> // Initialize the map var map = new mmr.gl.Map({ // map options }); // Set an event listener that fires // when a touchstart event occurs within the map. map.on('touchstart', function() { console.log('A touchstart event occurred.');</pre>
webglcontextlost	<p>Triggered when the WebGL context is lost.</p> <pre> // Initialize the map var map = new mmr.gl.Map({ // map options }); // Set an event listener that fires // when the WebGL context is lost. map.on('webglcontextlost', function() { console.log('A webglcontextlost event occurred.');</pre>

webglcontextrestored	<p>Triggered when the WebGL context is restored.</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when the WebGL context is restored. map.on('webglcontextrestored', function() { console.log('A webglcontextrestored event occurred.');</pre>
wheel	<p>Triggered when the map is scrolled by the wheel</p> <p>Parameters: data:MapMouseEvent</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // when a wheel event occurs within the map. map.on('wheel', function() { console.log('A wheel event occurred.');</pre>
zoom	<p>Triggered repeatedly during an animated transition from one zoom level to another as a result of user interaction or methods such as Map#flyTo</p> <p>Parameters: data:(MapMouseEvent MapTouchEvent)</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // repeatedly during a zoom transition. map.on('zoom', function() { console.log('A zoom event occurred.');</pre>

zoomend	<p>Triggered immediately after the map has completed transitioning from one zoom level to another, either as a result of user interaction or methods such as Map#flyTo.</p> <p>Parameters: data:(MapMouseEvent MapTouchEvent)</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just after a zoom transition finishes. map.on('zoomend', function() { console.log('A zoomend event occurred.');</pre>
zoomstart	<p>Triggered just before the map starts transitioning from one zoom level to another, either as a result of user interaction or methods such as Map#flyTo.</p> <p>Parameters: data:(MapMouseEvent MapTouchEvent)</p> <pre>// Initialize the map var map = new mmrgl.Map({ // map options }); // Set an event listener that fires // just before a zoom transition starts. map.on('zoomstart', function() { console.log('A zoomstart event occurred.');</pre>

2. Properties and options

accessToken:string

Map access token — the event type.

src/index.js

Example:

```
mmrgl.accessToken = myAccessToken
```

baseApiUrl:string

Initial API URL, which is used to get tiles, styles, sprites and glyphs

src/index.js

Example:

```
mmrgl.baseApiUrl = 'https://geo.rustore.ru/api';
```

workerCount:number

Number of web workers per page with GL JS maps. Half the number of cores (limited to 6) is set by default. Make sure you have set this parameter before initializing the map.

src/index.js

Example:

```
mmrgl.workerCount = 2;
```

maxParallelImageRequests:number

The maximum number of images (raster tiles, sprites, icons) to load in parallel. Affects performance in raster maps. 16 by default.

src/index.js

Example:

```
mmrgl.maxParallelImageRequests = 10;
```

supported:function

Checks browser support for GL JS

src/index.js

Example:

```
if (!mmrgl.supported()) {  
  alert('Your browser does not support MMR GL');  
}
```

version:string

Current build version MMR GL

src/index.js

Example:

```
console.log(mmrgl.version);  
//> 1.2.3;
```

setRTLTextPlugin:function

Installs a plugin for RTL support. Required for Arabic and Hebrew languages support.
src/index.js

Parameters:

pluginURL:string — path to RTL plugin.

callback:function(error:object) — this function is called in case of an error.

lazy:boolean — lazy loading.

Example:

```
mmrgl.setRTLTextPlugin(pluginUrl, function(error) {  
  if (error) {  
    console.log('something was wrong', error);  
  } else {  
    console.log('rtl-text-plugin loaded successfully');  
  }  
}, true);
```

getRTLTextPluginStatus:function

It is used to get RTL plug-in status. Plug-in status can be: unavailable (not requested or removed), loading, loaded or error. An error occurs if the status is loaded and the plug-in is requested again.

src/index.js

Example:

```
const pluginStatus = mmrgl.getRTLTextPluginStatus();
```

clearStorage:function

Clears cacheStorage which can store tiles cache

src/index.js

Parameters:

callback:function(error:object) — path to RTL plug-in.

Example:

```
mmrgl.clearStorage()
```

AnimationOptions (parameter group)

Animation options (which are used in the following methods: `Map#panBy`, `Map#easeTo`) keep control over duration and animation smoothing function (easing function). All options are optional.

`src/ui/camera.js`

Parameters:

duration:number — animation duration (in milliseconds).

easing:function — function that takes a time value in the range 0..1 and returns a number, where 0 is the initial state and 1 is the final state.

offset:PointLike — center shift from the real map center at the end of the animation.

animate:boolean — if false, animations will be disabled.

essential:boolean — if true, then the animation is considered essential and will not be affected by prefers-reduced-motion (reduced motion is preferred).

CameraOptions (parameter group)

Camera options (used in the methods: `Map#jumpTo`, `Map#easeTo` and `Map#flyTo`) keep control over location (starting position), zoom (scaling), bearing and pitch of the camera. All options are optional.

`src/ui/camera.js`

Parameters:

center:LngLatLike — map center.

zoom:number — map (scale) distance.

bearing:number — bearing is compass direction, which is "up". For example, bearing: 90 sets the map so that east is up.

pitch:number — desired map tilt in degrees. The pitch angle is measured with respect to the horizon, measured in degrees(0 - 60°). For example, pitch: 0 makes it appear to be looking straight down at the map, while pitch: 60 tilts the user's perspective toward the horizon. Increased tilt value is often used to display 3D objects.

around:LngLatLike — if zoom is set, around specifies the point around which the zoom is centered.

padding:PaddingOptions — applied to each side of the viewport to shift the vanishing point (relevant when the map is tilted).

PaddingOptions (parameter group)

Padding options (used in methods: `Map#fitBounds`, `Map#fitScreenCoordinates` and `Map#setPadding`). Adjust these options to set the amount of pixel padding added to the edges of the map. All object properties must be non-negative integers.

`src/ui/camera.js`

Parameters:

top:number — padding in pixels on the top of the map.

bottom:number — padding in pixels on the bottom of the map.

left:number — padding in pixels on the left of the map.

right:number — padding in pixels on the right of the map.

Examples:

```
var bbox = [[-79, 43], [-73, 45]];
map.fitBounds(bbox, {
  padding: {top: 10, bottom:25, left: 15, right: 5}
});
```

```
var bbox = [[-79, 43], [-73, 45]];
map.fitBounds(bbox, {
  padding: 20
});
```

RequestParameters (parameter group)

Object that is returned by the callback method `Map.options.transformRequest`

src/ui/ajax.js

Parameters:

url:string — request URL.

headers:object — headers sent with the request.

method:string — request method 'GET' | 'POST' | 'PUT'.

type:string — return type of request body (body response) 'string' | 'json' | 'arrayBuffer'.

credentials:string — 'same-origin' | 'include' Use 'include' to send cookies in cross domain requests.

collectResourceTiming:boolean — if true, the Resource Timing API information will be available for requests made by GeoJSON and Vector Tile (this information is not normally available from the main JavaScript thread). The information will be returned in the `ResourceTiming` property.

Example:

```
transformRequest: function(url, resourceType) {
  if (resourceType === 'Source' && url.indexOf('http://myHost') >
-1) {
    return {
      url: url.replace('http', 'https'),
      headers: { 'my-custom-header': true },
      credentials: 'include' // Include cookies for
cross-origin requests
    }
  }
};
```

StyleImageInterface (specification for developers)

It is not a method or a class. It is an interface for dynamically generated images. Images that implement this interface can be redrawn for each frame. They can be used to animate icons and patterns, or to make them react to user input data. Images can implement the `StyleImageInterface#render` method. This method is called frame by frame.

src/style/style_image.js

Parameters:

width: number

height: number

data: (Uint8Array|Uint8ClampedArray)

Example:

```
var flashingSquare = {
  width: 64,
  height: 64,
  data: new Uint8Array(64 * 64 * 4),

  onAdd: function(map) {
    this.map = map;
  },

  render: function() {
    // keep repainting while the icon is on the map
    this.map.triggerRepaint();

    // alternate between black and white based on the time
    var value = Math.round(Date.now() / 1000) % 2 === 0 ? 255 :
0;

    // check if image needs to be changed
    if (value !== this.previousValue) {
      this.previousValue = value;

      var bytesPerPixel = 4;
      for (var x = 0; x < this.width; x++) {
        for (var y = 0; y < this.height; y++) {
          var offset = (y * this.width + x) * bytesPerPixel;
          this.data[offset + 0] = value;
          this.data[offset + 1] = value;
          this.data[offset + 2] = value;
          this.data[offset + 3] = 255;
        }
      }
    }
  }
};
```

```

    }
  }

  // return true to indicate that the image changed
  return true;
}
}
}

```

```
map.addImage('flashing_square', flashingSquare);
```

CustomLayerInterface (specification for developers)

It is not a method or a class. It corresponds to a layer style customization interface. Custom layers allow the user to render directly into the GL context of the map using the camera. These layers can be added between any regular layers using `Map#addLayer`.

Custom layers must have a unique identifier and type "custom". They must implement `render` and may implement `prerender`, `onAdd` and `onRemove`. They can trigger a render with `Map#triggerRepaint` and should handle `Map.event:webglcontextlost` and `Map.event:webglcontextrestored` appropriately.

The `RenderingMode` property determines whether the layer will be rendered as a "2d" or "3d" map layer:

- «RenderingMode»: «3d» — to use the depth buffer and share it with other layers.
- «RenderingMode»: «2d» — to add a layer without depth. If you need to use a depth buffer for the "2d" layer, you must use an offscreen framebuffer and `CustomLayerInterface#prerender`

```
src/style/style_layer/custom_style_layer.js
```

Parameters:

id:string — unique ID layer.

type:string — layer type, must be «custom».

renderingMode:string — «3d» or «2d», by default «2d».

Example:

```

// Custom layer implemented as ES6 class
class NullIslandLayer {
  constructor() {
    this.id = 'null-island';
    this.type = 'custom';
    this.renderingMode = '2d';
  }
}

```

```

onAdd(map, gl) {
  const vertexSource = `
    uniform mat4 u_matrix;
    void main() {
      gl_Position = u_matrix * vec4(0.5, 0.5, 0.0, 1.0);
      gl_PointSize = 20.0;
    }`;

  const fragmentSource = `
    void main() {
      gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
    }`;

  const vertexShader = gl.createShader(gl.VERTEX_SHADER);
  gl.shaderSource(vertexShader, vertexSource);
  gl.compileShader(vertexShader);
  const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
  gl.shaderSource(fragmentShader, fragmentSource);
  gl.compileShader(fragmentShader);

  this.program = gl.createProgram();
  gl.attachShader(this.program, vertexShader);
  gl.attachShader(this.program, fragmentShader);
  gl.linkProgram(this.program);
}

render(gl, matrix) {
  gl.useProgram(this.program);
  gl.uniformMatrix4fv(gl.getUniformLocation(this.program,
"u_matrix"), false, matrix);
  gl.drawArrays(gl.POINTS, 0, 1);
}

}

map.on('load', function() {
  map.addLayer(new NullIslandLayer());
});

prewarm:function

```

Performs "prewarm" of all resources (for example, initializes web workers) to reduce the time to load the map. If `mmapgl.workerUrl` and `mmapgl.workerCount` are used, they must be defined before calling `prewarm()`.

By default, these resources are managed automatically, and they are initialized as needed (lazy) when the map is first created. When `prewarm()` is called, resources will be pre-created and will not be cleared when the last map is removed from the page. This allows them to be reused by new map instances. They can be cleared manually by calling the `mmapgl.clearPrewarmedResources()` function. This is only necessary if your web page remains active but stops using maps entirely.

This is useful when using maps in a Single Page Application (SPA) where the user will be navigating between different pages/screens, which can result in maps being instantiated and destroyed all the time.

`src/index.js`

Example:

```
mmapgl.prewarm()
```

clearPrewarmedResources: function

Clears resources that were previously created by `mmapgl.prewarm()`. Note that this is usually not necessary. You should only call this function if you expect the user of your application to not return to viewing the map at any point in your application.

`src/index.js`

Example:

```
mmapgl.clearPrewarmedResources()
```

3. Tags and controls

Marker

Creating a marker

src/ui/marker.js

Parameters

Name	Type	Description
element	HTMLElement	Use a DOM element as a marker. By default, a light blue, droplet-shaped svg marker is used (light blue, droplet-shaped).
anchor	string default: 'center'	Specifies the position of the icon relative to its center. Available values are 'center', 'top', 'bottom', 'left', 'right', 'top-left', 'top-right', 'bottom-left' and 'bottom-right'.
offset	PointLike	The shift is applied relative to the element center. Negative values point to the left and top.
color	string default: '#3FB1CE'	If options.element is not specified, then that color will be used. The default color is light blue.
scale	number default: 1	If options.element is not specified, then this scale will be used. By default, the scale is 41px high and 27px wide.
draggable	boolean default: false	If true, then you can drag markers to a new location (drag and drop)
clickTolerance	number default: 0	Maximum number of pixels that the user can point to while clicking on the marker for it to be considered a valid click (as opposed to dragging the handle). The map's clickTolerance inheritance is used by default.
rotation	number default: 0	Rotation angle of the marker relative to its corresponding rotationAlignment. A positive value will rotate the marker clockwise.
pitchAlignment	string default: 'auto'	<ul style="list-style-type: none">• 'map' aligns the marker to the map.• 'viewport' aligns the marker on the screen (viewport).• 'auto' automatically matches rotationAlignment value

rotationAlignment	string default: 'auto'	<ul style="list-style-type: none"> • 'map' aligns the rotation of the marker relative to the map, keeping the bearing when the map rotates. • 'viewport' aligns the rotation of the marker relative to the screen (viewport) regardless of the map rotation. • 'auto' equivalent to screen (viewport)
-------------------	---------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example:

```
var marker = new mmrgl.Marker()
    .setLngLat([30.5, 50.5])
    .addTo(map);
```

// Set options

```
var marker = new mmrgl.Marker({
    color: "#FFFFFF",
    draggable: true
}).setLngLat([30.5, 50.5])
    .addTo(map);
```

Methods

Name	Description
addTo(map)	Attaches a marker to a map object. Example: <pre>var marker = new mmrgl.Marker() .setLngLat([30.5, 50.5]) .addTo(map); // add the marker to the map</pre>
getElement()	Returns the marker HTML element.
getLngLat()	Get the geographic location of the marker. The longitude of the result may differ by a multiple of 360° from the longitude previously set by setLngLat because the Marker wraps the anchor longitude over the world copies to keep the marker on the screen.
getOffset()	Returns the marker offset.
getPitchAlignment()	Returns the market's current pitchAlignment property.

getPopup()	<p>Returns the popup instance associated with the Marker. Example:</p> <pre>var marker = new mmrgl.Marker() .setLngLat([0, 0]) .setPopup(new mmrgl.Popup().setHTML("<h1>Hello World!</h1>")) .addTo(map);</pre> <p>console.log(marker.getPopup()); // return the popup instance</p>
getRotation()	Returns the current marker rotation angle (in degrees).
getRotationAlignment()	Returns the marker's current rotationAlignment property.
isDraggable()	Returns true if the marker is draggable
remove()	<p>Removes a marker from the map Example:</p> <pre>var marker = new mmrgl.Marker().addTo(map); marker.remove();</pre>
setDraggable(shouldBeDraggable)	<p>Specifies the draggable property and marker functionality</p> <p>Parameters: shouldBeDraggable: boolean (by default false) Turns on/off the drag and drop feature</p>

<p>setLngLat(lnglat)</p>	<p>Sets the geographic position of the marker and moves it.</p> <p>Parameters: lnglat:LngLat where the marker should be placed.</p> <p>Example: <pre>// Create a new marker, set the longitude and latitude, and add it to the map new mmergl.Marker() .setLngLat([-65.017, -16.457]) .addTo(map);</pre></p>
<p>setOffset(offset)</p>	<p>Sets marker offset</p> <p>Parameters: offset:PointLike an offset in pixels to apply relative to the element's center. Negative values point to the left and up.</p>
<p>setPitchAlignment(alignment?)</p>	<p>Sets the marker's pitchAlignment property.</p> <p>Parameters: alignment:string sets the marker's pitchAlignment property. If "auto", it will automatically match the rotationAlignment.</p>
<p>setPopup(popup)</p>	<p>Binds the popup to a handle.</p> <p>Parameters: popup:Popup an instance of the popup class. If the value is undefined or null, then any popup set on this marker instance will be cleared.</p> <p>Example: <pre>var marker = new mmergl.Marker() .setLngLat([0, 0]) .setPopup(new mmergl.Popup().setHTML("<h1>Hello World!</h1>")) // add popup .addTo(map);</pre></p>
<p>setRotation(rotation)</p>	<p>Sets the marker's rotation property.</p> <p>Parameters:</p>

	rotation:number (by default 0) Marker rotation angle (clockwise, in degrees) relative to the corresponding Marker#setRotationAlignment setting
setRotationAlignment(alignment)	Sets the marker's rotationAlignment property. Parameters: alignment:string (by default 'auto') Sets the marker's rotationAlignment property.
togglePopup()	Opens or closes the popup instance associated with the handle, depending on the current state of the popup Example: <pre>var marker = new mmrgl.Marker() .setLngLat([0, 0]) .setPopup(new mmrgl.Popup().setHTML("<h1>Hello World!</h1>")) .addTo(map); marker.togglePopup(); // toggle popup open or closed</pre>

Events

Name	Description
drag	Triggered during a drag-and-drop
dragend	Triggered when the drag-and-drop is finished
dragstart	Triggered when a drag-and-drop is started

Popup

Creates a popup window.

src/ui/popup.js

Options

Name	Type	Description
------	------	-------------

maxWidth	string default: '240px'	A string that sets the maximum width CSS property of the popup window, such as '300px'. To ensure that the popup window size matches its content, set this property to "none" . Available values are listed here: https://developer.mozilla.org/en-US/docs/Web/CSS/max-width
anchor	string default: 'bottom'	Specifies the position of the icon relative to its center. Available values are 'center', 'top', 'bottom', 'left', 'right', 'top-left', 'top-right', 'bottom-left' and 'bottom-right'.
offset	number PointLike Object	Pixel offset applied to the popup window, specified as: <ul style="list-style-type: none"> • number, indicates the distance from the popup location • PointLike specifying a constant offset • point object {'center': PointLike, 'top': PointLike...}, specifies an offset for each anchor position. Negative offsets point to the left and up.
className	string	Container class name
focusAfterOpen	boolean default: true	If true, once a popup window appears, an attempt will be made to set the focus (cursor) to the first element inside the window, for example <input/>.
closeOnMove	boolean default: false	If true, then the popup will be closed when the map is moved.
closeOnClick	boolean default: true	If true, then the popup will be closed when the map is clicked.
closeButton	boolean default: true	If true, a close button will appear in the upper right corner of the popup.

Example:

```
var markerHeight = 50, markerRadius = 10, linearOffset = 25;
var popupOffsets = {
  'top': [0, 0],
  'top-left': [0,0],
  'top-right': [0,0],
  'bottom': [0, -markerHeight],
  'bottom-left': [linearOffset, (markerHeight - markerRadius +
linearOffset) * -1],
  'bottom-right': [-linearOffset, (markerHeight - markerRadius +
linearOffset) * -1],
  'left': [markerRadius, (markerHeight - markerRadius) * -1],
  'right': [-markerRadius, (markerHeight - markerRadius) * -1]
```

```

};
var popup = new mmr
gl.Popup({offset: popupOffsets, className: 'my-class'})
  .setLngLat(e.lngLat)
  .setHTML("<h1>Hello World!</h1>")
  .setMaxWidth("300px")
  .addTo(map);

```

Methods

Name	Description
addClassName(className)	<p>Adds a CSS class to the popup element.</p> <p>Options:</p> <p>className:string A non-empty string with the class name to add to popup</p> <p>Example:</p> <pre>let popup = new mmrgl.Popup() popup.addClassName('some-class')</pre>
addTo(map)	<p>Adds a popup window to the map.</p> <p>Example:</p> <pre>new mmrgl.Popup() .setLngLat([0, 0]) .setHTML("<h1>Null Island</h1>") .addTo(map);</pre>
getElement()	<p>Returns an HTML element of the popup window.</p> <p>Example:</p> <pre>// Change the `Popup` element's font size var popup = new mmrgl.Popup() .setLngLat([-96, 37.8]) .setHTML("<p>Hello World!</p>") .addTo(map); var popupElem = popup.getElement(); popupElem.style.fontSize = "25px";</pre>
getLngLat()	<p>Returns the geographic location of the popup anchor.</p> <p>The resulted longitude may differ by a multiple of 360° from the longitude previously set by setLngLat because the Popup wraps</p>

	the anchor longitude over the world copies to keep the popup on the screen.
getMaxWidth()	Returns the maximum popup width.
isOpen()	Returns true if the popup is open, false if it is closed.
remove()	Removes the popup from the map it was added to. Example: <pre>var popup = new mmrgl.Popup().addTo(map); popup.remove();</pre>
removeClassName(className)	Removes the CSS class from the container's popup element. Options: className:string A non-empty string with the name of the CSS class to remove from the popup container Example: <pre>let popup = new mmrgl.Popup() popup.removeClassName('some-class')</pre>
setDOMContent(htmlNode)	Sets the popup content to an element provided as a DOM node. Options: htmlNode:Node The DOM node to be used as the popup content. Example: <pre>// create an element with the popup content var div = window.document.createElement('div'); div.innerHTML = 'Hello, world!'; var popup = new mmrgl.Popup() .setLngLat(e.lngLat) .setDOMContent(div) .addTo(map);</pre>

<p>setHTML(html)</p>	<p>Sets the content of the popup to HTML code provided as a string.</p> <p>This method does not perform HTML filtering or sanitization and should only be used with trusted content. Consider <code>Popup#setText</code> if the content is an unreliable text string.</p> <p>Options: html:string A string representing the HTML popup content.</p> <p>Example:</p> <pre>var popup = new mmrgl.Popup() .setLngLat(e.lngLat) .setHTML("<h1>Hello World!</h1>") .addTo(map);</pre>
<p>setLngLat(Inglat)</p>	<p>Sets the geographic location of the popup anchor and moves it. Replaces <code>trackPointer()</code>.</p> <p>Options: Inglat:LngLatLike geographic location to set as the popup anchor.</p>
<p>setMaxWidth(maxWidth)</p>	<p>Sets the maximum popup width. This sets the <code>max-width</code> CSS property. Available values are listed below: https://developer.mozilla.org/en-US/docs/Web/CSS/max-width</p> <p>Options: maxWidth:string string representing the maximum width value.</p>
<p>setOffset(offset?)</p>	<p>Sets a popup offset.</p> <p>Options: offset:Offset popup offset</p>
<p>setText(text)</p>	<p>Sets the popup content as a string of text.</p> <p>This function creates a text node in DOM, so it cannot embed raw HTML. Use this method to protect against XSS if the popup content is user-provided.</p> <p>Options: text:string popup text content.</p> <p>Example:</p> <pre>var popup = new mmrgl.Popup() .setLngLat(e.lngLat) .setText('Hello, world!')</pre>

	<code>.addTo(map);</code>
<code>toggleClassName(className)</code>	<p>Adds or removes the given CSS class in the popup container, depending on whether the class is currently in the container.</p> <p>Options: className:string not-empty string with CSS class name to add/remove</p> <p>Example:</p> <pre>let popup = new mmrgl.Popup() popup.toggleClassName('toggleClass')</pre>
<code>trackPointer()</code>	<p>Tracks the popup anchor to the cursor position on screens using the pointer (it will be hidden on touch screens). Replaces <code>setLngLat</code> behavior. For most use cases, it is recommended to set <code>closeOnClick</code> and <code>closeButton</code> to <code>false</code>.</p> <p>Example:</p> <pre>var popup = new mmrgl.Popup({ closeOnClick: false, closeButton: false }) .setHTML("<h1>Hello World!</h1>") .trackPointer() .addTo(map);</pre>

Events

Name	Description
<code>close</code>	<p>Triggered when the popup is closed manually or automatically.</p> <p>Example:</p> <pre>// Create a popup var popup = new mmrgl.Popup(); // Set an event listener that will fire // any time the popup is closed popup.on('close', function(){ console.log('popup was closed'); });</pre>

open	<p>Triggered when a popup window is opened manually or automatically.</p> <p>Example:</p> <pre>// Create a popup var popup = new mmrgl.Popup(); // Set an event listener that will fire // any time the popup is opened popup.on('open', function(){ console.log('popup was opened'); });</pre>
------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

IControl (Developer specification)

IControl corresponds to an interface for interactive controls added to the map. This is a specification for developers: it is neither a method nor a class.

Controls must implement `onAdd` and `onRemove` and there must be `<div>` as well. Add the `mmrgl-ctrl` class to use the MMR GL JS control style.

`src/ui/map.js`

Example:

```
// Control implemented as ES6 class
class HelloWorldControl {
  onAdd(map) {
    this._map = map;
    this._container = document.createElement('div');
    this._container.className = 'mmrgl-ctrl';
    this._container.textContent = 'Hello, world';
    return this._container;
  }

  onRemove() {
    this._container.parentNode.removeChild(this._container);
    this._map = undefined;
  }
}
```

```
// Control implemented as ES5 prototypical class
function HelloWorldControl() { }
```

```
HelloWorldControl.prototype.onAdd = function(map) {
  this._map = map;
}
```



```

    this._container = document.createElement('div');
    this._container.className = 'mmrgl-ctrl';
    this._container.textContent = 'Hello, world';
    return this._container;
};

HelloWorldControl.prototype.onRemove = function () {
    this._container.parentNode.removeChild(this._container);
    this._map = undefined;
};

```

Methods

Name	Description
getDefaultPosition()	Whenever so required, specify a default position for this control. If this method is implemented and Map#addControl is called without a position parameter, then the value returned by getDefaultPosition will be used as the control's position.
onAdd(map)	Register the control on the map and let it register the event listener and resources. This method is called by Map#addControl internally.
onRemove(map)	Unregister the control from the map and give it a chance to detach the event listener and resources. This method is called by Map#removeControl internally.

NavigationControl

NavigationControl contains zoom buttons and a compass.

src/ui/control/navigation_control.js

Options

Name	Type	Description
showCompass	boolean default: true	If true, then the compass button is displayed.
showZoom	boolean default: true	If true, the in and zoom out buttons are displayed.
visualizePitch	boolean default: false	If true, then the step is rendered by rotating the x-axis.

Example:

```
var nav = new mmrgl.NavigationControl();
map.addControl(nav, 'top-left');
```

GeolocateControl

GeolocateControl features a button that uses the browser's geolocation API to locate the user on a map.

However, not all browsers support geolocation, and some users may choose to disable this feature. Geolocation support for modern browsers, including Chrome, requires sites to be served over HTTPS. If geolocation support is not available, then GeolocateControl will be displayed as disabled.

The zoom level will depend on the geolocation accuracy provided by the device.

GeolocateControl has two modes. If `trackUserLocation` is false (the default), then the control acts like a button: when clicked, it sets the map's camera to the user's location. If the user moves, the map is not updated. This is most suitable for the desktop. If `trackUserLocation` is true, then the control acts as a toggle button that tracks the user's movement. In this mode, the GeolocateControl has three interaction states:

- 'active': The map camera updates automatically as the user's location changes, keeping the location point in the center.
- 'passive': The user's location point is automatically updated, but the map camera is not. It occurs when the user initiates a map move.
- 'disabled': Occurs if Geolocation is not available, disabled, or restricted.

These interaction states cannot be automatically controlled, they are rather set based on user interaction.

`src/ui/control/geolocate_control.js`

Options

Name	Type	Description
<code>positionOptions</code>	Object default: { <code>enableHighAccuracy</code> :false, <code>timeout</code> :6000 }	PositionOptions geolocation API object.

fitBoundsOptions	Object default: { maxZoom:15 }	Map#fitBounds is an Options object that is used when the map is moved and scaled to the user's location. The default is maxZoom 15 to limit the map zoom to very precise locations.
trackUserLocation	boolean default: false	If true, the Geolocate control becomes a switch and once it is active the map will receive updates about the user's location as it changes.
showAccuracyCircle	boolean default: true	By default, if showUserLocation is set to true, a transparent circle will be drawn around the user's location, indicating the accuracy of the user's location (95% accuracy). Set to false to disable. Always disabled when showUserLocation is false.
showUserLocation	boolean default: true	By default, the point will be displayed on the map at the user's location. Set to false to disable.

Example:

```
map.addControl(new mmrgl.GeolocateControl({
  positionOptions: {
    enableHighAccuracy: true
  },
  trackUserLocation: true
}));
```

Methods

Name	Description
trigger	<p>Program it to request and move the map to the user's location.</p> <p>Example:</p> <pre>// Initialize the geolocate control. var geolocate = new mmrgl.GeolocateControl({ positionOptions: { enableHighAccuracy: true }, trackUserLocation: true }); // Add the control to the map. map.addControl(geolocate); map.on('load', function() { geolocate.trigger(); });</pre>

--	--

Events

Name	Description
error	<p>Triggered by Geolocation API updates, which was returned as an error.</p> <p>Options: data:PositionError returned PositionError object from the callback to Geolocation.getCurrentPosition() or Geolocation.watchPosition().</p> <p>Example:</p> <pre>// Initialize the geolocate control. var geolocate = new mmrgl.GeolocateControl({ positionOptions: { enableHighAccuracy: true }, trackUserLocation: true }); // Add the control to the map. map.addControl(geolocate); // Set an event listener that fires // when an error event occurs. geolocate.on('error', function() { console.log('An error event has occurred.') });</pre>

<p>geolocate</p>	<p>Triggered by Geolocation API updates, which was returned as a successful result.</p> <p>Options: data:Position returned Position object from a call to Geolocation.getCurrentPosition() or Geolocation.watchPosition().</p> <p>Example:</p> <pre>// Initialize the geolocate control. var geolocate = new mmrgl.GeolocateControl({ positionOptions: { enableHighAccuracy: true }, trackUserLocation: true }); // Add the control to the map. map.addControl(geolocate); // Set an event listener that fires // when a geolocate event occurs. geolocate.on('geolocate', function() { console.log('A geolocate event has occurred.') });</pre>
<p>outofmaxbounds</p>	<p>Triggered by Geolocation API updates, which was returned as a successful result, but the user's position is beyond maxBounds of the map.</p> <p>Options: data:Position returned Position object from a call to Geolocation.getCurrentPosition() or Geolocation.watchPosition().</p> <p>Example:</p> <pre>// Initialize the geolocate control. var geolocate = new mmrgl.GeolocateControl({ positionOptions: { enableHighAccuracy: true }, trackUserLocation: true }); // Add the control to the map. map.addControl(geolocate); // Set an event listener that fires // when an outofmaxbounds event occurs. geolocate.on('outofmaxbounds', function() { console.log('An outofmaxbounds event has occurred.') });</pre>

trackuserlocationend	<p>Triggered when the Geolocate control enters the background state. This happens when the user changes the camera while active. This only applies if trackUserLocation is set to true. In the background, a point on the map will update with location updates, but the camera will not be updated.</p> <p>Example:</p> <pre>// Initialize the geolocate control. var geolocate = new mmrgl.GeolocateControl({ positionOptions: { enableHighAccuracy: true }, trackUserLocation: true }); // Add the control to the map. map.addControl(geolocate); // Set an event listener that fires // when a trackuserlocationend event occurs. geolocate.on('trackuserlocationend', function() { console.log('A trackuserlocationend event has occurred.') });</pre>
trackuserlocationstart	<p>Triggered when the Geolocate control becomes active, or when the first time a successful geolocation API position is received for the user (followed by the geolocate event), or when the user presses the geolocate button while in the background, which uses the last known position to re-center map and enter the active state (the geolocate event will only occur if the user's location changes).</p> <p>Example:</p> <pre>// Initialize the geolocate control. var geolocate = new mmrgl.GeolocateControl({ positionOptions: { enableHighAccuracy: true }, trackUserLocation: true }); // Add the control to the map. map.addControl(geolocate); // Set an event listener that fires // when a trackuserlocationstart event occurs. geolocate.on('trackuserlocationstart', function() { console.log('A trackuserlocationstart event has occurred.') });</pre>

ScaleControl

ScaleControl control displays the ratio of the map distance to the corresponding ground distance.

src/ui/control/scale_control.js

Options

Name	Type	Description
maxWidth	number default: '100'	Maximum length of the zoom control, in pixels.
unit	string default: 'metric'	Distance unit ('imperial', 'metric' or 'nautical').

Example:

```
var scale = new mmlgl.ScaleControl({
  maxWidth: 80,
  unit: 'imperial'
});
map.addControl(scale);
scale.setUnit('metric')
```

Methods

Name	Description
setUnit(unit)	Set the distance unit on the scale Options: unit:Unit The unit of distance measurement ("imperial" , "metric" or "nautical").

FullscreenControl

FullscreenControl contains a button to switch the map to full screen mode and back.

src/ui/control/fullscreen_control.js

Options

Name	Type	Description
container	HTMLElement	container — DOM element to open in full screen mode. By default, the map container will be used for full screen mode.

Example:

```
map.addControl(new mmlgl.FullscreenControl({container:
  document.querySelector('body')
})
```

});

4. Geography and geometry

LngLat

LngLat object consists of longitude and latitude (in degrees). These coordinates are based on the WGS84 standard (EPSG:4326). MMR GL uses longitude and latitude coordinate order (as opposed to latitude and longitude values) according to the GeoJSON specification. Note that any MMR GL method that takes an LngLat object as an argument or option can also take an array of two numbers and perform an implicit conversion. This flexible type is documented as LngLatLike.

src/geo/lng_lat.js

Options

lng:number — longitude in degrees

lat:number — latitude in degrees

Example:

```
var ll = new mmrgl.LngLat(-123.9749, 40.7736);  
ll.lng; // = -123.9749
```

static methods

Name	Description
convert(input)	<p>Converts an array of two numbers or an object with properties lng and lat or lon and lat to an LngLat object.</p> <p>If a LngLat object is passed, the function returns it unchanged.</p> <p>Options: input:LngLatLike an array of two numbers, or an object to convert, or a LngLat object to return.</p> <p>Example: <pre>var arr = [-73.9749, 40.7736]; var ll = mmrgl.LngLat.convert(arr); ll; // = LngLat {lng: -73.9749, lat: 40.7736}</pre></p>

instance methods

Name	Description
------	-------------

distanceTo(LngLat)	<p>Returns the approximate distance between a pair of coordinates in meters using the Haversine formula (from R. W. Sinnott, "The Haversine Virtues", Sky and Telescope, vol. 68, no. 2, 1984, p. 159)</p> <p>Options: LngLat:LngLat coordinates to estimate distance to the target</p> <p>Example: <pre>var newYork = new mmrgl.LngLat(-74.0060, 40.7128); var losAngeles = new mmrgl.LngLat(-118.2437, 34.0522); newYork.distanceTo(losAngeles); // = 3935751.690893987, "true distance" calculated based on non-spherical approximation is about 3966 km</pre></p>
toArray()	<p>Returns the coordinates represented as an array of two numbers.</p> <p>Example: <pre>var ll = new mmrgl.LngLat(-73.9749, 40.7736); ll.toArray(); // = [-73.9749, 40.7736]</pre></p>
toBounds(radius)	<p>Returns LngLatBounds from coordinates extended by the given radius. The returned LngLatBounds contains the entire radius.</p> <p>Options: radius:number (default 0) Distance in meters from the coordinates to extend the bounds.</p> <p>Example: <pre>var ll = new mmrgl.LngLat(-73.9749, 40.7736); ll.toBounds(100).toArray(); // = [[-73.97501862141328, 40.77351016847229], [-73.97478137858673, 40.77368983152771]]</pre></p>
toString()	<p>Returns the coordinates represented as a string.</p> <p>Example: <pre>var ll = new mmrgl.LngLat(-73.9749, 40.7736); ll.toString(); // = "LngLat(-73.9749, 40.7736)"</pre></p>

wrap()	<p>Returns a new LngLat object whose longitude is wrapped in the range (from -180 to +180).</p> <p>Example:</p> <pre>var ll = new mmrgl.LngLat(286.0251, 40.7736); var wrapped = ll.wrap(); wrapped.lng; // = -73.9749</pre>
--------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

LngLatLike

LngLat can be an array of two numbers (longitude and latitude), or an object with lng and lat or lon and lat properties.

src/geo/lng_lat.js

Example:

```
var v1 = new mmrgl.LngLat(-122.420679, 37.772537);
var v2 = [-122.420679, 37.772537];
var v3 = {lon: -122.420679, lat: 37.772537};
```

LngLatBounds

The LngLatBounds object represents the boundary defined by its southwest and northeast points in longitude and latitude.

If no arguments are provided, a null boundary is created.

Any MMR GL method that takes a LngLatBounds object as an argument or option can also take an array of two LngLatLike constructs and perform an implicit conversion. This flexible type is documented as LngLatBoundsLike.

src/geo/lng_lat_bounds.js

Options

sw:number — longitude in degrees

ne:number — latitude in degrees

static methods

Name	Description
------	-------------

convert(input)	<p>Converts an array to a LngLatBounds object. If a LngLatBounds object is passed, the function returns it unchanged. Internally, the function calls LngLat#convert to convert arrays to LngLat values.</p> <p>Options: input:LngLatBoundsLike An array of two coordinates to transform, or LngLatBounds to return.</p> <p>Example:</p> <pre>var arr = [[-73.9876, 40.7661], [-73.9397, 40.8002]]; var llb = mmrgl.LngLatBounds.convert(arr); llb; // = LngLatBounds {_sw: LngLat {lng: -73.9876, lat: 40.7661}, _ne: LngLat {lng: -73.9397, lat: 40.8002}}</pre>
----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

instance methods

Name	Description
contains(Inglat)	<p>Check if the point is inside the bounding box.</p> <p>Options: Inglat:LngLatLike</p> <p>Example:</p> <pre>var llb = new mmrgl.LngLatBounds(new mmrgl.LngLat(-73.9876, 40.7661), new mmrgl.LngLat(-73.9397, 40.8002)); var ll = new mmrgl.LngLat(-73.9567, 40.7789); console.log(llb.contains(ll)); // = true</pre>
extend(obj)	<p>Expand the bounds to include the given LngLatLike or LngLatBounds</p> <p>Options: obj:(LngLatLike LngLatBoundsLike) extension object</p>

getCenter()	<p>Returns a geographic coordinate equidistant from the corners of the bounding box.</p> <p>Example:</p> <pre>var llb = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002]); llb.getCenter(); // = LngLat {lng: -73.96365, lat: 40.78315}</pre>
getEast()	Returns the east edge of the bounding box.
getNorth()	Returns the north edge of the bounding box.
getNorthEast()	Returns the northeast corner of the bounding box.
getNorthWest()	Returns the northwest corner of the bounding box.
getSouth()	Returns the south edge of the bounding box.
getSouthEast()	Returns the southeast corner of the bounding box.
getSouthWest()	Returns the southwest corner of the bounding box.
getWest()	Returns the southwest corner of the bounding box.
isEmpty()	Check if the bounding box is empty/null.
setNorthEast(ne)	<p>Sets the northeast corner of the bounding box.</p> <p>Options:</p> <p>ne:LngLatLike set the northeast corner of the bounding box</p>
setSouthWest(sw)	<p>Sets the southwest corner of the bounding box</p> <p>Options:</p> <p>ne:LngLatLike object describing the southwest corner of the bounding box</p>
toArray()	<p>Returns the bounding box represented as an array.</p> <p>Example:</p> <pre>var llb = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397, 40.8002]); llb.toArray(); // = [[-73.9876, 40.7661], [-73.9397, 40.8002]]</pre>

toString()

Returns the bounding box, represented as a string.

Example:

```
var llb = new mmrgl.LngLatBounds([-73.9876, 40.7661],  
[-73.9397, 40.8002]);  
llb.toString(); // = "LngLatBounds(LngLat(-73.9876,  
40.7661), LngLat(-73.9397, 40.8002))"
```

Example:

```
var sw = new mmrgl.LngLat(-73.9876, 40.7661);  
var ne = new mmrgl.LngLat(-73.9397, 40.8002);  
var llb = new mmrgl.LngLatBounds(sw, ne);
```

LngLatBoundsLike

The LngLatBounds object is an array of LngLatLike objects in the order [sw, ne] or an array of numbers in the order [west, south, east, north].

src/geo/lng_lat_bounds.js

Example:

```
var v1 = new mmrgl.LngLatBounds(  
new mmrgl.LngLat(-73.9876, 40.7661),  
new mmrgl.LngLat(-73.9397, 40.8002)  
);  
var v2 = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397,  
40.8002])  
var v3 = [[-73.9876, 40.7661], [-73.9397, 40.8002]];
```

Point

A Point object consists of x and y and indicates the location of the point on the map
src/ui/map.js

Example:

```
var point = new mmrgl.Point(-77, 38);
```

PointLike

A Point object or an array of two numbers representing the x and y screen coordinates in pixels.

src/ui/map.js

Example:

```
var p1 = new mmrgl.Point(-77, 38); // a PointLike which is a Point
var p2 = [-77, 38]; // a PointLike which is an array of two numbers
```

MercatorCoordinate

A MercatorCoordinate object represents a three-dimensional position.

MercatorCoordinate uses the Mercator Web Projection (EPSG:3857) with slightly different units:

- size of 1 unit of measure is the width of the projected world (instead of "meter mercator").
- origin of space is in the northwest corner, not in the middle.

Example:

MercatorCoordinate(0, 0, 0) — northwest corner of the mercator world, and
 MercatorCoordinate(1, 1, 0) — southeast corner. If you are familiar with vector tiles, it may be useful to represent the coordinate space as a 0/0/0 tile with a magnitude/scale of 1.

The Z-dimension of MercatorCoordinate is conformal. A cube in mercator coordinate space will be displayed as a cube.

src/geo/mercator_coordinate.js

Options

x:number — X position

y:number — Y position

z:number — Z position

static methods

Name	Description
fromLngLat(LngLatLike, altitude)	<p>Project LngLat to MercatorCoordinate.</p> <p>Options: LngLatLike:LngLatLike Location for the project. altitude:number (default 0) Position altitude in meters.</p> <p>Example:</p> <pre>var coord = mmrgl.MercatorCoordinate.fromLngLat({ lng: 0, lat: 0}, 0); coord; // MercatorCoordinate(0.5, 0.5, 0)</pre>

instance methods

Name	Description
------	-------------

meterInMercatorCoordinate Units()	Returns a distance of 1 meter in MercatorCoordinate units at the given latitude. For real-world coordinates using meters, this naturally provides the scale for conversion to MercatorCoordinate
toAltitude()	Returns altitude in meters from the coordinate. Example: <pre>var coord = new mmrgl.MercatorCoordinate(0, 0, 0.02); coord.toAltitude(); // 6914.281956295339</pre>
toLngLat()	Returns LngLat for the coordinate. Example: <pre>var coord = new mmrgl.MercatorCoordinate(0.5, 0.5, 0); var lngLat = coord.toLngLat(); // LngLat(0, 0)</pre>

Example:

```
var nullIsland = new mmrgl.MercatorCoordinate(0.5, 0.5, 0);
```

EdgeInsets

The EdgeInset object represents the screen padding applied to the viewport edges. This shifts the apparent map center or vanishing point. This is useful for adding floating UI elements on top of the map and moving the vanishing point when UI elements are resized.

src/geo/edge_insets.js

Options

top:number — 0 by default

bottom:number — 0 by default

left:number — 0 by default

right:number — 0 by default

static methods

Name	Description
------	-------------

<p>getCenter(width, height)</p>	<p>A utility method that calculates a new application center or vanishing point after inserts have been applied. In pixels and with top left value (0.0) and +y down.</p> <p>Options: width: number is the map width in pixels. height: number is the map height in pixels.</p>
<p>interpolate(start, target, t)</p>	<p>Interpolates an insert in place. This saves the current insertion value for any insertion not present in target.</p> <p>Options: start: (PaddingOptions EdgesInsets) padding options. target: PaddingOptions padding options for target. t: number interpolation variable.</p>
<p>toJSON()</p>	<p>Returns the current state as json (used to set a read-only view for the insert).</p>

5. Handlers

BoxZoomHandler

BoxZoomHandler allows the user to zoom the map to fit within the bounding box. To define a rectangle, you must simultaneously press and hold the Shift key and the left mouse button and drag the cursor.

src/ui/handler/box_zoom.js

Methods

Name	Description
disable()	Disables "box zoom" Example: <pre>map.boxZoom.disable();</pre>
enable()	Enables "box zoom" Example: <pre>map.boxZoom.enable();</pre>
isActive()	Returns true if "box zoom" is active
isEnabled()	Returns true if "box zoom" is enabled

ScrollZoomHandler

ScrollZoomHandler allows the user to zoom the map by scrolling.

src/ui/handler/scroll_zoom.js

Methods

Name	Description
------	-------------

disable()	<p>Disables "scroll to zoom"</p> <p>Example:</p> <pre>map.scrollZoom.disable()</pre>
enable(options?)	<p>Options:</p> <p>around — if "center" is passed, the map will scale around the map center</p> <p>Examples:</p> <pre>map.scrollZoom.enable(); map.scrollZoom.enable({ around: 'center' })</pre>
isEnabled()	Returns true if "scroll to zoom" is enabled
setWheelZoomRate(wheelZoomRate)	<p>Sets mouse wheel zoom speed</p> <p>Options</p> <p>wheelZoomRate — (default 1/450) Zoom speed.</p> <p>Example:</p> <pre>// Slow down zoom of mouse wheel map.scrollZoom.setWheelZoomRate(1/600);</pre>
setZoomRate(zoomRate)	<p>Sets trackpad zoom speed</p> <p>Options</p> <p>zoomRate — (default 1/100) Zoom rate.</p> <p>Example:</p> <pre>// Speed up trackpad zoom map.scrollZoom.setZoomRate(1/25);</pre>

DragPanHandler

DragPanHandler allows the user to move the map by clicking and dragging the cursor.
src/ui/handler/shim/drag_pan.js

Methods

Name	Description
------	-------------

disable()	<p>Disables "drag to pan"</p> <p>Example: <code>map.dragPan.disable();</code></p>
enable(options?)	<p>Enables "drag to pan"</p> <p>Options: linearity — used to scale drag velocity easing — easing function is applied to <code>map.panTo</code> when dragged. maxSpeed — maximum value of drag velocity. deceleration — speed at which movement decreases after touching.</p> <p>Examples: <code>map.dragPan.enable();</code></p> <pre>map.dragPan.enable({ linearity: 0.3, easing: bezier(0, 0, 0.3, 1), maxSpeed: 1400, deceleration: 2500, });</pre>
isActive()	Returns true if "drag to pan" is active
isEnabled()	Returns true if "drag to pan" is enabled

DragRotateHandler

DragRotateHandler allows the user to rotate the map by clicking and dragging the cursor while holding down the right mouse button or the ctrl key.

`src/ui/handler/shim/drag_rotate.js`

Methods

Name	Description
disable()	<p>Disables "drag to pan"</p> <p>Example: <code>map.dragRotate.disable();</code></p>

enable()	Enables "drag to pan" Example: <code>map.dragRotate.enable();</code>
isActive()	Returns true if "drag to rotate" is active
isEnabled()	Returns true if "drag to rotate" is enabled

KeyboardHandler

KeyboardHandler allows the user to zoom, rotate and pan the map using the following keyboard shortcut keys:

- = / + : Scale up to 1.
- Shift-= / Shift-+ : Scale up to 2.
- -: Signal level decrease by 1.
- Shift -: Signal level decrease by 2.
- Arrow keys: Pan 100px.
- Shift+→: Rotation increase by 15 degrees.
- Shift+←: Rotation decrease by 15 degrees.
- Shift+↑: Pitch increase by 10 degrees.
- Shift+↓: Pitch decrease by 10 degrees.

src/ui/handler/keyboard.js

Methods

Name	Description
disable()	Disables "keyboard rotate and zoom" Example: <code>map.keyboard.disable();</code>
disableRotation()	Disables "keyboard pan/rotate" but leaves "keyboard zoom" enabled Example: <code>map.keyboard.disableRotation();</code>
enable()	Enables "keyboard rotate and zoom" Example: <code>map.keyboard.enable();</code>

enableRotation()	Enables "keyboard pan/rotate" Example: <code>map.keyboard.enableRotation();</code>
isActive()	Returns true if the handler is enabled and zoom/rotate is detected
isEnabled()	Returns true if "keyboard rotate and zoom" interaction is enabled

DoubleClickZoomHandler

DoubleClickZoomHandler allows the user to zoom the map at a specific point by double-clicking the left mouse button or double-tapping.

src/ui/handler/shim/dblclick_zoom.js

Methods

Name	Description
disable()	Disables "double click to zoom" Example: <code>map.doubleClickZoom.disable();</code>
enable()	Enables "double click to zoom" Example: <code>map.doubleClickZoom.enable();</code>
isActive()	Returns true if "double click to zoom" is active
isEnabled()	Returns true if "double click to zoom" is enabled

TouchZoomRotateHandler

TouchZoomRotateHandler allows the user to zoom in and rotate the map by pressing on the touch screen.

It can zoom in via single touch by double tapping and dragging. On the second tap, hold your finger and drag up or down to zoom in or out.

src/ui/handler/shim/touch_zoom_rotate.js

Methods

Name	Description
<code>disable()</code>	Disables "pinch to rotate and zoom". Example: <pre>map.touchZoomRotate.disable();</pre>
<code>disableRotation()</code>	Disables "pinch to rotate", leaving "pinch to zoom". Example: <pre>map.touchZoomRotate.disableRotation();</pre>
<code>enable(options?)</code>	Enables "pinch to rotate and zoom" Options around — if "center" is passed, the map will be scaled around the map center Examples: <pre>map.touchZoomRotate.enable();</pre> <pre>map.touchZoomRotate.enable({ around: 'center' });</pre>
<code>enableRotation()</code>	Enables "pinch to rotate" Example: <pre>map.touchZoomRotate.enable();</pre> <pre>map.touchZoomRotate.enableRotation();</pre>
<code>isActive()</code>	Returns true if the handler is enabled and has detected zoom/rotate.
<code>isEnabled()</code>	Returns true if "pinch to rotate and zoom" is enabled

TouchPitchHandler

TouchPitchHandler allows the user to tilt the map by dragging it up and down with two fingers.

`src/ui/handler/touch_zoom_rotate.js`

Methods

Name	Description
<code>disable()</code>	Disables "drag to pitch" Example: <code>map.touchPitch.disable();</code>
<code>enable()</code>	Enables "drag to pitch" Example: <code>map.touchPitch.enable();</code>
<code>isActive()</code>	Returns true if "drag to pitch" is active
<code>isEnabled()</code>	Returns true if "drag to pitch" is enabled

6. Sources

GeoJSONSource

A source which contains GeoJSON.

src/source/geojson_source.js

Examples:

```
map.addSource('some id', {
  type: 'geojson',
  data:
    'https://d2ad6b4ur7yvpq.cloudfront.net/naturalearth-3.3.0/ne_10m_ports
    .geojson'
});
```

```
map.addSource('some id', {
  type: 'geojson',
  data: {
    "type": "FeatureCollection",
    "features": [{
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [
          -76.53063297271729,
          39.18174077994108
        ]
      }
    }
  ]
});
```

```
map.getSource('some id').setData({
  "type": "FeatureCollection",
  "features": [{
    "type": "Feature",
    "properties": { "name": "Null Island" },
    "geometry": {
      "type": "Point",
      "coordinates": [ 0, 0 ]
    }
  }
});
```

```

    }]
  });

```

Methods

Name	Description
getClusterChildren(clusterId, callback)	<p>For clustered sources, it retrieves dependent elements of the given cluster at the next zoom level (as an array of GeoJSON objects).</p> <p>Options clusterId: number — cluster ID callback: function<Array<GeoJSONFeature>> — callback function that will be called when fetching features ((error, features) => { ... }).</p>
getClusterExpansionZoom(clusterId, callback)	<p>For clustered sources, the scale at which this cluster expands is selected.</p> <p>Options clusterId: number — cluster ID callback: function<number> — callback function that will be called when getting the zoom ((error, features) => { ... })</p>
getClusterLeaves(clusterId, limit, offset, callback)	<p>For clustered sources, it retrieves the source points that belong to the cluster (as an array of GeoJSON objects).</p> <p>Options clusterId: number — cluster ID limit: number — maximum number of features offset: number — number of features to skip (for example, for pagination). callback: function<Array<GeoJSONFeature>> — callback function that will be called when objects are retrieved ((error, features) => { ... }).</p> <p>Example:</p> <pre> // Retrieve cluster leaves on click map.on('click', 'clusters', function(e) { var features = map.queryRenderedFeatures(e.point, { layers: ['clusters'] }); var clusterId = features[0].properties.cluster_id; var pointCount = features[0].properties.point_count; </pre>

	<pre> var clusterSource = map.getSource('clusters'); clusterSource.getClusterLeaves(clusterId, pointCount, 0, function(error, features) { // Print cluster leaves in the console console.log('Cluster leaves:', error, features); }) }); </pre>
setData(data)	<p>Sets the GeoJSON data and re-renders the map.</p> <p>Options data:(Object string) — GeoJSON or URL to it. The latter is preferable for large files.</p>

VideoSource

The data source which contains the video.

src/source/video_source.js

Example:

```

// add to map
map.addSource('some id', {
  type: 'video',
  url: [
    'path-to-video.mp4',
    'path-to-video.webm'
  ],
  coordinates: [
    [-76.54, 39.18],
    [-76.52, 39.18],
    [-76.52, 39.17],
    [-76.54, 39.17]
  ]
});

// update
var mySource = map.getSource('some id');
mySource.setCoordinates([
  [-76.54335737228394, 39.18579907229748],
  [-76.52803659439087, 39.1838364847587],

```

```
    [-76.5295386314392, 39.17683392507606],  
    [-76.54520273208618, 39.17876344106642]  
]);
```

```
map.removeSource('some id'); // remove
```

Methods

Name	Description
getVideo()	Returns an HTML video
pause()	Sets a video pause
play()	Continues video playback
setCoordinates()	Sets video coordinates and re-renders the map

ImageSource

Image data source

src/source/image_source.js

Example:

```
// add to map  
map.addSource('some id', {  
  type: 'image',  
  url: 'foo.png',  
  coordinates: [  
    [-76.54, 39.18],  
    [-76.52, 39.18],  
    [-76.52, 39.17],  
    [-76.54, 39.17]  
  ]  
});  
  
// update coordinates  
var mySource = map.getSource('some id');  
mySource.setCoordinates([  
  [-76.54335737228394, 39.18579907229748],  
  [-76.52803659439087, 39.1838364847587],  
  [-76.5295386314392, 39.17683392507606],
```

```

        [-76.54520273208618, 39.17876344106642]
    ]);

    // update url and coordinates simultaneously
    mySource.updateImage({
        url: 'bar.png',
        coordinates: [
            [-76.54335737228394, 39.18579907229748],
            [-76.52803659439087, 39.1838364847587],
            [-76.5295386314392, 39.17683392507606],
            [-76.54520273208618, 39.17876344106642]
        ]
    })

    map.removeSource('some id'); // remove

```

Methods

Name	Description
setCoordinates(coordinates)	Sets the image coordinates and re-renders the map.
updateImage(options)	<p>Updates the URL — image address and coordinates (optionally). To avoid image flicker, set the raster-fade-duration property to 0.</p> <p>Options</p> <p>url:string — Image URL</p> <p>coordinates:Array<Array<number>> — four coordinates, represented as arrays of longitude and latitude numbers, that define the image corners. The coordinates start at the top left corner of the image and go clockwise.</p>

CanvasSource

The data source contains the HTML canvas content.
src/source/canvas_source.js

Example:

```

// add to map
map.addSource('some id', {
    type: 'canvas',
    canvas: 'idOfMyHTMLCanvas',
    animate: true,
    coordinates: [

```

```

        [-76.54, 39.18],
        [-76.52, 39.18],
        [-76.52, 39.17],
        [-76.54, 39.17]
    ]
});

// update
var mySource = map.getSource('some id');
mySource.setCoordinates([
    [-76.54335737228394, 39.18579907229748],
    [-76.52803659439087, 39.1838364847587],
    [-76.5295386314392, 39.17683392507606],
    [-76.54520273208618, 39.17876344106642]
]);

map.removeSource('some id'); // remove

```

Methods

Name	Description
getCanvas()	Returns the HTML canvas
pause()	Disables animation. The map will display a static copy of the canvas image.
play()	Enables animation. The map will be interactive.
setCoordinates(coordinates)	<p>Sets canvas coordinates and re-renders the map.</p> <p>Options:</p> <p>coordinates: Array<Array<number>> — four coordinates, represented as arrays of longitude and latitude numbers, that define the canvas corners. Coordinates start at the top left corner of the canvas and go clockwise.</p>

CanvasSourceOptions

Options for adding a canvas source type to the map.

src/source/canvas_source.js

Parameters

type:string — source type, should be equal to "canvas".

canvas:(string | HTMLCanvasElement) — canvas source from which pixels can be read. This can be a string representing the canvas element ID, or the HTMLCanvasElement itself.

coordinates:Array<Array<number>> — four coordinates denoting where to place the canvas corners, set in [longitude, latitude] pairs.

animate:boolean — whether the canvas source is animated. If the canvas is static (i.e. pixels don't need to be re-read on every frame), animate should be set to false to improve performance.

7. Events

Evented

Listeners adding and removal methods

src/util/evented.js

Methods

Name	Description
off(type, listener)	Removes a previously registered listener. Options type:string — event type listener:function — listener function
on(type, listener)	Registers a new listener Options type:string — event type listener:function — listener function
once(type, listener)	Adds a listener that will only be called once for the specified event type. Options type:string — event type listener:function — listener function

MapMouseEvent

Mouse related events

src/ui/events.js

Example:

```
// The `click` event is an example of a `MapMouseEvent`.  
// Set up an event listener on the map.  
map.on('click', function(e) {  
  // The event object (e) contains information like the  
  // coordinates of the point on the map that was clicked.  
  console.log('A click event has occurred at ' + e.lngLat);  
});
```

Properties

Name	Description
lngLat	Cursor coordinates on the map
originalEvent	DOM events that triggered the map event
point	Map related pixel coordinates of the mouse cursor are measured from the top left corner.
preventDefault()	Prevents subsequent event processing. If this method is called, the following events will be stopped: <ul style="list-style-type: none"> • mousedown, DragPanHandler behavior • mousedown DragRotateHandler behavior • mousedown, BoxZoomHandler behavior • dblclick, DoubleClickZoomHandler behavior
target	Map object that triggered the event.
type	Event type (Map.event:mousedown, Map.event:mouseup, Map.event:click, Map.event:dblclick, Map.event:mousemove, Map.event:mouseover, Map.event:mouseenter, Map.event:mouseleave, Map.event:mouseout, Map.event:contextmenu).

MapTouchEvent

Touch events

src/ui/events.js

Properties

Name	Description
lngLat	Touch coordinates on the map
lngLats	Array of coordinates corresponding to touches on the map
originalEvent	DOM events that triggered the map event
point	Map related pixel coordinates of the mouse cursor are measured from the top left corner.
points	Array of pixel coordinates corresponding to touches.

preventDefault()	Prevents subsequent event processing. If this method is called, the following events will be stopped: <ul style="list-style-type: none"> • touchstart, DragPanHandler behavior • touchstart, TouchZoomRotateHandler behavior
target	Map object that triggered the event.
type	Event type

MapBoxZoomEvent

'boxzoom' related events used by BoxZoomHandler

src/ui/events.js

Options

originalEvent:MouseEvent — DOM event that triggered the 'boxzoom' eventm (MouseEvent or KeyboardEvent).

type:string — event type 'boxzoom' (boxzoomstart, boxzoomend or boxzoomcancel).

target:Map — map instance that raised the event.

MapDataEvent

MapDataEvent object is created along with the Map.event:data and Map.event:dataloading events. Possible values for data types are:

- «source»: data not associated with tiles, but associated with any source.
- «style»: map style

Options

type:string — event type.

dataType:string — modified data type ("source", "style").

isSourceLoaded:boolean — true if the event data type is "source" and the source has no outstanding network requests.

source:Object — source type object.

sourceDataType:string — enabled for source data type events and the event reports that the internal data has been received or changed. Possible values are 'metadata' , 'content' and 'visibility' .

tile:Object — tile to be loaded or changed for source data type events and tile loading events.

coord:Coordinate — tile coordinate for source data type events and tile loading events.

src/ui/events.js

```
// The sourcedata event is an example of MapDataEvent.
// Set up an event listener on the map.
```

```
map.on('sourcedata', function(e) {  
  if (e.isSourceLoaded) {  
    // Do something when the source has finished loading  
  }  
});
```

MapWheelEvent

Wheel related events

src/ui/events.js

Properties

Name	Description
originalEvent	DOM event that triggered the map event.
preventDefault()	Prevents further event handling. If this method is called, ScrollZoomHandler will be stopped.
target	Map object that triggered the event.
type	Event type

8. How to use the library in React applications

React is a JavaScript library used to create user interfaces. Because React manipulates the DOM, it can be difficult to link React with other libraries that also manipulate the DOM and manage the state similar to MMR GL JS.

In this section, you'll learn how to create a React app that uses MMR GL JS to render a map, display the coordinates of the map's center point and zoom level, and then update the display in the course of user interaction with the map. You can use the principles in this guide to build more complex apps using both React and MMR GL JS.

Setting up a React app structure

Create a new folder with the project name.

- **package.json** — this file is used to specify all packages required by your application.

Create a public folder in the project folder. In this folder, create two new files:

- **index.html** — This HTML file will display a rendered map that your users can interact with.
- **site.css** — This file will contain the CSS needed to format the map and sidebar properly. Creates an HTML wireframe

In the project folder, create another folder named src. In the src folder, create a new file:

- **index.js** — this JavaScript file will contain all the React logic needed to set up and manage the app state and display the map.

After creating these folders and files, the following file structure will be generated:

```
project
├── package.json
├── public
│   ├── index.html
│   └── site.css
└── src
    └── index.js
```

Then, run `npm install`.

Creating a React app

Open `public/index.html` and paste the following code into it:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>MMR GL JS and React</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,
initial-scale=1" />

  <link href="https://geo.rustore.ru/sdk/js/<version>/mmr-gl.css"
rel="stylesheet" />
```

```

    <link href="%PUBLIC_URL%/site.css" rel="stylesheet" />

    <script
src="https://geo.rustore.ru/sdk/js/<version>/mmr-gl.js"></script>
</head>
<body>
    <div id="app"></div>
</body>
</html>

```

This code creates the HTML page structure available to a user. The `<body>` of the page has a `<div>` tag with the application ID. This `<div>` is the container in which the React app will be rendered on the page.

This code also references two different style files in `<head>`. The first style is the MMR GL JS style, which ensures that the elements on your map are displayed correctly. The second style is the `site.css` file you created earlier, where you'll add a CSS for a specific app.

Installing a default state

Open `src/index.js`. Add the following imports to the file top:

`src/index.js`

With hooks

```

import React, { useRef, useEffect, useState } from 'react';
import ReactDOM from 'react-dom';

```

With classes

```

import React from 'react';
import ReactDOM from 'react-dom';

```

Import MMR GL and add your access token. Set an access token (`accessToken`):

```

mmrGl.accessToken = 'token';

```

You can also customize your React app. To create the structure where you'll add the code from the next few steps, add the following `index.js`:

With hooks

```

const Map = () => {};

```

```

ReactDOM.render(<Map />, document.getElementById('app'));

```

With classes

```
class Map extends React.PureComponent {}
```

```
ReactDOM.render(<Map />, document.getElementById('app'));
```

This defines the map and then specifies that it should be rendered in a <div> with an application ID.

Map drawing

Follow the following steps to render the map in your application. The entry point for map initialization in a React app is beyond one element, specified in return. Add the following code to your application:

src/index.js

With hooks

```
return (  
  <div>  
    <div className="map-container" ref={mapContainer} />  
  </div>  
);
```

With classes

```
render() {  
  const { lng, lat, zoom } = this.state;  
  return (  
    <div>  
      <div ref={this.mapContainer} className="map-container" />  
    </div>  
  );  
}
```

mapContainer ref specifies that the map should be drawn on the HTML page in a new <div> element.

The map needs several style rules to display correctly. Add the following code to the site.css file:

```
.map-container {  
  position: absolute;  
  top: 0;  
  right: 0;  
  left: 0;
```

```
    bottom: 0;
  }
```

Save your changes. At the command prompt, run `npm start`. This starts a local server and opens a new page in your browser along with your application.

When you open your browser console, you may see an error: 'map' is assigned a value but never used. There is nothing to worry about — you'll be using the map variable in the next step!

Coordinate storage

Once there, you need to create a function that will save the new latitude, longitude, and scale as the user interacts with the map. You should set an MMR GL JS `map.on('move')` function that indicates the state values when the user moves the map. If you are using hooks in `useEffect` or classes in `componentDidMount()`, add the following code:

With hooks

```
map.on('move', () => {
  setLng(map.getCenter().lng.toFixed(4));
  setLat(map.getCenter().lat.toFixed(4));
  setZoom(map.getZoom().toFixed(2));
});
```

With classes

```
map.on('move', () => {
  this.setState({
    lng: map.getCenter().lng.toFixed(4),
    lat: map.getCenter().lat.toFixed(4),
    zoom: map.getZoom().toFixed(2)
  });
});
```

This function possesses `useState()` if you're using hooks, or `setState()` if you're using classes, to reset the `lng`, `lat`, and `zoom` values when the map is moved. The function features the following methods:

- `getCenter()`, MMR GL JS method to get the new longitude and latitude of the map center point.
- `getZoom()`, MMR GL JS method to determine the zoom level the map is set to.
- `toFixed()`, a JavaScript method that allows you to cut the resulting floating point number to a given number of digits.

Coordinate display

When collecting and storing this information, you can use `return` to display it on the map. Inside the opening `<div>` tag created to store the map, add a new `<div>` to display the map longitude, latitude, and scale. Afterwards, the return will look like this:

With hooks

```
return (  
  <div>  
    <div className="sidebar">  
      Longitude: {lng} | Latitude: {lat} | Zoom: {zoom}  
    </div>  
    <div className="map-container" ref={mapContainer} />  
  </div>  
)
```

With classes

```
render() {  
  const { lng, lat, zoom } = this.state;  
  return (  
    <div>  
      <div className="sidebar">  
        Longitude: {lng} | Latitude: {lat} | Zoom: {zoom}  
      </div>  
      <div ref={this.mapContainer} className="map-container" />  
    </div>  
  );  
}
```

Apply a few styles to the sidebar for correct display on the page. Add the following CSS to `site.css`:

```
.sidebar {  
  background-color: rgba(35, 55, 75, 0.9);  
  color: #ffffff;  
  padding: 6px 12px;  
  font: 15px/24px monospace;  
  z-index: 1;  
  position: absolute;  
  top: 0;  
  left: 0;  
  margin: 12px;  
  border-radius: 4px;  
}
```


Click “Save” and return to the browser page. The top left map corner has a sidebar styled according to the CSS rules set in site.css. The sidebar shows the current latitude and longitude of the map center, as well as the zoom level. As you zoom and move the map, the sidebar contents will be updated.

Description of additional map objects

AttributionControl

The AttributionControl control provides information about the map's attributes (copywriting, and so on).

src/ui/control/attribution_control.js

Options:

compact: boolean — if true, then it will always be displayed in a compact view, if false, it will always be in full size, by default it works depending on the map size (viewport < 640 ? compact : full)

customAttribution: string|Array<string> — string or strings to display in map attributes

Example:

```
var map = new mmrgl.Map({attributionControl: false})
  .addControl(new mmrgl.AttributionControl({
    compact: true
  }));
```

LngLatBoundsLike

An LngLatBounds object, an array of LngLatLike objects in the order [sw, ne], or an array of numbers in the order [west, south, east, north].

src/geo/lng_lat_bound.js

Example:

```
var v1 = new mmrgl.LngLatBounds(
  new mmrgl.LngLat(-73.9876, 40.7661),
  new mmrgl.LngLat(-73.9397, 40.8002)
);
var v2 = new mmrgl.LngLatBounds([-73.9876, 40.7661], [-73.9397,
40.8002])
var v3 = [[-73.9876, 40.7661], [-73.9397, 40.8002]];
```

Static map

/staticmap/png — service that allows you to get a map image.

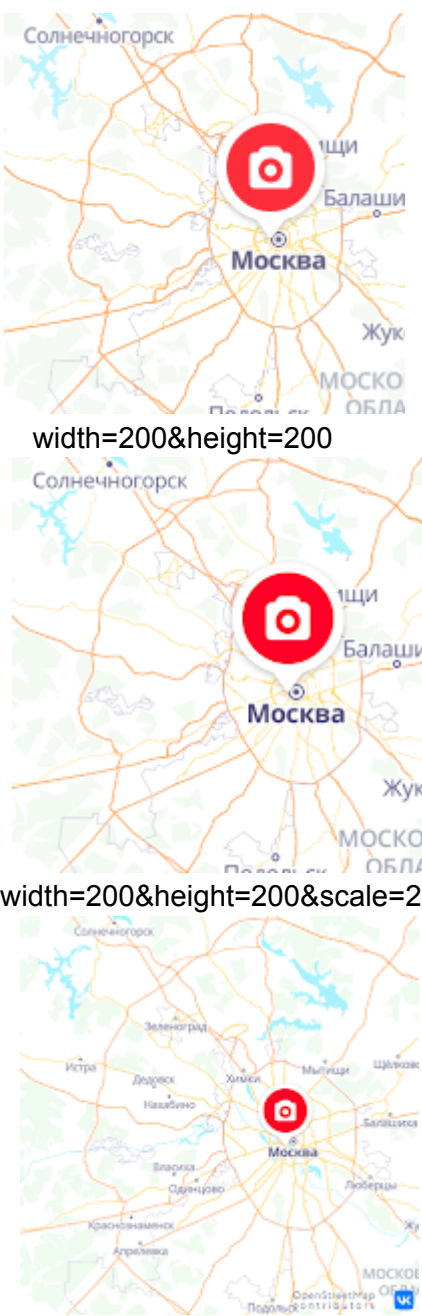
Request

Required request parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to services”	fa749bace6d8a3b1....
latlon	float,float	Latitude and longitude (in degrees) of the geometric center of the desired map image. Latitude and longitude are separated by commas <i>When specifying bbox in the request, the values specified in latlon are ignored</i>	latlon=55.727,37.59
bbox	float,float,float,float	Object area, which is described by a pair of coordinates in the order: lat1,lon1,lat2,lon2 The latitude and longitude of the coordinates are specified separated by commas. Coordinates are separated by comma	bbox=55.7,37.65,55.8,37.66

Additional request parameters

Field name	Format	Description	Example
zoom	integer	Zoom level ranging from 0 to 17, where <ul style="list-style-type: none"> • 0 (default) — corresponds to the world level of view; • 17 — corresponds to the building level of view. 	<code>zoom=13</code>
width	integer	Map image width. The value can range from 32 to 1024 pixels (default 512)	<code>width=640</code>
height	integer	Map image height. The value can range from 32 to 1024 pixels (default 512)	<code>height=480</code>
pins	string	Options that determine the location and type of pins added to the map. These options must be passed in the below format: lat1,lon1,icon1 lat2,lon2,icon2 ..., where <ul style="list-style-type: none"> • lat1...latN — pin location latitude in degrees; • lon1...lonN — pin location longitude in degrees; • icon1...iconN - pin format. The available pins formats are represented in the pin collection	<code>pins=55.7505,37.6165,blue_star 60.6543,38.1255,red_camera</code>
style	string	Map style selection option. The default style is: main	<code>style=light</code>
padding	integer	Map attribute shift option. The shift is defined in pixels and is equal to the distance from the right edge of the label to the right edge of the attributes. Restrictions: value can range from 5 to width/2 pixels (default 5)	<code>padding=40</code>

<p>scale</p>	<p>integer</p>	<p>Image scaling option. Possible values are 1 or 2 (default 1). If scale=2, then the final image dimensions will be 2 times larger due to the higher pixel density. For example, if you request width=200&height=200&scale=2, the resulting image will be 400x400 pixels, though it will look like the request width=200&height=200, not width=400&height=400:</p>  <p>width=200&height=200</p> <p>width=200&height=200&scale=2</p> <p>width=400&height=400</p>	<p>scale=2</p>
--------------	----------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------

Request (POST)

Required request parameters (GET)

Field name	Format	Description	Example
api_key	hex-string	See "Access to services"	<code>fa749bace6d8a3b1....</code>

Required request parameters (POST)

These parameters are accepted in the request body as part of a JSON object

Field name	Format	Description	Example
coord	coord — JSON object with float fields "lat" and "lon"	Latitude and longitude (in degrees) of the geometric center of the desired map image. <i>When specifying "bbox" in the request, the values specified in "coord" are ignored</i>	<pre>{ "lat": 55.727, "lon": 37.59 }</pre>
bbox	JSON array of 2 elements	Object area, which is described by a pair of coordinates.	<pre>"bbox": [{ "lat": 55.71, "lon": 37.65 }, { "lat": 55.8, "lon": 37.6535 }]</pre>

Additional request parameters

Field name	Format	Description	Example
------------	--------	-------------	---------

zoom	integer	Zoom level ranging from 0 to 17, where <ul style="list-style-type: none"> • 0 (default) — corresponds to the world level of view; • 17 — corresponds to the building level of view. 	"zoom": 13
width	integer	Map image width. The value can range from 32 to 1024 pixels (default 512)	"width": 640
height	integer	Map image height. The value can range from 32 to 1024 pixels (default 512)	"height": 480
pins	JSON array	Description of pins added to the map	<pre> "pins": [{ "coord": { "lat": 55.73, "lon": 37.59 }, "icon": { "symbol": "rustore-corp_photo" } }, { "coord": { "lat": 55.76, "lon": 37.59 }, "icon": { "base64": "iVBORw0KGgoA"... } }], { "coord": { "lat": 55.745, "lon": 37.67 }, "icon": { </pre>
coord	JSON array	Specifies the location of pins added to the map	
icon	JSON array	Description of pins appearance added to the map	
symbol	string	The "symbol" field contains the pin's visual format The available pins formats are represented in the pin collection	
base64	string	Base64 encoded image of a pin in PNG format (maximum 250k characters)	

url	string	PNG image URL. URL length must not exceed 1000 characters, image size must not exceed 512 KB. Only HTTPS transport, uncompressed transfer with no redirects.	<pre> "url": { "url": "https://geo.rustore.ru/welcome/static-map-logo-50x50.png" }] </pre>
features	GeoJSON object	GeoJSON describing the geometry to display on the map top layer. properties is optional.	<pre> "features": { "type": "FeatureCollection", "features": [{ "type": "Feature", "geometry": { "type": "Point", "coordinates": [37.6165, 55.7505] }, "properties": { "title": "points" } }, { "type": "Feature", "geometry": { "type": "LineString", "coordinates": [[37.6163, 55.7503], [37.6164, 55.7504], [37.6165, 55.7505], [37.6166, 55.7506]] }, "properties": { "title": "line" } }, { "type": "Feature", "geometry": { "type": "Polygon", "coordinates": [[[37.6395, 55.73], [37.6066, 55.73], [37.6066, 55.751], [37.6395, 55.751], [37.6395, 55.73]], </pre>

			<pre> [[[37.6345, 55.735], [37.6345, 55.746], [37.6116, 55.746], [37.6116, 55.735], [37.6345, 55.735]]], }, "properties": { "title": "polygon" } }] } </pre>
features-style	JSON object	<p>Sets the visual display of GeoJSON passed to features. Visualization of points, lines and polygons is configured separately:</p> <ul style="list-style-type: none"> ● point — point settings. The points are displayed as circles. <ul style="list-style-type: none"> ○ circle-color — circle color ○ circle-opacity — circle transparency ○ circle-radius — circle radius in pixels ● line — line settings <ul style="list-style-type: none"> ○ line-width — line thickness in pixels ○ line-color — line color ○ line-opacity — line transparency ● polygon — polygon settings <ul style="list-style-type: none"> ○ fill-color — fill color ○ fill-opacity — fill transparency 	<pre> "features-style": { "point": { "circle-color": "#ffffff", "circle-opacity": 0.8, "circle-radius": 4.0 }, "line": { "line-color": "#2688eb", "line-width": 2.0 }, "polygon": { "fill-color": "#2688eb", "fill-opacity": 0.5 } } </pre>

style	string	Map style selection option. The default style is: main	<code>"style": "light"</code>
padding	integer	Map attribute shift option. The shift is defined in pixels and is equal to the distance from the right edge of the label to the right edge of the attributes. Restrictions: value can range from 5 to width/2 pixels (default 5)	<code>"padding": 40</code>
scale	integer	Image scaling option. Possible values are 1 or 2 (default 1).	<code>"scale": 2</code>

Response

In response, you will receive an image of a part of the map that matches the parameters specified in the request.

Example

Request (GET)

```
https://geo.rustore.ru/api/staticmap/png?api_key=<YOUR_API_KEY>&latlon=55.727,37.59&style=main&zoom=10&width=1024&height=512&padding=5&pins=55.73,37.59,rustore-corp_photo|55.76,37.59,green_star|55.745,37.67,rustore-electric_a
```

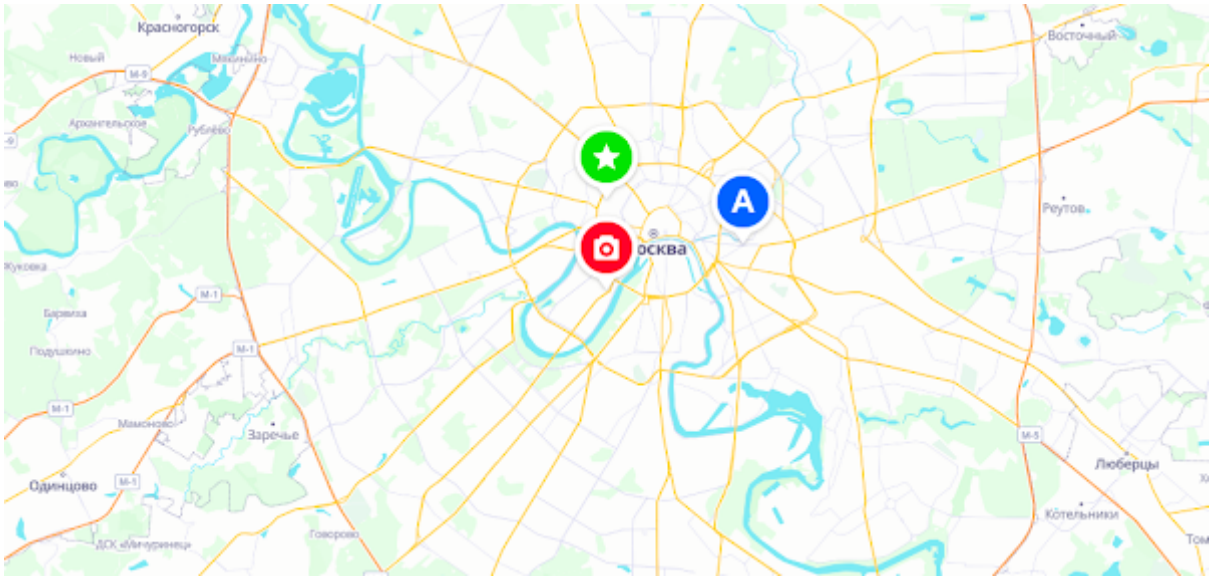
Request (POST)

```
{
  "width": 660,
```

```
"height": 600,
"bbox": [{
  "lat": 55.71,
  "lon": 37.65
}, {
  "lat": 55.8,
  "lon": 37.6535
}
],
"padding": 200,
"scale": 1,
"features": {},
"coord": {
  "lat": 55.7505,
  "lon": 37.6165
},
"pins": [
  {
    "coord": {
      "lat": 55.73,
      "lon": 37.59
    },
    "icon": {
      "symbol": "rustore-corp_photo"
    }
  },
  {
    "coord": {
      "lat": 55.76,
      "lon": 37.59
    },
    "icon": {
      "symbol": "green_star"
    }
  },
  {
    "coord": {
      "lat": 55.745,
      "lon": 37.67
    },
    "icon": {
```

```
        "symbol": "rustore-electric_a"  
      }  
    ],  
    "zoom": 11,  
    "style": "main"  
  }  
}
```

Response



Pin collection

Icon name (icon request parameter) is generated as "color"+"_"+"type". icon=color_image

For example, you can use icon=red_camera if you need a red pin with a camera image.

*When using a static map imager service, you can only use predefined pins.
If you need to add additional pins for your project, please contact RuStore support.*

Pin types

Icons

	color										
image	blue	mint	green	violet	gray	purple	pink	red	orange	mail-electric	mail-corp
target											
photo											
star											
pin											
info											
person											
warning											

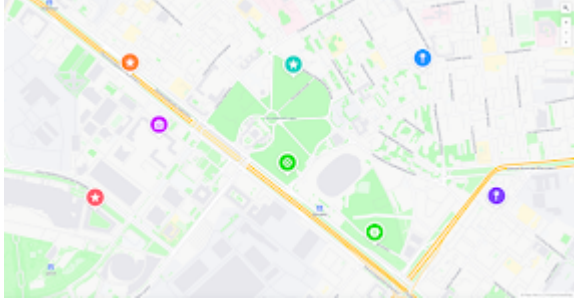
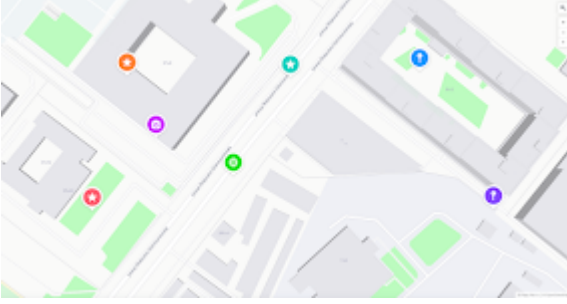
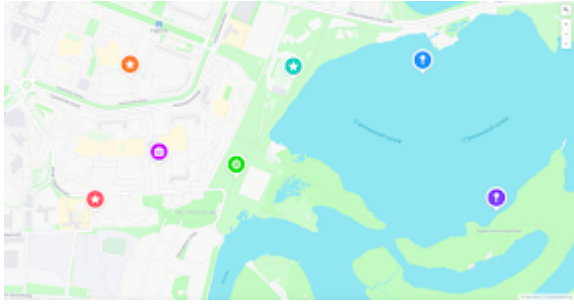
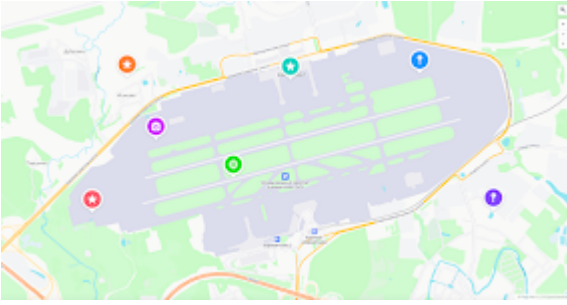
Digits

	color										
image	blue	mint	green	violet	gray	purple	pink	red	orange	mail-electric	mail-corp
1											
2											
3											
4											
5											
6											
7											
8											
9											
0											

Alphabet










Image	color										
	blue	mint	green	violet	gray	purple	pink	red	orange	mail-electric	mail-corp
a											
b											
b											
d											
e											
f											
g											
h											
i											
j											
k											
l											
m											
n											
o											
p											
q											
r											
s											
t											
u											
v											
w											
x											
y											
z											

Example view

Regular view	Close up view
 A satellite-style map showing a city area with several colored markers (red, orange, purple, green, blue) placed on various buildings and streets. A prominent yellow road runs diagonally across the scene.	 A zoomed-in view of the same city area, focusing on the buildings and streets around the markers. The yellow road is still visible.
Water	Airport + overlay
 The same city area as the previous maps, but with a large, irregular blue area overlaid on the right side, representing a body of water. The markers are still present.	 The same city area as the previous maps, but with a large, irregular purple area overlaid on the right side, representing an airport. The markers are still present.

Map styles

Currently, several map styles are available to be used with services. The list is constantly updated. If none of the styles meets your requirements, then we can develop a custom style for your project.

Style	zoom level = 5	zoom level = 10	zoom level = 15
light			
main			
dark			

Routing Services

Routing at the RuStore is currently represented by the following services:

- route planner;
- distance matrix calculation;
- reachable area calculation;
- best route planner.

Route Planner

Route Planner allows you to create car, bicycle and pedestrian routes between two points, with the ability to specify intermediate points of the route.

/directions — call point of the route planner which enables you to estimate several alternative routes in one request and supports traffic jams monitoring.

Request

An HTTP request is sent in JSON and consists of required and optional fields. Required parameters must be specified in the request url.

A simple JSON request example:

```
{
  "locations": [
    {
      "lat": 43.133200,
      "lon": 131.911300
    },
    {
      "lat": 50.266000,
      "lon": 127.535600
    }
  ],
  "costing": "auto",
  "costing_options": {
    "auto": {
      "use_border_crossing": 0
    }
  },
  "directions_options": {
    "units": "kilometers"
  },
  "id": "my_route"
}
```

The above request makes up a route between Vladivostok and Blagoveshchensk. It is being created to avoid a route through China, imposing a cross-border fine. The received route is displayed in kilometers.

Required request url-parameters

Parameter name	Format	Description	Example
api_key	hex-string	See "Access to services"	fa749bace6d8a3b1....

Required JSON fields in an HTTP request

Field name	Format	Description	Example
locations	list	<p>List of points to make up a route in a JSON array.</p> <p>The route between the points is built according to the request.</p> <p>Each point is set by parameters, which are described below</p>	<pre>"locations": [{ "lat":55.796932, "lon":37.537849, "heading":150 }, { "lat":55.865625, "lon":37.462290, "type":"via" }, { "lat":55.962139, "lon":37.406377 }]</pre>
lat	float	Route point latitude in degrees, 6 decimal places	"lat":55.796932
lon	float	Route point longitude in degrees, 6 decimal places	"lon":37.537849

Extra JSON fields in an HTTP request

Field name	Format	Description	Example
type	string	<p>Route point type. This optional request field affects two parameters: the ability to turn around at a given point and to create a separate reference branch in the legs list:</p> <ul style="list-style-type: none"> ● break (by default) — U-turns are allowed, a separate guiding branch will be created for this point in the legs list ● through — U-turns are not allowed, a separate guiding line will not be created for this point in the legs list ● via — U-turns are allowed, a separate guiding line will not be created for this point in the legs list ● break_through — U-turns are not allowed, a separate reference branch will be created for this point in the legs list <p><i>The types for the first and last route points should be ignored and always considered equal to break</i></p>	<pre>"type":"break" "type":"via"</pre>
heading	float	<p>Preferred driving direction at start (optional request parameter).</p> <p>The direction is indicated in degrees from north clockwise, where north — 0°, east — 90°, south — 180°, west — 270°</p>	<pre>"heading":150</pre>
costing	string	<p>Transport type for planning a route:</p> <ul style="list-style-type: none"> ● auto (default) — automobile; ● truck — cargo transport; ● pedestrian — pedestrian route; ● bicycle — bicycle route; ● taxi — taxis and other vehicles allowed to use public transport lanes. 	<pre>"costing":"pedestrian"</pre>

costing_options	dict	<p>List of route calculation parameters. Different transport types have various options and restrictions.</p> <p>When using route calculation parameters, you must specify the type of transport for which are subject to the above parameters:</p> <ul style="list-style-type: none"> ● auto (default) — automobile; ● truck — cargo transport; ● pedestrian — pedestrian route; ● bicycle — bicycle route; ● taxi — taxis and other vehicles allowed to use public transport lanes. 	<pre>"costing_options": { "auto": { "use_border_crossing":0, "use_tolls":0 } }</pre>
units	string	<p>Distance unit in response:</p> <ul style="list-style-type: none"> ● kilometers (by default) — kilometers; ● miles — miles. 	<pre>"units":"miles"</pre>
language	string	<p>Response language:</p> <ul style="list-style-type: none"> ● ru-RU (by default) — Russian; ● en-US — English. 	<pre>"language":"en-US"</pre>
id	string	<p>Request ID that is returned with the response, ensuring the exact match between the request and the response.</p>	<pre>"id":"route_to_airport"</pre>

directions_type	<ul style="list-style-type: none"> ● none ● maneuvers ● instructions 	<p>Enable description of Maneuvers.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● none (by default) — exclude Description of maneuvers from the response; ● maneuvers — include Description of maneuvers in the response. ● instructions — add instructions in the corresponding language to the Description of maneuvers 	<pre>"directions_type": "instructions"</pre>
avoid_locations	list	<p>A set of locations to avoid when building a route. Specified as an array of points with latitude and longitude</p>	<pre>"avoid_locations": [{ "lat":55.871899, "lon":37.457765 }, { "lat":55.884556, "lon":37.441633 }, { "lat":55.923995, "lon":37.395115 }]</pre>

date_time	list	<p>Date and time at starting point or destination point to determine more accurate route results.</p> <p>It is defined by two parameters:</p> <ul style="list-style-type: none"> ● type — date and time type: <ul style="list-style-type: none"> ○ 0 (by default) — current date and time at the point of departure, the value is ignored; ○ 1 — date and time of departure; ○ 2 — date and time of arrival. ● value — value of the required date and time is specified in ISO 8601 (YYYY-MM-DDThh:mm) in the local time zone of the departure or arrival, depending on the type parameter. 	<pre>"date_time": { "type":2, "value":"2020-12-33T21:00" }</pre>
alternates	integer	<p>Maximum number of alternative routes in addition to the main one (returned if any).</p> <p>Defined by a value from 0 to 4, where:</p> <ul style="list-style-type: none"> ● 0 (default) — one route will be built, without alternative ones; ● 4 — five routes will be calculated; <p>— fractional values are not allowed.</p>	<pre>"alternates":3</pre>
alternates_multi_points	boolean	<p>An attribute to enable the creation of alternative routes in case there are intermediate points in the request. The default — false.</p>	<pre>"alternates_multi_points":true</pre>
completeness	string	<p>An attribute for a possible response extension.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● minimal (by default) — minimal details return (only maneuvers, no edges); ● enriched — information about all edges in the route. 	<pre>"completeness":"enriched"</pre>

Costing options

Auto

Field name	Format	Description	Example
use_unpaved	float	<p>A value which indicates whether to include dirt roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none">● 0 — do not include dirt roads;● 1 — include dirt roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	<pre>"use_unpaved":0.5</pre>
use_highways	float	<p>A value which indicates whether to use highways when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none">● 0 — do not use highways;● 1 (default) — use highways; <p>— fractional values are allowed.</p>	<pre>"use_highways":0</pre>
use_tolls	float	<p>A value which indicates whether to include toll roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none">● 0 — do not include toll roads;● 1 — include toll roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	<pre>"use_tolls":0</pre>

use_ferry	float	<p>A value which indicates whether to include ferry crossings when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> • 0 — do not use ferries; • 1 — use ferries; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	<code>"use_ferry":0.1</code>
use_border_crossing	float	<p>A value which indicates whether to include roads that cross the borders of other states when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> • 0 — do not use cross-border routes; • 1 (default) — use cross-border routes; <p>— fractional values are allowed.</p>	<code>"use_border_crossing":0</code>
traffic	boolean	<p>Take into account traffic jams and road events when building a route.</p> <p>Possible values:</p> <ul style="list-style-type: none"> • false (by default) — ignore traffic jams and traffic events; • true — consider traffic jams and traffic events. 	<code>"traffic":true</code>

Truck

Field name	Format	Description	Example
------------	--------	-------------	---------

weight	float	<p>Indication of the car weight in tons, for building a route, taking into account the permissible weight loads on the road network.</p> <p>Default value: 21.77 tons.</p> <p>Road sign number according to GOST: 3.11.</p>	<code>"weight":10</code>
height	float	<p>Truck height in meters.</p> <p>Default value: 4.11 m.</p> <p>Road sign number according to GOST: 3.13.</p>	<code>"height":4.0</code>
width	float	<p>Truck width in meters.</p> <p>Default value: 9.07 m.</p> <p>Road sign number according to GOST: 3.14.</p>	<code>"width":11.1</code>
length	float	<p>Truck length in meters.</p> <p>Default value: 21.64 m.</p> <p>Road sign number according to GOST: 3.15.</p>	<code>"length":19.4</code>
axle_load	float	<p>Axle load in tons.</p> <p>Default value: 9.07 tons.</p> <p>Road sign number according to GOST: 3.12.</p>	<code>"axle_load":10.3</code>
hazmat	boolean	<p>A flag that determines if the truck carries hazardous materials.</p> <p>Default value: false.</p> <p>Road sign number according to GOST: 3.32.</p>	<code>"hazmat":true</code>

use_unpaved	float	<p>A value which indicates whether to include dirt roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not include dirt roads; ● 1 — include dirt roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_unpaved":0.5
use_highways	float	<p>A value which indicates whether to use highways when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use highways; ● 1 (default) — use highways; <p>— fractional values are allowed.</p>	"use_highways":0
use_tolls	float	<p>A value which indicates whether to include toll roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not include toll roads; ● 1 — include toll roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_tolls":0
use_ferry	float	<p>A value which indicates whether to include ferry crossings when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use ferries; ● 1 — use ferries; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_ferry":0.1

use_border_crossing	float	<p>A value which indicates whether to include roads that cross the borders of other states when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use cross-border routes; ● 1 (default) — use cross-border routes; <p>— fractional values are allowed.</p>	<code>"use_border_crossing":0</code>
traffic	boolean	<p>Take into account traffic jams and road events when building a route.</p> <p>Possible values:</p> <ul style="list-style-type: none"> ● false (by default) — ignore traffic jams and traffic events; ● true — consider traffic jams and traffic events. 	<code>"traffic":true</code>

Pedestrian

Field name	Format	Description	Example
walking_speed	float	<p>Average walking speed. Specified in kilometers per hour.</p> <p>Default value: 5.1 km/h</p>	<code>"walking_speed":3</code>
use_ferry	float	<p>A value which indicates whether to include ferry crossings when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use ferries; ● 1 — use ferries; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	<code>"use_ferry":0.1</code>

use_unpaved	float	<p>A value which indicates whether to include dirt roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not include dirt roads; ● 1 — include dirt roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_unpaved":0.5
use_roads	float	<p>A value which indicates whether to use highways when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use highways; ● 1 — use highways; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_road":0
use_border_crossing	float	<p>A value which indicates whether to include roads that cross the borders of other states when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use cross-border routes; ● 1 (default) — use cross-border routes; <p>— fractional values are allowed.</p>	"use_border_crossing":0

use_hills	float	<p>A value which indicates whether to include a route with elevation changes. It is used when selecting a suitable route and in the course of ETA calculation.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — avoid terrain changes as much as possible, even if it leads to building a longer route; ● 1 — do not avoid terrain changes. <p>Fractional values are allowed.</p> <p>Default value: 0.5.</p> <p>It is worth remembering that it is not possible in all cases to build an alternative route given use_hills (for example, if there is only one road leading to the top of the hill).</p>	<code>"use_hills":0.25</code>
step_penalty	float	<p>A value which indicates a penalty in seconds added to each climb or descend stairs/stairs. The higher the value, the more stairs are avoided.</p>	<code>"step_penalty":120.0</code>

Bicycle

Field name	Format	Description	Example
cycling_speed	float	<p>Average cycling speed. Specified in kilometers per hour.</p> <p>Default value: 20 km/h</p>	<code>"cycling_speed":30</code>

use_ferry	float	<p>A value which indicates whether to include ferry crossings when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use ferries; ● 1 — use ferries; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_ferry":0.1
use_unpaved	float	<p>A value which indicates whether to include dirt roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not include dirt roads; ● 1 — include dirt roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_unpaved":0.5
use_roads	float	<p>A value which indicates whether to use highways when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use highways; ● 1 (default) - use highways; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_road":0

use_border_crossing	float	<p>A value which indicates whether to include roads that cross the borders of other states when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use cross-border routes; ● 1 (default) — use cross-border routes; <p>— fractional values are allowed.</p>	<code>"use_border_crossing":0</code>
use_hills	float	<p>A value which indicates whether to include a route with elevation changes. It is used when selecting a suitable route and in the course of ETA calculation.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — avoid terrain changes as much as possible, even if it leads to building a longer route; ● 1 — do not avoid terrain changes. <p>Fractional values are allowed.</p> <p>Default value: 0.5.</p> <p>It is worth remembering that it is not possible in all cases to build an alternative route given use_hills (for example, if there is only one road leading to the top of the hill).</p>	<code>"use_hills":0.25</code>

Taxi

Field name	Format	Description	Example
use_unpaved	float	<p>A value which indicates whether to include dirt roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none">● 0 — do not include dirt roads;● 1 — include dirt roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	<code>"use_unpaved":0.5</code>
use_highways	float	<p>A value which indicates whether to use highways when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none">● 0 — do not use highways;● 1 (default) — use highways; <p>— fractional values are allowed.</p>	<code>"use_highways":0</code>
use_tolls	float	<p>A value which indicates whether to include toll roads when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none">● 0 — do not include toll roads;● 1 — include toll roads; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	<code>"use_tolls":0</code>

use_ferry	float	<p>A value which indicates whether to include ferry crossings when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use ferries; ● 1 — use ferries; <p>— fractional values are allowed.</p> <p>Default value: 0.5</p>	"use_ferry":0.1
use_border_crossing	float	<p>A value which indicates whether to include roads that cross the borders of other states when building a route.</p> <p>Defined by a value from 0 to 1, where:</p> <ul style="list-style-type: none"> ● 0 — do not use cross-border routes; ● 1 (default) — use cross-border routes; <p>— fractional values are allowed.</p>	"use_border_crossing":0

Response

Background

Field name	Format	Description	Example
trips	list	Root list of routes to-be returned. Includes multiple elements when requesting alternative routes. Consists of the trip objects described below	
trip	dict	Dictionary with full route description	
status	number	Error code	"status": 0

status_message	string	Error description	<code>"status_message": "Found 2 route(s) between points"</code>
units	string	Distance unit in response: <ul style="list-style-type: none"> • kilometers — kilometers; • miles — miles. 	<code>"units": "kilometers"</code>
language	string	Response language: <ul style="list-style-type: none"> • ru-RU — Russian; • en-US — English. 	<code>"language": "ru-RU"</code>

locations	list	List of route points as requested with additional information about them	<pre> "locations": [{ "original_index": 0, "type": "break", "lat": 55.796932, "lon": 37.537849, "heading": 150 }, { "original_index": 1, "lon": 37.462292, "lat": 55.865623, "type": "via" }, { "original_index": 2, "lon": 37.406376, "lat": 55.962139, "type": "break" }] </pre>
original_index	integer	Route point serial number	<pre> "original_index": 0 </pre>

type	string	<p>Route point type:</p> <ul style="list-style-type: none"> • break (by default) — U-turns are allowed, a separate guiding branch will be created for this point in the legs list • through — U-turns are not allowed, a separate guiding line will not be created for this point in the legs list • via — U-turns are allowed, a separate guiding line will not be created for this point in the legs list • break_through — U-turns are not allowed, a separate reference branch will be created for this point in the legs list 	<code>"type": "break"</code>
lat	float	Route point latitude in degrees, 6 decimal places	<code>"lat": 55.796932</code>
lon	float	Route point longitude in degrees, 6 decimal places	<code>"lon": 37.537849</code>
heading	float	<p>Preferred driving direction at start.</p> <p>The direction is indicated in degrees from north clockwise, where north — 0°, east — 90°, south — 180°, west — 270°</p>	<code>"heading": 150</code>
id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<code>"id": "route_to_airport"</code>

General info

Field name	Format	Description	Example
summary	dict	General route info	<pre> "summary": { "ll_boxes": [{ "max_lon": 37.546925, "max_lat": 55.962685, "min_lat": 55.793781, "min_lon": 37.406376 }], "time": 2168.574, "length": 29.121, }</pre>
ll_boxes	list	A set of bounding boxes that describe the route area. It generally consists of 1 element. When crossing the route through the antimeridian, two boxes will be returned (on opposite sides of the meridian).	<pre> "ll_boxes": [{ "max_lon": 37.546925, "max_lat": 55.962685, "min_lat": 55.793781, "min_lon": 37.406376 }]</pre>

max_lon	float	Maximum route point boundary longitude that includes the route, 6 decimal places	"max_lon": 37.546925
max_lat	float	Maximum route point boundary latitude that includes the route, 6 decimal places	"max_lat": 55.962685
time	float	Estimated time required to move along the route. Estimated time is specified in seconds.	"time": 2168.574
length	float	Total route length, indicated in the selected units.	"length": 29.121
min_lat	float	Maximum route point boundary longitude that includes the route, 6 decimal places	"min_lat": 55.793781
min_lon	float	Maximum route point boundary latitude that includes the route, 6 decimal places	"min_lon": 37.406376

Route and maneuvers

Field name	Format	Description	Example
------------	--------	-------------	---------

legs	dict	<p>The legs object describes the route and maneuvers between a pair of points that have the property "type":"break"</p> <p>For break-type n route points, the response contains n-1 route description elements (for a request without alternatives). For a request with alternatives, each pair of breakpoints can contain a number of legs less than or equal to the number of alternatives.</p> <p>Each route description object contains:</p> <ul style="list-style-type: none"> • summary dictionary similar to the general one, but referring only to its part between two break-type route points; • line shape; • list of maneuvers. 	<pre> "legs": [{ "shape": "...", "summary": { ... }, "maneuvers": [{ ... }, ...] }] </pre>
shape	string	<p>Polyline encoded format to store a series of latitude and longitude coordinates as a single string (see also: "Polyline path decoding")</p>	<p>see the response example</p>

edges	list	List of edges included in all maneuvers (returned if "completeness":"enriched" in the input request).	<pre>"edges" : [{ "id" : 12345, }, ... { ... }]</pre>
id	int	Edge unique identifier (edge attribute)	<pre>"id" : 3459922</pre>
length	float	Edge length (edge attribute), by default in km, units of measurement are specified in the "units" field	<pre>"length" : 0.314</pre>

use	string	<p>Edge type</p> <ul style="list-style-type: none"> ● road ● ramp ● turn_channel ● track ● driveway ● alley ● parking_aisle ● emergency_access ● drive_through ● culdesac ● living_street ● service_road ● cycleway ● mountain_bike ● sidewalk ● footway ● steps ● path ● pedestrian ● bridleway ● rest_area ● service_area ● other ● rail—ferry ● ferry ● rail ● bus ● egress_connection ● platform_connection ● transit_connection 	<code>"use": "cycleway"</code>
road_class	string	<p>Road class</p> <ul style="list-style-type: none"> ● motorway ● trunk ● primary ● secondary ● tertiary ● unclassified ● residential ● service_other 	<code>"road_class": "secondary"</code>
surface	string	<p>Coating type according to roughness (edge attribute):</p> <ul style="list-style-type: none"> ● paved_smooth ● paved ● paved_rough ● compacted ● dirt ● gravel ● path ● impassable 	<code>"surface": "gravel"</code>

toll	boolean	Sign of entry into the toll road section (edge attribute). Present only for auto, motorcycle, truck, taxi transport types	<code>"toll": true</code>
speed_limit	int	Maximum possible speed in km/h (edge attribute)	<code>"speed_limit": 60</code>
iso_code	string	ISO country code (edge attribute). Specified if the edge is a boundary edge for a cross-boundary route. Format ISO 3166-1 alpha-2 (two letter)	<code>"iso_code": "RU"</code>

maneuvers	list	List of maneuvers indicating the points of maneuvers on the polyline, as well as the length, duration and hints for maneuvers in the selected language	<pre> "maneuvers": [{ "travel_type": "car", "travel_mode": "drive", "verbal_pre_transition_instru ction": "Drive 9.5 km along M-11.", "verbal_transition_alert_inst ruction": "Continue on M-11.", "length": 9.490, "toll": true, "instruction": "Continue on M-11.", "end_shape_index": 459, "type": 8, "time": 421.573, "street_names": ["M-11"], "begin_shape_index": 344 }, ... { ... }] </pre>
travel_mode	string	Routing mode: <ul style="list-style-type: none"> ● drive — for all transports «costing»=«auto» and «costing»=«truck» ● pedestrian ● bicycle 	<pre> "travel_mode": "drive" </pre>

travel_type	string	<p>Routing type.</p> <ul style="list-style-type: none"> • car — for «travel_mode»=«drive» and «costing»=«auto» • tractor_trailer — for «travel_mode»=«drive» and «costing»=«truck» • foot — for «travel_mode»=«pedestrian» <p>Possible values for cycling navigation are:</p> <ul style="list-style-type: none"> • hybrid, road, cross, mountain. 	<code>"travel_type": "car"</code>
verbal_pre_transition_instruction	string	Text that can be used as a verbal message for a driver.	<code>"verbal_pre_transition_instruction": "Drive 9.5 km along M-11."</code>
verbal_transition_alert_instruction	string	Text that can be used as a verbal message for a driver just before the actual maneuver.	<code>"verbal_transition_alert_instruction": "Enter the roundabout and take the second exit."</code>
verbal_multi_cue	boolean	The variable has the value "true" in cases where the verbal message text "verbal_pre_transition_instruction" contains an indication of several successive, closely spaced maneuvers	<code>"verbal_multi_cue": true</code>
verbal_post_transition_instruction	string	Text that can be used as a verbal message immediately after the maneuver completion	<code>"verbal_post_transition_instruction": "Continue moving 100 meters."</code>
verbal_succinct_transition_instruction	string	Text that describes the maneuver briefly	

instruction	string	Text to-be displayed with a hint about the maneuver	<code>"instruction": "Take the left exit onto Leningradsky Prospekt."</code>
length	float	Total maneuver length within the boundaries between "begin_shape_index" and "end_shape_index", indicated in the selected units of measurement	<code>"length": 9.490</code>
begin_shape_index	integer	Pointer to the maneuver start on the polyline	<code>"begin_shape_index": 344</code>
end_shape_index	integer	Pointer to the maneuver end on the polyline	<code>"end_shape_index": 459</code>
begin_edge_index	integer	First edge index from the returned edges array to start the maneuver (returned if "completeness":"enriched" is present in the input request)	<code>"begin_edge_index": 5</code>
end_edge_index	integer	First edge index from the returned edges array to end the maneuver (returned if "completeness":"enriched" is present in the input request)	<code>"end_edge_index": 9</code>
type	integer	See below "Maneuver type code"	<code>"type": 8</code>

time	float	Estimated maneuver time within the boundaries between "begin_shape_index" and "end_shape_index". Time is estimated in seconds	"time": 421.573
cost	float	Maneuver or route difficulty level in some abstract units. Allows you to compare routes and maneuvers with each other and evaluate which one is more "demanding" in terms of time, distance, etc.	
street_names	list	List of maneuver street names	"street_names": ["M-11"]
toll	boolean	Variable is "true" if the maneuver or part of it is payable. Example: part of the maneuver is on a toll road	"toll": true
ferry	boolean	Variable is "true" if the maneuver is not possible without using the ferry	"ferry": true
rough	boolean	Variable is "true" if the maneuver is completely or partially carried out on unpaved roads	"rough": true
gate	boolean	Variable is "true" if the maneuver crosses gates/barriers	"gate": true

roundabout_ exit_count	integer	Reference number of the roundabout exit	"roundabout_exit_count": 2
---------------------------	---------	--------------------------------------------	----------------------------

"type" value for maneuvers

"type" value	Decryption
0	kNone
1	kStart
2	kStartRight
3	kStartLeft
4	kDestination
5	kDestinationRight
6	kDestinationLeft
7	kBecomes
8	kContinue
9	kSlightRight
10	kRight
11	kSharpRight
12	kUturnRight
13	kUturnLeft
14	kSharpLeft
15	kLeft
16	kSlightLeft
17	kRampStraight
18	kRampRight
19	kRampLeft
20	kExitRight
21	kExitLeft
22	kStayStraight

23	kStayRight
24	kStayLeft
25	kMerge
26	kRoundaboutEnter
27	kRoundaboutExit
28	kFerryEnter
29	kFerryExit
30	kTransit
31	kTransitTransfer
32	kTransitRemainOn
33	kTransitConnectionStart
34	kTransitConnectionTransfer
35	kTransitConnectionDestination
36	kPostTransitConnectionDestination
37	kMergeRight
38	kMergeLeft

Route not found

In case of zero request result, the response will look like this:

```
{"error_code":171,"error":"No suitable edges near location", "status_code":400, "status":"Bad Request"}
```

Example

Request

```
curl -X POST \  
  -H "Content-type: application/json" \  
  -H "Accept: application/json" \  
  -d \  
'{"locations":[{"lat":55.796932,"lon":37.537849,"heading":150},{"lat":  
55.865625,"lon":37.462290,"type":"via"}, {"lat":55.962139,"lon":37.4063  
77}], "costing": "auto", "language": "ru-RU", "directions_type": "instructio  
ns", "id": "route_to_airport"}' \  
  "https://geo.rustore.ru/api/directions?api_key=<YOUR_API_KEY>"
```

Response

```
{  
  "trips": [  
    {  
      "trip": {  
        "locations": [  
          {  
            "type": "break",  
            "lat": 55.796932,  
            "lon": 37.537849,  
            "heading": 150,  
            "city": "left",  
            "original_index": 0  
          },  
          {  
            "type": "via",  
            "lat": 55.865625,  
            "lon": 37.46229,  
            "original_index": 1  
          },  
          {  
            "type": "break",  
            "lat": 55.962139,  
            "lon": 37.406377,  
            "original_index": 2  
          }  
        ],  
      }  
    ],  
  }  
}
```

```

    "legs":[
      {
        "maneuvers":[
          {
            "type":3,
            "instruction":"Head northeast.",
            "verbal_succinct_transition_instruction":"Drive north-west, then turn
            right onto Leningradsky Prospekt.",
            "verbal_pre_transition_instruction":"Drive
            north-west, then turn right onto Leningradsky Prospekt.",
            "verbal_post_transition_instruction":"Continue
            driving for another 50 meters.",
            "time":8.72,
            "length":0.048,
            "cost":364.824,
            "begin_shape_index":0,
            "end_shape_index":3,
            "verbal_multi_cue":true,
            "travel_mode":"drive",
            "travel_type":"car"
          },
          {
            "type":10,
            "instruction":"Turn right onto Leningradsky
            Prospekt.",
            "verbal_transition_alert_instruction":"Turn
            right onto Leningradsky Prospekt.",
            "verbal_succinct_transition_instruction":"Turn
            right.",
            "verbal_pre_transition_instruction":"Turn
            right onto Leningradsky Prospekt.",
            "verbal_post_transition_instruction":"Continue
            driving for another 600 meters.",
            "street_names":[
              "Leningradsky Prospekt"
            ],
            "time":37.931,
            "length":0.593,
            "cost":72.21,
            "begin_shape_index":3,

```

```

        "end_shape_index":12,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":13,
        "instruction":"Turn left to keep dring along
Leningradsky Prospekt.",
        "verbal_transition_alert_instruction":"Turn
left to keep dring along Leningradsky Prospekt.",
        "verbal_succinct_transition_instruction":"Turn
left.",
        "verbal_pre_transition_instruction":"Turn left
to keep dring along Leningradsky Prospekt.",
        "verbal_post_transition_instruction":"Continue
driving for another 10 km.",
        "street_names":[
            "Leningradsky Prospekt"
        ],
        "time":535.509,
        "length":10.369,
        "cost":682.537,
        "begin_shape_index":12,
        "end_shape_index":164,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":23,
        "instruction":"Keep right at the fork.",
        "verbal_transition_alert_instruction":"Keep
right at the fork.",
        "verbal_pre_transition_instruction":"Keep
right at the fork.",
        "verbal_post_transition_instruction":"Continue
driving for another 400 meters.",
        "time":34.122,
        "length":0.378,
        "cost":55.773,
        "begin_shape_index":164,
        "end_shape_index":174,

```

```

        "travel_mode": "drive",
        "travel_type": "car"
    },
    {
        "type": 15,
        "instruction": "Turn left.",
        "verbal_transition_alert_instruction": "Turn
left.",
        "verbal_succinct_transition_instruction": "Turn
left.",
        "verbal_pre_transition_instruction": "Turn
left.",
        "verbal_post_transition_instruction": "Continue
driving for another 400 meters.",
        "time": 45.832,
        "length": 0.435,
        "cost": 75.448,
        "begin_shape_index": 174,
        "end_shape_index": 195,
        "travel_mode": "drive",
        "travel_type": "car"
    },
    {
        "type": 9,
        "instruction": "Turn right onto Leningradskoye
highway.",
        "verbal_transition_alert_instruction": "Turn
right onto Leningradskoye highway.",
        "verbal_succinct_transition_instruction": "Turn
right, then after 300 meters, Turn right.",
        "verbal_pre_transition_instruction": "Turn
right onto Leningradskoye highway, then after 300 meters, Turn
right.",
        "verbal_post_transition_instruction": "Continue
for another 300 meters.",
        "street_names": [
            "Leningradskoye highway"
        ],
        "time": 12.615,
        "length": 0.289,
        "cost": 28.807,

```

```

        "begin_shape_index":195,
        "end_shape_index":199,
        "verbal_multi_cue":true,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":9,
        "instruction":"Turn right.",
        "verbal_transition_alert_instruction":"Turn
right.",
        "verbal_succinct_transition_instruction":"Turn
right, then after 50 meters, turn right.",
        "verbal_pre_transition_instruction":"Turn
right, then after 50 meters, turn right.",
        "verbal_post_transition_instruction":"Continue
for another 50 meters.",
        "time":9.245,
        "length":0.05,
        "cost":91.808,
        "begin_shape_index":199,
        "end_shape_index":201,
        "verbal_multi_cue":true,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":9,
        "instruction":"Turn right.",
        "verbal_transition_alert_instruction":"Turn
right.",
        "verbal_succinct_transition_instruction":"Turn
right.",
        "verbal_pre_transition_instruction":"Turn
right.",
        "verbal_post_transition_instruction":"Continue
for another 100 meters.",
        "time":40.258,
        "length":0.123,
        "cost":759.908,
        "begin_shape_index":201,

```

```

        "end_shape_index":213,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":10,
        "instruction":"Turn right.",
        "verbal_transition_alert_instruction":"Turn
right.",
        "verbal_succinct_transition_instruction":"Turn
right.",
        "verbal_pre_transition_instruction":"Turn
right.",
        "verbal_post_transition_instruction":"Continue
for another 100 meters.",
        "time":24.898,
        "length":0.127,
        "cost":36.26,
        "begin_shape_index":213,
        "end_shape_index":217,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":15,
        "instruction":"Turn left.",
        "verbal_transition_alert_instruction":"Turn
left.",
        "verbal_succinct_transition_instruction":"Turn
left, then turn right onto Leningradskoe highway.",
        "verbal_pre_transition_instruction":"Turn
left, then turn right onto Leningradskoe highway.",
        "verbal_post_transition_instruction":"Continue
for another 20 meters.",
        "time":8.515,
        "length":0.018,
        "cost":22.563,
        "begin_shape_index":217,
        "end_shape_index":218,
        "verbal_multi_cue":true,
        "travel_mode":"drive",

```



```

        "travel_type": "car"
    },
    {
        "type": 10,
        "instruction": "Turn right onto Leningradskoye highway/Leningrad highway.",
        "verbal_transition_alert_instruction": "Turn right onto Leningradskoye highway.",
        "verbal_succinct_transition_instruction": "Turn right, then enter the roundabout and take the 3rd exit onto Belomorskaya Street.",
        "verbal_pre_transition_instruction": "Turn right onto Leningradskoye Shosse, Leningrad Avenue, then go to the roundabout and take the 3rd exit onto Belomorskaya Street.",
        "verbal_post_transition_instruction": "Continue for another 80 meters.",
        "street_names": [
            "Leningradskoe highway",
            "Leningradskoe highway"
        ],
        "time": 5.75,
        "length": 0.081,
        "cost": 20.979,
        "begin_shape_index": 218,
        "end_shape_index": 220,
        "verbal_multi_cue": true,
        "travel_mode": "drive",
        "travel_type": "car"
    },
    {
        "type": 26,
        "instruction": "Enter the roundabout and take the 3rd exit onto Belomorskaya Street.",
        "verbal_transition_alert_instruction": "Enter the roundabout and take the 3rd exit onto Belomorskaya Street.",
        "verbal_succinct_transition_instruction": "Enter the roundabout and take the 3rd exit.",
        "verbal_pre_transition_instruction": "Enter the roundabout and take the 3rd exit onto Belomorskaya Street.",
        "time": 20.267,

```

```

        "length":0.255,
        "cost":19.688,
        "begin_shape_index":220,
        "end_shape_index":239,
        "roundabout_exit_count":3,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":27,
        "instruction":"Exit the roundabout onto
Belomorskaya Street.",
        "verbal_succinct_transition_instruction":"Leave the roundabout.",
        "verbal_pre_transition_instruction":"Exit the
roundabout onto Belomorskaya Street.",
        "verbal_post_transition_instruction":"Continue
for another 200 meters.",
        "street_names":[
            "Belomorskaya street"
        ],
        "time":14.073,
        "length":0.189,
        "cost":24.333,
        "begin_shape_index":239,
        "end_shape_index":244,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":20,
        "instruction":"Take the exit to Leningradskoye
highway.",
        "verbal_transition_alert_instruction":"Take
the exit to Leningradskoye highway.",
        "verbal_pre_transition_instruction":"Take the
exit to Leningradskoye highway.",
        "verbal_post_transition_instruction":"Continue
for another 8 km.",
        "street_names":[
            "Leningradskoe highway"

```

```

    ],
    "time":394.793,
    "length":8.069,
    "cost":500.92,
    "begin_shape_index":244,
    "end_shape_index":355,
    "sign":{
      "exit_branch_elements":[
        {
          "text":"Leningradskoe highway"
        }
      ]
    },
    "travel_mode":"drive",
    "travel_type":"car"
  },
  {
    "type":23,
    "instruction":"Keep right towards
Sheremetyevo-2.",
    "verbal_transition_alert_instruction":"Keep
right towards Sheremetyevo-2.",
    "verbal_pre_transition_instruction":"Keep
right towards Sheremetyevo-2.",
    "verbal_post_transition_instruction":"Continue
for another 600 meters.",
    "street_names":[
      "46H-13925"
    ],
    "time":34.818,
    "length":0.578,
    "cost":38.3,
    "begin_shape_index":355,
    "end_shape_index":372,
    "sign":{

    },
    "travel_mode":"drive",
    "travel_type":"car"
  },
  {

```

```

        "type":24,
        "instruction":"Keep left to exit onto
46H-13925/Mezhdunarodnoe highway.",
        "verbal_transition_alert_instruction":"Keep
left to exit onto 46H-13925.",
        "verbal_pre_transition_instruction":"Keep left
to exit onto 46H-13925, Mezhdunarodnoe highway.",
        "verbal_post_transition_instruction":"Continue
for another 2.5 km.",
        "street_names":[
            "46H-13925",
            "Mezhdunarodnoe highway"
        ],
        "time":89.576,
        "length":2.654,
        "cost":75.352,
        "begin_shape_index":372,
        "end_shape_index":384,
        "travel_mode":"drive",
        "travel_type":"car"
    },
    {
        "type":24,
        "instruction":"Keep left to stay on
46H-13925/Mezhdunarodnoe highway.",
        "verbal_transition_alert_instruction":"Keep
left to stay on 46H-13925.",
        "verbal_pre_transition_instruction":"Keep left
to stay on 46H-13925, Mezhdunarodnoe highway.",
        "verbal_post_transition_instruction":"Continue
driving for another 1.5 km.",
        "street_names":[
            "46H-13925",
            "Mezhdunarodnoe highway"
        ],
        "time":50.683,
        "length":1.254,
        "cost":46.611,
        "begin_shape_index":384,
        "end_shape_index":403,
        "travel_mode":"drive",

```

```

        "travel_type": "car"
    },
    {
        "type": 24,
        "instruction": "Keep left towards Sheremetyevo,
Terminal D.",
        "verbal_transition_alert_instruction": "Keep
left towards Sheremetyevo.",
        "verbal_pre_transition_instruction": "Keep left
towards Sheremetyevo, Terminal D.",
        "verbal_post_transition_instruction": "Continue
for another 500 meters.",
        "time": 33.139,
        "length": 0.46,
        "cost": 38.839,
        "begin_shape_index": 403,
        "end_shape_index": 413,
        "sign": {

        },
        "travel_mode": "drive",
        "travel_type": "car"
    },
    {
        "type": 23,
        "instruction": "Keep right at the fork.",
        "verbal_transition_alert_instruction": "Keep
right at the fork.",
        "verbal_pre_transition_instruction": "Keep
right at the fork.",
        "verbal_post_transition_instruction": "Continue
for another 600 meters.",
        "time": 43.991,
        "length": 0.61,
        "cost": 63.974,
        "begin_shape_index": 413,
        "end_shape_index": 439,
        "travel_mode": "drive",
        "travel_type": "car"
    },
    {

```

```

        "type":4,
        "instruction":"You have reached your
destination.",
        "verbal_transition_alert_instruction":"You
have reached your destination.",
        "verbal_pre_transition_instruction":"You have
reached your destination.",
        "time":0.0,
        "length":0.0,
        "cost":0.0,
        "begin_shape_index":439,
        "end_shape_index":439,
        "travel_mode":"drive",
        "travel_type":"car"
    }
],
"summary":{
    "ll_boxes":[{
        "min_lat":55.79378,
        "min_lon":37.39366,
        "max_lat":55.962686,
        "max_lon":37.546925,
    }],
    "has_time_restrictions":false,
    "time":1444.743,
    "length":26.583,
    "cost":3019.145
},

```

```

"shape": "ydqliBeqcrfA_KyO_AyA{EsHv^aeA~FkPl[g_Ad[q~@nO}d@bt@stBtM}_@`C
iHvUmr@zAsF\\yEBcEKyDg@sDgAcDgKyPaBsA_Bc@iBEuBj@oBjBoDhFu0hf@mWlu@kCzH
}d`tA}f@fxAwTlo@mk@`bBuU|q@sKh[uCnIcAtCoG1TwTzu@aDlJm}@xoC{Ujq@{Off@i
Nza@yGpUuFxFxQoHv]wGz]mGj_@qFr^mHrf@o^nyBiLzr@}SdkAeO`z@{BhMgK`p@wG`i@ka
@pmC{QphAec@t1DoGfe@mFt^c`@r`C_I|e@aUfuAsJj1@wGn`@qDbSgDfPsCxLsEjPmDnK
mFvNmBvFQj@eIjPcGjLyFnJiEvGeE~FeEjFiFvFwFhFaEjDiEbDmZbTyCdCyHnFc`@bYmk
DdlCag@x_@WRa1BxxA}n@vf@q1CzoBcBhAic@b\\{eD|dCao@fd@qj@b^}b@p[snAp~@u|
@dq@}jAjbAs`@j^ii@pc@kGhFmmAv`A{a@n[uq@`j@mv@|k@}a@n[e`@`Zab@x[aMxJyaA
zv@{`BfnAwu@pj@arC~mBqbAdr@{fBtiAoFjDad@rYak@t^aQxKsy@vf@{e@|Xwo@l`@gn
A|v@oo@f`@ak@`]iz@ng@aZbRuk@r^}cBzcAup@|`sf@hZs^xTkiAls@m_Avj@ie@xXyR
tLcQ1K_h@h[yLnHcTzMkv@xc@}nAhu@mm@f_@kAt@{WdQ_[tRa@T_h@p[qNjIcBbAehAhp
@wh@v[mV|Ncu@lc@oj@t]_YvPsVb0{Vf0cADkHVcs@dPs1Es\\~F}GnBoFlCmFlDiJ|Jy

```

```
q@rw@xBrMjDxN`FrOdShm@jAdC`CnBfCl@~CKd0oFrHsF~HsGhIaJ~C{DhKsJtH{KhGy0h
Tar@z@oCbFkSlCuHnE}IjAy@jNmJzu@ke@x1Aiv@tT~@rEQlAdARpxCtCvC|KrA~JvAlAz
LmHf@kDmB}N@_FsAyJWmBlOuJh]ySpJ_GdGwDqBy0h\_\_SbLyG`Hq@`ET`Dd@xCx@pnAn\
\dCfA|@x@d@Fx@Kv@u@t@qBb@{CPoDAgEUeDg@{Ci@_C_AaCc@YuAgFcAaIcEyB@}Eme@o
NowAk@ePF{H~@{DlCgGnC_DtC{@bCMxCn@|GjFlDtJxFnOz@hCnCzGfJjW|BxGXpCLdC?r
CK|C_@lCaBnF}B`DkAt@{WdQ_[tRa@T_h@p[qNjIcBbAehAhp@wh@v[mV|Ncu@lc@oj@t]
_YvPsVb0{Vf0yI~EcWh0a0jJ}`@j[aKtKk\|\|]aW`]kVd_@gGhJyo@ddAsdCrwDw[jj@id
ApzA{||@rkAgZj_@_ApAsvAzdB}^~c@qi@pq@}pArbBy@fA_n@vx@qFdHy@dAmzA`rB_I`L
oUj[GH_TnYwb@ll@yp@|_Agc@rl@_e@xo@qb@pl@y_@bi@wPlTmh@hs@_UrYsi@hs@mThZ
kaAxrAst@bbActBhqCaJxLssAveBczAprByw@|dA{z@ziAqc@rj@{m@t{ucAhvAiBbCq^
`f@}^fh@kYdc@yNrUsaAxdBeFjI}JzQqOfX}n@fhAyyA~jCu`BluC}o@hfAegAjeBqZ`e@
kd@tr@iq@rfAgz@xrAmnCllEiTd}}b@bq@_mBtyCus@`gAsZ~0yS|K{KvFaf@pVyM|G}Iz
BaG~@sHFcHa@}HqAmIyCqIaEmUmNuRqJkc@oWcQgLo[yRu@e@{n@g`@k@_@_vD{CsnDa|
B}WsPahHqpEaBcAegHcpE_i@e\|g`HokEiVi0msBopA{ }@_k@sn@oa@uVcM{LqFoEoByQe
HwSuGw_{JwSqCsTiB_U_A}V_@meAj@iv@v@_UTeNNmi@Mcm@h@}e@nBgi@@i|@KaQ}{Nc
@{Km@uJu@kMoAef@kH}AMoL{@kD?gDLkEr@oDdBidJCoDxDqCtEaCtFaCfI}AjIoAxKy@v
MkLhsCk@nP]b0IfoV|OVrG`@vGnCfYpCx]nCd]bGfu@nFdq@nAt0"
```

```
}
```

```
],
```

```
"summary":{
```

```
  "ll_boxes":[{
```

```
    "min_lat":55.79378,
```

```
    "min_lon":37.39366,
```

```
    "max_lat":55.962686,
```

```
    "max_lon":37.546925,
```

```
  ]},
```

```
"has_time_restrictions":false,
```

```
"time":1444.743,
```

```
"length":26.583,
```

```
"cost":3019.145
```

```
}
```

```
}
```

```
}
```

```
],
```

```
"ll_boxes":[{
```

```
  "min_lat":55.79378,
```

```
  "min_lon":37.39366,
```

```
  "max_lat":55.962686,
```

```
  "max_lon":37.546925,
```

```
}],
```

```
"status_message":"Found 1 route(s) between points",
```

```
"status":0,  
"units":"kilometers",  
"language":"ru-RU",  
"id":"route_to_airport"  
}
```


Distance Matrix

The distance matrix service allows you to calculate ETA and distance for start and destination points that define the matrix. The service supports various matrices:

- one-to-many;
- many-to-many;
- many-to-one.

/dm — call point of the distance matrix calculation service.

Request

The request is sent using the HTTP POST method. The body of the POST request contains JSON with required and optional fields.

Empty JSON example:

```
{"sources": [{"lat": 55.796932, "lon": 37.537849}, {"lat": 55.801551, "lon": 37.531575}], "targets": [{"lat": 55.790412, "lon": 37.534313}, {"lat": 55.788644, "lon": 37.536507}], "costing": "pedestrian", "id": "DM_Test"}
```

This request calculates the time and distance matrix for the walking graph for start and destination points.

Required url-request parameters

Field name	Format	Description	Example
api_key	hex-string	See "Access to services"	api_key=fa749bace6d8a3b1....

Required JSON parameters

Field name	Format	Description	Example
------------	--------	-------------	---------

sources	list	List of departure points, where lat and lon are latitude and longitude in degrees (6 decimal places are used)	<pre>"sources": [{ "lat": 55.796932, "lon": 37.537849 }, { "lat": 55.801551, "lon": 37.531575 }]</pre>
---------	------	---------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------

targets	list	List of destination points, where lat and lon are latitude and longitude in degrees (6 decimal places are used)	<pre> "targets": [{ "lat": 55.790412, "lon": 37.534313 }, { "lat": 55.788644, "lon": 37.536507 }] </pre>
---------	------	-----------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------

Maximum total number of sources and targets should not exceed 50 points.

Extra JSON parameters

Field name	Format	Description	Example
costing	string	Transport type for planning a route: <ul style="list-style-type: none"> • auto (default) — automobile; • truck — cargo transport; • pedestrian — pedestrian route; • bicycle — bicycle route. 	<code>"costing": "pedestrian"</code>
id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<code>"id": "DM_Test"</code>

costing_options	dict	List of route calculation parameters. Different transport types have various options and restrictions.	<pre>"costing_options": { "use_border_crossing":0, "use_tolls":0 }</pre>
units	string	Distance unit in response: <ul style="list-style-type: none"> • kilometers (by default) — kilometers; • miles — miles. 	<pre>"units":"miles"</pre>

Response

Field name	Format	Description	Example
id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<pre>"id": "DM_Test"</pre>
targets	list	List of destination points, where lat and lon are latitude and longitude in degrees (6 decimal places are used)	<pre>"targets": [[{ "lon": 37.534313, "lat": 55.790413 }, { "lon": 37.536507, "lat": 55.788643 }]]</pre>

sources	list	List of departure points, where lat and lon are latitude and longitude in degrees (6 decimal places are used)	<pre> "sources": [[{ "lon": 37.537849, "lat": 55.796932 }, { "lon": 37.531574, "lat": 55.801552 }]] </pre>
---------	------	---------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------

sources_to_targets	list	Calculated matrix which contains the numbers of departure and destination matrix elements for each of the calculation results, as well as the calculated values of ETA and EDA	<pre> "sources_to_targets": [[{ "distance": 1.076, "time": 773, "to_index": 0, "from_index": 0 }, { "distance": 1.459, "time": 1041, "to_index": 1, "from_index": 0 }], [{ "distance": 1.828, "time": 1311, "to_index": 0, "from_index": 1 }, { "distance": 2.211, "time": 1579, "to_index": 1, "from_index": 1 }]] </pre>
--------------------	------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

distance	float	Total route length, indicated in the selected units of measurement (EDA)	"distance": 1.828
time	float	Estimated time required to move along the route (ETA). Time is estimated in seconds	"time": 1311
to_index	integer	Sequence number of the destination matrix element in the request	"to_index": 0
from_index	integer	Sequence number of the departure matrix element in the request	"from_index": 0
units	string	Unit of measurement in a response	"units": "kilometers"

If nothing is found, the response will look like this:

```
{"status_code":400,"status":"Bad Request"}
```

Example

POST request with curl utility

```
curl -X POST \
  -H "Content-type: application/json" \
  -H "Accept: application/json" \
  -d
'{"sources":[{"lat":55.796932,"lon":37.537849},{"lat":55.801551,"lon":
37.531575}], "targets":[{"lat":55.790412,"lon":37.534313}, {"lat":55.788
644,"lon":37.536507}], "costing":"pedestrian", "id":"DM_Test"}' \
  "https://geo.rustore.ru/api/dm?api_key=<YOUR_API_KEY>"
```

Reachable Area

Reachability area is a service that allows you to estimate the area counter curves that can be reached from a selected location under the requested restriction (for travel time or distance traveled). The estimated contour curves are returned as a closed line or polygon, which can be displayed on the map.

/iso — call point for the reachability estimation service. Possible counter curve options:

- isochrones (equal travel time curves);
- isodistants (equal distance curves).

Request

The JSON request is sent in the HTTP body. The JSON request consists of required and optional fields. Required parameters must be specified in the request url.

Simple JSON request example:

```
{
  "locations": [
    {
      "lat": 55.796932,
      "lon": 37.537849
    }
  ],
  "costing": "pedestrian",
  "speed": 5.6,
  "contours": [
    {
      "time": 15
    },
    {
      "time": 30
    }
  ],
  "id": "Iso_Test",
  "generalize": 5
}
```

This request generates two isochrones around the given coordinate, limiting the areas that are potentially reachable for a pedestrian to pass from the coordinate in 15 and 30 minutes. Pedestrian speed is estimated as 5.6 km/h.

Required url request parameters

Parameter name	Format	Description	Example
api_key	hex-string	See “Access to services”	fa749bace6d8a3b1....

Required JSON fields in a HTTP request

Field name	Format	Description	Example
locations	list	The reachability matrix estimation point, where lat refers to latitude and lon refers to longitude of the point in degrees (6 decimal places are used)	<pre>"locations": [{ "lat":55.796932, "lon":37.537849 }]</pre>
contours	list	<p>List of isoline parameters. Each parameter determines the metric (time or distance) and the contour color that can be reached by the selected metric when moving from the point defined by locations:</p> <ul style="list-style-type: none"> • time — time in minutes (from 1 to 120) to reach the contour boundary. Used to create isolines; • distance — distance in kilometers (may be fractional) <p><i>Maximum number of contours in one request: 3 (three)</i></p> <p><i>It is possible to set only one metric for each contour: either time or distance.</i></p>	<pre>"contours": [{ "time":15 }, { "time":30 }]</pre>

Extra JSON fields in a HTTP request

Field name	Format	Description	Example
speed	float	<p>Speed: km/h.</p> <p><i>Speed can only be specified for walking (costing=pedestrian) and cycling (costing=bicycle) isochrones.</i></p> <p><i>If the field is empty, the default speed is used: 5 km/h for pedestrians and 20 km/h for bicycles.</i></p>	<code>"speed":20.1</code>
costing	string	<p>Type of transport for creating a route:</p> <ul style="list-style-type: none"> • auto (default) — automobile; • truck (not supported) — truck; • pedestrian — pedestrian; • bicycle — bicycle. 	<code>"costing":"pedestrian"</code>
id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<code>"id":"Iso_Test"</code>
costing_options	dict	List of route estimation parameters. Different transport types have various options and restrictions.	<code>"costing_options": { "use_border_crossing":0, "use_tolls":0 }</code>
units	string	<p>Distance unit in response:</p> <ul style="list-style-type: none"> • kilometers (by default) — kilometers; • miles — miles. 	<code>"units":"miles"</code>

generalize	float	<p>Acceptable variation in meters when generalizing the contour curve using the Ramer–Douglas–Peucker algorithm.</p> <p><i>Reduced number of curve points can lead to the intersection of adjacent contours and to self-intersections.</i></p>	<code>"generalize":5</code>
polygons	boolean	<p>Defines a GeoJSON structure: return polygons or reachable contour curves in a response:</p> <ul style="list-style-type: none"> • false (default) — return curves; • true — return polygons. 	<code>"polygons":true</code>
show_locations	boolean	<p>Return along with contour lines, initial points (specified in locations) and points matched to the road graph (from which contour lines are created).</p> <p>Possible values:</p> <ul style="list-style-type: none"> • false (default) — do not return initial points and points “attached” to the road network; • true — return the above points. 	<code>"show_locations":true</code>

Response

The returned response goes along with the [GeoJSON](#) structure and contains:

Field name	Format	Description	Example
------------	--------	-------------	---------

id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<code>"id": "Iso_Test"</code>
properties	list	Description of reachability time and type for a contour line or polygon, where contour — time value in minutes from the request	<code>"properties": { "contour": 15, "metric": "time" }</code>

geometry	list	<p>Description of contour and its structure, containing:</p> <ul style="list-style-type: none"> • coordinates — set of coordinates describing a polygonal reachable contour in: [lon,lat] - longitude and latitude of the point in degrees (6 decimal places are used) • type — possible structure values: <ul style="list-style-type: none"> ○ Polygon — for reachability polygons; ○ LineString — for lines that describe reachability contour. 	<pre> "geometry": { "coordinates": [[37.537922, 55.798004], [37.536846, 55.797928], [37.536957, 55.796932], [37.537849, 55.796383], [37.539318, 55.796932], [37.537922, 55.798004]], "type": "LineString" } </pre>
----------	------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

If the request is empty, the response will look like this:

```
{"status_code":400,"status":"Bad Request"}
```

Example

Request

```
curl -X POST \  
  -H "Content-type: application/json" \  
  -H "Accept: application/json" \  
  -d \  
'{"locations":[{"lat":55.796932,"lon":37.537849}],"costing":"pedestria  
n","contours":[{"time":15,"color":"ff0000"}, {"time":30,"color":"00ff00  
"}],"id":"Iso_Test","generalize":5}' \  
  "https://geo.rustore.ru/api/iso?api_key=<YOUR_API_KEY>"
```

Response

```
{  
  "id": "Iso_Test",  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "properties": {  
        "contour": 30  
      },  
      "geometry": {  
        "coordinates": [[37.550148, 55.814228], [37.549480,  
55.814301], [37.549141, 55.814220], [37.548851, 55.814262],  
[37.546852, 55.813473], [37.545994, 55.814072], [37.545681,  
55.814106], [37.545200, 55.813931], [37.544643, 55.813137],  
[37.544186, 55.812927], [37.544849, 55.812748], [37.545116,  
55.812195], [37.545120, 55.811932], [37.544849, 55.811672],  
[37.543846, 55.811695], [37.543369, 55.811932], [37.543404,  
55.812378], [37.543674, 55.812927], [37.542850, 55.813446],  
[37.542519, 55.813602], [37.541729, 55.813808], [37.540901,  
55.813980], [37.539993, 55.814072], [37.539852, 55.814152],  
[37.539547, 55.813931], [37.539261, 55.813522], [37.539066,  
55.812931], [37.538933, 55.812855], [37.537849, 55.812763],  
[37.532852, 55.812763], [37.531853, 55.812664], [37.531693,  
55.812771], [37.531349, 55.813435], [37.530849, 55.813683],  
[37.527847, 55.813553], [37.527512, 55.813591], [37.526852,  
55.813824], [37.525101, 55.812931], [37.525043, 55.812744],  
[37.525280, 55.812359], [37.525257, 55.811932], [37.525066,  
55.811714], [37.524853, 55.811634], [37.522850, 55.811668],  
[37.520851, 55.811581], [37.520531, 55.811611], [37.519794,
```

55.811874], [37.518070, 55.811714], [37.517254, 55.811531],
[37.516560, 55.811222], [37.514851, 55.810818], [37.514248,
55.810532], [37.512482, 55.810295], [37.511650, 55.809933],
[37.510849, 55.809696], [37.510563, 55.809216], [37.509514,
55.808933], [37.508846, 55.808617], [37.508575, 55.808655],
[37.507908, 55.808990], [37.507851, 55.809124], [37.506790,
55.808994], [37.506432, 55.808353], [37.506348, 55.807926],
[37.506130, 55.807652], [37.505852, 55.807522], [37.504852,
55.807327], [37.504086, 55.806927], [37.504608, 55.805935],
[37.504890, 55.804932], [37.505463, 55.804539], [37.506413,
55.803490], [37.506596, 55.802933], [37.506027, 55.801758],
[37.505489, 55.801296], [37.505211, 55.800571], [37.504852,
55.800243], [37.503761, 55.800022], [37.503632, 55.799149],
[37.503334, 55.798931], [37.504494, 55.797573], [37.505337,
55.796932], [37.506489, 55.795567], [37.506851, 55.795300],
[37.507401, 55.794479], [37.508106, 55.794186], [37.508385,
55.793926], [37.507927, 55.791931], [37.508575, 55.791653],
[37.509132, 55.791214], [37.509377, 55.790932], [37.509697,
55.789928], [37.510567, 55.789654], [37.510849, 55.789474],
[37.511265, 55.789516], [37.511852, 55.789795], [37.512287,
55.789368], [37.513012, 55.789093], [37.514851, 55.789059],
[37.516853, 55.789448], [37.517147, 55.788933], [37.517376,
55.787933], [37.518219, 55.787567], [37.518467, 55.787548],
[37.518852, 55.787048], [37.520317, 55.787395], [37.520351,
55.786930], [37.519840, 55.785942], [37.519733, 55.784813],
[37.519814, 55.783936], [37.520851, 55.783707], [37.522850,
55.783810], [37.523712, 55.783798], [37.524124, 55.783657],
[37.524662, 55.783737], [37.524937, 55.782932], [37.525475,
55.782558], [37.526855, 55.782169], [37.527237, 55.781937],
[37.527851, 55.781296], [37.528114, 55.781197], [37.528477,
55.781303], [37.529850, 55.781086], [37.530849, 55.781151],
[37.531528, 55.780933], [37.531853, 55.780731], [37.532845,
55.780922], [37.533852, 55.780361], [37.534851, 55.780132],
[37.535851, 55.780499], [37.536850, 55.780613], [37.537590,
55.780933], [37.538418, 55.781933], [37.538094, 55.782177],
[37.536850, 55.782494], [37.536671, 55.782757], [37.536758,
55.783024], [37.537849, 55.783115], [37.538853, 55.783100],
[37.539192, 55.783272], [37.539516, 55.783272], [37.539516,
55.783596], [37.539848, 55.784061], [37.542732, 55.784050],
[37.543095, 55.784172], [37.543610, 55.784172], [37.543610,
55.784691], [37.543850, 55.785027], [37.543968, 55.784927],

[37.543980, 55.784069], [37.544151, 55.783630], [37.544151, 55.783234], [37.544548, 55.783234], [37.544849, 55.783089], [37.545853, 55.783096], [37.546055, 55.782932], [37.545853, 55.782459], [37.544849, 55.782249], [37.544636, 55.782143], [37.544563, 55.781651], [37.544788, 55.780872], [37.544926, 55.780857], [37.545509, 55.781269], [37.546185, 55.781269], [37.546432, 55.781349], [37.547852, 55.781448], [37.548126, 55.781658], [37.548252, 55.781929], [37.548279, 55.782932], [37.548252, 55.783337], [37.547714, 55.783791], [37.547665, 55.783928], [37.547745, 55.785042], [37.547852, 55.785114], [37.548851, 55.785156], [37.549404, 55.785378], [37.549690, 55.786087], [37.549850, 55.786221], [37.551277, 55.786362], [37.551544, 55.786240], [37.551849, 55.786335], [37.552105, 55.786674], [37.552048, 55.787128], [37.551849, 55.787205], [37.551548, 55.787628], [37.551472, 55.787937], [37.551559, 55.788227], [37.551849, 55.788536], [37.552174, 55.788609], [37.552494, 55.788929], [37.552151, 55.789234], [37.550850, 55.789551], [37.550755, 55.789837], [37.550903, 55.789989], [37.552853, 55.789989], [37.552979, 55.789814], [37.552906, 55.788986], [37.553734, 55.789055], [37.553852, 55.788998], [37.554855, 55.788998], [37.555008, 55.788773], [37.555096, 55.788177], [37.555695, 55.788090], [37.555851, 55.788006], [37.556850, 55.788013], [37.556938, 55.787930], [37.556973, 55.785805], [37.556450, 55.785332], [37.556080, 55.783703], [37.555851, 55.783459], [37.555420, 55.783360], [37.554760, 55.783024], [37.554733, 55.782936], [37.555012, 55.782093], [37.555439, 55.781933], [37.555851, 55.781876], [37.556301, 55.782482], [37.556850, 55.782600], [37.558849, 55.782555], [37.559853, 55.782436], [37.560852, 55.782436], [37.561218, 55.782566], [37.561405, 55.782490], [37.561852, 55.782536], [37.562847, 55.782425], [37.563286, 55.782501], [37.563850, 55.782726], [37.564850, 55.782578], [37.565845, 55.782696], [37.566853, 55.782688], [37.568306, 55.782932], [37.568703, 55.783081], [37.569214, 55.783569], [37.569412, 55.783932], [37.569366, 55.784931], [37.569077, 55.785160], [37.568851, 55.785187], [37.568340, 55.785419], [37.567787, 55.785931], [37.568188, 55.786594], [37.568398, 55.787384], [37.568851, 55.787849], [37.569027, 55.787930], [37.568558, 55.788933], [37.568157, 55.789230], [37.567852, 55.789318], [37.567600, 55.789680], [37.567585, 55.789932], [37.567852, 55.790161], [37.568855, 55.790195], [37.569851, 55.790462], [37.570263,


```

55.790337], [37.570557, 55.789932], [37.570576, 55.789658],
[37.570850, 55.789375], [37.571117, 55.789665], [37.571796,
55.789932], [37.572170, 55.790607], [37.572571, 55.790936],
[37.571640, 55.791931], [37.571121, 55.792206], [37.570133,
55.792931], [37.569618, 55.793930], [37.568855, 55.794933],
[37.568947, 55.795933], [37.569851, 55.796734], [37.570312,
55.796932], [37.570080, 55.797165], [37.569851, 55.797222],
[37.569534, 55.797619], [37.569134, 55.798931], [37.568851,
55.799126], [37.567608, 55.799686], [37.567398, 55.799927],
[37.567631, 55.800930], [37.568256, 55.801937], [37.568562,
55.802933], [37.568115, 55.803204], [37.567337, 55.803417],
[37.566853, 55.803654], [37.566193, 55.804276], [37.565849,
55.804287], [37.564850, 55.804539], [37.562851, 55.804642],
[37.562603, 55.804688], [37.562321, 55.804928], [37.562374,
55.805931], [37.562851, 55.806541], [37.563114, 55.806667],
[37.563370, 55.806927], [37.563446, 55.807926], [37.563210,
55.808292], [37.562851, 55.808395], [37.561855, 55.808449],
[37.561455, 55.808323], [37.561081, 55.807697], [37.560852,
55.807652], [37.560570, 55.807930], [37.560551, 55.808628],
[37.560448, 55.808937], [37.560158, 55.809238], [37.559292,
55.809376], [37.558571, 55.809658], [37.557850, 55.809826],
[37.557652, 55.809132], [37.557041, 55.808743], [37.556850,
55.808704], [37.555855, 55.809750], [37.555717, 55.809799],
[37.555286, 55.810364], [37.554852, 55.811588], [37.554558,
55.811642], [37.553555, 55.812637], [37.551849, 55.813408],
[37.550850, 55.813587], [37.550594, 55.813931], [37.550148,
55.814228]],
      "type": "LineString"
    },
    "type": "Feature"
  },
  {
    "properties": {
      "contour": 15
    },
    "geometry": {
      "coordinates": [[37.538876, 55.805958], [37.537895,
55.805969], [37.536850, 55.806187], [37.534851, 55.806030],
[37.533852, 55.806026], [37.530849, 55.805218], [37.530231,
55.804928], [37.530651, 55.803932], [37.530163, 55.803619],
[37.530067, 55.802933], [37.529850, 55.802757], [37.528851,

```

55.802734], [37.528469, 55.802933], [37.528511, 55.803272],
[37.529060, 55.803932], [37.528996, 55.804081], [37.527851,
55.804409], [37.526024, 55.804108], [37.524857, 55.803989],
[37.523849, 55.803200], [37.522850, 55.803543], [37.521851,
55.803284], [37.521061, 55.802933], [37.521778, 55.800926],
[37.521568, 55.799931], [37.521996, 55.798927], [37.522476,
55.798553], [37.522850, 55.798439], [37.523849, 55.798332],
[37.524094, 55.798180], [37.524254, 55.797932], [37.524155,
55.797630], [37.523849, 55.797279], [37.523552, 55.797230],
[37.522991, 55.796932], [37.523434, 55.795933], [37.523846,
55.795574], [37.524853, 55.795555], [37.525974, 55.795811],
[37.526047, 55.795933], [37.527344, 55.796440], [37.527584,
55.796932], [37.527706, 55.797081], [37.527851, 55.797115],
[37.528137, 55.796646], [37.528137, 55.796215], [37.528564,
55.796215], [37.528851, 55.796078], [37.529942, 55.796028],
[37.530178, 55.795601], [37.530178, 55.795261], [37.530521,
55.795261], [37.530849, 55.795101], [37.531178, 55.795261],
[37.531521, 55.795261], [37.531521, 55.795601], [37.531769,
55.796017], [37.533852, 55.796032], [37.534050, 55.796127],
[37.534657, 55.796127], [37.534657, 55.796738], [37.534790,
55.796989], [37.535896, 55.796978], [37.535950, 55.794933],
[37.535625, 55.794704], [37.535076, 55.794704], [37.534946,
55.792835], [37.534645, 55.792721], [37.534058, 55.792721],
[37.533852, 55.792637], [37.531853, 55.792431], [37.531441,
55.792336], [37.530727, 55.791931], [37.531166, 55.790932],
[37.531544, 55.790623], [37.532330, 55.790409], [37.533852,
55.790245], [37.534039, 55.790123], [37.534431, 55.789516],
[37.535152, 55.789234], [37.535465, 55.788933], [37.535500,
55.788578], [37.535851, 55.788170], [37.536205, 55.788574],
[37.536320, 55.788933], [37.536610, 55.789173], [37.536850,
55.789261], [37.538105, 55.789185], [37.538727, 55.788811],
[37.539024, 55.788757], [37.539921, 55.788868], [37.540951,
55.788834], [37.541851, 55.789043], [37.542439, 55.789349],
[37.542850, 55.789448], [37.543850, 55.789272], [37.544254,
55.789337], [37.544502, 55.789280], [37.544849, 55.789421],
[37.546322, 55.789463], [37.546852, 55.789597], [37.547276,
55.789356], [37.547852, 55.789242], [37.548000, 55.789787],
[37.548676, 55.789757], [37.549397, 55.789932], [37.549641,
55.790142], [37.549850, 55.790207], [37.551849, 55.790260],
[37.553345, 55.790436], [37.553852, 55.790600], [37.554203,
55.791931], [37.554337, 55.792934], [37.553852, 55.793671],

```
[37.553501, 55.793930], [37.553276, 55.794353], [37.553158,
55.794933], [37.552265, 55.795349], [37.551750, 55.795837],
[37.551640, 55.796932], [37.551228, 55.797928], [37.551395,
55.798931], [37.551365, 55.799931], [37.552040, 55.800930],
[37.552097, 55.801933], [37.551849, 55.802193], [37.551334,
55.802418], [37.548805, 55.802883], [37.548664, 55.803932],
[37.547852, 55.804455], [37.547508, 55.804588], [37.546852,
55.804703], [37.545853, 55.805096], [37.541851, 55.805519],
[37.540852, 55.805511], [37.539623, 55.805702], [37.538876,
55.805958]],
      "type": "LineString"
    },
    "type": "Feature"
  }
]
}
```

Best Route Planner

Best route planner — service that allows you to estimate the optimal route for a set of arbitrary target points. Car, pedestrian and bicycle graphs can be used to create the best route.

/optimal_route — call point for the best route estimation service. For an input set of coordinates, it creates an optimized route and generates a list of coordinates in the order they are visited.

Request

The JSON request is sent with required and optional request parameters.
Simple request example:

```
{ "locations": [ { "lon": 49.22088, "lat": 55.77055 }, { "lon": 49.21999, "lat": 55.77246 }, { "lon": 49.21933, "lat": 55.77222 }, { "lon": 49.22999, "lat": 55.78246 }, { "lon": 49.26842, "lat": 55.75043 } ], "costing": "pedestrian", "directions_options": { "units": "miles" }, "id": "optimal_route_test" }
```

This request calculates the best pedestrian route starting at the first location in the "locations" list and ending at the last location.

Required parameters

Field name	Format	Description	Example
api_key	hex-string	See "Access to services"	api_key=fa749bace6d8a3b1....

locations	list	<p>List of points to be sorted according to the order in which they were visited. The first and last points in the list remain as such, and the points between them can be rearranged to optimize the route in time.</p> <p>Minimum number of points: 2.</p>	<pre>"locations": [{ "lat":55.77055, "lon":49.22088 }, { "lat":55.796932, "lon":37.537849 }]</pre>
-----------	------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------

Extra parameters

Field name	Format	Description	Example
costing	string	<p>Transport type for planning a route:</p> <ul style="list-style-type: none"> • auto (default) — automobile; • truck — cargo transport; • pedestrian — pedestrian route; • bicycle — bicycle route. 	<pre>"costing":"pedestrian"</pre>
id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<pre>"id":"optimal_route_test"</pre>
costing_options	dict	<p>List of route calculation parameters. Different transport types have various options and restrictions similar to those used in route creating service</p> <p><i>Moreover, there is a "shortest" option, set as false by default. If shortest=false, then the route is time optimized. If shortest=true, then the route is distance optimized.</i></p>	<pre>"costing_options":{ "auto":{ "shortest":true, "use_tolls":0 } }</pre>

units	string	Distance unit in response: <ul style="list-style-type: none"> ● kilometers (by default) — kilometers; ● miles — miles. 	<code>"units":"miles"</code>
fix_destination	bool	Flag indicating whether the last of the listed coordinates in locations should be fixed as the finish line. If <code>fix_destination=true</code> , then the last of the specified locations in the service response is guaranteed to be the finish point. If <code>fix_destination=false</code> , then the last of the specified locations may be one of the intermediate route points. Default value: true.	<code>"fix_destination":"false"</code>

Response

The returned response goes along with the [GeoJSON](#) structure and contains:

Field name	Format	Description	Example
id	string	Request ID that is returned with the response, ensuring the exact match between the request and the response.	<code>"id": "optimal_route_test"</code>

trip	object map	<p>Route information is defined by the following parameters:</p> <ul style="list-style-type: none"> • locations — list of points sorted for the best route time; • legs — information about route polyline; • summary — brief information about the route; • status_message — textual interpretation of the request execution status; • status — request execution status; • units — units of measurement; • language — language in which information about maneuvers is presented. 	<pre> { "trip":{ "locations":[{ "type":"break", "lat":43.1332, "lon":131.9113, "original_index":0 }, { "type":"break", "lat":50.266, "lon":127.5356, "original_index":1 }], "legs":[{ "summary":{ "min_lat":43.131777, "min_lon":127.535648, "max_lat":50.291427, "max_lon":132.040931, "time":58965.898, "length":1424.861, "cost":54399.585 }, "shape":"shape. In this example we skip it." }], "summary":{ "min_lat":43.131777, "min_lon":127.535648, "max_lat":50.291427, "max_lon":132.040931, "time":58965.898, "length":1424.861, "cost":54399.585 }, "status_message":"Found route between points", </pre>
------	------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

			<pre> "status":0, "units":"kilometers", "language":"ru-RU" }, "id":"my_route" } </pre>
--	--	--	--------------------------------------------------------------------------------------------------------------------------------

If the request is empty, the response will look like this:

```

{"status_code":400,"status":"Bad Request"}

```

Example

Request

```

curl -X POST \
  -H "Content-type: application/json" \
  -H "Accept: application/json" \
  -d
'{"locations":[{"lon":49.22088,"lat":55.77055}, {"lon":49.21999,"lat":5
5.77246}, {"lon":49.26842,"lat":55.75043}], "costing":"pedestrian", "dire
ctions_options":{"units":"miles"}, "id":"optimal_route_test"}' \
  "https://geo.rustore.ru/api/optimal_route?api_key=<YOUR_API_KEY>"

```

Response

```

{
  "trip": {
    "locations": [{
      "type": "break",
      "lat": 55.77055,
      "lon": 49.22088,
      "original_index": 0
    }, {
      "type": "break",
      "lat": 55.77246,
      "lon": 49.21999,
      "original_index": 1
    }, {
      "type": "break",
      "lat": 55.75043,

```



```

        "lon": 49.26842,
        "original_index": 2
    }],
    "legs": [{
        "summary": {
            "min_lat": 55.77065,
            "min_lon": 49.21999,
            "max_lat": 55.772514,
            "max_lon": 49.220883,
            "time": 188.317,
            "length": 0.164,
            "cost": 193.317
        },
        "shape": "u`~jiBepe{|AMlh@gq@o@or@m@eMMAtP"
    }, {
        "summary": {
            "min_lat": 55.750237,
            "min_lon": 49.21999,
            "max_lat": 55.772515,
            "max_lon": 49.269982,
            "time": 3543.527,
            "length": 3.113,
            "cost": 3644.115
        },
        "shape":
"cuakiBmxc{|A@uPEiUHgkAF_n@BgYHghAPwvBFgr@XcrD@{ZBmGJupAG_SHw\`D}RNoaC
`@cNfAyGxHaGja@Y~JmFxKyLlBuBfUy[pI_Ux@uBfNkb~rFwTvMgx@dBkKlD}SrFs]bFa`
@pCgTdI{i@fC}R`@aD~Pk1AzBwObLqw@vIuf@lLop@p@oBrJuYfNmYbAwBtJwPfCuBxIkH
zMqIh_@sRfCWb@}SqAu`@_GcfB_Bqh@yBmk@w@}M{D}}@qG}yAuBqZ|D_G~oCshAhQmAzW
iRla@wYlbAw}@fCsFj{@ijBjWij@zZsp@fm@eqAdAgD`Ssn@dBqFNea@xVavAdZajBxEom
A|AeeBhHiNbLcAnAuGpT|VlD~DpaBf|@vkA|q@|eBjOv`@b[tPwGzDtA`s@`_AtGUjo@{v
AvUmm@iJyL"
    }],
    "summary": {
        "min_lat": 55.750237,
        "min_lon": 49.21999,
        "max_lat": 55.772515,
        "max_lon": 49.269982,
        "time": 3731.844,
        "length": 3.278,
        "cost": 3837.433
    }
}

```

```
    },  
    "status_message": "Found route between points",  
    "status": 0,  
    "units": "miles",  
    "language": "ru-RU"  
  },  
  "id": "optimal_route_test"  
}
```

Polyline Route Decoding

The route creation service uses an encoded polyline to store a series of latitude and longitude coordinates as a single string. Polyline coding greatly reduces the response size.

To decode a polyline into a series of coordinates, you can follow the steps below.

JavaScript

```
polyline.decode = function(str, precision) {
    var index = 0,
        lat = 0,
        lng = 0,
        coordinates = [],
        shift = 0,
        result = 0,
        byte = null,
        latitude_change,
        longitude_change,
        factor = Math.pow(10, precision || 6);

    // Coordinates have variable length when encoded, so just keep
    // track of whether we've hit the end of the string. In each
    // loop iteration, a single coordinate is decoded.
    while (index < str.length) {

        // Reset shift, result, and byte
        byte = null;
        shift = 0;
        result = 0;

        do {
            byte = str.charCodeAt(index++) - 63;
            result |= (byte & 0x1f) << shift;
            shift += 5;
        } while (byte >= 0x20);

        latitude_change = ((result & 1) ? ~(result >> 1) : (result >>
1));

        shift = result = 0;
    }
}
```

```

    do {
        byte = str.charCodeAt(index++) - 63;
        result |= (byte & 0x1f) << shift;
        shift += 5;
    } while (byte >= 0x20);

    longitude_change = ((result & 1) ? ~(result >> 1) : (result >>
1));

    lat += latitude_change;
    lng += longitude_change;

    coordinates.push([lat / factor, lng / factor]);
}

return coordinates;
};
C++ 11

```

```
#include <vector>
```

```
constexpr double kPolylinePrecision = 1E6;
constexpr double kInvPolylinePrecision = 1.0 / kPolylinePrecision;
```

```
struct PointLL {
    float lat;
    float lon;
};
```

```
std::vector<PointLL> decode(const std::string& encoded) {
    size_t i = 0;    // what byte are we looking at
```

```
    // Handy lambda to turn a few bytes of an encoded string into an
integer
```

```
    auto deserialize = [&encoded, &i](const int previous) {
        // Grab each 5 bits and mask it in where it belongs using the
shift
```

```
        int byte, shift = 0, result = 0;
        do {
            byte = static_cast<int>(encoded[i++]) - 63;
            result |= (byte & 0x1f) << shift;
            shift += 5;
```

```

    } while (byte >= 0x20);
    // Undo the left shift from above or the bit flipping and add to
previous
    // since its an offset
    return previous + (result & 1 ? ~(result >> 1) : (result >> 1));
};

// Iterate over all characters in the encoded string
std::vector<PointLL> shape;
int last_lon = 0, last_lat = 0;
while (i < encoded.length()) {
    // Decode the coordinates, lat first for some reason
    int lat = deserialize(last_lat);
    int lon = deserialize(last_lon);

    // Shift the decimal point 5 places to the left
    shape.emplace_back(static_cast<float>(static_cast<double>(lat) *
                                        kInvPolylinePrecision),
                       static_cast<float>(static_cast<double>(lon) *
                                        kInvPolylinePrecision));

    // Remember the last one we encountered
    last_lon = lon;
    last_lat = lat;
}
return shape;
}

```

Python

```

#!/usr/bin/env python

import sys

#six degrees of precision in valhalla
inv = 1.0 / 1e6;

def decode(encoded):
    """Decodes route polyline which is returned by rose."""

```

```

decoded = []
i = 0
previous_coords = {'lat': 0, 'lon': 0}
while i < len(encoded):
    coords = dict()
    for coord_name in ('lat', 'lon'):
        coord = 0
        shift = 0
        byte = 0x20
        # Keep decoding bytes until you have this coord.
        while byte >= 0x20:
            byte = ord(encoded[i]) - 63
            i += 1
            coord |= (byte & 0x1f) << shift
            shift += 5
        # Get the final value adding the previous offset and
        # remember it for the next.
        coords[coord_name] = previous_coords[coord_name] + (
            ~(coord >> 1)
            if coord & 1
            else (coord >> 1)
        )
    # Scale by the precision and chop off long coords.
    # Also flip the positions so its the far more standard
    # (lon, lat) instead of (lat, lon).
    decoded.append([
        float(f"{coords['lon'] * inv:.6f}"),
        float(f"{coords['lat'] * inv:.6f}"),
    ])
    previous_coords = coords
return decoded

print decode(sys.argv[1])

```

Maps Mobile SDK

Maps SDK allows you to add a map to your iOS and Android apps.

Android

How to connect Maps Mobile SDK

To connect Maps SDK, you need to add a maven repository link to the main gradle file of the project:

```
allprojects {
    repositories {
        maven {
            url =
uri("https://maven.pkg.github.com/GEORS/MAPS-SDK-ANDROID")
            credentials {
                username = "GITHUB_USER"
                password = "GITHUB_TOKEN"
            }
        }
    }
}
```

You should also specify the dependency in the gradle file, as shown in the example.

Dependency injection

```
implementation('ru.rustore.geo:mapkit:x.x.x')
```

where x.x.x — SDK version.

We recommend using the latest SDK version. Release versions of SDK are numbered 1.0.x

The latest SDK version is available at:

https://github.com/geors?tab=packages&repo_name=maps-sdk-android

How to use Maps Mobile SDK

It is required to initialize the card global settings before using any other SDK component, the best option is in the Application class successor. The global settings object is:

```
class MapViewConfig(  
    val apiKey: String, // unique key to grant access to SDK  
)
```

install the object:

```
MapGlobalConfig.setMapGlobalConfig(  
    MapViewConfig(  
        apiKey = apiKey  
    )  
)
```

To start working with SDK, a number of classes are provided that implement View in one form or another:

```
data class MapStartOptions(  
    val center: LatLon, // initial map location point (only for  
    lat, lon)  
    val zoomLevel: Float, // initial zoom level (zoomLevel)  
    val style: MapStyle, // map style, you can select the  
    corresponding enum or use your own  
    val compassLocationMode: CompassLocationMode, // compass  
    setting, can be selected from the corresponding enum  
    val logoConfig: LogoConfig // logo configuration  
)
```

You need to set the start settings as shown in the example.

```
MapGlobalConfig.setMapStartOptions(MapStartOptions(...))
```

LogoConfig example

```

data class LogoConfig(
    val logoAlignment: Alignment, // Alignment: BottomRight,
    BottomLeft, TopRight, TopLeft
    val logoAdditionalPaddings: AdditionalPaddings
)

```

To work with the SDK, a number of classes are provided that implement View in one form or another:

- MapView is the main view in the SDK that displays the map;
- ZoomView is used to display + and - controls to zoom in and out of the map;
- CurrentLocationView is a button to focus the map on the current user position and follow his position;
- CompassView is a component that displays the direction of a physical device relative to the north.

To display these controls, you must place them in the xml file.

```

<FrameLayout
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>

    <ru.rustore.geo.views.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</FrameLayout>

```

Map example

```

<FrameLayout
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>

    <ru.mail.maps.sdk.views.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

```

```
</FrameLayout>
```

This may be enough to work with the map if other elements are not required.

Controls example

```
<FrameLayout
    xmlns:custom="http://schemas.android.com/apk/res-auto"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>

    <ru.rustore.geo.views.MapView
        android:id="@+id/mapView"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <ru.rustore.geo.views.ZoomView
        android:id="@+id/zoomView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        ...
        custom:mapView="@+id/mapView" />

    <ru.rustore.geo.views.CompassView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        ...
        custom:mapView="@+id/mapView" />

    <ru.rustore.geo.views.CurrentLocationView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        ...
        custom:mapView="@+id/mapView" />
</FrameLayout>
```

All elements will be displayed on top of the map and will be linked to it. To call the map methods directly, you need to get the Map entity, for this you need to call the `getMapAsync` method on the map visual control:

```

mapView = findViewById(R.id.mapView)
mapView.getMapAsync { map -> // the Map object
    ...
}

```

For the `CurrentLocationView` and `CompassView` to work correctly, you must pass an implementation of the `LocationSource` interface to the `Map` object, for example:

```
map.setLocationSource(locationSource)
```

The `LocationSource` interface provides the following methods:

```

fun activate(listener: (mapLocation: MapLocation) -> Unit) // it is
used to start obtaining GPS fixes from the system
fun deactivate() // it is called to stop obtaining GPS data

```

SDK itself calls the `activate` and `deactivate` methods when it is required according to internal logic. When calling `activate` methods, `listener` is passed as a parameter, which must be called when each new `gps` fix is received from `MapLocation` type arguments:

```

data class MapLocation(
    val latitude: Double? = null, [-90, 90]
    val longitude: Double? = null, [-180, 180]
    val speed: Float? = null, //
    val bearing: Float? = null, [0, 360]
    val accuracy: Float? = null,
    val altitude: Double? = null
)

```

each of the class fields can be null, but for correct operation it is necessary to substitute the corresponding values received from the system. Apart from the `setLocationSource` method, the `Map` object has the following methods:

```

fun zoomIn(step: Float = .5f, animated: Boolean = true)
    // zoom in on a map location, where step – step map zoom value

fun zoomOut(step: Float = .5f, animated: Boolean = true)
    // zoom out on a map location, where step – step map zoom value

fun setBearing(bearing: Float, animated: Boolean = true)

```

```

    // sets a map direction, bearing is kept
    // within the semi interval [0, 360)

fun setZoom(zoom: Float, animated: Boolean = true)
    // sets a map zoom level, zoom is kept
    // within the semi interval (0, 20]

fun addMarker(marker: MarkerEntity)
    // add a marker to the map, where MarkerEntity - marker model,
    // id - unique ID for each marker,
    // coordinates - coordinates of the point to which
    // a marker will be linked (only latitude and longitude
    // are taken into account, the rest are null)
    // and image - one of MarkerImage enum

fun addMarker(markers: List<MarkerEntity>)
    // add a list of markers to a map

fun removeMarker(id: String)
    // delete a marker with a specified ID

fun removeAllMarkers()
    // delete all markers from a map

fun showPopUp(markerId: String, content: String)
    // display a Pop-up window above the marker,
    // markerId - marker ID to which the window will be linked;
    // content - an html string to be displayed in the window

fun hidePopUp(markerId: String)
    // hide a Pop-up window for the specific marker ID

fun setOnMarkerClickListener(onClickListener: (id: String, location:
MapLocation) -> Unit)
    // add callback method that will be called
    // when the user clicks on one of the markers
    // id - marker ID clicked by the user
    // location - coordinates that match the marker
    // (only latitude and longitude are taken into account,
    // the rest are null)

```

```

fun removeMarkerClickListener()
    // remove the callback method responsible for clicking on the
marker

fun moveMarker(id: String, location: MapLocation, animated: Boolean,
duration: Double)
    // set marker movement with the corresponding ID
    // to the position location. animated indicates whether
    // marker movement animation is needed
    // duration - move animation duration

fun addOnErrorListener(onErrorListener: (error: MapError) -> Unit)
    // add a callback that will be called in case
    // of any errors in SDK. All errors are
    // descendants of the MapError class

fun setCenter(center: MapLocation, animated: Boolean)
    // move the map so that the screen center would match
    // the center parameter (only latitude and longitude
    // are taken into account, the rest are null)

fun changeStyle(style: MapStyle)
    // change the map style with no reinitialization required

fun enableDragPan(enable: Boolean)
    // enable or disable gesture control

fun enableZoomRotate(enable: Boolean)
    // enable or disable map zoom and rotation by a user

fun setOnZoomChangedListener(listener: (zoom: Double) -> Unit)
    // set the callback method that will be called when the map scale
changes

fun removeZoomChangedListener()
    // remove the callback method responsible for changing the map
scale

fun setOnMapClickListener(onClickListener: (location: MapLocation,
screenLocation: ScreenLocation) -> Unit)
    // set the callback method responsible for the map click event

```

```

fun setOnMapLongClickListener(onClickListener: (location: MapLocation,
screenLocation: ScreenLocation) -> Unit)
    // set the callback method responsible for the long click event

fun addLayer(layer: Layer)
    // add a new layer to the map

fun addMapDataSource(source: MapDataSource)
    // add a new map data source

fun removeSource(sourceId: String)
    // remove data source by ID

fun removeLayer(layerId: String)
    // remove map layer by ID

fun addCluster(cluster: Cluster)
    // add a cluster to the map, where the Cluster object contains:
    // id - cluster ID, markers - list of marker objects,
    // radius - cluster radius in meters,
    // textColor - text color in hex ("#ff0000"),
    // backgroundColor - background color in hex ("#ffffff")

fun removeCluster(id: String)
    // remove a cluster by ID

```

Using these methods, you can manually control the map, regardless of the elements provided by SDK. All SDK methods must be called on the main thread. Callbacks are also returned to the main thread.

To use the `addMapDataSource` method, you must create a `MapDataSource` object, which can be one of three types: `CircleSource`, `GeojsonSource`, and `PolylineSource`.

```

class CircleSource(
    val id: String, // unique data source ID
    val center: LatLon, // circle center
    val radius: Double, // radius in meters
    val steps: Int // number of edges in circle interpolation
)

class GeojsonSource(

```

```

    val id: String, // unique data source ID
    val geojsonData: ByteArray // GeoJson string as ByteArray
)

class PolylineSource(
    val id: String, // unique data source ID
    val polylineData: String // string-encoded polyline info
)

```

To add the functionality of geocoding and/or isochrones a dependency is required.

Dependency injection

```
implementation('ru.mail.maps:vk-maps-api:x.x.x')
```

Next, you need to create an entity of the GeocodeApi interface through the Builder:

```
val mapsApi = VkMapsApi.Builder().apiKey(API_KEY).build()
```

To use the geocode and isochrones functionality, you must call the geocode and iso methods, respectively.

Geocoding

```
val response = mapsApi.geocode("Moscow")
```

Isochrones

```

val response = mapsApi.iso(locations = listOf(LatLon()),
    contours = listOf(
        Contour(time = 15, color = "ff0000", distance = 0f),
        Contour(time = 30, color = "00ff00", distance = 0f)
    )
)

```

A ResponseWrapper object is returned that can contain a Success geocoding result and a list of results of type Result in the value field, or an error of type HttpError or GeneralError.

System requirements

Minimum supported Android version: 7.0 (min sdk 24)

iOS

How to add the library

The library is connected through the Swift Package Manager.

In Xcode, select File -> Swift Packages -> Add Package Dependency from the menu and enter <https://github.com/geors/maps-sdk-ios> in the repository address field.

Select Maps SDK and add it to your application.

How to use the library

Import MapsSDK into the corresponding map file.

```
import MapsSDK
```

To get started with the map, you need:

1. Create a MapView object.
2. Configure the map.
3. Specify a map delegate.

Create MapView Object

```
let mapView = MapView()
```

Configure MapView Object

Create a MapViewConfig object containing the following information:

- API key to work with SDK;
- map center coordinates;
- entry level zoom;
- tile style.

Setup and configuration

```
let mapConfig = MapViewConfig(
    apiKey: "##API_KEY##",
    center: Coordinates(lng: 33, lat: 55),
    zoomLevel: 11,
    style: .automatic
)
mapView.setup(mapConfig)
```

zoomLevel can be set from 0 (farthest) to 22 (closest). The style of styles can take one of the following values:

- .automatic — automatic style selection depending on isDriveMode and the current device color scheme;
- .main — base style;
- .light — light style;
- .dark — dark style;
- .navMain — base style for navigation with an emphasis on roads;
- .navDark — dark style for navigation with an emphasis on roads

Map delegate specification

The delegate must implement the requirements of the MapViewDelegate protocol. It will be used to process notifications from the map about various events, such as:

- map loaded;
- user-triggered event (such as a touch) received;
- events triggered by map controls, and so on.

Integrate a map into an app

```
import MapsSDK

...

override func viewDidLoad() {
    let mapView = MapView()
```

```

let mapConfig = MapViewConfig(
    apiKey: "##API_KEY##",
    center: Coordinates(lng: 33, lat: 55),
    zoomLevel: 11,
    style: .automatic
)

mapView.setup(mapConfig)
mapView.delegate = self
view.addSubview(mapView)
}

```

Map management

Set user's current coordinate and direction

```
mapView.setCurrentLocation(Coordinates(lng: 33, lat: 55), bearing: 0,
accuracy: 0)
```

Set map direction

```
mapView.setBearing(90, animated: true)
```

Set map center coordinates

```
mapView.setCenter(Coordinates(lng: 33, lat: 55), animated: true)
```

Set initial zoom level

```
mapView.setZoom(11, animated: true)
```

Enable/disable control components

```
mapView.isZoomButtonsHidden = true
mapView.isCompassHidden = true
mapView.isMyLocationButtonHidden = true
```

Enable/disable gestures

```
mapView.isDragPanEnabled = true
mapView.isZoomRotateEnabled = true
```

Markers

When creating a marker, you need to specify its identifier, coordinate and image.

For a picture, it is recommended to use a 48x48 image. The image is displayed on the map in such a way that the middle of the bottom edge appears to be at the specified coordinate. You can use the available images or use your own ones.

Adding individual markers

```
let marker1 = Marker(id: "marker_id_1",
                    coords: Coordinates(lng: 33, lat: 55),
                    pin: .electricPin)

let marker2 = Marker(id: "marker_id_2",
                    coords: Coordinates(lng: 33, lat: 55),
                    pin: .electricInfo,
                    alignment: .bottomLeft)
```

In this case, images are transferred from a set of images preinstalled in the SDK. If necessary, you can upload your own image.

```
let markerImage = UIImage(...)
let marker3 = Marker(id: "marker_id_3",
                    coords: Coordinates(lng: 33, lat: 55),
                    pin: .custom(markerImage),
                    alignment: .center)
```

By default, marker alignment is set to `.center`, which means that the center of the marker is aligned with the passed coordinates. You have the ability to flexibly control this parameter by choosing one of the many available values, or pass an arbitrary offset.

```
let marker4 = Marker(id: "marker_id_4",
                    coords: Coordinates(lng: 33, lat: 55),
                    pin: .electricPhoto,
                    alignment: .bottom)

let marker5 = Marker(id: "marker_id_5",
                    coords: Coordinates(lng: 33, lat: 55),
                    pin: .electricStar,
                    alignment: .center(offsetByX: 10, byY:
-10))
```

To place a marker on the map, use the `addMarker(_: Marker)` method.

Add a single marker

```
mapView.addMarker(marker1)
```

```
mapView.addMarker(marker2)
```

Add markers in an array

```
mapView.addMarkers([marker1, marker2])
```

Delete a single marker via its ID

```
mapView.removeMarker(id: "marker_id_1")
```

Delete all markers

```
mapView.removeAllMarkers()
```

Tracing the marker click event

```
func MapDelegate: MapViewDelegate{
    func mapView(_ map: MapView, didSelectMarkerID markerID:
String) {
        // ...
        // markerID
    }
}
```

Additionally, when any marker is clicked, the delegate's `MapViewDelegate.mapView(_: MapView, didReceiveEvent: MapEvent)` method is called. In this case, the `didReceiveEvent.type` property is set to `.clickOnMarker`.

```
func MapDelegate: MapViewDelegate{
    func mapView(_ map: MapView, didReceiveEvent event: String) {
        // ...
        // event.type = .clickOnMarker
        // ...
    }
}
```

Pop-ups

To display a pop-up on the map, you need to specify marker ID and pop-up text.

```
mapView.displayPopup(markerId: "marker_id_1", content: "Hello world")
```

To hide the pop-up, you need to specify the marker ID.

```
mapView.hidePopup(markerId: "marker_id_1")
```

Display a pop-up once a marker on the map is selected

To do this, you must implement `mapView(_: onMarkerSelectID:)` of the `MapViewDelegate` delegate.

```
extension MyController: MapViewDelegate {  
    func mapView(_ mapView: MapView, onMarkerSelectID: String) {  
        mapView.displayPopup(markerId: id, content: "Hello world")  
    }  
}
```

Styles

The map supports changing styles. `.automatic` style means that the map will automatically change light and dark styles based on the system interface theme.

```
mapView.changeStyle(.automatic)  
mapView.changeStyle(.dark)
```

GeoJSON

The map supports drawing polygons and lines from a GeoJSON source.

Add sources and layers only once the map has been uploaded, which can be tracked in `mapViewDidLoad(_:)` delegate

```
let sourceData = Data(...)
```

```
let source = MapDataSource(id: "sourceID", type:  
    .geoJSON(sourceData))  
mapView.addSource(source)
```

```
let fillLayer = MapLayer(  
    id: "fillLayer",  
    sourceID: "sourceID",  
    paint: FillPaintProperties(fillColor: .iuColor(.green),  
    fillOpacity: .value(0.3))  
)
```

```
let strokeLayer = MapLayer(  
    id: "strokeLayer",  
    sourceID: "sourceID",  
    paint: StrokePaintProperties(strokeColor: .iuColor(.green),  
    strokeWidth: 2)
```

```

        id: "strokeLayer",
        sourceID: "sourceID",
        paint: LinePaintProperties()
    )

mapView.addLayer(fillLayer)
mapView.addLayer(strokeLayer)

```

You can delete sources and layers by specifying their IDs.

```

mapView.removeLayer("fillLayer")
mapView.removeLayer("strokeLayer")
mapView.removeSource("sourceID")

```

Route creation as an encoded string from the route creation service is supported.

```

let routeSource = MapDataSource(id: "routeSourceID", type:
    .encodedString(encodedRoute))
mapView.addSource(routeSource)

```

```

let routeLayer = MapLayer(
    id: "routeLineLayer",
    sourceID: "routeSourceID",
    paint: LinePaintProperties(lineColor: .iuColor(.red),
    lineWidth: 2)
)

```

```

mapView.addLayer(routeLayer)

```

Circles are supported. Since GeoJSON does not support circles, they are simulated as a polygon with a given number of sides.

```

mapView.addCircleSource(center: coords, radius: 500, steps: 32,
    id: sourceID)

```

If sources and layers are known in advance, then you can add them with a single call.

```
mapView.addSourcesAndLayers(
    sources: [
        source,
        routeSource
    ],
    layers: [
        fillLayer,
        strokeLayer,
        routeLayer
    ]
)
```

Clustering

Markers can be combined into clusters. Clustering creates a new data source on the map, so it should only be used after the map is uploaded. You can specify clusters radius in meters, as well as text and background color.

```
func mapViewDidLoad(_ mapView: MapView) {
    let markers: [Marker] = ...
    mapView.addCluster(markers, id: "clusterId", radius: 50,
        textColor: .white, backgroundColor: .blue)
}
```

Clusters are deleted by specifying the cluster ID.

```
mapView.removeCluster(id: "clusterId")
```

Traffic jams and isolines

The map can show traffic jams, metro lines and elevation levels (contours). To enable it, use the `setLayoutVisible` method.

```
mapView.setLayoutVisible(true, layout: .traffic)
mapView.setLayoutVisible(true, layout: .isolines)
mapView.setLayoutVisible(true, layout: .isolinesLabel)
mapView.setLayoutVisible(true, layout: .subway)
```

Error handling

Use the `mapView(, didFailWithError:)` method of `MapViewDelegate` delegate to handle errors.

```
extension MyController: MapViewDelegate {
```



```
func mapView(_ mapView: MapView, didFailWithError error: Error) {
    print("Did fail with error: \(error.localizedDescription)")
}
}
```

Geocoding

Geocoder — Maps SDK component for direct and reverse geocoding. An API key is required for initialization.

```
let geocoder = Geocoder(apiKey: "your apiKey")

geocoder.geocode(
    query: "Leningradsky Prospekt 39c79",
    lang: "ru",
    location: Coordinates(lng: 37.537892, lat: 55.796926)
) { result in
    switch result {
    case let .success(response):
        print(response)
    case let .failure(error):
        print(error.localizedDescription)
    }
}
```

SwiftUI

Apply a Map component to use the map in SwiftUI.

```
import SwiftUI
import MapsSDK

class StateObject: ObservableObject {
    let mapConfig = MapViewConfig(
        apiKey: apiKey,
        center: Coordinates(lng: 37.537892, lat: 55.796926)
        zoomLevel: 15,
        style: .automatic
    )

    let mapView = MapView()
}

extension StateObject: MapViewDelegate {
    // delegate methods
}

struct ContentView: View {
    @StateObject private var state = StateObject()

    var body: some View {
        Map(
            config: state.mapConfig,
            view: state.mapView,
            delegate: state
        )
    }
}
```

Restrictions

MapsSDK can be integrated into apps that support iOS 12.4 and later versions.

Starting with iPadOS 13, users can use multi-window mode. MapsSDK has not been optimized for this mode.

You can easily integrate MapsSDK into applications based on the UIKit or SwiftUI frameworks.

Additional services

Please refer to the table below to learn more about additional services that are essential when some supplementary details are required.

Services

Call point	Description
/elevation	Determines point or profile elevation along the route
/ip2geo	Determines location by IP address
/timezone	Determines the time zone at any point, as well as the offset of time relative to UTC.
/postcode	Determines postal code by address or addresses via postal code. For Russia only: you can search for a post office by postal code

Elevation assessment

/elevation — call point of a service that allows gathering elevation data. The service allows you to request elevation for:

- separate point on the map;
- set of unrelated points;
- route — a sequence of interconnected points (for this, the `resample_distance` field in json is used).

Request

API requests are made with HTTP POST.

Required POST-request parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to services”	<code>api_key=fa749bace6d8a3b1....</code>
json	string	POST request body with the parameters necessary to get the elevation data.	

JSON

JSON is passed in a request body.

Required fields

Field	Format	Description	Json example	Service response example
locations	<p>Coordinates can be passed to this field in one of 3 valid formats:</p> <ul style="list-style-type: none"> Coordinates list: <pre>[{"lat": 56.1, "lon": 43.2}, {"lat": 56.2, "lon": 43.3}]</pre> Encoded string: points specially encoded into a string (see also: Polyline route decoding). For example, a polyline in this format is returned by the route planner (response field "shape"). <pre>"s{cplAfiz{pCa]xBxBx`AhC gApBrz@[h..."</pre> String: one or more coordinates separated by . Coordinates are used as lat,lon, where: <ul style="list-style-type: none"> lat - latitude (up to 6 decimal places); lon - longitude (up to 6 decimal places) <pre>152.2, 34.8 158.8, 23.0 120.0, 10.0</pre> 	<p>One or more coordinates in any of three valid formats. For the given locations field, the service returns a set of elevations in meters. The number of elevations is equal to the number of points in locations.</p>	<pre>{ "locations": [{ "lat": 55.601897, "lon": 37.581305 }, { "lat": 55.885040, "lon": 37.680869 }] }</pre>	<pre>{ "locations": [{ "lat": 55.601897, "lon": 37.581305 }, { "lat": 55.885040, "lon": 37.680869 }], "height": [201, 139] }</pre>

Optional fields

Field	Format	Description	Json example	Service response example
-------	--------	-------------	--------------	--------------------------

range	Boolean	<p>Flag: when set to true, instead of a one-dimensional "height" array, the service returns a two-dimensional "range_height" array, which contains pairs of values. The first is distance in meters from the previous locations, the second is height. The flag is useful to create a profile of heights, as well as to calculate the tilt for ascents and descents. Default value: false.</p>	<pre>{ "range": true, "locations": [{ "lat": 55.601897, "lon": 37.581305 }, { "lat": 55.885040, "lon": 37.680869 }] }</pre>	<pre>{ "locations": [{ "lat": 55.601897, "lon": 37.581305 }, { "lat": 55.885040, "lon": 37.680869 }], "range_height": [[0, 201], [32131, 139]] }</pre>
-------	---------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

resample_distance	Integer	<p>Value in meters to indicate the distance between the height points. The original route from successive points is divided into segments such as resample_distance. This service returns the points formed in this way and the height values for these points. Use case examples: it is required to build an elevation profile in a user route with heights at a 50 m distance from each other. Default value: no field, the service returns one height value per specified coordinate.</p>	<pre>{ "resample_distance":10000, "locations":[{ "lat":55.601897, "lon":37.581305 }, { "lat":55.885040, "lon":37.680869 }] }</pre>	<pre>{ "locations":[{ "lat":55.601897, "lon":37.581305 }, { "lat":55.690027, "lon":37.612137 }, { "lat":55.778150, "lon":37.643109 }, { "lat":55.866264, "lon":37.674221 }, { "lat":55.885040, "lon":37.680869 }], "range_height":[[0, 201], [10000, 143], [20000, 161]], }</pre>
-------------------	---------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

				<pre>[30000, 151], [32131, 139]]</pre>
height_precision	Integer	Number of decimal places in the height value. Fractional height values may be required, for example, for greater accuracy and smoothness in the elevation profile. Possible values: 0-2. Default value: 0 (height is an integer).	<pre>{ "height_precision": 2, "locations":[{ "lat":55.601897, "lon":37.581305 }, { "lat":55.885040, "lon":37.680869 }] }</pre>	<pre>{ "locations":[{ "lat":55.601897, "lon":37.581305 }, { "lat":55.885040, "lon":37.680869 }], "height":[200.94, 138.84] }</pre>

Response

If height cannot be determined for some point, null is returned instead of a value in meters.

Field name	Format	Description	Example
------------	--------	-------------	---------

locations	Response field format matches the format selected for the request	This field returns the coordinates for which the height was requested.	<p>"list of coordinates" format:</p> <pre>"locations": [{ "lat": 55.601897, "lon": 37.581305 }, { "lat": 55.885040, "lon": 37.680869 }]</pre> <p>"encoded string" format:</p> <pre>"s{cplAfiz{pCa]xBxBx`AhC gApBrz@[h..."</pre> <p>"string" format:</p> <pre>152.2, 34.8 158.8, 23.0 120.0, 10.0</pre>
height	Height array	<p>This field will be present in the response if "range":true is not specified in the request.</p> <p>If the "resample_distance" field is not specified in the request, the number of values in the "height" array is the same as the number of points passed in the request.</p> <p>If the "resample_distance" field is specified, then the number of heights corresponds to the number of points on the locations polyline, between which the distance is equal to "resample_distance"</p>	<pre>"height": [221, 172, 206, 188, 153]</pre>

range_height	An array of x and y value pairs: x - total distance in meters from the route start. For the first coordinate it is always 0 m y - height in meters for the coordinate at the given index	This field is present in the response if "range":true is specified in the request. The number of pairs in the "range_height" array is determined by the same rules as in the "height" array	<code>"range_height":[[0,221],[248,172],[902,206],[1308,188],[1601,153]]</code>
--------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------

Examples

Request with a single coordinate

```
curl -X POST \
  -H "Content-type: application/json" \
  -H "Accept: application/json" \
  -d '{"locations":"55.601897, 37.581305"}' \
  "https://geo.rustore.ru/api/elevation?api_key=<YOUR_API_KEY>"
```

Response

```
{"locations":"55.601897, 37.581305","height":[201]}
```

Request with coordinates with no height data

```
curl -X POST \
  -H "Content-type: application/json" \
  -H "Accept: application/json" \
  -d '{"locations":[{"lat":0.0,"lon":0.0}, {"lat":0.1,"lon":0.1}]}' \
  "https://geo.rustore.ru/api/elevation?api_key=<YOUR_API_KEY>"
```

Response

```
{"locations":[{"lat":0.000000,"lon":0.000000}, {"lat":0.100000,"lon":0.100000}], "height":[null,null]}
```

IP Location Detection

`/ip2geo` — call point of the IP location service.

Request

Required request parameters

Field name	Format	Description	Example
api_key	hex-string	See "Access to service"	<code>fa749bace6d8a3b1....</code>

Extra request parameters

Field name	Format	Description	Example
q	string	Search request that contains IPv4 IP address in decimals. By default, if the requested IP address is not specified in the request, the IP address of the client will be automatically determined and used.	<code>192.168.1.1</code>
lang	2-character language code	Response language in one of the available languages. The object area language is used by default.	<code>lang=en</code>

Response

Field name	Format	Description	Example
geoid	object	Internal Geo ID	<code>"geo_id": "546"</code>
address	string	Object address	<code>"address": "Russia Moscow"</code>

bbox	list	Object location boundaries for map positioning	<pre>"bbox": [37.326228, 55.491308, 37.967428, 55.957772]</pre>
isocode	2char	2-character country code according to ISO 3166-1 alpha-2	<pre>"isocode": "RU"</pre>
type	string	Object type	<pre>"type": "locality"</pre>
ref	hex	Object ID can be used to get additional information in the Geocoder service <i>Object ID is not fixed and may be changed over time</i>	<pre>"ref": "1000000C4D63818"</pre>
pin	list	Object positioning data (longitude and latitude)	<pre>"pin": [37.538851, 55.796731]</pre>

Example

Request

```
https://geo.rustore.ru/api/ip2geo?api_key=<YOUR_API_KEY>&q=46.138.195.192
```

Response

```
{
  "request": "/ip2geo?q=46.138.195.192&api_key",
  "results": [
    {
```

```
"address": "Russia Moscow",
"bbox": [
  37.326228,
  55.491308,
  37.967428,
  55.957772
],
"geo_id": 5506,
"isocode": "RU",
"pin": [
  37.617494,
  55.750446
],
"ref": "030000000026FCFD",
"type": "city"
}
]
}
```

Time zone detection

/timezone — service call point to determine the time zone anywhere in the world, as well as the time offset relative to UTC.

Request

Required parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to service”	<code>fa749bace6d8a3b1....</code>
q	string	Search request body. For reverse geocoding — coordinates in lat, lon format, where: <ul style="list-style-type: none">• lat — latitude of the desired point in degrees (6 decimal places are used);• lon — longitude of the desired point in degrees (6 decimal places are used)	<code>q=55.479205, 37.32733</code>

Additional parameters

Field name	Format	Description	Example
timestamp	unix timestamp	Timestamp in UTC zone for which timezone parameters will be calculated. If not specified, the current time will be used.	<code>timestamp=1264753640</code>

Response

Field name	Format	Description	Example
tzid	string	Time zone ID defined by CLDR	<code>"tzid": "Europe/Moscow"</code>

tzname_short	string	Time zone short name	"tzname_short": "MSK"
utc_delta	int	Time zone offset of the requested location from UTC (in seconds)	"utc_delta": 10800

Example

Request

https://geo.rustore.ru/api/timezone?api_key=<YOUR_API_KEY>&q=55.479205,37.32733

Response

```
{
  "request": "/v1/timezone?api_key&q=55.479205,37.32733",
  "results": [{
    "tzid": "Europe/Moscow",
    "utc_delta": 10800
  }]
}
```


Postal code search

/postcode — helps to find:

- addresses which use the requested postal code;
- postal code to the address;
- for Russia, you can also request information about a post office

Request

Required parameters

Field name	Format	Description	Example
api_key	hex-string	See “Access to services”	<code>api_key=fa749bace6d8a3b1</code>
q	string	Search request containing postal code, ref addresses, or address for which the postal code should be found	<code>q=125167 q=020000003792BE0E q=Moscow, Naryshkina str, 5b1</code>

Additional parameters

Field name	Format	Description	Example
------------	--------	-------------	---------

fields	fieldname1,fieldname2,...fieldnameN	Select fields to display in the response. Possible values: <ul style="list-style-type: none"> ● postoffice — detailed information about the Russian Post office; ● postcode (default) — found/searched postal code; ● addresses (default) — addresses which use the found postal code. 	<code>fields=postoffice,postcode,addresses</code>
addresses	string	Response format for the addresses field: <ul style="list-style-type: none"> ● short (default) — short hierarchical representation of addresses used by the found postal code. 	<code>addresses=full</code>
isocode	char2	2-character country code according to ISO 3166-1 alpha-2 <i>Only RU is currently supported</i>	<code>isocode=ru</code>

Response

Field name	Format	Description	Example
request	string	Request	<code>"request": "125167"</code>

results	list	Results	<pre> "results": [{ "addresses": { "country": "Russia", "localities": [{ "name": "Moscow", "streets": [{ "buildings": [{ "b1": "22", "name": "Starokachalovskaya street" }] }, { "region": "Moscow", "subregion": "Southwestern Administrative District" }, { "postcode": "117216" }] }] } }] </pre>
postcode	string	Postal code	<pre> "postcode": 125167 </pre>

addresses	dict	Addresses which use the found postal code	<pre>addresses=short: "addresses": { "country": "Russia", "localities": [{ "name": "Moscow", "streets": [{ "buildings": ["8 K1", "1D", "1B", "1A", "1 K1"], "name": "Starokachalovskaya street" }, { "buildings": ["4G", "6", "6 b2", "2 b1", "2A"], "name": "Koktebelskaya street" }, { "buildings": ["16", "18", "20 b1", "10 b1", "1G", "9 b2"], "name": "Kulikovskaya street" }] }], "region": "Moscow", "subregion": "Southwestern Administrative District" }</pre>
-----------	------	-------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

postoffice	dict		<pre> "postoffice": { "address": "Russia, Moscow, South-Western Administrative District, Moscow, Severnoye Butovo district, microdistrict 3, Grina street, 5B", "address_details": { "building": "5B", "country": "Russia", "isocode": "RU", "locality": "Moscow", "postal_code": "117216", "region": "Moscow", "street": "Grina str", "sublocality": "microdistrict 3", "subregion": "Southwestern Administrative District", "suburb": "Northern Butovo" }, "pin": [37.570932, 55.566898], "ref": "22000000023FE098" } </pre>
------------	------	--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Post office fields description

Field name	Format	Description	Example
address	string	Full address	<pre> "address": "Russia, Moscow, Northern Administrative District, Moscow, Khoroshevsky, Leningradsky Prospekt, 56" </pre>

address_details	list	Detailed information about the found address	<pre>"address_details": { "building": "56", "country": "Russia", "isocode": "RU", "locality": "Moscow", "postal_code": "125167", "region": "Moscow", "street": "Leningradsky Prospekt", "subregion": "Northern Administrative District", "suburb": "Khoroshevsky" }</pre>
opening_hours	list	Post office hours	<pre>"opening_hours": { "Mo": [09:00-13:00,14:00-21:00], "Tu": [09:00-13:00,14:00-21:00], "We": [09:00-13:00,14:00-21:00], "Th": [09:00-13:00,14:00-21:00], "Fr": [09:00-13:00,14:00-21:00], "Sa": [10:00-18:00], "Su": [] }</pre>
ref	hex	Post Office ID <i>Object ID is not stable and may change</i>	<pre>"ref": "1000000C4D63818"</pre>
pin	list	Post office coordinates (longitude and latitude)	<pre>"pin": [37.538851, 55.796731</pre>

]
--	--	--	---

Examples

Request

`https://geo.rustore.ru/api/postcode?api_key=<YOUR_API_KEY>&q=125167&fields=*`

Response

```
{
  "request": "/postcode?q=125167&fields=*",
  "results": [
    {
      "addresses": {
        "country": "Russia",
        "localities": [
          {
            "name": "Moscow",
            "streets": [
              {
                "buildings": [
                  "56/2",
                  "37B",
                  "39 b3",
                  "39 b9",
                  "39",
                  "39",
                  "37",
                  "37 b8",
                  "39",
                  "37 b12",
                  "39 b17",
                  "36 b37",
                  "36 b30",
                  "36 b38"
                ],
                "name": "Leningradsky Prospekt"
              },
            ]
          }
        ]
      }
    }
  ]
}
```

```

{
  "buildings": [
    "7A",
    "5",
    "3",
    "4 c4",
    "4A",
    "4",
    "4",
    "4 b12",
    "4 b11",
    "4 b7",
    "2 b1",
    "4 b6",
    "4 b8"
  ],
  "name": "Novy Zykovsky proezd"
},
{
  "buildings": [
    "1",
    "5 ",
    "5 b1",
    "10",
    "3",
    "5 b3"
  ],
  "name": "Seregina str"
},
{
  "buildings": [
    "39 b12"
  ],
  "name": "Khodynsky Boulevard"
},
{
  "buildings": [
    "11",
    "11A",
    "11B",
    "3",

```



```

        "10",
        "11A b1",
        "6",
        "8 b3",
        "8",
        "8 b14",
        "8 b11",
        "1A"
    ],
    "name": "Airport passage"
},
{
    "buildings": [
        "5 b2",
        "5 b1"
    ],
    "name": "Naryshkinskaya alley"
},
{
    "buildings": [
        "4",
        "5 b3",
        "5 b2",
        "5 b1",
        "8 b2",
        "8 b1",
        "7",
        "6",
        "3",
        "3B"
    ],
    "name": "Konstantin Simonov street"
},
{
    "buildings": [
        "4",
        "6",
        "4A",
        "8",
        "5 b25",
        "5 b22",

```

```

        "5 b8",
        "5 b7",
        "5 b2",
        "5 b9",
        "5 b11",
        "5 b25",
        "5 b26",
        "5 b35",
        "5 b27A",
        "5 b19",
        "5 b20"
    ],
    "name": "Aviation Lane"
},
{
    "buildings": [
        "4",
        "5",
        "3"
    ],
    "name": "Eldoradovsky Lane"
},
{
    "buildings": [
        "2A",
        "2A b2",
        "3/5",
        "1A",
        "2 b1",
        "2 b2",
        "2 b3",
        "2 b5"
    ],
    "name": "Krasnoarmeyskaya street"
},
{
    "buildings": [
        "3"
    ],
    "name": "Stepan Suprun street"
},

```

```

    {
      "buildings": [
        "3",
        "4",
        "6 b2",
        "6 b1",
        "5"
      ],
      "name": "Old Zykovsky Proezd"
    },
    {
      "buildings": [
        "3",
        "3 b2"
      ],
      "name": "1st Street March 8"
    },
    {
      "buildings": [
        "3"
      ],
      "name": "Trudovaya alley"
    },
    {
      "buildings": [
        "7",
        "5",
        "9",
        "9A",
        "11"
      ],
      "name": "Pilot Nesterov street"
    }
  ]
}
],
"region": "Moscow",
"subregion": "Northern Administrative District"
},
"postcode": "125167",
"postoffice": {

```

```
    "address": "Russia, Moscow, Northern Administrative District,  
Moscow, Airport district, Leningradsky Prospekt, 56",  
    "address_details": {  
      "building": "56",  
      "country": "Russia",  
      "isocode": "RU",  
      "locality": "Moscow",  
      "region": "Moscow",  
      "street": "Leningradsky Prospekt",  
      "subregion": "Northern Administrative District",  
      "suburb": "Airport area"  
    },  
    "pin": [  
      37.537903,  
      55.798518  
    ],  
    "ref": "2100000045438C4B"  
  }  
}  
]  
}
```

RuStore Deeplinks

Using deeplinks, you can open some RuStore screens while within your app.

For example, you can open a screen with a list of all user's subscriptions using the code below:

```
try {
    startActivity(Intent(Intent.ACTION_VIEW,
Uri.parse("rustore://profile/subscriptions")))
} catch (ex: ActivityNotFoundException) {
    // Handle error when RuStore is not installed
}
```

where "rustore://profile/subscriptions" is a RuStore deeplink. To open the required RuStore app screen, you can replace it with any of the deeplinks listed below.

List of deeplinks

Function	Deeplinks
App Screen	<ul style="list-style-type: none">• rustore://apps.rustore.ru/app/{packageName} — opens the RuStore app;• market://details?id={packageName} — suggests to open RuStore or other app stores. where {packageName} — app package name.
Subscriptions Screen	rustore://profile/subscriptions
Account/Updates Screen	rustore://apps.rustore.ru/updates
Login Screen	rustore://auth

Task API

Task — asynchronous task that returns an error or value in the corresponding callback (onFailure, onSuccess).

Below is an implementation example which applies the SDK payment method `getProducts()`.

Task processing example

Async methods return `Task<t>`. For example, `RuStoreBillingClient.getProducts()` returns `Task<productsResponse>`. Thus, Task will return `ProductsResponse` if the method appears to be successful:

```
val task: Task<ProductsResponse> =  
RuStoreBillingClient.products.getProducts()
```

Add callback `OnSuccessListener` to Task in order to get the successful method result:

```
val task: Task<ProductsResponse> =  
RuStoreBillingClient.products.getProducts()  
task.addOnSuccessListener {  
    // Process success  
}
```

To get an execution error, add callback `OnFailureListener` to Task:

```
val task: Task<ProductsResponse> =  
RuStoreBillingClient.products.getProducts()  
task.addOnFailureListener {  
    // Process error  
}
```

Add `onCompleteListener` to Task if you need to get both a successful result and error:

```
val task: Task<ProductsResponse> =  
RuStoreBillingClient.products.getProducts()  
task.addOnCompleteListener(object :  
OnCompleteListener<ProductsResponse> {  
    override fun onFailure(throwable: Throwable) {  
        // Process Error  
    }  
})
```

```

        override fun onSuccess(result: ProductsResponse) {
            // Process success
        }
    })

```

Multithreading

Callback added to Task are performed on the main app thread. To run callback in other threads, transfer your Executor to the Callback method.

Adding executor via coroutines:

```

val task: Task<ProductsResponse> =
RuStoreBillingClient.products.getProducts()
task.addOnCompleteListener(Dispatchers.IO.asExecutor(), object :
OnCompleteListener<ProductsResponse> {
    override fun onFailure(throwable: Throwable) {
        // Process Error
    }

    override fun onSuccess(result: ProductsResponse) {
        // Process success
    }
})

```

Synchronous execution

You can use `task.await()` if your code is already executed in the working thread and you need to get the result synchronously:

```

try {
    val task: Task<ProductsResponse> =
RuStoreBillingClient.products.getProducts()
    task.await()
} catch (e: CancellationException) {
    // Process error
}

```

Please note that when calling `await()`, the main (UI) thread returns `IllegalStateException` error.

Task API processing with Coroutines

You can use the following code for Task processing with Coroutines:

```
suspend fun <T> Task<T>.wrapInCoroutine(): Result<T> {
    return suspendCancellableCoroutine { continuation ->
        addOnCompleteListener(object : OnCompleteListener<T> {

            override fun onSuccess(result: T) {
                if (continuation.isActive) {
                    continuation.resume(Result.success(result))
                }
            }

            override fun onFailure(throwable: Throwable) {
                if (continuation.isActive) {
                    continuation.resume(Result.failure(throwable))
                }
            }
        })

        continuation.invokeOnCancellation {
            cancel()
        }
    }
}
```


Getting started with RuStore APIs

The API host is public-api.rustore.ru

List of available RuStore API methods

Name	Method	Description
Retrieving payment and subscription data using API (common methods)		
Retrieve payment data using a purchase token	GET https://public-api.rustore.ru/public/purchase/{purchaseToken}	This method allows retrieving payment info for a purchase token.
Retrieve subscription data via a subscription token	GET https://public-api.rustore.ru/public/subscription/{subscriptionToken}	This method allows retrieving subscription info using a subscription token.
Retrieve subscription status by a subscription token	GET https://public-api.rustore.ru/public/subscription/{subscriptionToken}/state	This method allows retrieving subscription status using a subscription token.
Retrieving payment and subscription data using API (application methods)		
Retrieve subscription data via a subscription token (V2)	GET https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{subscriptionToken}	This method allows retrieving subscription info using a subscription token.

Confirm subscription via a subscription token	POST https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{purchaseToken}:acknowledge	The method allows confirming subscription using a subscription token.
Uploading and publishing applications using the RuStore API		
Creating a draft release	POST https://public-api.rustore.ru/public/v1/application/{packageName}/version	This method allows creating a draft app version with basic information.
Uploading an APK file	POST https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/apk	This method allows uploading an .apk file for publication.
Uploading an app icon	POST https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/icon	This method allows uploading an application icon.
Uploading screenshots	POST https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/screenshot/{orientation}/{ordinal}	This method allows uploading app screenshots.
Changing publication settings	POST https://public-api.rustore.ru/public/v1/application/{packa	This method allows changing the publication type, deferred publication

	geName}/version/{versionId }/publish-settings	date and % for partial release.
Getting app version status	GET https://public-api.rustore.ru/public/v1/application/{packageName}/version?ids=704095&page=0&size=2	This method allows retrieving basic app info, as well as checking the version status.
Submitting a draft app release for review	POST https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId }/commit?priorityUpdate={priorityUpdate}	This method allows submitting a draft app version for review
Manual publication	POST https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId }/publish	The method allows publishing a manually moderated version.
Deleting a draft release	DELETE https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId }	This method allows deleting previously created drafts.
Feedback management using the RuStore API		
Getting app feedback	GET https://public-api.rustore.ru/public/v1/application/{packageName}/comment?id={id}&page={number}&size={size}	This method allows retrieving a list of all the latest reviews for your app or either a single review.

Receiving feedback in .csv	GET https://public-api.rustore.ru/public/v1/application/{packageName}/comment/export?from={date_from}&to={date_to}	This method allows retrieving all reviews in .csv for a certain period of time.
Leaving a reply to review	POST https://public-api.rustore.ru/public/v1/application/{packageName}/feedback?commentId={commentId}	This method allows replying to a review.
Getting review response status	GET https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}	This method allows retrieving moderation status of a response to a review.
Getting the status of responses to reviews	GET https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/?id={id}&page={number}&size={size}	This method allows retrieving moderation status of a review response or obtaining a single response info.
Editing review response	POST https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}	This method allows editing a review response body.
Deleting review response	DELETE https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}	This method allows deleting a response to a review.

Getting app rating	POST https://public-api.rustore.ru/public/v1/application/{packageName}/comment/statistic	This method allows obtaining an app rating.
--------------------	---------------------------------------------------------------------------------------------	---------------------------------------------

How to sign up and start using API RuStore

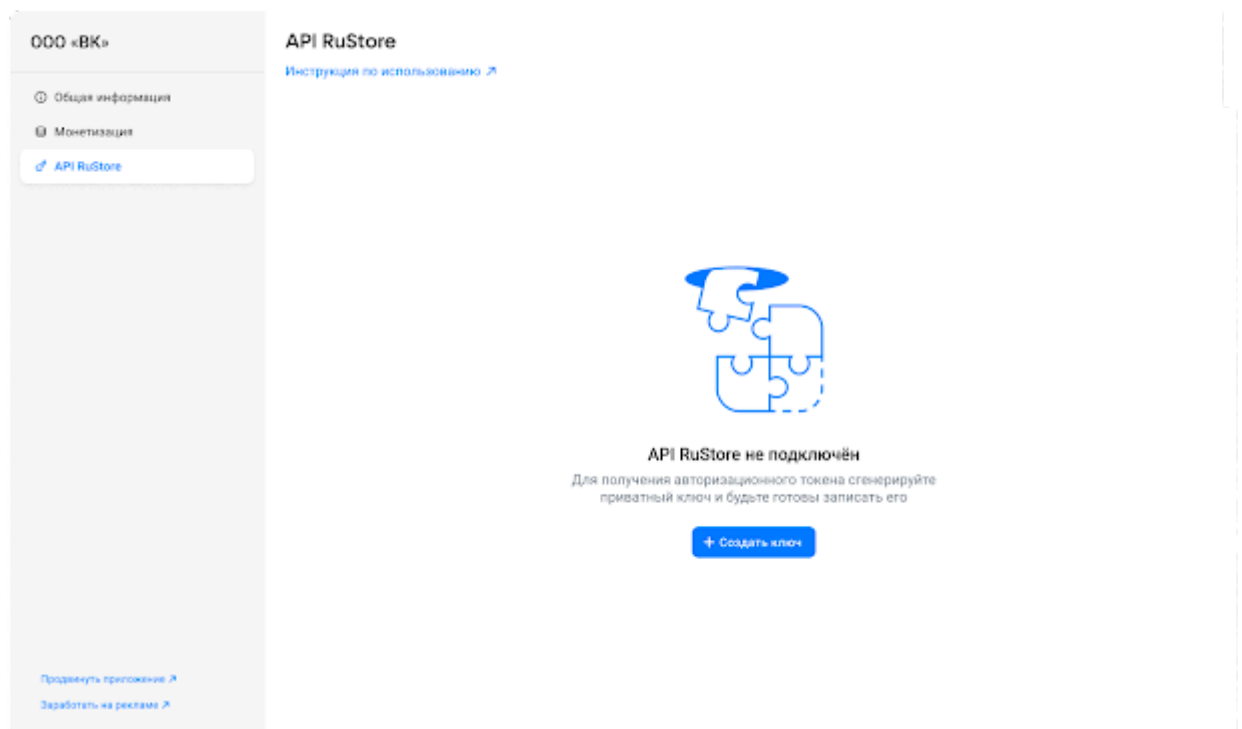
Please log in to gain access to the RuStore API. To do this, generate a key pair - public-private key - in RuStore Console and receive a JWE token.

Create API Keys

To generate keys, you must use the RSA encoding algorithms.

The keys will be generated using the RSA encoding algorithms.

1. Open the [RuStore Console](#).
2. Go to the "Company" tab in the upper part of the screen (be sure to log in to legal entity account).
3. On the left, select "API RuStore".
4. Click "Generate a key".



5. Enter the key name (up to 255 characters).

Создание ключа

Подробнее читайте [в справочнике разработчика](#)

Название ключа

Приложения ⓘ Приложения, к методам RuStore API которых можно получить доступ с помощью приватного ключа

Методы приложений ⓘ [Методы RuStore API](#), к которым будет иметь доступ приватный ключ

Общие методы ⓘ Некоторые [методы RuStore API](#) применяются ко всем приложениям разработчика вне зависимости от выбранных

- Select the applications to which the private key will be distributed.

Note. You can select one or more apps, or all at once. If you check the box next to “All apps” and then create a new app when generating a key, the key will not apply to it.

Выберите приложение

Все приложения

VK Почта

Маруся — голосовой помощник

ВКонтакте

- Select one or more RuStore API methods. You can choose both application methods the private key will have access to, and general ones. General methods will transfer data from all applications, regardless of which app the key is configured for.

The public key is generated automatically and stored in the database. To sign in, you only need to know the value of the generated private key.

Методы приложений ⓘ

Выберите метод ^

Получение данных платежа и подписки

- Получение данных подписки по токену подписки (V2)
- Подтверждение получения подписки по токену подписки

Работа с отзывами

- Получение отзывов приложения
- Получение отзывов в формате CSV

Общие методы

Выберите метод ^

Получение данных платежа и подписки

- Данные платежа по токену покупки
- Данные подписки по токену подписки
- Получение статуса подписки по токену подписки

8. Click "Generate a key".

Создание ключа

Подробнее читайте [в справочнике разработчика](#)

Название ключа

App key

Приложения ⓘ

VK Почта × ВКонтакте ×

Методы приложений ⓘ

Подтверждение получения подписки по токenu подписки ×

Получение отзывов приложения ×

Получение отзывов в формате .csv ×

Получение рейтинга приложения ×

Создание черновика версии × Удаление черновика ×

Общие методы ⓘ

Данные платежа по токenu покупки ×

Отмена

Сгенерировать ключ

9. Once the private key appears in the “Private Key” pop-up window, copy and save it properly.

The public key is generated automatically when it is generated and is recorded in the database regardless of the user. To sign in, you only need the value of the generated private key.

Key pair generation is available to company owners only.

If there is a key pair, the created private key will appear in RuStore Console in a table. It indicates the key name and ID, the selected methods and applications, and the key creation or update date.

Update a key

1. Open the [RuStore Console](#).
2. Go to the "Company" or "Developer" tab in the upper part of the screen.
3. In the names section, find the required key.
4. Click on the action bar.
5. Then select "Update".
6. The public key will be automatically renewed.

Remove API keys

You can remove a public and private key on the [RuStore Console](#).

1. Open the [RuStore Console](#).
2. Go to the "Company" or "Developer" tab in the upper part of the screen.
3. In the names section, find the required key.
4. Click on the action bar.
5. Click "Remove" in the key section.

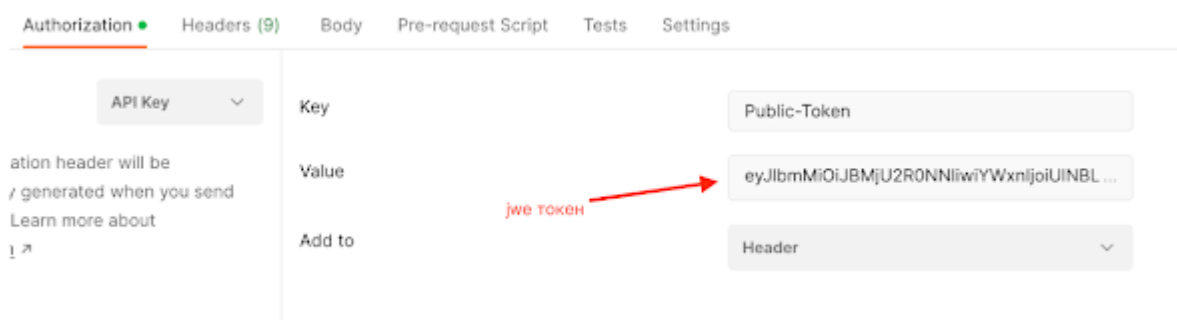
Receive a token

Use the generated private key to obtain the JWE token via the POST `/public/auth/` method.

The jwe token is valid for 900 seconds, then it must be obtained again by repeating the POST `/public/auth/` method.

When signed in

The resulting JWE token allows you to use the capabilities of the RuStore API. To do this, its value must be transferred to the "Authorization" section using the API key.



Generating an authorization token

This method allows you to generate a JWE token using a private key obtained on the RuStore Console. The method also checks the app owner's activities.

A private key is required to successfully complete the request.

Interaction parameters

POST /public/auth/

Attribute	Type	Required	Location	Description	Example content
companyId	string	Yes	body	Company id	123
keyId	string	Yes, if companyId is not specified	body	key ID	123
timestamp	string	Yes	body	Response time delay should not exceed 60 seconds compared to the current server time.	2022-07-08T13:24:41.8328711+03:00
signature	string	Yes	body	RSA-signature SHA-512 hash in a line that contains companyId and timestamp of a response. Signature layout and verification algorithm: SHA512withRSA Example: companyId: 123 timestamp: 2022-07-08T13:24:41.8328711+03:00 string for hash: 1232022-07-08T13:24:41.8328711+03:00 SHA-512 hash: 0976c61cce96fcd9daaae5f594db43dd287c0e266561669184276a2e86578c0e2a39cd0b183a458d0e47b17c68548daac83db97bc710dcd07d01bae40033235	

If you have problems understanding the signature parameter algorithm, use a [companyld jar file](#) or a [keyld jar file](#) to generate this parameter.

You need to pass the company id and the resulting private key. In response, you will receive a body for the POST request /public/auth/ with companyld, timestamp and signature parameters.

The generated signature parameter is valid for 1 minute, since it directly depends on the timestamp.

Response example

Successful

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	Yes	Response code definition	Range timestamp not valid
body{}	object	Yes	Body	
timestamp	timestampz	Yes	Time	2022-07-08T13:24:41.8328711+03:00

body{}

Attribute	Type	Required	Description	Example content
jwe	string	Yes	Security token to API RuStore (payload is below), re-usable.	eyJjdHkiOiJK...sv16aBl8tTg.VkWuTw
ttl	int	Yes	Token validity period, sec By default, the token validity periods are set to 900 seconds	900

Possible errors

code	message	Description	Possible solution
400	Range timestamp not valid	The timestamp parameter differs by more than 60 seconds	Update timestamp and signature
404	Company key not found	The private key was not found for the transmitted companyId	Check whether a private key has been created for this companyId and whether it is up to date

400	Company key disabled	The private key for the transferred companyld has been deleted	Check whether a private key has been created for this companyld and whether it is up to date
400	Signature encode error	The signature parameter was generated incorrectly	Regenerate the signature parameter
404	You cannot use this action because the company is not found	The company that corresponds to the passed companyld is missing or inactive	Check the companyld parameter to ensure it is up to date
400	You can't use this action because the company is banned	The company with the transferred company ld is blocked	Check the companyld parameter to ensure it is up to date

Request example

```
curl --location 'https://public-api.rustore.ru/public/auth' \
--header 'Content-Type: application/json' \
--data '{
  "companyld":"1275328",
  "timestamp":"2023-08-11T13:31:17.580+03:00",
  "signature":"U4kh.....nFkbuw=="
}
```

Successful response

```
{
  "code": "OK",
  "message": null,
  "body": {
    "jwe": "eyJlbnMiOiJBM.....nuuM227D_O1A",
    "ttl": 900
  },
  "timestamp": "2023-08-11T13:31:33.171847393+03:00"
}
```

How to retrieve payment data using a purchase token

This method enables you to retrieve payments using a purchase token.

Interaction examples

For real payments:

GET

Unset

```
https://public-api.rustore.ru/public/purchase/{purchaseToken}
```

For test payments:

GET

Unset

```
https://public-api.rustore.ru/public/sandbox/purchase/{purchaseToken}
```

If you want to work with Test Payments and Subscriptions, you'll need to create a new key, and when you create the key, you'll need to specify the methods to test.

GET <https://public-api.rustore.ru/public/purchase/{purchaseToken}>

Attribute	Type	Required	Description	Location	Example content
Public-Token	string	Security token to Public API Rustore	Yes	header	
subscription_token	string	How to get a purchase token	Yes	path	111.123

Response parameters

Attribute	Type	Description	Required	Location	Example content
code	number	Response code	Yes	body	OK, ERROR, BAD_REQUEST, NOT_FOUND

message	date	Response code definition	No	body	
timestamp	string	Response time	Yes	body	
body{}	object	Response body	No	body	

body{}

Attribute	Type	Description	Example
error {}	object	Error definition	
invoice_id	string	Account number	12345
invoice_date	string	Account creation date	2020-04-29T08:18:03+03

invoice_status	string	Account status	created - account successfully created; executed - user has selected a payment method, the payment is executed; cancelled - canceled by the user; paid - funds are reserved, the account is awaiting confirmation (for consumable products only); confirmed — payment was successful; reversed, refunded — funds from the account have been returned to the buyer.
invoice {}	object	Account info	
image	string	Link to picture	https://i-love-png.com/images/grim-reaper-icon.png
application_code	string	Application code	com.MashaAndTheBear.HairSalon
application_name	string	App name	Masha and the Bear Hair Salon
owner_code	string	App owner code	com.MashaAndTheBear
owner_name	string	App owner name	Masha and the Bear

payment_info {}	object	Payment info	
payment_methods{}	object	Payment tools	

body.error {}

Attribute	Type	Description	Example
user_message	string	Error text	
error_description	object	Error description	
error_code	number	Numeric error code	0

body.invoice {}

Attribute	Type	Description	Example
purchaser{}	object	Buyer information	
delivery_info{}	object	Delivery information	
invoice_params[]	array	Additional order parameters	
order{}	object	Order info	

body.invoice.purchaser {}

Attribute	Type	Description	Example
email	string	Buyer's email	qq@dd.eof
phone	string	Phone number	9123456789
contact	string	Preferred communication type	email

body.invoice.delivery_info {}

Attribute	Type	Description	Example
address{}	object	Address	
delivery_type	string	Delivery method	courier
description	string	Additional Information	Call back in 1.5 hours

body.invoice.delivery_info.address {}

Attribute	Type	Description	Example
country	string	Country ID	RU
city	string	City	Moscow
address	string	Address	st. Vavilova, 19, office 1

body.invoice.invoice_params []

Attribute	Type	Description	Example
key	string	Parameter name	packageName
value	string	Parameter value	com.MashaAndThe Bear.HairSalon

body.invoice.order {}

Attribute	Type	Description	Example
order_id	string	Unique order ID	d290f1ee-6c54-4b01-90e6-d701748f0851
order_number	string	Order number	145
order_date	string	Order date	2020-04-29T08:17:03+03
service_id	string	Service ID	223

amount	number	Order amount in minimum currency units (kopecks)	11836
currency	string	Currency code	RUB
purpose	string	Brief purpose of payment	Purchase in the game "Masha and the Bear Hair Salon"
description	string	Order Description	In-game item purchase in "Masha and the Bear Hair Salon"
language	string	Language	ru-RU
expiration_date	string	Account expiration date and time	2022-10-11T14:05:44.741Z
tax_system	number	Tax system	0
trade_name	string	Company trade name	Romashka
visual_name	string	Company name	Purchasing/renewing a subscription
org_name	string	Company name	LLC Romashka
org_inn	string	Company TIN	1234567890
visual_amount	string	Amount	1 500,45 rubles
order_bundle []	array	Order list	

body.invoice.order.order_bundle []

Attribute	Type	Description	Example
-----------	------	-------------	---------

position_id	number	Unique item ID	1
name	string	Item name and description	Crystals
item_params[]	array	Additional order parameters	
quantity{}	object	Description of the total number of product items	
item_amount	number	Total cost of all items in minimum currency units (in kopecks)	11836
currency	string	Currency code	RUB
item_code	string	Item number (ID)	com.MashaAndTheBear.HairSalon.crystal100
item_price	number	Cost of one product item in minimum currency units (kopecks)	11836
discount_type	string	Discount type	percent
discount_value	float	Discount value	5.25
interest_type	string	Agency fee type	agentPercent
interest_value	float	Agency fee value	15.105
tax_type	number	VAT rate	6

tax_sum	number	Tax amount in minimum currency units (kopecks)	2367
image	string	Link to picture	https://i-love-png.com/images/grim-reaper-icon.png

body.invoice.order.order_bundle.item_params []

Attribute	Type	Description	Example
key	string	Parameter name	packageName
value	string	Parameter value	com.MashaAndTheBear.HairSalon

body.invoice.order.order_bundle.quantity {}

Attribute	Type	Description	Example
value	float	Value	1.05
measure	string	Unit	kg

body.payment_info {}

Attribute	Type	Description	Example
payment_date	number	Payment date and time	2022-10-11T14:05:44.741Z
payment_id	string	Unique payment ID	d290f1ee-6c54-4b01-90e6-d701748f0851
payment_params{}	object	Additional payment options	
device_info{}	object	Device Information	

loyalty_info{}	object	Loyalty program information	
card_id	string	Unique bank card ID	ad454ffg-6c54-4b01-90e6-d701748f0851
name	string	Cardholder name	Main
paysys_code	string		RBS-shortname
masked_pan	string	Masked card number	**1111
expiry_date	string	Card expiration date	201912
cardholder	string	Cardholder name	Ivan Petrov
payment_system	string	Payment system	Visa
payment_system_image	string	Link to payment system logo	https://smartmarkettstift.online.sberbank.ru/icons/logo_visa.png
image	string	Link to card logo	https://smartmarkettstift.online.sberbank.ru/icons/sberbank_mastercard_league_legends.jpeg
paysys	string	Name of payment operator	RBS
paysys_image	string	Link to payment operator logo	https://www.sberbank.ru/common/img/uploaded/redirected/s_m_business/acquiring/assets/images/intro@2x.png
payment_way	string	Payment Method	SberPay
payment_way_code	string	Payment Method ID	SberPay

payment_way_logo	string	Link to payment method logo	https://cdn1.telegram.one/i/f7640dada78306b1c993e04001b8738d/828b1eb30921659e22e53a9edc92c4c4/24e01830d213d75deb99c22b9cd91ddd
bank_info{}	object	Bank information	

body.payment_info.payment_params {}

Attribute	Type	Description	Example
key	string	Parameter name	googlePurchaseToken
value	string	Parameter value	ameinkbophchljajnocadib

body.payment_info.device_info {}

Attribute	Type	Description	Example
device_platform_type	string	Device platform	iOS
device_platform_version	string	Platform OS version	13.6.1
device_model	string	Device model	iPhone 7
device_manufacturer	string	Manufacturer	Apple
device_id	string	Device serial number	83c3f257-46d8-41fe-951b-f79d04e288c2
surface	string		RuStore
surface_version	string	Software version	11.5.0

body.payment_info.loyalty_info {}

Attribute	Type	Description	Example
-----------	------	-------------	---------

service_code	string	Bonus program code	sbrf_spasibo
service_name	string	Bonus program name	Sberbank Spasibo
change_rate	number	Exchange rate for points to rubles	1
payment_bonus	number	Amount of bonus points used to pay the bill, in kopecks	19800
award_bonus	number	Amount of funds used to earn points when paying for an order. Indicated in minimum units (kopecks)	21850
image	string	Link to picture	https://i-love-png.com/images/grim-reaper-icon.png

body.payment_info.bank_info{}

Attribute	Type	Description	Example
bank_name	string	Issuing bank name	PJSC Sberbank
bank_country_code	string	Issuing bank country code	RU
bank_country_name	string	Bank country name	Russia
bank_image	string	Link to bank logo	https://emoji.slack-edge.com/TKK9DHN/CV/sber/ad2df81a6cd9812d.png

body.payment_methods {}

Attribute	Type	Description	Example
user_message	string	Message to user (optional)	To activate the subscription, save your bank card in the mobile app
methods[]	array	Payment options	

body.payment_methods.methods []

Attribute	Type	Description	Example
method	string	Code	QR
action	string	Name	Pay via QR code

Response example:

```

{
  "code": "OK",
  "message": null,
  "body": {
    "invoice_id": "2850",
    "invoice_date": "2023-07-18T14:31:33+03",
    "invoice_status": "confirmed",
    "application_code": "3399750",
    "application_name": "Masha and the Bear",
    "owner_code": "4384191",
    "owner_name": " \"Narana\ LLC\"",
    "payment_info": {
      "payment_date": "2023-07-18T14:31:42+03",
      "payment_id": "1736",
      "payment_params": null,
      "loyalty_info": null,
      "card_id": "193",
      "paysys_code": "RBS-shortname",
      "masked_pan": "XX1111",
      "expiry_date": "202412",
      "payment_system": "Visa",
      "payment_system_image":
"https://smartmarket.online.sberbank.ru/image/visa.png",
      "paysys_image": null,
      "payment_way": "Paid with a saved card",
      "payment_way_code": "CARD_BINDING",
      "payment_way_logo":
"https://static.tildacdn.com/tild6236-3530-4235-b966-326630656238/\_\_\_14\_-removebg-prev.png",
      "bank_info": {
        "bank_name": "Sberbank",
        "bank_country_code": "SU",
        "bank_country_name": null,
        "bank_image": null
      },
      "device_info": null,
      "name": null,
      "cardholder": "CARDHOLDER NAME",
      "image": null,
      "paysys": "RBS"
    },
    "payment_methods": null,
    "error": {
      "user_message": "Account verified",
      "error_description": "",

```

```

        "error_code": 0
    },
    "invoice": {
        "delivery_info": {
            "delivery_type": null,
            "address": {
                "country": null,
                "city": null,
                "address": null
            },
            "description": null
        },
        "invoice_params": [
            {
                "key":
                "inapp_serviceparam_message_about_loyalty",
                "value": "You can now pay for your
subscription with bonuses. Subscription renewal is available for
rubles only."
            },
            {
                "key": "inapp_serviceparam_action_name",
                "value": "Enable subscription"
            },
            {
                "key": "inapp_serviceparam_features",
                "value": "VERIFY"
            },
            {
                "key": "period_type",
                "value": "DAY"
            },
            {
                "key": "period_duration",
                "value": "1"
            },
            {
                "key": "current_period",
                "value": "STANDARD"
            },
            {
                "key": "payment_type",
                "value": "INITIAL"
            }
        ]
    },

```

```

    "purchaser": {
      "email": null,
      "phone": null,
      "contact": null
    },
    "order": {
      "order_id":
"a090a93c-ca06-493d-a90a-ce2bac722358",
      "order_number": "311",
      "order_date": "2023-07-18T14:31:33+03",
      "service_id": "4720",
      "expiration_date": "2023-07-18T14:51:33+03",
      "tax_system": null,
      "trade_name": null,
      "visual_name": "Purchasing/Renewing a
Subscription",
      "org_name": "Super LLC",
      "org_inn": "4419198349",
      "visual_amount": "1 ₺",
      "order_bundle": [
        {
          "position_id": 1,
          "item_params": [
            {
              "key":
"_auto_itemAttributes_agent_info.type",
              "value": "7"
            },
            {
              "key":
"_auto_itemAttributes_supplier_info.name",
              "value": "LLC \"Narana\""
            },
            {
              "key":
"_auto_itemAttributes_supplier_info.inn",
              "value": "4419198349"
            }
          ],
          "item_amount": 100,
          "item_code": "1day",
          "item_price": 100,
          "discount_type": null,
          "discount_value": null,
          "interest_type": null,

```

```

        "interest_value": null,
        "tax_type": 6,
        "tax_sum": null,
        "name": "Payment for purchasing a
\"1day\" subscription. Provider: \"Purchase/Renewal of
Subscription\"",
        "quantity": {
            "value": 1,
            "measure": "pc"
        },
        "currency": "RUB",
        "image": ""
    }
],
"amount": 100,
"currency": "RUB",
"purpose": "1day",
"description": "1day",
"language": "ru-RU"
}
},
"image": ""
},
"timestamp": "2023-08-02T10:11:04.655684723+03:00"
}

```

Response verification

The security token is generated by the security token method and then verified while getting the response:

- Owner and/or app should not be blocked;
- Token should be valid;
- application_code invoice should contain the application code which corresponds to the token owner.

How to retrieve subscription data via a subscription token

This method enables retrieval of subscription data from a payment service provider via a subscription token.

Interaction parameters

GET

```
https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{subscriptionToken}
```

Attribute	Type	Required	Description	Location	Example content
Public-Token	string	Security token to Public API Rustore	Yes	header	
subscription_token	string	How to get a purchase token	Yes	path	111.123

Response parameters

Attribute	Type	Description	Required	Location	Example content
code	number	Response code	Yes	body	OK, ERROR, BAD_REQUEST, NOT_FOUND
message	date	Response code definition	No	body	
timestamp	timestamp	Response time	Yes	body	
body{}	object	Response body	No	body	

body{}

Attribute	Type	Description	Required	Location	Example
code	number	Response code	Yes	body	
success	boolean	Response success flag	Yes	body	true
message	string	Response code description	No	body	Unknown error
body{}	object	Response body	No	body	

body.body{}

Attribute	Type	Description	Example
serviceName	string	Service name	Okko
subscriptionId	integer	Unique subscription ID	12345
addParameters	string	Additional subscription options	Something about subscription
productType	string	Product type	SUBSCRIPTION
productName	string	Product name	Okko optium
productCode	string	Product code	monthly_sub
recurrent	boolean	Recurrency sign	true

countOfDay	integer	Number of days	10
periodType	string	Period type	Available values: DAY, MONTH, YEAR
periodDuration	integer	Period duration	10
nextPaymentDate	string	Next payment date	2021-03-23
price	integer	Price in minimum currency unit (kopecks)	9999
currency	string	Currency	RUB
imageUrl	string	Link to product picture	

state	string	Subscription status	<p>Active states</p> <p>ACTIVATED — subscription is active.</p> <p>Intermediate statuses</p> <p>ACCEPTED - awaiting payment.</p> <p>DEPOSITED — payment has been made, subscription is awaiting activation;</p> <p>CLOSE_PENDING — subscription in is being terminated;</p> <p>REPEATING - subscription is at the renewal stage, attempted money withdrawal.</p> <p>Final states</p> <p>DECLINED - subscription is closed, all attempts at subsequent subscription payments have failed, GRACE and HOLD periods and the number of renewal attempts have ended</p> <p>CANCELED - payment failed by the user;</p> <p>CLOSED — subscription is closed; the user has disabled subscription auto-renewal, the paid period has</p>
-------	--------	---------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

			expired, provider has confirmed the subscription closed; REFUNDED — refund initiated to the user.
currentPeriod	string	Current period name	Available values: PROMO - free period; START — starting period with a reduced price; STANDARD — standard period; GRACE - grace period, the user has not paid for the subscription, attempts to withdraw continue, access to the content is maintained; HOLD — hold period, the user has not paid for the subscription, attempts to write off continue, access to the content must be terminated.
debtPaymentPeriod	string	Name of payment period	Null if the period is standard
description	string	Subscription product description	Description
tariffId	integer	Unique rate ID	12345

periods []	array[object]	Current period info	Rate periods
------------	---------------	---------------------	--------------

body.body.periods []

Attribute	Type	Description	Example
periodName	string	Current period name	Available values: PROMO, START, STANDARD, GRACE, HOLD
periodType	string	Period type	Available values: DAY, MONTH, YEAR
periodDuration	number	Period duration	10
periodPrice	number	Price in minimum currency unit (kopecks)	10000
nextPeriod	string	Next period name	Available values: PROMO, START, STANDARD, GRACE, HOLD

Response:

```
"code": "200",
"message": "something",
"timestamp": "2023-08-02T10:11:04.655684723+03:00",
"body":
{
  "code": 40401,
  "success": false,
  "message": "unknown error",
  "body": {
    "serviceName": "Okko",
    "subscriptionId": 100500,
    "addParameters": "something",
    "productType": "string",
    "productName": "OKKO optium",
    "productCode": "string",
    "recurrent": true,
    "countOfDay": 100,
    "periodType": "DAY",
    "periodDuration": 30,
    "nextPaymentDate": "2021-03-23",
    "price": 999,
    "currency": "RUB",
    "imageUrl":
"https://static-eu.insales.ru/images/products/1/7435/306650379/thumb\_1586524817849\_15832463664565053990106868.jpg",
    "state": "ACTIVATED",
    "currentPeriod": "STANDARD",
    "debtPaymentPeriod": "string",
    "description": "DESCRIPTION",
    "tariffId": 100500,
    "periods": [
      {
        "periodName": "STANDARD",
        "periodType": "DAY",
        "periodDuration": 10,
        "periodPrice": 10000,
        "nextPeriod": "STANDARD"
      }
    ]
  }
}
```

Response verification

The security token is generated by the security token method and then verified while getting the response:

- Owner and/or app should not be blocked;
- Token should be valid;
- The invoice in application_code response should contain the application code which corresponds to the token owner.

How to retrieve subscription data by an invoice ID (V2)

This method enables to retrieve subscription data from the payment provider via the first purchaseToken.

Interaction examples

GET /public/glike/subscription/{packageName}/{subscriptionId}/{purchaseToken}

Attribute	Type	Required	Description	Location	Example content
Public-Token	string	Security token to Public API Rustore	Yes	header	
packageName	string	App package name	Yes	path	
subscriptionId	string	Subscription / product code	Yes	path	
purchaseToken	string	The token consists of invoiceId and userId parameters. Token example for invoiceId = 111 and userId = 123: 111.123	Yes	path	

Successful response parameters

Attribute	Type	Description	Required	Example content
startTimeMillis	string	Subscription period, ms	Yes	1577826955637

expiryTimeMillis	string	Subscription expiry, ms	Yes	1609456386128
autoRenewing	boolean	Subscriptions renewing after expiry date	Yes	false
priceCurrencyCode	string	ISO 4217 currency code for subscription price. For example, if the price is indicated in £, then Pricecurrencycode will be "GBP".	Yes	RUB
priceAmountMicros	string	Subscription price. For tax-free countries the price does not include tax. Otherwise the price includes tax. The price is expressed in micro -units, where 1,000,000 micro -units is one unit of currency. For example, if the subscription price is 1.99 EUR, the price of Mountamoundicros is 1990,000	Yes	749000000
countryCode	string	Country/region code according to ISO 3166-1 Alpha-2 at the time of subscription	Yes	RU

paymentState	int	Subscription payment status. Possible values: 0. Waiting for payment 1. Obtaining a payment 2. Free trial version 3. Waiting for the delayed update/decrease There is no value for canceled subscriptions with an expired validity period.	No For active subscriptions only (state != CLOSED, REFUNDED, CLOSE_PENDING, ERROR, MIGRATED), it would not be passed otherwise	1
cancelReason	int	The reason why the subscription was canceled or not renewed automatically. Possible values: 0. Subscription canceled by a user 1. Subscription canceled by the system, for example, due to payment issues 2. Subscription replaced by a new subscription 3. Subscription canceled by the developer	No. For canceled subscriptions with the status of CLOSED, it would not be passed otherwise	0

orderId	string	Last repeating order ID related to the subscription purchase. If the subscription was canceled due to payment issues, this will be the order ID from the rejected payment order	Yes	41456..3
acknowledgementState	int	Subscription product status. Possible values: 0. Not yet confirmed 1. Confirmed	Yes	1
introductoryPriceInfo{}	object	Introductory subscription price information. This is true only if the subscription was purchased at the initial price. This field does not indicate that the subscription is currently purchased at the initial price.	No. Only for subscriptions with the PROMO period, it would not be passed otherwise	
kind	string	This type is an object of the purchase subscription in the androidpublisher service	Yes	androidpublisher#subscriptionPurchase

purchaseType	int	Subscription purchase type. This field is required only if this purchase was not made using the standard in-app invoicing process. Possible values: 0. Test (purchase via a license testing account) 1. Promo (purchase using a promo code)	No. For text environment only	0
--------------	-----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------	---

introductoryPriceInfo{}

Attribute	Type	Description	Required	Example content
introductoryPriceCurrencyCode	string	ISO 4217 currency code for the initial subscription price. For example, if the price is indicated in £, then PriceCurrencycode will be "GBP".	No. Only for promotional period subscriptions	RUB
introductoryPriceAmountMicros	string	The initial subscription price, tax excluded. The currency correspond to Pricecurrencycode. The price is expressed in minor units, where 1,000,000 minor units is one unit of currency. For example, if the subscription price is 1.99 EUR, the price of Mountamoundicross is 1990,000	No. Only for promotional period subscriptions	599000000

introductoryPricePeriod	string	The initial price period is indicated in the ISO 8601 format. The common values are (but not limited to) "P1W" (one week), "P1M" (one month), "P3M" (three months), "P6M" (six months) and "p1y" (one year)	No. Only for promotional period subscriptions	P1Y
introductoryPriceCycles	string	The number of estimated periods for the offered initial price	No. Only for promotional period subscriptions	1

JSON response example

```
{
  "startTimeMillis": "1694431707000",
  "expiryTimeMillis": "1697034507000",
  "autoRenewing": true,
  "priceCurrencyCode": "RUB",
  "priceAmountMicros": "749000000",
  "countryCode": "RU",
  "paymentState": 1,
  "orderId": "33252",
  "acknowledgementState": 1,
  "kind": "androidpublisher#subscriptionPurchase",
  "purchaseType": 0,
  "introductoryPriceInfo": { "introductoryPriceCurrencyCode": "RUB",
    "introductoryPriceAmountMicros": "599000000",
    "introductoryPricePeriod": "P1M", "introductoryPriceCycles": "1"
  }
}
```

Error response example

Attribute	Type	Description	Required	Example content
error{}	object	Object contains error data	Yes, if response code !=200	

error{}

Attribute	Type	Description	Required	Example content
code	number	Response http-code	Yes	
message	date	Response code description	No	

message

code	message
400	The subscription purchase token does not match the subscription ID
404	No subscription purchase matches the subscription ID

Getting subscription data (V3)

This method allows retrieving subscription info using a subscription token.

Interaction parameters

For real payments:

GET

Unset

```
https://public-api.rustore.ru/public/subscription/{subscriptionToken}/state
```

For test payments:

GET

Unset

<https://public-api.rustore.ru/public/sandbox/subscription/{purchaseToken}/state>

Attribute	Type	Required	Description	Location	Example content
Public-Token	string	Security token to Public API Rustore	Yes	header	N/a
packageName	string	App package name	Yes	path	com.MashaAndTheBear.HairSalon
subscriptionId	string	Product/subscription code	Yes	path	daily_sub
subscriptionToken	string	How to get a subscription token see in Billing SDK Documentation.	Yes	path	111.123

Attribute	Type	Description	Required	Location	Example
Public-Token	string	Jwe token to Public API Rstore			

Successful response

Attribute	Type	Description	Required	Example
startTimeMillis	string	Subscription provisioning time in milliseconds since epoch start.	Yes	1577826955637
expiryTimeMillis	string	Subscription expiration time in milliseconds from the epoch start.	Yes	1609456386128
autoRenewing	boolean	Whether the subscription will automatically renew at the end of the current expiry date.	Yes	false
priceCurrencyCode	string	ISO 4217 currency code for the subscription price.	Yes	RUB

priceAmount Micros	string	Subscription price. Price is expressed in micro units, where 1,000,000 micro units represents one unit of currency. For example, if the subscription price is 100 rubles, the price of AmountMicros is 100000000.	Yes	749000000
countryCode	string	The user's billing country/region code at the time the subscription is granted.	Yes	RU
paymentState	int	Subscription Payment Status. Available values: <ul style="list-style-type: none"> • 0 — awaiting payment • 1 — receiving payment; • 2 — free trial. <p>Not available for cancelled expired subscriptions.</p>	No, for active subscriptions only	1
cancelReason	int	The reason why the subscription was cancelled or not renewed	No, only for cancelled	0

		<p>automatically.</p> <p>Available values:</p> <ul style="list-style-type: none"> • 0 — subscription cancelled by the user • 1 — subscription cancelled by the system, e.g. due to a payment problem. 	<p>subscriptions with a status of</p> <p>CLOSED</p>	
orderId	string	<p>Last invoice ID associated with the subscription purchase. If there is more than one subscription account, their number is added to the identifier using the separator "...", starting from 0</p>	Yes	41456..3
acknowledgmentState	int	<p>Subscription Confirmation Status. Available values:</p> <ul style="list-style-type: none"> • 0 — not yet confirmed; • 1 — confirmed. 	Yes	1
introducerPriceInfo	object	<p>Information about the promotional subscription period. This field does not indicate that the</p>	No, only for subscriptions with a	N/A

		subscription is currently in a promotional period.		PROMO period.	
kind	string	It always passes the value androidpublisher#subscriptionPurchase	Yes		androidpublisher#subscriptionPurchase
purchaseType	int	It always passes 0.	Yes	0	

introductoryPriceInfo{}

Attribute	Type	Description	Required	Example
introductoryPriceCurrencyCode	string	ISO 4217 currency code for the initial subscription price.	No, only for subscriptions with a <small>PROMO</small> period.	RUB
introductoryPriceAmountMicros	string	Initial subscription price. Currency corresponds to priceCurrencyCode. Price is expressed in micro units, where 1,000,000 micro units represents one unit of currency. For example, if the subscription price is 100 rubles, the price of	No, only for subscriptions with a <small>PROMO</small> period.	599000000

		AmountMicros is 100000000.		
introductoryPricePeriod	string	Initial price period specified in ISO 8601 format. For example, P1W (one week), P1M (one month), P3M (three months), P6M (six months), and P1Y (one year).	No, only for subscriptions with a PROMO period.	P1Y
introductoryPriceCycles	string	Number of billing periods for the initial price offer.	No, only for subscriptions with a PROMO period.	1

Successful response

```
Unset
{
  "startTimeMillis": "1694431707000",
  "expiryTimeMillis": "1697034507000",
  "autoRenewing": true,
  "priceCurrencyCode": "RUB",
  "priceAmountMicros": "749000000",
  "countryCode": "RU",
  "paymentState": 1,
  "orderId": "33252..1",
  "acknowledgementState": 1,
  "kind": "androidpublisher#subscriptionPurchase",
  "purchaseType": 0,
  "introductoryPriceInfo":
  {
    "introductoryPriceCurrencyCode": "RUB",
    "introductoryPriceAmountMicros": "599000000",
    "introductoryPricePeriod": "P1M",
```

```
}
  "introductoryPriceCycles": "1"
}
```

Error parameters

Attribute	Type	Description	Required	Example
code	string	Response code	Yes	ERROR
message	string	Decoded response code	No	Jwe token is expired
body{}	object	Response body	No	N/A
timestamp	string	Response time	Yes	2024-01-21T13:34:31.067240345+03:00

Error response

```
Unset
{
  "code": "ERROR",
  "message": "Jwe token is expired",
  "body": null,
  "timestamp": "2024-01-21T13:51:59.654427798+03:00"
}
```

How to retrieve subscription status

This method enables to retrieve subscription status from the payment provider via a subscription token

For real payments:

GET

Unset

```
https://public-api.rustore.ru/public/subscription/{subscriptionToken}/state
```

For test payments:

GET

Unset

```
https://public-api.rustore.ru/public/sandbox/subscription/{purchaseToken}/state
```

If you want to work with Test Payments and Subscriptions, you'll need to create a new key, and when you create the key, you'll need to specify the methods to test.

Attribute	Type	Required	Description	Location	Example content
Public-Token	string	Security token to Public API Rustore	Yes	header	
subscription Token	string	The token consists of invoiceId and userId parameters. Token example for invoiceId = 111 and userId = 123: 111.123	Yes	path	111.123

Response parameters

Attribute	Type	Required	Description	Location	Example content
code	number	Response code	Yes	body	

message	string	Response code definition	No	body	
timestamp	timestampz	Response time	Yes	body	
body{}	object	Response body	No	body	

body{}

Attribute	Type	Required	Description	Location	Example content
is_active	boolean	Subscription activity flag. Calculated by subscription status: <ul style="list-style-type: none"> true — for currentPeriod PROMO, START, STANDARD, GRACE && state != CLOSED. false — for currentPeriod HOLD and/or state = CLOSED. 	Yes	body.body{}	true

Example:

```

"code": "200",
"message": "something",
"timestamp": "2023-08-02T10:11:04.655684723+03:00",
"body":
{
  "is_active": "true"
}

```

Response verification

The security token is generated by the security token method and then verified while getting the response:

- Owner and/or app should not be blocked;
- Token should be valid;

- The invoice in application_code response should contain the application code which corresponds to the token owner.

Confirming subscription

The method allows confirming subscription using a subscription token.

Subscriptions do not need to be confirmed to work.

Interaction parameters

For real payments:

GET

Unset

```
https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{purchaseToken}:acknowledge
```

For test payments:

GET

Unset

```
https://public-api.rustore.ru/public/sandbox/subscription/{packageName}/{subscriptionId}/{purchaseToken}:acknowledge
```

If you want to work with Test Payments and Subscriptions, you'll need to create a new key, and when you create the key, you'll need to specify the methods to test.

Attribute	Type	Description	Required	Location	Example
Public-Token	string	Jwe token to Public API Rustore			

Successful response

In case of a successful response, the response body is empty.

Error parameters

Attribute	Type	Description	Required	Example
code	string	Response code	Yes	ERROR
message	string	Decoded response code	No	Jwe token is expired
body{}	object	Response body	No	N/A
timestamp	string	Response time	Yes	2024-01-21T13:34:31.067240345+03:00

Error response

Unset

```
{  
  
  "code": "ERROR",  
  
  "message": "Jwe token is expired",  
  
  "body": null,  
  
  "timestamp": "2024-01-21T13:51:59.654427798+03:00"  
}
```

Response verification

The security token is generated by the [security token method](#) and then verified while getting the response:

- Owner and/or app should not be blocked;
- Token should be valid;
- subscription should be purchased in an application owned by a company that has received a Public API access token.

Uploading and Publishing Apps using the RuStore API

1. To publish an app, follow these steps:
 - First create a draft version with all the necessary app info using the Create Draft Version method.
 - Upload an APK file via the Upload APK file method.
 - Upload the required images, including the app icon and screenshots, using the Upload application icon and Upload screenshots methods.
2. Drafts cannot be edited, though they can be deleted using the Delete Draft method.
3. After filling out all the information, the version can be sent for review via the “Submitting a draft app release for review” method.
4. Get a list of app versions, their basic information and status via the “Getting app version status” method.
5. Change publication settings (publication type, date for delayed publication, and % for partial publication) via the “Change publication settings” method.
6. Once you get your app reviewed and select the publication type - manual, you can publish the version using the “Manual publication” method.

Please note that you must have at least 1 active app release on the RuStore Console to start using the RuStore API.

Creating a draft release

Using this method you can create a draft app release and fill it with basic info.

Limitations:

- You can only create one draft per app. Attempting to create multiple drafts will result in an error;
- drafts submitted via the API are not displayed in the web version of the RuStore Console. They will become available after the APK is submitted for review and when the release status changes;
- creation of a draft version is only possible when there is an active app version available.
- if any of the draft fields are left incomplete or contain missing data, the system will retrieve the corresponding information from the active app version to fill in the gaps;
- when publishing a draft, it is not possible to select the percentage of users who will have access to the application;
- the draft must align with the app type, whether it is free or paid. In the event that you submit a paid version as a draft for a free app, the system will consider the draft as free, adhering to the app type specified.
- for each app version type (appType) there are corresponding version categories. Attempting to download the application will result in an error if the category does not match.
- partial app publication is possible only with automatic (INSTANTLY) or manual (MANUAL) publication type (parameter - publishType).

Interaction options

POST <https://public-api.rustore.ru/public/v1/application/{packageName}/version>

Attribute	Type	Required	Location	Description	Example content
Public-Token	string	Yes	header	RuStore Public API Access Token	

packageName	string	Yes	path	App package name	com.myapp.example
appName	string	No	body	App release name Maximum length: 50 characters	My app
appType	string	No	body	App release types Possible limitations: <ul style="list-style-type: none"> • GAMES — for games; • MAIN — for non-gaming apps. Passed categories must be unique within a single request	GAMES

categories	string	No	body	Version categories Maximum number is 2 categories	"health", "news"
ageLegal	string	No	body	Age ratings Possible options: <ul style="list-style-type: none"> ● «18+»; ● «16+»; ● «12+»; ● «7+»; ● «3+». 	7+
shortDescription	string	No	body	Brief app release description	

				Maximum length: 80 characters	
fullDescription	string	No	body	General app release description Maximum length: 4000 characters	
whatsNew	string	No	body	What's New Maximum length: 500 characters	
moderInfo	string	No	body	Developer comment for moderator Maximum length: 180 characters	

priceValue	string	No	body	<p>App price in minimum currency units (in kopecks), for example, "87.99 rubles." = 8799</p> <p>Value should be >0</p>	8799
publishType (NEW)	string	No	body	<p>Publication type</p> <p>Possible values:</p> <ul style="list-style-type: none"> • MANU AL — manual publication; • INSTA NTLY — automatic publication immediately after review; • DELAY ED — delaye 	MANUAL

				<p>d publicat ion.</p> <p>Note: if this parameter is not specified, then it is taken as INSTANTLY by default.</p>	
publishDateTim e (NEW)	timesta mptz	No Yes, if publishTy pe = DELAYED	body	<p>Date and time for delayed publication format: yyyy-MM-dd'T'HH:mm:ssXX X.</p> <p>The specified date must be no earlier than 24 hours and no later than 60 days from the planned submission date. The delayed publication date can be changed.</p> <p>Note: if publishType is MANUAL or INSTANTLY, this parameter can be</p>	2022-07-08T13:24 :41.8328711+03:0 0

				anything and will not be taken into account.	
partialValue (NEW)	number	No	body	Percentage for partial publication Possible values: <ul style="list-style-type: none"> • 5% • 10% • 25% • 50% • 75% • 100% 	5

Response parameters

Attribute	Type	Required	Location	Description
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	

timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00
content{}	object	Yes		

content{}

Attribute	Type	Required	Description	Example
versionId	number	Yes	App version	243242

Example

```
curl --location --request POST
'https://public-api.rustore.ru/public/v1/application/com.package.com/version' \
--header 'Content-Type: application/json' \
--header 'Public-Token: {YOURtoken}' \
--data-raw '{
  "appName": "App for RuStore",
  "appType": "MAIN",
  "categories":
  [
    "news",
    "education"
  ],
  "ageLegal": "7+",
  "shortDescription": "App for RuStore",
  "fullDescription": "fullDescription - App for RuStore",
```



```
"whatsNew": "whatsNew - App for RuStore",  
"moderInfo": "moderInfo - App for RuStore",  
"priceValue": 1100  
'
```

Response

```
{  
  "code": "OK",  
  "message": null,  
  "body": 243242,  
  "timestamp": "2023-07-27T10:28:59.039649+03:00"  
}
```

Manual publication

Use this method to publish a previously moderated app version.

Limitations:

- You can publish a moderated version only;
- You can publish a version if the publication type is specified as manual.

Interaction parameters

POST

<https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/publish>

Attribute	Type	Required	Description	Example

Public-Token	string	Yes	header	Rustore Public API access token
packageName	string	Yes	path	App package name
versionId	number	Yes	path	App version

Response parameters

Attribute	Type	Required	Description	Example
code	string	Yes	Response code	error/OK
message	string	No	Response code description	
timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Request example

```
curl --location --request POST
'https://public-api.rustore.ru/public/v1/application/com.example.pblsh_v2/version/704372/publish' \

--header 'accept: application/json' \
```

```
--header 'Public-Token: {YOURtoken}'
```

Response example

```
{  
  
  "code": "OK",  
  
  "message": null,  
  
  "body": null,  
  
  "timestamp": "2023-08-14T15:34:44.016339151+03:00"  
  
}
```

Changing publication settings

Use this method to change the publication type, delayed publication date and % for partial publication.

Limitations:

- % for partial publication can only be edited upward;
- if partialValue is 100, the application is considered to be fully rolled out;

- you can change either partialValue or publishType and publishDateTime (for delayed publication).

Interaction Options

POST

https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/publish-settings

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	header	RuStore Public API Access Token	
packageName	string	Yes	path	App package name	
versionId	number	Yes	path	App version	
publishType	string	No	body	<p>Publication type</p> <p>Possible values:</p> <ul style="list-style-type: none"> • MANUAL — manual publication; • INSTANTLY — automatic publication immediately after review; • DELAYED — delayed publication. 	MANUAL

publishDate Time	timest amptz	No Yes, if publis hType = DELA YED	body	Date and time for delayed publication: format: yyyy-MM-dd'T'HH:m m:ssXXX. The specified date must be no earlier than 24 hours and no later than 60 days from the planned submission date. The delayed publication date can be changed. Note: if publishType is MANUAL or INSTANTLY, this parameter can be anything and will not be taken into account.	2022-07-08T1 3:24:41.8328 711+03:00
partialValue	numbe r	No	body	Percentage for partial publication Possible values: <ul style="list-style-type: none"> ● 5% ● 10% ● 25% ● 50% ● 75% ● 100% 	5

Response parameters

Attribute	Type	Required	Location	Description
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Example for changing % for partial publication

```
curl --location
'https://public-api.rustore.ru/public/v1/application/com.example.pblsh_v2/version/704372/
publish-settings' \
--header 'accept: application/json' \
--header 'Content-Type: application/json' \
--header 'Public-Token: {YOURtoken}' \
--data '{
  "partialValue": 100
}'
```

Response example

```
{  
  
  "code": "OK",  
  
  "message": null,  
  
  "body": null,  
  
  "timestamp": "2023-08-14T15:35:12.701709488+03:00"  
  
}
```

Getting app version status

Use this method to obtain basic version info or either check a version status.

Limitations:

- Note that 20 versions are displayed on each page by default, you can display up to 100 versions per page by specifying a value in the size parameter;
- You cannot use pagination and filtering parameters together according to version 1 (or pagination with a pair of parameters page and size or ids).

Interaction Options

GET

https://public-api.rustore.ru/public/v1/application/{packageName}/version?ids=704095&page=0&size=2

Attribute	Type	Required	Location	Description	Example content
Public-Token	string	Yes	header	RuStore Public API Access Token	
packageName	string	Yes	path	App package name	com.myapp.example
ids	number	No	query	id of a specific version Set if necessary to obtain a specific version	743103
page	number	No	query	Page number. Starts from 0	0
size	number	No	query	Number of reviews on a page. Default - 20 Maximum - 100	100

Response parameters

Attribute	Type	Required	Description	Example
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestamp ptz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00
body{}	object	Yes		

body{}

Attribute	Type	Required	Description	Example
content[]	massive	Yes	Array containing a list of versions	
pageNumber	number	Yes	Current page number	0
pageSize	number	Yes	Page size	2
totalElements	number	Yes	Total items	5
totalPages	number	Yes	Total pages	3

content[]

Attribute	Type	Required	Description	Example
versionId	number	Yes	version ID	704372
appName	string	Yes	App name	Test API
appType	string	Yes	App type	MAIN or GAME
versionName	string	Yes	App name	1.0
versionCode	number	Yes	Version code	6
versionStatus	string	Yes	Version status	Possible values ACTIVE, PARTIAL_ACTIVE, READY_FOR_PUBLICATION, PREVIOUS_ACTIVE, ARCHIVED, REJECTED_BY_MODERATOR, TAKEN_FOR_MODERATION, MODERATION, AUTO_CHECK, AUTO_CHECK_FAILED,

				DRAFT, DELETED_DRAFT, REJECTED_BY_SECURITY;
publishType	string	Yes	Publication type	Possible values <ul style="list-style-type: none"> ● MANUAL ● INSTANTLY ● DELAYED
publishDateTime	timestampz	Yes	Date and time for delayed publication	2023-08-04T09:36:06.431+00:00
sendDateForModer	timestampz	Yes	Submission date	2023-08-11T12:03:06.303+00:00
partialValue	number	Yes	Percentage for partial publication	-1 = 100% all other values are consistent
whatsNew	string	Yes	What's new?	Bugs fixed
priceValue	number	Yes	Price if the app is paid	0
paid	boolean	Yes	Is the application paid?	true/false

Request example

```
curl --location
'https://public-api.rustore.ru/public/v1/application/com.example.pblsh_v2/version?page=0
&size=2' \

--header 'accept: application/json' \

--header 'Public-Token: {YOURtoken}'
```

Response example

```
{
  "code": "OK",
  "message": null,
  "body": {
    "content": [
      {
        "versionId": 704372,
        "appName": "API test",
        "appType": "MAIN",
        "versionName": "1.0",
        "versionCode": 6,
        "versionStatus": "ACTIVE",
        "publishType": "MANUAL",
        "publishDateTime": "2023-08-14T12:34:43.925+00:00",
        "sendDateForModer": "2023-08-11T12:03:06.303+00:00",
        "partialValue": -1,
        "whatsNew": "Bugs fixed",
        "priceValue": 0,
        "paid": false
      },
      {
```

```
"versionId": 704197,  
"appName": "PO test API",  
"appType": "MAIN",  
"versionName": "1.0",  
"versionCode": 1,  
"versionStatus": "PREVIOUS_ACTIVE",  
"publishType": "INSTANTLY",  
"publishDateTime": "2023-08-04T09:36:06.431+00:00",  
"sendDateForModer": "2023-08-04T09:20:23.551+00:00",  
"partialValue": -1,  
"whatsNew": "First version",  
"priceValue": 0,  
"paid": false  
}  
],  
"pageNumber": 0,  
"pageSize": 2,  
"totalElements": 2,  
"totalPages": 1  
},  
"timestamp": "2023-08-14T15:38:50.413186769+03:00"
```

```
}
```

Uploading screenshots

Use this method to upload app screenshots.

Limitations:

- screenshots can be either vertical or horizontal;
- screenshot aspect ratio - 16:9 (vertical) and 9:16 (horizontal);
- uploaded file sides - not less than 320px and not over than 3840px;
- upload file format: .jpg or .png;
- file size: up to 3 MB;
- various orientations are not allowed. For instance, if the user has already uploaded a vertical orientation (PORTRAIT) and attempts to load a horizontal one (LANDSCAPE), an error will be returned in response to the request.
- when uploading more than 10 screenshots, existing screenshots will become inactive. For example, if you upload two screenshots with the “ordinal” = 7 parameter, the last uploaded screenshot will be active.

Interaction options

POST

https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/screenshot/{orientation}/{ordinal}

Attribute	Type	Required	Location	Description
Public-Token	string	Yes	header	RuStore Public API Access Token
packageName	number	Yes	path	App ID
versionId	number	Yes	path	App Release
orientation	string	Yes	path	Image orientations Possible options: <ul style="list-style-type: none">• LANDSCAPE — horizontal;• PORTRAIT — vertical.
ordinal	number	Yes	path	Screenshot serial number Possible values: 0 to 9

Form data

Attribute	Type	Required	Description
file	multipart/form-data	Yes	File

Response parameters

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Example

```
curl --location --request POST
'https://public-api.rustore.ru/public/v1/application/com.package.example/version/123/image/screenshot/landscape/1' \
--header 'Content-Type: application/json' \
```

```
--header 'Public-Token: {YOURtoken}' \  
  
--form 'file=@"/Users/User/Downloads/img.jpg"'
```

Uploading an app icon

Use this method to upload an app icon.

Limitations:

- jpeg and png files only;
- file size should not exceed 3 MB;
- uploaded image size is 512×512 px.

Interaction options

POST <https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/image/icon>

Attribute	Type	Required	Location	Description
Public-Token	string	Yes	header	RuStore Public API Access Token
packageName	number	Yes	path	App ID
versionId	number	Yes	path	App Release

Form data

Attribute	Type	Required	Description
file	multipart/form-data	Yes	File

Response parameters

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Example

```
curl --location --request POST
'https://public-api.rustore.ru/public/v1/application/com.package.example/version/123/image/icon' \
--header 'Content-Type: application/json' \
--header 'Public-Token: {YOURtoken}' \
--form 'file=@"/Users/User/Downloads/img.jpg"'
```

Uploading an APK file

Use this method to upload an apk file for publication. It can be further updated.

Limitations:

- .apk files only;
- file size should not exceed 2.5 MB;
- uploaded APK version should be higher than the current active one.

You can upload two APK files at once:

- with different developer signatures to exclude update errors from users. When uploading to the showcase, RuStore will render a file with a similar signature for each user separately.
- which support various services (Huawei Mobile Services and Google Mobile Services) for RuStore to give the user an APK file, which includes services adapted to his device.
- when uploading multiple files, be sure to specify which one is Huawei Mobile Services and which file will be available to all users by default. You can upload no more than 10 files.

Please note that as part of the July 20 update, a required `IsMainApk` parameter was added to the method below. We also ask you to update your pipelines.

Interaction options

POST

<https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/apk>

Attribute	Type	Required	Location	Description
Public-Token	string	Yes	header	RuStore Public API Access Token
packageName	string	Yes	path	App package name
versionId	number	Yes	path	App release
servicesType (NEW)	string	No	query	Type of service used by the app. Possible options: <ul style="list-style-type: none">• HMS — for APK-files with Huawei Mobile Services,• Unknown — is set by default if the field is empty
isMainApk (NEW)	boolean	Yes	query	Attribute that is assigned to the main apk file. Values: <ul style="list-style-type: none">• true — main APK file• false — by default

form data

Attribute	Type	Required	Location	Description
file	multipart	Yes		Binary File

Response parameters

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK

message	string	No	Response message	<p>Error details. Possible options:</p> <ul style="list-style-type: none"> • Maximum number of apk files is already uploaded • Main apk file is already uploaded • Apk file with the Huawei Mobile Services is already uploaded • Apk file with the Huawei Mobile Services can not be main file • Apk file has different version code than the one previously uploaded • The code of this version must be larger than that of the previous one • The package does not match the previous version • A package with this name already exists. Rename the package.
timestamp	timestamp tz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Example

```
curl --location --request POST
'https://public-api.rustore.ru/public/v1/application/com.package.example/version/123/a
pk?servicesType=Unknown&isMainApk=true' \

--header 'Public-Token: {YOURtoken}' \

--form 'file=@"/Users/User/Downloads/package.apk"'
```

List of App Categories

Category	App release type
business	MAIN
state	MAIN
foodAndDrink	MAIN
health	MAIN
books	MAIN
news	MAIN
lifestyle	MAIN
education	MAIN

social	MAIN
adsAndServices	MAIN
pets	MAIN
purchases	MAIN
tools	MAIN
travelling	MAIN
entertainment	MAIN
parenting	MAIN
sport	MAIN
gambling	MAIN
transport	MAIN
finance	MAIN
arcade	GAMES
quiz	GAMES
puzzle	GAMES

race	GAMES
children	GAMES
ar	GAMES
indie	GAMES
casino	GAMES
casual	GAMES
card	GAMES
music	GAMES
board	GAMES
adventure	GAMES
rolePlaying	GAMES
family	GAMES
simulator	GAMES
word	GAMES
sports	GAMES

strategy	GAMES
action	GAMES

Deleting a draft release

Use this method to delete unnecessary drafts.

Limitations:

- you can only delete versions that have not yet been published.

Interaction options

DELETE

<https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}>

Attribute	Type	Required	Location	Description
Public-Token	string	Yes	header	RuStore Public API Access Token
packageName	string	Yes	path	App package name
versionId	number	Yes	path	App Release

Response parameters

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Example

```
curl --location --request DELETE  
  
'https://public-api.rustore.ru/public/v1/application/com.package.example/version/123' \  
  
--header 'Content-Type: application/json' \  
  
--header 'Public-Token: {YOURtoken}'
```

Submitting a draft app release for review

Use this method to submit a draft app release for review.

Interaction options

POST

`https://public-api.rustore.ru/public/v1/application/{packageName}/version/{versionId}/commit?priorityUpdate={priorityUpdate}`

Attribute	Type	Required	Location	Description
Public-Token	string	Yes	header	RuStore Public API Access Token
packageName	string	Yes	path	App package name
versionId	number	Yes	path	App release
priorityUpdate	number	No	query	Update Priority Possible values: 0 to 5, where 0 is the minimum and 5 is the maximum value Default is 0

Response parameters

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	<p>Error detail. Possible values:</p> <ul style="list-style-type: none"> • Version must have at least one main non-HMS apk-file • Version must have not only HMS apk-file. • Packages for version with id = Is not found.
timestamp	timestampz	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Using the RuStore API for Review Management

General Review Management Workflow

1. Developers have several options for managing their app reviews using the RuStore API:
 - receive feedback on their application by utilizing the "Get App Feedback" method.
 - receive feedback in CSV through the "Receiving Feedback in .csv Format" method.
2. Responding to Reviews: Developers can actively engage with user reviews by employing the "Respond to Review" method.
3. Moderation Status: To check the moderation status of a response to a review, developers can use the "Get the Status of a Response to a Review" method
4. Editing Responses: Developers can also modify a previously published response to a review by utilizing the "Change Response to Review" method.
5. Deleting Responses: Developers can delete a response to a review using the "Delete a Response to a Review" method whenever so required.
6. Getting App Rating: To retrieve the overall app rating, developers can employ the "Get App Rating" method.

For all of these methods, it is essential to obtain an access token for the RuStore API, which has a lifespan of 15 minutes.

Possible error codes

Code	Value
200	OK
400	Invalid request
401	Not authorized

404	Not found
500	Internal Server Error

Example response in case of error

```
{  
  
  "code": "ERROR",  
  
  "message": "404",  
  
  "body": null,  
  
  "timestamp": "2023-05-30T20:08:14.120231216+03:00"  
  
}
```

Getting app feedback

This method enables you to access a list of the most recent reviews for your application, as well as retrieve individual reviews.

Key Points to Note:

- By default, each page displays 20 reviews. However, you can customize this to display up to 100 reviews per page by specifying a value in the "size" parameter.
- It's important to be aware that you cannot apply pagination and filtering parameters to retrieve just one review. Pagination should be used with a combination of parameters such as "page," "size," or "id."
- If a user edits a review, the review is assigned an updated review ID, and the "edited" attribute is set to "true."
- The default sorting order is based on the date the review was published, using the "updatedAt" parameter, with the most recent reviews appearing first.

Interaction Options

GET

<https://public-api.rustore.ru/public/v1/application/{packageName}/comment?id={id}&page={number}&size={size}>

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	header	RuStore API access token	
packageName	string	Yes	path	Application package name. Maximum length – 50 characters	com.myapp.example
id	number	No	query	ID of a specific review. Asked if you need to get specific feedback	743103
page	number	No	query	Page number. Starts from 0	0
size	number	No	query	Number of reviews on the page. Default - 20. Maximum - 100	100

Response options

Attribute	Type	Required	Description	Example
code	string	Yes	Response code	error/OK
message	string	No	Response code description	
timestamp	timestamp	Yes	Response time	2022-07-08T13:24:41.8328711+03:00
body{}	object	No		

body{}

Attribute	Type	Required	Location	Example
packageName	string	Yes	Application package name	com.myapp.example

appld	number	Yes	Application ID	385727
commentId	number	Yes	Review ID	697535
userName	string	Yes	Review author's name	Irina
appRating	number	Yes	Review rating	4
commentStatus	string	Yes	Review status	PUBLISHED
commentDate	timestampz	Yes	Review date	2023-05-22 16:32:08.008
commentText	string	Yes	Review text	Awesome!
likeCounter	number	Yes	Number of likes to a review from other users	5
dislikeCounter	number	Yes	Number of dislikes for a review from other users	0
updatedAt	timestampz	Yes	Date of review moderation and publication	2022-10-14 15:14:33.033
appVersionName	string	Yes	App version	1.4
edited	boolean	Yes	Sign of review edits	True - edited (review was edited and rewritten). False - no

Example for multiple reviews

```
curl --location 'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment' \
--header 'Content-Type: application/json' \
--header 'Public-Token:
eyJlbmMiOiJBMjU2R0NNliwiYWxnIjoiUINBLU9BRVAtMjU2In0.jrVI3YT99saGcata9fzN6_
QpoQhDsv8oBUAaj9p7UyR4Ga5PM8TTyNbpTKlealjGoEfsBMJx0aw0b7fhD04bhSp7SW
EBKGBTzFCjwOZ5_Fcezq0-NOMSayzoPttYa7oMRDiqNS4rqaUOdCUrf9qIDyEq3BGoU
YCaUD7L5399I51NhSLrWpoPx1I4ZLVJ5bjlhiCoPAtLArnulq5LBoDk3naoGaRHabkkffcuc
EjA45uNpCsq0fx77Lk4YhN30LOccylmE-O8fUq8YryMWv4w-ZmWjax1oT9nRgO95r9EY
G7Gwdekq2lLuWnsofiMXME8t1EiEuUmDpNTEyS9SiUMRwQ.ISJBV1mSmHehuqVs.8p
XL_GRLwEJgIWZOzkkIKgrsGKKJrG9kv1AldD0PPU8KtsY8GVAc5xaaQgeyjSsJiSUvma
_IohAalBwP-tjTRxrnzVinMhKAJMCbiMIVqsQSRDB5j_mf91nTNewQkWJwB33Rvxd9F4T
t-Tk-1QKALU8tAT_HXAl.v9WBgx8T6yFDpeMrjw-ECCQ'
```

Response example

```

{
  "code": "OK",
  "message": null,
  "body": [
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 2142370751,
      "userName": "Saber",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-06-13 11:58:06.006",
      "commentText": "Good App, nice done!",
      "likeCounter": 0,
      "dislikeCounter": 0,
      "updatedAt": "2023-06-13 11:59:50.050",
      "appVersionName": "1.4",
      "edited": false
    },
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 1981700287,
      "userName": "Victor",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-04-27 18:54:27.027",
      "commentText": "Nice app! ;#",
      "likeCounter": 0,
      "dislikeCounter": 0,
      "updatedAt": "2023-04-27 18:55:30.030",
      "appVersionName": "1.3",
      "edited": false
    },
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 1981699775,
      "userName": "Gregory",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-04-27 18:54:18.018",
      "commentText": "Nice app!",
      "likeCounter": 1,
      "dislikeCounter": 0,
      "updatedAt": "2023-04-27 18:55:30.030",
      "appVersionName": "1.3",
      "edited": false
    },
    {
      "packageName": "ru.voonsh.push",

```

```
"appId": 227169215,  
"commentId": 237681343,  
"userName": "Ibrahim",  
"appRating": 5,  
"commentStatus": "PUBLISHED",  
"commentDate": "2022-09-15 17:26:46.046",  
"commentText": "Increasing the DAU)))",  
"likeCounter": 1,  
"dislikeCounter": 1,  
"updatedAt": "2022-09-16 18:06:39.039",  
"appVersionName": null,  
"edited": false  
  }  
],  
"timestamp": "2023-06-15T07:32:55.505979576+03:00"  
}
```

Example for a specific review

```

{
  "code": "OK",
  "message": null,
  "body": [
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 2142370751,
      "userName": "Saber",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-06-13 11:58:06.006",
      "commentText": "Good App, nice done!",
      "likeCounter": 0,
      "dislikeCounter": 0,
      "updatedAt": "2023-06-13 11:59:50.050",
      "appVersionName": "1.4",
      "edited": false
    },
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 1981700287,
      "userName": "Victor",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-04-27 18:54:27.027",
      "commentText": "Nice app! ;#",
      "likeCounter": 0,
      "dislikeCounter": 0,
      "updatedAt": "2023-04-27 18:55:30.030",
      "appVersionName": "1.3",
      "edited": false
    },
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 1981699775,
      "userName": "Gregory",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-04-27 18:54:18.018",
      "commentText": "Nice app!",
      "likeCounter": 1,
      "dislikeCounter": 0,
      "updatedAt": "2023-04-27 18:55:30.030",
      "appVersionName": "1.3",
      "edited": false
    },
    {
      "packageName": "ru.voonsh.push",

```

```

    "appId": 227169215,
    "commentId": 237681343,
    "userName": "Ibrahim",
    "appRating": 5,
    "commentStatus": "PUBLISHED",
    "commentDate": "2022-09-15 17:26:46.046",
    "commentText": "Increasing the DAU)))",
    "likeCounter": 1,
    "dislikeCounter": 1,
    "updatedAt": "2022-09-16 18:06:39.039",
    "appVersionName": null,
    "edited": false
  }
],
"timestamp": "2023-06-15T07:32:55.505979576+03:00"
}

```

Example for a specific review

```

curl --location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment?id=1981699775' \
--header 'Content-Type: application/json' \
--header 'Public-Token:
eyJlbnMiOiJBMjU2R0NNliwiYWxnIjoiaUINBLU9BRVAtMjU2In0.h635qF_TZc43287jXQVI
mu_-o4eVFQwrQYe2WnjbywC_KUC4oX6W3ssyPWzIAugd2RELbVCTk1wiDdKwPIbkOJ
C_HdF0yAmnPg0PRxwyfCHblRuccuuEg_l6sKY1Fqrh6kH3D5N2i_HnDei-hTusAvHR333
ZstAK73dc-4Ecn24jb1XyBsdg0_KddKaRpEjTMSudIV6rdpBNMIQRUyQufrP2RMXK5Kc_
0gY0iA-tazQoOJmK4xstHmuFbSBx3J6oN5QIIYonx0LZ6ABf2fD0O1E7LFsVUMd2bOdLY
g5id5bTRXKd238iB5snmPhGJJN3d6v8xdoV5TdOMGPvjO0A5A.-mSbEMAvEglyEOJu.e
KxraHkohwEcn3cG4glUBnwyypjnapol4WnwhQGyKe-TDq9TGNj9CO4hnXGh4UPSR155w
73pJwSCuDe7LfsQ8zqBVjirT_HXypowHsWBwvBG-6rwSRvhZsad2YY8wHTZeXOddVyn
WnESoKAnXldlmafEbIWN1Hik.VJ-b1KxDxMrgJTz_Vuul2Q'

```


Example response

```
{
  "code": "OK",
  "message": null,
  "body": [
    {
      "packageName": "ru.voonsh.push",
      "appId": 227169215,
      "commentId": 1981699775,
      "userName": "Gregory",
      "appRating": 5,
      "commentStatus": "PUBLISHED",
      "commentDate": "2023-04-27 18:54:18.018",
      "commentText": "Nice app!",
      "likeCounter": 1,
      "dislikeCounter": 0,
      "updatedAt": "2023-04-27 18:55:30.030",
      "appVersionName": "1.3",
      "edited": false
    }
  ],
  "timestamp":
  "2023-06-15T07:13:16.309841987+03:00"
}
```

Receiving feedback in .csv

This method enables you to obtain all reviews in CSV format for a specified time frame.

Key Limitations:

- The minimum duration for selecting reviews is 1 day.
- The maximum duration for selecting reviews is limited to 92 days.

Interaction Choices

GET

https://public-api.rustore.ru/public/v1/application/{packageName}/comment/export?from={date_from}&to={date_to}

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	header	RuStore API access token	
packageName	string	Yes	path	Application package name. Maximum length: 50 characters	com.myapplication.example
from	date	Yes	query	Starting date for uploading reviews, specified in YYYY-MM-DD format.	2023-06-01
to	date	Yes	query	Ending date for uploading reviews, inclusive, specified in YYYY-MM-DD format.	2023-06-05

Response options

.csv file

user comment for application com.publicapi.publicapi from 2023-05-22 to 2023-05-23 (1)

Package Name	User App Version	Review Submit Date and Time	Review Last Update Date and Time	Star Rating	Review Text	User name	Comment id	Like counter	Dislike counter	edited
com.publicapi.publicapi		2023-05-22 16:32:08.446	2023-05-22 17:02:22.019	4	Клещи, но можно лучше	Irina	697535	0	0	false
com.publicapi.publicapi		2023-05-22 16:28:25.699	2023-05-22 17:02:59.293	2	Ну не знаю, мне не нравится	Afkaa	697279	0	0	false

File content

Column title	Description
Package Name	Application package name
User App Version	Application version
Review Submit Date and Time	Review date
Review Last Update Date and Time	Date of moderation and publication
Star Rating	Review rating
Review Text	Review text
User name	Review author's name

Comment Id	Review ID
Like counter	Number of likes to a review from other users
Dislike counter	Number of dislikes to a review from other users
edited	Sign of review edits

Example

```
curl --location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment/export?from=2
023-03-03&to=2023-04-28' \
--header 'Public-Token:
eyJlbnMiOiJBMjU2R0NNliwiYWxnljoiUINBLU9BRVAtMjU2In0.h635qF_TZc43287jXQVI
mu_-o4eVFQwrQYe2WnjbywC_KUC4oX6W3ssyPWzIAugd2RELBVCTk1wiDdKwPIbkOJ
C_HdF0yAmnPg0PRxwyfCHbIRuccuuEg_l6sKY1Fqrh6kH3D5N2i_HnDei-hTusAvHR333
ZstAK73dc-4Ecn24jb1XyBsdg0_KddKaRpEjTMSudIV6rdpBNMIQRUyQufrP2RMXK5Kc_
0gY0iA-tazQoOJmK4xstHmuFbSBx3J6oN5QIIYonx0LZ6ABf2fD0O1E7LFsVUMd2bOdLY
g5id5bTRXKd238iB5snmPhGJJN3d6v8xdoV5TdOMGPvjO0A5A.-mSbEMAvEglyEOJu.e
KxraHkohwEcn3cG4glUBnwyjnapol4WnwhQGyKe-TDq9TGNj9CO4hnXGh4UPSRI55w
73pJwSCuDe7LfsQ8zqBVjirT_HXypowHsWBwvBG-6rwSRvhZsad2YY8wHTZeXOddVyn
WnESoKAnXldlmafEbIWN1Hik.VJ-b1KxDxMrgJTz_Vuul2Q'
```

Leaving a reply to review

This method allows you to leave a response to a review.

Interaction Options

POST

`https://public-api.rustore.ru/public/v1/application/{packageName}/feedback?commentId={commentId}`

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	header	RuStore Public API Access Token	
packageName	string	Yes	path	App package name. Maximum length – 50 characters	com.myapp.example
commentId	number	Yes	query	User Review ID	748223
message	string	Yes	body	Text of response to review	Thanks for your feedback. It means a lot to us

Interaction options

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestamp	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

body{}	object	Yes		
--------	--------	-----	--	--

body{}

Attribute	Type	Required	Description	Example content
id	number	Yes	User Review ID	748479

Example

```
curl --location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback?commentId=2
142370751' \
--header 'Content-Type: application/json' \
--header 'Public-Token:
eyJlbnMiOiJBMjU2R0NNliwiYWxnljoiUINBLU9BRVAtMjU2In0.ziOR4J-_3A36M55IMdJqx
ck4Lk9GFe6vvdRFBkIICA6z4WLLUxdEyHNdMWYomBs9MkJecCaCStOQ5YtmHvFIR
K8aoj4c386WwUBXmTDXN_BJg2puwuLivMJWuAhgveZpC7afZCwM6m5RgLi538BAjFV
_gE8XvbSUKIhWhkEvlzgrK2zk211SRUVXaAdrWEz5NNSsrQhyEv1fiMgQNmV9Sehp8fx
P7G_9HkAWVtfNvgEiTbFHMT0-qpwtdh5Ts440Du9MC7PL59IUmXts1Khx6xbuUWLQe
3WHPQCBmKezpxtl-l9Ms4F-iopZy-bXzUaUMFsaQu8Jh4kFvztFenYg.UPvgd5jKQcFO-M
RE.zHHu-P5GSwUTvEMae-bu337jxppq25TbftTC7oF8r0APCfUqdx55CbO-PuleGdjN08K4
IC8GsWhLmKr9mqEeV9L-Dh5QsqA6M7GsLH8LAqrR1UX0Z849pyhrOt_Pz4SJ3YeHdu-
ITfTL5Ysr0kECMXWIMcE24X8U.zHXeYuxDJf-Wtl9Vn0betw' \
--data '{
  "message": "Thank you"
}'
```

Example response

```
{
  "code": "OK",
  "message": null,
  "body": {
    "id": 2149775551
  },
  "timestamp": "2023-06-02T16:36:57.847391009+03:00"
}
```

Getting review response status

Use this method to obtain the response-to-review moderation status or either information on a separate response.

Restrictions:

- By default, each page displays 20 reviews. You can display up to 100 reviews per page by specifying a value in the size parameter;
- You cannot use pagination and filtering parameters together based on 1 response to a review (or pagination by specifying a pair of parameters page and size or id).

Interaction Options

GET

`https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/?id={id}&page={number}&size={size}`

Attribute	Type	Required	Location	Description	Example content
Public-Token	string	Yes	header	RuStore Public API Access Token	
packageName	string	Yes	path	App package name. Maximum length: 50 characters	com.myapp.example
feedbackId	number	No	path	id of a specific response-to-review, it is set to get information about a specific response to a review	743103
id	number	No	query	id of a specific response-to-review, set if necessary to receive a specific review	743103

page	number	No	query	Page number. Starts from 0	0
size	number	No	query	Number of reviews on a page. Default - 20, maximum - 100	100

Response parameters

Attribute	Type	Required	Description	Example content
code	string	Yes	Response code	error/OK
message	string	No	Response decoded message	
timestamp	timestamp	Yes	Response time	2022-07-08T13:24:41.832871 +03:00
body{}	object	Yes		

body{}

Attribute	Type	Required	Description	Example content
id	number	Yes	Review response ID	748479
commentId	number	Yes	Review ID	748223
text	string	Yes	Response text	This is a response to a user review

status	string	Yes	Review response status	PUBLISHED — everything is published successfully; MODERATION — under moderation; REJECTED — rejected by moderation; DELETED — deleted by developer or changed.
date	string	Yes	Last modified date and time	2023-06-01 18:10:43.043

Example 1

```
curl --location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback?id=2177559743' \
--header 'Public-Token: eyJlb.....VcQVmNyw'
```

Response 1

```
{
  "code": "OK",
  "message": null,
  "body": [
    {
      "id": "2177559743",
      "commentId": "2142370751",
      "text": "Thank you",
      "status": "PUBLISHED",
      "date": "2023-06-22T09:46:18.115+00:00"
    }
  ],
  "timestamp": "2023-06-22T12:46:30.193419294+03:00"
}
```


Confirm subscription via a subscription token

This method allows confirming subscription via a subscription token.

At that, confirmation of subscription is not required for subscriptions to function.

Interaction Options

POST

`https://public-api.rustore.ru/public/glike/subscription/{packageName}/{subscriptionId}/{purchaseToken};acknowledge`

Attribute	Type	Description	Required	Location	Example content
Public-Token	string	Rustore Public API access token	Yes	header	
packageName	string	App package name	Yes	path	
subscriptionId	string	Subscription product code	Yes	path	
purchaseToken	string	Retrieving a subscription token	Yes	path	

Successful Response Options

If your response is successful, the response body remains empty.

Request Validation

When a request is received, the authorization token obtained by GET method (Getting authorization token) is verified:

- owner and/or application must not be blocked;
- token must be valid;
- subscription was purchased in an app owned by the company that received the Public API access token.

Editing review response

Use this method to change your review response text.

Interaction Options

POST

<https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}>

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	Header	RuStore API access token	
packageName	string	Yes	path	Application package name. Maximum length – 50 characters	com.myapp.example
feedbackId	number	No	path	Review response ID that needs to be changed	743103
message	string	Yes	body	Edited response	We found a misprint in our response so we edited it

Response options

Attribute	Type	Required	Description	Example
code	string	Yes	Response code	error/OK
message	string	No	Response code description	
timestamp	timestamp	Yes	Response time	2022-07-08T13:24:41.8328711+03:00
body{}	string	Yes		

body{}

Attribute	Type	Required	Description	Example
id	number	Yes	New review response ID. The previous ID takes on the DELETED status. If there is a change, it will further be displayed with the updated ID	748479

Example

```
curl --location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback/2149775551' \
--header 'Content-Type: application/json' \
--header 'Public-Token:
eyJlbnMiOiJBMjU2R0NNIiwiaWwiYWxnIjoiUINBLU9BRVAtMjU2In0.ziOR4J-_3A36M55IMdJqx
ck4Ltk9GFfe6vvdRFBkIIICA6z4WLLUxdEyHNdMWYomBs9MkJecCaCStOQ5YtmHvFIR
K8aoj4c386WwUBXmTDXN_BJg2puwuLivMJWuAhgveZpC7afZCwM6m5RgLi538BAjFV
_gE8XvbSUKIhWhkEvlzgrK2zk211SRUVXaAdrWEz5NNSsrQhyEv1fiMgQNmV9Sehp8fx
P7G_9HkAWVtfNvgEiTbFHMT0-qpWtdh5Ts440Du9MC7PL59IUmXts1Khx6xbuUWLQe
3WHPQCBmKezpxtl-l9Ms4F-iopZy-bXzUaUMFsaQu8Jh4kFvztFenYg.UPvgd5jKQcFO-M
RE.zHHu-P5GSwUTvEMae-bu337jxpxq25TbftTC7oF8r0APCfUqdx55CbO-PuleGdjN08K4
IC8GsWhLmKr9mqEeV9L-Dh5QsqA6M7GsLH8LAqrR1UX0Z849pyhrOt_Pz4SJ3YeHdu-
ITfTL5Ysr0kECMXWIMcE24X8U.zHXeYuxDJf-Wtl9Vn0betw' \
--data '{
  "message": "Thank you very much!"
}'
```

Response example

```
{
  "code": "OK",
  "message": null,
  "body": {
    "id": 2149807039
  },
  "timestamp": "2023-06-15T08:26:25.355326578+03:00"
}
```

Deleting review response

Use this method to delete a review response.

Interaction Options

DELETE

<https://public-api.rustore.ru/public/v1/application/{packageName}/feedback/{feedbackId}>

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	header	RuStore API access token	
packageName	string	Yes	path	Application package name. Maximum length – 50 characters	com.myapp.example
feedbackId	number	No	path	Response ID that needs to be deleted	743103

Response parameters

Attribute	Type	Required	Description	Example
code	string	Yes	Response code	error/OK
message	string	No	Response code description	
timestamp	timestamp	Yes	Response time	2022-07-08T13:24:41.8328711+03:00

Example

```
curl --location --request DELETE
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/feedback/2149807039' \
--header 'Content-Type: application/json' \
--header 'Public-Token:
eyJlbmMiOiJBMjU2R0NNliwiYWxnIjoiUINBLU9BRVAtMjU2In0.1r3cxOdxuNpypJSWXMq
4oAVYhqh6_3RlqKfltkhthTzisiRYnboOkZpw_r5J9w0S5G8u-BexQeganyoG3MbCJ5QP2
X6945wQMxIPki81UKewkZuFrjsH36USk6dnnMbjT8Yw8nA4Yr0n8Oinspj3zkw66kZd-57E
JvoMfneCEyTBY1mYEoc2DnfUa99syX1klgX7Jfipn4yRm3pxWad5aesCK3eQFIP57CBir
m8qGecDzkKcV1DeBx-qXK1S72FaXld11zN-rbe14U1z8jCCiEHhYrTIT9ci7OKF7OqF2kg
RRVdCoO3eRWI4JWF-JfGAeYcv7rEpNLC32pEm5FOCA.khXZSeTodz2mVoNd.fKVjmk
HUEM9AW7S_mYf-LFO4T26Lqf4RNSyjnMfFsxZybDKahZgnaJ4IXYq-MPVN-o39eg1jIS
moJcBonqS-0rIfE1P3CAM5cbNiSsTCX1r-cVdf4ei998KKGMg8bZL24-uLfxgcJSBBTgmU
kyvf_KqH_dcxmQ.DwbK_08RLgHibat3h5dvkQ'
```

Response example

```
{
  "code": "OK",
  "message": null,
  "body": null,
  "timestamp": "2023-05-30T20:07:22.365025849+03:00"
}
```

Getting app rating

Use this method to get an app rating.

Interaction Options

POST <https://public-api.rustore.ru/public/v1/application/{packageName}/comment/statistic>

Attribute	Type	Required	Location	Description	Example
Public-Token	string	Yes	header	RuStore API access token	
packageName	string	Yes	path	Application package name. Maximum length – 50 characters	com.myapp.example

Response parameters

Attribute	Type	Required	Description	Example
code	string	Yes	Response code	error/OK
message	string	No	Response code description	
timestamp	timestamp ptz	Yes	Response time	2022-07-08T13:24:41.8328 711+03:00
body{}	object	Yes		

body{}

Attribute	Type	Required	Description	Example
ratings	object	Yes		
averageUserRating	number	Yes	Average user app rating	4
totalRatings	number	Yes	Total number of ratings	0
totalResponses	number	Yes	Total number of unanswered reviews	0
ratingsNoComments	number	Yes	Total number of ratings without reviews	0

ratings{}

Attribute	Type	Required	Description	Example
amountFive	number	Yes	Total 5-star rates	0
amountFour	number	Yes	Total 4-star rates	3
amountThree	number	Yes	Total 3-star rates	0
amountTwo	number	Yes	Total 2-star rates	0
amountOne	number	Yes	Total 1-star rates	0

Example

```
curl --location
'https://public-api.rustore.ru/public/v1/application/ru.voonsh.push/comment/statistic' \
--header 'Public-Token:
eyJlbnMiOiJBMjU2R0NNliwiYWxnljoiUINBLU9BRVAtMjU2In0.1r3cxOdxuNpypJSWXMQ
4oAVYhqh6_3RlqKfltkhthTzisiRYnboOkZpw_r5J9w0S5G8u-BexQeganyoG3MbCJ5QP2
X6945wQMxIPki81UKewkZuFrjsH36USk6dnnMbjT8Yw8nA4Yr0n8Oinspj3zkw66kZd-57E
JvoMfneCEyTBY1mYEoc2DnfUa99syX1klgX7Jfipn4yRm3pxWad5aesCK3eQFIP57CBir
m8qGecDzkKcV1DeBx-qXK1S72FaXld11zN-rbe14U1z8jCCiEHhYrTIT9ci7OKF7OqF2kg
RRVdCoO3eRWI4JWF-JfGAeYcv7rEpNLC32pEm5FOCA.khXZSeTodz2mVoNd.fKVjmk
HUEM9AW7S_mYf-LFO4T26Lqf4RNSyjnNmFfsxZybDKahZgnaJ4IXYq-MPVN-o39eg1jIS
moJcBonqS-0rIFe1P3CAM5cbNiSsTCX1r-cVdf4ei998KKGMg8bZL24-uLfxgcJSBBTgmU
kyvf_KqH_dcxmQ.DwbK_08RLgHibat3h5dvkQ'
```

Response example

```
{
  "code": "OK",
  "message": null,
  "body": {
    "ratings": {
      "amountFive": 13,
      "amountFour": 2,
      "amountThree": 0,
      "amountTwo": 0,
      "amountOne": 0
    },
    "averageUserRating": 4.87,
    "totalRatings": 15,
    "totalResponses": 9,
    "ratingsNoComments": 4
  },
  "timestamp": "2023-06-15T08:35:06.412194896+03:00"
}
```


Using RuStore API for Access Control

App access information

This method returns a list of all accesses granted by application.

Interaction parameters

GET <https://public-api.rustore.ru/public/v1/application/{packageName}/developer>

Attribute	Type	Required	Location	Description	Example
Public-TokenstringYesHeader	HYPERLINK "/work-with-RuStore-API/api-authorization-token"			Access token to API RuStore.	

	N/A
--	-----

packageNamestringYespathApp package namecom.myapp.example

pageSize	number	No	query	Number of users and their accesses per page.	Default 20, min - 1, max - 100
----------	--------	----	-------	----------------------------------------------	--------------------------------

pageTokenstring	No	query	The API includes a pageToken element in the response if the access list continues to another page. Use the value obtained from the previous request.	Nzk0MjQ3Mzcw
-----------------	----	-------	------------------------------------------------------------------------------------------------------------------------------------------------------	--------------

Response example

Attribute	Type	Required	Description	Example
-----------	------	----------	-------------	---------

codestringYesResponse codeerror/OK

message	string	No	Decoded response code	N/A
---------	--------	----	-----------------------	-----

body{}	object	Yes	N/A	N/A
--------	--------	-----	-----	-----

timestamptimestampzYesResponse time2022-07-08T13:24:41.8328711+03:00

body

Attribute	Type	Required	Description	Example
-----------	------	----------	-------------	---------

content[]	massive	Yes	Array with a list of users and passwords.	
-----------	---------	-----	-------------------------------------------	--

pageToken

	No	No	The API includes a pageToken element in the response if the access list continues to another page.	Nzk0MjQ3Mzcw
--	----	----	----------------------------------------------------------------------------------------------------	--------------

body.content[]

Attribute	Type	Required	Description	Example
devVklId				
	string	Yes	vkid	161930531

role

	string	Yes	access type	• OWNER; • NON_RESIDENT_OWNER; • INDIVIDUAL_OWNER; • ADMIN; • RELEASE_MANAGER; • DEV; • FINANCIAL_MANAGER; • SUPPORT.
--	--------	-----	-------------	-----------------------------------------------------------------------------------------------------------------------

firstName

	string	Yes	User name	John
--	--------	-----	-----------	------

lastName

	string	Yes	Last name	Doe
--	--------	-----	-----------	-----

Possible errors

codemessage

	Description	Solution	
400	Incorrect parameter role	Such a role does not exist	Make sure that the specified role is available.

Request example

```
curl --location 'https://public-api.rustore.ru/public/v1/application/com.package.example/developer?pageSize=20' \
--header 'Public-Token: {YOURtoken}'
```

Response example

```
{
  "code": "OK",
  "message": null,
  "body": {
    "content": [
      {
        "devVklId": "1111111111",
        "role": "DEV",
        "firstName": "John",
        "lastName": "Smith"
      },
      {
        "devVklId": "0000000000",
        "role": "OWNER",
        "firstName": "Mike",
        "lastName": "Brown"
      }
    ],
    "pageToken": null
  },
  "timestamp": "2024-03-25T20:48:23.584572102+03:00"
}
```

Revoke access

This method allows you to revoke the user's access.

Interaction parameters

DELETE <https://public-api.rustore.ru/public/v1/application/{packageName}/developer/{devVklId}/role/{roleName}>

Attribute	Type	Required	Location	Description	Example
Public-TokenstringYesHeader				HYPERLINK "/work-with-RuStore-API/api-authorization-token"	Access token to API RuStore.

	N/A
--	-----

packageNamestringYespathApp package namecom.myapp.example

devVklIdstringYespathVK ID of the user whose access is to be revoked.743103

Attribute	Type	Required	Location	Description	Example
roleNamestring	Yes	path	User's role to be revoked.	• OWNER;• NON_RESIDENT_OWNER;• INDIVIDUAL_OWNER; • ADMIN;• RELEASE_MANAGER;• DEV; • FINANCIAL_MANAGER; • SUPPORT.	

Read more about each role [HYPERLINK "/developers/developer-account/user-roles/"](#) here.

Response example

Attribute	Type	Required	Description	Example
-----------	------	----------	-------------	---------

codestringYesResponse codeerror/OK

Attribute	Type	Required	Description	Example
message	string	No	Decoded response code	N/A

timestamptimestampzYesResponse time2022-07-08T13:24:41.8328711+03:00

Attribute	Type	Required	Description	Example
HYPERLINK \ "body" body{}	string	Yes	N/A	N/A

Possible errors

Code	Description	Solution	
400	Incorrect parameter role	Such a role does not exist	Make sure that the specified role is available.
400	Owner role can not be revoked	You cannot remove an owner	

404	No application developer with this role found	We have not found a developer with this role	Make sure that the parameters devVklId and roleName are set correctly.
-----	-----------------------------------------------	----------------------------------------------	------------------------------------------------------------------------

Request example

```
curl --location --request DELETE
'https://public-api.rustore.ru/public/v1/application/com.package.example/developer/1111111111/role/DEV' \
--header 'Public-Token: {YOURtoken}'
```

Response example

```
{
  "code": "OK",
  "message": null,
  "body": null,
  "timestamp": "2024-03-25T20:50:36.360850021+03:00"
}
```